



Zbigniew
Huzar

LOTOS

język
formalnych
specyfikacji
systemów
informatycznych

Medium transmisyjne



Zbigniew Huzar

LOTOS
– język formalnych specyfikacji
systemów informatycznych



Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2007

Wydział Informatyki i Zarządzania, Instytut Informatyki Stosowanej

Recenzenci
Jan Magott
Tomasz Szmuc

Opracowanie redakcyjne i korekta
Alicja Kordas

Projekt okładki
Zofia i Dariusz Godlewscy

Wszelkie prawa zastrzeżone. Opracowanie w całości ani we fragmentach nie może być powielane ani rozpowszechniane za pomocą urządzeń elektronicznych, mechanicznych, kopiujących, nagrywających i innych bez pisemnej zgody posiadacza praw autorskich.

© Copyright by Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2007

ISBN 978-83-7493-335-3

Oficyna Wydawnicza Politechniki Wrocławskiej
Wybrzeże Wyspiańskiego 27, 53-370 Wrocław
<http://www.oficyna.pwr.wroc.pl>
oficwyd@pwr.wroc.pl

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 623/2007

Spis treści

Przedmowa	5
1. Wstęp	8
1.1. Wytwarzanie oprogramowania.....	8
1.2. Specyfikacja wymagań.....	11
1.3. Modele systemów.....	13
1.4. Specyfikacja funkcjonalna systemów oprogramowania.....	16
1.5. Metody formalne w procesie wytwarzania systemów.....	20
1.6. LOTOS w specyfikacji standardów sieci komputerowych	21
1.7. Zakres monografii	23
2. Język CCS	25
2.1. Wstęp	25
2.2. Składnia i nieformalna semantyka.....	28
2.3. Semantyka operacyjna.....	31
2.4. Relacje równoważności.....	38
2.5. Prawa równościowe.....	40
2.6. Twierdzenie o ekspansji	42
2.7. Uwagi końcowe.....	43
3. Rozszerzenia czasowe CCS	45
3.1. Wstęp	45
3.2. TCCS – wprowadzenie.....	46
3.3. TCCS – definicja formalna.....	48
3.4. RTCCS – wprowadzenie	52
3.5. RTCCS – definicja formalna.....	53
3.6. RTCCS – przykłady	60
3.7. Uwagi końcowe.....	64
4. Abstrakcyjne typy danych w LOTOSie	65
4.1. Podstawowe pojęcia algebraiczne	65
4.2. Algebra termów	67
4.3. Algebra ilorazowa termów	68
4.4. Specyfikacja równościowa typów abstrakcyjnych	70
4.5. Semantyka specyfikacji typów	73
4.6. Strukturalizacja specyfikacji	75
4.7. Przykłady specyfikacji	80
4.8. Uwagi końcowe.....	84
5. LOTOS – opis języka.....	86
5.1. Akcje komunikacyjne.....	86
5.2. Procesy i wyrażenia behawioralne	89
5.3. Semantyka operacyjna.....	93
5.4. Graf tranzycji	97
5.5. Strukturyzacja specyfikacji	102
5.6. Uwagi końcowe.....	107

6. Specyfikacja usług i protokołów sieciowych.....	109
6.1. Elementy modelu referencyjnego ISO/OSI	109
6.2. Reprezentacja modelu referencyjnego	113
6.3. Przykłady specyfikacji	117
6.4. Uwagi końcowe.....	128
7. Metodyka specyfikowania.....	129
7.1. Proces specyfikowania	129
7.2. Równoważność obserwacyjna.....	130
7.3. Równoważność testowa i implementacyjna	133
7.4. Style specyfikowania.....	136
7.5. Przykładowy problem	139
7.6. Środowiska wspomagające specyfikowanie w LOTOSie	143
7.7. Uwagi końcowe.....	145
8. Problem blokad w LOTOSie.....	146
8.1. Ukryte blokady	146
8.2. Semantyki zmodyfikowane	147
8.3. Wykrywanie blokad	151
8.4. Uwagi końcowe.....	159
9. Rozszerzenia czasowe LOTOSa.....	160
9.1. Wstęp	160
9.2. Składnia i semantyka języka TE-B-LOTOS	162
9.3. Wybrane własności języka TE-B-LOTOS	166
9.4. Składnia i nieformalna semantyka języka RT-B-LOTOS	168
9.5. Semantyka formalna języka RT-B-LOTOS	169
9.6. Wybrane własności języka RT-B-LOTOSa	177
9.7. Język RT-LOTOS	182
9.8. Przykład specyfikacji w RT-LOTOSie	183
9.9. Uwagi końcowe.....	189
10. Rozszerzenia wydajnościowe LOTOSa.....	191
10.1. Wstęp.....	191
10.2. Składnia i nieformalna semantyka języka MB-LOTOS	192
10.3. Semantyka formalna języka MB-LOTOS	197
10.4. Wyprowadzanie łańcuchów Markowa	204
10.5. Silna bisymulacja markowowska	213
10.6. Uwagi końcowe.....	216
11. Zakończenie.....	218
11.1. LOTOS a inne techniki formalne	218
11.2. LOTOS a UML	220
11.3. Diagramy stanów UML a LOTOS	222
11.4. Perspektywy technik formalnych	229
Dodatek 1.....	231
Dodatek 2.....	242
Literatura	252

Przedmowa

W monografii przedstawiono język specyfikacji formalnych LOTOS oraz omówiono na tym tle czasowe i wydajnościowe rozszerzenia języka.

Historia powstania języka LOTOS sięga końca lat osiemdziesiątych ubiegłego wieku. W tym okresie, zwłaszcza w ramach ISO, prowadzono intensywne prace nad architekturą otwartych sieci komputerowych. Głównymi elementami opracowanego modelu referencyjnego ISO/OSI (*Open Systems Interconnection*) był hierarchiczny układ warstw funkcjonalnych i związane z nimi pojęcia usług i protokołów warstwowych. Potrzeba precyzyjnego opisu usług i protokołów stała się inspiracją do opracowania formalnych technik specyfikacji. Pod egidą ISO uruchomiono międzynarodowe projekty, których rezultatem było opracowanie trzech języków specyfikacji: SDL [Belina, Hogrefe 1989], ESTELLE [ISO 9074] oraz LOTOS [ISO 8870]. Przegląd i porównanie tych języków zawiera książka [Turner 1993].

Pierwszy z nich – SDL – jest językiem półformalnym, opartym na notacji graficznej. Konstrukcja języka bazuje na koncepcji maszyn skończonego stanu. Był on rozwijany od początku lat siedemdziesiątych, najpierw przez nieistniejący już obecnie komitet CCITT (*International Consultative Committee on Telegraph and Telephony*), a później przez CCITT we współpracy z ISO. Wersja SDL-92 jest przedstawiona w rekomendacji Z100 [CCITT 1992].

Język ESTELLE jest językiem formalnym, stanowiącym – od strony notacyjnej – pewnego rodzaju rozszerzenie imperatywnych języków programowania. Modelem, na którym jest oparta konstrukcja języka, są automaty skończone. ESTELLE jest opisany między innymi w pracach [Budkowski 1992], [Budkowski, Dembiński 1987, 1989].

LOTOS jest również językiem formalnym, abstrahującym od języków programowania, opartym na solidnych, algebraicznych podstawach matematycznych. Na podstawy te składają się dwa elementy:

- koncepcja algebraicznej specyfikacji abstrakcyjnych typów danych, oparta na pracach H. Ehriga [Ehrig, Mahr 1985],
- koncepcja specyfikacji behawioralnych (oparta na pracach R. Milner) zawartych w języku CCS [Milner 1980].

Oprócz bezpośredniego wykorzystania do specyfikacji standardów sieciowych, każdy z wymienionych języków dał początek pracom, których wynikiem było opracowanie metodyk posługiwania się nimi przy specyfikacji systemów rozproszonych, opracowanie narzędzi wspierających tworzenie i badanie specyfikacji, a także narzędzi do transformacji specyfikacji w implementację oprogramowania.

Początkowo wydawało się, że języki te będą wykorzystywane nie tylko do specyfikacji standardów sieciowych, ale także w specyfikacji i projektowaniu dowolnych

systemów rozproszonych. Tak się jednak nie stało, a głównym powodem było to, że języki te, powstające w końcu lat osiemdziesiątych ubiegłego stulecia, opierały się na paradygmacie projektowania strukturalnego i zostały zdominowane przez języki oparte na wyłaniającym się w tym okresie paradygmacie obiektowym. Współcześnie rola tych języków uległa zmianie, gdyż projektowanie systemów informatycznych prowadzi się na podstawie podejścia obiektowego, a najpowszechniej używanym językiem do specyfikacji i projektowania jest język UML, utworzony w połowie lat dziewięćdziesiątych, który ciągle się rozwija. LOTOS rozwijał się równolegle z językiem UML; na początku bieżącego dziesięciolecia powstała rozszerzona wersja języka E-LOTOS [ISO/IEC 15437, 2001], istotnie modyfikująca sposób reprezentacji typów danych, ale – poza zmianą składni – pozostawiająca część behawioralną języka w zasadzie bez zmian.

Niezależnie od zastosowań praktycznych język LOTOS dał początek wielu pracom badawczym, do których między innymi należy rozwój algebr procesowych, dających podstawy dla analizy modelowej (*model checking*), formalnej semantyki aktualnie rozwijanych języków specyfikacji i projektowania, a także narzędzi wspomagających ich użycie [Garavel 1998], [Garavel, Lang, Mateescu 2001].

Wyróżnia się dwie wersje języka LOTOS: wersją bazową, która skupia się tylko na specyfikacji zachowań, abstrahując od komunikowanych danych, oraz wersję pełną, uwzględniającą komunikowane dane, a tym samym obejmującą definiowanie typów danych.

Zawartość książki jest następująca:

W rozdziale 1. przedstawiono tło dalej prowadzonych rozważań.

W rozdziałach 2. i 3. zaprezentowano język CCS oraz jego czasowe rozszerzenia. Na tle rozszerzeń czasowych spotykanych w literaturze przedstawiono opracowaną przez autora propozycję czasowego rozszerzenia CCS.

W dwóch kolejnych rozdziałach przedstawiono klasyczną wersję języka LOTOS. W rozdziale 4. omówiono abstrakcyjne typy danych, a w rozdziale 5. zawarto pełną prezentację języka LOTOS. Przykłady ilustrujące konstrukcje języka i ich zastosowanie pochodzą z wcześniejszych prac autora.

Zastosowania języka zaprezentowano w dwóch kolejnych rozdziałach: w rozdziale 6. omówiono ogólne zasady prezentacji standardów sieciowych w języku LOTOS, a w rozdziale 7. – metodykę stosowania języka. Podobnie jak w poprzednich dwóch rozdziałach rozważania są ilustrowane przykładami z prac autora.

W rozdziale 8. przeprowadzono krytyczną analizę semantyki pewnych konstrukcji języka LOTOS i zaproponowano pewną semantykę alternatywną. Opiera się ona na wspólnych pracach autora i Ludwika Kuźniarza.

Czasowe i wydajnościowe rozszerzenia języka LOTOS przedstawiono w kolejnych dwóch rozdziałach. W rozdziale 9. zaprezentowano dwa rozszerzenia czasowe:

pierwsze jest oparte na pracach, które powstawały w ramach projektów pod egidą ISO i które ostatecznie znalazły swój wyraz w definicji rozszerzonego języka LOTOS, drugie natomiast jest wynikiem prac autora prowadzonych wspólnie z Janem Magottem.

W rozdziale 10. omówiono rozszerzenie wydajnościowe języka LOTOS, którego podstawą były prace prowadzone wspólnie z Janem Magottem.

W ostatni rozdziale 11. krótko omówiono związki pomiędzy LOTOSEm a innymi technikami formalnymi.

Książka powstała w znacznym zakresie na podstawie prac indywidualnych autora oraz prowadzonych we współpracy z kolegami z Politechniki Wrocławskiej. Główny udział we wspólnych pracach ma profesor Jan Magott z Instytutu Informatyki, Automatyki i Robotyki, któremu składam serdeczne podziękowania za lata współpracy. Dziękuję również za współpracę i inspirację drowi Ludwikowi Kuźniarzowi, drowi inż. Zdzisławowi Sptawskiemu i dr inż. Bogumile Hnatkowskiej z Instytutu Informatyki Stosowanej.

Zbigniew Huzar

1. Wstęp

1.1. Wytwarzanie oprogramowania

Oprogramowanie (system oprogramowania) jest zwykle fragmentem systemu informatycznego, składającego się z oprogramowania i sprzętu, który dla oprogramowania stanowi środowisko wykonawcze.

Cykl życia systemu informatycznego jest zestawem czynności związanych z jego wytwarzaniem i użytkowaniem – od chwili podjęcia decyzji o jego wytworzeniu, przez konstrukcję, wdrożenie, eksploatację, aż do wycofania z użycia. Cykl życia oprogramowania należy widzieć na tle życia systemu informatycznego. Ponieważ wytwarzanie oprogramowania jest zwykle znacznie bardziej złożone niż projektowanie części sprzętowej, często cykl życia systemu informatycznego utożsamia się z cyklem życia oprogramowania.

Do grupy tradycyjnych modeli cyklu życia oprogramowania można zaliczyć [Behforooz, Hudson 1996], [Jaskiewicz 1997], [Górski 2001], [Szejko 2002], [Linger, Lipson, McHugh, Mead, Sledge 2002]: klasyczny model kaskadowy i różne jego warianty – model V, model spiralny, model przyrostowo-ewolucyjny. Wśród wymienionych modeli cyklu życia oprogramowania szczególną rolę odgrywa klasyczny model *kaskadowy*, nazywany także *wodospadowym*, gdyż znajduje odzwierciedlenie w różnych metodykach wytwarzania oprogramowania.

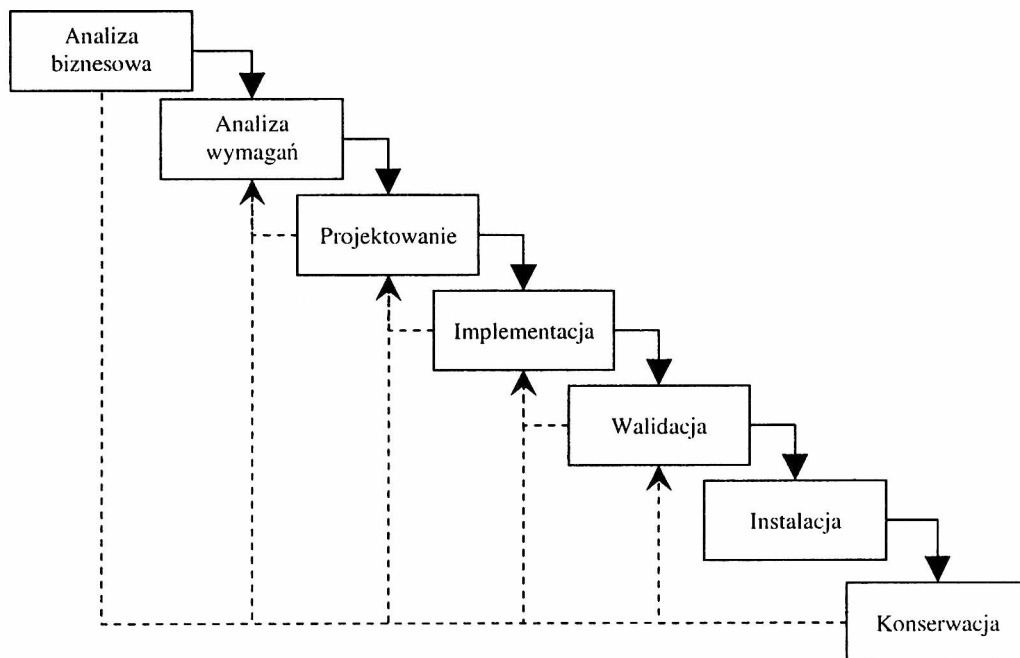
Model kaskadowy (rys. 1.1) wyróżnia następujące fazy życia oprogramowania:

- analizę domenową albo dziedzinową, niekiedy też nazywaną analizą biznesową,
- analizę wymagań,
- projektowanie,
- implementację albo kodowanie,
- walidację,
- instalację,
- konserwację (utrzymanie),
- wycofanie z użytkowania.

Cztery fazy – analizę wymagań, projektowanie, implementację i walidację – wyróżnia się jako fazy stanowiące cykl wytwarzania oprogramowania – fragment cyklu życia oprogramowania.

Analiza dziedzinowa (domenowa, biznesowa) wiąże się z miejscem – organizacją lub instytucją, dla której ma powstać system informatyczny. System taki angażuje ludzi, sprzęt komputerowy i oprogramowanie, realizując pewien obieg i przetwarzanie danych. Celem analizy jest ustalenie zasad funkcjonowania danej organizacji, ustalenie tych jej fragmentów, które mają być poddane informatyzacji, a także określenie ogólnych oczekiwań w stosunku do mającego powstać systemu.

Analiza wymagań odnosi się do przewidywanego systemu informatycznego. Polega na rozpoznaniu tych wszystkich aspektów rzeczywistości, które mogą mieć wpływ na postać oczekiwanego systemu lub na sposób jego budowy, wdrożenia bądź funkcjonowania. Wynikiem analizy jest określenie wymagań użytkownika do systemu informatycznego.



Rys. 1.1. Model kaskadowy cyklu życia oprogramowania

Projektowanie polega w pierwszej kolejności na określeniu architektury systemu informatycznego, czyli składowych systemu i wzajemnego ich powiązania. Wyróżnia się składowe sprzętowe – węzły (komputery lub urządzenia wejścia–wyjścia) i składowe informacyjne – komponenty (programy, bazy danych, biblioteki programowe). Komponenty są odpowiednio rozmieszczane w węzłach. Powiązania między węzłami mogą być realizowane przez pojedyncze łącza lub sieci komunikacyjne. Powiązania pomiędzy komponentami opierają się na wzajemnym świadczeniu usług. W następnej kolejności projektowanie polega na szczegółowej specyfikacji poszczególnych składowych architektury. Specyfikacja powinna jednoznacznie określić rodzaj wymaganego sprzętu oraz funkcje poszczególnych komponentów oprogramowania tak, aby można napisać odpowiedni kod w wybranym języku programowania. Faza projektowania może być złożona, dlatego niektóre metodyki dekomponują projektowanie na podfazy, na przykład metodyka ROPES [Douglass 1999], opisana też w pracy [Huzar 2001], wyróżnia projektowanie ogólne, pośrednie i szczegółowe.

W ramach *implementacji* wyróżnia się dwa rodzaje czynności: kodowanie i testowanie wstępne. Kodowanie polega na napisaniu tekstu programów odpowiednich komponentów w wybranych językach programowania, a testowanie wstępne polega na eksperymentalnym sprawdzeniu czy system działa poprawnie. Wyróżnia się przy tym testowanie jednostkowe, polegające na oddzielnym sprawdzeniu poprawności poszczególnych komponentów programowych, oraz testowanie integracyjne, polegające na sprawdzeniu poprawności współpracy wszystkich komponentów.

Faza walidacji ma na celu sprawdzenie czy zbudowany system działa poprawnie w środowisku użytkownika oraz czy rzeczywiście spełnia jego oczekiwania. Sprawdzenie to polega na wykonaniu testów akceptacyjnych. W testowaniu w poprzedniej fazie chodziło o stwierdzenie czy system działa zgodnie ze specyfikacją ustaloną podczas projektowania, w obecnej fazie idzie natomiast o stwierdzenie spełnienia oczekiwań użytkownika, które – w odróżnieniu od specyfikacji projektowych – być może nie zostały całkowicie precyzyjnie ustalone w wymaganiach użytkownika. Przeprowadzane tu testowanie ma charakter walidacyjny, natomiast testowanie w fazie implementacyjnej ma charakter weryfikacyjny.

Faza instalacji wiąże się z wdrożeniem zrealizowanego i wytestowanego systemu w środowisku użytkownika. Wdrażanie nowego systemu może polegać na okresowym jednoczesnym funkcjonowaniu systemu starego i nowego. Jest to strategia kosztowna, ale bezpieczna dla funkcjonującej instytucji. Możliwe są oczywiście inne strategie, lecz z ich zastosowaniem wiąże się potrzeba oszacowania ryzyka niepowodzenia przedsięwzięcia.

Faza konserwacji (utrzymania) jest związana z eksploatacją oprogramowania. Wykonywanymi tu czynnościami są generacja i instalacja oprogramowania w konkretnym środowisku wykonawczym, rekonfiguracja oprogramowania powodowana zmieniającymi się warunkami eksploatacji, usuwanie zauważonych błędów, a także modyfikacje oprogramowania na skutek zmieniających się wymogów użytkowych. W przypadku istotnych modyfikacji należy dokonać powrotu aż do fazy analizy. Właśnie ten fakt jest główną wadą modelu, gdyż błędy poczynione w etapie analizy wymagań powodują duże straty finansowe.

Fazy te – w zasadzie – występują po sobie kolejno; wynikowe artefakty – dowolne informacje w materialnej postaci – danej fazy stanowią podstawę do prac w fazie następnej. W praktyce często fazy nakładają się na siebie, a dodatkowo dopuszcza się powroty z danej fazy do faz wcześniejszych w celu poprawek, uściśleń lub uzupełnień wcześniej opracowanych artefaktów (rys. 1.1). Przyczyną powrotu może być wykrycie niespójności, niejednoznaczności lub braku określoności w artefaktach poprzednich faz. Powroty do faz wcześniejszych pociągają za sobą dodatkowe koszty, dlatego stale udoskonala się metody wytwarzania oprogramowania tak, aby redukować potrzebę powrotów do wcześniejszych faz.

Model kaskadowy wprowadza nie tylko pewną kolejność wykonywania różnych czynności, ale także określa pewne ramy organizacyjne związane z wyznaczaniem zespołów,

przydzielania im odpowiednich zasobów i rozliczania rezultatów prac. Model jest podstawą do wyróżnienia różnych kategorii specjalistów zaangażowanych w budowę i funkcjonowanie systemów informatycznych, dlatego wyróżnia się na przykład analityków, projektantów, programistów, inspektorów, administratorów czy konserwatorów.

1.2. Specyfikacja wymagań

Precyzyjna specyfikacja wymagań jest niezbędna nie tylko do uzyskania systemu oprogramowania odpowiedniej jakości, ale także do efektywności całego cyklu wytwarzania oprogramowania [Wallace, Recker 2001]. Zwykle wyróżnia się specyfikację wymagań funkcjonalnych i нефunkcjonalnych, ale spotyka się także inne podejścia [Sawyer, Kotonaya 2001]. Można wyróżnić przynajmniej trzy grupy podejść dotyczące tego, co specyfikacja powinna ujmować:

- normy ISO (International Organization for Standardization) serii 9621,
- normy IEC (International Electrotechnical Commission) serii 50-191 i 300, odnoszące się do systemów związanych z bezpieczeństwem,
- inne dotyczące specyficznych systemów, na przykład systemów ekspertowych czy systemów informacyjnych.

Najszerze jest podejście odzwierciedlone w serii norm ISO 9621, które specyfikację wymagań rozpatruje w kontekście jakości systemu oprogramowania. Wyróżnia się tu sześć głównych charakterystyk jakości, które z kolei dzieli się na 21 podcharakterystyk. Głównymi charakterystykami i ich wybranymi podcharakterystykami są:

- *funkcjonalność*, jako podstawowa charakterystyka, obejmuje następujące podcharakterystyki: dostarczenie odpowiednich funkcji (usług) wraz z zapewnieniem dostarczania odpowiednio dokładnych wyników obliczeń, współdziałanie z innymi systemami, zabezpieczenie przed nieuprawnionym dostępem do danych,
- *niezawodność*, rozumiana jako zdolność do świadczenia usług w określonych warunkach na odpowiednim poziomie efektywności, obejmuje: dojrzałość, odporność na awarie, zdolność przywracania ze stanu po awarii,
- *użytkowalność*, reprezentując punkt widzenia użytkowników systemu, odnosi się do: łatwości nauczania posługiwania się systemem, łatwości użytkowania, atrakcyjności usług dla użytkownika,
- *efektywność*, odnosząc się do związku pomiędzy wydajnością systemu a stopniem wykorzystywania zasobów, obejmuje własności czasowe (szybkość reakcji, przepustowość), wykorzystywanie zasobów systemu,
- *konserwowalność*, odnosząc się do nakładu pracy potrzebnego do utrzymania i modyfikacji systemu, obejmuje takie podcharakterystyki, jak: analizowalność, modyfikowalność, testowalność,
- *przenośność*, charakteryzując zdolność oprogramowania do przenoszenia do innych organizacji czy platform wykonawczych, wskazuje na podcharakterystyki:

adaptowalność, instalowalność, współistnienie z innym oprogramowaniem, zastępowalność.

Dodatkowo zakłada się, że w ramach każdej z charakterystyk mogą być definiowane podcharakterystyki określające zgodność oprogramowania ze specyficznymi standardami lub zaleceniami.

Normy serii ISO 9621 definiują dokładniej wymienione charakterystyki i podcharakterystyki oraz proponują zestaw metryk, którymi można się posługiwać w celu ich ilościowej oceny. Specyfikacja systemu (w terminach normy ISO specyfikacja jego jakości) jest wyrażona przez zbiór pożądanych wartości metryk [Dubielewicz, Hnatkowska, Huzar, Tuzinkiewicz 2006].

Normy IEC dotyczą systemów związanych z bezpieczeństwem, które często są systemami czasu rzeczywistego [Douglass 1998, 1999]. Specyfikacja wymagań obejmuje wymagania funkcjonalne i niefunkcjonalne, które z kolei dzielą się na dwie kategorie:

- wydajnościowe,
- wiarygodnościowe.

Do wymagań *wydajnościowych* można na przykład zaliczyć: przepustowość systemu, czasy reakcji i czasy obsługi użytkowników. Tego rodzaju wymagania są oczywiście związane z budową systemów czasu rzeczywistego, ale także z budową systemów nieczasowych.

Na wymagania *wiarygodnościowe* składają się: dyspozycyjność, niezawodność, bezpieczeństwo (*safety*) i ochrona (*security*).

Dyspozycyjność jest miarą gotowości systemu do użycia. Przykładami systemów o ostrych wymogach dyspozycyjności są system sterowania reaktorami w elektrowniach jądrowych czy system sterowania międzynarodowym lub międzymiastowym ruchem telekomunikacyjnym. Systemy takie powinny pracować całą dobę bez przerw, przez wszystkie dni roku.

Niezawodność określa odpowiednio długie okresy czasu bezawaryjnej pracy. Bezpośrednio pojęcia niezawodności nie można odnosić do oprogramowania, gdyż oprogramowanie może być tylko poprawne albo niepoprawne, ale oprogramowanie, funkcjonując w zawodnym środowisku wykonawczym, powinno przewidywać awarie środowiska wykonawczego i reagować na ich zajście w taki sposób, aby zachować poprawność wykonywanych funkcji. Przykładami systemów, z którymi wiąże się taki wymóg, są systemy sterowania raketami kosmicznymi.

Wymóg *bezpieczeństwa* wiąże się w pewnym zakresie z wymogiem niezawodności. Oznacza on również odpowiednie reagowanie na awarie, przy czym nie chodzi o awarie środowiska wykonawczego, ale o awarie w otoczeniu systemu, na które system powinien reagować. W razie zaistnienia takich awarii system powinien podejmować działania specyficzne, odbiegające od rutynowych. Przykładem może być system ste-

rowania reaktorem atomowym, który w przypadku stwierdzenia zajścia krytycznych lub niedopuszczalnych zjawisk w reaktorze powinien rozpocząć procedurę awaryjnego wyłączenia reaktora. Na ogół wymóg bezpieczeństwa odnosi się do tych systemów, których funkcjonowanie jest związane z bezpieczeństwem ludzi lub środowiska.

Ochrona (zabezpieczenie) systemów wiąże się z zapobieganiem nieupoważnionemu dostępowi i manipulacji na danych pamiętanych w systemie. Systemy bankowe lub osobowe są typowymi przykładami systemów, w których zabezpieczenie odgrywa szczególną rolę.

Oprócz wymogów wydajnościowych i wiarygodnościowych mogą być formułowane inne, na przykład ważnym wymogiem, który można obecnie spotkać, w stosunku do wielu systemów jest skalowalność. *Skalowalność* oznacza, że w przypadku rozbudowy system zachowuje poprzednio wymienione wymogi. Skalowalność może być pełna lub częściowa, w zależności od tego czy żąda się zachowania wszystkich, czy tylko niektórych wymogów. Można na przykład żądać, aby w przypadku zwiększenia liczby użytkowników system gwarantował stałe czasy obsługi, zachowywał dyspozycyjność oraz niezawodność itp.

W przypadku systemów specyficznych istotne mogą być inne charakterystyki. Na przykład w systemach ekspertowych ważne jest uzyskanie odpowiedzi w ustalonym limicie czasu, a także czytelne przedstawienie procesu wnioskowania, w systemach informacyjnych natomiast ważną cechą jest łatwość formułowania zapytań i precyzyja udzielanych odpowiedzi.

Wymagania niefunkcjonalne mogą się odnosić nie tylko do jakości systemu, ale także do procesu jego wytwarzania. Wymagania takie mogą się odnosić metodyki projektowania, używanych narzędzi wspomagających, sposobu dokumentowania, gromadzenia i udostępniania artefaktów itp.

1.3. Modele systemów

Współczesne metody wytwarzania systemów oprogramowania opierają się na paradygmacie modelowania obiektowego. Oznacza to, że cykl wytwarzania oprogramowania jest postrzegany jako proces budowy ciągu modeli.

Model jest pewnym odzwierciedleniem wybranej dziedziny – jest pewną *abstrakcją* modelowanej dziedziny, świadomym uproszczeniem modelowanej dziedziny, a uproszczenie to zależy od przyjętego punktu widzenia – od wybranej *perspektywy* modelowania.

Perspektywa, która jest punktem wyjścia przy tworzeniu modeli, określa te aspekty należące do modelowanej dziedziny, które są istotne dla danego modelu, i te, które należy pominąć. Perspektywa ustala też stopień szczegółowości przedstawiania wybranych aspektów.

Od modelu oczekuje się, że odzwierciedla dwa różne aspekty modelowanego systemu: aspekt statyczny i aspekt dynamiczny. Rozdzielenie obu tych aspektów można traktować jako zastosowanie zasady abstrakcji. Aspekt statyczny odnosi się do struktury modelowanego systemu, czyli ukazuje zbiór elementów składowych wraz z wiążącymi je relacjami. Aspekt dynamiczny (behawioralny) odnosi do zachowania systemu, czyli do obserwowalnych efektów jego działania w czasie.

Językiem, który obecnie jest używany najpowszechniej do reprezentacji modeli obiektowych w procesie wytwarzania oprogramowania jest UML (*Unified Modeling Language*) [UML 2003]. UML ma wiele zalet, ale ma także pewne ograniczenia. Głównym jego ograniczeniem jest to, że w fazie analizy wymagań pozwala w zasadzie tylko na definiowanie wymagań funkcjonalnych. Dodatkowo jego słabością jest brak formalnej semantyki. Omawiany w książce język LOTOS nie jest językiem obiektowym, chociaż ma wiele cech wspólnych z językami obiektowymi [Hnatkowska 1998]. W stosunku do języka UML jest językiem uboższym, gdyż jest tylko językiem specyfikacji funkcjonalnych, podczas gdy UML jest językiem specyfikacji i projektowania. LOTOS ma natomiast nad językiem UML wyraźną przewagę w zakresie precyzyjnej i kompletnej specyfikacji zachowań. LOTOS jest ponadto językiem wykonywanym, co pozwala na szybkie prototypowanie, testowanie i analizę specyfikowanych zachowań.

Systemy informatyczne są ogromnie zróżnicowane. Istnieje wiele klasyfikacji koncentrujących się na wybranych aspektach ich architektury lub zastosowań. Spotyka się na przykład podziały systemów na:

- sprzętowe/programowe,
- jedno-/wieloprocesorowe,
- sekwencyjne/wielowątkowe,
- tradycyjne/czasu rzeczywistego,
- wbudowane/lokalne/rozproszone.

Przedstawiony podział nie jest jednorodny, ukazuje on raczej bogactwo wewnętrznych własności i zastosowań systemów informatycznych.

Ze względu na zakres dalej prowadzonych rozważań przedstawimy inną, abstrakcyjną klasyfikację modeli systemów z punktu widzenia ich współpracy z otoczeniem. W klasyfikacji tej system jest traktowany jako *czarna skrzynka*, która komunikuje się z otoczeniem przez wyróżnione punkty komunikacji – bramki komunikacyjne. Takie widzenie skupia się na usługach systemu, a nie na tym, w jaki sposób usługi te są realizowane. Klasyfikacja wyróżnia trzy kategorie systemów [Huzar, Sławski 1989], [Schneider 2004]:

- transformacyjne,
- interakcyjne,
- reaktywne.

Działanie *systemów transformacyjnych* polega na przetworzeniu zadanej wartości wejściowej i wygenerowaniu pewnej wartości wyjściowej. Wartości wejściowe i wyjściowe są elementami pewnych wskazanych zbiorów. Wartości te mogą być wartościami złożonymi. Nie wnikając w ich wewnętrzną strukturę i przyjmując, że X jest zbiorem wartości wejściowych, a Y wyjściowych, schemat funkcjonowania systemów transformacyjnych można określić bądź jako funkcję częściową $f: X \rightarrow Y$, bądź jako relację $R \subseteq X \times Y$. Przez funkcję f jest opisywany system deterministyczny, jeśli $f(x) = y$, to oznacza, że dla danej wejściowej wartości x system generuje y jako wartość wyjściową. Częściowość funkcji oznacza, że dla wartości wejściowej x , dla której funkcja f jest nieokreślona, system nie kończy działania i nie dostarcza tym samym wyniku końcowego. Przez relację R jest opisywany system niedeterministyczny, jeśli $\langle x, y \rangle \in R$, to dla danej wejściowej wartości x system może generować y jako wartość wyjściową. Jeśli dla danego $x \in X$ nie istnieje $y \in Y$ takie, że $\langle x, y \rangle \in R$, to oznacza, że – podobnie jak w przypadku funkcji – system nie dostarcza wyniku końcowego.

Działanie *systemów interaktywnych* jest bardziej złożone niż systemów transformacyjnych. System nie wytwarza pojedynczej wartości, ale współdziała ze swoim otoczeniem. Współdziałanie odbywa się za pośrednictwem bramek komunikacyjnych, przez które otoczenie kieruje dane wejściowe do systemu, a system po ich przetworzeniu generuje dane wyjściowe do otoczenia. System przyjmuje dane wejściowe, a dopiero po ich przetworzeniu, i ewentualnym wygenerowaniu danych wyjściowych, jest gotowy do pobrania kolejnych danych wejściowych. Istotne we współdziałaniu systemu z otoczeniem jest to, że kolejna wymiana danych odbywa się po zakończeniu przetwarzania przez system poprzednio wymienionych danych. Działanie systemu nie musi się zakończyć po skończonej liczbie takich cykli. Przyjmując, jak w przypadku systemów transformacyjnych, że X jest zbiorem wartości wejściowych, a Y wyjściowych, schemat funkcjonowania deterministycznych systemów reaktywnych można wyrazić przez funkcję o sygnaturze $f: X^* \rightarrow Y^*$, gdzie X^* jest zbiorem skończonych ciągów nad X . System interaktywny przetwarza nie pojedyncze wartości, ale ciągi wartości. Charakteryzująca system funkcja f musi przy tym spełniać warunek domkniętości prefiksowej

jeżeli $f(x_1, \dots, x_n) = y_1, \dots, y_n$, oraz $f(x_1, \dots, x_n, x_{n+1})$ jest określona,
to $f(x_1, \dots, x_n, x_{n+1}) = y_1, \dots, y_n, y_{n+1}$.

Dla niedeterministycznych systemów interaktywnych schemat ich działania określa się za pomocą relacji $R \subseteq X^* \times Y^*$, spełniającej analogiczny warunek.

Działanie *systemów reaktywnych* jest jeszcze bardziej złożone niż systemów interaktywnych. Wynika to z nałożenia na system dodatkowych ograniczeń czasowych. System nie tylko powinien generować wartości wynikowe, ale generować je w odpowiednim czasie, a otoczenie może kierować do systemu kolejne wartości wejściowe w dowolnie określonych chwilach, niezależnie od tego, czy system przetworzył poprzednio do niego skierowane wartości wejściowe. Wejścia i wyjścia systemu są okre-

ślone parami: wartość, chwila czasowa. Wprowadzając zbiór chwil czasowych T , wraz z relacją porządku częściowego $\leq$, oraz oznaczając, jak poprzednio, X jako zbiór wartości wejściowych, a Y wyjściowych, schemat funkcjonowania deterministycznych systemów reaktywnych może być wyrażony funkcją o sygnaturze

$$f: (X \times T)^* \rightarrow (Y \times T)^*.$$

Funkcja może być określona tylko dla ciągów wejściowych $\langle x_1, t_1 \rangle, \dots, \langle x_n, t_n \rangle$ takich, że $t_1 \leq \dots \leq t_n$. Ten sam warunek czasowego uporządkowania spełniają również ciągi wyjściowe. W taki sposób określona funkcja musi spełniać określony wcześniej warunek domkniętości prefiksowej.

Systemy interaktywne i reaktywne najczęściej nie produkują określonego wyniku, lecz reagują na dane kierowane do nich z otoczenia, w którym funkcjonują (przykładem programów tego typu są systemy operacyjne, systemy rezerwacji biletów, poczty elektronicznej itp.).

Głównym przedmiotem naszego zainteresowania będą systemy reaktywne. Wśród nich należy wyróżnić dwie kategorie systemów:

- systemy z silnymi ograniczeniami czasowymi,
- systemy ze słabymi ograniczeniami czasowymi.

Do systemów z *silnymi ograniczeniami czasowymi* zalicza się te systemy, dla których muszą być zawsze spełnione zadane ograniczenia czasowe. Naruszenie tych ograniczeń może powodować niepożądane, katastrofalne skutki. Przykładami takich systemów są systemy sterowania elektrowniami jądrowymi, systemy awioniki lotniczej, czy też ogólniej – systemy związane z bezpieczeństwem. Brak reakcji lub spóźniona reakcja na pewne zdarzenia może powodować na przykład awarię elektrowni lub niebezpieczeństwo utraty sterowności samolotu.

Do systemów ze *słabymi ograniczeniami czasowymi* zalicza się te systemy, dla których zadane ograniczenia czasowe powinny być spełnione, ale których naruszenie, powodując pewne straty, nie pociąga jednak katastrofalnych skutków. Przykładami takich systemów są systemy obsługi bankowej czy rezerwacji biletów. W takich przypadkach żąda się wprowadzić obsługi użytkownika w pewnym ograniczonym czasie, ale przekroczenie takiego ograniczenia, na przykład czasu reakcji, skutkuje tylko niezadowolaniem klienta.

1.4. Specyfikacja funkcjonalna systemów oprogramowania

Specyfikacja funkcjonalna opisuje zewnętrznie dostrzegalne efekty działania programu. Możliwe są dwa sposoby rozumienia tych efektów [Huzar, Spławski 1989]: pierwszy zakłada, że obserwowalne są *zmiany stanów* systemu, drugi – że obserwowane są *interakcje* pomiędzy systemem a jego otoczeniem. Obserwacja stanów zakłada

wgląd we wnętrze systemu, natomiast obserwacja interakcji jest punktem widzenia użytkownika systemu – mówimy w tym przypadku, że obserwujemy zachowanie systemu. W dalszym ciągu zakładamy, że interesuje nas wyłącznie punkt widzenia zewnętrznego obserwatora systemu.

Opis zachowania systemu jest konieczną składową specyfikacji systemu. Wymagania funkcjonalne są określane na dwa powiązane ze sobą sposoby:

Sposób pierwszy odwołuje się do pojęcia usług systemu. System opisuje się tu przez określenie usług, jakich system dostarcza, a także usług, z jakich ma korzystać. Samą usługę wyraża się w terminach zachowań – ciągów komunikatów wymienianych pomiędzy usługobiorcą a usługodawcą w celu realizacji żądanej usługi. Pojęcie usługi jest wygodne i naturalne przy opisie szerokiej klasy systemów służących wspomaganiu jednostek administracyjnych czy gospodarczych.

Sposób drugi odwołuje się bezpośrednio do pojęcia zachowań systemu. Jest to sposób naturalny przy specyfikacji systemów sterujących, na przykład obiektami technologicznymi. W tym przypadku wskazuje się na zachowania wymagane, a także można wskazywać na inne zachowania jako dopuszczalne lub zabronione.

Dla uproszczenia będziemy dalej prowadzić rozważania, zakładając, że schemat funkcjonowania systemu jest funkcją. Nie ogranicza to ogólności rozważań w tym sensie, że przedstawiane rozważania będzie można również zastosować do systemów, których schemat funkcjonowania jest określony relacją. Opis systemu oznacza więc opis funkcji wyrażającej jego schemat funkcjonowania. Podobnie jak w przypadku definiowania dowolnych zbiorów, istnieją dwa podejścia definiowania funkcji [Huzar 2002]:

- rekursywne,
- ekstensjonalne.

Podejście pierwsze – *rekursywne (algorytmiczne, wykonywalne)* – polega na przedstawieniu sposobu generowania wszystkich wartości funkcji na podstawie przyjętego zestawu wartości dla wybranych argumentów.

Podejście drugie – *ekstensjonalne (logiczne)* – polega na zdefiniowaniu funkcji przez określenie jej własności.

Za obu podejściami kryje się różnorodność wykorzystywanych języków i metod.

Do pierwszego podejścia należą metody wykorzystujące pojęcia abstrakcyjnych maszyn stanowych (rozszerzeń automatów skończonych) [Börger, Stärk 2003], [Harel, Marely 2003] etykietowanych systemów przejść oraz algebr procesowych [Baeten, Middelburg 2002], [Hermanns, Herzog, Katoen, 2002]. Do grupy tej można także zaliczyć sieci Petriego i różnorodne ich rozszerzenia [Reisig 1988], [Girault 2003], [Magott 2005].

Do drugiego podejścia należą metody oparte na logikach klasycznych i modalnych. Należą do nich na przykład logika programów Hoare’a [Huzar 1989], [Apt, Olderog

1991] i rachunek najłagodniejszych warunków wstępnych Dijkstry [Dijkstra 1978], mające zastosowanie do systemów transakcyjnych. Do systemów interaktywnych i reaktywnych są stosowane różne formy logiki temporalnej [Hennessy, Milner 1985], [Manna, Pnueli 1992], [Clark, Emerson, Sistla 1983], a także inne, jak na przykład logika dynamiczna [Harel 1979], [Harel, Kozen, Tiuryn 2000] czy μ -rachunek [Schneider 2004].

Oba podejścia są w pewnym sensie komplementarne i dlatego bardzo często do opisu systemu stosuje się oba podejścia. Podejście rekursywne jest wprawdzie bardzo wygodne przy przechodzeniu od specyfikacji do implementacji, ale bezpośrednio jest trudne do analizy, w celu sprawdzenia czy ma pewne pożądane własności. Podejście logiczne – odwrotnie – bezpośrednio może wyrażać pożądane własności, natomiast bezpośrednio jest trudne do przekształcenia w implementację. Jednoczesny opis wykonywalny i logiczny wprowadza redundancję, co pomaga w uzyskaniu większej wiarygodności, ale jednocześnie wymaga stwierdzenia zgodności obu opisów.

Do specyficznych kategorii własności systemów, które wyraża się w sposób ekstensjonalny należą między innymi własności:

- bezpieczeństwa,
- żywotności,
- trwałości,
- bezstronności (uczciwości).

Własność *bezpieczeństwa* jest tutaj rozumiana, inaczej niż poprzednio (p. 1.3), jako niezmiennik wyrażający to, że w działaniu systemu *nigdy nie osiągnie się niepożądaną sytuację (nigdy nie zdarzy się coś złego)*. Określenie tego, co jest złe, zależy oczywiście od konkretnego systemu, na przykład że program nigdy nie zablokuje swego działania.

Własność *żywotności* jest niezmiennikiem wyrażającym to, że w działaniu systemu *zawsze osiągnie się pożądaną sytuację (zawsze zdarzy się coś dobrego)*. Na przykład program zakończy swoje działanie lub osiągnie określony stan.

Trwałość odnosi się do stabilizacji pewnych własności. Oznacza to, że pewna własność będzie trwale zachodzić od pewnego momentu czasu działania systemu. Na przykład będą trwale przechowywane raz zarejestrowane dane.

Bezstronność oznacza, że pewna własność będzie w czasie działania systemu zachodzić nieskończenie wiele razy. Własność ta jest wyprowadzona z analizy działania systemów równoległych i wyraża postulat, że każdemu z równoległych procesów system operacyjny będzie przydzielał dostęp do procesora.

W dalszym ciągu nie będziemy się zajmować własnościami logicznymi, a będziemy się głównie opierać na *algebrach procesowych*, w ramach podejścia wykonywalnego, które się odnosi do systemów interaktywnych i reaktywnych.

Algebry procesowe są rozwijane od początku lat osiemdziesiątych ubiegłego stulecia. Za pionierskie można uznać prace Hoare'a związane z procesami CSP [Hoare 1978,

1985], Milnera – wprowadzające rachunek CCS [Milner 1980], [Milner 1989] czy też prace [Bergstra, Klop 1985], [Baeten, Bergstra, Smolka 1995] definiujące algebrę ACP. Początkowo algebry związane głównie z systemami interaktywnymi zostały przeniesione na grunt systemów reaktywnych. Przykładem pracy pokazującej zastosowanie ACP do specyfikacji systemów czasu rzeczywistego jest [Groote 1990]. Historię prac nad rozwojem algebr procesowych przedstawia raport [Baeten 2004].

Obecnie można wyróżnić trzy kategorie algebr procesowych: czasowe, probabilistyczne i stochastyczne.

Algebry czasowe powstały jako uogólnienie klasycznych algebr procesowych przez wprowadzenie do algebry pojęcia czasu i akcji czasowych. Różnice pomiędzy algebrami czasowymi dotyczą przyjmowanych modeli czasu (czas ciągły i dyskretny) oraz interpretacji wykonywania akcji czasowych. Przykładami są algebry oparte na wcześniej wymienionych językach: ACP [Baeten, Bergstra 1991], CSP [Schneider 1995], LOTOS [Bolognesi, Lucidi, Trigilla 1995], [Leduc, Leonard 1995]. Powodem rozwoju algebr czasowych była potrzeba dostarczenia formalnego języka specyfikacji systemów reaktywnych, czyli specyfikacji zachowań i związanych z nim ograniczeń czasowych. Algebry czasowe są właściwym narzędziem specyfikacji systemów czasu rzeczywistego z silnymi ograniczeniami czasowymi. Przegląd prac dotyczących algebr czasowych zawiera publikacja [Nicollin, Sifakis 1992].

Algebry probabilistyczne są specjalizacją algebr czasowych polegającą na zastąpieniu niedeterminizmu występowania akcji probabilizmem. Takie podejście umożliwia badanie wydajności specyfikowanych systemów, a zwłaszcza wyliczanie charakterystyk probabilistycznych [Hermanns, Herzog, Katoen 2002]. Algebry probabilistyczne są właściwym narzędziem specyfikacji systemów czasu rzeczywistego ze słabymi ograniczeniami czasowymi. Przykładami takich algebr są: dla ACP [Baeten, Bergstra, Smolka 1995], dla CSP [Hanson, Jonsson 1990] oraz dla LOTOSa [Miguel, Fernandez, Vidaller 1993].

Kolejne uogólnienie algebr procesowych – *algebry stochastyczne* – polega na wprowadzeniu akcji o stochastycznej charakterystyce czasów wykonania. Algebry te stanowią właściwe narzędzie analizy systemów kolejkowych. W odróżnieniu od innych mechanizmów, na przykład stochastycznych sieci Petriego, umożliwiają one kompozycyjną specyfikację i generację łańcuchów Markowa. Przykładami takich algebr są: TIPP [Götz, Herzog, Rettelbach 1993], PEPA [Hilston 1996], EMPA [Bernardo, Gorrieri 1998], a dla LOTOSa [Rico, Bochmann 1991].

Głównym obszarem zastosowania algebr czasowych jest specyfikacja systemów czasu rzeczywistego z silnymi ograniczeniami czasowymi, natomiast algebr probabilistycznych i stochastycznych – systemy ze słabymi ograniczeniami czasowymi.

W ostatnim okresie, oprócz rozwijania klasycznych algebr ACP, CCS i CSP, powstała nowa grupa algebr procesowych, do których należą między innymi π -rachunek Mil-

nera [Milner 1999] oraz rachunek otoczeń (*ambient calculus*) Cardelliego i Gordona [Cardelli, Gordon 1998].

1.5. Metody formalne w procesie wytwarzania systemów

Wytwarzanie systemów informatycznych jest złożonym procesem. Różne metodyki definiują strukturę takich procesów przez określenie faz (etapów), czynności i wynikowych artefaktów. Współcześnie dominujące obiektowe metodyki wytwarzania systemów oprogramowania opierają się na paradygmacie modelowania. Oznacza to, że proces wytwarzania systemów oprogramowania jest widziany jako pewien ciąg powiązanych ze sobą modeli. Typowe powiązania pomiędzy modelami to relacja uściślenia i realizacji.

W różnych miejscach procesu wytwarzania znajdują zastosowanie metody formalne.

Formalność metody oznacza w pierwszej kolejności *formalność* języka użytego do opisu modeli. Oznacza to, że język ma ściśle zdefiniowaną składnię i semantykę, dzięki czemu opis systemu w tym języku może być *precyzyjny* i *jednoznaczny*. Język powinien mieć dostateczną *siłę ekspresji* po to, aby dało się tworzyć modele *kompletne* z punktu widzenia przyjętej perspektywy. Jednocześnie powinien być możliwie *abstrakcyjny*, aby unikać wnikania w szczegóły dotyczące implementacji systemu.

Powody stosowania metod formalnych są przynajmniej dwa: po pierwsze – formalizacja modeli umożliwia precyzyjne formułowanie i badanie ich własności, oraz po drugie – umożliwia definiowanie transformacji pomiędzy modelami. Oznacza to, że metody formalne w wytwarzaniu oprogramowania stosuje się do:

- pisania specyfikacji formalnych,
- dowodzenia własności tych specyfikacji,
- wyprowadzania implementacji na podstawie specyfikacji,
- weryfikacji zgodności implementacji względem specyfikacji.

Metody formalne mają zalety, ale mają też wady: w stosunku do języków naturalnych lub półformalnych są mniej ekspresywne i są *szttywne* – wymagają ścisłego, ograniczonego sposobu posługiwania się nimi podczas tworzenia modeli.

Stosowanie metod formalnych jest szczególnie uzasadnione w przypadku systemów współbieżnych, związanych z bezpieczeństwem, a także w przypadku definiowania pewnych standardów. Właśnie potrzeba ścisłego zdefiniowania standardów systemów otwartych ISO/OSI doprowadziła pod koniec lat osiemdziesiątych ubiegłego wieku do podjęcia prac nad rozwojem formalnych technik specyfikacji usług i protokołów sieci komputerowych. Powstały wówczas, rozwijane do dzisiaj, techniki formalne FDT (*Formal Description Techniques*), do których zalicza się ESTELLE, SDL oraz LOTOS [Turner 1993]. Właśnie LOTOS, ze względu na swoje walory elegancji i ścisłości podstaw, został wybrany jako przedmiot niniejszej monografii.

Korzyści, jakie wynikają z posługiwania się nimi, oprócz precyzji i jednoznaczności, to możliwość badania sensowności modeli, to znaczy czy istnieje dla nich jakakolwiek interpretacja (czy model nie jest sprzeczny), czy mają wskazane własności. Dodatkowo ich stosowanie jest warunkiem koniecznym przy konstrukcji systemów wspomagających wytwarzanie oprogramowania. Ta ostatnia okoliczność staje się szczególnie ważna, gdyż rosnący stopień złożoności wytwarzanych systemów informatycznych wymaga coraz bardziej silnego wsparcia przy badaniu i transformacji modeli.

1.6. LOTOS w specyfikacji standardów sieci komputerowych

LOTOS powstał w ramach prac standaryzacyjnych, prowadzonych przez ISO pod koniec lat osiemdziesiątych XX wieku, w zakresie sieci komputerowych. Pierwotnym rezultatem prac ISO w tym okresie było opracowanie modelu referencyjnego ISO/OSI otwartych sieci komputerowych [ISO 7498], z czego wyłonił się problem definiowania standardów usług i protokołów sieciowych. Działające w ramach podkomitetu ISO/TC97/SC16, a później w ramach ISO/TC97/SC21, grupy robocze postawiły za cel opracowanie matematycznych podstaw umożliwiających:

- pisanie jasnych, jednoznacznych i zwartych specyfikacji standardów sieciowych,
- weryfikację poprawności specyfikacji,
- analizę własności specyfikacji,
- konstruowanie oprogramowania na podstawie specyfikacji,
- badanie zgodności implementacji ze specyfikacją.

Rezultatem prowadzonych prac było opracowanie języków formalnych ESTELLE [ISO 9074] i LOTOS [ISO 8870] oraz półformalnego języka SDL [Belina, Hogrefe 1989]. Wyniki prac spełniają tylko częściowo zarysowane wcześniej zamierzenia, gdyż ograniczają się do opracowania samych języków specyfikacji, z pominięciem pozostałych zagadnień. Należy jednak podkreślić, że były prowadzone – poza bezpośrednim nadzorem ISO – inne prace nad zrealizowaniem pozostałych, wymienionych poprzednio, zamierzeń. Język LOTOS został uznanym międzynarodowym standardem.

LOTOS jako narzędzie formalnej specyfikacji spełnia postulaty, których oczekuje się od dowolnego języka formalnych specyfikacji. Sama formalność języka oznacza, że ma on jednoznacznie zdefiniowaną składnię i semantykę, co pozwala na niesprzeczną i jednoznaczną specyfikację danego systemu. Specyfikacja jest też abstrakcyjna, co oznacza, że nie określa ani nie narzuca szczegółów implementacji systemu. Ważnym aspektem formalności jest również to, że sama specyfikacja, traktowana jako pewien abstrakcyjny obiekt, może być przedmiotem analizy metodami matematycznymi.

Język specyfikacji powinien być dostatecznie ekspresywny, aby w pełni opisywać odpowiednią klasę obiektów. W przypadku języka LOTOS chodziło o możliwość opisu usług i protokołów w poszczególnych warstwach modelu referencyjnego. Oka-

zało się, że LOTOS, spełniając te wymogi, może być narzędziem specyfikacji praktycznie dowolnych systemów rozproszonych.

Język specyfikacji, ze względu na złożoność specyfikowanych systemów, powinien dostarczać zestawu mechanizmów strukturalizacji tworzonych opisów, mających dobrze ugruntowane intuicje. Dobrze zestrukturalizowana specyfikacja podnosi czytelność i ułatwia jej pielęgnację.

Język ESTELLE bazuje na imperatywnych koncepcjach spotykanych w tradycyjnych językach programowania (punktem odniesienia był tu Pascal), język LOTOS wykorzystuje natomiast idee abstrahowania danych i zachowań, zrodzone na przełomie lat siedemdziesiątych i osiemdziesiątych ubiegłego wieku. Są nimi:

- koncepcja algebraicznej specyfikacji abstrakcyjnych typów danych,
- koncepcja specyfikacji behawioralnych, tzn. opisu zachowania systemów poprzez zbiór ciągów oddziaływań systemu ze swoim otoczeniem.

Abstrakcyjne typy danych mają w literaturze współczesnej wiele różnych modeli. Typy danych, które adaptowano na potrzeby języka LOTOS, pochodzą od języka algebraicznych specyfikacji ACT ONE, opracowanego w Uniwersytecie Technicznym w Berlinie Zachodnim pod kierunkiem H. Ehriga [Ehrig, Mahr 1985].

Opis zachowania systemów opiera się na koncepcjach zaproponowanych przez R. Milnera z Uniwersytetu w Edynburgu, zawartych w języku CCS [Milner 1980]. System jest traktowany jako pewien obiekt (zestaw równolegle działających procesów), który poprzez pewne punkty interakcji (bramki, porty) wymienia dane ze swoim otoczeniem.

Specyfikacja systemu rozproszonego w LOTOSie składa się z:

- specyfikacji abstrakcyjnych typów danych (komunikatów wymienianych pomiędzy systemem a jego otoczeniem),
- specyfikacji zachowań, czyli specyfikacji behawioralnej.

Ważną rolę w rozwijaniu formalnych technik opisu oprogramowania odegrał projekt SEDOS (*Software Environment for the Design of Open Distributed Systems*), realizowany pod koniec lat osiemdziesiątych, w ramach europejskiego programu ESPRIT [van Eijk, Vissers, Diaz 1989]. Prace przyczyniły się do ostatecznej standaryzacji języka i weryfikacji jego praktycznej przydatności, a ponadto opracowano programowe narzędzia służące między innymi do edycji i symulacji specyfikacji oraz do wspomagania tworzenia implementacji na podstawie specyfikacji.

W języku LOTOS przedstawiono pełny opis wielu powszechnie stosowanych standardów, na przykład:

- IEEE Connectionless Interneting Protocol,
- ISO Network Service,

- ISO Transport Protocol,
- ISO Transport Service,
- ISO Session Protocol,
- ISO Session Service,
- ISO Presentation Protocol,
- ISO Transaction Protocol.

Opisano także niektóre proponowane standardy, które nie znalazły jednak szerszego zastosowania: standardy usług i protokołów transferu plików i zadań.

Pod koniec lat dziewięćdziesiątych ubiegłego wieku, na bazie zdobytych doświadczeń w stosowaniu języka, rozpoczęto prace nad czasowym rozszerzeniem LOTOSa. Znalazły one wyraz w propozycji standardu [ISO/IEC FDIS 15437, 2001]. Standard nowego języka, nazywanego E-LOTOS, oprócz wprowadzenia czasu, wprowadza różne modyfikacje i usprawnienia, wśród których najistotniejsza jest zmiana specyfikacji danych. Przyjęto pragmatyczne podejście do definiowania typów danych uwzględniające sposoby definiowania stosowane we współczesnych językach programowania.

1.7. Zakres monografii

W monografii przedstawiono język LOTOS, jego zastosowania, a także jego rozszerzenia, które umożliwiają wykorzystanie języka w procesie specyfikacji i projektowania systemów czasu rzeczywistego. Omawiane są dwa rodzaje rozszerzeń – rozszerzenia czasowe, które umożliwiają specyfikowanie systemów czasu rzeczywistego z silnymi ograniczeniami czasowymi, oraz rozszerzenia wydajnościowe, które umożliwiają analizę wydajnościową systemów czasu rzeczywistego ze słabymi ograniczeniami czasowymi.

Zawartość książki jest następująca:

Dwa kolejne rozdziały – 2. oraz 3. – dotyczą języka CCS oraz jego rozszerzeń czasowych. Prezentacja języka i jego rozszerzeń jest sformalizowana. Opis semantyki jest oparty na strukturalnym podejściu operacyjnym.

W rozdziale 4. omówiono abstrakcyjne typy danych w LOTOSie.

Na informacjach z rozdziałów 2. i 4. opiera się rozdział 5., w którym opisano język LOTOS.

W rozdziale 6. omówiono najważniejsze zastosowanie języka LOTOS jako języka specyfikacji standardów sieciowych, a w rozdziale 7. – metodykę użycia języka LOTOS w procesie tworzenia i testowania specyfikacji. Przedstawiono także krótko narzędzia programowe wspomagające wykorzystanie języka.

W rozdziale 8. skupiono się na pewnej własności języka LOTOS, którą można uważać za niepożądany efekt – tak zwane ukryte blokady – wynikający z przyjętego sposobu komunikacji i synchronizacji.

W dwóch następnych rozdziałach omówiono rozszerzenia języka LOTOS: w rozdziale 9. zaprezentowano dwa rozszerzenia czasowe, a w rozdziale 10. – rozszerzenie wydajnościowe LOTOSa.

W ostatnim rozdziale 11. omówiono związki LOTOSa z innymi metodami formalnymi i półformalnymi, a zwłaszcza z językiem UML.

2. Język CCS

2.1. Wstęp

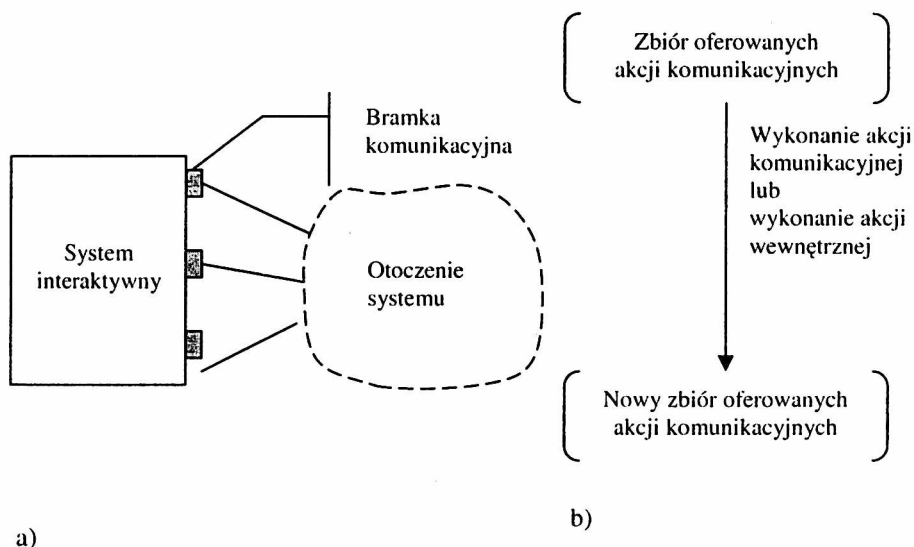
W tym rozdziale przedstawiono język CCS (*Calculus of Communicating Systems*), który był podstawą do opracowania języka LOTOS. CCS powstał na bazie prac R. Milnera, inspirowanego wcześniejszymi pracami prowadzonymi przez Hoare'a nad językiem programowania współbieżnego CSP (*Communicating Sequential Processes*) [Hoare 1978, 1985]. Przeznaczeniem CCS było formalne specyfikowanie zachowania systemów interaktywnych. W dojrzałej postaci przedstawił go Milner w książce [1980]. Wcześniejsze prace Milnera były inspiracją zwłaszcza dla jego współpracowników z Uniwersytetu w Edynburgu, na przykład: [Hennessy, Plotkin 1979, 1980], [Plotkin 1981], [Hennessy 1988], [Hennessy, Reagan 1991], były też podstawą dalszych prac Milnera [1989, 1999], jak i prac w innych ośrodkach, na przykład [Parrow 1985].

Systemy interaktywne (zob. rozdz. 1.) są szeroką kategorią systemów informatycznych, stanowiącą uogólnienie klasycznych systemów transformacyjnych. Działanie systemów transformacyjnych można opisać zależnością funkcyjną lub relacyjną pomiędzy początkowymi i końcowymi stanami systemu. Tymczasem występują systemy, w których nie chodzi o osiągnięcie pewnego stanu końcowego, lecz o to, aby bieżące działanie systemu, rozumiane jako jego współdziałanie ze swoim otoczeniem, było zgodne z określonym wzorcem. Przykładem takich systemów są systemy operacyjne, systemy rezerwacji biletów, sieci komputerowe itp. W przypadku takich systemów konieczne jest inne podejście do opisu ich działania, nazywane *podejściem behawioralnym*.

Podejście to polega na opisie działania systemu przez opis komunikacji, która może zachodzić pomiędzy systemem a jego otoczeniem. System jest widziany jako „czarna skrzynka”, która wykonuje w swoim wnętrzu pewne nieobserwowalne czynności obliczeniowe – akcje wewnętrzne i komunikuje się ze swoim otoczeniem przez obserwowalną wymianę danych na swoich *bramkach* (portach) – wykonywanie akcji komunikacyjnych (rys. 2.1a).

W podejściu behawioralnym nacisk kładzie się na komunikację. Pojedyncza komunikacja jest wynikiem *interakcji*, czyli realizacji dwóch *synchronizujących się akcji komunikacyjnych* – akcji wysłania i akcji odbioru danych – na bramce systemu. Jedna z akcji (akcja wysłania lub odbioru) jest wykonywana przez system, a druga (akcja odbioru lub wysłania) – przez jego otoczenie. Rezultatem interakcji jest wymiana pomiędzy systemem a jego otoczeniem pewnego zestawu danych.

Interakcja jest czynnością elementarną, to znaczy nie może być przerywalna ani dzielona. Ponieważ nie uwzględnia się przy tym czasu trwania interakcji, zakłada się, że interakcja wykonuje się natychmiastowo.



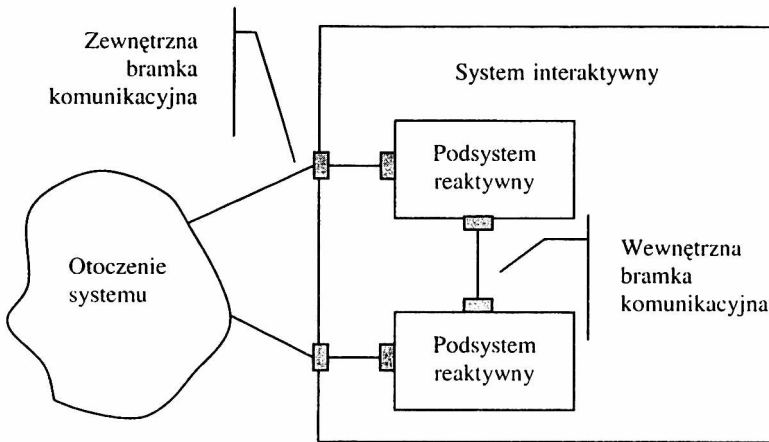
Rys. 2.1. System interaktywny jako „czarna skrzynka” (a), schemat zmiany stanów systemu (b)

Działanie systemu interaktywnego przebiega zgodnie z następującym schematem (rys. 2.1b): W danym momencie czasu system znajduje się w pewnym stanie. Z punktu widzenia otoczenia systemu stan określa zbiór oferowanych akcji komunikacyjnych. Jeżeli otoczenie skorzysta z pewnej oferowanej akcji komunikacyjnej, zostanie zrealizowana odpowiednia interakcja, w wyniku której nastąpi przesłanie pewnych danych oraz nastąpi zmiana stanu systemu. W nowym stanie system oferuje otoczeniu nowy zbiór akcji komunikacyjnych, po czym powtarzają się opisane czynności. Zmiana stanu może również nastąpić bez interakcji z otoczeniem, na skutek wykonania przez system akcji wewnętrznej. Dla otoczenia zajście takiej akcji wewnętrznej może być zauważone tylko przez zmianę zbioru oferowanych akcji komunikacyjnych.

System reaktywny może być też widziany jako „szara skrzynka” (rys. 2.2). Oznacza to, że system składa się z mniejszych „czarnych skrzynek”, które są powiązane ze sobą wewnętrznymi bramkami, przeznaczonymi do komunikacji wewnętrznej. Konceptcja „szarej skrzynki” pozwala na dekomponowanie złożonego systemu reaktywnego na elementy składowe. Elementy składowe systemu, podobnie jak cały system, nazywa się *procesami* lub *agentami*. Składowe procesy systemu można również dekomponować dalej, przedstawiając je jako złożenie innych podprocesów, a tych z kolei jako dalszych podprocesów itd.

Interakcje pomiędzy systemem a jego otoczeniem bądź pomiędzy jego procesami składowymi można przedstawiać na dwóch poziomach szczegółowości. Na poziomie ogólnym interakcja jest traktowana tylko jako *zdarzenie komunikacyjne* zachodzące na

danej bramce. Na poziomie szczegółowym interakcja jest określona jako zdarzenie komunikacyjne, któremu towarzyszy wymiana wskazanych danych.



Rys. 2.2. System reaktywny jako „szara skrzynka”

Odpowiednio do tych poziomów szczegółowości język CCS ma dwie wersje: wersję *bazową*, która przedstawia interakcje z pomijaniem danych wymienianych, i wersję *pełną*, która dane te uwzględnia. W dalszym ciągu rozdziału prezentację języka CCS ograniczono do wersji bazowej.

Specyfikacja systemu interaktywnego jest dokonywana z punktu widzenia zewnętrznego obserwatora systemu i polega na określeniu zbioru wszystkich możliwych ciągów interakcji, jakie mogą zachodzić, w czasie jego życia, pomiędzy nim a jego otoczeniem.

Specyfikacja jest wyrażana przez tak zwane *wyrażenia behawioralne* języka CCS. Stanowią one rekursywne złożenie akcji komunikacyjnych. Akcje mogą być powiązane różnymi operatorami składni, a rekursja jest wyrażana przez wywoływanie procesów (lub agentów).

Proces reprezentuje system lub jego składową, która przez własne bramki może się komunikować ze swoim otoczeniem. W danym stanie proces oferuje swemu otoczeniu pewne akcje komunikacyjne. To, która z tych akcji zostanie wykonana, zależy od synchronizacji procesu z otoczeniem. Po zsynchronizowaniu się akcji następuje ich realizacja (interakcja), po czym proces przechodzi do kolejnego stanu, w którym przedstawia swemu otoczeniu nową ofertę akcji komunikacyjnych.

Znaczeniem wyrażeń behawioralnych jest *zachowanie systemu*, określone przez zbiór ciągów interakcji, jakie zachodzą pomiędzy systemem a jego otoczeniem. W przypadku nietrywialnych systemów reaktywnych zbiór takich ciągów jest często nieskończony, a ciągi mogą mieć nieskończoną długość.

2.2. Składnia i nieformalna semantyka

Przedstawiona składnia języka odbiega w drobnych szczegółach od oryginału. Zostało to podyktowane chęcią zachowania konwencji stosowanej przez język LOTOS. Różnice te natomiast nie zmieniają semantyki języka.

Wprowadźmy następujące oznaczenia:

Niech G oznacza dowolny, co najwyżej przeliczalny, zbiór – nazywany zbiorem nazw *bramek komunikacyjnych*.

Zbiór akcji wejściowych (bez uwzględniania przesyłanych danych) będziemy oznaczać przez $G?$, akcji wyjściowych przez $G!$ oraz przez A sumę wejściowych i wyjściowych akcji komunikacyjnych, czyli $A = G? \cup G!$.

Dodatkowo wprowadzamy akcję τ , którą będziemy nazywać akcją wewnętrzną. Akcja wewnętrzna reprezentuje pewną czynność obliczeniową lub komunikacyjną, która – wykonywana wewnątrz systemu – nie jest widoczna dla jego otoczenia.

Zbiór wszystkich akcji oznaczamy $Act = A \cup \{\tau\}$. Zbiór A jest zbiorem akcji, których zajście jest obserwowalne przez otoczenie systemu, natomiast τ jest akcją, której zajście jest nieobserwowalne przez otoczenie systemu.

Akcję *wejściową* na bramce $g \in G$ będziemy oznaczać jako $g?$ ($g? \in G?$), akcję *wyjściową* na bramce $g \in G$ jako $g!$ ($g! \in G!$); ta sama bramka może być użyta do realizacji akcji wejściowej lub wyjściowej. Akcje $g?$ oraz $g!$ nazywa się akcjami *komplementarnymi*.

Jeżeli $\alpha \in A$, to jej akcję komplementarną oznacza się $\bar{\alpha}$, oczywiście $\bar{\bar{\alpha}} = \alpha$. Dla akcji wewnętrznej $\bar{\tau} = \tau$.

Bramka, na której jest realizowana akcja $\alpha \in A$, będzie oznaczana przez $gate(\alpha)$. Akcje komplementarne α oraz $\bar{\alpha}$ synchronizują się na tej samej bramce, dlatego $gate(\alpha) = gate(\bar{\alpha})$. Jeżeli $S \subseteq A$, to z definicji $gate(S) = \bigcup_{\alpha \in S} \{gate(\alpha)\}$.

Przez $Proc$ oznaczmy zbiór nazw procesów. Każdy proces ma swoją definicję, a zbiór definicji procesów Def jest określony następująco:

$$D ::= \mu P. B \mid D, \mu P. B \quad (2.1)$$

gdzie:

$D \in Def$, $P \in Proc$ jest nazwą procesu,

$B \in Beh$ jest wyrażeniem behawioralnym, stanowiącym treść procesu.

Zbiór wyrażeń behawioralnych Beh , w notacji BNF, jest określony następująco:

$$B ::= 0 \mid P \mid B_1 \setminus S \mid B_1[f] \mid (B_1) \mid \alpha; B_1 \mid B_1 \parallel B_2 \mid B_1 + B_2 \quad (2.2)$$

gdzie:

$B, B_1, B_2 \in Beh$, $P \in Proc$, $S \subseteq A$, $\alpha \in A$ oraz $f: G \rightarrow G$ jest funkcją przemianowania (zmiany nazwy) bramek komunikacyjnych. Kolejność wprowadzenia operatorów odpowiada malejącej kolejności priorytetów ich użycia;

0 jest stałą reprezentującą *proces pusty*, to znaczy taki, który nie oferuje swemu otoczeniu żadnej akcji komunikacyjnych.

P jest *instancją procesu*. Każdy proces, oprócz procesu pustego, ma dokładnie jedną definicję wyrażoną w postaci $\mu P.B$ (P jest jego nazwą, a B jest pewnym wyrażeniem behawioralnym, nazywanym treścią procesu).

Operator μ w *definicji rekursywnej* $\mu P.B$ wiąże wszystkie wolne wystąpienia nazwy procesu P w treści procesu B . Zachowanie definiowane przez wyrażenie $\mu P.B$ jest takie same jak zmodyfikowanego wyrażenia B , w którym każde wystąpienie P jest zastępowane przez $\mu P.B$.

Wolne wystąpienia nazw procesów określa funkcja pomocnicza $FProc: Beh \rightarrow 2^{Proc}$, która dla danego wyrażenia behawioralnego B wyznacza zbiór *wolnych nazw procesów*. Jest ona zdefiniowana rekursywnie w tabeli 2.1.

Tabela 2.1

B	$FProc(B)$
0	$\emptyset$
P	$\{P\}$
$\mu P.B_1$	$FProc(B_1) \setminus \{P\}$
$B_1 \setminus S$	$FProc(B_1)$
$B_1[f]$	$FProc(B_1)$
(B_1)	$FProc(B_1)$
$\alpha; B_1$	$FProc(B_1)$
$B_1 \parallel B_2$	$FProc(B_1) \cup FProc(B_2)$
$B_1 + B_2$	$FProc(B_1) \cup FProc(B_2)$

W wyrażeniu $B_1 \setminus S$, gdzie $S \subseteq A$, $\setminus$ jest operatorem przesłaniania (restrykcji). Operator ten wyklucza komunikację wyrażenia behawioralnego B_1 z jego otoczeniem na bramkach $gate(S)$. Inaczej: wyrażenie $B_1 \setminus S$ nie może się komunikować ze swoim otoczeniem na bramkach $gate(S)$.

$B_1[f]$ jest wyrażeniem, w którym nazwy bramek akcji zostają przemianowane przez funkcję $f: G \rightarrow G$; dla akcji α , w wyrażeniu B_1 , bramka $gate(\alpha) = g$ zostaje zastąpiona bramką $f(g)$. Zakładamy, że funkcja f jest bijekcją, to znaczy funkcją wzajemnie jednoznaczną. W wyrażeniu $B_1[f]$ symbol $[f]$ jest operatorem przemianowania parametryzowanym funkcją f .

Nawiasy w wyrażeniu (B_1) służą grupowaniu i strukturalizacji wewnętrznej wyrażeń. Zachowanie wyrażenia (B_1) jest dokładnie takie jak wyrażenia B_1 .

Wyrażenie $\alpha; B_1$ jest nazywane wyrażeniem prefiksowania akcją α wyrażenia B_1 albo krótko – wyrażeniem prefiksującym. Symbol $;$ jest operatorem prefiksowania. Wyrażenie $\alpha; B_1$ oferuje swemu otoczeniu tylko jedną akcję komunikacyjną α . Jeżeli akcja ta zostanie zaakceptowana przez otoczenie, to po jej wykonaniu kolejne oferty komunikacyjne są określone przez wyrażenie B_1 .

Wyrażenie złożenia równoległego $B_1 \parallel B_2$, gdzie $\parallel$ jest operatorem złożenia równoległego, zachowuje się tak, jak zachowują się jednocześnie wyrażenia B_1 oraz B_2 , to znaczy oferuje swemu otoczeniu akcje komunikacyjne oferowane przez wyrażenie B_1 oraz przez wyrażenie B_2 . Oba wyrażenia, poza tym że oferują akcje komunikacyjne swemu otoczeniu, dodatkowo mogą się wzajemnie komunikować. Komunikacja taka wymaga synchronizacji dwóch komplementarnych akcji na wspólnej bramce.

Wyrażenie wyboru $B_1 + B_2$, gdzie $+$ jest operatorem wyboru, zachowuje się jak B_1 albo B_2 . Wybór zachowania zależy od otoczenia wyrażenia $B_1 + B_2$, które wskazuje na B_1 albo B_2 , wybierając do realizacji albo jedną z akcji oferowanych przez B_1 , albo jedną z akcji oferowanych przez B_2 .

Definicja 2.1

Specyfikacja zachowania w języku CCS jest zdefiniowana jako para

$$Spec = \langle B_0, Def_P \rangle \quad (2.3)$$

gdzie: $B_0 \in Beh$ jest początkowym wyrażeniem behawioralnym, a $Def_P \in Def$ jest skończoną listą definicji pewnych procesów $P_1, \dots, P_n$.

Definicja procesu o nazwie P_i ($i = 1, \dots, n$) ma postać $\mu P_i. B_i$, gdzie wyrażenie behawioralne $B_i \in Beh$ jest treścią procesu. Każdy proces, którego instancja występuje w wyrażeniach behawioralnych $B_0, \dots, B_n$ ma swoją definicję w zbiorze Def . Instancje procesów o tej samej nazwie mają jedną wspólną definicję.

Wprowadzamy funkcję pomocniczą $FAct : Beh \rightarrow Act$, która dla danego wyrażenia behawioralnego B wyznacza zbiór akcji obserwowalnych. Funkcja jest zdefiniowana rekursywnie w tabeli 2.2.

Funkcja umożliwia zdefiniowanie dwóch rodzajów specyfikacji:

Definicja 2.2

Specyfikację $Spec = \langle B_0, Def \rangle$, dla której $FAct(B_0) = \emptyset$ nazywamy specyfikacją komunikacyjnie zamkniętą, a komunikacyjnie otwartą w przypadku przeciwnym.

System, którego specyfikacja jest komunikacyjnie otwarta, jest systemem, którego funkcjonowanie jest uwarunkowane działaniem jego otoczenia, działanie (obliczenie) systemu, którego specyfikacja jest komunikacyjnie zamknięta, jest natomiast niezależne od jego otoczenia (system nie komunikuje się ze swoim otoczeniem).

Tabela 2.2

B	$FAct(B)$
0	$\emptyset$
P	$FAct(B_1)$ gdzie $\mu P.B_1$
$\mu P.B_1$	$FAct(B_1)$
$B_1 \setminus S$	$FAct(B_1) \setminus S^*$
$B_1[f]$	$FAct(B_1)[f]$
(B_1)	$FAct(B_1)$
$\alpha; B_1$	$\{\alpha\} \cup FAct(B_1)$
$B_1 \parallel B_2$	$FAct(B_1) \cup FAct(B_2)$
$B_1 + B_2$	$FAct(B_1) \cup FAct(B_2)$

^{*)} Symbol $\setminus$ występuje w prawej kolumnie w roli odejmowania mnogościowego, w lewej kolumnie natomiast występuje w roli operatora restrykcji.

Czasem jest dogodne operowanie wyrażeniami behawioralnymi w pewnych postaciach kanonicznych.

Wyrażenie behawioralne B jest w *sekwencyjnej postaci normalnej*, gdy przyjmuje postać

$$B \equiv \sum_{i \in \{1, 2, \dots, n\}} B_i \equiv B_1 + \dots + B_n \quad (2.4)$$

gdzie:

B_i to składowe sekwencyjne wyrażenia B ,

symbol $\equiv$ oznacza równoważność (identyczność) tekstową.

Wyrażenie behawioralne B jest w *równoległej postaci normalnej*, gdy przyjmuje postać

$$B \equiv \parallel_{i \in \{1, 2, \dots, n\}} B_i \setminus S \equiv (B_1 \parallel \dots \parallel B_n) \setminus S \quad (2.5)$$

gdzie B_i to składowe równoległe wyrażenia B .

2.3. Semantyka operacyjna

Semantyka operacyjna specyfikacji $Spec = \langle B_0, Def \rangle$ jest definiowana w sposób strukturalny na podstawie definicji składniowych wyrażeń behawioralnych. Definicja opiera się na pojęciu etykietowanego systemu przejść postaci

$$TS(Spec) = \langle Beh, Act, Tr, B_0 \rangle \quad (2.6)$$

gdzie:

Beh jest zbiorem wyrażeń behawioralnych;

Act jest zbiorem akcji;

$Tr = \{ \xrightarrow{\alpha} \subseteq Beh \times Beh \mid \alpha \in Act \}$ jest rodziną relacji przejść pomiędzy wyrażeniami behawioralnymi.

Przejście postaci

$$B_1 \xrightarrow{\alpha} B_2$$

oznacza, że proces, którego stan jest reprezentowany wyrażeniem behawioralnym B_1 , w wyniku realizacji akcji $\alpha \in Act$, zmienia swój stan na stan reprezentowany wyrażeniem B_2 . Jeżeli α jest akcją obserwowalną, to jej wykonanie i stowarzyszone z nią dane zostają wymienione pomiędzy systemem a jego otoczeniem, jeżeli natomiast α jest akcją wewnętrzną, to przejściu od B_1 do B_2 towarzyszy komunikacja wewnętrzna, która nie jest widoczna dla otoczenia procesu; B_0 jest początkowym wyrażeniem behawioralnym.

Rodzina relacji przejść jest definiowana na podstawie rekursji strukturalnej, to znaczy każdemu operatorowi wyrażenia behawioralnego odpowiada pewien aksjomat lub reguła systemu aksjomatycznego.

Aksjomaty mają postać ustalonych przejść (schematów przejść) pomiędzy wyrażeniami behawioralnymi. Przesłankami i wnioskami reguł są wybrane przejścia. Dodatkowo z regułami wiąże się warunek ich stosowania.

Reguły zapisuje się w postaci

$$\frac{B'_1 \xrightarrow{\alpha_1} B'_1 \quad B'_2 \xrightarrow{\alpha_2} B''_2}{B_1 \xrightarrow{\alpha} B_2} \quad \text{war}$$

gdzie:

$B'_i \xrightarrow{\alpha} B''_i$ ($i = 1, 2$) są przejściami-przesłankami,

$B_1 \xrightarrow{\alpha} B_2$ jest przejściem-konkluzją,

war jest warunkiem stosowania reguły.

Aksjomat jest szczególną postacią reguły, gdy zbiór przesłanek jest pusty, dlatego aksjomaty zapisuje się w postaci przejścia-konkluzji.

Proces pusty – brak aksjomatów i reguł.

Instancja procesu

$$\frac{\mu P.B \xrightarrow{\alpha} B'}{P \xrightarrow{\alpha} B'} \quad (R-ins)$$

Rekursja

$$\frac{B[P ::= \mu P.B] \xrightarrow{\alpha} B'}{\mu P.B \xrightarrow{\alpha} B'} \quad (R-rec)$$

gdzie $[P ::= B']$ jest pomocniczym operatorem (jednokrotnego) zastąpienia tekstowego. Notacja postaci $B[P ::= B']$ oznacza modyfikację wyrażenia B , polegającą na

tekstowym zastąpieniu każdego wolnego wystąpienia nazwy procesu P w wyrażeniu B wyrażeniem B' .

Reguła ($R-ins$) stwierdza, że wywołanie (instancja) procesu ma zachowanie takie same jak jego treść. Reguła ($R-rec$) stwierdza natomiast, że zachowanie procesu rekursywnego P jest określone przez treść tego procesu, zmodyfikowaną w taki sposób, że każde wolne wystąpienie jego nazwy jest zastąpione tekstowo przez treść tego samego procesu P .

Restrykcja

$$\frac{B \xrightarrow{\alpha} B'}{B \setminus S \xrightarrow{\alpha} B' \setminus S} \quad gate(\alpha) \notin S \quad (R-res_1)$$

$$\frac{B \xrightarrow{\alpha} B'}{B \setminus S \xrightarrow{\tau} B' \setminus S} \quad gate(\alpha) \in S \quad (R-res_2)$$

Pierwsza reguła restrykcji ($R-res_1$) dotyczy akcji nienależącej do zbioru przesłanianych akcji – wykonanie takiej akcji jest obserwowalne. Druga reguła ($R-res_2$) dotyczy akcji ze zbioru akcji przesłanianych – jej wykonanie jest nieobserwowalne, jest traktowane jako wykonanie akcji wewnętrznej.

Przemianowanie

$$\frac{B \xrightarrow{\alpha} B'}{B[f] \xrightarrow{f(\alpha)} B'[f]} \quad \text{gdzie } f(\alpha) = \begin{cases} f(g)! & \text{dla } \alpha \equiv g! \\ f(g)? & \text{dla } \alpha \equiv g? \end{cases} \quad (R-ren)$$

Reguła stwierdza, że zachowanie wyrażenia behawioralnego $B[f]$ z bramkami przemianowanymi funkcją f jest takie jak zachowanie wyrażenia B zmodyfikowanego w taki sposób, że każda akcja na danej bramce g zostaje zastąpiona akcją na bramce $f(g)$.

Nawiasy

$$\frac{B \xrightarrow{\alpha} B'}{(B) \xrightarrow{\alpha} B'} \quad \text{gdzie } B \equiv B_1 + B_2 \quad (R-p_1)$$

$$\frac{B \xrightarrow{\alpha} B'}{(B) \xrightarrow{\alpha} (B')} \quad \text{gdzie } B \equiv B_1 \parallel B_2 \quad (R-p_2)$$

Z użyciem nawiasów wiążą się dwie reguły, gdyż ich zastosowanie polega na grupowaniu wyrażeń wyboru lub wyrażeń złożenia równoległego. W przypadku pierwszym – reguła ($R-p_1$) – wyrażenie $(B_1 + B_2)$, wykonując pierwszą akcję, dokonuje wyboru jednego z wyrażeń składowych, a zatem przechodzi do wyrażenia, w którym nie ma potrzeby dalszego użycia nawiasów, gdyż występuje tylko jedno wyrażenie składowe. Wykonanie natomiast dowolnej akcji przez wyrażenie złożenia równoległego $(B_1 \parallel B_2)$

pozostawia oba wyrażenia składowe, które dalej powinny być ujęte w nawiasy – reguła ($R-p_2$).

Prefiksowanie akcją

$$\alpha; B \xrightarrow{\alpha} B \quad (A-pre)$$

Zachowanie wyrażenia prefiksowanego akcją α ; B jest opisane aksjوماتem, który stwierdza, że po wykonaniu akcji prefiksującej α dalsze zachowanie wyrażenia jest określone przez zachowanie wyrażenia B .

Złożenie równoległe

$$\frac{B_1 \xrightarrow{\alpha} B'_1}{B_1 \parallel B_2 \xrightarrow{\alpha} B'_1 \parallel B_2} \quad (R-par_1)$$

$$\frac{B_2 \xrightarrow{\alpha} B'_2}{B_1 \parallel B_2 \xrightarrow{\alpha} B_1 \parallel B'_2} \quad (R-par_2)$$

$$\frac{B_1 \xrightarrow{\alpha} B'_1 \quad B_2 \xrightarrow{\bar{\alpha}} B'_2}{B_1 \parallel B_2 \xrightarrow{\tau} B'_1 \parallel B'_2} \quad (R-par_3)$$

Dwie pierwsze reguły ($R-par_1$) i ($R-par_2$) odnoszą się do przypadku, gdy jedna ze składowych złożenia równoległego wykonuje akcję, druga składowa natomiast nie wykonuje żadnej akcji. Oznacza to, że jeśli jest wykonywana akcja komunikacyjna, to partnerem komunikacji jest otoczenie wyrażenia $B_1 \parallel B_2$. Wynikiem realizacji takiej akcji jest modyfikacja składowej, która uczestniczy w akcji, i pozostawienie bez zmiany drugiej składowej. Reguła ($R-par_3$) dotyczy przypadku, gdy obie składowe uczestniczą w realizacji pary komplementarnych akcji komunikacyjnych. Wówczas rezultatem realizacji jest odpowiednia modyfikacja obu składowych.

Wybór

$$\frac{B_1 \xrightarrow{\alpha} B'_1}{B_1 + B_2 \xrightarrow{\alpha} B'_1} \quad (R-ch_1)$$

$$\frac{B_2 \xrightarrow{\alpha} B'_2}{B_1 + B_2 \xrightarrow{\alpha} B'_2} \quad (R-ch_2)$$

Reguły ($R-ch_1$) i ($R-ch_2$) odzwierciedlają dwa przypadki odnoszące się do wyboru jednej akcji, należącej albo do B_1 , albo do B_2 . Wybór takiej akcji determinuje, że dalsze zachowanie przebiega zgodnie z wyrażeniem, do którego należy wybrana akcja. Wybór zależy całkowicie od otoczenia wyrażenia $B_1 + B_2$, gdy obie składowe oferują wyłącznie akcje komunikacyjne. Gdy natomiast obie składowe oferują tylko akcje

wewnętrzne, na przykład jak dla wyrażenia $\tau; B_1 + \tau; B_2$, wybór jednej z tych akcji wewnętrznych jest niedeterministyczny i nie zależy od otoczenia.

Podane aksjomaty i reguły umożliwiają zdefiniowanie ciągów akcji, które generuje dana specyfikacja *Spec*.

W celu zdefiniowania tych ciągów wprowadźmy oznaczenia:

Z^* – zbiór wszystkich ciągów skończonych nad zbiorem Z ; ciąg pusty będzie oznaczany przez ε ,

Z^∞ – zbiór wszystkich ciągów nieskończonych nad zbiorem Z .

Jeżeli $s_1 \in Z^*$ oraz $s_2 \in Z^* \cup Z^\infty$, to przez $s_1 \wedge s_2$ będziemy oznaczać konkatenację tych ciągów.

Ciąg tranzycji (obliczenie) generowany przez system przejść $TS(Spec)$ ma postać

$$B_0 \xrightarrow{\alpha_1} B_1 \xrightarrow{\alpha_2} B_2 \dots \xrightarrow{\alpha_n} B_n \quad (2.7a)$$

lub

$$B_0 \xrightarrow{\alpha_1} B_1 \xrightarrow{\alpha_2} B_2 \dots \xrightarrow{\alpha_n} B_n \xrightarrow{\alpha_{n+1}} \dots \quad (2.7b)$$

gdzie:

B_0 jest początkowym wyrażeniem specyfikacji *Spec*,

B_n w przypadku obliczenia skończonego (2.7a) jest wyrażeniem końcowym, to znaczy takim, dla którego nie istnieją przejścia do innych wyrażen.

Dla dwóch wyrażen behawioralnych $B, B' \in Beh$ oraz dla skończonego ciągu akcji $s \in Act^*$, gdzie $s \equiv \alpha_1 \dots \alpha_n$, zapis

$$B \xrightarrow{s} B' \quad (2.8)$$

oznacza, że wyrażenie behawioralne B' jest osiągalne z wyrażenia B przez ciąg akcji s , czyli

$$\exists B_1, \dots, B_{n-1} \bullet B \xrightarrow{\alpha_1} B_1 \dots \xrightarrow{\alpha_{n-1}} B_{n-1} \xrightarrow{\alpha_n} B' \quad (2.9)$$

Będziemy mówić, że z wyrażenia B ciąg akcji s prowadzi do pewnego wyrażenia osiągalnego, co zapisuje się

$$B \xrightarrow{s} \quad (2.10)$$

gdy $\exists B' \bullet B \xrightarrow{s} B'$.

Zbiór $\{B' \mid \exists B' \bullet B \xrightarrow{s} B'\}$ jest zbiorem wyrażen behawioralnych osiągalnych z wyrażenia B .

Dla $s \in Act^\infty$ zbiór wszystkich jego początkowych podciągów oznaczmy przez

$$pref(s) = \{s' \in Act^* \mid \exists s'' \in Act^\infty \bullet s \equiv s' \wedge s''\} \quad (2.11)$$

Definicja 2.3

Specyfikacja $Spec = \langle B_0, Decl \rangle$ generuje zbiór ciągów akcji

$$Seq(B_0) = SeqFin(B_0) \cup SeqInf(B_0) \quad (2.12)$$

gdzie:

$$SeqFin(B_0) = \{s \in Act^* \mid B_0 \xrightarrow{s} \} \quad (2.13)$$

$$SeqInf(B_0) = \{s \in Act^\infty \mid \forall s' \in pref(s) \bullet B_0 \xrightarrow{s'} \} \quad (2.14)$$

Zbiór ciągów $Seq(B_0)$ może być uważany za pełną semantykę wyrażenia behawioralnego B_0 , gdyż bierze pod uwagę wszystkie ciągi, w których występują zarówno akcje obserwowalne, jak i nieobserwowalne. Inny rodzaj semantyki bierze pod uwagę tylko ciągi akcji obserwowalnych. W celu przedstawienia tej semantyki wprowadzimy dodatkowe pojęcia.

Najpierw wprowadzimy *relację przejść obserwowalnych* pomiędzy wyrażeniami behawioralnymi dla $s \equiv \alpha_1 \dots \alpha_n \in A^*$, oznaczaną

$$B = s \Rightarrow B' \quad (2.15)$$

i zdefiniowaną następująco:

$B = \varepsilon \Rightarrow B'$ oznacza, że istnieje, być może pusty, ciąg przejść

$$B_0 \xrightarrow{\tau} B_1 \xrightarrow{\tau} B_2 \dots \xrightarrow{\tau} B_n \text{ dla } n \in Nat \quad (2.16)$$

gdzie: $B \equiv B_0$ oraz $B' \equiv B_n$.

$B = \alpha \Rightarrow B'$, dla $\alpha \in A$, oznacza, że istnieją B_1 oraz B_2 , takie że

$$B = \varepsilon \Rightarrow B_1 \wedge B_1 \xrightarrow{\alpha} B_2 \wedge B_2 = \varepsilon \Rightarrow B' \quad (2.17)$$

Dalej, dla $s \equiv \alpha_1 \dots \alpha_n \in A^*$

$$B = s \Rightarrow B' \quad (2.18)$$

oznacza, że

$$\exists B_1 \dots B_n \bullet B = \alpha_1 \Rightarrow B_1 \wedge \dots \wedge B_n = \alpha_n \Rightarrow B' \quad (2.19)$$

oraz

$$B = s \Rightarrow \quad (2.20)$$

oznacza, że

$$\exists B' \bullet B = s \Rightarrow B' \quad (2.21)$$

Niech $s \in Act^*$ będzie postaci $s \equiv \tau^{k_0} \alpha_1 \tau^{k_1} \alpha_2 \tau^{k_2} \dots \alpha_n \tau^{k_n}$, gdzie

$$\tau^{k_i} \equiv \underbrace{\tau \tau \dots \tau}_{k_i \text{ razy}} \text{ dla } k_i \in Nat \text{ oraz } i = 0, 1, \dots, n$$

są podciągami złożonymi wyłącznie z akcji wewnętrznych. Przez s^{obs} oznaczmy podciąg ciągu s złożony z akcji obserwowalnych, czyli

$$s^{obs} \equiv \alpha_1 \dots \alpha_n.$$

W szczególności $\alpha^{obs} = \alpha$, gdy $\alpha \neq \tau$, oraz $\alpha^{obs} = \varepsilon$ w przypadku przeciwnym.

Definicja 2.4

Specyfikacja $Spec = \langle B_0, Decl \rangle$ generuje zbiór obserwowalnych ciągów akcji

$$SeqObs(B_0) = SeqObsFin(B_0) \cup SeqObsInf(B_0) \quad (2.22)$$

gdzie:

$$SeqObsFin(B_0) = \{s \in A^* \mid B_0 = s \Rightarrow\} \quad (2.23)$$

$$SeqObsInf(B_0) = \{s \in Act^\infty \mid \forall s' \in pref(s) \bullet B_0 = s' \Rightarrow\} \quad (2.24)$$

W dalszej części będziemy rozpatrywać wyrażenia dozorowane behawioralne, określone przez definicję 2.5.

Definicja 2.5

Wyrażenie behawioralne B nazywa się wyrażeniem *dozorowanym*, jeśli każde wystąpienie instancji dowolnego procesu, z wyjątkiem procesu pustego, w wyrażeniu B , jest prefiksowane akcją.

Pojęcie wyrażenia dozorowanego eliminuje pewne niewygodne, „patologiczne” wyrażenia, których interpretacja jest co najmniej kłopotliwa. Przykładami takich wyrażeń są instancje procesów opartych na definicjach:

$$\mu P.P$$

$$\mu Q.(a; B + Q)$$

$$\mu P.Q \quad \mu Q.P$$

Czasem również rozważania ogranicza się do wyrażeń zbudowanych regularnie.

Definicja 2.6

Dozorowane wyrażenie behawioralne B nazywa się wyrażeniem *regularnie zbudowanym*, jeśli dla dowolnego procesu P występującego w B o definicji $\mu P.B_P$ jego treść B_P nie zawiera rekursywnego wystąpienia operatora $\parallel$.

Pojęcie wyrażenia regularnie zbudowanego eliminuje natomiast sytuację, gdy kolejne przejścia od danego wyrażenia prowadzą do nieograniczonej rozbudowy wyrażenia. Rozpatrzmy na przykład proces

$$\mu P.(a; P \parallel b; 0)$$

Wywołaniu tego procesu może towarzyszyć ciąg tranzycji

$$(a; P \parallel b; 0) \xrightarrow{a} ((a; P \parallel b; 0) \parallel b; 0) \xrightarrow{a} (((a; P \parallel b; 0) \parallel b; 0) \parallel b; 0) \xrightarrow{a} \dots$$

w którym następuje nieograniczone rozwijanie wyrażenia.

2.4. Relacje równoważności

Tekstowe porównanie wyrażeń behawioralnych nie pozwala bezpośrednio na stwierdzenie, czy stanowią one różną reprezentację takich samych zachowań. Podstawą do stwierdzania równoważności zachowań dwóch wyrażeń jest następujący punkt widzenia:

jeżeli jedno z wyrażeń jest zdolne do wykonania pewnej akcji α , a następnie do wykonywania akcji zgodnie z nowym wyrażeniem B , to drugie – równoważne mu wyrażenie – musi być również zdolne do wykonania tej samej akcji α , a następnie do zachowania się zgodnie z wyrażeniem, które jest równoważne wyrażeniu B .

Tego rodzaju równoważność zachowań jest określana mianem równoważności bisymulacyjnej. Podane dalej definicje uściślają to pojęcie, a ponadto rozróżniają pojęcie silnej i słabej bisymulacji.

Definicja 2.7

Relację binarną $R \subseteq \text{Beh}^2$ nazywa się *bisymulacją*, jeśli dla każdej pary wyrażeń behawioralnych $\langle B_1, B_2 \rangle \in R$ i dowolnych akcji $\alpha \in \text{Act}$ spełnione są następujące warunki:

- a) jeżeli $B_1 \xrightarrow{\alpha} B'_1$, to $\exists B'_2 \bullet B_2 \xrightarrow{\alpha} B'_2 \wedge \langle B'_1, B'_2 \rangle \in R$
- b) jeżeli $B_2 \xrightarrow{\alpha} B'_2$, to $\exists B'_1 \bullet B_1 \xrightarrow{\alpha} B'_1 \wedge \langle B'_1, B'_2 \rangle \in R$

Definicja 2.8

Dwa wyrażenia B_1 oraz B_2 są *bisymulacyjnie silnie* równoważne, co oznacza się $B_1 \sim B_2$, gdy istnieje bisymulacja R taka, że $\langle B_1, B_2 \rangle \in R$.

Z definicji wynika, że relacja $\sim$ jest relacją silnej bisymulacji, gdy

$$\sim =_{\text{def}} \bigcup \{ R \mid R \text{ jest relacją bisymulacji} \} \quad (2.25)$$

Relacja silnej bisymulacji traktuje akcje komunikacyjne w taki sam sposób jak akcje wewnętrzne i dlatego w przypadku, gdy akcje wewnętrzne są obserwowalne, nie za-

chodzą własności, których można by w tym przypadku oczekiwać, na przykład relacja ta nie zachodzi pomiędzy wyrażeniami $\alpha; \tau; B$ oraz $\alpha; B$. Uzasadnia to wprowadzenie relacji słabej bisymulacji, która jest określona następująco:

Definicja 2.9

Relację binarną $S \subseteq Beh^2$ nazywa się *słabą bisymulacją*, jeśli dla każdej pary wyrażen behawioralnych $\langle B_1, B_2 \rangle$ i dowolnych akcji $\alpha \in Act$ są spełnione następujące warunki:

- a) jeżeli $B_1 \xrightarrow{\alpha} B'_1$, to $\exists B'_2 \bullet B_2 = \alpha^{obs} \Rightarrow B'_2 \wedge \langle B'_1, B'_2 \rangle \in S$
- b) jeżeli $B_2 \xrightarrow{\alpha} B'_2$, to $\exists B'_1 \bullet B_1 = \alpha^{obs} \Rightarrow B'_1 \wedge \langle B'_1, B'_2 \rangle \in S$

Definicja 2.10

Dwa wyrażenia B_1 oraz B_2 są *obserwacyjnie równoważne*, co oznacza się $B_1 \approx B_2$, gdy istnieje słaba bisymulacja S taka, że $\langle B_1, B_2 \rangle \in S$.

Oznacza to, że relacja $\approx$ jest relacją obserwacyjnej równoważności, gdy

$$\approx =_{\text{def}} \bigcup \{ R \mid R \text{ jest relacją słabej bisymulacji} \} \quad (2.26)$$

Należy zauważyć, że równoważność obserwacyjna nie sprowadza się do zwykłego pomijania akcji wewnętrznych w porównywanych wyrażeniach behawioralnych. Na przykład dwa wyrażenia $\alpha; B$ oraz $\tau; \alpha; B$ są równoważne obserwacyjnie, ale dwa wyrażenia $\alpha_1; B_1 + \tau; \alpha_2; B_2$ oraz $\alpha_1; B_1 + \alpha_2; B_2$ nie są równoważne obserwacyjnie.

Łatwo sprawdzić, że zarówno bisymulacja, jak i słaba bisymulacja są relacjami równoważności na zbiorze wyrażen behawioralnych, czyli że są relacjami zwrotnymi, symetrycznymi i przechodnimi. Dodatkowo relacja silnej bisymulacji jest kongruencją, relacja obserwacyjnej bisymulacji kongruencją natomiast nie jest.

Relacja kongruencji nad dowolnym zbiorem termów $Term$ jest zdefiniowana następująco:

Definicja 2.11

Niech $t'_1 \in Term$ będzie podtermem termu $t_1 \in Term$, co dalej będziemy zapisywać $subt(t'_1, t_1)$, oraz niech t_2 będzie termem, który powstaje z termu t_1 przez tekstowe zastąpienie podtermu t'_1 termem t'_2 , czyli $t_2 \equiv t_1[t'_1 := t'_2]$.

Relacja równoważności R nad dowolnym zbiorem termów $Term$ jest *kongruencją*, jeżeli z faktu, że $\langle t'_1, t'_2 \rangle \in R$ wynika, że $\langle t_1, t_2 \rangle \in R$.

Uwaga: Operator zastąpienia tekstowego $[t_1 := t_2]$ należy odróżniać od poprzednio wprowadzonego operatora zastąpienia tekstowego $[P ::= B]$.

Jeżeli jako zbiór termów rozpatrzmy zbiór wyrażeń behawioralnych Beh oraz wyrażenie behawioralne B_1 zmodyfikujemy w taki sposób, że pewne jego podwyrażenie B'_1 zastąpimy podwyrażeniem B'_2 , to otrzymane w ten sposób nowe wyrażenie B_2 pozostaje w relacji R z wyrażeniem B_1 , jeśli tylko podwyrażenia B'_1 oraz B'_2 są ze sobą w relacji R .

Definicja 2.12

Dwa wyrażenia behawioralne B_1 oraz B_2 są *kongruencyjnie obserwacyjnie* równoważne, co zapisujemy $B_1 \approx^c B_2$, jeśli są spełnione następujące warunki:

- a) $B_1 \approx B_2$,
- b) dla dowolnego wyrażenia B takiego, że $subt(B_1, B)$, jeżeli $B_1 \approx B_2$, to

$$B \approx B[B_1 := B_2]$$

Kongruencyjna równoważność obserwacyjna pozwala na stwierdzenie, że dwa wyrażenia behawioralne, kongruencyjnie równoważne obserwacyjnie można wzajemnie zastępować w dowolnym wyrażeniu bez obawy zmiany obserwowanego zachowania całego wyrażenia.

Bezpośrednio z definicji relacji wynika, że

$$\sim \subseteq \approx^c \subseteq \approx \quad (2.27)$$

2.5. Prawa równościowe

Jednym ze sposobów badania równoważności wyrażeń behawioralnych jest porównanie grafów wyrażeń osiągalnych dla tych wyrażeń – jest to podejście algorytmiczne. Algorytm badania silnej równoważności bisymulacyjnej jest efektywny obliczeniowo – ma złożoność wielomianową. Podobnie, przy pewnych ograniczeniach (zbiory $Seq(B_1)$ i $Seq(B_2)$ odpowiadające porównywanym wyrażeniom muszą być skończone), efektywny jest algorytm badania słabej równości bisymulacyjnej [Milner 1989], a przegląd wcześniejszych prac na ten temat zawiera między innymi publikacja [Bolognesi, Smolka 1987].

Inny sposób badania równoważności polega na tekstowych przekształceniach wyrażeń opartych na regułach równościowych – jest to podejście algebraiczne [Milner 1989].

System dowodowy składa się ze zbioru aksjomatów i reguł wnioskowania. Aksjomaty przyjmują postać równości

$$B_1 = B_2,$$

gdzie symbol $=$ może oznaczać jedną z równoważności: $\sim$, $\approx^c$, $\approx$.

Zbiór reguł, ze względu na własności równoważności, obejmuje reguły zwrotności, symetrii i przechodniości:

$$B_1 = B_1 \quad (R\text{-refl})$$

$$\frac{B_1 = B_2}{B_2 = B_1} \quad (R\text{-sym})$$

$$\frac{B_1 = B_2 \quad B_2 = B_3}{B_1 = B_3} \quad (R\text{-trans})$$

Poniżej przedstawiono zbiór aksjomatów dla badania kongruencyjnej równoważności obserwacyjnej $\approx^c$.

Aksjomaty dla operatora wyboru:

$$B_1 + B_2 \approx^c B_2 + B_1 \quad (A_1\text{-ch})$$

$$B_1 + (B_2 + B_3) \approx^c (B_1 + B_2) + B_3 \quad (A_2\text{-ch})$$

$$B + B \approx^c B \quad (A_3\text{-ch})$$

$$B + \mathbf{0} \approx^c B \quad (A_4\text{-ch})$$

Aksjomaty dla operatora prefiksowania akcją:

$$\alpha; \tau; B \approx^c \alpha; B \quad (A_1\text{-pre})$$

$$\tau; B + B \approx^c \tau; B \quad (A_2\text{-pre})$$

$$\alpha; (B_1 + \tau; B_2) + \alpha; B_2 \approx^c \alpha; (B_1 + \tau; B_2) \quad (A_3\text{-pre})$$

Aksjomaty dla operatora restrykcji:

$$B \setminus S \approx^c B \text{ jeżeli } gate(S) \cap gate(FAct(B)) = \emptyset \quad (A_1\text{-res})$$

$$B \setminus S_1 \setminus S_2 \approx^c B \setminus (S_1 \cup S_2) \quad (A_2\text{-res})$$

$$(B_1 \parallel B_2) \setminus S \approx^c B_1 \setminus S \parallel B_2 \setminus S$$

$$\text{jeżeli } gate(S) \cap gate(FAct(B_1)) \cap gate(FAct(B_2)) = \emptyset \quad (A_3\text{-res})$$

$$B[f] \setminus S \approx^c B \setminus f^{-1}(S)[f] \quad (A_4\text{-res})$$

Aksjomaty dla operatora przemianowania:

$$B[id] \approx^c B \quad (A_1\text{-ren})$$

$$B[f] \approx^c B[f'] \text{ jeżeli } f|_{gate(FAct(B))} = f'|_{gate(FAct(B))} \quad (A_2\text{-ren})$$

$$B[f][f'] \approx^c B[f' \circ f] \text{ gdzie } \circ \text{ jest symbolem złożenia funkcji} \quad (A_3\text{-ren})$$

$$(B_1 \parallel B_2)[f] \approx^c B_1[f] \parallel B_2[f] \text{ jeżeli funkcja } f \text{ jest bijekcją} \quad (A\text{-ren})$$

Aksjomaty dla operatora kompozycji równoległej:

$$B_1 \parallel B_2 \approx^c B_2 \parallel B_1 \quad (A_1\text{-par})$$

$$B_1 \parallel (B_2 \parallel B_3) \approx^c (B_1 \parallel B_2) \parallel B_3 \quad (A_2\text{-par})$$

$$B \parallel \mathbf{0} \approx^c B \quad (A_3\text{-par})$$

Zbiór reguł specyficznych dla kongruencyjnej równoważności obserwacyjnej obejmuje reguły związane z rekursją

$$\frac{\mu P.B}{P \approx^c B} \quad (R_1\text{-rec})$$

Reguła $R_1\text{-rec}$ stwierdza, że wywołanie (instancja) procesu jest równoważna treści procesu.

$$\frac{P \approx^c B[P ::= X] \quad Q \approx^c B[Q ::= X]}{P \approx^c Q} \quad \text{gdzie } X \text{ jest wyrażeniem regularnie zbudowanym} \quad (R_2\text{-rec})$$

Reguła $(R_2\text{-rec})$ jest szczególnym przypadkiem reguły podanej niżej. Reguła ta odnosi się do zbioru wzajemnie rekursywnie zdefiniowanych procesów. Wymaga ona wprowadzenia pomocniczych oznaczeń. Niech

$$\tilde{P} \equiv P_1, \dots, P_n$$

$$\tilde{Q} \equiv Q_1, \dots, Q_n$$

$$\tilde{X} \equiv X_1, \dots, X_n$$

wówczas

$$\frac{\mu \tilde{P}.\tilde{B}[\tilde{P} ::= \tilde{X}] \quad \mu \tilde{Q}.\tilde{B}[\tilde{Q} ::= \tilde{X}]}{\tilde{P} \approx^c \tilde{Q}} \quad \text{gdzie } \tilde{X} \text{ jest wyrażeniem regularnie zbudowanym} \quad (R_3\text{-rec})$$

2.6. Twierdzenie o ekspansji

Badanie równoważności wyrażeń behawioralnych często prowadzi się po uprzednim sprowadzeniu ich do postaci normalnych. Przedstawione niżej prawo o ekspansji pokazuje, jak można wyeliminować z wyrażenia operatory złożenia równoległego.

Prawo (twierdzenie) o ekspansji dotyczy wyrażeń behawioralnych w normalnej postaci równoległej

$$B \equiv \parallel_{i \in \{1, 2, \dots, n\}} B_i \setminus S \equiv (B_1 \parallel \dots \parallel B_n) \setminus S$$

Możliwe są dwa rodzaje tranzycji wyrażenia B do wyrażenia następnego.

Pierwszy rodzaj polega na tranzycji tylko jednego z wyrażeń składowych. Możliwe jest to wówczas, gdy akcja jest wynikiem komunikacji wyrażenia B na bramce spoza zbioru S , czyli

$$\sum \left\{ \alpha; (B_1 \parallel \dots \parallel B'_i \parallel \dots \parallel B_n) \setminus S \mid B_i \xrightarrow{\alpha} B'_i \wedge gate(\alpha) \notin S \right\}$$

Drugi rodzaj tranzycji jest wynikiem komunikacji pomiędzy dwoma składowymi wyrażeniami B , przez bramkę ze zbioru S , czyli jest przejściem do jednej z sytuacji opisanych wyrażeniem behawioralnym

$$\sum \left\{ \tau; (B_1 \parallel \dots \parallel B'_i \parallel \dots \parallel B'_j \parallel \dots \parallel B_n) \setminus S \mid B_i \xrightarrow{\alpha} B'_i \wedge B_j \xrightarrow{\bar{\alpha}} B'_j \wedge i < j \wedge gate(\alpha) \in S \right\}$$

Zachodzi zatem twierdzenie:

Twierdzenie 2.1

Jeżeli

$$B \equiv (B_1 \parallel \dots \parallel B_n) \setminus S$$

to

$$B \approx^c \sum \left\{ \tau; (B_1 \parallel \dots \parallel B'_i \parallel \dots \parallel B'_j \parallel \dots \parallel B_n) \setminus S \mid B_i \xrightarrow{\alpha} B'_i \wedge B_j \xrightarrow{\bar{\alpha}} B'_j \wedge i < j \right\} \\ + \sum \left\{ \alpha; (B_1 \parallel \dots \parallel B'_i \parallel \dots \parallel B_n) \setminus S \mid B_i \xrightarrow{\alpha} B'_i \wedge gate(\alpha) \notin S \right\}$$

Znaczenie twierdzenia polega na tym, że umożliwia ono krokową transformację wyrażenia behawioralnego do postaci, która zawiera tylko operatory prefiksowania akcją i operatora wyboru. Transformacja taka, oczywiście, nie dla każdego wyrażenia kończy się uzyskaniem wyrażenia w tej postaci o skończonej długości.

2.7. Uwagi końcowe

W tym rozdziale przedstawiono podstawowe informacje o języku CCS. Na bazie CCS opracowano pewne nowe wersje, między innymi CCS^{*} oraz SCCS^{*} [Fencott 1996]. Ten ostatni język jest odpowiednikiem języka SCCS (*Synchronous Calculus of Communicating Systems*) opracowanego przez Milnera [1983]. Zasadnicza różnica pomiędzy językami dotyczy sposobu opisu semantyki: semantyka SCCS jest semantyką przepłotową, natomiast SCCS^{*} – semantyką w pełni równoległą.

Wersje te powstały na gruncie rozważań na temat sposobu opisu semantyki języka. Przedstawiona semantyka języka CCS jest semantyką przepłotową. Oznacza to dwie rzeczy:

- tranzycje pomiędzy wyrażeniami następują w wyniku zajścia tylko pojedynczej interakcji,

- interakcje, które mogą zachodzić równocześnie, są reprezentowane zbiorem ciągów tranzycji, stanowiących wszystkie permutacje tych interakcji.

Wykorzystany opis przeplotowej semantyki operacyjnej opiera się na klasycznym już obecnie podejściu strukturalnej rekursji, zaproponowanej w pracach Plotkina [Plotkin, Hennessy 1979], [Plotkin 1981]. Zaletą semantyki przeplotowej jest prosta formalizacja, wadą natomiast, że może się wydawać nienaturalna.

Język CCS* jest pewną modyfikacją CCS, jego semantyka jest semantyką pełnej równoległości (*truly concurrency semantic*). Tranzycje w tej semantyce są widziane jako rezultat jednoczesnego zajścia wielu interakcji. Formalnie, jednocześnie zachodzące interakcje są reprezentowane wielozbiorami akcji. Semantyka pełnej równoległości jest wprawdzie naturalna, ale stwarza kłopoty techniczne przy analizie i dowodzeniu własności wyrażeń behawioralnych. Związki pomiędzy przeplotowym i nieprzeplotowym (równoległym) podejściem do definiowania semantyki sieci procesów omawia na przykład Dembiński [1997].

Język SCCS* jest synchroniczną wersją języka CCS*. Synchroniczna wersja wprowadza niejawnie czas dyskretny. Czas jest podzielony na kolejne przedziały, w których mogą zachodzić interakcje. W danym odcinku czasu proces albo wykonuje akcje, albo beczynnie czeka na kolejny odcinek czasu. Proces nie może jednak czekać beczynnie, jeśli nie jest to wyrażone jawnie. Opóźnianie oczekiwania na wykonanie akcji musi być wyrażone jawnie za pomocą dodatkowo wprowadzonej akcji jednostkowego opóźnienia. Rozszerzenie języka polega w pierwszej kolejności na wprowadzeniu tej dodatkowej akcji jednostkowego opóźnienia, a w następnej – na odpowiedniej definicji semantyki.

Różne rodzaje relacji równoważności służące do porównywania procesów są szeroko omawiane, między innymi w pracach: [Bolognesi, Smolka 1987], [De Nicola, Hennessy 1984], [Hennessy 1988], [Milner 1989].

Od specyfikacji w języku CCS, jak i w innych językach algebraicznych, oczekuje się spełnienia pewnych własności, jak bezpieczeństwa, żywotności, bezstronności itp. (zob. rozdz. 1.). Własności takie wyraża się w językach logiki, dla których są konstruowane odpowiednie systemy dowodzenia. W niniejszej monografii zagadnień tych się nie porusza. Są one omawiane między innymi w pracach: [Clarke, Emerson, Sistla 1983], [Hennessy, Reagan 1991], [Manna, Pnueli 1992].

3. Rozszerzenia czasowe CCS

3.1. Wstęp

Specyfikacja systemów czasu rzeczywistego musi dysponować pojęciem czasu. Czas do specyfikacji, dokładniej do języka specyfikacji, można wprowadzać w sposób niejawni lub jawny.

Z niejawnym wprowadzeniem czasu mamy do czynienia w językach SCCS i SCCS*, wspomnianych w zakończeniu poprzedniego rozdziału. Wynika to z założenia, że działanie systemu, w tym wykonywanie akcji, odbywa się w kolejnych okresach, wyznaczonych podziałem czasu globalnego zegara na kwanty czasu. Przy takim podejściu modelowany jest właściwie nie czas, ale „tykanie” globalnego zegara. W okresie pomiędzy kolejnymi „tyknięciami” specyfikacje procesów muszą jednoznacznie określać, czy proces wykonuje jakieś akcje, czy też czeka do następnego „tyknięcia”. Podejście takie może być odpowiednie tylko do niektórych systemów, wydaje się na przykład właściwe do modelowania synchronicznych układów cyfrowych, których działanie, z założenia, jest synchronizowane globalnym zegarem. Nie jest natomiast wystarczające do specyfikacji systemów czasu rzeczywistego.

Jawnie można wprowadzać czas na dwa sposoby:

Pierwszy polega na przypisaniu każdej akcji pewnego odcinka czasu przeznaczonego na jej realizację. Akcje komplementarne mają, oczywiście, jednakowy czas trwania. Akcje, które w danym momencie są oferowane, ale nie mogą być wykonane, muszą być opóźniane albo do momentu synchronizacji z akcjami komplementarnymi, albo do momentu wycofania ich oferty. Podejście takie ma odzwierciedlenie w wielu językach programowania czasu rzeczywistego.

Drugi sposób polega na wyróżnieniu dwóch rodzajów akcji: natychmiastowych i czasowych. Akcja czasowa wprowadza opóźnienie o zadany odcinek czasu. W ten sposób staje się możliwe modelowanie akcji komunikacyjnych, których realizacja zajmuje pewien odcinek czasu. Mianowicie czasową akcją komunikacyjną, która ma trwać pewien odcinek czasu, można modelować jako sekwencyjne złożenie trzech akcji:

- akcji natychmiastowej, reprezentującej początek realizacji akcji komunikacyjnej,
- akcji czasowej, reprezentującej wykonanie akcji komunikacyjnej (wymianę danych pomiędzy komunikującymi się partnerami),
- akcji natychmiastowej, reprezentującej zakończenie akcji komunikacyjnej.

Akcje opóźniające można interpretować dwojako. Interpretacja *silna* oznacza, że proces po upływie opóźnienia *musi* wykonać akcję natychmiastową, interpretacja *słaba* oznacza natomiast, że *może* wykonać taką akcję.

Drugi sposób wprowadzania czasu do języka specyfikacji ma dwie zalety. Pierwsza wiąże się z tym, że takie rozszerzenie nie modyfikuje istotnie znaczenia istniejących operatorów języka, a druga wyraża się w tym, że użycie czasu w wielu systemach czasu rzeczywistego wiąże się z oczekiwaniem na pewne akcje.

Z pojęciem czasu wiąże się struktura czasowa – zbiór wartości chwil czasowych wraz z pewnym uporządkowaniem. *Strukturą czasową* nazywa się parę

$$SC = \langle T, \leq \rangle \quad (3.1)$$

gdzie $\leq \subseteq T^2$ jest relacją porządku, która porządkuje chwile w sensie chwila wcześniejsza–późniejsza.

Jeżeli $\leq$ jest relacją porządku częściowego (to znaczy jest zwrotna, antysymetryczna i przechodnia), to mamy do czynienia ze strukturą czasu *rozgałęzionego*, a jeśli jest relacją porządku liniowego (to znaczy jest zwrotna, antysymetryczna, przechodnia i spójna), to mamy do czynienia ze strukturą czasu *liniowego*.

Strukturę czasową nazywa się *gęstą*, gdy

$$\forall t_1 \in T \bullet \forall t_2 \in T \bullet \exists t_3 \in T \bullet (t_1 < t_2 \Rightarrow t_1 < t_3 \wedge t_3 < t_2) \quad (3.2)$$

dyskretną prawostronnie, gdy

$$\begin{aligned} &\forall t_1 \in T \bullet \forall t_2 \in T \bullet ((t_1 < t_2 \wedge t_1 \neq t_2) \Rightarrow \\ &(\exists t_3 \in T \bullet (t_1 < t_3 \wedge t_1 \neq t_3) \wedge \neg \exists t_4 \in T \bullet (t_1 < t_4 \wedge t_4 < t_3 \wedge t_3 \neq t_4))) \end{aligned} \quad (3.3)$$

dyskretną lewostronnie, gdy

$$\begin{aligned} &\forall t_1 \in T \bullet \forall t_2 \in T \bullet (t_1 < t_2 \wedge t_1 \neq t_2) \Rightarrow \\ &(\exists t_3 \in T \bullet (t_3 < t_2 \wedge t_3 \neq t_2) \wedge \neg \exists t_4 \in T \bullet (t_4 < t_2 \wedge t_3 < t_4 \wedge t_3 \neq t_4))) \end{aligned} \quad (3.4)$$

Przykładem zbioru, na którym można zbudować strukturę ciągłą, jest zbiór liczb wymiernych, a dyskretną – zbiór liczb naturalnych. W dalszej części rozdziału zakłada się dyskretną lewo- i prawostronnie strukturę czasu liniowego. Wykorzystuje się jako zbiory chwil zbiór liczb naturalnych Nat lub zbiór liczb wymiernych Wym z relacją porządku $\leq$ w zbiorze liczb naturalnych lub liczb wymiernych, $n \leq m$ oznacza: chwila n nie jest późniejsza od chwili m . Strukturami czasowymi są zatem $\langle Nat, \leq \rangle$ oraz $\langle Wym, \leq \rangle$. Na strukturach tych są też wykonywane operacje dodawania i brania minimum dwóch chwil czasowych.

3.2. TCCS – wprowadzenie

Język TCCS (*Timed CCS*) jest czasowym rozszerzeniem CCS, polegającym na zastosowaniu drugiego z omówionych wyżej sposobów wprowadzania czasu do języka.

Operuje on dwoma rodzajami akcji języka CCS, czyli akcjami komunikacyjnymi i akcją wewnętrzną, które są realizowane natychmiastowo, oraz dodatkowo wprowadza akcje czasowe, których realizacja wymaga pewnego odcinka czasu. Akcja czasowa wprowadza opóźnienie o zadany odcinek czasu.

Czasową akcję komunikacyjną α w języku TCCS, która ma trwać odcinek czasu t , modeluje się jako sekwencyjne złożenie trzech akcji

$$\overline{\alpha Start}; (t); \overline{\alpha End}$$

gdzie:

$\overline{\alpha Start}$ oraz $\overline{\alpha End}$ są akcjami natychmiastowymi z języka CCS,
 (t) jest nowo wprowadzoną akcją czasową (opóźniającą).

Z wprowadzeniem akcji czasowych – jak wspomniano wcześniej – wiążą się dwie możliwe interpretacje dotyczące momentów wykonywania akcji. Pierwsza interpretacja – silniejsza – oznacza, że rozpoczęcie akcji czasowej *musi* nastąpić natychmiast po zakończeniu poprzedzającej ją akcji. Druga interpretacja – słabsza – oznacza, że rozpoczęcie akcji czasowej *może* nastąpić w dowolnym momencie po zakończeniu poprzedzającej ją akcji. Wyróżnia się więc dwie wersje języka TCCS: silną – sTCCS (*strong TCCS*) [Moller, Tofts 1989] i słabą – wTCCS (*weak TCCS*) [Tofts 1989].

Opisywana tu wersja języka TCCS [Fencott 1996] jest wersją pośrednią, gdyż – przyjmując silną interpretację opóźnień – wprowadza dodatkowe mechanizmy uelastyczniające tę interpretację. Potrzeba ich wprowadzenia wynika z faktu, że wyłączenie stosowania silnej interpretacji opóźnień powoduje znaczne ograniczenie ekspresji języka.

Pierwszy z dwóch wprowadzonych mechanizmów pozwala na określenie czy dana akcja musi być wykonana natychmiast w najwcześniejszym możliwym momencie, czy też może oczekiwać na wykonanie dowolnie długo. Akcje, które mogą oczekiwać na swoją realizację dowolnie długo, wyróżnia się składniowo przez podkreślenie ich nazwy. Nazwy akcji natomiast, które muszą być wykonywane w najwcześniejszym możliwym momencie nie są podkreślane. W sekwencyjnym złożeniu akcji

$$\alpha; \underline{\beta}; \gamma$$

akcja α musi być wykonana natychmiast, akcja $\underline{\beta}$ w dowolnym momencie po zakończeniu akcji α , natomiast akcja γ natychmiast po zakończeniu akcji $\underline{\beta}$.

Drugi mechanizm polega na wprowadzeniu, oprócz operatora $+$ z języka CCS, dodatkowego rodzaju operatora wyboru reprezentowanego symbolem $++$. Potrzeba dodatkowego operatora wiąże się ze sposobem reagowania na upływ czasu. Operator $+$ jest nazywany słabym, a $++$ – silnym operatorem wyboru. Różnice w ich interpretacji wyjaśnia przykład. Niech będzie dany proces

$$\mu P.((3); \alpha; P + (5); \underline{\beta}; P)$$

Obie części składowe wyrażenia wyboru są prefiksowane akcjami opóźnienia (3) oraz (5). Operator + dopuszcza wybór obu części składowych tego wyrażenia, przy czym czyni to w sposób następujący: Jeżeli po upływie opóźnienia 3 jednostek czasu od wywołania procesu P nastąpi realizacja akcji α , to następnie jest powtarzane wykonanie wywołania P . Jeżeli natomiast po upływie 3 jednostek czasu akcja α nie zostanie wykonana, to następuje opóźnienie o dalsze 2 jednostki czasu (do zakończenia opóźnienia wyrażanego akcją czasową (5)), po czym musi nastąpić realizacja akcji β , a po jej wykonaniu następuje ponowne wywołanie procesu P .

Jeżeli operator + zastąpimy operatorem ++, to interpretacja procesu

$$\mu P.((3); \alpha; P ++ (5); \beta; P)$$

będzie całkiem inna. Operator dopuszcza mianowicie wybór tylko tej części składowej, która jest opóźniana krócej. W naszym przykładzie oznacza to, że może być wybrana tylko lewa składowa (3); $\alpha; P$, natomiast prawa składowa (5); $\beta; P$ nigdy nie będzie wybrana. Po upływie zatem opóźnienia 3 jednostek czasu od wywołania procesu P musi nastąpić realizacja akcji α , a następnie będzie powtarzane wykonanie wywołania P .

3.3. TCCS – definicja formalna

TCCS wprowadza nową kategorię semantyczną – dziedzinę czasową w postaci struktury relacyjnej

$$\langle Time, \leq, + \rangle \quad (3.5)$$

gdzie:

$Time$ jest dowolnym zbiorem przeliczalnym, zwykle przyjmuje się, że jest to zbiór liczb naturalnych,

$\leq$ jest relacją porządku liniowego na zbiorze $Time$,

$+$: $Time^2 \rightarrow Time$ jest operacją dodawania.

Symbol + jest przeciążony, w zależności od kontekstu wyznaczonego przez jego argumenty oznacza funkcję dodawania bądź operator wyboru wyrażeń behawioralnych.

Składnia zbioru wyrażeń czasowych behawioralnych $TBeh$ języka TCCS, w notacji BNF, jest określona następująco:

$$B ::= 0 \mid \underline{0} \mid P \mid B \setminus S \mid B[f] \mid (B) \mid \alpha; B \mid (t); B \mid \underline{\alpha}; B \mid B \parallel B \mid B + B \mid B ++ B \quad (3.6)$$

gdzie:

$B \in TBeh$, $P \in Proc$, $S \subseteq A$, $f: G \rightarrow G$ to funkcja przemianowania bramek akcji,

$t \in Time$ jest długością czasu opóźnienia. Kolejność wprowadzenia operatorów odpowiada malejącej kolejności ich priorytetów.

Lista definicji procesów czasowych $TDef$ jest określona tak samo jak dla bezczasowego CCS.

Semantyka operacyjna specyfikacji czasowej $TSpec = \langle B_0, Def_P \rangle$, gdzie: $B_0 \in TBeh$, a $TDef_P \in TDef$ jest listą definicji procesów czasowych, jest definiowana na podstawie czasowego etykietowanego systemu przejść postaci

$$TS(TSpec) = \langle TBeh, TAct, Time, TTr, B_0 \rangle \quad (3.7)$$

gdzie:

$TBeh$ jest zbiorem czasowych wyrażeń behawioralnych,

$TAct = Act \cup \{(t) \mid t \in Time\}$ jest zbiorem akcji,

$Time$ jest zbiorem chwil,

$TTr = \{\xrightarrow{\alpha} \subseteq TBeh \times TBeh \mid \alpha \in Act\} \cup \{\xrightarrow{t} \subseteq TBeh \times TBeh \mid t \in Time\}$ jest rodziną relacji przejść pomiędzy wyrażeniami behawioralnymi,
 B_0 jest początkowym wyrażeniem behawioralnym.

Przejście postaci (symbol akcji α nad strzałką)

$$B_1 \xrightarrow{\alpha} B_2 \quad (3.8)$$

zwane *przejściem akcyjnym*, oznacza – jak w przypadku CCS – że proces, którego stan jest reprezentowany wyrażeniem behawioralnym $B_1 \in TBeh$, w wyniku realizacji akcji $\alpha \in TAct$, zmienia się na stan reprezentowany wyrażeniem $B_2 \in TBeh$.

Przejście natomiast postaci (symbol upływu czasu t pod strzałką)

$$B_1 \xrightarrow{t} B_2 \quad (3.9)$$

zwane *przejściem czasowym*, oznacza, że proces, którego stan jest reprezentowany wyrażeniem behawioralnym $B_1 \in TBeh$, w wyniku upływu odcinka czasu o długości $t \in Time$, zmienia się na stan reprezentowany wyrażeniem $B_2 \in TBeh$.

Rodzina relacji przejść jest definiowana rekursywnie. Zbiór aksjomatów i reguł dla TCCS jest w zasadzie rozszerzeniem odpowiedniego zbioru dla CCS. Rozszerzenie jest związane z dołączeniem przejść czasowych.

W zbiorze reguł dotyczących przejść akcyjnych rozszerza się reguły dotyczące operatora wyboru. Mianowicie reguły (*R-ch*) odnoszą się teraz do obu operatorów wyboru $+$ oraz $++$, czyli

$$\text{dla } op \in \{+, ++\} \quad (R-ch)$$

Wprowadza się też dodatkowy aksjomat dla akcji, których rozpoczęcie może być dowolnie opóźniane

$$\alpha; B \xrightarrow{\alpha} B \quad (A-pre)$$

Zbiór aksjomatów i reguł dotyczących przejść czasowych jest dosyć złożony. Bezpośrednio wprowadza się następujące aksjomaty i reguły:

Opóźnianie

$$\underline{0} \xrightarrow{t} \underline{0} \quad \underline{\alpha}; B \xrightarrow{t} \underline{\alpha}; B \quad (A\text{-del})$$

Prefiksowanie akcją czasową

$$(t); B \xrightarrow{t} B \quad (A_1\text{-}t\text{-pre})$$

$$(t + s); B \xrightarrow{t} (s); B \quad (A_2\text{-}t\text{-pre})$$

$$\frac{B \xrightarrow{t} B'}{(t + s); B \xrightarrow{t+s} B'} \quad (R\text{-}t\text{-pre})$$

Dalszy zestaw reguł wymaga użycia pomocniczych własności, stwierdzających czy dane czasowe wyrażenie behawioralne może być opóźniane o zadany odcinek czasu.

Niech prawdziwość formuły $del(B, t)$, gdzie del jest predykatem, oznacza, że czasowe wyrażenie behawioralne B może być opóźniane o odcinek czasu t , czyli że po upływie t jednostek czasu od dowolnie ustalonej chwili wyrażenie B będzie oferować swojemu otoczeniu ten sam zbiór akcji komunikacyjnych. Predykat del jest zdefiniowany rekursywnie przez następujący zbiór aksjomatów i reguł:

$$del(\underline{0}, t)$$

$$del(\underline{\alpha}; B, t)$$

$$del((t); B, s) \quad \text{dla } s \leq t$$

$$\frac{del(B, t_0)}{del((t); B, s)} \quad \text{dla } s \leq t_0 + t$$

$$\frac{del(B_i, t)}{del(B_1 + B_2, t)} \quad \text{dla } i = 1, 2$$

$$\frac{del(B_1, t) \quad del(B_2, t)}{del(B_1 ++ B_2, t)}$$

$$\frac{del(B_1, t) \quad del(B_2, t)}{del(B_1 \parallel B_2, t)}$$

$$\frac{del(B, t)}{del(B \setminus S, t)}$$

$$\frac{del(B, t)}{del(B[f], t)}$$

$$\frac{del(B, t)}{del(P, t)} \quad \text{dla } \mu P.B$$

Wybór czasowy

$$\frac{B_1 \xrightarrow{t} B'_1}{B_1 + B_2 \xrightarrow{t} B'_1} \quad \neg del(B_2, t) \quad (R1-t-ch)$$

$$\frac{B_2 \xrightarrow{t} B'_2}{B_1 + B_2 \xrightarrow{t} B'_2} \quad \neg del(B_1, t) \quad (R2-t-ch)$$

$$\frac{B_1 \xrightarrow{t} B'_1 \quad B_2 \xrightarrow{t} B'_2}{B_1 + B_2 \xrightarrow{t} B'_1 + B'_2} \quad (R3-t-ch)$$

$$\frac{B_1 \xrightarrow{t} B'_1 \quad B_2 \xrightarrow{t} B'_2}{B_1 ++ B_2 \xrightarrow{t} B'_1 + B'_2} \quad (R3-t-ch)$$

Złożenie równoległe czasowe

$$\frac{B_1 \xrightarrow{t} B'_1 \quad B_2 \xrightarrow{t} B'_2}{B_1 \parallel B_2 \xrightarrow{t} B'_1 \parallel B'_2} \quad (R-t-par)$$

Restrykcja czasowa

$$\frac{B \xrightarrow{t} B'}{B \setminus S \xrightarrow{t} B' \setminus S} \quad (R-t-res)$$

Przemianowanie czasowe

$$\frac{B \xrightarrow{t} B'}{B[f] \xrightarrow{t} B'[f]} \quad (R-t-ren)$$

Instancja procesu czasowa

$$\frac{B \xrightarrow{t} B'}{P \xrightarrow{t} B'} \quad \text{dla } \mu P.B \quad (R1-t-ins)$$

$$\frac{B \xrightarrow{t} B'}{B \xrightarrow{t} P} \quad \text{dla } \mu P.B \quad (R2-t-ins)$$

Wprowadźmy oznaczenie: niech

$$B_1 \xrightarrow{at} B_2,$$

gdzie $at \in TAct$, oznacza

$$B_1 \xrightarrow{at} B_2, \quad \text{gdzie } at \in Act,$$

oraz

$$B_1 \xrightarrow{at} B_2, \quad \text{gdzie } at \notin Act.$$

Ciąg tranzycji (obliczenie) generowany przez czasowy system przejść $TS(TSpec)$ ma postać

$$B_0 \xrightarrow{at_1} B_1 \xrightarrow{at_2} B_2 \dots \xrightarrow{at_n} B_n \quad (3.10a)$$

lub

$$B_0 \xrightarrow{at_1} B_1 \xrightarrow{at_2} B_2 \dots \xrightarrow{at_n} B_n \xrightarrow{at_{n+1}} \dots \quad (3.10b)$$

gdzie:

B_0 jest początkowym wyrażeniem specyfikacji $TSpec$,

B_n w przypadku obliczenia skończonego (a) jest wyrażeniem końcowym, to znaczy takim, dla którego nie istnieją przejścia do innych wyrażeń.

Z zastosowaniem poprzednio wprowadzonych oznaczeń określa się następujące zbiory ciągów:

Definicja 3.1

Specyfikacja czasowa $TSpec = \langle B_0, TDecl \rangle$ generuje zbiór ciągów akcji

$$TSeq(B_0) = TSeqFin(B_0) \cup TSeqInf(B_0) \quad (3.11)$$

gdzie:

$$TSeqFin(B_0) = \{st \in (Act \cup Time)^* \mid B_0 \xrightarrow{st} \} \quad (3.12)$$

$$TSeqInf(B_0) = \{st \in (Act \cup Time)^\infty \mid \forall st' \in pref(st) \bullet B_0 \xrightarrow{st'} \} \quad (3.13)$$

3.4. RTCCS – wprowadzenie

Język RTCCS jest oryginalnym rozszerzeniem czasowym języka CCS [Huzar 1996]. Rozszerzenie opiera się na dwóch głównych założeniach:

- wszystkie akcje (w tym akcje wewnętrzne) są akcjami czasowymi, ich realizacja może wymagać niepustego okresu czasu,
- zachodzenie akcji obserwowalnych jest wynikiem synchronizacji komunikujących się procesów, co oznacza, że rozpoczęcie realizacji pary komplementarnych akcji czasowych jest wyznaczone momentem ich synchronizacji, a zainicjowana komunikacja pary procesów jest nieprzerywalna.

Przyjmuje się model relatywnego, gęstego czasu globalnego. Dziedzina czasowa $Time$ jest reprezentowana przez liczby wymierne.

Przedstawiane tu rozszerzenie ogranicza się do synchronicznej komunikacji dwóch procesów. W pracy [Huzar 1996] rozpatruje się model ogólniejszy, dopuszczający synchroniczną komunikację więcej niż dwóch procesów. Do takiego modelu współpracy procesów wrócimy w następnych rozdziałach.

Niech A będzie zbiorem obserwowalnych akcji komunikacyjnych oraz niech i będzie nieobserwowalną akcją wewnętrzną. Niech $Act = A \cup \{i\}$. Każdej akcji $a \in Act$

przypisuje się pewien zbiór $\Delta(a)$ dopuszczalnych czasów wykonywania. Zakłada się, że $u \in \Delta(a)$ jest wartością dodatnią. Założenie to wyklucza sytuację, że w pewnej chwili, gdy akcje mają zerowy czas wykonywania, może być wykonywana nieskończona liczba akcji.

Zbiór $Del = \{\Delta(a) \mid a \in Act\}$, gdzie $\Delta(a) \subseteq Time \setminus \{0\}$, nazwiemy zbiorem własności dynamicznych akcji.

Zbiór zdarzeń czasowych jest określony jako $TimeEv = Time \times Act$.

Realizacja akcji czasowej a będzie reprezentowana dwoma zdarzeniami czasowymi: zdarzenie czasowe reprezentujące początek akcji a jest parą $\langle t_p, a \rangle$, natomiast zdarzenie reprezentujące zakończenie akcji – parą $\langle t_k, i \rangle$, gdzie $t_k, t_p \in Time$ oraz $(t_k - t_p) \in \Delta(a)$.

W przypadku szczególnym realizację akcji wewnętrznej i reprezentują dwa zdarzenia czasowe: $\langle t_p, i \rangle$ oraz $\langle t_k, i \rangle$.

Para komunikujących się akcji wykonuje się wspólnie przez dowolny odcinek czasu, dlatego uściślamy pojęcie czasowych akcji komplementarnych. Akcje $\alpha, \beta \in Act$ są akcjami komplementarnymi, gdy $\alpha = \bar{\beta}$, w takim samym znaczeniu jak w języku CCS, to znaczy jedna z nich jest akcją wejściową, a druga akcją wyjściową na tej samej bramce, oraz dodatkowo, gdy $\Delta(\alpha) \cap \Delta(\bar{\beta}) \neq \emptyset$, co oznacza, że obie akcje dopuszczają ten sam czas realizacji. Należy zwrócić uwagę, że czas realizacji komplementarnych akcji może być wyznaczony niejednoznacznie – w takim przypadku mamy do czynienia z niedeterminizmem wyboru czasu wykonywania.

W dalszej części, gdy będziemy mówić o realizacji komplementarnych akcjach czasowych α oraz $\bar{\alpha}$, będziemy zawsze zakładać, że wspólny okres ich realizacji $u \in Time$ spełnia warunek $u \in \Delta(\alpha) \cap \Delta(\bar{\alpha})$.

3.5. RTCCS – definicja formalna

Składnia

Przez $Proc$ oznaczmy zbiór nazw procesów.

Zbiór wyrażeń behawioralnych $RTBeh$, w notacji BNF, jest określony następująco:

$$B ::= 0 \mid P \mid B \setminus S \mid B[f] \mid (B) \mid \alpha; B \mid \delta(u); B \mid B \mid U \mid B \mid B + B \quad (3.14)$$

gdzie: $B \in RTBeh$, $P \in Proc$, $S, U \subseteq A$, $\alpha \in Act$ oraz $f: G \rightarrow G$ jest funkcją przemianowania bramek akcji. Kolejność wprowadzenia operatorów odpowiada malejącej kolejności ich priorytetów.

Zbiór list definicji procesów $RTDef$ jest określony następująco:

$$RTD ::= \mu P. B \mid RTD, \mu P. B \quad (3.15)$$

Składnia wprowadza dwie konstrukcje, które są inne od konstrukcji języka CCS. Są to: wyrażenie opóźnienia $\delta(u)$; B oraz zmodyfikowana forma złożenia równoległego $B \mid U \mid B$. Ich znaczenie jest następujące:

Wyrażenie opóźniające $\delta(u)$; B w danym momencie t nie oferuje swemu otoczeniu żadnej akcji komunikacyjnej, natomiast po upływie odcinka czasu długości u oferuje akcje komunikacyjne reprezentowane przez wyrażenie B .

Wyrażenie złożenia równoległego $B_1 \mid U \mid B_2$, gdzie $\mid U \mid$ jest operatorem złożenia równoległego parametryzowanego podzbiorem akcji U , zachowuje się tak, jak zachowują się jednocześnie wyrażenia B_1 oraz B_2 . Dodatkowo wyrażenia te mogą się wzajemnie komunikować przez swoje akcje komplementarne tylko na bramkach ze zbioru $gate(U)$.

Pojęcia *dozorowanego i regularnie zbudowanego* wyrażenia są rozumiane tak samo jak dla CCS.

Specyfikacja zachowania w języku RTCCS jest zdefiniowana jako para

$$RTSpec = \langle B_0, RTDef_P \rangle \quad (3.16)$$

gdzie:

$B_0 \in RTBeh$ jest początkowym wyrażeniem behawioralnym,

$RTDef_P \in RTDef$ jest skończoną listą definicji pewnych procesów czasu rzeczywistego $P_1, \dots, P_n$.

Definicja procesu o nazwie P_i ($i = 1, \dots, n$) ma postać $\mu P_i.B_i$, gdzie wyrażenie behawioralne $B_i \in RTBeh$ jest treścią procesu. Każdy proces, którego instancja występuje w wyrażeniach behawioralnych $B_0, \dots, B_n$ ma swoją definicję w zbiorze $RTDef$. Instancje procesów o tej samej nazwie mają jedną wspólną definicję.

Semantyka

Semantyka operacyjna specyfikacji czasu rzeczywistego $RTSpec = \langle B_0, RTDef_P \rangle$ jest definiowana na podstawie czasowego etykietowanego systemu przejść postaci

$$TS(RTSpec) = \langle Conf, TimeEv, RTTr, C_0 \rangle \quad (3.17)$$

gdzie:

$Conf = RTBeh \times Time$ jest zbiorem konfiguracji czasowych; para $\langle B, t \rangle \in Conf$ reprezentuje specyfikowany system, poczynając od chwili t – wyrażenie behawioralne B określa zachowanie tego systemu od chwili t ,

Act jest zbiorem zdarzeń czasowych,

$RTTr \subseteq Conf \times TimeEv \times Conf$ jest relacją przejścia pomiędzy czasowymi wyrażeniami behawioralnymi (relację zmian konfiguracji),

C_0 jest konfiguracją początkową, czyli parą $\langle B_0, t_0 \rangle$ (B_0 jest początkowym wyrażeniem behawioralnym, t_0 – chwilą początkową).

Trójka $\langle\langle B_1, t_1 \rangle, \langle t_2, a \rangle, \langle B_2, t_2 \rangle\rangle \in RTTr$, nazywana przejściem, będzie zapisywana w postaci

$$\langle B_1, t_1 \rangle \xrightarrow{\langle t_2, a \rangle} \langle B_2, t_2 \rangle$$

albo, nieco krócej, w postaci

$$\langle B_1, t_1 \rangle \xrightarrow{\lambda} \langle B_2, t_2 \rangle,$$

gdzie $\lambda \in TimeEv$.

Przejście oznacza, że system ze swoją konfiguracją $\langle t_1, B_1 \rangle$, odnoszącą się do chwili t_1 , na skutek zajścia zdarzenia czasowego $\langle t_2, a \rangle$ w chwili t_2 , gdzie $t_1 \leq t_2$, zmienia swą konfigurację na $\langle t_2, B_2 \rangle$.

Relacja przejścia pomiędzy czasowymi wyrażeniami behawioralnymi jest definiowana rekursywnie przez podany niżej zbiór aksjomatów i reguł. Reguły te są budowane według następujących założeń:

- komunikacja odbywa się dokładnie pomiędzy dwoma procesami,
- synchronizacja pomiędzy komunikującymi się procesami następuje w najwcześniejszym możliwie momencie (*as soon as possible principle*),
- dwa zsynchronizowane komunikujące się procesy pozostają powiązane ze sobą przez wspólnie ustalony odcinek czasu.

W konstrukcji reguł będą wykorzystane pewne funkcje pomocnicze zdefiniowane rekursywnie w tabeli 3.1.

Tabela 3.1

B	$F(B)$	$C(B)$
0	$\emptyset$	$\emptyset$
P	$F(B)$ dla $\mu P.B$	$C(B)$ dla $\mu P.B$
$B \setminus S$	$F(B) \setminus S \cup h(F(B) \cap S)$	$C(B) \setminus S \cup h(C(B) \cap S)$
$B[f]$	$F(B)[f]$	$C(B)[f]$
(B)	$F(B)$	$C(B)$
$\alpha; B$	$\{\alpha\}$	$\emptyset$
$\delta(u); B$	$\{i\}^*$	$\{i\}^*$
$B_1 \mid U \mid B_2$	$F(B_1) \cup F(B_2)$	$C(B_1) \setminus U \cup C(B_2) \setminus U \cup F(B_1) \cap F(B_2) \cap U$
$B_1 + B_2$	$F(B_1) \cup F(B_2)$	$C(B_1) \cup C(B_2)$

^{*)} Natychmiastowa akcja wewnętrzna i wynika z przyjętego sposobu reprezentacji realizacji akcji $\delta(u)$ przez zdarzenia czasowe $\langle t_p, i \rangle$ oraz $\langle t_k, i \rangle$.

Pierwsza funkcja $F: RTBeh \rightarrow Act$ służy do wyznaczania, dla danego wyrażenia behawioralnego B , zbioru tych akcji, które są oferowane przez wyrażenie do wykonania

w pierwszej kolejności. To, która spośród tych instancji akcji zostanie wybrana do wykonania, zależy od decyzji otoczenia wyrażenia B . Także czas zainicjowania wybranej akcji zależy od otoczenia, przy czym – w razie zbyt długiego odraczania inicjacji akcji – może wcześniej nastąpić akcja wewnętrzna.

Druga funkcja $C: RTBeh \rightarrow Act$ wyznacza podzbiór tych akcji, które mogą być wykonywane jako pierwsze, niezależnie od gotowości otoczenia wyrażenia B .

Występująca w tabeli pomocnicza funkcja h jest zdefiniowana następująco:

$$h(U) = \begin{cases} \emptyset & \text{dla } U = \emptyset \\ \{i\} & \text{dla } U \neq \emptyset \end{cases}$$

Należy zwrócić uwagę na to, że dowolna akcja $\alpha \in Act$ w wyrażeniu behawioralnym B może mieć wiele wystąpień (instancji). Jeżeli $\alpha \in F(B)$ lub $\alpha \in C(B)$, to może to oznaczać wiele wystąpień tej akcji, które spełniają warunki określone przez definicje funkcji F oraz C . Na przykład wyrażenie B postaci

$$\alpha; \beta; P_1 + \alpha; \gamma; \alpha; P_2$$

ma trzy wystąpienia akcji α , z czego dwa wystąpienia – pierwsze wystąpienia α po lewej i po prawej stronie wyrażenia B – należą do zbioru $F(B) = \{\alpha\}$.

Następne dwie funkcje, zdefiniowane w tabeli 3.2, są związane z wyznaczaniem wpływu czasu.

Tabela 3.2

B	$Age(B, u)$	$Ev(B)$
$\mathbf{0}$	$\mathbf{0}$	∞
$\alpha; B$	$\alpha; B$	∞ dla $\alpha \in A$
$\delta(u'); B$	$\delta(u' - u); B$ gdy $u' \geq u$	u' gdy $u' > 0$
P	$Age(B, u)$ dla $\mu P.B$	$Ev(B)$ dla $\mu P.B$
$B \setminus S$	$Age(B, u) \setminus S$	$\begin{cases} 0 & \text{gdy } C(B) \cap S \neq \emptyset \\ Ev(B) & \text{gdy } C(B) \cap S = \emptyset \end{cases}$
$B[f]$	$Age(B, u)[f]$	$Ev(B)$
(B)	$Age(B, u)$	$Ev(B)$
$B_1 \mid U \mid B_2$	$Age(B_1, u) \mid U \mid Age(B_2, u)$	$\begin{cases} 0 & \text{gdy } C(B) \cap U \neq \emptyset \\ \min(Ev(B_1), Ev(B_2)) & \text{gdy } C(B) \cap U = \emptyset \end{cases}$
$B_1 + B_2$	$Age(B_1, u) + Age(B_2, u)$	$\min(Ev(B_1), Ev(B_2))$

Pierwsza z nich $Age : RTBeh \times Time \rightarrow RTBeh$ określa transformację wyrażenia behawioralnego na skutek samego upływu czasu przy braku zajścia zdarzeń czasowych.

Druga funkcja $Ev : RTBeh \rightarrow Time$ wyznacza dla danego wyrażenia behawioralnego najwcześniejszy moment czasu względem bieżącej chwili, w którym może nastąpić zmiana konfiguracji niezależna od otoczenia wyrażenia. Zmiana ta jest wynikiem zajścia zdarzenia czasowego niezależnego od otoczenia wyrażenia.

Mając wyżej zdefiniowane funkcje, można zdefiniować rekursywnie relację zmian konfiguracji.

Proces pusty – brak aksjomatów.

Instancja procesu

$$\frac{\langle \mu P.B, t \rangle \xrightarrow{\lambda} \langle B', t' \rangle}{\langle P, t \rangle \xrightarrow{\lambda} \langle B', t' \rangle} \quad (R-ins)$$

Rekursja

$$\frac{\langle B[P ::= \mu P.B], t \rangle \xrightarrow{\lambda} B'}{\langle \mu P.B, t \rangle \xrightarrow{\lambda} B'} \quad (R-rec)$$

gdzie notacja postaci $B[P ::= B']$ oznacza modyfikację wyrażenia B , polegającą na tekstowym zastąpieniu każdego wolnego wystąpienia nazwy procesu P (zob. tab. 2.1) w wyrażeniu B wyrażeniem B' .

Restrykcja

$$\frac{\langle B, t \rangle \xrightarrow{\langle i', \alpha \rangle} \langle B', t' \rangle}{\langle B \setminus S, t \rangle \xrightarrow{\langle i', \alpha \rangle} \langle B' \setminus S, t' \rangle} \quad gate(\alpha) \notin S \quad (R_1-res)$$

$$\frac{\langle B, t \rangle \xrightarrow{\langle i', \alpha \rangle} \langle B', t' \rangle}{\langle B \setminus S, t \rangle \xrightarrow{\langle i', i \rangle} \langle B' \setminus S, t \rangle} \quad gate(\alpha) \in S \wedge \alpha \in C(B) \quad (R_2-res)$$

Przemianowanie

$$\frac{\langle B, t \rangle \xrightarrow{\langle i', \alpha \rangle} \langle B', t' \rangle}{\langle B[f], t \rangle \xrightarrow{\langle i', f(\alpha) \rangle} \langle B'[f], t' \rangle} \quad \text{gdzie } f(\alpha) = \begin{cases} f(g)! & \text{dla } \alpha \equiv g! \\ f(g)? & \text{dla } \alpha \equiv g? \end{cases} \quad (R-ren)$$

Prefiksowanie akcji

$$\langle \alpha; B, t \rangle \xrightarrow{\langle i', \alpha \rangle} \langle \delta(u); B, t' \rangle \quad \text{dla } t \leq t' \quad (A_1-pre)$$

Opóźnienie

$$\langle \delta(u); B, t \rangle \xrightarrow{\langle t+u, i \rangle} \langle B, t+u \rangle \quad (A_2-pre)$$

Złożenie równoległe

$$\frac{\langle B_1, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_1, t' \rangle}{\langle B_1 \parallel B_2, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_1 \parallel [U] \mid \text{Age}(B_2, t' - t), t' \rangle} \quad \text{dla } \alpha \notin U \wedge t' \leq \text{Ev}(B_2) \quad (R_1\text{-par})$$

$$\frac{\langle B_2, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_2, t' \rangle}{\langle B_1 \parallel B_2, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle \text{Age}(B_1, t' - t) \parallel [U] \mid B'_2, t' \rangle} \quad \text{dla } \alpha \notin U \wedge t' \leq \text{Ev}(B_1) \quad (R_2\text{-par})$$

$$\frac{\begin{array}{c} \langle B_1, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_1, t' \rangle \\ \langle B_2, t \rangle \xrightarrow{\langle t', \bar{\alpha} \rangle} \langle B'_2, t' \rangle \end{array}}{\langle B_1 \parallel B_2, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_1 \parallel [U] \mid B'_2, t' \rangle} \quad \text{dla } \alpha \in U \quad (R_3\text{-par})$$

Reguła $R_3\text{-par}$ odnosi się do synchronizacji akcji czasowych. Wiąże się z nią założenie o komplementarności synchronizujących się akcji czasowych. Synchronizacja następuje w pierwszym możliwym momencie. Reguła nie określa natomiast ani sposobu wyboru wspólnego czasu realizacji akcji α i $\bar{\alpha}$, ani nie określa jawnie wybranego czasu. Konsekwencją wyboru wspólnej wartości czasu jest następująca własność wyrażeń B'_1 i B'_2 :

Własność

Niech $\delta(u_1)$; B''_1 oraz $\delta(u_2)$; B''_2 będą podwyrażeniami wyrażeń B'_1 oraz B'_2 , takimi że:

$$B_1 \equiv B'_1 [\alpha; B''_1 := \delta(u_1); B''_1]$$

$$B_2 \equiv B'_2 [\bar{\alpha}; B''_2 := \delta(u_2); B''_2]$$

wówczas $u_1 = u_2$.

Do reguły $R_3\text{-par}$ można dołączyć dowolny mechanizm takiego wyboru – deterministyczny lub probabilistyczny.

Wybór

$$\frac{\langle B_1, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_1, t' \rangle}{\langle B_1 + B_2, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_1, t' \rangle} \quad \text{dla } t' \leq \text{Ev}(B_2) \quad (R_1\text{-ch})$$

$$\frac{\langle B_2, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_2, t' \rangle}{\langle B_1 + B_2, t \rangle \xrightarrow{\langle t', \alpha \rangle} \langle B'_2, t' \rangle} \quad \text{dla } t' \leq \text{Ev}(B_1) \quad (R_2\text{-ch})$$

Podane aksjomaty i reguły generują ciągi przejść czasowych, które wyznaczają semantykę specyfikacji.

Ciąg tranzycji (obliczenie) generowany przez czasowy system przejść $TS(RTSpec)$ ma postać

$$\langle B_0, t_0 \rangle \xrightarrow{\langle t_2, a_1 \rangle} \langle B_1, t_1 \rangle \xrightarrow{\langle t_2, a_2 \rangle} \langle B_2, t_2 \rangle \dots \xrightarrow{\langle t_n, a_n \rangle} \langle B_n, t_n \rangle \quad (3.18a)$$

lub

$$\langle B_0, t_0 \rangle \xrightarrow{\langle t_2, a_1 \rangle} \langle B_1, t_1 \rangle \xrightarrow{\langle t_2, a_2 \rangle} \langle B_2, t_2 \rangle \dots \xrightarrow{\langle t_n, a_n \rangle} \langle B_n, t_n \rangle \dots \quad (3.18b)$$

gdzie:

$$t_1 \leq t_2 \leq \dots t_n \leq \dots$$

B_0 jest początkowym wyrażeniem behawioralnym,

$\langle B_0, t_0 \rangle$ jest konfiguracją początkową specyfikacji $RTSpec$,

$\langle B_n, t_n \rangle$ w przypadku obliczenia skończonego (a) jest konfiguracją końcową, to znaczy taką, z której nie ma przejścia do innych konfiguracji.

Definicja 3.1.

Specyfikacja czasowa $RTSpec = \langle B_0, RTDecl \rangle$ generuje zbiór ciągów akcji

$$RTSeq(B_0) = RTSeqFin(B_0) \cup RTSeqInf(B_0), \quad (3.19)$$

gdzie:

$$RTSeqFin(B_0) = \{c \in Conf^* \mid \langle B_0, t_0 \rangle \xrightarrow{c} \} \quad (3.20)$$

$$RTSeqInf(B_0) = \{c \in Conf^\infty \mid \forall c' \in pref(c) \bullet \langle B_0, t_0 \rangle \xrightarrow{c'} \} \quad (3.21)P$$

Zdefiniowana semantyka jest semantyką przeplotową czasu rzeczywistego. Przeplot odnosi się do zdarzeń występujących w tej samej chwili.

Bezpośrednio z reguł definiujących relację przejścia wynika, że kolejne przejścia są uporządkowane czasowo. Na ogół moment przejścia z danej konfiguracji do następnej zależy od otoczenia wyrażenia. Jeżeli specyfikacja jest komunikacyjnie zamknięta, to zachodzi lemat:

Lemat 3.1

Jeżeli specyfikacja reprezentowana wyrażeniem behawioralnym B jest komunikacyjnie zamknięta oraz istnieje przejście

$$\langle B, t \rangle \xrightarrow{\lambda} \langle B', t' \rangle, \quad (3.22)$$

to czas t' jest wyznaczony jednoznacznie.

Dowód jest przedstawiony w pracy [Huzar 1996].

Niech Del_1 oraz Del_2 będą dwoma zbiorami własności dynamicznych akcji komunikacyjnych. Będziemy mówić, że Del_1 jest podzbiorem Del_2 , co oznaczamy $Del_1 \subseteq Del_2$, gdy dla każdego $a \in Act$ zachodzi $\Delta_1(a) \subseteq \Delta_2(a)$. Zachodzi oczywista własność

Lemat 3.2

Dla dowolnego wyrażenia behawioralnego B jeżeli $Del_1 \subseteq Del_2$, to $RTSeq(B) \subseteq RTSeq(B)$.

3.6. RTCCS – przykłady

Rozpatrzmy teraz serię prostych przykładów ilustrujących stosowanie reguł semantycznych.

Przykład 1

Niech $\langle \mu P.a; P, t_0 \rangle$ będzie konfiguracją początkową, mamy wówczas następujący ciąg tranzycji czasowych

$$\langle \mu P.a; P, t_0 \rangle \xrightarrow{\langle t_1, a \rangle} \langle \mu P.\delta(u); P, t_1 \rangle \xrightarrow{\langle t_1+u, i \rangle} \langle \mu P.P, t_1 + u \rangle$$

po którym następuje powtórzenie tego ciągu z przesunięciem czasowym. Oczywiście $t_0 \leq t_1 < t_1 + u$ oraz $u \in \Delta(a)$.

Przykład 2

Dla konfiguracji początkowej

$$\langle a; B_1 + \delta(u); B_2, t_0 \rangle$$

możliwe są dwa ciągi tranzycji

$$\langle a; B_1 + \delta(u); B_2, t_0 \rangle \xrightarrow{\langle t_1, a \rangle} \langle \delta(u_1); B_1, t_1 \rangle \xrightarrow{\langle t_1+u_1, i \rangle} \langle B_1, t_1 + u_1 \rangle$$

gdzie $t_1 < u$, oraz

$$\langle a; B_1 + \delta(u); B_2, t_0 \rangle \xrightarrow{\langle t_0+u, i \rangle} \langle B_2, t_0 + u \rangle$$

Analizowane wyrażenie behawioralne

$$a; B_1 + \delta(u); B_2$$

ma interesującą własność, reprezentuje ono mianowicie mechanizm przeterminowania. Składowa $\delta(u); B_2$ wyznacza okres przeterminowania o długości u . Jeżeli w odcinku czasu u od momentu osiągnięcia konfiguracji $\langle a; B_1 + \delta(u); B_2, t_0 \rangle$ nie zostanie zainicjowane wykonanie akcji a , to w chwili $t_0 + u$ nastąpi przejście do konfiguracji $\langle B_2, t_0 + u \rangle$, a tym samym zostanie wykluczona możliwość wykonania wyrażenia $a; B_1$.

Przykład 3

Rozpatrzmy teraz konfigurację zawierającą wyrażenie behawioralne ze złożeniem równoległym

$$\langle a; B_1 + \delta(u_2); B_2 \parallel [\{a\}] a; B_3 + b; B_4, t_0 \rangle$$

Dla pierwszej składowej złożenia równoległego, zgodnie z przykładem 2., są możliwe dwa przejścia. Podobnie dwa przejścia są możliwe dla drugiej składowej. Dla całego natomiast wyrażenia złożenia równoległego jest możliwe tylko jedno przejście

$$\langle a; B_1 + \delta(u_2); B_2 \mid [\{a\}] \mid \bar{a}; B_3 + b; B_4, t_0 \rangle \xrightarrow{\langle t_0, a \rangle} \langle \delta(u); B_1 \mid [\{a\}] \mid \delta(u); B_3, t_0 \rangle$$

Wynika to z faktu, że para akcji a raz $\bar{a}$ synchronizuje się w chwili t_0 , podczas gdy pozostałe akcje mogą zajść dopiero później.

Kolejne przejścia są postaci:

$$\langle \delta(u); B_1 \mid [\{a\}] \mid \delta(u); B_3, t_0 \rangle \xrightarrow{\langle t_0 + u, i \rangle}$$

$$\langle B_1 \mid [\{a\}] \mid \delta(u); B_3, t_0 + u \rangle \xrightarrow{\langle t_0 + u, i \rangle}$$

$$\langle B_1 \mid [\{a\}] \mid B_3, t_0 + u \rangle$$

Specyfiką tych przejść jest dwukrotne wystąpienie akcji wewnętrznej $\langle t_0 + u, i \rangle$, oddzielnie w każdej z równoległych składowych wyrażenia behawioralnego. Wynika to z przyjętych zasad reprezentacji czasowych akcji komunikacyjnych. Ze względu na przeplutowy charakter semantyki kolejność wystąpienia tych akcji jest oczywiście dowolna.

Przykład 4

Rozpatrzmy teraz trzy przypadki konfiguracji z wyrażeniem przesłaniania. Pierwsza konfiguracja ma postać

$$\langle (a; B_1 \mid [\{a\}] \mid a; B_2 + b; B_3) \setminus \{a\}, t \rangle$$

Na podstawie reguł dla restrykcji możemy wyprowadzić dwa przejścia:

$$\langle (a; B_1 \mid [\{a\}] \mid a; B_2 + b; B_3) \setminus \{a\}, t \rangle \xrightarrow{\langle t, i \rangle} \langle (\delta(u); B_1 \mid [\{a\}] \mid \delta(u); B_2) \setminus \{a\}, t \rangle$$

dla $u \in \Delta(a)$, oraz

$$\langle (a; B_1 \mid [\{a\}] \mid a; B_2 + b; B_3) \setminus \{a\}, t \rangle \xrightarrow{\langle t, b \rangle} \langle (a; B_1 \mid [\{a\}] \mid \delta(u); B_3) \setminus \{a\}, t \rangle$$

dla $u \in \Delta(b)$.

Druga konfiguracja ma postać

$$\langle (a; B_1 \mid [\{a\}] \mid a; B_2 + b; B_3) \setminus \{b\}, t \rangle$$

Różni się ona od pierwszej konfiguracji tylko przesłanianą akcją. Dla konfiguracji tej jest możliwe tylko jedno przejście

$$\langle (a; B_1 \mid [\{a\}] \mid a; B_2 + b; B_3) \setminus \{b\}, t \rangle \xrightarrow{\langle l, a \rangle} \langle (\delta(u); B_1 \mid [\{a\}] \mid \delta(u); B_2) \setminus \{b\}, t \rangle$$

Nie jest natomiast możliwe przejście związane z wykonaniem akcji b . Przejście to nie jest możliwe, gdyż – zgodnie z przyjętą zasadą komunikacji – w jej realizacji muszą uczestniczyć dwaj partnerzy. Proces $a; B_1 \mid [\{a\}] \mid a; B_2 + b; B_3$ jest gotowy do wykonania akcji a oraz b , przy czym możliwość wykonania akcji a nie zależy od otoczenia tego procesu. Proces $(a; B_1 \mid [\{a\}] \mid a; B_2 + b; B_3) \setminus \{b\}$ nie może natomiast wykonać akcji b , gdyż – z powodu restrykcji – nie ma możliwości komunikacji ze swym otoczeniem na bramce b .

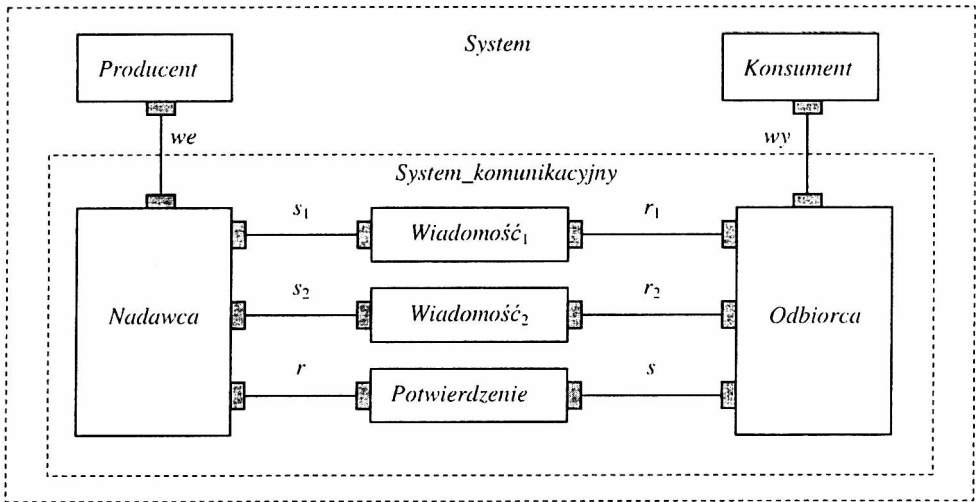
Wreszcie trzecia konfiguracja ma postać

$$\langle (a; B) \setminus \{a\}, t \rangle$$

Z poprzedniego wyjaśnienia wynika, że nie ma dla niej żadnej możliwości przejścia do innej konfiguracji, co oznacza, że konfiguracja wyraża sytuację blokady (zakleszczenia).

Przykład 5

Przykład dotyczy systemu komunikujących się zdalnie agentów reprezentowanego diagramem na rysunku 3.1.



Rys. 3.1. Struktura przykładowego systemu

Agent *Producent* wysyła wiadomości do agenta *Konsument* za pośrednictwem agenta *System komunikacyjny*. Po wysłaniu wiadomości *Producent* oczekuje na potwierdzenie jej odbioru przez *Konsumenta*, a po otrzymaniu tego potwierdzenia wysyła kolejną wiadomość. *System komunikacyjny* nie jest jednak niez-

wodny, gdyż linie komunikacyjne, które wchodzą w jego skład, mogą przekazywane od nadawcy wiadomości przesyłać do odbiorcy bez zniekształceń albo je gubić.

System komunikacyjny składa się z *Nadawcy* i *Odbiorcy*, połączonych dwiema liniami – *Wiadomość₁* oraz *Wiadomość₂* – do przesyłania wiadomości i jedną linią – *Potwierdzenie* – do przekazywania potwierdzeń.

Wspólnym modelem każdej z linii jest proces postaci

$$Linia =_{\text{def}} \mu Linia.s; (r; Linia + \delta(u_L); Linia)$$

gdzie $u_L > 0$ jest stałą charakteryzującą linię. Konkretnie linie są zdefiniowane następująco jako wywołania procesów *Linia*:

$$Potwierdzenie =_{\text{def}} Linia$$

$$Wiadomość_1 =_{\text{def}} Linia[s_1/s, r_1/r],$$

$$Wiadomość_2 =_{\text{def}} Linia[s_2/s, r_2/r],$$

Nadawca odbiera wiadomość od *Producenta* i przesyła ją najpierw linią *Wiadomość₁*, a następnie oczekuje na potwierdzenie odbioru wiadomości przez *Odbiorcę* na linii *Potwierdzenie*. Jeśli potwierdzenie otrzyma, powtarza działanie, pobierając od *Producenta* kolejną wiadomość. Jeżeli natomiast nie otrzyma potwierdzenia w określonym czasie, to ponownie przesyła tę samą wiadomość linią *Wiadomość₂* i znów oczekuje na potwierdzenie jej odbioru na linii *Potwierdzenie*. Jeśli potwierdzenie takie otrzyma, rozpoczyna ponownie cykl pracy, pobierając od *Producenta* kolejną wiadomość, a w przypadku przeciwnym powtarza wielokrotnie wysłanie tej samej wiadomości linią *Wiadomość₂*, aż do otrzymania potwierdzenia jej odbioru linią *Potwierdzenie*.

Proces *Nadawca* jest zdefiniowany jako

$$Nadawca =_{\text{def}} \mu N.we; s_1; (r; N + \delta(u_N); N')$$

gdzie

$$N' =_{\text{def}} \mu N'.s_2; (r; N + \delta(u_N); N'),$$

Odbiorca oczekuje na wiadomość z linii *Wiadomość₁*. Po jej otrzymaniu wysyła potwierdzenie jej odbioru linią *Potwierdzenie*, a odebraną wiadomość przekazuje do *Konsumenta*. Następnie oczekuje nadejścia wiadomości na linii *Wiadomość₁* lub *Wiadomość₂*. Otrzymanie wiadomości na linii *Wiadomość₁* oznacza, że *Nadawca* po otrzymaniu potwierdzenia poprzednio wysłanej wiadomości przesyła kolejną, nową wiadomość. Otrzymanie natomiast wiadomości na linii *Wiadomość₂* oznacza, że *Nadawca* powtarza wysyłanie wiadomości, gdyż nie otrzymał potwierdzenia jej odbioru. W tym przypadku *Odbiorca* ponownie przesyła potwierdzenie odbioru wiadomości, ale nie przesyła jej już do *Konsumenta*.

Proces *Odbiorca* jest zdefiniowany następująco:

$$Odbiorca =_{\text{def}} \mu O. r_1; s; wy; O'$$

gdzie

$$O' =_{\text{def}} \mu O'. (O + r_2; s; O')$$

System komunikacyjny jest równoległym złożeniem procesów: *Nadawca*, *Odbiorca*, *Wiadomość₁*, *Wiadomość₂* i *Potwierdzenie*.

$$\text{System komunikacyjny} =_{\text{def}}$$

(*Nadawca*

$$[[\{s_1, s_2, r\}]]$$

(*Wiadomość₁* $[[\emptyset]]$ *Wiadomość₁* $[[\emptyset]]$ *Potwierdzenie*)

$$[[\{r_1, r_2, s\}]]$$

Odbiorca) \ {*r*, *r₁*, *r₂*, *s*, *s₁*, *s₂*}.

I wreszcie, cały system jest zdefiniowany jako wyrażenie behawioralne, stanowiące złożenie równoległe procesów *Producent*, *Konsument* i *System komunikacyjny*:

$$\text{System} =_{\text{def}}$$

$$(\text{Producent} [[\{we\}]] \text{System komunikacyjny} [[\{wy\}]] \text{Konsument}) \setminus \{we, wy\}$$

3.7. Uwagi końcowe

W tym rozdziale przedstawiono dwa czasowe rozszerzenia języka CCS. Pierwsze z nich – język TCCS – jest rozszerzeniem polegającym na wprowadzeniu akcji natychmiastowych i opóźniających, drugie – język RTCCS – na wprowadzeniu wyłączanie akcji czasowych wraz z niedeterministycznym mechanizmem wyznaczania czasu realizacji akcji komplementarnych.

Skupiono się na semantyce języków. Wprowadzenie czasu do języka znacznie komplikuje definicję jego semantyki – nie daje się stosować bez ograniczeń semantyki przepływowej. Złożoność definicji wynika z konieczności uwzględnienia upływu czasu i lokowania w czasie zachodzących zdarzeń.

Ograniczono się do przedstawienia przykładów zastosowania tylko języka RTCCS, gdyż przykłady ilustrujące język TCCS, a także innych, skojarzonych z nim języków, można znaleźć w literaturze, na przykład: [Moller, Tofts 1989], [Tofts 1989], [Fencott 1996].

Warto przypomnieć, że równoległe do prac nad językiem CCS i jego czasowymi rozszerzeniami, były prowadzone prace nad czasowymi rozszerzeniami języka CSP [Hoare 1985], [Roscoe 1998], na przykład: [Reed, Roscoe 1986], [Wang 1991], [Reed, Roscoe 1999], oraz dla języka ACP [Baeten, Bergstra, Smolka 1995], na przykład: [Groote 1990], [Baeten, Bergstra 1991], [Schneider 2004].

Przykład czasowego rozszerzenia języka CSP, nazywanego RTCSP, wraz z systemem specyfikowania i dowodzenia poprawności, przedstawiono w monografii [Huzar 1989].

4. Abstrakcyjne typy danych w LOTOSie

4.1. Podstawowe pojęcia algebraiczne

Elementarny typ danych w znaczeniu wąskim jest utożsamiany z pewnym zbiorem wartości. Elementarny typ danych w znaczeniu szerokim, które jest tu używane, jest zbiorem wartości wraz ze zbiorem operacji na tych wartościach. Matematycznym modelem elementarnych typów są jednorodziejowe algebry abstrakcyjne. Mając do dyspozycji pewien repertuar typów elementarnych, można tworzyć nowe, złożone typy – ich modelem matematycznym są wielorodziejowe algebry abstrakcyjne.

LOTOS wykorzystuje algebraiczne podejście do definiowania abstrakcyjnych typów danych. Z tego względu poniżej przypomina się podstawowe pojęcia algebraiczne. Przegląd tych pojęć oparto na książce [Ehrig, Mahr 1985]. Informacje z tego zakresu można znaleźć także w innych pozycjach, na przykład [Tiurny 2003].

Definiowanie algebry abstrakcyjnej rozpoczyna się od opisu jej struktury, wyrażonej przez sygnaturę.

Sygnaturą algebry nazywa się parę

$$Sig =_{\text{def}} \langle S, OP \rangle \quad (4.1)$$

gdzie:

S jest niepustym zbiorem rodzajów, czyli identyfikatorów (nazw) nośników algebry,
 OP jest zbiorem deklaracji operacji.

Deklaracja operacji będzie zapisywana w postaci

$$op : s_1 s_2 \dots s_n \rightarrow s \quad (4.2)$$

gdzie:

op jest identyfikatorem (nazwą) operacji,
 $s_1 s_2 \dots s_n$ jest listą, której elementy $s_1, s_2, \dots, s_n \in S$ są identyfikatorami (nazwami) rodzajów argumentów,
 $s \in S$ jest identyfikatorem (nazwą) rodzaju wartości operacji.

Deklaracja operacji o nazwie op wskazuje na nazwy zbiorów jej argumentów i nazwę zbioru jej wartości. Jeżeli op jest operacją zeroargumentową, czyli stałą, to jej deklaracja ma postać

$$op : \rightarrow s \quad (4.3)$$

Zakłada się, że każda deklaracja operacji ma różną nazwę operacji, dlatego dalej, zamiast pisać $(op : s_1 s_2 \dots s_n \rightarrow s) \in OP$, będzie się pisać krótko $op \in OP$.

Może być wiele algebr mających tę samą sygnaturę. Konkretna algebra nad daną sygnaturą może być uważana za interpretację sygnatury.

Algebrą nad sygnaturą Sig , krótko *Sig-algebrą*, nazywa się parę

$$ALG =_{\text{def}} \langle A, F \rangle \quad (4.4)$$

gdzie:

$A =_{\text{def}} \{A_s \mid s \in S\}$ jest rodziną zbiorów zwanych *nośnikami* lub *dziedzinaми* algebry,
 $F =_{\text{def}} \{f_{op} \mid op \in OP\}$ jest rodziną funkcji zwanych *operacjami* algebry, przy czym
 każdej deklaracji operacji $op \in OP$

$$op : s_1 s_2 \dots s_n \rightarrow s \quad (4.5)$$

odpowiada funkcja

$$f_{op} : A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s \quad (4.6)$$

Czasem jest wygodnie wyróżnić stałe (operacje zeroargumentowe) od pozostałych operacji, wówczas algebrę zapisuje się w postaci

$$ALG =_{\text{def}} \langle \{A_1, \dots, A_k\}, \{c_1, \dots, c_m\} \cup \{f_1, \dots, f_n\} \rangle \quad (4.7)$$

gdzie:

$\{A_1, \dots, A_k\}$, dla $k \in \text{Nat} \setminus \{0\}$, jest rodziną nośników algebry,
 $\{c_1, \dots, c_m\}$, dla $m \in \text{Nat}$, jest zbiorem operacji zeroargumentowych,
 $\{f_1, \dots, f_n\}$, dla $n \in \text{Nat} \setminus \{0\}$, jest zbiorem pozostałych operacji.

Dwie algebry o tej samej sygnaturze Sig nazywa się algebrami *podobnymi*. Zbiór wszystkich algebr podobnych nazywamy klasą *Sig-algebr*.

Jeżeli rodzina A zawiera tylko jeden nośnik, to mówimy o algebrze jednorodziejowej, a w przeciwnym razie o algebrze wielorodziejowej.

Rodzina podzbiorów $\{A_s^0 \mid s \in S\}$ takich, że $A_s^0 \subseteq A_s$ dla $s \in S$, wyznacza *podalgebrę* ALG^0 , gdy zbiory te są zamknięte ze względu na wszystkie operacje algebry ALG .

Jeżeli rodziny $\{A_s^0 \mid s \in S\}$ oraz $\{A_s^1 \mid s \in S\}$ wyznaczają dwie podalgebry ALG^0 oraz ALG^1 algebry ALG , to rodzina ich iloczynów $\{A_s^0 \cap A_s^1 \mid s \in S\}$ wyznacza również podalgebrę algebry ALG . Ogólniej – iloczyn dowolnego zbioru podalgebr danej algebry jest także podalgebrą tej algebry.

Dla dowolnie przyjętej rodziny zbiorów $\{A_s^0 \mid s \in S\}$ istnieje najmniejsza podalgebra zawierająca tę rodzinę.

Homomorfizmem algebry $ALG_A = \langle \{A_s \mid s \in S\}, \{f_{op} \mid op \in OP\} \rangle$ w algebrę podobną $ALG_B = \langle \{B_s \mid s \in S\}, \{g_{op} \mid op \in OP\} \rangle$ nazywamy zbiór takich odwzorowań

$$H =_{\text{def}} \{h_s : A_s \rightarrow B_s \mid s \in S\} \quad (4.8)$$

że dla każdego działania $f_{op} : A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$ dla $op \in OP$ i dowolnych argumentów $a_1 \in A_{s_1}, \dots, a_n \in A_{s_n}$ zachodzi

$$h_s(f_{op}(a_1, \dots, a_n)) = g_{op}(h_{s_1}(a_1), \dots, h_{s_n}(a_n)) \quad (4.9)$$

Homomorfizm algebr zapisuje się w postaci

$$H : ALG_A \rightarrow ALG_B \quad (4.10)$$

Algebrę ALG nazywa się *algebrą wolną* w klasie Sig -algebr, jeśli istnieje dla niej taka rodzina zbiorów generatorów $\{A_s^0 \mid s \in S\}$ o takiej własności, że każda rodzina odwzorowań $\{f_s : A_s^0 \rightarrow B_s \mid s \in S\}$, gdzie $\{B_s \mid s \in S\}$ jest rodziną nośników dowolnej Sig -algebry, daje się rozszerzyć do homomorfizmu $\{h_s : A_s \rightarrow B_s \mid s \in S\}$. Zbiór $\{A_s^0 \mid s \in S\}$ nazywa się *zbiorem generatorów wolnych*.

4.2. Algebra termów

Z każdą algebrą jest związany pewien zbiór napisów, które powstają ze złożenia symboli stałych, zmiennych i operacji algebry. Zbiór ten nazywa się *zbiorem termów* i jest definiowany następująco:

Niech V_s będzie zbiorem zmiennych rodzaju A_s dla $s \in S$, co będzie zapisywane w postaci $v : A_s$.

Dalej, zamiast pisać rodzaj A_s , będzie się pisać krótko rodzaj s .

Zbiorem wszystkich zmiennych jest

$$V = \bigcup_{s \in S} V_s \quad (4.11)$$

Działania zeroargumentowe, czyli stałe, także mają swój rodzaj. Stała $c : \rightarrow A_s$ jest rodzaju s , czyli jest elementem zbioru A_s dla $s \in S$.

Zbiór termów rodzaju s , dla $s \in S$, dla algebry wielorodzajowej ALG nad zbiorem zmiennych V , oznaczany $Term_s(V)$, jest zdefiniowany rekursywnie w sposób następujący:

- jeżeli $c : \rightarrow A_s$, to $c \in Term_s(V)$,
- $V_s \subseteq Term_s(V)$,
- jeżeli napisy $t_1, \dots, t_k$ są termami rodzajów $s_1, \dots, s_k$ oraz

$$f : A_{s_1} \times \dots \times A_{s_k} \rightarrow A_s \quad (4.12)$$

jest działaniem k -argumentowym, to napis postaci $f(t_1, \dots, t_n)$ jest termem rodzaju s , czyli $f(t_1, \dots, t_n) \in Term_s(V)$.

Zbiór wszystkich termów dla algebry wielorodzajowej ALG nad zbiorem zmiennych V , oznaczany $Term(V)$, jest określony jako mnogościowa suma

$$Term(V) = \bigcup_{s \in S} Term_s(V) \quad (4.13)$$

Zbiór termów nad pustym zbiorem zmiennych, czyli $Term(\emptyset)$, nazywa się zbiorem termów *stałych*.

Zauważmy, że zbiór termów jest wyznaczony jednoznacznie przez samą sygnaturę algebry, algebry podobne mają więc te same zbiory termów.

Zbiór termów $Term(V)$ nad ustalonym zbiorem zmiennych V , generowany przez algebrę $ALG = \langle A, F \rangle$ o sygnaturze $Sig = \langle S, OP \rangle$, może być podstawą do utworzenia nowej algebry wielorodzajowej, zwanej *algebrą termów*, która jest podobna do algebry ALG .

Algebrę termów

$$ALG_{Term} =_{\text{def}} \langle A_{Term}, F_{Term} \rangle \quad (4.14)$$

dla algebry ALG definiuje się następująco:

$A_{Term} = \{Term_s(V) \mid s \in S\}$ jest rodziną nośników algebry termów,

$F_{Term} = \{f_{op} \mid op \in OP\}$ jest zbiorem operacji algebry termów, przy czym operacja f_{op} ma sygnaturę

$$f_{op} : Term_{s_1}(V) \times \dots \times Term_{s_n}(V) \rightarrow Term_s(V)$$

gdy deklaracja operacji ma postać: $op : s_1 s_2 \dots s_n \rightarrow s$, ($n \geq 0$) i jest zdefiniowana następująco:

$$\text{jeżeli } t_j \in Term_{s_j}(V), \text{ dla } j = 1, \dots, n, \text{ to } f_{op}(t_1, \dots, t_n) =_{\text{def}} op(t_1, \dots, t_n).$$

Każdemu nośnikowi A_s i każdej operacji op w algebrze ALG odpowiadają nośnik $Term_s(V)$ i operacja f_{op} w algebrze termów ALG_{Term} .

Algebra termów w klasie Sig -algebr wyróżnia się tym, że jest algebrą wolną.

4.3. Algebra ilorazowa termów

Termy mają interpretację, która – w zależności od wartościowania zmiennych – przypisuje im wartość odpowiedniego rodzaju.

Wartościowanie zmiennych jest wyrażane przez funkcję ν o sygnaturze

$$\nu : V \rightarrow A \quad (4.15)$$

gdzie:

$$V = \bigcup_{s \in S} V_s \quad \text{oraz} \quad A = \bigcup_{s \in S} A_s$$

przy czym zmiennej $v : V_s$ można przyporządkowywać wartości tylko ze zbioru A_s .

Niech $WAR_\nu(t)$ oznacza wartość termu t przy wartościowaniu ν . Funkcję WAR_ν , obliczającą wartości termów przy wartościowaniu ν , można zdefiniować rekursywnie w sposób następujący:

- jeżeli term jest postaci x , gdzie x jest zmienną, czyli $x \in V$, to $WAR_v(x) = v(x)$,
- jeżeli term jest postaci c , gdzie c jest stałą, to $WAR_v(c) = c$,
- jeżeli term jest postaci $f(t_1, \dots, t_k)$, gdzie f jest k -argumentowym działaniem, a $t_1, \dots, t_k$ są termami, to $WAR_v(f(t_1, \dots, t_k)) = f(WAR_v(t_1), \dots, WAR_v(t_k))$.

Zauważmy, że wartość termu stałego t nie zależy od wartościowania v . Dla termu stałego i dowolnych dwóch wartościowań v oraz v' zachodzi zatem

$$WAR_v(t) = WAR_{v'}(t). \quad (4.16)$$

Niech $WAR(t)$ oznacza wartość termu stałego t . Na zbiorze termów stałych rodzaju s definiuje się relację binarną $=_s$, określoną następująco:

jeżeli $t_1, t_2 \in Term_s(\emptyset)$, to $t_1 =_s t_2$ wtedy i tylko wtedy, gdy $WAR(t_1) = WAR(t_2)$.

Relacja $=_s$ jest oczywiście relacją równoważności i wyznacza podział zbioru termów stałych rodzaju s na klasy abstrakcji. Do jednej klasy abstrakcji należą wszystkie termy tego samego rodzaju, które reprezentują tę samą wartość. Przez $Term_s(\emptyset)/=_s$ będziemy oznaczać zbiór ilorazowy termów wyznaczony przez relację równoważności $=_s$.

Relacja równoważności $=_s$ ma następującą własność:

jeżeli t_{s_j}, t'_{s_j} są termami rodzaju s_j oraz $t_{s_j} =_{s_j} t'_{s_j}$, dla $j = 1, \dots, n$,

$$\text{to } f_{op}(t_{s_1}, \dots, t_{s_n}) =_s f_{op}(t'_{s_1}, \dots, t'_{s_n}) \text{ dla } op \in OP.$$

Rodzinę relacji równoważności $\{=_s \mid s \in S\}$ nazywa się *kongruencją*. Kongruencja jest podstawą do zdefiniowania algebry, nazywanej *ilorazową algebrą termów stałych* dla algebry ALG_{Term} .

Definicja ilorazowej algebry termów, $\overline{ALG_{Term}}$ jest następująca:

$$\overline{ALG_{Term}} =_{\text{def}} \langle \overline{A_{Term}}, \overline{F_{Term}} \rangle \quad (4.17)$$

gdzie:

$\overline{A_{Term}} = \{ Term_s(\emptyset)/=_s \mid s \in S \}$ jest rodziną zbiorów ilorazowych termów rodzajów $s \in S$;

$\overline{F_{Term}} = \{ \overline{f_{op}} \mid op \in OP \}$ jest zbiorem operacji ilorazowej algebry termów, przy czym operacja $\overline{f_{op}}$ ma sygnaturę

$$\overline{f_{op}} : Term_{s_1}(\emptyset)/=_s \times \dots \times Term_{s_n}(\emptyset)/=_s \rightarrow Term_s(\emptyset)/=_s \quad (4.18)$$

gdy deklaracja operacji ma postać: $op : s_1 s_2 \dots s_n \rightarrow s$, i jest zdefiniowana następująco:

jeżeli $[t_j] \in Term_{s_j}(\emptyset)/=_s$, dla $j = 1, \dots, n$, to $\overline{f_{op}}([t_1], \dots, [t_n]) =_{\text{def}} [f_{op}(t_1, \dots, t_n)]$;

$[t_j]$ oznacza klasę abstrakcji w zbiorze ilorazowym $Term_{s_j}(\emptyset)/=_s$, generowaną przez term t_j .

Algebra termów jest homomorficzna względem ilorazowej algebry termów, to znaczy istnieje homomorfizm $H : \overline{ALG_{Term}} \rightarrow \overline{ALG_{Term}}$.

4.4. Specyfikacja równościowa typów abstrakcyjnych

Typy danych w LOTOSie specyfikuje się przez podanie sygnatury algebry oraz własności jej operacji. Semantyką takiej specyfikacji jest pewna konkretna algebra – ilorazowa algebra termów generowana przez specyfikację. W bieżącym podrozdziale przedstawiono nieformalnie składnię specyfikacji typów danych, a jej semantykę – w następnym podrozdziale.

Specyfiką definicji typów w LOTOSie jest równościowe definiowanie własności operacji. Jest to podejście algebraiczne, inne od podejścia spotykanego w większości języków specyfikacji, a zwłaszcza programowania polegającego na definiowaniu operacji jako procedur.

W języku LOTOS podstawowa forma specyfikacji typu danych ma następującą strukturę:

```

type Nazwa_typu is
    sorts Lista_nazw-rodzajów
    opns Lista_symboli_operacji
    eqns Lista_równości
endtype

```

(4.19)

Nazwa_typu jest identyfikatorem. Listy są ciągami odpowiednich elementów oddzielonych separatorami.

Lista_nazw-rodzajów jest ciągiem identyfikatorów rodzajów oddzielanych przecinkami, a *Lista_symboli_operacji* – ciągiem symboli operacji oddzielanych wierszami. Obie te listy wyznaczają sygnaturę algebry.

Na przykład specyfikacja:

```

type NatLog is
    sorts   nat, log
    opns   z      :  $\rightarrow nat$ 
            succ   :  $nat \rightarrow nat$ 
            pred   :  $nat \rightarrow nat$ 
            plus   :  $nat, nat \rightarrow nat$ 
            minus  :  $nat, nat \rightarrow nat$ 
            p      :  $\rightarrow log$ 
            neg    :  $log \rightarrow log$ 
            impl   :  $log, log \rightarrow log$ 

```

r : *nat*, *nat* -> *log*
mr : *nat*, *nat* -> *log*

endtype

określa sygnaturę algebry dwurodzajowej o nazwie *NatLog*, z nośnikami o nazwach *nat* i *log* oraz z operacjami: dwiema zeroargumentowymi o nazwach *z* i *p*, trzema jednoargumentowymi o nazwach *succ*, *pred* i *neg* oraz pięcioma dwuargumentowymi o nazwach *plus*, *minus*, *impl*, *r* i *mr*.

Sama sygnatura pozwala na wyznaczenie zbioru termów algebry. Dla typu *NatLog*, jeśli *x*, *y* są zmiennymi rodzaju *nat*, to termami rodzaju *nat* są na przykład

succ(x), *z*, *plus(x,y)*, *plus(succ(z), succ(x))*.

Dwie pierwsze listy definiują sygnaturę algebry, trzecia natomiast lista *Lista_równości* definiuje własności wprowadzonych operacji. Własności te definiuje się rekursywnie za pomocą zbioru *równości*. Wyróżnia się dwa rodzaje równości: równość zwykłą i warunkową.

Równość zwykła ma postać

$$t_1 = t_2 \quad (4.20)$$

gdzie: t_1, t_2 są dowolnymi termami tego samego rodzaju, a symbol $=$ oznacza równość semantyczną termów. Jeżeli termy są tego samego rodzaju *s*, to symbol równości $=$ oznacza to samo, co poprzednio symbol $=_s$. Równość zwykła jest takiego rodzaju jak jej termy składowe.

Równość warunkowa ma postać

$$e_1, \dots, e_n \Rightarrow e \quad (4.21)$$

gdzie $e_1, \dots, e_n, e$ są zwykłymi równościami. Równość warunkowa jest takiego samego rodzaju jak rodzaj równości *e*.

Przykładowa *Lista_równości*, stanowiąca uzupełnienie podanej wyżej sygnatury, ma postać

```
type NatLog is
sorts  nat, log
opns   z      : -> nat
       succ   : nat -> nat
       pred   : nat -> nat
       plus   : nat, nat -> nat
       minus  : nat, nat -> nat
       p      : -> log
       neg    : log -> log
       impl   : log, log -> log
```

```

      r      : nat, nat -> log
      mr     : nat, nat -> log

eqns
  forall x : nat, y : nat, a : log, b : log
  ofsort nat
    plus(x, y) = plus(y, x) ;
    plus(succ(x), y) = succ(plus(x, y)) ;
    pred(z) = z ;
    pred(succ(x)) = x ;
    minus(x, z) = z ;
    minus(x, succ(y)) = pred(minus(x, y)) ;
  ofsort log
    neg(neg(b)) = b ;
    impl(p, p) = p ;
    impl(p, neg(p)) = neg(p) ;
    impl(neg(p), a) = p ;
    r(x, x) = p ;
    mr(z, x) = p ;
    mr(succ(x), succ(y)) = mr(x, y) ;

endtype

```

Dwie listy równości są poprzedzone wierszem **forall**, wprowadzającym po dwie zmienne rodzaju *nat* i *log*, oraz przedzielone wierszami **ofsort**, określającymi rodzaje występujących równości. Ogranicznikiem końca równości jest tu średnik. W przykładzie występują tylko zwykłe równości, następny przykład specyfikacji wykorzystuje także równości warunkowe

```

type NatX is
  sorts   nat
  opns    z : -> nat
          succ : nat : -> nat
          plus : nat nat -> nat

eqns
  forall x : nat, y : nat, t : nat
  ofsort nat
    plus(x, z) = x ;
    plus(x, y) = plus(y, x) ;
    plus(x, y) = t => plus(succ(x), y) = succ(t) ;

endtype

```

Równości definiują własności algebry, a dokładniej własności jej operacji.

4.5. Semantyka specyfikacji typów

W abstrakcyjnym ujęciu specyfikacja typu jest widziana jako trójka

$$TSpec =_{\text{def}} \langle S, OP, Eq \rangle \quad (4.22)$$

gdzie: $\langle S, OP \rangle$ jest sygnaturą, a Eq jest zbiorem równości.

Specyfikacja typu określa zatem sygnaturę i własności algebry, ale nie wyznacza jawnie konkretnej algebry. Algebr o danej sygnaturze może być nieskończenie wiele, podobnie nieskończenie wiele może być algebr o wskazanych własnościach. Pośród wszystkich algebr o danej sygnaturze i danych własnościach wskazuje się na szczególny rodzaj algebr, zwanych algebrami *początkowymi*. Charakteryzując je nieformalnie, można stwierdzić, że są to te algebry, które mają wszystkie wskazane własności i tylko te własności, które są wyrażone odpowiednimi równościami – inaczej: są to algebry, które nie spełniają żadnych dodatkowych równości. Dwie różne algebry początkowe są izomorficzne. Standardowo spośród wszystkich algebr początkowych wybiera się *ilorazową algebrę termów*. Algebra ta jest generowana przez specyfikację typu danych. Przedstawione niżej rozważania pokazują sposób generacji ilorazowej algebry termów na podstawie analizy specyfikacji.

Zbiór równości Eq , składających się na specyfikację typu danych $TSpec$, wyznacza system dowodowy

$$D_{TSpec} = \langle GrEq_{TSpec}, AxEq_{TSpec}, InfEq_{TSpec} \rangle \quad (4.23)$$

gdzie: $GrEq_{TSpec}$ jest pewnym zbiorem równości stałych, $AxEq_{TSpec}$ – zbiorem równości aksjomatów, $InfEq_{TSpec}$ – zbiorem reguł wnioskowania. Definicje poszczególnych elementów są następujące:

Niech dane będą termy t_1 i t oraz zmienna x . Notacja $t_1[x ::= t]$ oznacza modyfikację termu t_1 , polegającą na tekstowym zastąpieniu każdego wystąpienia zmiennej x w termie t_1 przez term t . Dalej będziemy rozpatrywać tylko takie termy t , które są termami stałymi, czyli $t \in Term_{TSpec}(\emptyset)$.

Jeżeli $e \in Eq$ jest zwykłą równością postaci $t_1 = t_2$ oraz $t \in Term_{TSpec}(\emptyset)$, to równość

$$t_1[x ::= t] = t_2[x ::= t] \quad (4.24)$$

jest *instancją* równości e i oznaczamy ją $e[x ::= t]$.

Jeżeli $e \in Eq$ jest warunkową równością postaci $e_1, \dots, e_n \Rightarrow e$ oraz $t \in Term_{TSpec}(\emptyset)$, to instancją tej równości jest równość warunkowa postaci

$$e_1[x ::= t], \dots, e_n[x ::= t] \Rightarrow e[x ::= t] \quad (4.25)$$

Jeżeli instancja równości nie zawiera zmiennych, to nazywamy ją *instancją stałą*.

$GrEq_{TSpec}$ jest zbiorem wszystkich stałych instancji zwykłych równości występujących w specyfikacji $TSpec$.

Jeżeli $e \in Eq$ jest zwykłą równością, to $AxEq(e)$ oznacza zbiór wszystkich stałych instancji równości e , jeżeli natomiast $e \in Eq$ jest równością warunkową, to $AxEq(e)$ jest zbiorem pustym. Przez ID oznaczmy zbiór wszystkich równości stałych postaci $t = t$, wówczas

$$AxEq_{TSpec} = \bigcup_{e \in Eq} AxEq(e) \cup ID \quad (4.26)$$

Jeżeli $e \in Eq$ jest zwykłą równością, to $InfEq(e)$ jest zbiorem pustym, jeżeli natomiast $e \in Eq$ jest równością warunkową postaci $e_1, \dots, e_n \Rightarrow e$, to $InfEq(e)$ oznacza zbiór reguł wnioskowania postaci

$$\frac{e'_1, \dots, e'_n}{e'} \quad (4.27)$$

gdzie $e'_1, \dots, e'_n, e'$ są stałymi instancjami równości $e_1, \dots, e_n, e$ osiągniętymi przez podstawienie za zbiór zmiennych V występujących w tych równościach termów stałych $Term_{TSpec}(\emptyset)$.

Przez Inf oznaczmy zbiór reguł wnioskowania postaci

$$\frac{t_1 = t_2}{t_2 = t_1} \quad (4.28)$$

dla wszystkich termów stałych t_1, t_2

$$\frac{t_1 = t_2, t_2 = t_3}{t_1 = t_3} \quad (4.29)$$

dla wszystkich termów stałych t_1, t_2, t_3

$$\frac{t_1 = t'_1, \dots, t_n = t'_n}{op(t_1, \dots, t_n) = op(t'_1, \dots, t'_n)} \quad (4.30)$$

dla wszystkich operacji $op \in OP$ z $n > 0$ i dla wszystkich termów stałych t_i, t'_i rodzaju s_i dla $i = 1, \dots, n$, wówczas

$$InfEq_{TSpec} = \bigcup_{e \in Eq} InfEq(e) \cup Inf \quad (4.31)$$

Dwa termy stałe t_1 i t_2 nazywa się termami *kongruentnymi* względem specyfikacji $TSpec$, co zapisujemy w postaci $t_1 =_{TSpec} t_2$, wtedy i tylko wtedy, gdy $D \vdash t_1 = t_2$, czyli gdy w systemie dowodzenia D istnieje dowód równości $t_1 = t_2$.

Relacja kongruencji wyznacza podział zbioru termów stałych $Term_{TSpec}(\emptyset)$ na klasy abstrakcji. Klasa abstrakcji generowana przez term t jest określona jako zbiór

$$[t] =_{\text{def}} \{t' \mid t' =_{TSpec} t\} \quad (4.32)$$

Zbiór klas abstrakcji termów stałych danego rodzaju s , oznaczany $Term_s(\emptyset)/=_{\bar{s}}$, wyznacza zbiór wartości rodzaju i ilorazowej algebry termów

$$A_{TSpec} =_{\text{def}} \langle A_{TSpec}, \bar{F}_{TSpec} \rangle \quad (4.33)$$

zdefiniowanej następująco:

- $A_{TSpec} = \{Term_s(\emptyset)/=_{\bar{s}} \mid s \in S\}$ jest rodziną zbiorów ilorazowych termów rodzajów $s \in S$,
- $F_{TSpec} = \{f_{op} \mid op \in OP\}$ jest zbiorem operacji ilorazowej algebry termów, przy czym operacja f_{op} ma sygnaturę

$$f_{op} : Term_{s_1}(\emptyset)/=_{\bar{s}_1} \times \dots \times Term_{s_n}(\emptyset)/=_{\bar{s}_n} \rightarrow Term_s(\emptyset)/=_{\bar{s}} \quad (4.34)$$

gdy deklaracja operacji ma postać: $op : s_1 s_2 \dots s_n \rightarrow s$ i jest zdefiniowana następująco: jeżeli $[t_j] \in Term_{s_j}(\emptyset)/=_{\bar{s}_j}$ dla $j = 1, \dots, n$, to

$$f_{op}([t_1], \dots, [t_n]) =_{\text{def}} [op(t_1, \dots, t_n)]. \quad (4.35)$$

4.6. Strukturalizacja specyfikacji

Podstawową techniką stosowaną podczas tworzenia złożonych specyfikacji typów danych jest ich strukturalizacja. Język LOTOS oferuje dwa zasadnicze mechanizmy strukturalizacji: rozszerzanie specyfikacji i specyfikacje parametryzowane (genericzne). Oba mechanizmy będą omówione nieformalnie na podstawie prostych przykładów.

Rozszerzanie specyfikacji polega na definiowaniu nowego typu danych jako rozszerzenia wcześniej zdefiniowanego typu. Rozpatrzmy przykład – załóżmy, że dana jest specyfikacja typu:

```

type Boolean is
sorts   Bool
opns   true, false      : -> Bool
        not              : Bool -> Bool
        _and_, _or_, _xor_, _implies_, _iff_ : Bool, Bool -> Bool
eqns
  forall  $x, y : Bool$ 
  ofsort Bool
    not(true) = false      ;
    not(false) = true     ;
     $x \text{ and } true = x$     ;
     $x \text{ and } false = false$  ;

```



```

x or true = true ;
x xor y = (x and not(y)) or (y and not(x)) ;
x implies y = y or not(x) ;
x iff y = (x implies y) and (y implies x) ;

```

endtype

Specyfikacja definiuje typ *Boolean* – jednorodzącą algebrę Boole’a – standardowy typ logiczny w większości języków programowania. Typ ten jest częścią składową wielu innych typów, na przykład liczb naturalnych. Symbole podkreśleń po lewej i prawej stronie operacji dwuargumentowych wskazują na użycie notacji wrostkowej (zamiast standardowej przedrostkowej).

Nowy typ *NatY* można zdefiniować, odwołując się do typu *Boolean*, w sposób następujący:

```

type NatY is Boolean
sorts   nat
opns    z : -> nat
          succ : nat : -> nat
          plus : nat nat -> nat
          eq : nat nat -> Bool
          gt : nat nat -> Bool

eqns
  forall x : nat, y : nat, t : nat
  ofsort nat
    plus(x, z) = x ;
    plus(x, y) = plus(y, x) ;
    plus(x, y) = t => plus(succ(x), y) = succ(t) ;
  ofsort Bool
    z eq z = true ;
    z eq succ(x) = false ;
    succ(x) eq z = false ;
    z gt z = false ;
    z gt succ(x) = true ;
    succ(x) gt succ(y) = x gt y ;

```

endtype

Przedstawiona specyfikacja jest równoważna następującej rozwiniętej tekstowo specyfikacji:

```

type NatY is
sorts   Bool, nat

```

```

opns   true, false      : -> Bool
        not              : Bool -> Bool
        _and_ , or_      : Bool, Bool -> Bool
        z : -> nat
        succ : nat : -> nat
        plus : nat nat -> nat
        eq : nat nat -> Bool
        gt : nat nat -> Bool

eqns
  forall x, y : Bool, x : nat, y : nat, t : nat
  ofsort Bool
    not(true) = false ;
    not(false) = true ;
    x and true = x ;
    x and false = false ;
    x or true = true ;
    x or false = x ;
  ofsort nat
    plus(x, z) = x ;
    plus(x, y) = plus(y, x) ;
    plus(x, y) = t => plus(succ(x), y) = succ(t) ;
  ofsort Bool
    z eq z = true ;
    z eq succ(x) = false ;
    succ(x) eq z = false ;
    z gt z = false ;
    z gt succ(x) = true ;
    succ(x) gt succ(y) = x gt y ;

endtype

```

Specyfikacja w formie rozwiniętej, nazywanej też specyfikacją w formie *kanonicznej*, powstaje przez tekstowe połączenie specyfikacji typu *Boolean* z nowym tekstem. Takie połączenie tekstów daje jednoznaczny efekt, pod warunkiem unikalności nazw rodzajów i operacji.

Specyfikacje typów nazywamy specyfikacjami rozłącznymi, gdy ich sygnatury mają rozłączne zbiory rodzajów i operacji.

Gdy specyfikacje nie są rozłączne, należy dokonać odpowiedniego przemianowania nazw, do czego służy pomocnicza konstrukcja przemianowania:

```

type Stara_nazwa_typu is Nowa_nazwa_typu
  sortnames Stara_nazwa_rodzaju for Nowa_nazwa_rodzaju
  ....
  opnames Stara_nazwa_operacji for Nowa_nazwa_operacji
  ....
endtype

```

Konstrukcja wprowadza nowy typ o nowej nazwie, który od typu starego różni się tylko nazwami rodzajów i nazwami operacji.

W przypadku wielu złożonych typów okazuje się często, że ogólna struktura tworzących je elementów nie zależy od elementów składowych. Jest tak na przykład w przypadku tablic, plików, stosów czy kolejek, używa się wówczas typów sparametryzowanych, ich specyfikacja ma postać:

```

type Nazwa_typu is
  formalsorts Lista_nazw_rodzajów
  formalopns Lista_symboli_operacji
  formaleqns Lista_równości
  sorts Lista_nazw_rodzajów
  opns Lista_symboli_operacji
  eqns Lista_równości
endtype

```

Jest to rozszerzona tekstowo forma specyfikacji kanonicznej, wprowadzająca formalne parametry, którymi mogą być rodzaje, operacje i równości. Typ sparametryzowany staje się typem konkretnym po zastąpieniu jego parametrów formalnych parametrami aktualnymi. Służy do tego konstrukcja aktualizacji typu postaci:

```

type Nazwa-typu is Nazwa-typu-sparametryzowanego
actualizedby Nazwa-typu using
  Lista_podstawień_rodzajów
  Lista_podstawień_operacji
endtype

```

gdzie podstawienie_rodzaju ma postać:

```

sortnames nazwa_rodzaju for nazwa_rodzaju_formalnego

```

podstawienie_operacji ma postać:

```

opnames nazwa_operacji for nazwa_operacji_formalnej

```

Dobłą ilustracją obu tych konstrukcji jest specyfikacja stosu formalnego i jego uaktualnienia. Najpierw specyfikujemy stos jako typ sparametryzowany, niezależny od typu elementów, na których operuje:

```

type Stos is
  formalsorts    elem
  formalopns    e0 : -> elem
  sorts          stos
  opns
    pusty        : -> stos
    odczyt       : stos -> elem
    ze_stosu     : stos -> stos
    na_stos      : stos elem -> stos
  eqns
    forall e : elem, s : stos
      ofsort elem
        odczyt(pusty) = e0 ;
        odczyt(na_stos(s, e)) = e ;
      ofsort stos
        ze_stosu(pusty) = pusty ;
        ze_stosu(na_stos(s, e)) = s ;
  endtype

```

Rodzaj *elem* oraz stała *e0* są parametrami formalnymi typu *Stos*, przyporządkowanie im konkretnego niesparametryzowanego rodzaju oraz wskazanej wartości tego rodzaju pozwala na definicję konkretnego typu stosowego. Na przykład stos liczb naturalnych uzyskamy, podając specyfikację:

```

type StosNat is Stos actualizedby NatLog using
  sortnames nat for elem
  opnnames z    for e0
endtype

```

LOTOS definiuje pewien zestaw typów predefiniowanych, zestawionych w bibliotekę typów standardowych. Do specyfikacji tych typów można się odwoływać przez wskazanie nazwy typu, z zaznaczeniem, że jest to typ należący do biblioteki typów standardowych. Odwołanie to ma postać:

```

library Nazwa_typu_standardowego
endlib

```

Zbiór specyfikacji niesparametryzowanych typów daje się sprowadzić do postaci kanonicznej. Polega to wykonaniu następujących kroków:

- każdą specyfikację złożoną, to znaczy taką, w której występują omówione konstrukcje strukturalne, należy przekształcić do specyfikacji kanonicznej,
- nazwy w zbiorze specyfikacji typów należy przemianować w taki sposób, aby otrzymać zbiór typów rozłącznych,
- listy wszystkich rodzajów oraz listy wszystkich symboli operacji występujących w specyfikacjach składowych należy połączyć we wspólne listy.

Semantyka zbioru specyfikacji typów jest zdefiniowana jako semantyka jego postaci kanonicznej.

Podczas stosowania równościowego definiowania własności operacji pojawiają się dwa problemy. Problem pierwszy wiąże się z pytaniem, czy podany zestaw równości jest kompletny – czy uwzględnia wszystkie potrzebne własności. Problem drugi wiąże się z pytaniem, czy podany zestaw własności jest niesprzeczny, czyli czy operacje mają niepustą semantykę. Ogólnie są to problemy nierozstrzygalne, co oznacza, że nie istnieją algorytmy do automatycznego badania własności zupełności i niesprzeczności.

Typy danych w językach specyfikacji i programowania są definiowane jeszcze na inne sposoby. Dominujące podejście, nazwane tu podejściem „programistycznym”, polega na zdefiniowaniu pewnej liczby typów elementarnych, na przykład typy liczbowe, napisowe, logiczne. Każdy typ elementarny ma określony zbiór wartości oraz określony zbiór operacji. Typy elementarne można składać ze sobą, definiując w ten sposób typy złożone, które można składać ze sobą ponownie, definiując coraz bardziej rozbudowane typy. Typowymi sposobami składania typów jest tworzenie rekordów (krotek), ciągów, zbiorów. Specyfiką takiego podejścia jest to, że związane z nimi zbiory operacji są ograniczone w zasadzie tylko do operacji selekcji elementów składowych typów złożonych. Operacje, które w omawianym podejściu algebraicznym stanowią element integralny definiowanych typów, można definiować poza definicją typu, na przykład w postaci definicji procedur.

Należy zwrócić uwagę na jeszcze jedną różnicę pomiędzy podejściem algebraicznym a „programistycznym”. Operacje w podejściu algebraicznym muszą być funkcjami całkowicie określonymi, podejście „programistyczne” dopuszcza natomiast definiowanie operacji, które mogą być funkcjami określonymi częściowo.

4.7. Przykłady specyfikacji

Przedstawiony dalej zestaw przykładów specyfikacji typów opiera się na typach wybranych z biblioteki typów języka LOTOS. Spośród wybranych typów bibliotecznych, dwa typy *Element* oraz *BasicNaturalNumber* są nieco zmodyfikowane, w celu dostosowania do potrzeb dalszych rozdziałów. Przedstawiamy specyfikację typu reprezen-

tującego zbiór operacji mnogościowych na zbiorze elementów dowolnie ustalonego typu. Typ ten, nazwany *Set*, jest typem parametryzowanym, a inne przedstawiane tu typy pełnią rolę pomocniczą – są niezbędne do jego specyfikacji. Typami pomocniczymi są typy: *Boolean*, *Element*, *NaturalNumber*. Pierwszy z tych typów był przedstawiony w poprzednim podrozdziale.

Typ *Element* jest typem złożonym, jest on rozszerzeniem typu *Boolean*:

```

type Element is Boolean
formalsorts   Elem
formalopns   _eqEl_, _neEl_ : Elem, Elem -> Bool
formaleqns   forall x, y : Elem
                 ofsort Elem
                   x eqEl y = true => x = y ;
                 ofsort Bool
                   x = y => x eqEl y = true ;
                   x neEl y = not(x eqEl y) ;
endtype

```

Modyfikacja typu *Element* względem specyfikacji standardowej polega tu na wykorzystaniu typu *Boolean* zamiast typu parametryzowanego *FBool*.

Typ *Element* reprezentuje dowolnie określoną zbiorowość, w której operacje *eqEl* oraz *neEl* mają służyć stwierdzeniu, czy dwa elementy tej zbiorowości są jednakowe czy różne. Zbiorowość tę, reprezentowaną formalnym rodzajem *Elem*, i podobnie operacje formalne *eqEl*, *neEl* można ukonkretnić. Ukonkretniony typ zachowuje wszystkie własności operacji określone równościami formalnymi.

Typ *NaturalNumber* jest również typem złożonym. Do jego definicji jest wymagany wcześniej zdefiniowany typ *Boolean* oraz typ *BasicNaturalNumber* definiowany poniżej:

```

type   BasicNaturalNumber is
sorts  Nat
opns   0 : -> Nat
          Succ, Pred : Nat -> Nat
          _+_ , _*_ , _**_ : Nat, Nat -> Nat
eqns   forall m, n : Nat
          ofsort Nat
            Pred(0) = 0 ;
            Pred(Succ(n)) = n ;
            m + 0 = m ;
            m + Succ(n) = Succ(m) + n ;

```

$$\begin{aligned}
 m * 0 &= 0 && ; \\
 m * Succ(n) &= m + (m * n) && ; \\
 m ** 0 &= Succ(0) && ; \\
 m ** Succ(n) &= m * (m ** n) && ;
 \end{aligned}$$

endtype

Modyfikacja typu *BasicNaturalNumber* względem specyfikacji standardowej polega tu na wprowadzeniu operacji *Pred*. Operacja ta jest zdefiniowana jako funkcja całkowicie określona, co wynika – jak wspomniano wcześniej – z ogólnych własności definicji operacji poprzez równości. Odbiega to od powszechnie przyjmowanego znaczenia tej operacji, która dla argumentu 0 jest nieokreślona.

Definicja typu *NaturalNumber* przedstawia się następująco:

```

type   NaturalNumber is BasicNaturalNumber, Boolean
opns   _eq_, _ne_, _lt_, _le_, _ge_, _gt_ : Nat, Nat -> Bool
eqns   forall m, n : Nat
          ofsort Bool
            0 eq 0 = true ;
            0 eq Succ(m) = false ;
            Succ(m) eq 0 = false ;
            Succ(n) eq Succ(m) = n eq m ;
            m ne n = not(m eq n) ;
            0 lt 0 = false ;
            0 lt Succ(n) = true ;
            Succ(n) lt 0 = false ;
            Succ(m) lt Succ(n) = m lt n ;
            m le n = (m lt n) or (m eq n) ;
            m ge n = not(m lt n) ;
            m gt n = not(m le n) ;

```

endtype

Mając zdefiniowane typy składowe, możemy przedstawić specyfikację parametryzowanego typu zbiorowego *Set*:

```

type   Set is Element, Boolean, BasicNaturalNumber
sorts  Set
opns   {} : -> Set
          Insert, Remove, _IsIn_, _NotIn_, _NotIn_ : Element, Set -> Set
          _Union_, _Ints_, _Minus_ : Set, Set -> Set

```

```

    _eq_, _ne_, _Includes_, _IsSubsetOf_      : Set, Set -> Bool
    Card                                      : Set -> Nat

eqns forall x, y : Elem, s, t : Set
    ofsort Set
        Insert(x, Insert(x, s)) = Insert(x, s) ;
        Insert(x, Insert(y, s)) = Insert(y, Insert(x, s)) ;
        Remove(x, { }) = { } ;
        x eq y = true =>
        Remove(x, Insert(y, s)) = Remove(x, s) ;
        x ne y = true =>
        Remove(x, Insert(y, s)) = Insert(y, Remove(x, s)) ;
        { } Union s = s ;
        Insert(x, s) Union t = Insert(x, s Union t) ;
        { } Ints s = { } ;
        x IsIn t = true =>
        Insert(x, s) Ints t = Insert(x, s Ints t) ;
        x NotIn t = true =>
        Insert(x, s) Ints t = s Ints t ;
        s Minus { } = s ;
        s Minus Insert(x, t) = Remove(x, s) Minus t ;
    ofsort Bool
        x IsIn { } = false ;
        x eq y =>
        x IsIn Insert(y, s) = true ;
        x ne y =>
        x IsIn Insert(y, s) = x IsIn s ;
        x NotIn s = not(x IsIn s) ;
        s Includes { } = true ;
        s Includes Insert(x, t) = (x IsIn s) and (s Includes t) ;
        s IsSubsetOf t = t Includes s ;
        s eq t = (s Includes t) and (t Includes s) ;
        s ne t = not(s eq t) ;
    ofsort Nat
        Card({ }) = 0 ;
        x NotIn s = true =>
        Card(Insert(x, s)) = Succ(Card(s)) ;

endtype

```


Typ parametryczny *Set* można w różny sposób ukonkretniać. Przyjmując na przykład, że nośnikiem będzie zbiór liczb naturalnych, odpowiednia specyfikacja przyjmie postać:

```

type SetOfNaturalNumbers is Set
actualizedby NaturalNumber using
  sortnames
    Nat for Elem
  opnames
    eq for eqEl
    ne for neEl
endtype

```

4.8. Uwagi końcowe

Abstrakcyjne typy danych były przedmiotem wielu prac i wielu ujęć, na przykład: [Burstall, Goguen 1982], [Tatcher, Wagner, Wright 1982]. Omówione w niniejszym rozdziale podejście oparte na ACT ONE wybrano dlatego, że znalazło odzwierciedlenie w języku LOTOS.

Warto też dodać, że abstrakcyjne typy danych były również uwzględnione w początkowych pracach nad językiem specyfikacji SDL: [ITU-T, 1999], [Ellsberg, Hogrefe, Sarma 1997].

Przedstawione podejście równościowego definiowania abstrakcyjnych typów danych ma eleganckie matematyczne podstawy, ale ma także wady. We wcześniejszej części rozdziału zwrócono już uwagę na dwa ograniczenia wewnętrzne:

Pierwsze wiąże się z tym, że w obrębie abstrakcyjnego typu danych możliwa jest definicja tylko takich operacji, które są funkcjami całkowitymi. Praktycznie bardzo często posługujemy się funkcjami częściowo określonymi wraz z mechanizmami obsługi wyjątków.

Drugie wiąże się z koniecznością sprawdzania niesprzeczności i kompletności (zupełności) zdefiniowanych typów. Ponieważ badanie równoważności termów dla dowolnej algebry jest problemem nierozstrzygalnym, nie można użytkowników wspomagać w pełnym zakresie odpowiednimi narzędziami programowymi. Ostatecznie zatem obowiązek sprawdzenia niesprzeczności spoczywa na użytkowniku języka. Dodatkowo, stosowane w implementacji różne strategie przekształcania zbioru równości w zbiór reguł systemu dowodzenia, a także stosowania tego systemu mogą prowadzić do różnych wyników – podczas, gdy jeden system może dawać odpowiedź „tak” albo „nie” na zadane pytanie, inny może dawać odpowiedź „nie wiem”.

Problem zupełności jest jeszcze bardziej złożony, gdyż wymaga dodatkowo określenia „bazy” odniesienia, czyli tego, względem czego jest badana zupełność.

Można jeszcze wskazywać na inne wady. Na przykład strukturalizacja typów nie zapobiega możliwości rozproszenia podobnych rodzajów i operacji w wielu miejscach, a każda funkcja definiowana w typie jest widziana jako globalna – nie ma możliwości deklarowania funkcji jako lokalnych, dodanie nowej równości może istotnie zmienić semantykę typu.

Zwraca się jeszcze uwagę na to, że – ze względu na zamierzony zakres zastosowania języka LOTOS – intelektualny wysiłek potrzebny do zrozumienia podstaw teoretycznych nie pozostaje w proporcji do praktycznych zamierzeń, na przykład do specyfikacji komunikatów protokołu komunikacyjnego. Dla wielu użytkowników abstrakcyjne typy danych pozostają raczej problemem niż rozwiązaniem.

Wymienione problemy związane z użyciem równościowej specyfikacji abstrakcyjnych typów danych były jedną z przyczyn rewizji języka LOTOS [Garavel, Sighireanu 1996a], [Garavel, Sighireanu 1996b] i opracowania wersji rozszerzonej E-LOTOS [ISO/IEC FDIS 15437, 2001]. W nowej wersji języka zdecydowano się na wprowadzenie mieszanych mechanizmów definiowania typów danych; są pozostawione, nieco zmodyfikowane, mechanizmy abstrakcyjnych typów danych, wraz z dołączonymi wybranymi mechanizmami specyfikacji typów danych z funkcjonalnego języka programowania ML [Milner, Tofte, Harper 1990].

Podobnie postąpiono w rozwoju języka SDL – do wcześniej wprowadzonych abstrakcyjnych typów danych dołączono typy danych spotykane w imperatywnych językach programowania (języki C i C++).

Dodatkowo narzędzia programistyczne wspomagające projektowanie w SDL umożliwiły stosowanie notacji ASN.1 (*Abstract Syntax Notation One*). ASN.1 jest językiem formalnym do zapisu składni abstrakcyjnej typów danych oraz do kodowania danych w postaci ciągów zero-jedynkowych. Jako standard ISO jest opisany w serii norm ISO/IEC 8824-3, 4 oraz ISO/IEC 8825-1, 2, a przystępny jego opis zawiera książka [Kosmulska-Bochenek 2002].

5. LOTOS – opis języka

5.1. Akcje komunikacyjne

Specyfikacja systemu rozproszonego w LOTOSie jest traktowana jako proces. Procesy są podstawową jednostką strukturalizacji specyfikacji. Opisywany system przedstawia się jako hierarchię procesów, z których najbardziej zewnętrzny reprezentuje cały system, a procesy w nim zagnieżdżone reprezentują składowe tego systemu. Komunikacja jest niepodzielną czynnością, podczas której pomiędzy procesem a jego otoczeniem następuje przepływ danych. Realizacja komunikacji wymaga synchronizacji komunikujących się procesów.

Proces w LOTOSie, podobnie jak proces w CCS, w danym stanie oferuje swemu otoczeniu zbiór ofert komunikacji. Oferta komunikacji ma postać akcji. Realizację akcji nazywa się interakcją. Każdej interakcji towarzyszy synchronizacja oraz wymiana informacji. Szczególnym rodzajem jest interakcja, której towarzyszy pusta wymiana danych, czyli interakcja ograniczająca się tylko do synchronizacji komunikujących się partnerów. Takie właśnie zdarzenia są dopuszczalne w omawianym tutaj LOTOSie bazowym – podzbiorze pełnego języka LOTOS.

Mechanizmy komunikacji w LOTOSie, w porównaniu do komunikacji w CCS, są znacznie rozbudowane. Wynika to z dwóch powodów: po pierwsze, oprócz synchronizacji komunikujących się procesów, mamy do czynienia z wymianą danych, a po drugie – oprócz komunikacji bilateralnej dopuszcza się także komunikację multilateralną. Z tych powodów definicja akcji komunikacyjnych jest bardziej złożona. Niech, jak poprzednio, G oznacza dowolny, co najwyżej przeliczalny zbiór nazw bramek. Wyróżnia się elementarne i strukturalne akcje komunikacyjne.

Akcje elementarne mają postać:

$$g ! t \quad \text{oraz} \quad g ? x : s \quad (5.1)$$

gdzie:

$g \in G$ oznacza nazwę bramki,

e jest pewnym wyrażeniem – termem stałym (zob. poprzedni rozdział),

x jest zmienną,

s jest identyfikatorem rodzaju.

Pierwsza z tych akcji, nazywana elementarną akcją wyjścia, polega na wysłaniu, przez proces realizujący tę akcję, wartości wyrażenia t przez bramkę g , element $!t$ nazywamy wyjściowym elementem komunikacyjnym. Druga z tych akcji, nazywana elementarną akcją wejścia, polega na odbiorze, przez proces realizujący tę akcję, pewnej wartości rodzaju t z bramki g i przypisaniu tej wartości zmiennej x , element $?x : s$ nazywamy wejściowym elementem komunikacyjnym.

Najprostszy przypadek komunikacji, w której uczestniczą dwa procesy (komunikacja bilateralna), wymaga, aby jeden z nich był gotowy do realizacji elementarnej akcji wejścia, a drugi – elementarnej akcji wyjścia. Akcje te muszą być zgodne, tzn. obie muszą się odnosić do tej samej bramki, a wyrażenie e w akcji wyjścia musi być takiego samego rodzaju jak zmienna x w akcji wejścia. Realizację pary zgodnych akcji nazywa się interakcją.

W bardziej ogólnym przypadku komunikacja pary procesów może polegać na realizacji innych zestawów akcji elementarnych. Możliwości te są zgromadzone w tabeli 5.1, w której przedstawiono komunikację pomiędzy dwoma procesami p oraz q .

Tabela 5.1

Lp.	Akcja w procesie p	Akcja w procesie q	Warunek zgodności akcji	Wynik realizacji akcji
1	$g ! t$	$g ? y : s$	$sort(t) = s$	$y := [t]$
2	$g ? x : s$	$g ! t$	$sort(t) = s$	$x := [t]$
3	$g ! t_1$	$g ! t_2$	$[t_1] = [t_2]$	
4	$g ? x : s_1$	$g ? y : s_2$	$s_1 = s_2$	$x := v, y := v,$ gdzie v jest dowolną wartością rodzaju s_1

Funkcja $sort(t)$ oznacza rodzaj wyrażenia (termu) t .

Przypomnijmy, że – zgodnie z rozważaniami poprzedniego rozdziału – wartością termu stałego t jest klasa abstrakcji $[t]$, określona na zbiorze termów stałych rodzaju s .

Oznaczenie $x := v$ jest przypisaniem wartości v zmiennej x . Używany w LOTOSie zmiennym można przypisywać wartości tylko jednokrotnie, co znaczy, że raz przypisanej wartości nie można już zmieniać – LOTOS jest językiem aplikatywnym.

Wspólnym warunkiem realizacji akcji, niezapisanym w tabeli 5.1, jest – podobnie jak w przypadku języka CCS – synchronizacja obu procesów p oraz q . Dwa pierwsze przypadki ujęte w tabeli odpowiadają omówionemu wyżej przesłaniu wartości pomiędzy dwoma procesami. W trzecim przypadku procesy nie przesyłają pomiędzy sobą żadnych wartości, a tylko synchronizują się wzajemnie. Czwarty przypadek obrazuje interakcję, której wynikiem jest wygenerowanie, w sposób niedeterministyczny, pewnej wartości wspólnie zapamiętanej przez oba procesy.

Bardziej rozbudowaną akcją komunikacyjną jest *akcja strukturalna*. Akcje strukturalne służą do przekazywania, podczas jednej interakcji, zestawu wartości, a nie tylko pojedynczej wartości, jak w przypadku akcji elementarnych. Akcja strukturalna ma postać

$$g \alpha_1 \alpha_2 \dots \alpha_k \quad \text{dla } k \geq 0 \quad (5.2)$$

gdzie α_i są wcześniej określonymi elementami komunikacyjnymi postaci (5.1), czyli

$$? x_i : s_i \quad \text{lub} \quad ! t_i. \quad (5.3)$$

Dwie akcje strukturalne:

$$g \alpha_1 \alpha_2 \dots \alpha_k \quad \text{oraz} \quad g \beta_1 \beta_2 \dots \beta_l \quad (5.4)$$

są *zgodne* wtedy i tylko wtedy, gdy $k = l$ oraz dla każdego $i = 1, \dots, k$ akcje elementarne $g \alpha_i$ oraz $g \beta_i$ są zgodne w sensie tabeli 5.1.

Realizację pary akcji, czyli interakcję, będziemy oznaczać

$$g \nu_1, \dots, \nu_k \quad (5.5)$$

gdzie ν_i jest wartością przesłaną lub wygenerowaną przez elementy komunikacyjne α_i oraz β_i dla $i = 1, \dots, k$.

Najbardziej ogólna postać akcji strukturalnej przedstawia się następująco:

$$g \alpha_1 \alpha_2 \dots \alpha_k [c] \quad (5.6)$$

gdzie c jest predykatem – wyrażeniem logicznym, czyli termem rodzaju *Bool*. Predykat c zawęża dopuszczalne wykonania tej akcji tylko do tych interakcji, które spełniają ten predykat.

Dla akcji $g \alpha_1 \alpha_2 \dots \alpha_k$ wyznaczmy zbiór zmiennych V , występujących w wejściowych elementach komunikacyjnych

$$V = \{x_i \mid \alpha_i \equiv ? x_i : s_i, i = 1, \dots, k\}. \quad (5.7)$$

Niech $V = \{x_{j_1}, \dots, x_{j_m}\}$, wówczas interakcja $g \nu_1, \dots, \nu_k$ musi spełniać warunek

$$D \vdash c[x_{j_1} ::= t_{j_1}, \dots, x_{j_m} ::= t_{j_m}] = \text{true} \quad (5.8)$$

albo krótko

$$D \vdash c[x_{j_1} ::= t_{j_1}, \dots, x_{j_m} ::= t_{j_m}] \quad (5.9)$$

gdzie $\nu_{j_i} = [t_{j_i}]$ dla $i = 1, \dots, m$. Symbol $\vdash$, zgodnie z oznaczeniami rozdziału 4., oznacza istnienie dowodu równości w systemie dowodzenia D , generowanym przez specyfikację typów danych.

Dotychczas omawiano komunikację, w której uczestniczyły tylko dwa procesy. LOTOS dopuszcza interakcje, w których bierze udział większa liczba procesów. Zbiór akcji jest zbiorem akcji zgodnych, jeżeli istnieje wspólna interakcja, która jest realizowalna przez każdą z tych akcji.

Oprócz opisanych wyżej akcji komunikacyjnych, podobnie jak w CCS, wyróżnia się w LOTOSie jeszcze akcje wewnętrzne, oznaczane symbolem *i*. Akcja wewnętrzna jest odzwierciedleniem pewnej czynności obliczeniowej lub komunikacyjnej, która – z punktu widzenia zewnętrznego obserwatora procesu – jest całkowicie niewidocz-

na. Akcja zachodzi autonomicznie na skutek decyzji pewnego obiektu wewnętrznego danego procesu. Dla zewnętrznego obserwatora procesu akcja wewnętrzna jest nieobserwowalna, a informacja o zajściu akcji i mówi tylko o tym, że we wnętrzu procesu wykonano pewne obliczenie lub zaszła pewna komunikacja.

5.2. Procesy i wyrażenia behawioralne

Składnia LOTOSa jest rozszerzeniem składni CCS. Rozszerzenie wynika z:

- uwzględnienia przekazywania danych,
- wprowadzenia dodatkowych operatorów wyrażeń behawioralnych, wynikających między innymi z wprowadzenia komunikacji multilateralnej (oprócz bilateralnej),
- rozbudowania deklaracji procesów i typów.

Szkieletowa postać definicji procesów w postaci kanoniczej przedstawia się następująco:

```

process  nazwa_procesu [lista_bramek_formalnych]
           (lista_formalnych_parametrów_wartościowych) :
           funkcjonalność :=
           wyrażenie_behawioralne
endproc
    
```

(5.10)

W przypadku procesów kanonicznych zakłada się, że nie mają one lokalnych definicji typów lub procesów wewnętrznych, zakłada się natomiast, że ich wspólnym kontekstem jest kanoniczna specyfikacja typów danych. Procesy te dysponują więc rodzajami i operacjami należącymi do kanonicznej specyfikacji typów danych.

Definicje typów omówiliśmy w rozdziale poprzednim, obecnie omówimy wyrażenia behawioralne stanowiące treści procesów.

Słowa kluczowe wyróżnia się wytłuszczoną czcionką, a symbole pomocnicze – kursywą.

Poszczególne elementy definicji procesów mają znaczenie następujące:

Lista_bramek_formalnych, postaci $g_1, \dots, g_n$, dla $n \geq 0$, oraz $g_i \neq g_j$, dla $i \neq j$, określa nazwy formalne bramek, przez które proces może się komunikować ze swoim otoczeniem. Instancja procesu o zadanej definicji musi nazwy formalne zastąpić nazwami aktualnymi.

Procesy mogą dysponować zmiennymi lokalnymi, których wartościowanie może być ustalane w momencie tworzenia instancji procesu. Służy do tego *lista_formalnych_parametrów_wartościowych*, postaci $x_1 : s_1, \dots, x_n : s_n$, dla $n \geq 0$, gdzie $x_1, \dots, x_n$ są zmiennymi, a $s_1, \dots, s_n$ są rodzajami tych zmiennych.

Funkcjonalność procesu charakteryzuje sposób kończenia działania procesu i przyjmuje jedną z postaci:

$$\begin{aligned} & \mathbf{noexit} \\ & \mathbf{exit}(s_1, \dots, s_n) \text{ dla } n \geq 0. \end{aligned} \tag{5.11}$$

Pierwsza postać oznacza, że proces albo nigdy się nie kończy, albo – kończąc się – staje się procesem nieaktywnym. Druga postać oznacza, że jeśli proces się kończy, to produkuje zestaw wartości rodzajów $s_1, \dots, s_n$, które może przekazać do zainicjowania innego procesu – jego następnika.

Pojęcie funkcjonalności odnosi się także do wyrażeń behawioralnych, dlatego powstaje wymóg, by funkcjonalność procesu była zgodna z funkcjonalnością wyrażenia behawioralnego stanowiącego jego treść.

Zbiór wyrażeń behawioralnych *Beh* jest określony następująco:

$$\begin{aligned} B ::= & \mathbf{stop} \mid \\ & \mathbf{i}; B_1 \mid \\ & g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k; B_1 \mid \\ & g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k[c]; B_1 \mid \\ & \mathbf{exit}(t_1, \dots, t_n) \mid \\ & B_1 \gg \mathbf{accept} \ x_1 : s_1, \dots, x_n : s_n \ \mathbf{in} \ B_2 \mid \\ & B_1 \ [\] \ B_2 \mid \\ & \mathbf{hide} \ g_1, \dots, g_n \ \mathbf{in} \ B_1 \mid \\ & B_1 \ [> \ B_2 \mid \\ & B_1 \ [[g_1, \dots, g_n] \mid B_2 \mid \\ & [c] \rightarrow B_1 \mid \\ & \mathbf{let} \ x_1 = t_1, \dots, x_n = t_n \ \mathbf{in} \ B_1 \mid \\ & \mathbf{choice} \ g \ \mathbf{in} \ [g_1, \dots, g_n] \ B_1 \mid \\ & \mathbf{choice} \ x \ [\] \ B_1 \mid \\ & \mathbf{par} \ g \ \mathbf{in} \ [g_1, \dots, g_n] \ [[h_1, \dots, h_m] \mid B_1 \mid \\ & p[g_1, \dots, g_n](t_1, \dots, t_m) \mid \\ & (B_1) \end{aligned} \tag{5.12}$$

Symbole B, B_1, B_2 oznaczają wyrażenia behawioralne, g, h (z ewentualnymi indeksami) oznaczają bramki komunikacyjne, $x_1, \dots, x_n$ – zmienne, $s_1, \dots, s_n$ – rodzaje oraz $t_1, \dots, t_m$ – termy.

W przypadku wyrażenia $\mathbf{exit}(t_1, \dots, t_n)$ symbol t_i ($i = 1, \dots, n$) może dodatkowo przyjmować postać $\mathbf{any} \ s_i$. Symbol $\mathbf{any} \ s_i$ służy na oznaczenie dowolnego termu rodzaju s_i .

Zbiór wyrażeń behawioralnych musi spełniać ograniczenia kontekstowe, wynikające z wymagań funkcjonalnych. Po oznaczeniu przez $funct(B)$ funkcjonalności wyrażenia B ograniczenia te można przedstawić tabelą 5.2.

Tabela 5.2

B	$funct(B)$	Warunek zgodności
stop	noexit	
$a; B_1$	$funct(B_1)$	a – dowolna akcja
exit ($t_1, \dots, t_n$)	exit ($s_1, \dots, s_n$)	$sort(t_1) = s_1, \dots, sort(t_n) = s_n$
$B_1 \gg \text{accept } x_1 : s_1, \dots, x_n : s_n \text{ in } B_2$	$funct(B_2)$	$funct(B_1) = \text{exit}(s_1, \dots, s_n)$
$B_1 [\] B_2$	$funct(B_1)$	$funct(B_1) = funct(B_2)$
hide $g_1, \dots, g_n \text{ in } B_1$	$funct(B_1)$	
$B_1 [> B_2$	$funct(B_1)$	$funct(B_1) = funct(B_2)$
$B_1 [[g_1, \dots, g_n] B_2$	$funct(B_1)$	$funct(B_1) = funct(B_2)$
$[c] \rightarrow B_1$	$funct(B_1)$	
let $x_1 = t_1, \dots, x_n = t_n \text{ in } B_1$	$funct(B_1)$	
choice $g \text{ in } [g_1, \dots, g_n] B_1$	$funct(B_1)$	
choice $x [\] B_1$	$funct(B_1)$	
par $g \text{ in } [g_1, \dots, g_n] [[h_1, \dots, h_m] B_1$	$funct(B_1)$	
$p[g_1, \dots, g_n](t_1, \dots, t_m)$	$funct(B_1)$	gdzie B_1 – treść procesu p
(B_1)	$funct(B_1)$	

Znaczenie poszczególnych konstrukcji wyrażeń behawioralnych przedstawiają się następująco:

stop jest procesem elementarnym reprezentującym *proces pusty*, to znaczy taki, który nie oferuje swemu otoczeniu żadnych akcji komunikacyjnych.

Prefiksowanie akcją wewnętrzną i ; B oznacza zachowanie, na które składa się wykonanie akcji wewnętrznej, a dalsze zachowanie jest określone przez wyrażenie behawioralne B . Podobne znaczenie mają prefiksowanie akcją komunikacyjną bez ograniczenia $g \alpha_1 \alpha_2 \dots \alpha_k$; B i z ograniczeniem $g \alpha_1 \alpha_2 \dots \alpha_k[c]$; B , to znaczy po wykonaniu akcji w wyniku synchronizacji z otoczeniem dalsze zachowanie jest określone przez wyrażenie behawioralne B .

Wyrażenie **exit**($t_1, \dots, t_n$) jest procesem elementarnym, którego jedyną czynnością jest obliczenie wartości termów $t_1, \dots, t_n$ i przekazanie ich do innego procesu, który jest jego następnikiem. Wyrażenie to należy odróżnić od tekstu **exit**($s_1, \dots, s_n$), który oznacza funkcjonalność.

Pełne wyjaśnienie roli procesu **exit**($t_1, \dots, t_n$) wiąże się z wyrażeniem złożenia aktywowującego postaci: $B_1 \gg \text{accept } x_1 : s_1, \dots, x_n : s_n \text{ in } B_2$.

Operator binarny $\gg$ oznacza sekwencyjne wykonanie wyrażeń najpierw wyrażenia B_1 , a następnie wyrażenia **accept** $x_1 : s_1, \dots, x_n : s_n$ **in** B_2 .

Od wyrażenia B_1 wymaga się, aby miało ono funkcjonalność **exit**($s_1, \dots, s_n$), czyli aby – kończąc się – produkowało zestaw wartości $v_1, \dots, v_n$ rodzajów $s_1, \dots, s_n$. Oznacza to, że wyrażenie B_1 – kończąc się – ma wykonać proces **exit**($t_1, \dots, t_n$). Proces ten wytworzony zestaw wartości przekazuje do wyrażenia **accept** $x_1 : s_1, \dots, x_n : s_n$ **in** B_2 za pośrednictwem specjalnej, lokalnej bramki komunikacyjnej. Bramka ta jest oznaczana jako δ . Interakcja pomiędzy wyrażeniami po lewej i prawej stronie operatora $\gg$ ma postać: $\delta v_1, \dots, v_n$. Zachowanie reprezentowane przez wyrażenie $B_1 \gg \text{accept } x_1 : s_1, \dots, x_n : s_n \text{ in } B_2$ jest sekwencyjnym złożeniem zachowań jego wyrażeń składowych.

Zachowanie reprezentowane przez wyrażenie wyboru $B_1 [] B_2$ jest mnogościową sumą zachowań wyrażeń składowych. Oznacza to, że wyrażenie oferuje swemu otoczeniu akcje obu wyrażeń składowych, a po wyborze akcji należącej do jednego z tych wyrażeń, np. B_1 , dalsze zachowanie przebiega zgodnie z zachowaniem określonym przez B_1 .

Wyrażenie przesłaniania **hide** $g_1, \dots, g_n$ **in** B_1 oznacza, że bramki $g_1, \dots, g_n$ mogą być użyte do komunikacji tylko wewnątrz wyrażenia B_1 . Bramki te nie mogą być wykorzystane do komunikacji z otoczeniem tego wyrażenia. Komunikacja na dowolnej z tych bramek, przez zewnętrznego obserwatora wyrażenia **hide** $g_1, \dots, g_n$ **in** B_1 , będzie traktowana jako akcja wewnętrzna.

Zachowanie wyrażenia złożenia deaktywującego $B_1 [> B_2$ można nieformalnie opisać jako zachowanie dwóch równoległych procesów reprezentowanych wyrażeniami B_1 oraz B_2 . Proces reprezentowany przez B_1 wykonuje się aż do chwili, gdy rozpocznie działanie proces reprezentowany wyrażeniem B_2 . Jeżeli B_1 zakończy działanie zanim działanie rozpocznie B_2 , oznacza to zakończenie działania całego wyrażenia $B_1 [> B_2$. Jeżeli natomiast B_2 rozpocznie działanie przed zakończeniem B_1 , to następuje przerwanie działania B_1 i dalsze zachowanie przebiega zgodnie z B_2 .

Wyrażenie złożenia równoległego $B_1 [|g_1, \dots, g_n|] B_2$ oznacza dwa równoległe procesy reprezentowane wyrażeniami B_1 oraz B_2 . Bramki $g_1, \dots, g_n$ służą do synchronizacji tych procesów, co oznacza, że w każdej komunikacji z udziałem tych bramek muszą jednocześnie uczestniczyć B_1 oraz B_2 .

Wyrażenie złożenia równoległego ma dwie szczególne postaci: $B_1 ||| B_2$ oraz $B_1 || B_2$. Pierwsza z nich jest równoważna wyrażeniu $B_1 [| []] B_2$, co oznacza, że zbiór wspólnych bramek komunikacyjnych jest pusty, czyli procesy reprezentowane przez B_1 oraz B_2 nie synchronizują się ze sobą. Druga – jest równoważna $B_1 [|g_1, \dots, g_n|] B_2$, gdzie $\{g_1, \dots, g_n\}$ oznaczają zbiór wszystkich bramek, czyli że procesy reprezentowane przez B_1 oraz B_2 muszą się synchronizować ze sobą podczas każdej komunikacji.

Wyrażenie dozorowane $[c] \rightarrow B_1$ reprezentuje takie zachowanie jak B_1 , pod warunkiem, że prawdziwy jest predykat c . Jeżeli predykat c jest fałszywy, znaczeniem całego wyrażenia jest proces pusty.

Wyrażenie **let** $x_1 = t_1, \dots, x_n = t_n$ **in** B_1 definiuje lokalne wartościowania zmiennych $x_1, \dots, x_n$ wykorzystywane w wyrażeniu B_1 .

Wyrażenie niedeterministycznego wyboru bramki **choice** g **in** $[g_1, \dots, g_n]$ B_1 oznacza niedeterministyczne zastąpienie bramki g jedną z bramek $g_1, \dots, g_n$. Zachowanie wyrażenia **choice** g **in** $[g_1, \dots, g_n]$ B_1 jest takie, jak wyrażenia B_1 , w którym bramkę g zastąpiono dowolną z bramek $g_1, \dots, g_n$.

Wyrażenie niedeterministycznego wyboru wartości **choice** $x: s$ B_1 oznacza przypisanie zmiennej x niedeterministycznie ustalonej wartości rodzaju s . Zachowanie reprezentowane przez to wyrażenie jest takie, jak zachowanie wyrażenia B_1 , zmodyfikowanego w ten sposób, że zmiennej x jest przypisana pewna wartość rodzaju s .

Wyrażenie powielenia bramki **par** g **in** $[g_1, \dots, g_n]$ $[h_1, \dots, h_m]$ B_1 oznacza n -krotne złożenie równoległe powielonego zmodyfikowanego wyrażenia B_1 . Kolejna modyfikacja wyrażenia B_1 polega na zastąpieniu bramki g bramką g_i , dla $i = 1, \dots, n$, natomiast bramki $h_1, \dots, h_m$ są parametrami złożenia równoległego.

Wyrażenie $p[h_1, \dots, h_n](t_1, \dots, t_m)$ jest instancją procesu o definicji

process $p[g_1, \dots, g_n] (x_1 : s_1, \dots, x_m : s_m) : \mathbf{exit}(s'_1, \dots, s'_k) := B$
endproc

gdzie:

- B jest treścią procesu,
- $h_1, \dots, h_n$ jest listą bramek aktualnych,
- $t_1, \dots, t_m$ jest listą termów rodzajów $s_1, \dots, s_m$.

Zachowanie reprezentowane przez to wyrażenie jest takie jak zachowanie wyrażenia B , zmodyfikowanego w taki sposób, że nazwy bramek $g_1, \dots, g_n$ zostają zastąpione nazwami $h_1, \dots, h_n$ a zmiennym $x_1 : s_1, \dots, x_m : s_m$ zostają przypisane wartości $t_1, \dots, t_m$.

5.3. Semantyka operacyjna

Kanoniczna specyfikacja zachowania jest określana jako para $BSpec = \langle p_0, Proc \rangle$, gdzie p_0 jest procesem początkowym, a $Proc$ – zbiorem definicji procesów. Kontekstem specyfikacji zachowań jest specyfikacja typów $TSpec$.

Semantyka operacyjna specyfikacji jest definiowana w sposób strukturalny na podstawie definicji składniowych wyrażeń behawioralnych. Wynikiem definicji semantyki jest etykietowany system przejść postaci

$$TS(Spec) = \langle Beh, Act, Tr, B_0 \rangle \quad (5.13)$$

gdzie:

- Beh jest zbiorem wyrażeń behawioralnych,

$Act = \{g \vee \mid g \in G \cup \{\delta\}, v \in Term_{TSpec}(\emptyset)^* \} \cup \{i\}$ jest zbiorem interakcji,
 $Tr = \{ \xrightarrow{a} \mid a \in Act \}$ jest zbiorem relacji przejść,
 B_0 jest treścią procesu początkowego p_0 .

Uwaga: Symbol **i** oznacza akcję wewnętrzną, natomiast *i* oznacza zdarzenie realizacji akcji wewnętrznej.

Zbiór relacji przejść Tr jest definiowany rekursywnie za pomocą następującego zestawu aksjomatów i reguł:

Proces pusty stop – brak aksjomatów.

Prefiksowanie akcją wewnętrzną $B \equiv i; B_1$

$$B \xrightarrow{i} B_1 \quad (A-pre_{int})$$

Prefiksowanie akcją komunikacyjną $B \equiv g \alpha_1 \alpha_2 \dots \alpha_k[c]; B_1$

$$B \xrightarrow{g v_1 \dots v_n} [x_{j_1} ::= t_{j_1}, \dots, x_{j_m} ::= t_{j_m}] B_1 \quad (A-pre_{comm})$$

wtedy i tylko wtedy, gdy:

$v_i = [t_i]$, gdy $\alpha_i \equiv !t_i$ oraz t_i jest termem stałym, dla $i = 1, \dots, n$,

$v_i \in Term_{s_i}(\emptyset) / =_{s_i}$, gdy $\alpha_i \equiv ?x_i : s_i$, dla $i = 1, \dots, n$,

x_{j_i} jest zmienną taką, że $\alpha_{j_i} \equiv ?x_{j_i} : s_{j_i}$, dla $i = 1, \dots, n$,

oraz

$$D \vdash c[x_{j_1} ::= t_{j_1}, \dots, x_{j_m} ::= t_{j_m}]$$

Proces zakończenia $B \equiv \mathbf{exit}(t_1, \dots, t_n)$

$$B \xrightarrow{\delta v_1 \dots v_n} \mathbf{stop} \quad (A-exit)$$

wtedy i tylko wtedy, gdy:

$v_i = [t_i]$, gdy t_i jest termem stałym, dla $i = 1, \dots, n$,

$v_i \in Term_{s_i}(\emptyset) / =_{s_i}$, gdy $t_i = \mathbf{any} \ s_i$, dla $i = 1, \dots, n$.

Sekwencyjne złożenie $B \equiv B_1 >> \mathbf{accept} \ x_1 : s_1, \dots, x_n : s_n \ \mathbf{in} \ B_2$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1 >> \mathbf{accept} \ x_1 : s_1, \dots, x_n : s_n \ \mathbf{in} \ B_2} \quad name(a) \neq \delta \quad (R-accept_1)$$

$$\frac{B_1 \xrightarrow{\delta v_1 \dots v_n} B'_1}{B \xrightarrow{i} [x_1 ::= t_1, \dots, x_n ::= t_n] B_2} \quad \text{gdzie } [t_1] = v_1, \dots, [t_n] = v_n \quad (R-accept_2)$$

Wybór $B \equiv B_1 \mid B_2$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad (R\text{-choice}_1)$$

$$\frac{B_2 \xrightarrow{a} B'_2}{B \xrightarrow{a} B'_2} \quad (R\text{-choice}_2)$$

Przestonięcie $B \equiv \text{hide } g_1, \dots, g_n \text{ in } B_1$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad \text{gd}y \text{ name}(a) \notin \{g_1, \dots, g_n\} \quad (R\text{-hide}_1)$$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{i} B'_1} \quad \text{gd}y \text{ name}(a) \in \{g_1, \dots, g_n\} \quad (R\text{-hide}_2)$$

Złożenie deaktywujące $B \equiv B_1 [> B_2]$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1 [> B_2]} \quad \text{gd}y \text{ name}(a) \neq \delta \quad (R\text{-dis}_1)$$

$$\frac{B_1 \xrightarrow{\delta v_1 \dots v_n} B'_1}{B \xrightarrow{\delta v_1 \dots v_n} B'_1} \quad (R\text{-dis}_2)$$

$$\frac{B_2 \xrightarrow{a} B'_2}{B \xrightarrow{a} B'_2} \quad (R\text{-dis}_3)$$

Złożenie równoległe $B \equiv B_1 \parallel [g_1, \dots, g_n] B_2$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1 \parallel [g_1, \dots, g_n] B_2} \quad \text{gd}y \text{ name}(a) \notin \{g_1, \dots, g_n, \delta\} \quad (R\text{-par}_1)$$

$$\frac{B_2 \xrightarrow{a} B'_2}{B \xrightarrow{a} B_1 \parallel [g_1, \dots, g_n] B'_2} \quad \text{gd}y \text{ name}(a) \notin \{g_1, \dots, g_n, \delta\} \quad (R\text{-par}_2)$$

$$\frac{B_1 \xrightarrow{a} B'_1 \quad B_2 \xrightarrow{a} B'_2}{B \xrightarrow{a} B'_1 \parallel [g_1, \dots, g_n] B'_2} \quad \text{gd}y \text{ name}(a) \in \{g_1, \dots, g_n, \delta\} \quad (R\text{-par}_3)$$

Wyrażenie warunkowe $B \equiv [c] \rightarrow B_1$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad \text{gdzie } D \vdash c \quad (R\text{-cond})$$

Lokalne wartościowanie zmiennych $B \equiv \text{let } x_1 = t_1, \dots, x_n = t_n \text{ in } B_1$

$$\frac{[x_1 := t_1, \dots, x_n := t_n] B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad (R\text{-let})$$

Niedeterministyczny wybór bramki $B \equiv \text{choice } g \text{ in } [g_1, \dots, g_n] [] B_1$

$$\frac{B_1[g := g_i] \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad \text{dla } i = 1, \dots, n \quad (R\text{-gate})$$

Niedeterministyczny wybór wartości $B \equiv \text{choice } x : s [] B_1$

$$\frac{[x := t] B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad \text{dla } t \in \text{Term}_s(\emptyset) \quad (R\text{-val})$$

Powielenie bramki $B \equiv \text{par } g \text{ in } [g_1, \dots, g_n] |[h_1, \dots, h_m]| B_1$

$$\frac{B_1[g := g_i] |[h_1, \dots, h_m]| \dots |[h_1, \dots, h_m]| B_1[g := g_n] \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad (R\text{-par-gate})$$

Instancja procesu $B \equiv p[g_1, \dots, g_n](t_1, \dots, t_m)$

$$\frac{[x_1 := t_1, \dots, x_m := t_m] B_1[h_1 := g_1, \dots, h_n := g_n] \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad (R\text{-inst})$$

gdzie B_1 jest treścią procesu o specyfikacji

process $p[h_1, \dots, h_n](x_1 : s_1, \dots, x_n : s_n) : \dots := B_1$ **endproc**

Podobnie jak w przypadku języka CCS, specyfikacja w LOTOSie generuje zbiory ciągów akcji. Różnica w definicji wynika z różnicy definicji specyfikacji, gdyż w przypadku CCS specyfikacja jest rozumiana jako para: wyrażenie behawioralne i zbiór definicji procesów, natomiast w przypadku LOTOSa specyfikacja jest rozumiana jako para: definicja wyróżnionego procesu i zbiór definicji pozostałych procesów. Ponieważ wyróżniony proces początkowy w specyfikacji w LOTOSie może mieć różne instancje, stąd:

Definicja 5.1

Specyfikacja $BSpec = \langle p_0, Proc \rangle$, gdzie p_0 jest procesem o definicji

process $p[h_1, \dots, h_n](x_1 : s_1, \dots, x_m : s_m) : fun_p := B_p$ **endproc**

generuje rodzinę zbiorów ciągów akcji:

$$\begin{aligned} & Seq(p[g_1, \dots, g_n](t_1, \dots, t_m)) \\ & = SeqFin(p[g_1, \dots, g_n](t_1, \dots, t_m)) \cup SeqInf(p[g_1, \dots, g_n](t_1, \dots, t_m)) \end{aligned} \quad (5.14)$$

gdzie: $g_1, \dots, g_n$ są dowolnymi bramkami, a $t_1, \dots, t_m$ – dowolnymi termami stałymi rodzajów $s_1, \dots, s_m$.

Podobnie definiuje się rodzinę zbiorów ciągów akcji obserwowalnych:

$$\begin{aligned} & SeqObs(p[g_1, \dots, g_n](t_1, \dots, t_m)) \\ & = SeqObsFin(p[g_1, \dots, g_n](t_1, \dots, t_m)) \\ & \cup SeqObsInf(p[g_1, \dots, g_n](t_1, \dots, t_m)) \end{aligned} \quad (5.15)$$

5.4. Graf tranzycji

Grafy są jedną z często stosowanych form wizualizacji. Stosuje się je także do wizualizacji obliczeń (zachowań) wyrażeń behawioralnych.

Graficzną reprezentacją relacji tranzycji dla wyrażeń behawioralnych są grafy tranzycji. *Graf tranzycji* definiujemy jako etykietowany graf skierowany, którego wierzchołki są etykietowane wyrażeniami behawioralnymi, a łuki – interakcjami, w taki sposób, że łuk pomiędzy wierzchołkami etykietowanymi wyrażeniami B_1 oraz B_2 jest etykietowany interakcją $g \ v_1 \dots v_n$ taką, że $B_1 \xrightarrow{g \ v_1 \dots v_n} B_2$. Graf tranzycji dla danego wyrażenia behawioralnego B jest grafem, którego jeden z wierzchołków, wyróżniony jako wierzchołek początkowy, jest etykietowany wyrażeniem B , a pozostałe wierzchołki są wierzchołkami osiągalnymi z wierzchołka początkowego.

Oczywiście nawet dla prostych wyrażeń behawioralnych grafy tranzycji mogą być nieskończone. Ilustruje to przykład przedstawiający specyfikację komórki pamięci, która może pamiętać jedną wartość określonego rodzaju – dla ustalenia uwagi przyjmujemy, że będzie to rodzaj *nat*. Na komórce można wykonywać dwie operacje. Operacja wpisania wartości powoduje umieszczenie wskazanej wartości, co przyczynia się do utraty poprzednio pamiętanej wartości. Operacja odczytu dostarcza wartości pamiętanej w komórce bez zmiany jej zawartości:

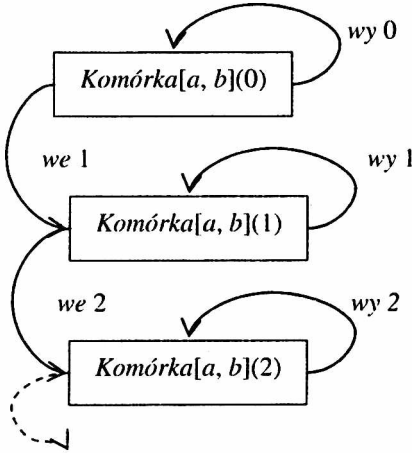
```

process  Komórka[we, wy] (z : nat) : noexit :=
    we ? x : nat; Komórka[we, wy](x)
    []      wy ! z; Komórka[we, wy](z)
endproc

```

Powołanie instancji procesu wymaga określenia początkowej zawartości komórki, na przykład: *Komórka*[a, b](0).

Graf tranzycji dla wyrażenia $Komórka[a, b](0)$ jest oczywiście nieskończony, przykładowy jego fragment przedstawiono na rysunku 5.1.



Rys. 5.1. Przykładowy graf tranzycji

Powodem nieskończoności grafu jest nieskończony zbiór interakcji, a w konsekwencji nieskończoność wyrażeń behawioralnych, w naszym przykładzie, instancji tego samego procesu $Komórka$, które różnią się od siebie tylko wartościowaniem lokalnych zmiennych – parametrów formalnych procesu. W celu wyeliminowania tego rodzaju różnorodności wyrażeń wprowadzamy pojęcie zredukowanego grafu tranzycji.

Definicja 5.2.

Dwa wyrażenia B_1 oraz B_2 są *strukturalnie równoważne* wtedy i tylko wtedy, gdy

$$[x_1 ::= *, \dots, x_n ::= *]B_1 \equiv [y_1 ::= *, \dots, y_m ::= *]B_2$$

gdzie: $\{x_1, \dots, x_n\}$ oraz $\{y_1, \dots, y_m\}$ są zbiorami zmiennych występujących w wyrażeniach B_1 oraz B_2 , $*$ jest wybranym symbolem służącym do tekstowego zastąpienia zmiennych, a $\equiv$ jest relacją równoważności tekstowej

Relacja równoważności strukturalnej dla wyrażeń jest oczywiście relacją równoważności.

Wierzchołkami *zredukowanego grafu tranzycji* są klasy abstrakcji określone przez relację strukturalnej równoważności na zbiorze wierzchołków grafu tranzycji. Klasę równoważności generowaną przez wyrażenie B będziemy oznaczać

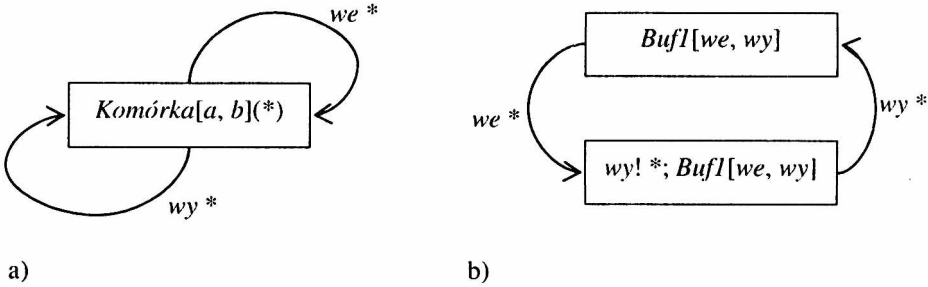
$$[x_1 ::= *, \dots, x_n ::= *]B$$

Łuki zredukowanego grafu tranzycji są określone następująco: wszystkie łuki w grafie tranzycji prowadzące od wierzchołka B_1 do wierzchołka B_2 , etykietowane interakcjami

$g \ v_1 \dots v_n$, są reprezentowane w zredukowanym grafie tranzycji jednym łukiem prowadzącym od wierzchołka reprezentującego klasę abstrakcji $[x_1 ::= *, \dots, x_n ::= *]B_1$ do wierzchołka reprezentującego klasę abstrakcji $[x_1 ::= *, \dots, x_n ::= *]B_2$, etykietowanym schematem interakcji postaci $g \underbrace{* \dots *}_n$.

W przypadku akcji wewnętrznych łuki prowadzące pomiędzy wierzchołkami zredukowanego grafu tranzycji są etykietowane symbolem i .

Wynikiem redukcji grafu tranzycji z rysunku 5.1 jest zredukowany graf tranzycji na rysunku 5.2a.



Rys. 5.2. Przykładowe zredukowane grafy tranzycji

Kolejny przykład przedstawia bufor jednopozycyjny, który różni się od poprzednio rozważanej komórki tym, że odczyt wprowadzonej wartości powoduje skasowanie zawartości bufora

```

process  Buf1[we, wy] : noexit :=
    we? x : nat; wy! x; Buf1[we, wy]
endproc
    
```

Zredukowany graf tranzycji dla instancji procesu $Buf1[we, wy]$ przedstawiono na rysunku 5.2b. Bufora jednopozycyjnego można użyć jako elementu składowego do budowy dwupozycyjnego bufora przesuwającego:

```

process  Buf2[we, wy] : noexit :=
    hide w in
        Buf1[we, w] || [w] Buf1[w, wy]
where
    process Buf1[we, wy] : noexit :=
        we? x : Nat; wy! x; Buf1[we, wy]
    endproc
endproc
    
```



```

v0 : hide w, w' in BufI[we, w] |[w]| BufI[w, w'] |[w']| BufI[w', wy]
v1 : hide w, w' in w *; BufI[we, w] |[w]| BufI[w, w'] |[w']| BufI[w', wy]
v2 : hide w, w' in BufI[we, w] |[w]| w' *; BufI[w, w'] |[w']| BufI[w', wy]
v3 : hide w, w' in w *; BufI[we, w] |[w]| w' *; BufI[w, w'] |[w']| BufI[w', wy]
v4 : hide w, w' in BufI[we, w] |[w]| BufI[w, w'] |[w']| wy *; BufI[w', wy]
v5 : hide w, w' in w *; BufI[we, w] |[w]| BufI[w, w'] |[w']| wy *; BufI[w', wy]
v6 : hide w, w' in BufI[we, w] |[w]| w' *; BufI[w, w'] |[w']| wy *; BufI[w', wy]
v7 : hide w, w' in BufI[we, w] |[w]| BufI[w, w'] |[w']| BufI[w', wy]

```

Zwróćmy uwagę, że rekursywne zagnieżdżenie operatora przesłonięcia wymaga przemianowania przesłanianych bramek, stąd użycie konstrukcji:

```
hide w, w' in BufI[we, w] |[w]| BufI[w, w'] |[w']| BufI[w', wy]
```

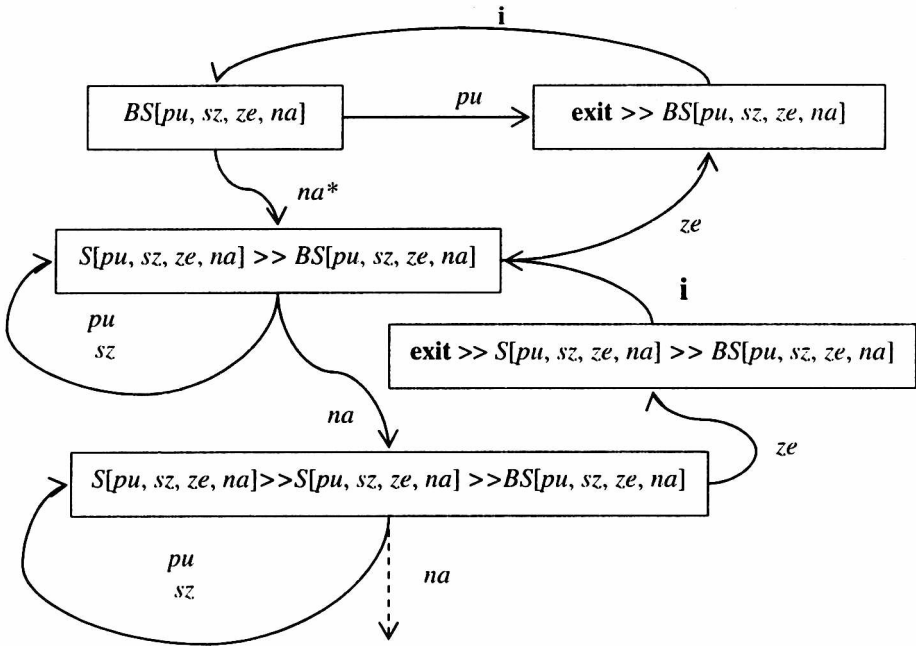
Kolejny przykład jest interesujący z dwóch względów. Po pierwsze – stos był zdefiniowany w poprzednim rozdziale jako typ danych, natomiast tu jest zdefiniowany jako proces – jest to ilustracja ogólniejszego stwierdzenia, że typ danych może być modelem dowolnego programu sekwencyjnego. Po drugie – przykład pokazuje, że zredukowany graf tranzycji może być nieskończony.

```

process BStos[pusty, szczyt, ze_stosu, na_stos] : noexit :=
  ( pusty ! true; exit
    [] na_stos ? x : elem; Stos[pusty, szczyt, ze_stosu, na_stos](x)
  ) >> BStos[pusty, szczyt, ze_stosu, na_stos]
where
  process Stos[pusty, szczyt, ze_stosu, na_stos](x : elem) : exit :=
    pusty ! false; Stos[pusty, szczyt, ze_stosu, na_stos](x)
  [] szczyt ! x; Stos[pusty, szczyt, ze_stosu, na_stos](x)
  [] ze_stosu: exit
  [] na_stos ? y : elem;
    Stos[pusty, szczyt, ze_stosu, na_stos](y)
  >> Stos[pusty, szczyt, ze_stosu, na_stos](x)
endproc
endproc

```

Przykład stosu reprezentuje sytuację, kiedy zredukowany graf tranzycji nie jest skończony (rys. 5.4). W celu skrócenia zapisu wyrażeń behawioralnych na rysunku zamiast pełnych nazw procesów i bramek użyto tylko ich skrótów.



Rys. 5.4. Nieskończony zredukowany graf tranzycji

Przyczyną nieskończoności zredukowanego grafu tranzycji jest rekursja, w której występują operatory złożenia równoległego złożenia aktywującego lub deaktywującego.

Wystarczającymi, ale niekoniecznymi, warunkami skończoności są:

- Jeżeli wyrażenie behawioralne $B_1 \parallel \dots \parallel B_2$ jest podwyrażeniem treści procesu p , to ani B_1 , ani B_2 nie zawierają wywołania procesu p lub wywołań innych procesów, które pośrednio lub bezpośrednio wywołują proces p .
- Jeżeli wyrażenie behawioralne $B_1 [> B_2$ lub $B_1 >> B_2$ jest podwyrażeniem treści procesu p , to B_1 nie zawiera wywołania procesu p lub wywołań innych procesów, które pośrednio lub bezpośrednio wywołują proces p .

5.5. Strukturyzacja specyfikacji

Specyfikacja behawioralna jest zbiorem definicji procesów z wyróżnioną definicją procesu początkowego. Złożone specyfikacje behawioralne są strukturalizowane. Przyjęta konwencja strukturalizacji jest odzwierciedleniem podejścia zstępującego – od ogółu do szczegółu. Znajduje to odzwierciedlenie w następującym szkieletie specyfikacji:

```

specification   nazwa_specyfikacji[lista_bramek_formalnych]
                  (lista_formalnych_parametrów_wartościowych) :
                  funkcjonalność

    definicja_typu
    .....
    definicja_typu
behavior
    wyrażenie_behawioralne
where
    definicja_procesu
    .....
    definicja_procesu
endspec
    
```

Nazwą wyróżnionego procesu początkowego jest *nazwa_specyfikacji*, jego parametrami są *lista_bramek_formalnych* oraz *lista_parametrów_wartościowych*, a jego treścią jest *wyrażenie_behawioralne*. Dodatkowo specyfikację, podobnie jak każdy proces, charakteryzuje *funkcjonalność*, określająca sposób jego kończenia się.

Definicje typów danych, opisane w poprzednim rozdziale, są typami globalnymi specyfikacji, które mogą być wykorzystywane przez procesy składowe specyfikacji. Typy mogą być typami standardowymi, pochodzącymi z biblioteki typów. W przypadku użycia typów standardowych $typ_1, \dots, typ_n$ ich deklaracja ma postać:

```

library typ1, ..., typn endlib
    
```

Po słowie kluczowym **where** występuje lista definicji procesów. Szkieletowa postać definicji procesów przedstawia się następująco:

```

process nazwa_procesu [lista_bramek_formalnych]
          (lista_formalnych_parametrów_wartościowych) :
          funkcjonalność :=
          wyrażenie_behawioralne
where
    definicje_lokalne
endproc
    
```

Każdy proces może mieć opcjonalnie własne *definicje_lokalne*, które są definicjami typów lub procesów wewnętrznych.

Każda specyfikacja strukturalna może zostać przetransformowana do postaci kanonicznej. Proces takiej transformacji, nazywany spłaszczaniem specyfikacji, jest przed-

stawiony w [ISO 8807]. Spłaszczanie polega na takiej zamianie nazw bramek i procesów, aby stały się one unikalne w ramach całej specyfikacji, inaczej: chodzi o przeniesienie wszystkich definicji lokalnych na jeden poziom globalny. W podobny sposób można dokonać spłaszczenia typów, to znaczy przeniesienia ich definicji na jeden globalny poziom. Semantyka dowolnej specyfikacji strukturalnej jest równoważna semantyce jej transformacji do postaci kanonicznej.

Przykład złożonej specyfikacji opisuje funkcjonowanie automatu sprzedaży biletów parkingowych. Scenariusz pracy urządzenia jest następujący: Działanie automatu składa się z dwóch faz: płacenia (proces *Płacenie*) i wydania biletu (proces *Wydanie*). W pierwszej fazie użytkownik deklaruje wartość biletu, który chce uzyskać, wprowadzając odpowiednią wartość przez bramkę *kwota* (akcja *kwota?k : nat*). Pomijamy tu szczegóły wpisywania wprowadzonej wartości. Następnie następuje płacenie, które polega na wrzucaniu kolejnych monet, aż do momentu, gdy ich wartość osiągnie przynajmniej zadeklarowaną wartość biletu (akcja *moneta?m : nat*). Przed osiągnięciem tego momentu użytkownik może się wycofać z transakcji przez interakcję na bramce *wycofaj* (akcja *wycofaj*, której nie towarzyszy wymiana danych), po czym następuje przejście do drugiej fazy – do zwrotu wprowadzonej wartości (akcja *reszta!r*). Jeżeli użytkownik nie wycofa się i wpłaci pełną kwotę, także następuje przejście do drugiej fazy – wydania biletu i ewentualnej reszty.

specification *Parkomat*[*kwota*, *moneta*, *wycofaj*, *reszta*, *bilet*, *wyłącz*] : **noexit**
behavior

```
(
    Płacenie [kwota, moneta, wycofaj]
    >> Wydanie[bilet, reszta]
    >> Parkomat[kwota, moneta, wycofaj, reszta, bilet, wyłącz]
) [> (i; stop [] wyłącz; stop)
```

where

process *Płacenie* [*kwota*, *moneta*, *wycofaj*] : **exit**(*nat*) :=

kwota ? *k* : *nat*;

Pobierz[*moneta*, *wycofaj*](*k*, 0)

where

process *Pobierz*[*moneta*, *wycofaj*](*k* : *nat*, *s* : *nat*) : **exit**(*nat*, *nat*) :=

([*s* < *k*] -> *moneta* ? *m* : *nat*; *Pobierz*[*moneta*, *wycofaj*](*k*, *s* + *m*)

[] [*s* = *k*] -> **exit**(*k*, 0)

[] [*not*(*s* < *k*)] -> **exit**(*k*, *s* - *k*)

[] *wycofaj*; **exit**(0, *s*)

endproc(**Pobierz**)

endproc(**Płacenie**)

process *Wydanie*[*bilet*, *reszta*] : **exit** :=

```

let  $k : nat, s : nat$  in
  [ $k > 0$  and  $r > 0$ ] -> bilet !  $k$ ; reszta !  $r$ ; exit
  [] [ $k > 0$  and  $r = 0$ ] -> bilet !  $k$ ; exit
  [] [ $k = 0$  and  $r > 0$ ] -> reszta !  $r$ ; exit
endproc (*Wydanie*)
endspec (*Parkomat*)

```

Przedstawiony model uwzględnia dodatkowo możliwość przerywania pracy parkomatu albo przez wyłączenie – akcja *wyłącz*, albo z powodu awarii – akcja wewnętrzna *i*.

Kolejny przykład jest specyfikacją systemu sortującego. Działanie tego systemu opiera się na algorytmie sortowania przez wtłaczanie. Koncepcja algorytmu jest następująca: Niech

$$c_0 = \langle a_1, \dots, a_n \rangle$$

będzie nieposortowanym ciągiem elementów – dalej zakładamy, że są to liczby naturalne, reprezentowane typem standardowym *NaturalNumber*. Ciąg c_0 będziemy przekształcać kolejno w ciągi $c_1, \dots, c_n$. Ciąg c_k , dla $k = 1, \dots, n$, stanowi posortowany ciąg początkowych k elementów, czyli $a_1, \dots, a_k$, ciągu c_0 . Ciąg c_n będzie zatem posortowanym ciągiem wszystkich elementów ciągu c_0 . Przekształcanie ciągów $c_1, \dots, c_n$ odbywa się według następujących zasad:

- Ciąg $c_1 =_{\text{def}} \langle a_1 \rangle$.
- Jeżeli utworzono ciąg $c_k = \langle a_{i_1}, \dots, a_{i_k} \rangle$, gdzie $i_1, \dots, i_k$ jest permutacją $1, \dots, k$, to bierze się element a_{k+1} z ciągu c_0 i następnie, przeglądając ciąg c_k od pozycji pierwszej do ostatniej, znajduje się w nim taką pozycję m , że

$$a_{i_{m-1}} \leq a_{k+1} \leq a_{i_m}$$

i ciąg c_{k+1} wyznacza się jako

$$c_{k+1} = \langle a_{i_1}, \dots, a_{i_{m-1}}, a_{k+1}, a_{i_m}, \dots, a_{i_k} \rangle$$

System sortujący opisywany w LOTOSie działa w taki sposób, że przez jedną ze swych bramek wczytuje się kolejne liczby nieposortowanego ciągu, a następnie – przez drugą bramkę – wyprowadza się ciąg posortowany. Parametrem specyfikacji jest liczba określająca długość sortowanego ciągu

```

specification   Sortowanie[we, wy]( $n : Nat$ ) : noexit
  library Boolean, NaturalNumber endlb
behavior
  Wtłaczanie[we, wy]( $n, 0$ )
where
  process Wtłaczanie[we, wy]( $i : Nat, j : Nat$ ) : noexit :=

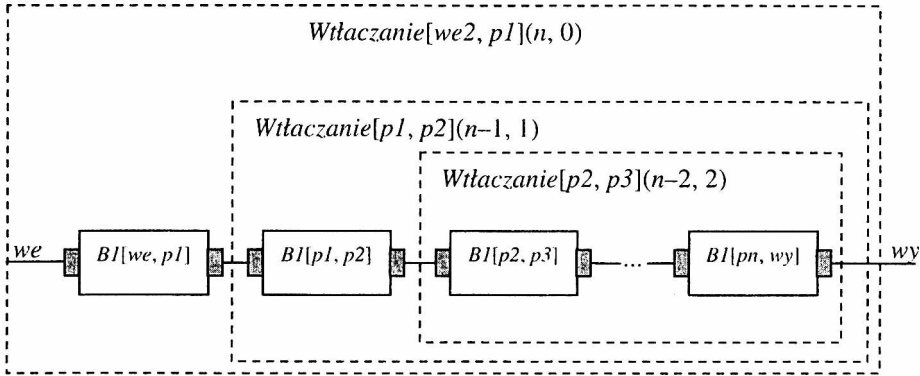
```

```

    [Succ(0) lt i] -> hide p in
        Buf1[we, p](i, j) |[p]| Właczanie[p, wy](Pred(i), Succ(j))
    [] [Succ(0) eq i] -> Buf1[we, wy](i, j)
endproc
where
    process Buf1[we, wy](i : Nat, j : Nat) : noexit :=
        (* Komentarz: instancja procesu odpowiada ustalonej pozycji w sortowanym
           ciągu i służy do wprowadzenia pierwszego elementu na tę pozycję. Para-
           metr i oznacza numer pozycji liczonej od lewej do prawej strony (rys. 5.5);
           parametr j jest dopełnieniem i do liczby n, będącej długością sortowanego
           ciągu. *)
        we ? x:Nat ; Buf2[we, wy](x, Pred(i), j)
    endproc
where
    process Buf2[we, wy](a : Nat, i : Nat, j : Nat) : noexit :=
        (* instancja procesu służy do porównywania elementu wczytanego od
           lewego sąsiada – instancji procesu Buf1 – z elementem zapamięta-
           nym i przesłania większego z nich do sąsiada po prawej stronie
           (rys. 5.5) *)
        [0 lt i] -> we ? x : Nat ;
        ( [x le a] -> wy!a ; Buf2[we, wy](x, Pred(i), j)
          [] [a le x] -> wy!x ; Buf2[we, wy](a, Pred(i), j)
        )
        [] [i eq 0] -> Buf3[we, wy](a, j)
    endproc
where
    process Buf3[we, wy](a : Nat, j : Nat) : noexit :=
        (* instancja procesu służy do wyprowadzania elementów posor-
           towanego ciągu do sąsiada po prawej stronie (rys. 5.5) *)
        [0 lt j] -> wy ! a ; we?x:Nat; Buf3[we, wy](x, Pred(j))
        [] [j eq 0] -> wy!a ; stop
    endproc
endproc
endproc
endproc
endspec

```

Na rysunku 5.5 przedstawiono strukturę systemu sortującego wyrażoną przez powiązanie instancji procesów *Buf1*. Warto zwrócić uwagę na to, że w przedstawionej postaci system daje możliwość równoległego wykonywania wielu operacji porównujących – wynika to z równoległego złożenia instancji procesów.



Rys. 5.5. Struktura systemu sortującego

5.6. Uwagi końcowe

W tym rozdziale przedstawiono, w sposób tylko częściowo sformalizowany, opis składni i semantyki języka LOTOS. Pełny opis zawiera dokument standaryzacyjny [ISO 8870]. Wcześniejsza jego wersja jest zawarta w opracowaniach: [Brinksma 1988], [Bolognesi, Brinksma 1989]. Dokładny opis projektu, w ramach którego prowadzono prace nad LOTOSem, przedstawia książka [van Eijk, Vissers, Diaz 1989].

Przykłady ilustrujące zostały zaczerpnięte między innymi z prac [Huzar, Kuźniarz 1990a, 1990b, 1990c].

Język LOTOS był punktem odniesienia do definiowania innych języków specyfikacji formalnych. Przykładem próby definiowania języka ukierunkowanego na specyfikację systemów rozproszonych zgodnych ze standardami [ISO/IEC 10746-2, 1995], [ISO/IEC 10746-3, 1995] jest rozprawa doktorska [Hnatkowska 1998]. Zdefiniowany język O-LOTOS ma wprawdzie taką samą siłę ekspresji jak LOTOS (abstrakcyjna implementacja jest zdefiniowana jako transformacja języka O-LOTOS na język LOTOS), ale umożliwia zwiększenie efektywności fazy analizy wytwarzania oprogramowania, a uzyskana specyfikacja może być wykorzystana podczas projektowania interfejsu użytkowego.

LOTOS był również stosowany do definiowania semantyki innych języków. Przykładem jest praca [Hnatkowska, Huzar 2001], przedstawiająca semantykę diagramów stanów języka UML [UML 1.3, 1999] w postaci wyrażeń behawioralnych LOTOSa.

Zastosowaniom LOTOSa towarzyszył rozwój metodyk i narzędzi programistycznych. Głównymi europejskimi ośrodkami uczestniczącymi w tego rodzaju przedsięwzięciach były uniwersytety w Madrycie i Liège, a przede wszystkim INRIA Rhône-Alpes (Institut National de Recherche en Informatique et en Automatique) – instytut, w którym opracowano najbogatsze środowisko (zob. rozdz. 7.) wspomagające specyfiko-

wanie i badanie własności specyfikacji w LOTOSie. Poza Europą ośrodkiem zajmującym się LOTOSem jest uniwersytet w Ottawie. Informacje o różnych ośrodkach związanych z LOTOSem można uzyskać między innymi na stronach internetowych:

<http://www.inrialpes.fr/vasy/elotos/>

<http://www-run.montefiore.ulg.ac.be/Research/Topics/index.php?topic=Lotos>

<http://www.cs.stir.ac.uk/~kjt/research/well/>

Podobnie jak każdy język, LOTOS ulegał ewolucji. Głównymi przyczynami ewolucji były krytyka stosowanego podejścia do typów danych (skomentowana w zakończeniu poprzedniego rozdziału) oraz potrzeba specyfikowania ograniczeń czasowych. Od 1997 roku prowadzono prace, które w 2001 roku zakończyły się ustanowieniem nowego standardu E-LOTOS [ISO/IEC 15437:2001].

6. Specyfikacja usług i protokołów sieciowych

6.1. Elementy modelu referencyjnego ISO/OSI

Głównym obszarem zastosowań języka LOTOS, opracowanego w ramach ISO [ISO 8807] jako jedna z formalnych technik opisu standardów sieciowych, jest specyfikacja usług i protokołów.

Sieć komputerowa jest systemem. Powiązania pomiędzy elementami systemu tworzą jego strukturę. Z systemem jest związany cel funkcjonowania, z tego względu system jest niepodzielną całością. Oznacza to, że każdy, dowolnie wyodrębniony podzbiór elementów ma wpływ na funkcjonowanie systemu, ale żaden z podzbiorów nie ma wyłącznego wpływu, czyli usunięcie dowolnego elementu nie pozwala pozostałym elementom na pełną realizację celu systemu. Elementami składowymi złożonych systemów mogą być inne systemy – jego podsystemy.

Struktura sieci komputerowej jest wyznaczona przez zbiór komputerów połączonych ze sobą łączami komunikacyjnymi. Składowe komputery, ze względu na ich wewnętrzną złożoność, określa się również jako systemy – podsystemy sieci komputerowej. Celem funkcjonowania sieci komputerowej jest umożliwienie użytkownikom dostępu do usług informacyjnych i obliczeniowych, oferowanych przez współpracujące systemy komputerowe.

Możliwość właściwego funkcjonowania sieci komputerowej zapewnia zbiór standardów określających zasady, które powinny być akceptowane przez wszystkie współpracujące systemy komputerowe. Podstawowym odniesieniem dla wszystkich tworzonych standardów jest model referencyjny ISO/OSI (*Open Systems Interconnection*) [ISO 7498], nazywany też modelem architektury otwartych systemów komputerowych. Otwartość jest tu rozumiana jako gotowość do współpracy systemu komputerowego z każdym innym systemem, który spełnia określone wymagania. Istota modelu referencyjnego ISO/OSI polega na przyjęciu standardowej dekompozycji funkcji systemów otwartych. Poniżej omówiono krótko model ISO/OSI, gdyż przyjęte tu ogólne zasady są akceptowane także przez inne modele [Comer 1997].

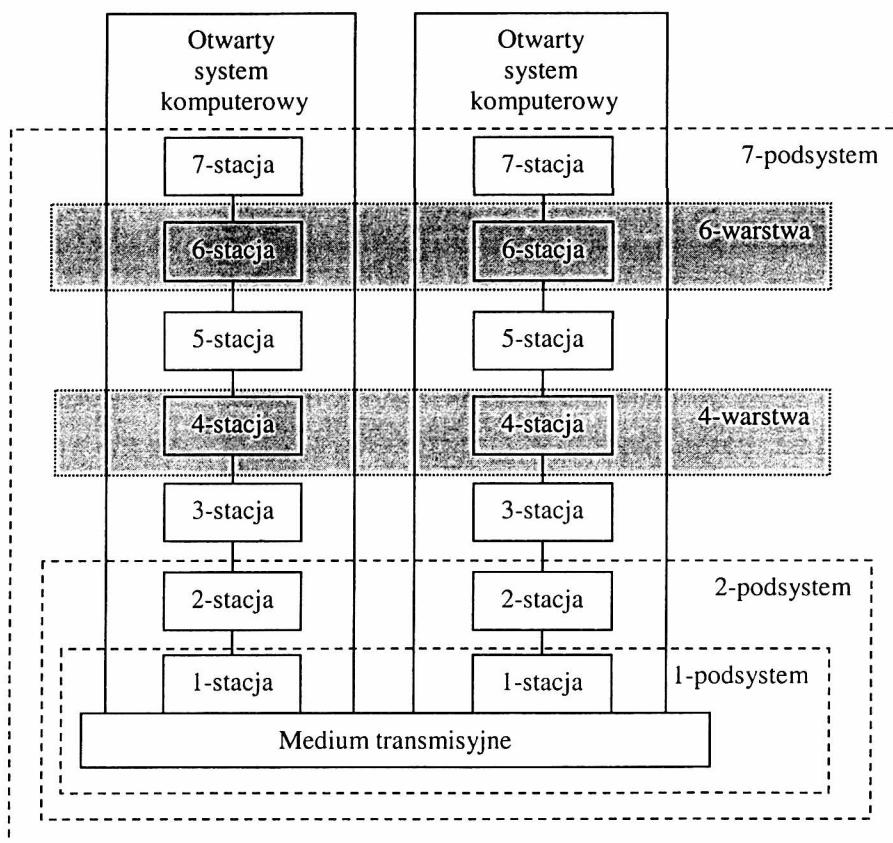
Dekompozycja funkcji polega na wprowadzeniu liniowo uporządkowanego zestawu *warstw funkcjonalnych* (usługowych) systemów otwartych. Model OSI wyróżnia siedem takich warstw, nazywanych – poczynając od najniższej do najwyższej – warstwą: fizyczną, liniową, sieciową, transportową, sesyjną, prezentacyjną i aplikacyjną (rys. 6.1).

Warstwy pozostają ze sobą w relacji *usługodawca–usługobiorca*. Warstwa wyższa jest usługobiorcą usług sąsiadującej warstwy niższej – usługodawcą. Warstwa najniższa nie korzysta już z usług innej warstwy, a warstwa najwyższa dostarcza swych usług użytkownikom, którzy stanowią element otoczenia systemu.

Usługi i funkcje wykonywane w obrębie poszczególnych warstw szczegółowo zostały omówione między innymi w pracach [Bilski, Dubielewicz 1991, 1993].

Usługi danej warstwy są definiowane przez zbiór poleceń usługowych, za pomocą których warstwa wyższa komunikuje się z warstwą niższą. Polecenia są abstrakcyjne w tym sensie, że nie określają one sposobu implementacji, są tylko specyfikacjami pewnych funkcjonalności (usług).

Warstwę reprezentuje zbiór *stacji partnerskich* (*peer entities*). Stacje te, realizując usługi żądane od warstwy wyższej, współpracują ze sobą poprzez wymianę danych. Wymiany tej nie dokonują bezpośrednio, lecz za pośrednictwem obiektów partnerskich w niższych warstwach (rys. 6.1).

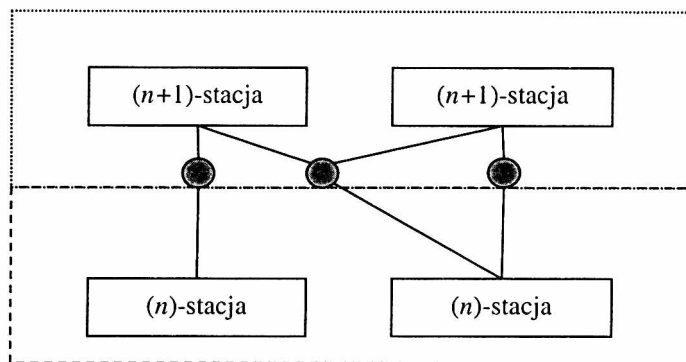


Rys. 6.1. Architektura warstwowa modelu ISO/OSI

Zasady współpracy stacji partnerskich określa się mianem *protokołu* danej warstwy. Protokół można uważać za abstrakcyjną implementację usługi danej warstwy.

Zbiór n dolnych warstw, dla $n = 1, \dots, 7$, stanowi n -podsystem sieci komputerowej. Stwierdzenie, że stacja z warstwy $n+1$ korzysta z usług warstwy n , oznacza w istocie, że stacja korzysta z usług n -podsystemu. Realizacja polecenia skierowanego przez stację z danej warstwy może pociągać zaangażowanie stacji w warstwach niższych, zarówno po stronie systemu otwartego, z którego zostało skierowane polecenie, jak i po stronie systemu otwartego, z którym współpracuje dana stacja.

Strukturę powiązań pomiędzy stacjami danego systemu otwartego zilustrowano na rysunku 6.2.

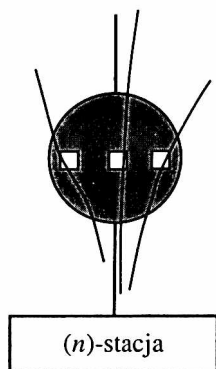


Rys. 6.2. Powiązania stacji w sąsiednich warstwach

● Punkt dostępu do usługi (SAP)

Dany system otwarty może w danej warstwie mieć wiele stacji partnerskich. Należące do jednej warstwy stacje mogą współpracować ze sobą. Zasadniczy schemat współpracy polega na tym, że stacje ustalają ze sobą połączenie na pewien okres, w trakcie którego mogą wymieniać pomiędzy sobą ustalone dane. W połączeniu najczęściej uczestniczą dwaj partnerzy, z których jeden jest inicjatorem, a drugi respondentem połączenia, ale możliwe są także połączenia z udziałem wielu partnerów. Każde połączenie ma swój identyfikator. W trakcie trwania połączenia wyróżnia się trzy fazy: faza nawiązania połączenia, podczas której ustala się warunki współpracy, fazę zasadniczą, w której prowadzi się wymianę danych i fazę rozłączenia, kiedy stacje potwierdzają zakończenie współpracy. Połączenie jest identyfikowane przez zbiór identyfikatorów końców połączeń CEP (*Connection End Point Identifier*). Współpraca stacji może się też odbywać bez ustanawiania połączeń i polega na przesyłaniu bez zapowiedzi danych od danej stacji do innej o wskazanym adresie.

Stacje partnerskie danej warstwy nie komunikują się bezpośrednio, lecz za pośrednictwem stacji warstw niższych. Stacje sąsiadujących warstw, w tym samym systemie otwartym, są ze sobą powiązane za pośrednictwem punktów dostępu do usług SAP (*Service Access Point*). Identyfikator danego punktu dostępu do usług n warstwy nazywa się n -adresem. Punkt dostępu jest związany dokładnie z jedną n -stacją i może



□ Końcowy punkt połączenia (CEP)

być związany z wieloma $(n+1)$ -stacjami. Oznacza to, że pojedynczy punkt dostępu do usługi może służyć do jednoczesnego przekazywania wielu strumieni poleceń, a pojedyncza (n) -stacja może być zaangażowana w jednoczesne prowadzenie komunikacji pomiędzy wieloma $(n+1)$ -stacjami, czyli utrzymywanie (n) -połączeń (zob. rys. 6.3).

Rys. 6.3. Końcowe punkty połączeń

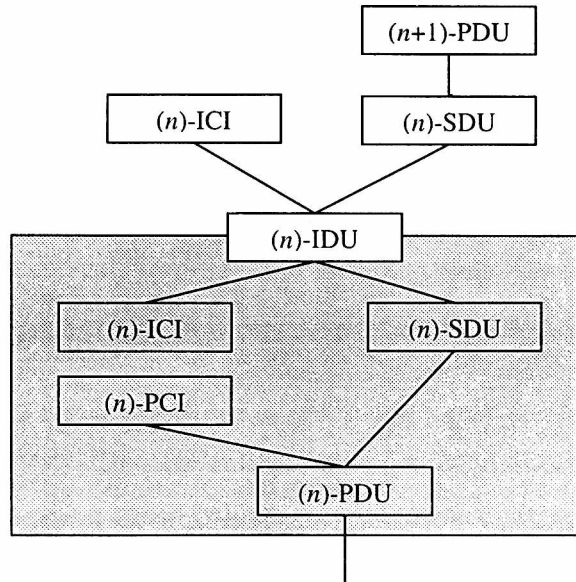
dla $(n+1)$ -stacji w punkcie dostępu do usługi

$(n+1)$ -stacja kieruje polecenia żądania do (n) -stacji przez wybrany punkt dostępu do usługi. Polecenia mają nazwę i ewentualne parametry. (n) -stacja przyjmuje polecenia żądania i, po ich zrealizowaniu, przekazuje poleceniem zwrotnym informację o wykonaniu lub niewykonaniu polecenia żądania.

Opis usług (n) -warstwy polega więc – jak określono to wyżej – na zestawieniu poleceń żądań i poleceń zwrotnych, jakie mogą być wymieniane pomiędzy $(n+1)$ -stacją a (n) -stacją, oraz na podaniu dopuszczalnych ciągów wymiany tych poleceń.

Określenie protokołu (n) -warstwy – jako abstrakcyjnej implementacji usług – wymaga, w pierwszej kolejności, określenia świadczonych usług, czyli (n) -usług oraz wykorzystywanych usług, czyli $(n-1)$ -usług. Następnie wymaga definicji komunikatów protokołowych (PDU – *Protocol Data Unit*), czyli danych, które wymieniają pomiędzy sobą stacje (n) -warstwy, oraz reguł transformacji tych komunikatów na polecenia usługowe do warstwy niższej. Wymaga wreszcie określenia dopuszczalnych ciągów wymian komunikatów protokołowych oraz akcji wykonywanych przez (n) -stacje.

$(n+1)$ -stacje wymieniają pomiędzy sobą komunikaty protokołu (n) -warstwy, dalej w skrócie oznaczane jako $(n+1)$ -PDU. Komunikaty te są wymieniane za pośrednictwem (n) -stacji, do której $(n+1)$ -stacja kieruje polecenia w postaci ustalonej przez interfejs pomiędzy stacjami. Komunikaty interfejsu – (n) -IDU (*Interface Data Unit*) – składają się z części sterującej – (n) -ICI (*Interface Control Information*) oraz części usługowej – (n) -SDU (*Service Data Unit*), którą jest jednostka protokołowa $(n+1)$ -PDU (zob. rys. 6.4). Komunikat interfejsu odebrany przez (n) -stację jest dalej przekształcany, co polega – w pierwszej kolejności – na wyodrębnieniu części sterującej i serwisowej, a następnie na przygotowaniu komunikatu protokołu (n) -PDU. Komunikat (n) -PDU przenosi, jako swoją część, komunikat $(n+1)$ -PDU oraz informację sterującą protokołu (n) -warstwy – (n) -PCI (*Protocol Control Information*).



Rys. 6.4. Odzworowanie jednostek danych pomiędzy dwiema sąsiednimi warstwami

6.2. Reprezentacja modelu referencyjnego

Specyfikacje usług i protokołów sieciowych w LOTOSie mają pewne wspólne własności. W tym podrozdziale przedstawiono nieformalnie tylko uproszczone zasady reprezentacji elementów statycznych i dynamicznych modelu ISO/OSI w języku LOTOS.

Granice pomiędzy warstwami modelu OSI są w LOTOSie reprezentowane przez pojedynczą bramkę. Przyjęto następującą konwencję symboli bramek reprezentujących poszczególne warstwy:

- warstwa fizyczna – ph
- warstwa liniowa – dl
- warstwa sieciowa – n
- warstwa transportowa – t
- warstwa sesyjna – s
- warstwa prezentacyjna – p
- warstwa aplikacyjna – a

Wszystkie akcje komunikacyjne z udziałem punktu dostępu do usługi (n) -SAP, o identyfikatorze *adres* rodzaju *IdentyfikatorSAP*, są odwzorowywane w klasę akcji języka LOTOS o postaci:

$n!adres \dots$

lub

$n?adr : IdentyfikatorSAP \dots$

Rodzaj *Identyfikator* powinien być elementem abstrakcyjnego typu danych gwarantującego co najmniej rozróżnialność adresów. Na przykład definicja odpowiedniego typu może mieć postać

```

type Identifier is Boolean
sorts IdentifierSAP
opns someAddress : -> IdentifierSAP
      anotherAddress : IdentifierSAP -> IdentifierSAP
      _eq_, _ne_ : IdentifierSAP , IdentifierSAP -> Bool
eqns forall x, y : IdentifierSAP ofsort Bool
      x eq y = true ;
      someAddress eq anotherAddress(x) = false ;
      anotherAddress(x) eq someAddress = false ;
      anotherAddress(x) eq anotherAddress(y) = x eq y ;
      x ne y = not(x eq y) ;

endtype

```

Podana specyfikacja wskazuje tylko istotne własności, które powinien mieć typ. Mianowicie operacja zeroargumentowa *someAddress* wyznacza pewien ustalony identyfikator, a operacja jednoargumentowa *anotherAddress* jest generatorem nowych, unikatowych identyfikatorów.

Końcowy punkt połączenia (*n*)-CEP, o identyfikatorze *połączenie* rodzaju *IdentyfikatorCEP*, związany z powyższym punktem dostępu, a dokładniej wszystkie akcje z udziałem tego punktu, są odwzorowywane w klasę akcji

n!adres!idCEP ...

lub

n?adr : IdentyfikatorSAP?id : IdentyfikatorCEP...

Do rodzaju *IdentyfikatorCEP* odnoszą się takie same postulaty jak do rodzaju *IdentyfikatorSAP*.

Polecenia usługowe odwzorowuje się w akcje postaci

n!adres!idCEP!polec(...)

lub

n?adr : IdentyfikatorSAP?id : IdentyfikatorCEP?polec : ServicePrimitive

gdzie *polec(...)* jest termem rodzaju *ServicePrimitive*. Polecenia usługowe są różne dla stacji różnych warstw. W celu wyjaśnienia rozumienia termów reprezentujących polecenia rozpatrzmy przykład podzbioru poleceń usługowych, które służą do nawiązywania połączenia pomiędzy stacjami warstwy sesji. Specyfikacja odpowiedniego typu ma postać

```

type   SessionServiceConnectionPrimitive is
        SessionAddress, SessionReqms, SessionData

sorts  SSCP

opns   SCONreq, SCONind, SCONcnf, SCONrsp :
        SAddress, SData, SReqms -> SSCP
        SCadr : SSCP -> SAddress
        SCdata : SSCP -> SData
        SCreqms : SSCP -> SReqms

eqns   forall adr : SAddress, data : SData, reqm : SReqms
        ofsort SAddress
        SCadr(SCONreq(adr, data, reqms)) = adr ;
        SCadr(SCONind(adr, data, reqms)) = adr ;
        SCadr(SCONcnf(adr, data, reqms)) = adr ;
        SCadr(SCONrsp(adr, data, reqms)) = adr ;
        ofsort SData
        SCdata(SCONreq(adr, data, reqms)) = data ;
        SCdata(SCONind(adr, data, reqms)) = data ;
        SCdata(SCONcnf(adr, data, reqms)) = data ;
        SCdata(SCONrsp(adr, data, reqms)) = data ;
        ofsort SReqms
        SCreqms(SCONreq(adr, data, reqms)) = reqms ;
        SCreqms(SCONind(adr, data, reqms)) = reqms ;
        SCreqms(SCONcnf(adr, data, reqms)) = reqms ;
        SCreqms(SCONrsp(adr, data, reqms)) = reqms ;

endtype

```

Przedstawiony typ abstrakcyjny *SessionServiceConnectionPrimitive* wraz z pozostałymi typami, których jest rozszerzeniem, pochodzą ze standardu ISO dotyczącego warstwy sesji. Typy *SessionAddress*, *SessionReqms*, *SessionData*, które nie są tu definiowane, służą do zdefiniowania parametrów poleceń usługowych *SCONreq*, *SCONind*, *SCONcnf*, *SCONrsp*. Typy te reprezentują odpowiednio adresy końców ustanawianych połączeń sesyjnych, danych wymienianych podczas połączenia sesyjnego oraz danych sterujących do ustanawiania parametrów połączenia sesyjnego. Wprowadzone operacje służą tylko do selekcji elementów poleceń usługowych.

Usługi (n)-warstwy przedstawia się, podając opis funkcjonowania (n)-stacji. (n)-stacja – pośrednicząc pomiędzy ($n+1$)-warstwą a ($n-1$)-warstwą – przekształca (n)-polecenia usługowe na ($n-1$)-polecenia usługowe. Specyfikację zachowania (n)-stacji przedstawia się w LOTOSie jako proces, który powinien spełniać trzy rodzaje ograniczeń:

- ograniczenie dopuszczalnych sekwencji wymian poleceń usługowych pomiędzy (n) -stacją a $(n+1)$ -warstwą – *G-ograniczenie*,
- ograniczenie dopuszczalnych sekwencji wymian poleceń usługowych pomiędzy (n) -stacją a $(n-1)$ -warstwą – *D-ograniczenie*,
- ograniczenie dopuszczalnych sekwencji wymian poleceń usługowych pomiędzy $(n+1)$ -warstwą i $(n-1)$ -warstwą – *GD-ograniczenie*.

Zakładając, że każde z wymienionych ograniczeń jest reprezentowane odpowiednim wyrażeniem behawioralnym, strukturę całego procesu można przedstawić w uproszczonej postaci

```

process  $(n)$ -stacja[ $g, d$ ]
    ( $GIdSAP : GIdentyfikatorSAP, DIdSAP : DIdentyfikatorSAP$ ) :
    noexit :=
     $G$ -ograniczenia[ $g$ ]( $GIdSAP : GIdentyfikatorSAP$ )
    |[ $g$ ]
     $GD$ -ograniczenia[ $g, d$ ]( $GIdSAP : GIdentyfikatorSAP,$ 
     $DIdSAP : DIdentyfikatorSAP$ )
    |[ $d$ ]
     $G$ -ograniczenia[ $d$ ]( $DIdSAP : DIdentyfikatorSAP$ )
endproc

```

gdzie: g, d są nazwami bramek, a $GIdentyfikatorSAP$ oraz $DIdentyfikatorSAP$ są rodzajami reprezentującymi zbiór identyfikatorów punktów dostępu do usługi odpowiednio warstwy n oraz $n-1$.

W uproszczeniu usługi (n) -warstwy reprezentuje instancja procesu o podanej wyżej definicji z dowolnym wyborem parametrów aktualnych. Możemy usługi te wyrazić w postaci definicji nowego procesu

```

process  $(n)$ -usługi[ $g$ ] : noexit :=
    hide  $d$  in
    choice  $GIdSAP : GIdentyfikatorSAP$  []
    choice  $DIdSAP : DIdentyfikatorSAP$  []  $(n)$ -stacja[ $g, d$ ]( $GIdSAP, DIdSAP$ )
endproc

```

Uproszczenie przedstawionego schematu stacji wiąże się z pozostaniem d jako bramki wewnętrznej, podczas gdy w standardowej postaci bramki takiej się nie używa. Odpowiedni przykład przedstawiono w następnym podrozdziale.

Protokół (n) -warstwy reprezentuje w LOTOSie wyrażenie, które jest złożeniem równoległym procesów reprezentujących przynajmniej dwie (n) -stacje oraz wyrażenia reprezentującego $(n-1)$ -usługi. Uproszczony schemat takiego wyrażenia ma postać

```

process (n)-protokół[g] : noexit :=
  hide d in
    ( (n)-stacja[g, d](GId1SAP, Did1SAP)
      |[]|
      (n)-stacja[g, d](GId2SAP, Did2SAP)
    )
    |[d]|
    (n-1)-usługi[d]
endproc

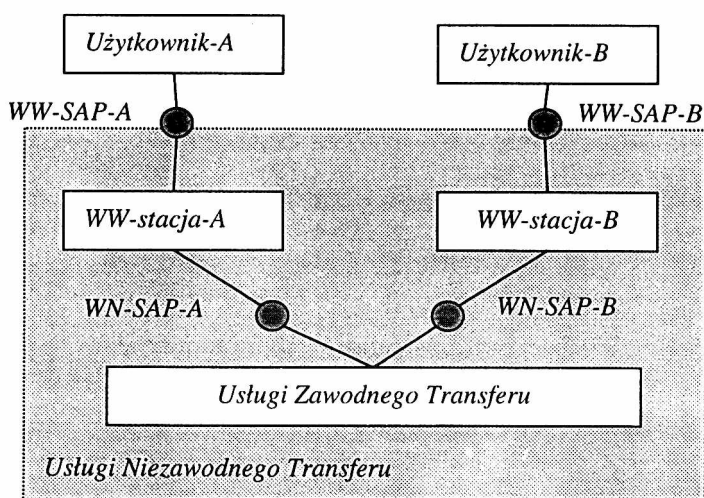
```

6.3. Przykłady specyfikacji

Podano dwa przykłady, które stanowią opis prostych usług komunikacyjnych, przy czym pierwszy ma charakter wprowadzający, a drugi nawiązuje do istniejącego standardu opisu usług warstwy transportowej.

Przykład 1

Opisujemy warstwę świadczącą usługi niezawodnej komunikacji, która opiera się na usługach warstwy dostarczającej zawodnych usług komunikacyjnych (zob. rys. 6.5).



Rys. 6.5. Architektura systemu transferu

Zakłada się, że w danej chwili pomiędzy użytkownikami może być prowadzona tylko jedna komunikacja.

Specyfikacja usług świadczonych przez warstwę wyższą WW

Warstwa ma za zadanie przesyłanie komunikatów, wysyłanych pomiędzy dwoma użytkownikami, nazywanymi *Użytkownik-A* oraz *Użytkownik-B*.

Dostęp do warstwy odbywa się przez przekazywanie poleceń usługowych na bramce *unzt* (usługi niezawodnego transferu). Specyfikacja typu określającego polecenia usługowe jest następująca:

```

type   PoleceniaWW is DaneWW, Boolean
sorts  PolWW
opns   weWW, wyWW : DaneWW -> PolWW
          zawartośćWW : PolWW -> DaneWW
          jestWeWW : PolWW -> Bool
          jestWyWW : PolWW -> Bool
eqns   forall dane : DaneWW
          ofsort DaneWW
            zawartośćWW(weWW(dane)) = dane;
            zawartośćWW(wyWW(dane)) = dane;
          ofsort Bool
            jestWeWW(weWW(daneWW)) = true;
            jestWeWW(wyWW(daneWW)) = false;
            jestWyWW(wyWW(daneWW)) = true;
            jestWyWW(weWW(daneWW)) = false;
endtype
```

gdzie *DaneWW* jest nazwą rodzaju i niezdefiniowanego tu typu określającego dane przesyłane w warstwie *WW*.

Operacja *weWW* służy do przekazania danych do przesłania, a *wyWW* – do odebrania przesłanych danych. Operacje *jestWeWW* oraz *jestWyWW* służą do rozpoznawania, czy dane polecenie usługowe jest operacją *weWW* czy *wyWW*.

Akcje wykonywane na tej bramce mogą mieć jedną z postaci:

```

unzt !idSAP !idCEP !polWW
unzt !idSAP !idCEP ?pol : PolWW [jestWeWW]
unzt !idSAP !idCEP ?pol : PolWW [jestWyWW]
```

gdzie:

idSAP jest identyfikatorem punktu dostępu do usług warstwy *WW*, rodzaju *WWIdentSAP*,

idCEP jest identyfikatorem połączenia w warstwie *WW*, rodzaju *WWIdentCEP*,

polWW jest poleceniem usługowym (termem generowanym w ramach typu *PoleceniaWW*),

[*jestWeWW*] oraz [*jestWyWW*] są opcjonalnymi warunkami ograniczającymi interakcje zachodzące na bramce *uznt*.

Świadczone usługi można wyspecyfikować za pomocą procesu

```
process  UsługiNiezwTran[unzt](idA : WWIdentSAP, idB : WWIdentSAP,
                                     idAB : WWIdentCEP) : noexit :=
    unzt!idA!idAB?we(komWW);
    unzt!idB!idAB?wy(komWW);
    UsługiNiezwTran[unzt](idA, idB, idAB)
endproc
```

Usługi warstwy wyższej polegają na niezawodnym transferze pomiędzy dwoma jej użytkownikami, mającymi dostęp do wskazanych punktów dostępu do usługi o identyfikatorach *idA* oraz *idB*, w ramach ustanowionego połączenia o identyfikatorze *idAB*. Pierwszy identyfikator *idA* wskazuje na użytkownika, który w transferze pełni rolę nadawcy, a drugi *idB* wskazuje na odbiorcę.

Specyfikacja wykorzystywanych usług warstwy niższej WN

Warstwa niższa dostarcza usług zawodnego transferu. Tak samo jak poprzednio, wyróżnia się komunikaty normalne i puste. Przesyłanie komunikatów za pomocą zawodnego medium może gubić normalne komunikaty, ale nie może ich powielać ani przekłamywać. Przesyłanie komunikatów pustych jest natomiast niezawodne. Warstwa niższa *WN* komunikuje się z warstwą wyższą *WW* przez bramkę *uzt*.

Dostęp do warstwy odbywa się przez przekazywanie poleceń usługowych na bramce *uzt* (usługi zawodnego transferu). Specyfikacja typu określającego polecenia usługowe jest następująca:

```
type    PoleceniaWN is DaneWN, DaneWW
sorts   PolWN
opns    pusty : -> DaneWW
          poprawny, zgubiony : -> DaneWN
          weWN, wyWN : DaneWW -> PolWN
          potwierdzenie : DaneWN -> PolWN
          zawartośćWN : PolWN -> DaneWN
eqns    forall daneWN : DaneWN, daneWW : DaneWW
          ofsort DaneWN
          zawartośćWN(potwierdzenie(daneWN)) = daneWN;
          zawartośćWN(potwierdzenie(daneWN)) = daneWN;
```

ofsort *DaneWW*

zawartośćWN(*weWN*(*daneWW*)) = *daneWW*;

zawartośćWN(*wyWN*(*daneWW*)) = *daneWW*;

ofsort *Bool*

jestWeWN(*weWN*(*daneWW*)) = *true*;

jestWeWN(*wyWN*(*daneWW*)) = *false*;

jestWyWN(*wyWN*(*daneWW*)) = *true*;

jestWyWN(*weWN*(*daneWW*)) = *false*;

endtype

Stała *pusty* służy do modelowania zgubionego komunikatu. Stałe *poprawny* i *zgubiony* są wartościami służącymi do potwierdzania, czy komunikaty zostały przesłane poprawnie, czy też zostały zgubione. Operacje *weWN* i *wyWN* mają znaczenie takie jak *weWW* i *wyWW*, z tym że odnoszą się do punktów dostępu do usług do warstwy niższej. Analogiczna uwaga odnosi się do operacji *zawartośćWN* oraz *jestWeWN* i *jestWyWN*.

Akcje wykonywane na bramce *uzt* mogą mieć postać analogiczną do akcji wykonywanych na bramce *unzt*.

Usługi zawodnego transferu można wyspecyfikować za pomocą procesu

process *UsługiZawTrans*[*uzt*](*idA* : *WNIdentSAP*, *idB* : *WNIdentSAP*,
idAB : *IdentCEP*)

: **noexit** :=

uzt !*idA* !*poł* ?*pol* : *PolWN* [*jestWeWN*(*pol*)];

(*i*; *uzt* !*idB* !*poł* !*wyWN*(*zawartość*(*pol*));

UsługiZawTrans[*uzt*](*idA*, *idB*, *idAB*)

[] *i*; *uzt* !*idB* !*poł* !*wyWN*(*pusty*);

UsługiZawTrans[*uzt*](*idA*, *idB*, *idAB*)

)

[] *uzt* !*idA* !*poł* !*weWN*(*pusty*);

uzt !*idB* !*poł* !*wyWN*(*pusty*);

UsługiZawTrans[*uzt*](*idA*, *idB*, *idAB*)

[] *uzt* !*idB* !*poł* ?*pol* : *PolWN* [*jestWeWN*(*pol*)];

(*i*; *uzt* !*idA* !*poł* !*wyWN*(*zawartość*(*pol*));

UsługiZawTrans[*uzt*](*idA*, *idB*, *idAB*)

[] *i*; *uzt* !*idA* !*poł* !*wyWN*(*pusty*);

UsługiZawTrans[*uzt*](*idA*, *idB*, *idAB*)

)

[] *uzt* !*idB* !*poł* !*weWN*(*pusty*);

```

    uzt!idA!poł!wyWN(pusty);
    UsługiZawTrans[uzt](idA, idB, idAB)

```

endproc

Zawodność transferu jest modelowana za pomocą akcji wewnętrznych i prefiksujących składowe behawioralnego wyrażenia wyboru.

Specyfikacja protokołu

Zadaniem protokołu jest zapewnienie niezawodnej transmisji komunikatu od nadawcy do odbiorcy.

Zdefiniowanie stacji WW wymaga określenia dwóch procesów pomocniczych. Określają one dwie role, jakie może pełnić stacja podczas świadczenia usługi transferu danych – może to być rola nadawcy albo rola odbiorcy

```

process Nadajnik[unzt, uzt](idA : WWIdentSAP, idB : WWIdentSAP,
                               idAB : WWIdentCEP) : noexit :=
    unzt!idA!idAB?pol : PolWW [jestWwWW];
    uzt!idB!idAB!wyWN(zawartośćWW(pol)) ;
    Potwierdzenie[uzt](idA, idB, idAB, zawartośćWW(pol))

where
    process Potwierdzenie[uzt](idA : WWIdentSAP, idB : WWIdentSAP,
                               idAB : WWIdentCEP, dane : DaneWW) : noexit :=
        uzt !idB !idAB ?pol : PolecenieWN [jestWeWN];
        ( [zawartośćWN(pol) = zgubiony] ->
            uzt!idB!idAB!przyjmij(zawartość(pol)) ;
            Potwierdzenie[uzt](idA, idB, idAB, kom)
        [] [zawartośćWN(pol) = poprawny] ->
            Nadajnik[unzt, uzt](idA, idB, idAB)
        )

endproc
endproc

```

Proces określający rolę odbiorcy ma postać

```

process Odbiornik[unzt, uzt](idA : WWIdentSAP, idB : WWIdentSAP,
                               idAB : WWIdentCEP) : noexit :=
    uzt !idA !idAB ?pol : PolWN [jestWyWN];
    ( [not(zawartośćWN(pol) = pusty)] ->
        unzt !idB !idAB !wyWW(zawartość(pol)) ;
        unzt !idA !idAB !wyWNj(poprawny) ;
    )

```

```

        Odbiornik[unzt, uztf](idA, idB, idAB)
[]      [zawartośćWN(pol) = pusty] ->
        unzt!idA!idAB!wyslij(zgubiony) ;
        Odbiornik[unzt, uztf](idA, idB, idAB)

```

endproc

Specyfikację stacji można przedstawić jako następujący proces

```

process WWstacja[unzt, uztf](idA : WWidentSAP, idB : WWidentSAP,
                               idAB : WWidentCEP) : noexit :=
    Odbiornik[unzt, uztf](idA, idB, idAB) [] Nadajnik[unzt, uztf](idA, idB, idAB)
where
    process Nadajnik[unzt, uztf](...) ...
    process Odbiornik[unzt, uztf](...) ...
endproc

```

Stacja w danej chwili może tylko wysyłać albo odbierać dane.

Specyfikacja protokołu ma postać

```

process WWprotokół[unzt](idA : WWidentSAP, idB : WWidentSAP,
                          idAB : WWidentCEP) : noexit :=
    hide uztf in
        WWstacja[unzt, uztf](idA, idB, idAB)
    [[unzt]] UsługiZawodnegoTran[unzt, uztf](idA, idB, idAB)
where
    process WWstacja[unzt, uztf](...) .....
    process UsługiZawodnegoTran[unzt, uztf](...) .....
endproc

```

Przykład 2

W przykładzie opisano szkielet usług transportowych. Pokazano strukturę usług – nawiązywanie połączenia, transfer danych, rozłączenie. Dla uproszczenia opisano tylko nazwy poleceń usługowych z pominięciem ich parametrów, z wyjątkiem poleceń dotyczących nawiązania połączenia transportowego (*TConInd*) oraz transferu danych (*TDataInd*). Typ definiujący transportowe polecenia usługowe jest określony jako suma mnogościowa typów reprezentujących różne kategorie poleceń:

```

type T-polecenie is TConReq, TConInd, TConResp, TConConf,
                    TDataReq, TDataInd, TDisReq, TDisInd
endtype

```

Definicji poszczególnych kategorii poleceń dalej nie definiujemy, dla czytelności zakłada się natomiast, że nazwy rodzajów dla tych typów są takie same jak nazwy typów. Z poleceń dowolnych kategorii można, oczywiście, selekcjonować ich argumenty. W przedstawianych specyfikacjach użyte operacje selekcji są oddzielnie komentowane.

Nie podaje się też specyfikacji typów punktów dostępu do usług *TPDU*, końcowych punktów połączeń *KPP* oraz transportowych jednostek usługowych *TPU*.

Pierwszy z przedstawianych opisów pomija dodatkowo zarządzanie przesyłanymi danymi [Bilski, Dubielewicz 1991, 1993]

specification *ProsteUslugiTransportowe[t]*

type *TPDU* is **endtype**

type *KPP* is **endtype**

type *TJU* is **endtype**

type *T-polecenie* is *TConReq*, *TConInd*, *TConResp*, *TConConf*,
TDataReq, *TDataInd*, *TDisReq*, *TDisInd*

endtype

behaviour

T-uslugi[t]

where

process *T-uslugi[t]* : **noexit** :=

t?pdu1 : *TPDU* ?*kpp1* : *KPP* ? *prym* : *TConReq*;

(*T-uslugi[t]*

|||

let *pdu2* : *TPDU* = *przeznaczeniePDU(prym)* **in**

(* *przeznaczeniePDU* jest selektorem jednego z parametrów polecenia usługowego *prym*, które jest żądaniem nawiązania połączenia

*)

(*OtwórzPołączenie[t]*(*pdu1*, *kpp1*, *pdu2*)

>> **accept** *kpp2* : *KPP* **in**

TransferDwukierunkowy[t](*pdu1*, *kpp1*, *pdu2*, *kpp2*)

[> *ZamknijPołączenie[t]*(*pdu1*, *kpp1*, *pdu2*, *kpp2*))

)

where

process *OtwórzPołączenie[t]*

(*pdu1* : *TPDU*, *kpp1* : *KPP*, *pdu2* : *TPDU*) :

exit(*KPP*) :=


```

    t !pdu2 ?kpp2 : KPP !TConInd(pdu1) ;
    (* jest tu pokazany jawnie parametr pdu1 polecenia TConInd *)
    (
      t !pdu2 !kpp2 !TConResp ;
      t !pdu1 ?kpp1 !TConConf ;
      exit(kpp2)
    [] t !pdu2 !kpp2 !TDisInd ;
      stop
    )
  endproc
process TransferDwukierunkowy[t]
  (pdu1 : TPDU, kpp1 : KPP, pdu2 : TPDU) :
  exit(KPP) :=
    JednokierunkowyTransfer[t](pdu1, kpp1, pdu2, kpp2)
    |||
    JednokierunkowyTransfer[t](pdu2, kpp2, pdu1, kpp1)
  where
    process JednokierunkowyTransfer[t]
      (pdu1 : TPDU, kpp1 : KPP, pdu2 : TPDU, kpp2 : KPP) :
    noexit :=
      t !pdu1 !kpp1 ?prym : TDataReq ;
      JednokierunkowyTransfer[t](pdu1, kpp1, pdu2, kpp2)
    [] t !pdu2 !kpp2 ?prym : TDataInd ;
      JednokierunkowyTransfer[t](pdu1, kpp1, pdu2, kpp2)
    endproc
  endproc
process ZamknijPotaczenie[t]
  (pdu1 : TPDU, kpp1 : KPP, pdu2 : TPDU) :
  noexit(KPP) :=
    t !pdu1 !kpp1 !TDisReq ;
    Zamknij[t](pdu2, kpp2)
  [] t !pdu2 !kpp2 !TDisReq ;
    Zamknij[t](pdu1, kpp1)
  where
    process Zamknij[t](pdu : TPDU, kpp : KPP) : noexit(KPP) :=
      t !pdu !kpp !TDisReq ; stop
    [] t !pdu !kpp !TDisInd ; stop
    [] t !pdu !kpp ?prym : TDataReq ; Zamknij[t](pdu, kpp)

```

```

    [] t !pdu !kpp ?prym : TDataInd ; Zamknij[t](pdu, kpp)
  endproc
endproc
endproc
endspec

```

Specyfikacja *ProsteUsługiTransportowe[t]* jest definicją najbardziej zewnętrznego procesu. Definicja ta określa używane typy, a zasadniczą jej częścią jest definicja procesu *T-usługi[t]*. Usługi transportowe polegają na przesyłaniu danych przez połączenie transportowe, określone przez dwa identyfikatory końcowych punktów połączenia przypisanych do, być może tych samych, punktów dostępu do usług. Realizacja usług przebiega w trzech fazach: ustanawiania połączenia, transferu danych, zwalniania połączenia. W pierwszej fazie obowiązuje zasada potwierdzania realizacji poleceń. Oznacza to, że polecenie żądania nawiązania połączenia *TConReq*, skierowane do warstwy transportowej przez inicjatora ustanowienia połączenia, dociera od warstwy transportowej do adresata w postaci polecenia wskazania *TConInd*. Adresat odpowiada na tę propozycję poleceniem *TConResp*, kierowanym do warstwy transportowej, która przekazuje je do inicjatora w postaci polecenia *TConCnf*. W fazie nawiązywania połączenia, oprócz wskazania adresata, negocjowane są parametry transferu danych. Zaprezentowana wyżej specyfikacja nie przedstawia negocjowanych parametrów. W pozostałych fazach żądanie przesłania danych *TDataReq* oraz żądanie zwolnienia połączenia *TDisReq* przez jednego z partnerów połączenia są przekazywane przez warstwę transportową do drugiego partnera, odpowiednio w postaci poleceń *TDataInd* oraz *TDisInd*, których nie trzeba już potwierdzać.

Warto zwrócić uwagę na funkcjonalności procesów reprezentujących poszczególne fazy transferu. Procesy *OtwórzPołączenie* oraz *ZamknijPołączenie* mają funkcjonalność **exit**, natomiast proces *TransferDwukierunkowy* – funkcjonalność **noexit**. Bezpośrednio oznacza to, że proces *TransferDwukierunkowy* nie kończy się, ale nie oznacza to, że faza transferu danych ma trwać nieograniczenie długo, gdyż jest ona przerywana przez proces *ZamknijPołączenie* – złożenie deaktywujące

```

TransferDwukierunkowy[t](pdu1, kpp1, pdu2, kpp2)
[> ZamknijPołączenie[t](pdu1, kpp1, pdu2, kpp2) ]

```

Jak wspomniano, przedstawiona specyfikacja jest uproszczona, nie uwzględnia wszystkich ograniczeń nałożonych na usługi transportowe. Dodatkowe ograniczenia, które należy uwzględnić, można wyrazić przez rozbudowę tej specyfikacji. Poniżej przedstawiono przykład takiej rozbudowy, która wiąże się z koniecznością zapewniania przesyłania danych do odbiorcy w takiej kolejności, w jakiej wysłał je nadawca.

Rozbudowa polega na wprowadzeniu procesu, który nadzoruje wprowadzanie i wyprowadzanie do kolejki przesyłanych danych. Podstawowym typem, wykorzystywa-

nym przez ten proces, będzie *Kolejka*. Generyczny (sparametryzowany) typ kolejkowy ma następującą definicję

```

type Kolejka is Boolean
  formalsorts element
  sorts kolejka
  opns pustaKolejka : -> kolejka
        dołącz : element, kolejka -> kolejka
        pierwszy : kolejka -> element
        usuńPierwszy : kolejka -> kolejka
        jestPusta : kolejka -> Bool
  eqns forall x, y : element, q : kolejka
    ofsort element
      pierwszy(dołącz(x, pustaKolejka)) = x ;
      pierwszy(dołącz(x, dołącz(y, q))) = pierwszy(dołącz(y, q)) ;
    ofsort kolejka
      usuńPierwszy(dołącz(x, pustaKolejka)) = pustaKolejka ;
      usuńPierwszy(dołącz(x, dołącz(y, q))) =
        dołącz(x, usuńPierwszy(dołącz(y, q))) ;
    ofsort Bool
      jestPusta(pustaKolejka) = true ;
      jestPusta(dołącz(x, q)) = false ;
endtype

```

Przedstawiony typ wymaga aktualizacji, formalny rodzaj *element* powinien być zastąpiony rodzajem reprezentującym transportowe jednostki usługowe, co może mieć postać

```

type KolejkaTJU is Kolejka
  actualizedby TJU using
  sortnames tju for element
endtype

```

gdzie *tju* jest nazwą rodzaju wcześniej wprowadzonego, ale niezdefiniowanego typu *TJU*.

Nowa wersja specyfikacji różni się tylko definicją procesu *T-usługi*, w którym pojawia się nowy proces składowy *ZarządcaKolejki*, a pozostałe procesy składowe pozostają niezmienione

```

process T-usługi[t] : noexit :=
  t?pdul : TPDU ?kpp1 : KKP ? prym : TConReq;

```

```

(  T-usługi[t]
    |||
    let pdu2 : TPDU = przeznaczeniePDU(prym) in
      (OtwórzPołączenie[t](pdu1, kpp1, pdu2)
        >>  accept kkp2 : KPP in
          ZarządcaKolejki[t](pdu1, kpp1, pdu2, kkp2, pustaKolejka)
          ||
          ZarządcaKolejki[t](pdu1, kpp1, pdu2, kkp2, pustaKolejka)
          ||
          (  TransferDwukierunkowy[t](pdu1, kpp1, pdu2, kkp2)
            [> ZamknijPołączenie[t](pdu1, kpp1, pdu2, kkp2) ]
          )
        )
      )
)

where

process  OtwórzPołączenie[t]
  (pdu1 : TPDU, kpp1 : KPP, pdu2 : TPDU) :
exit(KPP) :=
  .....
endproc

process  TransferDwukierunkowy[t]
  (pdu1 : TPDU, kpp1 : KPP, pdu2 : TPDU) :
exit(KPP) :=
  .....
endproc

process  ZamknijPołączenie[t]
  (pdu1 : TPDU, kpp1 : KPP, pdu2 : TPDU) :
noexit(KPP) :=
  .....
endproc

process  ZarządcaKolejki[t](pdu1 : TPDU, kpp1 : KPP,
                           pdu2 : TPDU, q : kolejkaTJU) :
noexit :=
  t !pdu1 !kpp1 ?prym : TDataReq ;
  ZarządcaKolejki[t](pdu1, kpp1, pdu2, kpp2, dołącz(dane(prym), q))
[] [not(jestPusta(q)) -> t !pdu2 !kpp2 ?prym : TDataInd(pierwszy(q)) ;
  (* jest tu jawnie pokazany parametr pierwszy(q) polecenia TDataInd *)
  ZarządcaKolejki[t](pdu1, kpp1, pdu2, kpp2, usuńPierwszy(q))
endproc

endproc

```

Nowe ograniczenie reprezentowane przez proces *ZarządcaKolejki*, dokładniej przez dwie instancje tego procesu po obu stronach połączenia, jest składane równolegle z procesem *TransferDwukierunkowy*.

6.4. Uwagi końcowe

W tym rozdziale omówiono zwięźle model referencyjny ISO/OSI i zasady wykorzystania języka LOTOS do reprezentacji usług i protokołów. Prezentacja ma charakter nieformalny, odwołujący się do wprowadzie do prostych przykładów, ale dobranych tak, aby zilustrować możliwości języka i przedstawić pewne przyjęte konwencje jego zastosowania. Bardziej szczegółowe omówienie zasad i konwencji zawiera książka [van Eijk, Vissers, Diaz 1989]. Przykłady opracowano na podstawie pracy [Huzar 1990].

Specyfikacja standardów sieciowych, poza normami dotyczącymi modelu ISO/OSI oraz wybranych standardów internetowych, na przykład:

- IEEE Connectionless Interneting Protocol,
- ISO Network Service,
- ISO Transport Protocol,
- ISO Transport Service,
- ISO Session Protocol,
- ISO Session Service,
- ISO Presentation Protocol,
- ISO Transaction Protocol,
- ISO File Transfer Service and Protocol,
- ISO Job Transfer and Manipulation Service and Protocol,

była również przedmiotem wielu innych prac, na przykład: [van Eijk, Vissers, Diaz 1989], [Schneider 1996], [Comer 1997].

7. Metodyka specyfikowania

7.1. Proces specyfikowania

Systematyczne tworzenie systemu oprogramowania powinno się odbywać zgodnie z przyjętą metodyką. Wyznacza się więc pewne etapy, które – niezależnie od konkretnej metodyki – obejmują specyfikację, projektowanie i implementację. Każdy z etapów może się składać z wielu kroków. W dalszym ciągu skupimy się tylko na etapie specyfikacji w LOTOSie. Etap ten może się składać z sekwencji kroków, których rezultatem są kolejne specyfikacje:

$$Spec_1, Spec_2, \dots, Spec_n$$

Pierwsza specyfikacja $Spec_1$ powstaje zwykle na podstawie opisu tekstowego w języku naturalnym lub w języku formalnym, a ostatnia $Spec_n$ powinna prowadzić przynajmniej do prototypu implementacji.

Pomiędzy specyfikacjami powinny zachodzić odpowiednie relacje. Kolejne specyfikacje są rozwinięciami lub uszczegółowieniami specyfikacji wcześniejszych. Pomiędzy dwiema sąsiednimi specyfikacjami $Spec_{k-1}$ i $Spec_k$ ($k = 2, \dots, n$) można wyróżnić, jako podstawowe, następujące rodzaje zależności:

- $Spec_k$ może być *strukturalnym uściśleniem* $Spec_{k-1}$. Przedstawiony w postaci „czarnej skrzynki” opis systemu lub jego fragmentu może być zastąpiony opisem w postaci „szarej” lub „białej skrzynki”. Pomiędzy specyfikacjami powinna zachodzić relacja równoważności obserwowalnej (zob. p. 7.2) lub równoważności testowej (zob. p. 7.3).
- $Spec_k$ może być *restrukturyzacją* $Spec_{k-1}$. Przedstawiony w postaci „białej skrzynki” opis systemu lub jego fragmentu może być zastąpiony opisem w postaci innej „białej skrzynki”, której wewnętrzna struktura może lepiej odpowiadać potrzebom projektowym. Również w tym przypadku pomiędzy specyfikacjami powinna zachodzić relacja równoważności obserwowalnej lub testowej.
- $Spec_k$ może być *rozszerzeniem* $Spec_{k-1}$. Rozszerzenie jest oczywiste tam, gdzie proces specyfikowania rozpoczyna się od ustalenia podstawowych zachowań, a w kolejnych krokach specyfikację uzupełnia się dodatkowo zidentyfikowanymi zachowaniami. Jeżeli na przykład specyfikacja $Spec_{k-1}$ jest reprezentowana przez wyrażenie behawioralne B_{k-1} , to specyfikacja $Spec_k$ może być reprezentowana przez wyrażenie $B_{k-1} [] B$, gdzie B jest wyrażeniem reprezentującym dodatkowe zachowania.
- $Spec_k$ może być *redukcją* $Spec_{k-1}$. Redukcja, chociaż jej stosowanie może się wydawać niepotrzebnie restryktywne, często się przydaje, gdy specyfikacja ma być bezpośrednio przetransformowana w implementację systemu. W takich sytu-

acjach redukcja może służyć do eliminacji dopuszczalnych wcześniej opcji lub niedeterminizmu, który dopuszcza specyfikacja. Od implementacji najczęściej oczekuje się własności niedeterminizmu. Specyfikacja $Spec_{k-1}$ może na przykład dopuszczać zachowania reprezentowane wyrażeniem behawioralnym $a; B_1 [] a; B_2$, a specyfikacja $Spec_k$ postaci $a; B_2$ będzie wyrażać pewną decyzję projektową – postępowania zgodnie z wyrażeniem B_2 , gdy nastąpi realizacja akcji a .

7.2. Równoważność obserwacyjna

Zdefiniowane w rozdziale 2. relacje równoważności dla języka CCS mają swoje odpowiedniki w języku LOTOS. Pojęcia silnej i słabej bisymulacji dla LOTOSa są definiowane tak samo, jak poprzednio dla CCS (definicje 2.7 i 2.9). Podobnie definiuje się relację kongruencyjnej równoważności obserwacyjnej (definicja 2.12).

Przypomnijmy: wyrażenia B_1 oraz B_2 są *bisymulacyjnie silnie* równoważne, co oznacza się $B_1 \sim B_2$, gdy istnieje relacja bisymulacji R taka, że $\langle B_1, B_2 \rangle \in R$, oraz są *obserwacyjnie* równoważne, co oznacza się $B_1 \approx B_2$, gdy istnieje słaba bisymulacja S taka, że $\langle B_1, B_2 \rangle \in S$.

Kongruencyjna równoważność obserwacyjna $B_1 \approx^c B_2$ oznacza, że te dwa wyrażenia behawioralne można wzajemnie zastępować w dowolnym wyrażeniu (kontekście) bez zmiany obserwowanego zachowania całego wyrażenia.

Badanie kongruencyjnej równoważności obserwacyjnej można prowadzić analitycznie na podstawie odpowiednich praw. Poniżej przedstawiono zbiór aksjomatów do badania kongruencyjnej równoważności obserwacyjnej [ISO 8807]:

Prawa dla prefiksowania akcją

$$\begin{aligned} g...? x : s... [c]; B &\approx^c g...? y : s... [x := y][c]; [x := y]B \\ g...? x : s... [c]; B &\approx^c \text{choice } x : s [] g...! x; B \\ g !t_1 ... !t_n [c]; B &\approx^c [c] \rightarrow g !t_1 ... !t_n [c]; B \end{aligned}$$

Prawa dla wyboru

$$\begin{aligned} B_1 [] B_2 &\approx^c B_2 [] B_1 \\ B_1 [] (B_2 [] B_3) &\approx^c (B_1 [] B_2) [] B_3 \\ B [] B &\approx^c B \\ B [] \text{stop} &\approx^c B \\ [x := e]B [] \text{choice } x : s [] B &\approx^c \text{choice } x : s [] B \\ &\quad \text{jeśli } [e] \in \text{Term}_s(\emptyset) / =_s, \\ \text{choice } x : s [] B &\approx^c B \quad \text{jeśli } x \text{ nie jest zmienną wolną w } B \\ \text{choice } x : s [] \text{exit}(..., x, ...) &\approx^c \text{exit}(..., \text{any}, ...) \end{aligned}$$

Prawa dla złożenia równoległego

W zapisie tych praw będą stosowane następujące konwencje: symbol $|$ oznacza tu operator złożenia równoległego z dowolnym ciągiem bramek synchronizujących, l, l_1, l'_1 są ciągami bramek synchronizujących, przez $set(l)$ będzie oznaczany zbiór elementów listy l , a przez $FG(B)$ – zbiór bramek wolnych wyrażenia behawioralnego B (funkcja FG nie jest tu definiowana; jej definicja jest oczywista – zob. na przykład definicje funkcji $FAct$ w tabeli 2.2 oraz FB w tabeli 3.1).

$$B_1 | B_2 \approx^c B_2 | B_1$$

$$B_1 | (B_2 | B_3) \approx^c (B_1 | B_2) | B_3$$

$$\mathbf{exit}(t_1, \dots, t_n) | \mathbf{exit}(t'_1, \dots, t'_m) \approx^c \mathbf{exit}(t_1, \dots, t_n)$$

gdy $n = m$, $[t_i] = [t'_i]$ lub $t'_i \equiv \mathbf{any} \ s$ oraz $\text{sort}(t_i) = s$ dla $i = 1, \dots, n$

$$\mathbf{exit}(t_1, \dots, t_n) | \mathbf{exit}(t'_1, \dots, t'_m) \approx^c \mathbf{stop} \quad \text{w przeciwnym przypadku}$$

$$\mathbf{exit}(\dots) | \mathbf{stop} \approx^c \mathbf{stop}$$

$$B_1 || [l_1] B_2 \approx^c B_1 | [l'_1] B_2 \quad \text{jeśli } l'_1 \text{ zawiera te same elementy, co } l_1$$

$$B_1 || [l_1] B_2 \approx^c B_1 | [l'_1] B_2 \quad \text{jeśli } set(l'_1)$$

$$= set(l_1) \cap (set(FG(B_1)) \cup set(FG(B_2)))$$

$$B_1 || [l] B_2 \approx^c B_1 || B_2 \quad \text{jeśli } set(FG(B_1)) \cup set(FG(B_2)) \subseteq set(l)$$

$$B_1 || [] B_2 \approx^c B_1 ||| B_2$$

Prawa dla złożenia aktywowującego

Symbol $>>^*$ oznacza dowolne złożenie aktywujące, to znaczy przekazujące dowolne zestawy wartości pomiędzy składowymi wyrażeniami aktywowującego:

$$\mathbf{stop} >>^* B \approx^c \mathbf{stop}$$

$$\mathbf{exit} >> B \approx^c \mathbf{i}; B$$

$$\mathbf{exit}(t_1, \dots, t_n) >> \mathbf{accept} \ x_1: s_1, \dots, x_n: s_n \ \mathbf{in} \ B \approx^c \mathbf{i}; [x_1 := t_1, \dots, x_n := t_n] B$$

$$(B_1 >> B_2) >> B_3 \approx^c B_1 >> (B_2 >> B_3)$$

$$B >>^* \mathbf{stop} \approx^c B ||| \mathbf{stop}$$

Prawa dla złożenia deaktywowującego

$$\mathbf{stop} [> B \approx^c B$$

$$\mathbf{exit}(\dots) [> B \approx^c \mathbf{exit}(\dots) [] B$$

$$(B_1 [> B_2) [> B_3 \approx^c B_1 [> (B_2 [> B_3)$$

$$(B_1 [> B_2) [] B_2 \approx^c B_1 [> B_2$$

$$B [> \mathbf{stop} \approx^c B$$

Prawa dla przestonięcia

hide l in $B \approx^c$ hide l' in B	jeśli l' zawiera te same elementy, co l
hide l in $B \approx^c$ hide l' in B	jeśli $set(l') = set(l) \cap FG(B)$
hide l in $B \approx^c B$	jeśli $set(l) \cap FG(B) = \emptyset$
hide l in hide l' in $B \approx^c$ hide l'' in B	jeśli $set(l'') = set(l) \cup set(l')$
hide l in $g!t_1 \dots !t_n; B \approx^c i; \mathbf{hide} \ l \ \mathbf{in} \ B$	jeśli $g \in set(l)$
hide l in $g; B \approx^c g; \mathbf{hide} \ l \ \mathbf{in} \ B$	jeśli $g \notin set(l)$
hide l in $B_1 [] B_2 \approx^c$ hide l in $B_1 [] \mathbf{hide} \ l \ \mathbf{in} \ B_2$	
hide l in $B_1 [l'] B_2 \approx^c$ (hide l in B_1) $[l']$ (hide l in B_2)	jeśli $set(l) \cap set(l') = \emptyset$
hide l in $B_1 >>^* B_2 \approx^c$ (hide l in B_1) $>>^*$ (hide l in B_2)	
hide l in $B_1 [> B_2 \approx^c$ (hide l in B_1) $>$ (hide l in B_2)	
hide l in $[c] \rightarrow B \approx^c [c] \rightarrow$ (hide l in B)	

Prawa dla dozorów

$[c] \rightarrow B \approx^c B$	jeżeli $c = true$
$[c] \rightarrow B \approx^c \mathbf{stop}$	jeżeli $c = false$

Prawo dla instancji procesu

$P[h_1, \dots, h_m](t_1, \dots, t_n) \approx^c [x_1 := t_1, \dots, x_n := t_n] B[g_1 := h_1, \dots, g_m := h_m]$
 jeśli
process $p[g_1, \dots, g_m](x_1: s_1, \dots, x_n: s_n) : \text{funkcjonalność} := B \mathbf{endproc}$

Prawa dla przemianowania bramek

Niech $[S]$ oznacza dowolne przemianowanie bramek postaci $[g_i := h_1, \dots, g_n := h_n]$, gdzie $g_i \neq g_j$ dla $i \neq j$ ($i, j = 1, \dots, n$). Przemianowanie to wyznacza funkcję przemianowania bramek S , określoną następująco:

$$S(g_i) = h_i \quad \text{dla } i = 1, \dots, n$$

$$S(g) = g \quad \text{dla } g \neq g_i \text{ dla } i = 1, \dots, n$$

stop $[S] \approx^c \mathbf{stop}$

exit $(...)[S] \approx^c \mathbf{exit}(...)$

$(a; B)[S] \approx^c S(a); B[S]$

$(B_1 [] B_2)[S] \approx^c B_1[S] [] B_2[S]$

$(B_1 [|l|] B_2)[S] \approx^c B_1[S] [|l|] B_2[S]$ jeśli S jest iniekcją na $FG(B_1) \cup FG(B_2) \cup set(l)$

$$(B_1 \gg^* B_2)[S] \approx^c B_1[S] \gg^* B_2[S]$$

$$(B_1 [> B_2])[S] \approx^c B_1[S] [> B_2[S]]$$

$$(\text{hide } l' \text{ in } B)[S] \approx^c \text{hide } l \text{ in } B[S] \quad \text{jeśli } S \text{ jest iniekcją na } FG(B) \cup l' \text{ oraz } S(l') = l$$

$$B[S] \approx^c B \quad \text{jeśli } S \text{ jest identycznością na } FG(B)$$

$$B[S_1] \approx^c B[S_2] \quad \text{jeśli } S_1(g) = S_2(g) \text{ dla } g \in FG(B)$$

$$B[S_1][S_2] \approx^c B[S_1 \circ S_2] \quad \text{gdzie } \circ \text{ jest symbolem złożenia funkcji}$$

Prawa dla akcji wewnętrznych

$$a; i; B \approx^c a; B$$

$$B [] i; B \approx^c i; B$$

$$a; (B_1 [] i; B_2) [] a; B_2 \approx^c a; (B_1 [] i; B_2)$$

$$[x := t]B [] \text{choice } x:s [] i; B \approx^c \text{choice } x:s [] i; B \quad \text{jeśli } [t] \in \text{Term}_s(\emptyset) / =_s$$

7.3. Równoważność testowa i implementacyjna

Badanie czy pomiędzy dwiema specyfikacjami zachodzi relacja równoważności obserwacyjnej jest równie trudne, jak weryfikacja poprawności programów względem danych specyfikacji. W przypadku specyfikacji, dla których odpowiadający etykietowany system przejść jest skończony, istnieją algorytmy badania spełnialności relacji równoważności obserwacyjnej, nie ma natomiast takich algorytmów dla dowolnych etykietowanych systemów przejść. Nawet przy prostych specyfikacjach, z powodu eksplozji stanów, algorytmy okazują się nieefektywne obliczeniowo [Turner 1993]. Z tych powodów weryfikację zastępuje się testowaniem. Praktyczne podejście do badania równoważności obserwacyjnej specyfikacjami stosuje pojęcie równoważności testowej.

Relacja *równoważności testowej* dwóch specyfikacji oznacza, że specyfikacji tych nie można rozróżnić przez testowanie pewnym zbiorem testów $TEST$. Dla dwóch zbiorów testów takich, że $TEST_1 \subseteq TEST_2$, jeśli dwie specyfikacje są równoważne testowo względem $TEST_2$, to są również równoważne testowo względem $TEST_1$.

Jeśli dwie specyfikacje są równoważne obserwacyjnie, to są także, oczywiście, równoważne testowo. Odwrotna implikacja natomiast nie zachodzi. Jeżeli zbiór testów jest dostateczny (odpowiednio duży), to zachodzenie równoważności testowej pociąga zachodzenie równoważności obserwacyjnej specyfikacji. Problem określenia zbioru testów jest trudny. Podstawowa trudność wynika z tego, że dostateczny zbiór testów jest często nieskończony albo – w przypadku skończoności – zbyt duży, aby przeprowadzać pełne testowanie. Generowanie odpowiednich testów jest oddzielnym, trudnym problemem [Behforoz, Hudson 1996], [Górski 1999], [Maciaszek 2004]. Stosuje się wielorakie techniki automatycznej generacji testów. Dla języka LOTOS istnieją

środowiska programistyczne, w pewnym zakresie wspomagające generację testów, na przykład w środowisku LOLA [Quemada, Pavón, Fernandez 1989] lub CADP [Garavel, Lang, Mateescu 2001].

Relacja równoważności testowej abstrahuje od wewnętrznej struktury specyfikacji, skupiając się tylko na komunikacji na zewnętrznych brankach.

Test dla instancji specyfikacji o funkcjonalności **noexit** postaci

$$Spec[g_1, \dots, g_n](t_1, \dots, t_m)$$

można określić jako instancję dowolnego procesu postaci

$$Test[g_1, \dots, g_n, sukces](\dots)$$

Układem testującym specyfikacji *Spec* nazywamy wyrażenia behawioralne postaci:

$$Spec[g_1, \dots, g_n](t_1, \dots, t_m)$$

$$|[g_1, \dots, g_n]|$$

$$Test[g_1, \dots, g_n, sukces](\dots)$$

Układ testujący dla specyfikacji o funkcjonalności **exit** jest postaci:

$$Spec[g_1, \dots, g_n](t_1, \dots, t_m) >> sukces; \text{stop}$$

$$|[g_1, \dots, g_n]|$$

$$Test[g_1, \dots, g_n, sukces](\dots)$$

Każde obliczenie układu testującego, które zakończy się zajściem zdarzenia na wyróżnionej bramce *sukces*, oznacza zakończenie testu z powodzeniem. Jeżeli natomiast obliczenie zakończy się, ale na bramce *sukces* nie zajdzie zdarzenie, oznacza to niepowodzenie testu.

Dwie specyfikacje *deterministyczne* są *równoważne testowo* względem pewnego zbioru testów (wyrażonego instancjami procesów), jeżeli dla dowolnego procesu testującego obliczenie układu testującego dla obu specyfikacji kończy się zawsze tym samym rezultatem (powodzenie albo niepowodzenie).

Dwie specyfikacje *niedeterministyczne* są *równoważne testowo* względem pewnego zbioru testów, gdy dla dowolnego procesu testującego istnieje obliczenie kończące się sukcesem (niepowodzeniem) dla jednej ze specyfikacji, wówczas także istnieje obliczenie kończące się sukcesem (niepowodzeniem) dla drugiej specyfikacji.

Z podanych określeń bezpośrednio wynika, że badanie równoważności testowej specyfikacji niedeterministycznych jest bardziej złożone od badania specyfikacji deterministycznych.

Wyrażenia behawioralne LOTOSa mogą być przedmiotem badania, ale mogą także być wykorzystane do definiowania testów budowanego systemu na podstawie jego

tekstowej specyfikacji. W przypadku takich opisów często mamy do czynienia z dwoma rodzajami testów: z testami akceptacyjnymi i testami odrzucającymi. Testy *akceptacyjne* określają dopuszczalne zachowania systemu podczas współdziałania z otoczeniem. Testy *odrzucające* określają natomiast zbiory zdarzeń, których system w danym stanie nie powinien akceptować.

Pojedynczy test akceptacyjny wyznacza pojedynczy ciąg zdarzeń zdefiniowany jako proces o strukturze

```
process TestAkceptujący[ $g_1, \dots, g_n, sukces$ ] : noexit :=  
     $e_1 ; \dots ; e_m ; sukces ; \text{stop}$   
endproc
```

gdzie e_i , dla $i = 1, \dots, m$, jest akcją komunikacyjną na jednej z bramek $g_1, \dots, g_n$.

Pojedynczy test odrzucający, stosowany do danego stanu testowanego systemu, jest zdefiniowany jako proces o strukturze

```
process TestOdrzucający[ $g_1, \dots, g_n, sukces$ ] : noexit :=  
     $e_1 ; \text{stop}$   
    []  $e_2 ; \text{stop}$   
    .....  
    []  $e_m ; \text{stop}$   
    [] i ;  $sukces ; \text{stop}$   
endproc
```

gdzie e_i mają znaczenie jak poprzednio, ale zajęcie któregośkolwiek z nich oznacza niepowodzenie testu, gdyż testowany system akceptuje to, czego nie powinien. Dodatkowo należy przyjąć, że w tym teście akcje komunikacyjne mają priorytet wyższy od priorytetu akcji wewnętrznej, co oznacza, że akcja wewnętrzna jest wykonywana tylko wtedy, gdy nie można wykonać żadnej akcji komunikacyjnej.

Pomiędzy specyfikacją a implementacją systemu powinien zawsze zachodzić odpowiedni związek. W przypadku specyfikacji wyrażonej w LOTOSie związek ten jest określony jako relacja implementacji.

Relacja *implementacji* jest określana [Brinksma, Scollo 1986], [Turner 1993] jako połączenie dwóch relacji: relacji redukcji i rozszerzenia. Połączenie to należy rozumieć jako zastosowanie jednej z tych relacji lub ich złożenie w dowolnej kolejności. Kolejność składania relacji odzwierciedla kolejność podejmowania decyzji projektowych podczas tworzenia implementacji.

Relacja redukcji oznacza eliminację ze specyfikacji niedeterminizmu powodowanego istnieniem akcji wewnętrznych. Polega to na przepisaniu specyfikacji z ustaleniem opcjonalnych wyborów.

Relacja rozszerzenia oznacza taką rozbudowę specyfikacji, która dopuszcza nowe, zachowując wszystkie stare zachowania. Inaczej: dopuszcza dołączenie do specyfikacji nowych funkcjonalności.

Obie relacje są wyrażane przez odpowiedzi na tzw. testy akceptująco-odrzucające, które stanowią sekwencyjne połączenie dwóch poprzednio zdefiniowanych testów: testów akceptujących i testów odrzucających. Testy te mają sprawdzać akceptację danych sekwencji obserwowalnych zdarzeń, a po ich zajściu odrzucanie pewnych zbiorów zdarzeń. Mają one następującą strukturę

```

process TestAkceptującoOdrzucający[ $g_1, \dots, g_n, sukcesAkcept, sukcesOdrz$ ] :
noexit :=
     $akcept_1 ; \dots ; akcept_m ; sukcesAkcept ;$ 
    (
         $odr_1 ; stop$ 
        []  $odr_2 ; stop$ 
        .....
        []  $odr_k ; stop$ 
        []  $i ; sukcesOdrz ; stop$ 
    )
endproc

```

gdzie: $akcept_i$, dla $i = 1, \dots, m$, oraz odr_j , dla $j = 1, \dots, k$, są akcjami komunikacyjnymi na jednej z bramek $g_1, \dots, g_n$.

Specyfikacja $Spec_1$ jest *testowo zgodna* ze specyfikacją $Spec_2$, jeśli dowolny test akceptująco-odrzucający, zbudowany w taki sposób, że jego część akceptująca jest testem dla specyfikacji $Spec_2$, a część odrzucająca jest oparta na dowolnym zbiorze akcji obu specyfikacji, jest testem akceptująco-odrzucającym dla specyfikacji $Spec_1$, to jest także testem akceptująco-odrzucającym dla specyfikacji $Spec_2$.

Specyfikacja $Spec_2$ jest *rozszerzeniem* specyfikacji $Spec_1$, jeżeli obie specyfikacje są testowo zgodne oraz $SeqObs(Spec_1) \subseteq SeqObs(Spec_2)$.

Specyfikacja $Spec_2$ jest *redukcją* specyfikacji $Spec_1$, jeżeli obie specyfikacje są testowo zgodne oraz $SeqObs(Spec_2) \subseteq SeqObs(Spec_1)$.

Na zakończenie warto wspomnieć, że – oprócz języka LOTOS – do definiowania testów można używać innych języków. Standardem ISO do specyfikacji testów dla systemów komunikacyjnych jest zwłaszcza TTCN (*Tree and Tabular Combined Notation*) [ISO/IEC 9646-3, 1998].

7.4. Style specyfikowania

LOTOS jest przeznaczony przede wszystkim do tworzenia specyfikacji zachowań systemów. Wyróżnia się dwie kategorie specyfikacji: specyfikacje ekstensjonalne

i intensjonalne. Specyfikacje *ekstensjonalne* są formułowane w oderwaniu od przyszłej implementacji, w tym sensie, że specyfikacja nie narzuca ograniczeń na strukturę przyszłej implementacji. Struktura specyfikacji *intensjonalnej* jest natomiast sugestią postaci przyszłej implementacji. Podany podział jest nieco umowny, gdyż wszystkie specyfikacje w LOTOSie, jako wykonywalne, można traktować jako prototyp implementacji. W ramach każdej z tych kategorii można wyróżnić po dwa style specyfikowania – dla specyfikacji ekstensjonalnych:

- monolityczny,
- zorientowany na ograniczenia,

a dla specyfikacji intensjonalnych:

- zorientowany na maszyny stanowe,
- zorientowany na zasoby (inaczej: zorientowany na implementację).

Styl *monolityczny* wiąże się z twierdzeniem o ekspansji, z którego wynika, że dowolne wyrażenie behawioralne, na drodze pojedynczych transformacji, daje się sprowadzić do postaci, która zawiera tylko operatory wyboru i prefiksowania akcją. Twierdzenie o ekspansji dla języka CCS przedstawiono w rozdziale 2. Siła ekspresji języka LOTOS jest taka sama jak języka CCS, dlatego twierdzenie o ekspansji odnosi się także do języka LOTOS. Jeżeli stosowanie transformacji opisanych twierdzeniem o ekspansji prowadzi do skończonej długości wyrażenia behawioralnego, to jest ono zapisane w stylu monolitycznym. Styl monolityczny dopuszcza też użycie operatora rekursji procesów do definiowania pętli. Na ogół konstruowanie specyfikacji wiąże się z wyobrażeniem drzewa zachowań budowanego systemu. Można uznać ten styl za bardziej abstrakcyjny od pozostałych.

W stylu *zorientowanym na definiowanie ograniczeń* specyfikacja jest złożeniem równoległym procesów, przy czym każdy proces wyznacza pewien zbiór ograniczeń, które muszą być łącznie zachowane przez cały system. Oznacza to, że zachowanie procesu składowego specyfikacji jest ograniczone zachowaniem innych procesów, z którymi dany proces się synchronizuje. Gdy procesy składowe nie synchronizują się ze sobą, obserwowalne zachowania systemu są pewnego rodzaju „sumą” – przeplotem zachowań procesów składowych, gdy natomiast procesy składowe synchronizują się ze sobą, obserwowane zachowanie systemu jest pewnym „przekrojem” zachowań procesów składowych. Podstawową konstrukcją jest złożenie równoległe procesów, często używane do synchronizacji multilateralnej. Drugą konstrukcją jest predykat ograniczający akcje komunikacyjne.

Styl *zorientowany na definiowanie ograniczeń* jest stosowany w przypadku podejścia zstępującego do tworzenia specyfikacji, jak na przykład w definiowaniu standardów sieci komputerowych (zob. rozdz. 6.).

Styl *zorientowany na maszyny stanowe* polega na przedstawieniu specyfikacji jako pewnej sekwencyjnej maszyny stanowej (automatu skończonego z pamięcią). Przy

takim podejściu wyklucza się użycie konstrukcji złożenia równoległego. Można wyróżnić dwie odmiany stylu, różniące się sposobem identyfikacji stanów maszyny stanowej. Pierwsza polega na wykorzystaniu zmiennych do identyfikacji stanów, a druga – na wykorzystaniu procesów. Schematycznie pierwsza odmiana prowadzi do specyfikacji, których treść przyjmuje schemat

```

process Spec[ ...](stan : int) : ... :=
    [stan = 0] ->  $a_1$ ; Spec[...](1)
    [] [stan = 1] ->  $a_2$ ; Spec[...](2)
    .....
    [] [stan =  $N$ ] ->  $a_n$ ; Spec[...]( $N$ )
endproc

```

gdzie a_i , dla $i = 1, \dots, N$, są akcjami wykonywanymi w przejściach pomiędzy stanami. Poszczególne stany są reprezentowane wywołaniami procesów *Spec*[...](j) dla $j = 0, \dots, N$.

Schemat specyfikacji w drugiej odmianie ma postać

```

process Spec[ ...] : ... :=
    Stan0[...](...)
where
    process Stan0[ ...] : ... :=
         $a_1$ ; Stan1[...]
        []  $a_2$ ; Stan2[...]
        .....
    endproc
    .....
    process StanN[ ...] : ... :=
         $b_1$ ; Stan1[...]
        []  $b_2$ ; Stan2[...]
        .....
    endproc
endproc

```

Tak jak w poprzedniej odmianie, $a_1, a_2, \dots, b_1, b_2, \dots$ są akcjami wykonywanymi w przejściach pomiędzy stanami, poszczególne stany są reprezentowane instancjami procesów *Stanj*[...].

Styl zorientowany na definiowanie maszyn stanowych jest podobny do stylu monolitycznego w tym, że wykorzystuje w zasadzie ten sam zestaw operatorów języka. Może być zalecany wtedy, gdy zależy nam na bezpośredniej transformacji specyfikacji w program.

Styl *zorientowany na zasoby* jest najbardziej intuicyjnym, powiązanym z implementacją stylem specyfikowania. W tym podejściu specyfikowany system jest widziany jako zbiór komunikujących się procesów. Procesy reprezentują elementy składowe (moduły albo zasoby) systemu, które mają współpracować ze sobą za pośrednictwem wspólnych bramek. Synchronizacja procesów odzwierciedla zachodzącą pomiędzy nimi komunikację i dlatego mamy do czynienia tylko z synchronizacją par procesów. Operator przesłonięcia służy do przesłaniania komunikacji wewnętrznych składowych procesów. Styl ten jest stosowany w przypadku podejścia wstępującego do tworzenia specyfikacji.

Często w obrębie jednej specyfikacji mamy do czynienia z kilkoma stylami. Sytuacja taka występuje w dalej przedstawionym przykładzie, w którym możemy odnaleźć zastosowanie stylu zorientowanego na definiowanie maszyn stanowych i zorientowanego na zasoby.

7.5. Przykładowy problem

Rozpatrzmy prosty system przydziału zwrotnych zasobów, na który składają się: magazyn zasobów, magazynier i dyspozytor. Z systemu korzystają klienci, którzy zwracają się dyspozytora z zapotrzebowaniem na określoną ilość zasobów. Dyspozytor, na podstawie analizy stanu magazynu zasobów, podejmuje decyzję o ilości zasobów dla danego zapotrzebowania. W szczególności, odmawiając przydziału, ilość przydzielonych zasobów określa jako liczbę zero. Po uzyskaniu decyzji o przydziale, klient zwraca się do magazyniera po odbiór zasobów, przedstawiając mu decyzję. Magazynier pobiera z magazynu zasoby i przekazuje je klientowi. Klient po wykorzystaniu zasobów, za pośrednictwem magazyniera, zwraca je do magazynu.

Tak bardzo ogólnie sformułowany system można ukonkretniać, przypisując specyficzną interpretacją jego elementom. Może to być na przykład element systemu bankowego, w którym magazynem jest skarbiec, magazynierem – kasjer, dyspozytorem – pracownik banku upoważniony do udzielania pożyczek, a klientami – osoby fizyczne lub prawne.

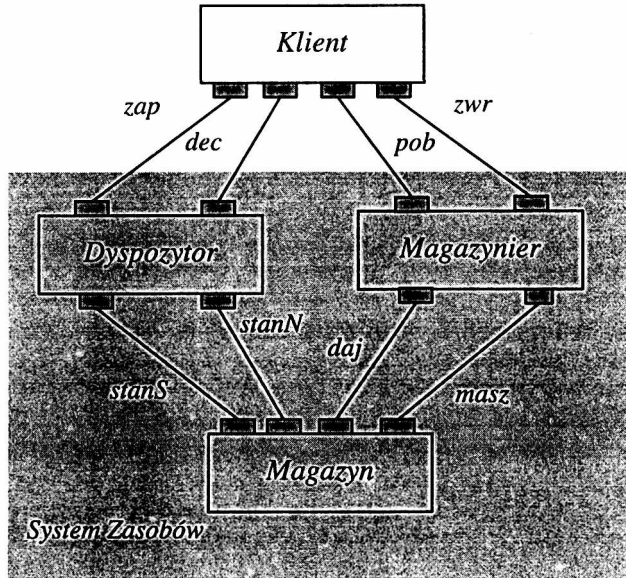
Strukturę systemu zasobów przedstawiono na rysunku 7.1. Elementy składowe komunikują się ze sobą przez bramki, z których *wnk*, *dec*, *poż*, *zwr* służą do udostępniania usług systemu, a pozostałe są wewnętrznymi bramkami systemu.

W przypadku wstępującego tworzenia specyfikacji może być zastosowany styl zorientowany na zasoby. Załóżmy, że wcześniej zostały utworzone specyfikacje procesów modelujących składowe elementy systemu

```

process Magazyn[stanS, stanN, daj, masz](n : Nat) : noexit := ... endproc
process Magazynier[pob, zwr, daj, masz] : noexit := ... endproc
process Dyspozytor[zap, dec, stanS, stanN] : noexit := ... endproc
process Klient[zap, dec, pob, zwr] : noexit := ... endproc

```

Rys. 7.1. Struktura prostego systemu zasobów

Pomijamy dalej treści niektórych procesów, ograniczając się tylko do przedstawienia ich sygnatur. Stosując styl zorientowany na zasoby, definicje te można potraktować jako elementy składowe (zasoby), które służą do utworzenia specyfikacji całego systemu. Specyfikacja taka, zgodna z rysunkiem 7.1, ma postać

```

Klient[zap, dec, pob, zwr]
  |[zap, dec, pob, zwr]|
hide stanS, stanN, daj, masz in
  ( Dyspozytor[zap, dec, stanS, stanN]
    |[ ]|
    Magazynier[pob, zwr, daj, masz]
  )
  |[stanS, stanN, daj, masz]|
  Magazyn[stanS, stanS, daj, masz]

```

Specyfikacje procesów składowych można uzyskać, stosując styl zorientowany na definiowanie maszyn stanowych. Rozważmy na przykład definicje dwóch procesów *Magazyn* oraz *Dyspozytor*

```

process Magazyn[stanS, stanN, daj, masz](n : int) : noexit :=
  stanS !n; Magazyn[stanS, stanN, daj, masz](n)
  [] stanN ?m : int; Magazyn[stan, daj, masz](m)

```

```

[] daj ?m : int; Magazyn[stan, daj, masz](n)
[] masz ?m : int; Magazyn[stan, daj, masz](n + m)
endproc
process Dyspozytor[zap, dec, stanS, stanN] : noexit :=
  zap ?id : Identyfikator ?ilość : int;
  stanS ?zapas : int;
  ( [ ilość < zapas ] ->
    ( dec!id!ilość;
      stanN !(zapas – ilość);
      Dyspozytor[zap, dec, stanS, stanN]
    )
    [] i; dec!id!0;
      Dyspozytor[zap, dec, stanS, stanN]
  )
  [] [zapas < ilość] -> dec!id!0; Dyspozytor[zap, dec, stanS, stanN]
)
endproc

```

W przedstawionej specyfikacji, dla czytelności, wykorzystano typ całkowitoliczbowy (*int*) w tradycyjnej programistycznej notacji.

Łatwo zauważyć, że struktura wyrażenia będącego treścią procesu *Magazyn* stanowi odmianę stylu, polegającą na wykorzystaniu zmiennych do identyfikacji stanów maszyny stanowej – wartościowanie występującej w tym wyrażeniu zmiennej *n* reprezentuje stan procesu. Podobne uwagi można odnieść do drugiego z procesów.

Przedstawiony przykład jest ilustracją nie tylko różnych stylów specyfikowania, ale służy również do omówienia jeszcze innych problemów.

Problem pierwszy wiąże się z przyjętym interfejsem komunikacyjnym pomiędzy procesami, a zwłaszcza komunikacji procesu *Klient* z systemem zasobów. Przyjęta w przykładzie koncepcja interfejsu różni się do koncepcji przedstawionej w rozdziale 6. Poprzednio komunikacja pomiędzy warstwami sieci komputerowej odbywała się za pośrednictwem jednej bramki, obecnie komunikacja odbywa się za pośrednictwem kilku bramek. Łatwo zauważyć, że pomiędzy takimi formami interfejsu daje się ustanowić wzajemnie jednoznaczny związek. Jeżeli mianowicie przyjąć, że komunikaty wymieniane pomiędzy dwoma procesami za pośrednictwem jednej bramki mają postać

polecenie(*argument*)

gdzie zbiór różnych poleceń jest skończony, a argumenty mają określone typy, to komunikacja może się odbywać za pośrednictwem zbioru bramek, z których każda odpowiada dokładnie jednemu poleceniu. Wysłanie więc lub odebranie takiego komunikatu będzie wyrażone odpowiednio przez akcje komunikacyjne:

polecenie ! *argument* lub *polecenie* ? *argument* : *typ*.

Drugi problem wynika z niedeterminizmu i nieokreśloności w specyfikacji. W przykładzie własności te odnoszą się do definicji procesu *Dyspozytor*.

Z treści procesu *Dyspozytor* wynika, że dyspozytor może, ale nie musi, przydzielić zasoby w żądanej ilości tylko wtedy, gdy liczba zasobów w magazynie jest większa od liczby żądanej. W wyniku realizacji akcji wewnętrznej *i* dyspozytor odmawia przydziału zasobu. Pozostawienie niedeterminizmu w treści procesu przesłania sposób podejmowania przez dyspozytora odpowiedniej decyzji. Takie użycie niedeterminizmu jest typowym sposobem przesłaniania procesów decyzyjnych w procesie tworzenia specyfikacji. Inny problem jest związany z przypadkiem, gdy *zapas = ilość*. Jest to problem nieokreśloności – dla tego przypadku definicja nie określa zachowania procesu.

Eliminacja obu przypadków – niedeterminizmu i nieokreśloności – może być podstawą konstrukcji kolejnej specyfikacji. Eliminacja niedeterminizmu prowadzi do nowej specyfikacji, na przykład

```

process Dyspozytor1[zap, dec, stanS, stanN] : noexit :=
  zap ?id : Identyfikator ?ilość : int;
  stanS ?zapas : int;
  ( [ilość < zapas] ->
    dec!id!ilość;
    stanN !(zapas - ilość);
    Dyspozytor1[zap, dec, stanS, stanN]
  [] [zapas < ilość] -> dec!id!0; Dyspozytor1[zap, dec, stanS, stanN]
  )
endproc

```

Łatwo zauważyć, że proces *Dyspozytor* jest testowo zgodny z procesem *Dyspozytor1* oraz $\text{SeqObs}(\text{Dyspozytor1}) \subseteq \text{SeqObs}(\text{Dyspozytor})$, proces *Dyspozytor1* jest zatem redukcją procesu *Dyspozytor*.

Eliminacja nieokreśloności, która nadal pozostaje w procesie *Dyspozytor1*, prowadzi na przykład do specyfikacji

```

process Dyspozytor2[zap, dec, stanS, stanN] : noexit :=
  zap ?id : Identyfikator ?ilość : int;
  stanS ?zapas : int;
  ( [ilość < zapas] ->
    dec!id!ilość;
    stanN !(zapas - ilość);
    Dyspozytor2[zap, dec, stanS, stanN]
  [] [zapas ≤ ilość] -> dec!id!0; Dyspozytor2[zap, dec, stanS, stanN]
  )
endproc

```

Podobnie jak wcześniej, proces *Dyspozytor1* jest testowo zgodny z procesem *Dyspozytor2* oraz $SeqObs(Dyspozytor1) \subseteq SeqObs(Dyspozytor2)$, proces *Dyspozytor2* jest zatem rozszerzeniem procesu *Dyspozytor1*.

Trzeci problem to rozwinięcie specyfikacji, polegające na dopuszczeniu jednoczesnej obsługi wielu klientów. Jednoczesna obsługa wymaga takiej modyfikacji procesów *Dyspozytor* i *Magazyn*, aby były one zdolne do równoległej obsługi klientów. Poniżej pokazano tylko modyfikację ostatniej wersji procesu *Dyspozytor2*

```

process Dyspozytor3[zap, dec, stanS, stanN] : noexit :=
  choice id : Identyfikator in
    zap !id ?ilość : int;
    stanS ?zapas : int;
    ( [ilość < zapas] ->
      dec!id!ilość;
      stanN !(zapas – ilość);
      Dyspozytor3[zap, dec, stanS, stanN]
    [] [zapas ≤ ilość] -> dec!id!0; Dyspozytor3[zap, dec, stanS, stanN]
  )
  [| ]|
  Dyspozytor3[zap, dec, stanS, stanN]
endproc

```

Przedstawiona wersja jest bardzo „rozrzutna”, gdyż wywołanie procesu pociąga za sobą nieograniczoną liczbę równoległych, niezależnych, to znaczy niekomunikujących się ze sobą, kopii procesu *Dyspozytor3*. Każda z kopii będzie się komunikować tylko z jednym procesem *Klient* o ustalonym identyfikatorze. Proces *Dyspozytor3* jest rozszerzeniem procesu *Dyspozytor2*.

7.6. Środowiska wspomagające specyfikowanie w LOTOSie

Rozwojowi LOTOSu towarzyszył rozwój środowisk programowych wspomagających używanie języka do specyfikowania i projektowania systemów oprogramowania. Przykładem jednego z najwcześniejszych jest pakiet LOLA (od *LOtos Laboratory*), opracowany na uniwersytecie w Madrycie [Quemada, Pavón, Fernandez 1989], udostępniany w Internecie w wersji na komputery typu IBM PC.

Obecnie najbardziej rozwinięty jest pakiet CADP – akronim oznaczający początkowo *CÆSAR/ALDEBARAN Development Package*, obecnie stanowiący rozwinięcie od *Construction and Analysis of Distributed Processes*. Jest on wynikiem prac, prowadzonych od początku lat dziewięćdziesiątych ubiegłego wieku, w INRIA (Institut National de Recherche en Informatique et en Automatique), we współpra-

cy z innymi ośrodkami europejskimi i kanadyjskimi [Garavel, Lang, Mateescu 2001].

Pakiet CADP, umożliwiając pisanie specyfikacji w LOTOSie, pozwala na analizę własności tworzonych specyfikacji oraz na automatyczną generację i testowanie programów. Pakiet składa się z modułów działających w otwartym środowisku OPEN/CÆSAR. Środowisko przyjmuje format *BCG (Binary Coded Graphs)* reprezentacji etykietowanych systemów przejść oraz oferuje kolekcję bibliotek i programów związanych z tym formatem.

Głównymi modułami funkcjonującymi w środowisku OPEN/CÆSAR są:

- Kompilatory *CAESAR* i *CAESAR.ADT* – służące do translacji specyfikacji w LOTOSie w kod w języku programowania C. Pierwszy z nich dokonuje translacji części behawioralnej specyfikacji w LOTOSie na kod w języku C, a drugi kompiluje abstrakcyjne typy zdefiniowane w specyfikacji na typy i funkcje języka C.
- Symulator *OCIS* – umożliwiający wizualizację i śledzenie obliczeń specyfikacji napisanych w LOTOSie. Dopuszcza kilka form wizualizacji (m.in. ciągi i drzewa zdarzeń, diagramy sekwencji), różne sposoby prowadzenia obliczeń (m.in. praca krokowa, według zadanych scenariuszy), modyfikację i rekompilację specyfikacji).
- Analizator *ALDEBARAN* – służący do weryfikacji systemów komunikacyjnych reprezentowanych w postaci etykietowanych systemów przejść. Umożliwia on redukcję etykietowanych systemów przejść względem wybranych relacji równoważności (różnych relacji bisymulacji) i stwierdzanie równoważności zredukowanych etykietowanych systemów przejść, opartych na algorytmach Paige–Tarjana [Paige, Tarjan 1987] oraz Fernandez–Mouniera [Fernandez, Mounier 1995].
- *EVALUATOR* – weryfikator formuł temporalnych, który pozwala na sprawdzenie, czy specyfikacja w LOTOSie ma własności dające się wyrazić w językach logiki temporalnej, między innymi: HML [Hennessy, Milner 1985], CTL [Clarke, Emerson, Sistla 1983], ACTL [de Nicola, Vaandrager 1990] i LTAC [Queille, Sifakis 1983].
- Generator testów *TVG* – służy do wyprowadzania zestawu testów ze specyfikacji formalnej na podstawie celu testu. Cel testu określa się przez wskazanie stanów akceptujących albo stanów odrzucających.
- Graficzny interfejs *EUCALYTUS* – umożliwiający jednolity dostęp do zintegrowanych modułów.

Pakiet CADP ciągle się rozwija. Są dwa zasadnicze kierunki rozwoju:

Pierwszy wynika z zamiaru integracji różnych formalnych języków specyfikacji, dlatego przyjęto etykietowane systemy przejść jako jednolitą formę abstrakcyjnej repre-

zentacji specyfikacji, gdyż możliwe jest sprowadzenie do niej nie tylko specyfikacji w LOTOSie, ale także w innych językach formalnych, na przykład: SDL i UML/RT.

Drugi kierunek wiąże się z przystosowaniem narzędzia do potrzeb przemysłowych. Zwraca się uwagę na efektywność narzędzi tak, aby mogły dokonywać analiz specyfikacji o odpowiednio dużych rozmiarach, a także aby uwzględniać te języki specyfikacji, które znajdują uznanie i zastosowanie w przemyśle, stąd między innymi, wynika uwzględnienie języków SDL i UML/RT.

7.7. Uwagi końcowe

W tym rozdziale dokonano przeglądu stylów tworzenia specyfikacji w języku LOTOS. Ponieważ tworzenie specyfikacji może być procesem wieloetapowym, zarysowano problemy związane z porównywaniem kolejno po sobie następujących specyfikacji. Zagadnienia te są dokładniej omawiane w wielu publikacjach, między innymi w: [Visser, Scollo, van Sinderen 1988], [Brinksma 1989], [Logrippo, Probert, Ural 1990], [Garavel, Sifakis 1990], [Visser, Scollo, van Sinderen, Brinksma 1991], [Leduc 1992]. Przeglądowym podsumowaniem tych prac jest książka [Turner 1993].

Style specyfikowania zilustrowano oryginalnym, prostym przykładem, który pokazuje, że w specyfikacji stosunkowo prostego zagadnienia stosuje się pewną mieszaninę stylów elementarnych. Wydaje się, że nie jest to szczególna własność języka LOTOS, ale można ją odnieść również do innych technik, nie tylko formalnych. Brak jednolitego stylu, albo – ogólniej – uniwersalnej metodyki stosowania języka LOTOS nie tylko w specyfikacji, ale w projektowaniu i implementacji systemów, jest jedną z przyczyn krytycznych ocen języka wyrażanych przez przedstawicieli przemysłu [Logrippo 2000], [Babich, Deonto 2002].

Badanie równoważności testowej, omówione w rozdziale bardzo ogólnie, było przedmiotem wielu prac, na przykład: [De Nicola, Hennessy 1984], [Bolognesi, Smolka 1987].

Samo testowanie jest natomiast przedmiotem między innymi norm ISO. W powiązaniu z rozwojem formalnych technik specyfikacji opracowano język TTCN (*Tree and Tabular Combined Notation*) specyfikacji testów. Norma [ISO/IEC 9646-3, 1998] jest jedną z wcześniejszych szeregu norm dotyczących TTCN-1, standardu, który praktycznie wyszedł z użycia. Obecnie prace nad rozwojem tej grupy standardów są prowadzone pod auspicjami ETSI (European Telecommunications Standards Institute). Ich wynikiem jest opracowanie grupy standardów TTCN-3, a także technik i narzędzi testowania – szczegółowe informacje na ten temat można znaleźć między innymi na stronach internetowych:

<http://www.etsi.org>

<http://www.ttcn-3.org>

8. Problem blokad w LOTOSie

8.1. Ukryte blokady

Standardowa semantyka języka LOTOS, przedstawiona w rozdziale 6., ma pewien defekt, który można określić mianem ukrywania blokad [Huzar, Kuźniarz 1993]. Każda semantyka jest formalną reprezentacją pewnych postulatów w LOTOSie, podobnie jak w CCS, dotyczących sposobu komunikacji procesów.

Specyfikacja w LOTOSie wyznacza kolekcję procesów. Dany proces wykonuje akcje komunikacyjne, synchronizując się z innymi procesami, które stanowią jego otoczenie, bądź wykonując akcje wewnętrzne. Akcje wewnętrzne mogą być określane jawnie lub mogą być rezultatem przesłonięcia komunikacji pomiędzy procesami składowymi danego procesu. Intencja związana z akcją wewnętrzną, stanowiącą wynik przesłonięcia komunikacji, jest następująca:

jeżeli na przesłoniętej bramce nastąpiła komunikacja pomiędzy procesami (co najmniej dwoma!), to akcja taka, z punktu widzenia obserwatora niedostrzegającego przesłoniętej bramki, jest dostrzegana jako akcja wewnętrzna.

Standardowa semantyka tymczasem postulatu tego nie spełnia. Rozpatrzmy wyrażenie behawioralne postaci

$$g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k; B$$

gdzie: $\alpha_1 \ \alpha_2 \ \dots \ \alpha_k$ są elementami komunikacyjnymi związanymi z akcją na bramce g .

Jedyną możliwą tranzycją dla tego wyrażenia jest, oczywiście, tranzycja wynikająca z komunikacji na bramce g

$$g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k; B \xrightarrow{g \ v_1 \dots v_k} B' \quad (8.1)$$

Tranzycję tę interpretujemy następująco: wyrażenie $g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k; B$ w rezultacie wykonania akcji komunikacyjnej na bramce g (to znaczy na skutek synchronizacji ze swoim otoczeniem i wymianie odpowiednich wartości) będzie się dalej zachowywać zgodnie z wyrażeniem B' , które jest modyfikacją wyrażenia B , wynikającą z podstawienia za zmienne wolne występujące w akcji $g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k$ komunikowanych wartości – aksjomat ($A\text{-}pre_{comm}$).

Rozpatrzmy teraz wyrażenie behawioralne postaci

$$\text{hide } g \text{ in } g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k; B \quad (8.2)$$

Z aksjomatu ($A\text{-}pre_{comm}$) oraz reguły ($R_2\text{-}hide$) wynika, że zachodzi tranzycja

$$\text{hide } g \text{ in } g; \alpha_1 \ \alpha_2 \ \dots \ \alpha_k; B \xrightarrow{i} \text{hide } g \text{ in } B' \quad (8.3)$$

gdzie B' ma być odpowiednią modyfikacją wyrażenia B . Nasuwa się pytanie: jaka to ma być modyfikacja? Ponieważ nie zachodzi tu komunikacja z innym procesem, wejściowym elementom komunikacyjnym można przypisać, w niedeterministyczny sposób, dowolne wartości odpowiadających im typów; semantyka obserwacyjna **hide** g **in** $g; \alpha_1 \alpha_2 \dots \alpha_k; B$ jest więc równoważna semantyce obserwacyjnej wyrażenia **choice** $x : s \square B$.

Nasuwa się pytanie: jak interpretować tranzycję (8.3)? Nie można jej traktować tak jak tranzycji (8.1), gdyż na bramce g nie zachodzi komunikacja z innym procesem, ale jednocześnie postać wyrażenia B' jest taka jak w przypadku poprzednim. Nasuwa się natomiast sugestia, by wyrażenie **hide** g **in** $g; \alpha_1 \alpha_2 \dots \alpha_k; B$ interpretować jako wyrażenie z wewnętrzną blokadą, czyli takie wyrażenie, które nie oferuje swemu otoczeniu żadnych komunikacji i jednocześnie nie może wykonać żadnej akcji wewnętrznej.

Zauważmy jednocześnie, że wyrażenie behawioralne postaci

$$\mathbf{hide} \ g \ \mathbf{in} \ g? \ x : s; B_1 \parallel [g] \ g!t; B_2$$

nie stwarza wątpliwości interpretacyjnych. W tranzycji

$$\mathbf{hide} \ g \ \mathbf{in} \ g? \ x : s; B_1 \parallel [g] \ g!t; B_2 \xrightarrow{i} \mathbf{hide} \ g \ \mathbf{in} \ [t/x]B_1 \parallel [g] \ B_2$$

akcja wewnętrzna wyraża przesłonięcie komunikacji pomiędzy wyrażeniami składowymi złożenia równoległego.

Spełnienie postulatu, aby właściwie interpretować wyrażenie przesłonięcia bramki prowadzi do nowej semantyki języka. Semantyka ta jest przedstawiona w następnym podrozdziale.

8.2. Semantyki zmodyfikowane

Przedstawiono dwa rozwiązania problemu w postaci dwóch równoważnych semantyk.

Rozwiązanie 1

Zasadnicza idea, na której opiera się pierwsza semantyka, jest następująca: Aksjomat (*A-pre_{comm}*)

$$g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k; B \xrightarrow{g \ v_1 \dots v_k} B'$$

przedstawia tylko potencjalną tranzycję. Tranzycja ta powinna zajść tylko wtedy, gdy znajdzie się komunikacyjny partner na bramce g . Dlatego tranzycje postaci

$$B_1 \xrightarrow{g \ v_1 \dots v_k} B_2 \quad (8.4)$$

zastąpimy tranzycjami postaci

$$B_1 \xrightarrow{\langle H, g \ v_1 \dots v_k \rangle} B_2 \quad (8.5)$$

gdzie $n \geq 1$ jest dodatkowym parametrem etykiety, oznaczającym liczbę procesów uczestniczących w komunikacji na bramce g . Jeżeli $n = 1$, to tranzycja jest tylko potencjalna – jej zajście jest uwarunkowane kontekstem, jeżeli $n > 1$, to tranzycja może zajść.

Nowy etykietowany system przejść jest postaci

$$TS_1(Spec) = \langle Beh, Act_1, Tr_1, B_0 \rangle \quad (8.6)$$

gdzie:

Beh jest, jak poprzednio, zbiorem wyrażeń behawioralnych,

$Act_1 = \{ \langle n, g \ v \rangle \mid n \geq 1, g \in G \cup \{ \delta \}, v \in Term_{TSpec}(\emptyset)^* \} \cup \{ \langle 1, i \rangle \}$ jest zbiorem interakcji,

$Tr_1 = \{ \xrightarrow{a} \mid a \in Act_1 \}$ jest zbiorem relacji przejść,

B_0 jest treścią procesu początkowego p_0 .

Zbiór aksjomatów i reguł jest modyfikacją $TS_1(Spec)$, która – oprócz wprowadzenia dodatkowego parametru etykiety tranzycji – zmienia tylko aksjomaty oraz dwie reguły systemu $TS(Spec)$.

Prefiksowanie akcją wewnętrzną $B \equiv i; B_1$

$$B \xrightarrow{\langle n, i \rangle} B_1 \quad (A_1-pre_{int})$$

Prefiksowanie akcją komunikacyjną $B \equiv g \ \alpha_1 \ \alpha_2 \ \dots \ \alpha_k [c]; B_1$

$$B \xrightarrow{\langle 1, g v_1 \dots v_n \rangle} [x_{j_1} ::= t_{j_1}, \dots, x_{j_m} ::= t_{j_m}] B_1 \quad (A_1-precomm)$$

wtedy i tylko wtedy, gdy:

$v_i = [t_i]$, gdy $\alpha_i \equiv !t_i$ oraz t_i jest termem stałym, dla $i = 1, \dots, n$,

$v_i \in Term_{s_i}(\emptyset) / =_{s_i}$, gdy $\alpha_i \equiv ?x_i : s_i$, dla $i = 1, \dots, n$,

x_{j_i} jest zmienną taką, że $\alpha_{j_i} \equiv ?x_{j_i} : s_{j_i}$, dla $i = 1, \dots, n$,

oraz

$$D \vdash c[x_{j_1} ::= t_{j_1}, \dots, x_{j_m} ::= t_{j_m}]$$

Proces zakończenia $B \equiv \mathbf{exit}(t_1, \dots, t_n)$

$$B \xrightarrow{\langle 1, i \rangle} \mathbf{stop} \quad (A_1-exit)$$

Prześlonięcie $B \equiv \mathbf{hide} \ g_1, \dots, g_n \ \mathbf{in} \ B_1$

$$\frac{B_1 \xrightarrow{\langle n, a \rangle} B'_1}{B \xrightarrow{\langle n, a \rangle} B'_1} \quad \text{gdy } name(a) \notin \{g_1, \dots, g_n\} \quad (R_1-hide_1)$$

$$\frac{B_1 \xrightarrow{\langle n, a \rangle} B'_1}{B \xrightarrow{\langle 1, i \rangle} B'_1} \quad \text{gdy } name(a) \in \{g_1, \dots, g_n\}, n > 1 \quad (R_1-hide_2)$$

Istotna jest tu reguła ($R_1\text{-hide}_2$), gdyż wprowadza ograniczenie na możliwe tranzycje – wyklucza ona tranzycję dla wyrażenia postaci **hide** g **in** $g \alpha_1 \alpha_2 \dots \alpha_k; B$.

Złożenie równoległe $B \equiv B_1 \parallel [g_1, \dots, g_n] B_2$

$$\frac{B_1 \xrightarrow{\langle n, a \rangle} B'_1}{B \xrightarrow{\langle n, a \rangle} B'_1 \parallel [g_1, \dots, g_n] B_2} \quad \text{gdy } \text{name}(a) \notin \{g_1, \dots, g_n, \delta\} \quad (R_1\text{-par}_1)$$

$$\frac{B_2 \xrightarrow{\langle n, a \rangle} B'_2}{B \xrightarrow{\langle n, a \rangle} B_1 \parallel [g_1, \dots, g_n] B'_2} \quad \text{gdy } \text{name}(a) \notin \{g_1, \dots, g_n, \delta\} \quad (R_1\text{-par}_2)$$

$$\frac{\begin{array}{c} B_1 \xrightarrow{\langle n, a \rangle} B'_1 \\ B_2 \xrightarrow{\langle m, a \rangle} B'_2 \end{array}}{B \xrightarrow{\langle n+m, a \rangle} B'_1 \parallel [g_1, \dots, g_n] B'_2} \quad \text{gdy } \text{name}(a) \in \{g_1, \dots, g_n, \delta\} \quad (R_1\text{-par}_3)$$

Reguła ($R_1\text{-par}_3$) różni się od reguły ($R\text{-par}_3$) tylko tym, że wprowadza liczenie procesów uczestniczących w akcji na tej samej bramce.

Semantyki akcyjna i obserwacyjna są definiowane tak, jak w przypadku semantyki standardowej.

Rozwiązanie 2

Drugie rozwiązanie zakłada, że postać tranzycji nie ulega zmianie, lecz wprowadza się funkcje pomocnicze, które służą do definiowania ograniczeń na zachodzenie tranzycji. Są to funkcje:

$$\begin{aligned} F : \text{Beh} &\rightarrow 2^G \\ C : \text{Beh} &\rightarrow 2^G \end{aligned} \quad (8.7)$$

Ich definicje są zawarte w tabeli 8.1. W definicji funkcji C wykorzystuje się funkcję pomocniczą h , określoną następująco:

$$h(Z) = \begin{cases} \emptyset & \text{gdy } Z = \emptyset \\ \{i\} & \text{gdy } Z \neq \emptyset \end{cases} \quad (8.8)$$

Funkcja $F(B)$ dla wyrażenia behawioralnego B wyznacza zbiór tych jego akcji, które ze względu na składnię mogą być wykonane w pierwszej kolejności. To, która z tych akcji zajdzie jako pierwsza, zależy od otoczenia danego wyrażenia. Funkcja $C(B)$ wyznacza natomiast na podstawie zbioru $F(B)$ podzbiór tych akcji wewnętrznych, które mogą być wykonane niezależnie od otoczenia wyrażenia B .

Tabela 8.1

B	$F(B)$	$C(B)$
stop	$\emptyset$	$\emptyset$
i ; B_1	$\{i\}$	$\{i\}$
$g \alpha_1 \alpha_n$; B_1	$\{g\}$	$\emptyset$
exit ($t_1, \dots, t_n$)	$\{i\}$	$\{i\}$
$B_1 \gg \text{accept } x_1 : s_1, \dots, x_n : s_n \text{ in } B_2$	$F(B_1)$	$C(B_1)$
$B_1 [] B_2$	$F(B_1) \cup F(B_2)$	$C(B_1) \cup C(B_2)$
hide $g_1, \dots, g_n \text{ in } B_1$	$F(B_1) \setminus \{g_1, \dots, g_n\} \cup h(F(B_1) \cap \{g_1, \dots, g_n\})$	$C(B_1) \setminus \{g_1, \dots, g_n\} \cup C(B_2) \setminus \{g_1, \dots, g_n\} \cup F(B_1) \cap F(B_2) \cap \{g_1, \dots, g_n\}$
$B_1 [> B_2$	$F(B_1) \cup F(B_2)$	$C(B_1) \cup C(B_2)$
$B_1 [[g_1, \dots, g_n] B_2$	$F(B_1) \cup F(B_2)$	$C(B_1) \setminus \{g_1, \dots, g_n\} \cup h(C(B_1) \cap \{g_1, \dots, g_n\})$
$[c] \rightarrow B_1$	$F(B_1)$	$C(B_1)$
let $x_1 = t_1, \dots, x_n = t_n \text{ in } B_1$	$F(B_1)$	$C(B_1)$
choice $g \text{ in } [g_1, \dots, g_n] B_1$	$F(B_1[g_1/g]) \cup \dots \cup F(B_1[g_n/g])$	$C(B_1[g_1/g]) \cup \dots \cup C(B_1[g_n/g])$
choice $x : s [] B_1$	$F(B_1)$	$C(B_1)$
par $g \text{ in } [g_1, \dots, g_n] [[h_1, \dots, h_m] B_1$	$F(B_1[g_1/g]) \cup \dots \cup F(B_1[g_n/g])$	$C(B_1[g_1/g]) \cup \dots \cup C(B_1[g_n/g])$
$p[g_1, \dots, g_n](t_1, \dots, t_m)$	$F(B_1[g_1/h_1, \dots, g_n/h_n])$	$C(B_1[g_1/h_1, \dots, g_n/h_n])$
(B_1)	$F(B_1)$	$C(B_1)$

Etykietowany system przejść jest postaci

$$TS_2(Spec) = \langle Beh, Act, Tr_2, B_0 \rangle \quad (8.9)$$

gdzie inaczej, w stosunku do semantyki standardowej, jest zdefiniowana tylko relacja tranzycji. Jest ona określona przez taki sam zbiór aksjomatów jak semantyka standardowa i tylko ma inną regułę dla przesłonięcia.

Przesłonięcie $B \equiv \text{hide } g_1, \dots, g_n \text{ in } B_1$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{a} B'_1} \quad \text{gd } name(a) \notin \{g_1, \dots, g_n\} \quad (R_2\text{-hide}_1)$$

$$\frac{B_1 \xrightarrow{a} B'_1}{B \xrightarrow{i} B'_1} \quad \text{gd } name(a) \in \{g_1, \dots, g_n\} \cap C(B_1) \quad (R_2\text{-hide}_2)$$

Reguła ($R_2\text{-hide}_2$), przez wykorzystanie funkcji C , wprowadza ograniczenie na możliwe tranzycje.

Semantyki akcyjna i obserwacyjna są definiowane tak, jak w przypadku semantyki standardowej.

Oznaczając przez $Seq_1(B)$ oraz $Seq_2(B)$ zmodyfikowane obliczenia, można pokazać, że są one równoważne, czyli $Seq_1(B) = Seq_2(B)$.

Twierdzenie i dowód równoważności są przedstawione w artykule [Huzar, Kuźniarz 1993].

Rozwiązanie drugie jest bardziej eleganckie i bliższe prezentacji standardowej semantyki, dlatego rozważania dalszej części rozdziału są oparte na nim. Zmodyfikowany zbiór ciągów akcji generowany przez wyrażenie behawioralnego B będziemy oznaczać dalej przez $Seq_{mod}(B)$.

8.3. Wykrywanie blokad

Blokada jest niepożądaną własnością specyfikacji. Z punktu widzenia zewnętrznego obserwatora zachowania reprezentowanego przez dane wyrażenie behawioralne blokada jest nieodróżnialna od zakończenia aktywności wyrażenia.

W rozważaniach prowadzonych w dalszej części rozdziału ograniczymy się do bazowej wersji języka LOTOS, która abstrahuje od wartości komunikowanych pomiędzy procesami. Ograniczenie to wynika przede wszystkim stąd, że daje ono możliwość jednoznacznego rozstrzygnięcia o zachodzeniu blokad na podstawie analizy tekstu specyfikacji. W przypadku natomiast pełnej wersji języka pojawianie się blokad może zależeć od konkretnego obliczenia związanego ze specyfikacją, a przebieg obliczenia zależy od komunikowanych wartości.

Składnia uproszczonej wersji języka jest następująca:

$$B ::= \text{stop} \mid i; B_1 \mid g; B_1 \mid \text{exit} \mid B_1 >> B_2 \mid B_1 [\] B_2 \mid \text{hide } g_1, \dots, g_n \text{ in } B_1 \mid B_1 [> B_2 \mid B_1 \mid [g_1, \dots, g_n] \mid B_2 \mid p[g_1, \dots, g_n] \mid (B_1) \quad (8.10)$$

W wersji tej rezygnujemy również z konstrukcji:

choice g **in** $[g_1, \dots, g_n]$ B_1
par g **in** $[g_1, \dots, g_n]$ $[[h_1, \dots, h_m]] B_1$

gdź można je wyrazić za pomocą konstrukcji pozostałych.

Najpierw rozpatrzmy dwa przykłady, które wyjaśnią rozumienie behawioralnego wyrażenia z blokadą:

$g; \text{stop} \mid [g, h] \mid h; \text{stop}$
hide g **in** $g; \text{stop}$

Pierwsze z wyrażeń jest wyrażeniem z blokadą zarówno w sensie semantyki standardowej, jak i semantyki zmodyfikowanej, drugie natomiast jest wyrażeniem z blokadą tylko w sensie semantyki zmodyfikowanej.

Mamy również inne wyrażenia, które – nie będąc wyrażeniem **stop** – są semantycznie temu równoważne, będąc pewnym złożeniem procesów **stop**, na przykład:

stop [] stop

stop [> stop

Wprowadzimy rodzinę wyrażeń *STOP*, zdefiniowaną rekursywnie następująco:

$$\begin{aligned} stop ::= & \quad \mathbf{stop} \mid stop [] stop \mid [S] \mid stop \mid \mathbf{hide} \ g \ \mathbf{in} \ stop \mid \\ & \quad stop >> stop \mid stop [> stop \mid (stop) \end{aligned}$$

gdzie *stop* jest metazmienną.

Rodzina *STOP* wyznacza ten zbiór wyrażeń behawioralnych, wewnątrz których nie ma akcji komunikacyjnych i które są równoważne procesowi pustemu. Rodzinę tę będziemy odróżniać od zbioru wyrażeń z blokadą, wewnątrz których są wprowadzone akcje komunikacyjne, ale nie mogą być zrealizowane.

Definicja 8.1

Wyrażenie behawioralne *B* jest *wyrażeniem z blokadą* wtedy i tylko wtedy, gdy spełnia predykat *D(B)*, zdefiniowany następująco:

$$D(B) =_{\text{def}} (Seq_{mod}(B) = \emptyset \wedge B \notin STOP) \quad (8.11)$$

Tabela 8.2

<i>B</i>	<i>P(B)</i>
1	2
stop	$\emptyset$
i ; <i>B</i> ₁	<i>P(B</i> ₁ <i>)</i>
<i>g</i> ; <i>B</i> ₁	<i>P(B</i> ₁ <i>)</i>
exit	$\emptyset$
<i>B</i> ₁ >> <i>B</i> ₂	<i>P(B</i> ₁ <i>)</i> ∪ <i>P(B</i> ₂ <i>)</i>
<i>B</i> ₁ [] <i>B</i> ₂	<i>P(B</i> ₁ <i>)</i> ∪ <i>P(B</i> ₂ <i>)</i>
hide <i>g</i> ₁ , ..., <i>g</i> _{<i>n</i>} in <i>B</i> ₁	<i>P(B</i> ₁ <i>)</i>
<i>B</i> ₁ [> <i>B</i> ₂	<i>P(B</i> ₁ <i>)</i> ∪ <i>P(B</i> ₂ <i>)</i>
<i>B</i> ₁ [<i>g</i> ₁ , ..., <i>g</i> _{<i>n</i>}] <i>B</i> ₂	<i>P(B</i> ₁ <i>)</i> ∪ <i>P(B</i> ₂ <i>)</i>
<i>p</i> [<i>g</i> ₁ , ..., <i>g</i> _{<i>n</i>}](<i>t</i> ₁ , ..., <i>t</i> _{<i>m</i>})	{ <i>p</i> }
(<i>B</i> ₁)	<i>P(B</i> ₁ <i>)</i>

Rozpatrując dane wyrażenie, jesteśmy zainteresowani nie tylko tym, czy wyrażenie jest wyrażeniem z blokadą, ale także tym, czy wyrażenia osiągalne z danego mogą być rów-

niez wyrażeniami z blokadami. Rozważania będą ograniczone do wyrażen regularnie zbudowanych. Wyrażenia te zdefiniujemy tu ściśle, wprowadzając następujące pojęcia:

Mówimy, że proces p wywołuje bezpośrednio proces q , jeżeli wywołanie procesu q jest zawarte w wyrażeniu B_p , stanowiącym treść procesu p . Przez $CALL \subseteq Proc \times Proc$, gdzie $Proc$ jest zbiorem nazw procesów danej specyfikacji, oznaczmy relację *bezpośredniego wywołania* procesów

$$CALL = \{ \langle p, q \rangle \mid q \in P(B_p) \}$$

gdzie zbiór $P(B_p)$ jest zdefiniowany tabelą 8.2.

Niech $CALL^+$ będzie tranzytywnym domknięciem relacji $CALL$.

Definicja 8.2

Wyrażenie behawioralne B jest *regularnie zbudowane*, gdy dla każdego procesu $p \in Proc$ jest spełniony warunek: dla każdego podwyrażenia wyrażenia B_p , stanowiącego treść procesu p , postaci

$$B_1 \text{ op } B_2, \text{ dla } op \in \{ |[S]|, >>, [> \},$$

jeśli $q \in P(B_1) \cup P(B_2)$, to $\langle q, p \rangle \notin CALL^+$.

Lemat 8.1

Jeżeli wyrażenie B jest regularnie zbudowane, to zbiór wyrażen osiągalnych z B jest skończony.

Dowód (szkic)

Szczegółowy dowód, przeprowadzony metodą indukcji strukturalnej, jest prosty, ale uciążliwy – przy rozpatrywaniu kolejnych postaci wyrażenia B indukcja przebiega względem długości ciągu tranzycji. Główna idea jest natomiast prosta: jeżeli wyrażenie B jest regularnie zbudowane, to rekursywne wywołania procesów są skończone. Istotnie, liczba wyrażen, do których może nastąpić tranzycja z wyrażenia regularnie zbudowanego B , jest skończona. Dla dowolnego procesu $p \in Proc$ każde z podwyrażen wyrażenia B , jeśli zawiera wywołanie jakiegoś procesu, to – z definicji wyrażenia regularnego – nie prowadzi ani bezpośrednio, ani pośrednio do wywołania procesu p . Zbiór wyrażen osiągalnych z wyrażenia B jest zatem również skończony.

Wprowadzamy dwie kategorie wyrażen z blokadą: wyrażenia silnie i słabo blokujące.

W celu sprecyzowania tych pojęć w zbiorze $SeqFin(B)$ wyróżnimy zbiór *ciągów terminalnych* – podzbiór zbioru ciągów skończonych wyznaczanego przez semantykę akcyjną

$$SeqTerm(B) =_{\text{def}} \{ s \in SeqFin(B) \mid \neg \exists s' \bullet s' \neq \varepsilon \wedge s \wedge s' \in SeqFin(B) \} \quad (8.12)$$

Uwaga: Jeżeli wyrażenie B jest regularnie zbudowane, nie oznacza to skończoności zbioru ciągów terminalnych $SeqTerm(B)$.

Wyrażenie behawioralne B jest *silnie blokujące* wtedy i tylko wtedy, gdy spełnia predykat

$$SD(B) =_{\text{def}} SeqInf(B) = \emptyset \wedge \forall s \in SeqTerm(B) \bullet B \xrightarrow{s} B' \wedge D(B') \quad (8.13)$$

Oznacza to, że semantyka wyrażenia nie ma nieskończonych ciągów tranzycji, a każdy skończony terminalny ciąg tranzycji z wyrażenia B prowadzi do wyrażenia z blokadą.

Wyrażenie behawioralne B jest *ślabo blokujące* wtedy i tylko wtedy, gdy spełnia predykat

$$WD(B) =_{\text{def}} \exists s \in SeqTerm(B) \bullet B \xrightarrow{s} B' \wedge D(B') \quad (8.14)$$

Oznacza to, że dla wyrażenia istnieje skończony ciąg tranzycji terminalnych, który z wyrażenia B prowadzi do wyrażenia z blokadą.

Przykład 8.1

W celu zilustrowania wyrażen blokujących rozpatrzmy następujące wyrażenia behawioralne:

$B_1 = a; \text{stop}$

$B_2 = a; \text{stop} \parallel [a, b] \parallel b; \text{stop}$

$B_3 = \text{hide } a \text{ in } a; \text{stop}$

$B_4 = a; P[a]$ gdzie **process** $P[a] := a; P$ **end**

$B_5 = a; P[a] [] b; \text{stop}$

$B_6 = B_4 [] B_2$

$B_7 = B_4 [> B_2]$

$B_8 = B_4 [> a; B_2]$

Semantyka klasyczna oraz zmodyfikowana tych wyrażen przedstawiają się następująco:

$$Seq(B_1) = Seq_{mod}(B_1) = \{a\}$$

$$Seq(B_2) = Seq_{mod}(B_2) = \emptyset$$

$$Seq(B_3) = \{i\} \quad \text{natomiast} \quad Seq_{mod}(B_3) = \emptyset$$

$$Seq(B_4) = Seq_{mod}(B_4) = \{a, aa, \dots, aa \dots a, \dots\} \cup \{aa \dots a \dots\}$$

$$Seq(B_5) = Seq_{mod}(B_5) = \{b, a, aa, \dots, aa \dots a, \dots\} \cup \{aa \dots a \dots\}$$

$$Seq(B_6) = Seq_{mod}(B_6) = \{a, aa, \dots, aa \dots a, \dots\} \cup \{aa \dots a \dots\}$$

$$Seq(B_7) = Seq_{mod}(B_7) = \{a, aa, \dots, aa \dots a, \dots\} \cup \{aa \dots a \dots\}$$

$$Seq(B_8) = Seq_{mod}(B_8) = \{a, aa, \dots, aa \dots a, \dots\} \cup \{aa \dots a \dots\}$$

Zbiory ciągów terminalnych wyznaczonych dla zmodyfikowanej semantyki:

$$SeqTerm_{mod}(B_1) = \{a\}$$

$$SeqTerm_{mod}(B_2) = \emptyset$$

$$SeqTerm_{mod}(B_3) = \emptyset$$

$$SeqTerm_{mod}(B_4) = \emptyset$$

$$SeqTerm_{mod}(B_5) = \{b\}$$

$$SeqTerm_{mod}(B_6) = \emptyset$$

$$SeqTerm_{mod}(B_7) = \emptyset$$

$$SeqTerm_{mod}(B_8) = \{a, aa, \dots, aa \dots a, \dots\}$$

Zauważmy, że chociaż

$$Seq_{mod}(B_7) = Seq_{mod}(B_8),$$

to

$$SeqTerm_{mod}(B_7) \neq SeqTerm_{mod}(B_8).$$

Na podstawie definicji łatwo obliczyć warunki silnego i słabego blokowania (tab. 8.3).

Tabela 8.3

B	$SD(B)$	$WD(B)$
B_1	<i>fałsz</i>	<i>fałsz</i>
B_2	<i>prawda</i>	<i>prawda</i>
B_3	<i>prawda</i>	<i>prawda</i>
B_4	<i>fałsz</i>	<i>fałsz</i>
B_5	<i>fałsz</i>	<i>fałsz</i>
B_6	<i>fałsz</i>	<i>fałsz</i>
B_7	<i>fałsz</i>	<i>fałsz</i>
B_8	<i>fałsz</i>	<i>prawda</i>

Z definicji wynika, że badanie czy dane wyrażenie jest silnie lub słabo blokujące wymaga zbadania, czy zbiór ciągów $SeqInf(B)$ jest niepusty oraz czy zbiór ciągów $SeqTerm(B)$ prowadzi do wyrażeń z blokadą.

Teraz przedstawiamy dwa algorytmy związane z tymi dwoma badaniami. W prezentacji algorytmów będziemy wykorzystywać dla danego wyrażenia behawioralnego B zbiór wyrażeń, który bezpośrednio po nim następuje w wyniku wykonania pojedynczej tranzycji

$$Succ(B) = \{B' \mid \exists a \in Act \bullet B \xrightarrow{a} B'\} \quad (8.15)$$

Dla danego wyrażenia B zbiór ten można wyznaczyć się na podstawie zbioru aksjomatów i reguł określających relację tranzycji w systemie przejść.

Algorytm badania czy zbiór ciągów $SeqInf(B)$ jest niepusty przedstawimy w postaci rekursywnego obliczania wartości funkcji logicznej $NotEmptySeqInf(B)$. Funkcja oblicza wartość *prawda*, gdy zbiór $SeqInf(B)$ jest niepusty, oraz *fałsz*, w przypadku przeciwnym. Jej definicja wyraża się przez funkcję pomocniczą $NESI(B, pred)$, gdzie $pred$ jest zmienną pomocniczą, której wartościami są podzbiory wyrażeń behawioralnych. Definicja jest przedstawiona w samowyjaśniającej się konwencji programistycznej

$$NotEmptySeqInf(B) := NESI(B, \emptyset)$$

gdzie:

```

NESI(B, pred) :=
  begin
    if  $F(B) = \emptyset$  then return(fałsz) endif ;           -- 1
    for  $B' \in Succ(B)$  do
      if  $B' \in pred$  then return(prawda) endif ;       -- 2
    endfor ;

     $pred := pred \cup \{B\}$ ;
    for  $B' \in Succ(B)$  do
      if  $NESI(B', pred)$  then return(prawda) endif ;   -- 3
    endfor ;
    return(fałsz);                                     -- 4
  end ;

```

Objaśnijmy algorytm. Parametr formalny $pred$ reprezentuje zbiór wyrażeń behawioralnych, które w pewnym obliczeniu, prowadzącym do wyrażenia B , mogą wyrażenie to poprzedzać. Oznacza to, że istnieje obliczenie postaci

$$B_1 \xrightarrow{\alpha_1} B_2 \xrightarrow{\alpha_2} B_3 \dots B_{n-1} \xrightarrow{\alpha_n} B \quad (8.16)$$

gdzie $B_1, B_2, \dots, B_{n-1} \in pred$. Początkową wartością parametru $pred$ jest zbiór pusty.

Algorytm, badając zbiór następników $Succ(B)$, kończy się na mocy lematu, gdyż zbiór wyrażeń osiągalnych z danego wyrażenia B jest skończony. Zakończenie algorytmu następuje w czterech przypadkach:

W pierwszym przypadku (linia z etykietą 1) algorytm kończy się obliczeniem wartości *fałsz*, gdy wyrażenie B spełnia warunek $F(B) = \emptyset$. Oznacza to, oczywiście, że $Seq_{mod}(B) = \emptyset$, więc i $SeqInf_{mod}(B) = \emptyset$.

W drugim przypadku (linia z etykietą 2) algorytm kończy się obliczeniem wartości prawda, gdy napotka się wyrażenie B' , osiągalne z B , które może prowadzić do pewnego wyrażenia ze zbioru $pred$. Oznacza to, że istnieje pętla – obliczenie nieskończone.

Jeśli nie stwierdzi się istnienia pętli, przechodzi się do obliczania wartości funkcji $NESI$ dla wyrażeń ze zbioru $Succ(B)$, w poszerzonym zbiorze poprzedników $pred \cup \{B\}$. Jeśli $B' \in Succ(B)$, oznacza to istnienie obliczenia

$$B_1 \xrightarrow{\alpha_1} B_2 \xrightarrow{\alpha_2} B_3 \dots B_{n-1} \xrightarrow{\alpha_n} B \xrightarrow{\alpha'} B' \quad (8.17)$$

W tym przypadku (linia z etykietą 3) zakończenie algorytmu następuje, gdy stwierdzi się, że wartość funkcji $NESI$ dla pewnego wyrażenia $B' \in Succ(B)$ jest prawdą. Jeśli natomiast dla wszystkich wyrażeń ze zbioru $Succ(B)$ wartość funkcji $NESI$ jest fałszywa, to oznacza, że wartość tej funkcji jest również fałszywa dla wyrażenia B (linia z etykietą 4).

Drugi algorytm bada, czy wyrażenie jest silnie blokujące. Jest on przedstawiony w postaci rekursywnego obliczania wartości funkcji logicznej $StrongDeadock(B)$. Funkcja oblicza wartość *prawda*, gdy zbiór $SeqFin(B)$ jest pusty i każde wyrażenie osiągalne z B , które nie ma następników, nie należy do zbioru wyrażeń $STOP$, czyli spełnia predykat SD . W przypadku przeciwnym funkcja oblicza wartość *fałsz*. Jej definicja wyraża się za pomocą poprzednio zdefiniowanej funkcji $NotEmptySeqInf(B)$ oraz nowej funkcji $SD(B, pred)$, gdzie $pred$ jest zmienną pomocniczą, której wartościami są podzbiory wyrażeń behawioralnych.

$StrongDeadock(B) :=$

begin

if $NotEmptySeqInf(B)$ **then return**(*fałsz*); **endif** ;

$SD(B, \emptyset)$;

end ;

gdzie:

$SD(B, pred) :=$

begin

if $Succ(B) \setminus pred = \emptyset$

then

if $B \in STOP$ **then return**(*fałsz*) **endif**

else

for $B' \in Succ(B)$ **do**

if not $SD(B', pred \cup Succ(B))$ **then return**(*fałsz*) **endif**

endfor;

return(*prawda*);

end ;

Obliczenie czy wyrażenie jest słabo blokujące jest przedstawione w postaci rekursywnego obliczania wartości funkcji logicznej *WeakDeadock(B)*. Funkcja oblicza wartość *prawda*, gdy istnieje wyrażenie osiągalne z *B*, które nie ma następników, nie należy do zbioru wyrażeń *STOP*, czyli spełnia predykat *SD*, a w przypadku przeciwnym funkcja oblicza wartość *fałsz*. Jej definicja wyraża się za pomocą funkcji *WD(B, pred)*, gdzie *pred* jest zmienną pomocniczą, której wartościami są podzbiory wyrażeń behawioralnych.

$$\text{WeakDeadock}(B) := WD(B, \emptyset);$$

gdzie:

```
WD(B, pred) :=
begin
  if Succ(B) \ pred = ∅
  then
    if Succ(B) = ∅ ∧ B ∉ STOP then return(prawda) endif
  else
    for B' ∈ Succ(B) do
      if WD(B', pred ∪ Succ(B)) then return(prawda) endif
    endfor;
  return(fałsz);
end
```

Przeprowadzona analiza blokad dotyczyła wyrażeń behawioralnych bazowej wersji języka LOTOS, abstrahującej od wartości komunikowanych pomiędzy procesami. Komunikacja wartości może być przyczyną nowych sytuacji, w których następują blokady. Rozpatrzmy przykład prostego wyrażenia

$$a ! \text{true}; \text{stop} \mid [a] \mid a ! \text{false}; \text{stop}$$

Jest to wyrażenie z silną blokadą, chociaż jego odpowiednik w bazowym LOTOSie

$$a; \text{stop} \mid [a] \mid a; \text{stop}$$

jest wyrażeniem wolnym od blokad. Wynika z tego oczywisty wniosek, że przeprowadzona analiza określa warunki konieczne, ale niewystarczające do oceny zachodzenia blokad w wyrażeniach behawioralnych. Równie oczywiste jest to, że nie jest możliwa analiza blokad dla pełnego LOTOSa oparta wyłącznie na analizie tekstowej, bez analizy obliczenia wyrażenia.

Warto natomiast zaznaczyć, że przedstawione algorytmy wykrywania blokad można, po prostej korekcie, zastosować do bazowego, oryginalnego LOTOSa. Korekta wynika z różnicy semantyk, wyrażających się funkcją *Succ*.

Związki pomiędzy blokadami w obu wersjach języka są oczywiste: jeżeli dla wyrażenia behawioralnego B w wersji oryginalnej zachodzi $SD(B)$ (albo $WD(B)$), to także zachodzi $SD(B)$ (albo $WD(B)$) w wersji zmodyfikowanej.

8.4. Uwagi końcowe

W tym rozdziale omówiono specyficzny problem ukrytych blokad w standardowej semantyce języka LOTOS, wynikający z niekonsekwentnego traktowania komunikacji między procesami. Skorygowana semantyka, pozbawiona tego defektu, była przedmiotem artykułu [Huzar, Kuźniarz 1993]. Raport [Huzar, Kuźniarz, Łach 1997] przedstawia implementację algorytmów wykrywania blokad w zmodyfikowanym języku LOTOS. Implementację oparto na języku programowania ML [Milner, Tofte, Harper 1990].

Warto zaznaczyć, że w języku LOTOS problem blokad występuje niezależnie od omawianych tu blokad ukrytych. Na przykład takim wyrażeniem z blokadą jest

$$\mathbf{hide} \ g, g_1 \ \mathbf{in} \ g; B_1 \parallel [g] \ g_1; B_2$$

Do wykrywania blokad w standardowym LOTOSie można stosować opisany w poprzednim rozdziale pakiet CADP [Garavel, Lang, Mateescu 2001].

9. Rozszerzenia czasowe LOTOSa

9.1. Wstęp

Przedstawiony w poprzednich rozdziałach język LOTOS nie pozwala na specyfikację systemów czasu rzeczywistego. Czas może być modelowany tylko jakościowo za pomocą akcji wewnętrznych. W tym rozdziale zaprezentowano dwa podejścia do czasowego rozszerzenia języka LOTOS. Podejścia te różnią się sposobem modelowania akcji.

Pierwsze podejście zakłada, że akcje są natychmiastowe, a ich własności czasowe określają przedziały czasu, w których akcje te mogą zachodzić. Dokładniej: z akcją komunikacyjną wiąże się przedział czasu, w którym ta akcja *może* nastąpić, natomiast z akcją wewnętrzną jest związany przedział czasu, w którym akcja *musi* zajść. Podejście to było prezentowane między innymi w artykułach: [Bolognesi, Lucidi 1992], [Leduc, Leonard 1992], [Quemada, Azcorra, Frutos 1990], znalazło też odbicie w nowej wersji języka E-LOTOS [ISO/IEC FDIS 15437]. Wspólnym elementem tych prac było założenie, że wszystkie rodzaje akcji są akcjami natychmiastowymi.

Dla akcji komunikacyjnej wprowadza się notację

$$g \{t \text{ in } t^-. t^+\} \quad (9.1)$$

która oznacza, że akcja może zajść w przedziale czasu $t^-..t^+$, gdzie $0 < t^- \leq t^+$. Jeżeli akcja na bramce g nie zajdzie w zadanym przedziale czasu, to po jego upływie już zajść nie może.

Podobną notację wprowadza się dla akcji wewnętrznej i akcji **exit**

$$i \{t \text{ in } t^-. t^+\} \quad \text{exit} \{t \text{ in } t^-. t^+\} \quad (9.2)$$

W odróżnieniu od akcji komunikacyjnej, notacja oznacza, że akcja musi zajść w przedziale czasu $t^-..t^+$, gdzie $0 < t^- \leq t^+$. Jeżeli akcja nie zajdzie przed chwilą t^+ , to musi zajść w chwili t^+ .

Opis akcji czasowych i komunikacji zachodzącej w czasie rzeczywistym wymaga pojęcia chwil i przedziałów czasowych. Chwile czasowe będą się odnosić do momentów rozpoczynania i kończenia akcji, przedziały zaś będą się odnosić do opóźnień i czasów przeterminowania. Zakładamy przy tym strukturę liniowego i gęstego czasu absolutnego. Dla ukonkretnienia przyjmujemy, że dziedziną takiej struktury jest zbiór liczb wymiernych, dalej oznaczany symbolem *Time*. Strukturę taką można zdefiniować jako typ abstrakcyjny [Huzar, Magott 1997a].

Przedstawione podejście to ma tę zaletę, że pozwala na zachowanie zgodności z LOTOSem bezczasowym, jego wadą wydaje się brak naturalności, co może powodować większą złożoność specyfikacji.

Drugie podejście do czasowego rozszerzenia LOTOSa zakłada istnienie akcji czasowych i było przedstawiane w pracach: [Huzar 1991], [Huzar, Magott 1995a], [Huzar, Magott 1995b], [Huzar, Magott 1996], [Huzar, Magott 1997a], [Huzar, Magott 1997b]. Własności czasowe akcji, podobnie jak w przypadku języka RT-CSS (rozdz. 3.), są wyrażane przez zbiór dopuszczalnych czasów wykonywania się akcji.

Dla uproszczenia rozważania będą odnoszone do takiej samej bazowej wersji języka LOTOS, jaką rozpatrywano w rozdziale 8. (8.5). Czasowe rozszerzenie tej wersji będzie oznaczane dalej jako język RT-B-LOTOS. Rozszerzenie polega na wprowadzeniu dwóch elementów:

- zastąpieniu akcji (bezczasowych) akcjami czasowymi,
- wprowadzeniu nowego rodzaju akcji, zwanej akcją przeterminowania.

Akcje na bramce g (podobnie akcje wewnętrzne i) będą zastąpione akcjami czasowymi $g[\Delta]$ (odpowiednio $i[\Delta]$), gdzie $\Delta \subseteq \text{Time}$ oznacza zbiór dopuszczalnych czasów wykonania akcji czasowej. Wykonaniu akcji czasowej towarzyszą takie dwie chwile czasowe: rozpoczęcia t_1 oraz zakończenia t_2 , że $t_2 - t_1 \in \Delta$. Dla uproszczenia rozważań będziemy zakładać, że różnym akcjom czasowym na bramce g będzie zawsze odpowiadać taki sam zbiór Δ , różnym czasowym akcjom wewnętrznym mogą natomiast odpowiadać różne zbiory Δ .

Zakładamy, że akcje czasowe są nieprzerywalne: akcja raz rozpoczęta musi być wykonana do końca.

Komunikacja zbioru n akcji czasowych na bramce g przebiega w dwóch fazach. W pierwszej fazie akcje oczekują na synchronizację. Synchronizacja następuje w najwcześniejszej możliwej chwili, zgodnie z zasadą ASAP (*As Soon As Possible*). W chwili t_1 nastąpienia synchronizacji, co będzie modelowane zajściem zdarzenia czasowego $\langle t_1, g\text{-beg} \rangle$, rozpoczyna się faza druga – faza wymiany danych, która trwa do chwili t_2 , takiej, że $t_2 - t_1 \in \Delta$. Zakończenie tej fazy będzie modelowane zajściem zdarzenia czasowego $\langle t_2, g\text{-end} \rangle$.

Nowa akcja czasowa – akcja przeterminowania – ma postać **timeout**(δ), gdzie $\delta \in \text{Time}$. Jest to akcja, której realizacja rozpoczyna się w momencie jej zaoferowania t_1 i trwa przez odcinek czasu δ . Rola tej akcji polega na sygnalizacji upływu odcinka czasu długości δ , mierzonego od momentu rozpoczęcia akcji. Wykonanie akcji przeterminowania będzie modelowane tylko zajściem zdarzenia czasowego $\langle t_2, \text{timeout} \rangle$, określającego jej zakończenie w chwili $t_2 = t_1 + \delta$, jej rozpoczęcie natomiast nie będzie wyróżniane odrębnym zdarzeniem.

W tym rozdziale przedstawiono oba podejścia do czasowego rozszerzenia LOTOSa, przy czym – w celu uproszczenia – ograniczono się tylko do wersji bazowej języka, dodatkowo z wyłączeniem niektórych konstrukcji, na przykład **choice**, **let**, **accept**. Podejście pierwsze jest reprezentowane przez TE-B-LOTOS, a drugie przez RT-B-LOTOS.

9.2. Składnia i semantyka języka TE-B-LOTOS

Uproszczona, ograniczona do części bazowej, bez komunikacji danych, wersja języka TE-B-LOTOS ma następującą składnię:

$$\begin{aligned}
 Q ::= & \text{stop} \mid \text{exit}\{t \text{ in } d^-..d^+\} \mid g\{t \text{ in } d^-..d^+\}; Q \mid i\{t \text{ in } d^-..d^+\}; Q_1 \mid \text{wait}(d); Q \mid \\
 & Q_1 \parallel Q_2 \mid Q_1 \parallel [R] \parallel Q_2 \mid \text{hide } R \text{ in } Q \mid Q_1 >> Q_2 \mid Q_1 [> Q_2 \mid p[g_1, \dots, g_n] \mid \\
 & (Q) \mid \text{Age}(d, Q)
 \end{aligned} \tag{9.3}$$

Symbole Q, Q_1, Q_2 są metazmiennymi, oznaczającymi wyrażenia behawioralne; będą one często używane zamiast symboli $B, B_1, \dots$, w celu bezpośredniego odróżnienia wyrażen języka TE-B-LOTOS od języka RT-B-LOTOS.

Ostatni element przedstawionej składni $\text{Age}(d, Q)$ nie należy do języka TE-B-LOTOS, to znaczy konstrukcji tej nie używa się podczas tworzenia specyfikacji. Jest on natomiast stosowany jako element pomocniczy podczas definiowania semantyki języka, dlatego, ściśle traktując, składnia przedstawia rozszerzony język TE-B-LOTOS, nazywany dalej TE-B-LOTOS⁺. Zbiór wyrażen behawioralnych języka TE-B-LOTOS⁺ będzie oznaczany przez $TEBeh^+$, a zbiór wyrażen języka TE-B-LOTOS – przez $TEBeh$.

Deklaracje procesów będą miały postać

$$\text{process } P[S] : \text{funkcjonalność} := P \text{ endproc}$$

gdzie: $P \in Proc$, $S \in SeqG$ oraz $P \in TEBeh$, a *funkcjonalność* przyjmuje jedną z postaci **exit** albo **noexit**. Zbiór deklaracji procesów będzie oznaczany przez $Decl$.

Specyfikacja $TESpec$ w języku TE-B-LOTOS jest określona tak samo, jak specyfikacja w LOTOSie bazowym, to znaczy składa się z dwóch elementów:

- czasowego wyrażenia behawioralnego $P \in TEBeh$,
- zbioru deklaracji procesów $Decl = \{D_1, \dots, D_n\}$. Elementarne procesy **stop** i **exit** nie wymagają, oczywiście, deklaracji. Zakładamy przy tym, że realizacja procesu **exit** jest natychmiastowa, proces **stop** trwa natomiast nieskończenie długo.

Semantyka języka jest wyznaczana przez etykietowany system przejść

$$TES(RTSpec) = \langle TEBeh^+, TEvent, TETrans, P \rangle \tag{9.4}$$

gdzie:

$$TEvent = \{g \mid g \in G \cup \{\text{exit}\}\} \cup \{i\} \cup \{d \mid d \in Time\}$$

$$TETrans = \{\xrightarrow{e} \subseteq TEBeh^+ \times TEBeh^+ \mid e \in G \cup \{\text{exit}\} \cup \{i\}\}$$

$$\cup \{\xrightarrow{d} \subseteq TEBeh^+ \times TEBeh^+ \mid d \in Time\}$$

jest zbiorem relacji przejść.

Wyróżnia się dwa rodzaje przejść: zdarzeniowe $\xrightarrow{e}$ i czasowe $\xrightarrow{d}$.

Przejście zdarzeniowe $Q \xrightarrow{e} Q'$ oznacza, że wyrażenie Q może się zaangażować w wykonanie akcji e , a po jej wykonaniu dalsze zachowanie jest określone przez wyrażenie Q' . Przejście czasowe $Q \xrightarrow{d} Q'$ oznacza, że wyrażenie Q może być opóźniane przez odcinek czasu o długości d , a dalsze jego zachowanie jest określone przez wyrażenie Q' .

Dalej będą stosowane oznaczenia:

$Q \xrightarrow{e}$ oznacza, że istnieje Q' takie, że $Q \xrightarrow{e} Q'$,

$Q \not\xrightarrow{e}$ oznacza, że nie istnieje Q' takie, że $Q \xrightarrow{e} Q'$

$[t ::= t']Q$ oznacza, że ograniczenia czasowe dotyczące akcji występujących w wyrażeniu Q są zmodyfikowane tekstowo w taki sposób, że zmienna t jest zastępowana tekstowo wyrażeniem t' .

Zbiór relacji przejść *TETrans* jest definiowany rekursywnie za pomocą przedstawionego niżej zestawu aksjomatów i reguł.

Aksjomaty dla prefiksowania akcją komunikacyjną:

$$g\{t \text{ in } 0..d^+\}; Q_1 \xrightarrow{g} [t ::= 0]Q_1 \quad (TE\text{-}pre_{comm-1})$$

$$g\{t \text{ in } d^- + d..d^+ + d\}; Q_1 \xrightarrow{d} g\{t \text{ in } d^-..d^+\}; [t ::= t + d]Q_1 \quad (TE\text{-}pre_{comm-2})$$

$$g\{t \text{ in } d^-..d^+\}; Q_1 \xrightarrow{d} \text{stop} \quad \text{gdy } d > d^+ \quad (TE\text{-}pre_{comm-3})$$

$$g\{t \text{ in } d^-..d^+ + d\}; Q_1 \xrightarrow{d} g\{t \text{ in } 0..d^+\}; [t ::= t + d]Q_1 \quad \text{gdy } d > d^- \quad (TE\text{-}pre_{comm-4})$$

Aksjomat (*TE-pre_{comm-3}*) odnosi się do sytuacji, gdy akcja na bramce g nie została wykonana w przedziale czasu $d^-..d^+$. Oznacza to zablokowanie dalszych obliczeń całego wyrażenia prefiksowanego akcją na bramce g .

Aksjomaty dla prefiksowania akcją wewnętrzną:

$$i\{t \text{ in } 0..d^+\}; Q_1 \xrightarrow{i} [t ::= 0]Q_1 \quad (TE\text{-}pre_{int-1})$$

$$i\{t \text{ in } d^- + d..d^+ + d\}; Q_1 \xrightarrow{d} i\{t \text{ in } d^-..d^+\}; [t ::= t + d]Q_1 \quad (TE\text{-}pre_{int-2})$$

$$i\{t \text{ in } d^-..d^+ + d\}; Q_1 \xrightarrow{d} i\{t \text{ in } 0..d^+\}; [t ::= t + d]Q_1 \quad \text{gdy } d > d^- \quad (TE\text{-}pre_{int-3})$$

Aksjomaty dla procesu *exit*:

$$\text{exit}\{t \text{ in } 0..d^+\} \xrightarrow{\text{exit}} \text{stop} \quad (TE\text{-}exit_1)$$

$$\text{exit}\{t \text{ in } d^- + d..d^+ + d\}; Q_1 \xrightarrow{d} \text{exit}\{t \text{ in } d^-..d^+\} \quad (TE\text{-}exit_2)$$

$$\text{exit}\{t \text{ in } d^-..d^+\}; Q_1 \xrightarrow{d} \text{stop} \quad \text{gdy } d > d^+ \quad (\text{TE-exit}_3)$$

$$\text{exit}\{t \text{ in } 0..d^+ + d\}; Q_1 \xrightarrow{d} \text{exit}\{t \text{ in } 0..d^+\} \quad (\text{TE-exit}_4)$$

Aksjomat i reguły dla prefiksowania opóźnieniem:

$$\frac{P \xrightarrow{g} P'}{\text{wait}(0); P \xrightarrow{g} P'} \quad (\text{TE-wait}_1)$$

$$\text{wait}(d' + d); P \xrightarrow{d} \text{wait}(d'); P' \quad (\text{TE-wait}_2)$$

$$\frac{P \xrightarrow{d} P'}{\text{wait}(d'); P \xrightarrow{d+d'} P'} \quad (\text{TE-wait}_3)$$

Reguły dla wyboru $B \equiv P[]Q$:

$$\frac{P \xrightarrow{g} P'}{B \xrightarrow{g} P'} \quad (\text{TE-choice}_1)$$

$$\frac{Q \xrightarrow{g} Q'}{B \xrightarrow{g} P'} \quad (\text{TE-choice}_2)$$

$$\frac{P \xrightarrow{d} P' \quad Q \xrightarrow{d} Q'}{B \xrightarrow{d+d'} P'[]Q'} \quad (\text{TE-choice}_3)$$

Reguły dla prefiksowania przesunięcia czasowego:

$$\frac{P \xrightarrow{d} P'' \quad P'' \xrightarrow{g} P'}{\text{Age}(d, P) \xrightarrow{g} P'} \quad (\text{TE-age}_1)$$

$$\frac{P \xrightarrow{d+d'} P'}{\text{Age}(d', P) \xrightarrow{d} P'} \quad (\text{TE-age}_2)$$

$\text{Age}(d, P)$ zachowuje się tak, jakby zachowywało się wyrażenie P po upływie odcinka czasu d . Jeśli $P \xrightarrow{d}$, oznacza to, że $\text{Age}(d, P)$ jest procesem nieaktywnym, równoważnym **stop**.

Reguły dla złożenia równoległego $B \equiv P|[S]|Q$:

$$\frac{P \xrightarrow{g} P'}{B \xrightarrow{g} P'|[S]|Q} \quad (\text{TE-par}_1)$$

$$\frac{Q \xrightarrow{g} Q'}{B \xrightarrow{g} P|[S]|Q'} \quad (\text{TE-par}_2)$$

$$\frac{P \xrightarrow{g} P' \quad Q \xrightarrow{g} Q'}{B \xrightarrow{g} P' \parallel [S] \parallel Q'} \quad (TE-par_3)$$

$$\frac{P \xrightarrow{d} P' \quad Q \xrightarrow{d} Q'}{B \xrightarrow{d} P' \parallel [S] \parallel Q'} \quad (TE-par_4)$$

Reguły dla przesłonięcia $B \equiv \text{hide } S \text{ in } P$:

$$\frac{P \xrightarrow{g} P'}{B \xrightarrow{g} \text{hide } S \text{ in } P'} \quad g \notin \text{set}(S) \quad (TE-hide_1)$$

$$\frac{P \xrightarrow{g} P'}{B \xrightarrow{i} \text{hide } S \text{ in } P'} \quad g \in \text{set}(S) \quad (TE-hide_2)$$

$$\frac{P \xrightarrow{d} P' \quad \forall g \in \text{set}(S) \bullet (P \xrightarrow{g} \wedge \forall P'' \forall d' < d \bullet (P \xrightarrow{d'} P'' \Rightarrow P'' \xrightarrow{g}))}{B \xrightarrow{d} \text{hide } S \text{ in } P'} \quad (TE-hide_3)$$

Ostatnia reguła oznacza, że przesunięcie czasu może nastąpić, co najwyżej, do najwcześniejszego wystąpienia akcji komunikacyjnej, a akcje na przesłoniętych bramkach są wykonywane zgodnie z zasadą maksymalnego postępu (*as soon as possible*).

Reguły dla złożenia aktywowującego $B \equiv P >> Q$:

$$\frac{P \xrightarrow{g} P'}{B \xrightarrow{g} P' >> Q} \quad g.event \neq exit \quad (TE-accept_1)$$

$$\frac{P \xrightarrow{exit} P'}{B \xrightarrow{i} P' >> Q} \quad (TE-accept_2)$$

$$\frac{P \xrightarrow{d} P', P \xrightarrow{exit} \wedge \forall P'' \forall d' < d \bullet (P \xrightarrow{d'} P'' \Rightarrow P'' \xrightarrow{exit})}{B \xrightarrow{d} P' >> Q} \quad (TE-accept_3)$$

Ostatnia reguła oznacza, że przesunięcie czasu może nastąpić pod warunkiem, że nie jest realizowana akcja *exit*; akcja ta jest wykonywana zgodnie z zasadą maksymalnego postępu.

Reguły dla złożenia deaktywowującego $B \equiv P [> Q$:

$$\frac{P \xrightarrow{g} P'}{B \xrightarrow{g} P' [> Q} \quad g.event \neq exit \quad (TE-dis_1)$$

$$\frac{Q \xrightarrow{g} Q'}{B \xrightarrow{g} Q'} \quad (TE-dis_2)$$

$$\frac{P \xrightarrow{\text{exit}} P'}{B \xrightarrow{\text{exit}} P'} \quad (\text{TE-dis}_3)$$

$$\frac{P \xrightarrow{d} P', Q \xrightarrow{d} Q'}{B \xrightarrow{d} P' \gg Q'} \quad (\text{TE-dis}_4)$$

Reguły dla wywołania procesu $B \equiv p[h_1, \dots, h_n]$:

$$\frac{Q[g_1 ::= h_1, \dots, g_n ::= h_n] \xrightarrow{g} Q'}{B \xrightarrow{g} Q'} \quad (\text{TE-inst}_1)$$

$$\frac{Q[g_1 ::= h_1, \dots, g_n ::= h_n] \xrightarrow{d} Q'}{B \xrightarrow{d} Q'} \quad (\text{TE-inst}_2)$$

9.3. Wybrane własności języka TE-B-LOTOS

Własności języka TE-B-LOTOS są zgodne z własnościami języka LOTOS. Należą do nich własności zdefiniowane dla LOTOSa w podrozdziale 7.2.

Zachodzi też twierdzenie o ekspansji. Wprowadzimy następujące oznaczenia:

$\sum_{i \in I} P_i$ jest skrótem zapisu $P_{i_1} [] \dots [] P_{i_n}$, gdzie $i_1, \dots, i_n \in I$.

$\sum_{i \in I} \{P_i \mid c\}$, gdzie c jest wyrażeniem logicznym, wyznacza podzbiór podwyrażeń składowych wyrażenia $\sum_{i \in I} P_i$, które spełniają warunek c .

Niech będą dane dwa czasowe wyrażenia behawioralne w normalnej sekwencyjnej postaci normalnej:

$$P \equiv \sum_{i \in I} a_i \{t_i \text{ in } d_i^- .. d_i^+\}; P_i [] \sum_{j \in J} i \{t_j \text{ in } d_j^- .. d_j^+\}; P_j \quad I \cap J = \emptyset \quad (9.5)$$

$$Q \equiv \sum_{k \in K} b_k \{t_k \text{ in } d_k^- .. d_k^+\}; Q_k [] \sum_{l \in L} i \{t_l \text{ in } d_l^- .. d_l^+\}; Q_l \quad K \cap L = \emptyset$$

Twierdzenie o ekspansji pokazuje, że różne złożenia takich wyrażen daje się również wyrazić w sekwencyjnej postaci normalnej.

Twierdzenie 9.1

Zachodzą następujące równoważności w sensie relacji silnej bisymulacji:

dla złożenia równoległego:

$$\begin{aligned}
 P \mid [S] \mid Q \sim & \sum_{i \in I} \{a_i \{t_i \text{ in } d_i^- .. d_i^+\}; (P_i \mid [S] \mid \text{Age}(t_i, Q)) \mid a_i.\text{event} \notin \text{set}(S) \cup \{\text{exit}\}\} \\
 & \square \sum_{k \in K} \{b_k \{t_k \text{ in } d_k^- .. d_k^+\}; (Q_k \mid [S] \mid \text{Age}(t_k, P)) \mid b_k.\text{event} \notin \text{set}(S) \cup \{\text{exit}\}\} \\
 & \square \sum_{i \in I} \{c \{t \text{ in } d_i^- .. d_i^+\}; (P_i \mid [S] \mid [t_k ::= t_i] Q_k) \mid c = a_i = b_k \wedge \\
 & \quad c_i.\text{event} \in \text{set}(S) \wedge k \in K\} \\
 & \square \sum_{j \in J} i_j \{t_j \text{ in } d_i^- .. d_i^+\}; (P_j \mid [S] \mid \text{Age}(t_j, Q)) \\
 & \square \sum_{l \in L} i_l \{t_l \text{ in } d_i^- .. d_i^+\}; (\text{Age}(t_l, P) \mid [S] \mid Q_l)
 \end{aligned}$$

dla złożenia deaktywującego:

$$\begin{aligned}
 P \mid > Q \sim & Q \\
 & \square \sum_{i \in I} \{a_i \{t_i \text{ in } d_i^- .. d_i^+\}; (P_i \mid > \text{Age}(t_i, Q)) \mid a_i.\text{event} \notin \text{set}(S) \cup \{\text{exit}\}\} \\
 & \square \sum_{i \in I} \{a_i \{t_i \text{ in } d_i^- .. d_i^+\}; P_i \mid a_i.\text{event} = \text{exit}\} \\
 & \square \sum_{j \in J} i_j \{t_j \text{ in } d_i^- .. d_i^+\}; (P_j \mid > \text{Age}(t_j, Q))
 \end{aligned}$$

dla złożenia aktywującego:

$$\begin{aligned}
 P >> Q \sim & \sum_{i \in I} a_i \{t_i \text{ in } d_i^- .. d_i^+\}; (P_i >> \text{Age}(t_i, Q)) \\
 & \square \sum_{j \in J} i_j \{t_j \text{ in } d_i^- .. d_i^+\}; (P_j >> \text{Age}(t_j, Q))
 \end{aligned}$$

dla przestłonięcia:

$$\begin{aligned}
 \text{hide } S \text{ in } P \sim & \sum_{i \in I} \{a_i \{t_i \text{ in } d_i^- .. d_i^+\}; \text{hide } S \text{ in } P_i \mid a_i.\text{event} \notin \text{set}(S) \cup \{\text{exit}\}\} \\
 & \square \sum_{i \in I} \{i \{t_i \text{ in } d_i^- .. d_i^+\}; \text{hide } S \text{ in } P_i \mid a_i.\text{event} \in \text{set}(S) \cup \{\text{exit}\}\} \\
 & \square \sum_{j \in J} i_j \{t_j \text{ in } d_i^- .. d_i^+\}; \text{hide } S \text{ in } P_j
 \end{aligned}$$

dla opóźnienia:

$$\begin{aligned}
 \text{wait}(d); P \sim & \sum_{i \in I} a_i \{t_i \text{ in } d_i^- .. d_i^+\}; [t_i ::= t_i - d] P_i \\
 & \square \sum_{j \in J} i_j \{t_j \text{ in } d_i^- .. d_i^+\}; [t_i ::= t_i - d] P_j
 \end{aligned}$$

dla przesunięcia czasowego:

$$\begin{aligned} Age(d, P) \sim & \sum_{i \in I} a_i \{t_i \text{ in } d_i^- - d..d_i^+ - d\}; [t_i ::= t_i + d] P_i \\ & [] \sum_{j \in J} i_j \{t_j \text{ in } d_i^- - d..d_i^+ - d\}; [t_i ::= t_i + d] P_j \end{aligned}$$

9.4. Składnia i nieformalna semantyka języka RT-B-LOTOS

Wprowadźmy oznaczenia:

G – zbiór nazw bramek obserwowalnych,
 $Proc$ – zbiór nazw procesów,
 $Time$ – dziedzina chwil czasowych,
 $TBeh$ – zbiór czasowych wyrażeń behawioralnych.

Zbiór wszystkich akcji czasowych jest zdefiniowany jako suma mnogościowa

$$Act = \{g[\Delta] \mid g \in G, \Delta \subseteq Time\} \cup \{i[\Delta] \mid \Delta \subseteq Time\} \cup \{\text{timeout}(\delta) \mid \delta \in Time\} \quad (9.6)$$

Niech $SeqG$ oznacza zbiór takich ciągów nad zbiorem G , w których nie ma powtórzeń dwóch tych samych nazw bramek, to znaczy

$$SeqG = \{g_1, \dots, g_n \mid g_i \in G, g_i \neq g_j \text{ dla } i \neq j, n = 0, 1, \dots\} \quad (9.7)$$

Jeżeli $R \in SeqG$ oraz $R = g_1, \dots, g_n$, to $set(R) = \{g_1, \dots, g_n\}$ oraz $len(R) = n$ dla $n \geq 0$.

Deklaracje procesów będą miały postać

$$\text{process } P[S] : \text{funkcjonalność} := B \text{ endproc}$$

gdzie: $P \in Proc$, $S \in SeqG$ oraz $B \in TBeh$, a *funkcjonalność* przyjmuje jedną z postaci **exit** albo **noexit**. Zbiór deklaracji procesów będzie oznaczany przez $Decl$.

Instancją procesu będzie wyrażenie postaci $P[R]$, gdzie $R \in SeqG$ oraz $len(R) = len(S)$. Wyrażenie to jest szczególną postacią czasowego wyrażenia behawioralnego. Zbiór wszystkich czasowych wyrażeń behawioralnych $TBeh$, w notacji BNF, jest określony następująco:

$$\begin{aligned} B ::= & \text{stop} \mid \text{exit} \mid g[\Delta]; B_1 \mid i[\Delta]; B_1 \mid \text{timeout}(\delta); B_1 \mid B_1 >> B_2 \mid B_1 [] B_2 \mid \\ & \text{hide } R \text{ in } B_1 \mid B_1 [> B_2 \mid B_1 |[R]| B_2 \mid P[R] \mid (B_1) \end{aligned} \quad (9.8)$$

gdzie: $B, B_1, B_2 \in TBeh$, $P \in Proc$, $R \in SeqG$. Kolejność wprowadzenia operatorów odpowiada malejącej kolejności ich priorytetów.

Nieformalnie semantyka czasowych wyrażeń behawioralnych, oprócz czasowej rozciągłości akcji, jest podobna do semantyki wyrażeń LOTOSa standardowego. Dodat-

kowe różnice wynikają z wprowadzenia akcji przeterminowania. Bezpośrednio sama akcja **timeout**(δ) oznacza opóźnienie o odcinek czasu długości δ . Obecność natomiast tej akcji w różnych kontekstach ma wpływ na przebieg obliczeń. Rozpatrzmy na przykład trzy wyrażenia behawioralne, w których może ona wystąpić:

$$g[\Delta]; B_1 [] \text{timeout}(\delta); B_2 \quad (9.9)$$

$$B_1 [> \text{timeout}(\delta); B_2 \quad (9.10)$$

$$B_1 |[R]| \text{timeout}(\delta); B_2 \quad (9.11)$$

Pierwsze wyrażenie (9.9) reprezentuje następujące zachowanie: jeżeli w odcinku czasu o długości δ , mierzonym od momentu jego przygotowania, rozpocznie się realizacja akcji czasowej na bramce g , to dalsze zachowanie – jak przy konstrukcji wyboru – przebiegnie zgodnie z wyrażeniem B_1 , w przypadku natomiast przeciwnym w momencie upływu odcinka czasu δ rozpocznie się realizacja wyrażenia B_2 .

Drugie wyrażenie (9.10) reprezentuje następujące zachowanie: jeżeli w odcinku czasu o długości δ , mierzonym od momentu jego przygotowania, nie nastąpi zakończenie wyrażenia B_1 , to w momencie upływu odcinka czasu δ nastąpi jego przerwanie i rozpocznie się realizacja wyrażenia B_2 .

W przypadku natomiast trzeciego wyrażenia (9.11) akcja przeterminowania odgrywa tylko rolę opóźnienia momentu rozpoczęcia wykonania wyrażenia B_2 .

Dwa pierwsze wyrażenia są typowymi konstrukcjami, które można spotkać w językach programowania czasu rzeczywistego, na przykład w języku Ada 95 [Huzar i inni 1998].

Definiowany język zawiera te same źródła niedeterminizmu, co LOTOS standardowy – dotyczy to konstrukcji wyboru oraz selekcji zestawów komunikujących się procesów. W rozważaniach przyjmujemy niedeterministyczny wybór czasów wykonania akcji czasowej ze zbioru Δ , określającego dopuszczalne czasy wykonania. Możliwe są też inne sposoby, na przykład wybór losowy zgodnie z zadany rozkładem prawdopodobieństwa, tak jak to przedstawiono w artykule [Huzar, Magott 1997a].

Specyfikacja *RTSpec* w języku RT-B-LOTOS jest określona tak samo jak specyfikacja w LOTOSie bazowym, to znaczy składa się z dwóch elementów:

- czasowego wyrażenia behawioralnego $B \in TBeh$,
- zbioru deklaracji procesów $\{D_1, \dots, D_n\}$, gdzie $D_i \in Decl$. Elementarne procesy **stop** i **exit** nie wymagają oczywiście deklaracji. Zakładamy przy tym, że realizacja procesu **exit** jest natychmiastowa, proces **stop** trwa natomiast nieskończenie długo.

9.5. Semantyka formalna języka RT-B-LOTOS

Semantyka operacyjna specyfikacji jest definiowana w sposób strukturalny na podstawie definicji składniowych wyrażeń behawioralnych.

Wprowadzenie czasu powoduje, że w wyrażaniu semantyki konieczne staje się rozważanie przejściowych sytuacji, w których oczekuje się jedynie na upływ czasu. Z tego względu wprowadza się pomocniczą akcję opóźnienia, oznaczaną $\mathbf{delay}(g, \delta)$, gdzie $g \in G \cup \{i\}$, której jedynym znaczeniem jest opóźnianie akcji na bramce g o zadany odcinek czasu δ .

Wprowadzenie akcji opóźniającej rozszerza zbiór wyrażen behawioralnych. Rozszerzony zbiór czasowych wyrażen behawioralnych, oznaczany $XTBeh$, w notacji BNF, jest zdefiniowany następująco: Niech B jest metazmienną reprezentującą wyrażenie behawioralne ze zbioru $TBeh$, a XB , XB_1 , XB_2 są metazmiennymi reprezentującymi zbiór wyrażen $XTBeh$, wówczas

$$XB ::= B \mid \mathbf{delay}(g, \delta); B \mid XB_1 \parallel [S] \parallel XB_2 \mid \mathbf{hide } S \text{ in } XB \mid XB >> B \mid XB [> B \quad (9.12)$$

Rozszerzony zbiór wyrażen behawioralnych służy do przedstawiania konfiguracji zachodzących w trakcie obliczeń – są to zarówno konfiguracje zasadnicze, reprezentowane przez wyrażenia behawioralne języka RT-B-LOTOS, jak i konfiguracje pomocnicze, reprezentowane przez rozszerzone wyrażenia behawioralne. Składnia wyrażen rozszerzonych jest zdefiniowana tak, aby uwzględniać tylko możliwe do osiągnięcia konfiguracje pomocnicze. Nie jest na przykład możliwa do osiągnięcia konfiguracja, która byłaby reprezentowana przez wyrażenie $XB_1 [] XB_2$, gdyż oznaczałaby przejściową konfigurację, w której trwa realizacja wyrażen po obu stronach operatora wyboru. Jest to oczywiście niemożliwe, gdyż realizacja wyrażenia behawioralnego z operatorem wyboru prowadzi do wyboru tylko jednego wyrażenia składowego.

Wprowadzamy zbiór nazw zdarzeń czasowych *Event*, zdefiniowany jako

$$Event = \{g\text{-beg} \mid g \in G\} \cup \{g\text{-end} \mid g \in G\} \cup \{i\text{-beg}, i\text{-end}, exit, timeout\} \quad (9.13)$$

Jeżeli $S \in SeqG$, to

$$Event(S) = \{g\text{-beg} \mid g \in set(S)\} \cup \{g\text{-end} \mid g \in set(S)\} \quad (9.14)$$

Zbiór zdarzeń czasowych $TEvent$ jest zdefiniowany jako produkt

$$TEvent = Time \times Event$$

Jeżeli $e = \langle t, h \rangle \in TEvent$, to

$$e.time = t \text{ oraz } e.event = h.$$

Dla specyfikacji $RTSpec = \langle \{B, \{D_1, \dots, D_n\}\rangle$ etykietowany system przejść czasowych jest zdefiniowany jako

$$RTS(RTSpec) = \langle XBeh, TEvent, RTrans, B \rangle \quad (9.15)$$

gdzie

$$RTrans = \{ \xrightarrow{e} \subseteq XBeh \times XBeh \mid e \in TEvent \}$$
 jest zbiorem relacji przejść.

Zbiór relacji przejść $RTrans$ jest definiowany rekursywnie za pomocą przedstawionego niżej zestawu aksjomatów i reguł. W regułach występuje kilka funkcji pomocniczych.

Pierwsza z nich, funkcja o sygnaturze

$$First : XTBeh \rightarrow 2^G$$

dla danego rozszerzonego czasowego wyrażenia behawioralnego B , wyznacza zbiór bramek, które są przygotowane do komunikacji. Definicję funkcji przedstawiono w tabeli 9.1.

Tabela 9.1

B	$First(B)$
stop	$\emptyset$
exit	$\emptyset$
$g[\Delta]; B_1$	$\{g\}$
$i[\Delta]; B_1$	$\emptyset$
delay (g, δ); B_1 ($g \in G \cup \{i\}$)	$\emptyset$
timeout (δ); B_1	$\emptyset$
$B_1 [] B_2$	$First(B_1) \cup First(B_2)$
$B_1 [S] B_2$	$(First(B_1) \setminus set(S)) \cup (First(B_2) \setminus set(S))$ $\cup (First(B_1) \cap First(B_2) \cap set(S))$
hide S in B_1	$First(B_1) \setminus set(S)$
$P[R]$	$First(B_1[S::=R])$ B_1 jest treścią procesu P
$B_1 >> B_2$	$First(B_1)$
$B_1 [> B_2$	$First(B_1) \cup First(B_2)$
(B_1)	$First(B_1)$

Dwie następne funkcji o sygnaturach:

$$Age : XTBeh \times Time \rightarrow XTBeh$$

$$Next : XTBeh \rightarrow Time_{\infty}$$

służą do modelowaniu upływu czasu. Pierwsza z nich, $Age(B, \delta)$, transformuje rozszerzone czasowe wyrażenie behawioralne w nowe wyrażenie, które jest tylko rezultatem upływu czasu o odcinek δ .

Druga funkcja, $Next(B)$, dla danego rozszerzonego czasowego wyrażenia behawioralnego B , wyznacza, poczynając od danej chwili, najwcześniejszy moment, w którym

może się rozpocząć akcja czasowa, niezależnie od otoczenia wyrażenia B . Zbiór wartości funkcji $Time_\infty = Time \cup \{\infty\}$, gdzie element ∞ ma następujące własności:

$$\forall t \in Time \bullet t < \infty \wedge t + \infty = \infty$$

Jeżeli najwcześniejszy moment akcji czasowej dla wyrażenia B zależy wyłącznie od jego otoczenia, to wartość funkcji jest zdefiniowana jako $Next(B) = \infty$. Funkcje są zdefiniowane w tabeli 9.2.

Tabela 9.2

B	$Age(B, \delta)$	$Next(B)$
stop	stop	∞
exit	exit	∞
$g[\Delta]; B_1$	$g[\Delta]; B_1$	∞
$i[\Delta]; B_1$	$i[\Delta]; B_1$ ¹⁾	0
delay (g, δ_1); B_1 $g \in G \cup \{i\}$	delay ($g, \delta_1 - \delta$); B_1 ²⁾	δ_1
timeout (δ_1); B_1	timeout ($\delta_1 - \delta$); B_1 ²⁾	δ_1
$B_1 [] B_2$	$Age(B_1, \delta) [] Age(B_2, \delta)$ ³⁾	$\min(Next(B_1), Next(B_2))$
$B_1 [S] B_2$	$Age(B_1, \delta) [S] Age(B_2, \delta)$ ³⁾	$\min(Next(B_1), Next(B_2))$
hide S in B_1	hide S in $Age(B_1, \delta)$ ⁴⁾	0 gdy $First(B_1) \cap set(S) \neq \emptyset$ $Next(B_1)$ w przypadku przeciwnym
$P[R]$	$Age(B_1[S ::= R], \delta)$ ⁴⁾ B_1 jest treścią procesu P	$Next(B_1[S ::= R])$
$B_1 >> B_2$	$Age(B_1, \delta) >> B_2$ ⁴⁾	$Next(B_1)$
$B_1 [> B_2$	$Age(B_1, \delta) [> Age(B_2, \delta)$ ³⁾	$\min(Next(B_1), Next(B_2))$
(B_1)	$Age(B_1, \delta)$ ⁴⁾	$Next(B_1)$

¹⁾ Zdefiniowane tylko dla $\delta = 0$.

²⁾ Zdefiniowane tylko dla $\delta \leq \delta_1$.

³⁾ Zdefiniowane tylko dla $\delta \leq \min(Next(B_1), Next(B_2))$.

⁴⁾ Zdefiniowane tylko dla $\delta \leq \min(Next(B_1))$.

Wreszcie ostatnia funkcja pomocnicza, o sygnaturze

$$Abort : XTBeh \rightarrow XTBeh$$

dokonyuje transformacji rozszerzonego czasowego wyrażenia behawioralnego B , która odzwierciedla konsekwencje zerwania obliczeń wyrażenia. Jedynym powodem zerwania obliczenia jest zastosowanie konstrukcji złożenia deaktywującego $B_1 [> B_2$. Jeżeli wyrażenie B_2 rozpocznie swą aktywność, to musi nastąpić zakończenie wyrażenia B_1 . Ponieważ akcje czasowe są nieprzerywalne, zakończenie wyrażenia B_1 nie następuje

w momencie rozpoczęcia wyrażenia B_2 , lecz dopiero wówczas, gdy zakończą się wszystkie aktualnie wykonywane akcje czasowe wyrażenia B_1 . Taki łagodny model zrywania obliczenia (*soft abortion*) wyrażenia behawioralnego wyraża funkcja *Abort*, zdefiniowana w tabeli 9.3.

Tabela 9.3

B	$Abort(B)$
stop	stop
exit	stop
$g[\Delta]; B_1$	stop
$i[\Delta]; B_1$	stop
delay (g, δ); B_1 $g \in G \cup \{i\}$	delay (g, δ); stop
timeout (δ); B_1	stop
$B_1 [] B_2$	$Abort(B_1) [] Abort(B_2)$
$i[F_1]; B_1 [] i[F_2]; B_2$	stop
$B_1 [S] B_2$	$Abort(B_1) [S] Abort(B_2)$
hide S in B_1	hide S in $Abort(B_1)$
$P[R]$	$Abort(B_1[S ::= R])$ B_1 jest treścią procesu P
$B_1 >> B_2$	$Abort(B_1)$
$B_1 [> B_2$	$Abort(B_1) [\emptyset] Abort(B_2)$
(B_1)	$Abort(B_1)$

Proces pusty **stop** – brak aksjomatów.

Proces zakończenia $B \equiv \mathbf{exit}$

$$B \xrightarrow{\langle t, \mathbf{exit} \rangle} \mathbf{stop} \quad (RT\text{-}\mathbf{exit})$$

gdzie $t \geq 0$.

Proces **exit** wykonuje się natychmiast, jeśli tylko w jego otoczeniu istnieją już przygotowane procesy, które się z nim synchronizują. W przypadku przeciwnym proces **exit** oczekuje na synchronizację.

Prefiksowanie akcją komunikacyjną $B \equiv g[\Delta]; B_1$

$$B \xrightarrow{\langle t, g\text{-}\mathbf{beg} \rangle} \mathbf{delay}(g, \delta); B_1 \quad (RT\text{-}\mathbf{pre}_{comm})$$

gdzie $t \geq 0$, $\delta \in \Delta$ i $g \in G$.

Akcja komunikacyjna rozpoczyna się w chwili synchronizacji z otoczeniem, co może nastąpić w dowolnym momencie $t \geq 0$, i trwa odcinek czasu $\delta \in \Delta$, uzgodniony przez komunikujące się procesy.

Prefiksowanie akcją wewnętrzną $B \equiv i[\Delta]; B_1$

$$B \xrightarrow{\langle 0, i\text{-beg} \rangle} \mathbf{delay}(i, \delta); B_1 \quad (RT\text{-}pre_{int})$$

gdzie $\delta \in \Delta$.

Akcja wewnętrzna rozpoczyna się w chwili jej zaoferowania otoczeniu i trwa odcinek czasu $\delta \in \Delta$, ustalony przez proces zawierający akcję, bez wpływu otoczenia.

Prefiksowanie akcją opóźniającą $B \equiv \mathbf{delay}(g, \delta); B_1$

$$B \xrightarrow{\langle \delta, g\text{-end} \rangle} B_1 \quad (RT\text{-}pre_{delay})$$

Rozpoczęta akcja opóźniająca trwa nieprzerwanie, aż do zakończenia.

Prefiksowanie akcją przeterminowania $B \equiv \mathbf{timeout}(\delta); B_1$

$$B \xrightarrow{\langle \delta, \mathbf{timeout} \rangle} B_1 \quad (RT\text{-}pre_{timeout})$$

Rozpoczęta akcja przeterminowania kończy się po upływie zadanego odcinka czasu δ , ale – w odróżnieniu od akcji wewnętrznej – może być przerywana: wynika to z dalej przedstawionych reguł dotyczących konstrukcji wyboru i złożenia deaktywującego.

Wybór $B \equiv B_1 \mid B_2$

$$\frac{B_1 \xrightarrow{e} B'_1}{B \xrightarrow{e} B'_1} \quad (RT\text{-}choice_1)$$

jeśli $e.time \leq Next(B_2)$ oraz

$$e.event = g\text{-beg}, \text{ dla } g \in G \cup \{i\} \text{ lub } e.event = \mathbf{timeout}$$

$$\frac{B_2 \xrightarrow{e} B'_2}{B \xrightarrow{e} B'_2} \quad (RT\text{-}choice_2)$$

jeśli $e.time \leq Next(B_1)$ oraz

$$e.event = g\text{-beg}, \text{ dla } g \in G \cup \{i\} \text{ lub } e.event = \mathbf{timeout}$$

Reguły opisujące wybór sprowadzają się do stwierdzenia, że wybiera się to wyrażenie składowe, dla którego najwcześniej zajdzie zdarzenie czasowe. W przypadku równoczesnego wystąpienia zdarzeń dla obu wyrażeń składowych wybór jest niedeterministyczny.

Złożenie równoległe $B \equiv B_1 \parallel [S] \mid B_2$

$$\frac{B_1 \xrightarrow{e} B'_1}{B \xrightarrow{e} B'_1 \parallel [S] \mid \mathbf{Age}(B_2, e.time)} \quad (RT\text{-}par_1)$$

jeśli $e.event \notin Event(S) \cup \{exit\}$ oraz $e.time \leq Next(B_2)$

$$\frac{B_2 \xrightarrow{e} B'_2}{B \xrightarrow{e} \text{Age}(B_1, e.time) \parallel [S] \parallel B'_2} \quad (RT\text{-}par_2)$$

jeśli $e.event \notin \text{Event}(S) \cup \{\text{exit}\}$ oraz $e.time \leq \text{Next}(B_1)$

$$\frac{B_1 \xrightarrow{e} B'_1 \quad \frac{B_2 \xrightarrow{e} B'_2}{B \xrightarrow{e} B'_1 \parallel [S] \parallel B'_2}}{B \xrightarrow{e} B'_1 \parallel [S] \parallel B'_2} \quad (RT\text{-}par_3)$$

jeśli $e.event \in \text{Event}(S) \cup \{\text{exit}\}$ oraz $e.time \leq \min(\text{Next}(B_1), \text{Next}(B_2))$

Dwie pierwsze reguły opisują transformację złożenia równoległego, gdy nastąpi zdarzenie czasowe w jednym z wyrażeń składowych, które nie angażuje wspólnych bramek komunikacyjnych. Najwcześniejsze zajście takiego zdarzenia powoduje odpowiednią transformację składowej, w której ono zaszło; transformacja drugiej składowej polega natomiast tylko na modyfikacji wynikającej z upływu czasu.

Trzecia reguła dotyczy sytuacji, gdy zachodzi zdarzenie związane ze wspólną bramką komunikacyjną obu składowych wyrażeń.

Przesłonięcie $B \equiv \text{hide } S \text{ in } B_1$

$$\frac{B_1 \xrightarrow{\langle 0, g\text{-beg} \rangle} B'_1}{B \xrightarrow{\langle 0, i\text{-beg} \rangle} B'_1} \quad (RT\text{-}hide_1)$$

dla $g \in \text{set}(S)$

$$\frac{B_1 \xrightarrow{\langle t, g\text{-end} \rangle} B'_1}{B \xrightarrow{\langle t, i\text{-end} \rangle} B'_1} \quad (RT\text{-}hide_2)$$

dla $g \in \text{set}(S)$ oraz $t \geq 0$

$$\frac{B_1 \xrightarrow{e} B'_1}{B \xrightarrow{e} B'_1} \quad (RT\text{-}hide_3)$$

dla $e.event \notin \text{set}(S)$

Reguły dotyczące przesłonięcia wynikają z przyjęcia semantyki standardowego języka LOTOS, a nie języka zmodyfikowanego, przedstawionego w rozdziale 8. Dwie pierwsze reguły są odpowiednikiem pojedynczej reguły $(RT\text{-}hide_1)$ dla języka LOTOS i wynikają stąd, że pojedyncza akcja w języku LOTOS jest reprezentowana dwoma zdarzeniami czasowymi w RT-B-LOTOS. Trzecia reguła jest taka sama jak druga reguła dla języka LOTOS.

Instancja procesu $B \equiv P[S]$

$$\frac{B_1[R := S] \xrightarrow{e} B'_1}{B \xrightarrow{e} B'_1} \quad (RT-inst)$$

gdzie B_1 jest treścią procesu P .

Obliczenie instancji procesu przebiega tak jak obliczenie zmodyfikowanego wyrażenia B_1 , stanowiącego treść procesu. Modyfikacja polega na zastąpieniu listy bramek formalnych R listą bramek aktualnych S – oznaczenie $B_1[R ::= S]$.

Złożenie aktywujące $B \equiv B_1 >> B_2$

$$\frac{B_1 \xrightarrow{e} B'_1}{B \xrightarrow{e} B'_1 >> B_2} \quad (RT-enabl_1)$$

jeśli $e.event \neq exit$

$$\frac{B_1 \xrightarrow{\langle 0, exit \rangle} B'_1}{B \xrightarrow{\langle 0, exit \rangle} B_2} \quad (RT-enabl_2)$$

Przedstawione reguły, oprócz wprowadzenia zdarzeń czasowych, nie różnią się od reguł dla złożenia aktywującego dla języka LOTOS.

Złożenie deaktywujące $B \equiv B_1 [> B_2]$

$$\frac{B_1 \xrightarrow{e} B'_1}{B \xrightarrow{e} B'_1 [> Age(B_2, e.time)]} \quad (RT-dis_1)$$

jeżeli $e.event \neq exit$ oraz $e.time \leq Next(B_2)$

$$\frac{B_1 \xrightarrow{\langle 0, exit \rangle} B'_1}{B \xrightarrow{\langle 0, exit \rangle} B'_1} \quad (RT-dis_2)$$

$$\frac{B_2 \xrightarrow{e} B'_2}{B \xrightarrow{e} Abort(B_1) || [\emptyset] | B'_2} \quad (RT-dis_3)$$

Pierwsza reguła dotyczy sytuacji, gdy najwcześniej zachodzącym jest zdarzenie czasowe w składowej B_1 . Transformacja całego wyrażenia jest w tym przypadku podobna do sytuacji opisanej regułą $R-par_1$ dla złożenia równoległego. Druga reguła, dotycząca również składowej B_1 , ale zajścia zdarzenia kończącego tę składową, jest taka sama jak odpowiednia reguła dla języka LOTOS. Trzecia reguła jest specyficzna tym, że zajście zdarzenia rozpoczynającego B_2 uruchamia działania zmierzające do zakończenia składowej B_1 , dokładniej – do dokończenia wszystkich aktualnie trwających akcji czasowych.

Nawiasy $B \equiv (B_1)$

$$\frac{B_1 \xrightarrow{e} B'_1}{B \xrightarrow{e} B'_1} \quad (RT\text{-}brace)$$

Warto porównać sposób definiowania języka RT-B-LOTOS z definiowaniem języka TE-B-LOTOS.

Pierwsza różnica odnosi się do tranzycji: w języku RT-B-LOTOS mamy do czynienia tylko z jednym rodzajem tranzycji związanych ze zdarzeniami czasowymi, w TE-B-LOTOS są natomiast dwa rodzaje: tranzycje czasowe i tranzycje zdarzeniowe. Druga różnica dotyczy postaci reguł: w języku TE-B-LOTOS występują reguły z przesłankami negatywnymi (trzecia reguła dla przesłonięcia i dla złożenia aktywującego), których nie ma w RT-B-LOTOSie. Reguł z przesłankami negatywnymi [Groote 1990] udało się w RT-B-LOTOSie uniknąć, dzięki wprowadzeniu funkcji *Next*, która – w razie braku komunikacji z otoczeniem – wyznacza najwcześniejsze zdarzenie powodujące tranzycję.

Podobieństwem w obu sposobach definiowania jest wprowadzenie języków pomocniczych, stanowiących rozszerzenie TE-B-LOTOSa – chodzi o TE-B-LOTOS⁺, oraz RT-B-LOTOSa – chodzi o rozszerzony zbiór wyrażeń behawioralnych *XTBeh*. W obu przypadkach powodem rozszerzeń jest opis sytuacji przejściowych podczas wykonywania tranzycji.

9.6. Wybrane własności języka RT-B-LOTOSa

Etykietowany system przejść czasowych *RTS(RTSpec)* wyznacza zbiór obliczeń dla specyfikacji *RTSpec*. Obliczenie jest zdefiniowane jako skończony albo nieskończony ciąg tranzycji postaci:

$$B \xrightarrow{e_1} B_1 \xrightarrow{e_2} B_2 \dots \xrightarrow{e_n} B_n \quad (9.16a)$$

$$B \xrightarrow{e_1} B_1 \xrightarrow{e_2} B_2 \dots \xrightarrow{e_n} B_n \xrightarrow{e_{n+1}} \dots \quad (9.16b)$$

gdzie: $B_1, \dots, B_n, \dots \in XBeh$, $e_1, \dots, e_n, e_{n+1}, \dots \in TEvent$.

Semantykę akcyjną i obserwowalną definiuje się podobnie jak we wcześniej rozważanym przypadku języka LOTOS (rozdz. 5.).

Na podstawie definicji relacji tranzycji i definicji zbioru wyrażeń *XTBeh* (9.12) łatwo sprawdzić, że zachodzi własność:

Twierdzenie 9.2

Dla obliczenia skończonego postaci (9.16a) $B_n \in TBeh$.

Dowód

Wystarczy zauważyć, że dowolne wyrażenie behawioralne $B \in XTBeh \setminus TBeh$ zawiera przynajmniej jedną akcję opóźnienia **delay**. Z definicji zbioru wyrażeń $XTBeh$ (9.12) i funkcji *First* (tab. 9.1) wynika, że każda akcja opóźnienia jest elementem zbioru $First(B)$. Akcja **delay** prefiksuje pewne podwyrażenie wyrażenia B . Zachodzą tu tylko następujące możliwości:

$$B \equiv \mathbf{delay}(g, \delta); B_1$$

$$B \equiv \mathbf{delay}(g, \delta); B_1 [] B_2$$

$$B \equiv \mathbf{delay}(g, \delta); B_1 |[S]| B_2$$

$$B \equiv \mathbf{delay}(g, \delta); B_1 >> B_2$$

$$B \equiv \mathbf{delay}(g, \delta); B_1 [> B_2$$

W każdym z tych przypadków istnieje tranzycja z wyrażenia B do innego wyrażenia. Jeśli zatem $B \in XTBeh \setminus TBeh$, to B nie może być wyrażeniem końcowym skończonego obliczenia.

Należy zwrócić uwagę na to, że w definiowaniu semantyki zastosowano konwencję czasu względnego. Oznacza to, że moment zachodzenia kolejnego zdarzenia czasowego jest odniesiony do chwili osiągnięcia wyrażenia behawioralnego, z którego następuje tranzycja pod wpływem tego zdarzenia. Jest to konwencja wygodniejsza od konwencji czasu bezwzględnego, która była stosowana w definiowaniu semantyki języka RT-CCS (rozdz. 3.). Chwila t_i , dla $i = 1, 2, \dots$, w której nastąpiła tranzycja do wyrażenia B_i wynosi

$$t_i = e_1.time + \dots + e_i.time$$

Porównując definicję semantyki LOTOSa bezczasowego i czasowego, łatwo zauważyć podobieństwo reguły tranzycji. Semantyka standardowego języka LOTOS jest semantyką przepłotową, to znaczy wyznacza ona zbiór obliczeń, w których pewne akcje mogą być przedstawione w permutacji. Wprowadzenie czasu wyklucza możliwość takiego podejścia i wymaga semantyki prawdziwie równoległej (*true concurrency semantic*). Taka jest właśnie semantyka języka RT-B-LOTOS. Semantyka ta różni się od semantyki przepłotowej tym, że na zbiór obliczeń wyznaczonych przez semantykę przepłotową narzuca pewne ograniczenia na dopuszczalne ciągi akcji (zdarzeń czasowych). Ograniczenia te mają postać warunków dopuszczających stosowanie poszczególnych reguł. W warunkach tych występują, jako zasadnicze, funkcje pomocnicze związane z modelowaniem upływu czasu.

Język RT-B-LOTOS zachowuje część własności języka LOTOS (wyrażonych w p. 7.2) dotyczących silnej bisymulacji.

Twierdzenie 9.3

Zachodzą następujące związki:

Prawa dla wyboru

$$B_1 [] B_2 \sim B_2 [] B_1$$

$$B_1 [] (B_2 [] B_3) \sim (B_1 [] B_2) [] B_3$$

$$B [] B \sim B$$

$$B [] \text{stop} \sim B$$

Prawa dla złożenia równoległego

$$B_1 |[S]| B_2 \sim B_2 |[S]| B_1$$

$$B_1 |[S]| (B_2 |[R]| B_3) \sim (B_1 |[S]| B_2) |[R]| B_3$$

$$\text{exit } |[S]| \text{stop} \sim \text{stop}$$

Prawa dla złożenia aktywującego

$$\text{stop} >> B \sim \text{stop}$$

$$(B_1 >> B_2) >> B_3 \sim B_1 >> (B_2 >> B_3)$$

$$B >> \text{stop} \sim B |[\emptyset]| \text{stop}$$

Prawa dla złożenia deaktywującego

$$\text{stop } [> B \sim B$$

$$\text{exit } [> B \sim \text{exit } [] B$$

$$(B_1 [> B_2) [> B_3 \sim B_1 [> (B_2 [> B_3)$$

$$(B_1 [> B_2) [] B_2 \sim B_1 [> B_2$$

$$B [> \text{stop} \sim B$$

Prawa dla przesłonięcia

$$\text{hide } S \text{ in } B \sim \text{hide } S' \text{ in } B \quad \text{jeśli } \text{set}(S') = \text{set}(S) \cap FG(B)$$

$$\text{hide } S \text{ in } B \sim B \quad \text{jeśli } \text{set}(S) \cap FG(B) = \emptyset$$

gdzie $FG(B)$ jest zbiorem bramek, które występują w wyrażeniu B jako bramki wolne, to znaczy niezwiązane operatorem przesłonięcia (formalna definicja funkcji FG jest tu pominięta).

$$\text{hide } S \text{ in } \text{hide } S' \text{ in } B \sim \text{hide } S'' \text{ in } B$$

$$\text{jeśli } \text{set}(S'') = \text{set}(S) \cup \text{set}(S')$$

$$\text{hide } S \text{ in } g[\Delta] ; B \sim i[\Delta] ; \text{hide } S \text{ in } B \quad \text{jeśli } g \in \text{set}(S)$$

$$\text{hide } S \text{ in } g[\Delta] ; B \sim g[\Delta] ; \text{hide } S \text{ in } B \quad \text{jeśli } g \notin \text{set}(S)$$

$$\text{hide } S \text{ in } B_1 [] B_2 \sim \text{hide } S \text{ in } B_1 [] \text{hide } S \text{ in } B_2$$

$$\text{hide } S \text{ in } B_1 \parallel [S'] B_2 \sim (\text{hide } S \text{ in } B_1) \parallel [S'] (\text{hide } S \text{ in } B_2)$$

$$\text{jeśli } \text{set}(S) \cap \text{set}(S') = \emptyset$$

$$\text{hide } S \text{ in } B_1 >> B_2 \sim (\text{hide } S \text{ in } B_1) >> (\text{hide } S \text{ in } B_2)$$

$$\text{hide } S \text{ in } B_1 [> B_2 \sim (\text{hide } S \text{ in } B_1) [> (\text{hide } S \text{ in } B_2)$$

Prawo dla instancji procesu

$$P[S] \sim B[S/R] \text{ jeśli } \text{process } p[R] : \text{funkcjonalność} := B \text{ endproc}$$

Dowód

Pełny dowód, oparty na indukcji strukturą, jest długi i uciążliwy, ze względu na dużą liczbę przypadków wymagających rozważenia. Poniżej przedstawiamy tylko szkic dowodu, przy czym ograniczymy się tutaj tylko do własności dla złożenia równoległego.

Rozpatrzmy równość wyrażającą przemienność złożenia równoległego

$$B_1 \parallel [S] B_2 \sim B_2 \parallel [S] B_1$$

Wynika ona z reguł tranzykcji (*RT-par₁*), (*RT-par₂*) i (*RT-par₃*). Na podstawie tych reguł zawsze jako pierwsze do obliczenia jest wybierane zdarzenie czasowe, spośród zdarzeń oferowanych przez B_1 i przez B_2 , które występuje najwcześniej. Reguła (*RT-par₁*) dotyczy przypadku, gdy najwcześniej jest oferowane zdarzenie czasowe przez wyrażenie B_1 , które nie angażuje bramki ze zbioru $\text{set}(S)$, czyli zachodzi niezależnie od wyrażenia B_2 . Podobnie reguła (*RT-par₂*) dotyczy przypadku najwcześniejszego zdarzenia niezależnego oferowanego przez B_2 . Reguła (*RT-par₃*) dotyczy natomiast przypadku, gdy najwcześniejsze zdarzenie czasowe zachodzi z udziałem obu składowych, czyli gdy jest zaangażowana bramka ze zbioru $\text{set}(S)$. Gdy najwcześniejsze zdarzenia oferowane przez B_1 i przez B_2 lub jednocześnie przez oba te wyrażenia odnoszą się do tej samej chwili, mamy niedeterminizm, co wyraża się zbiorem obliczeń, w których zdarzenia te występują w przeplocie.

Po wykonaniu tranzykcji wynikających z uwzględnienia najwcześniejszego zdarzenia czasowego e

$$B_1 \parallel [S] B_2 \xrightarrow{e} B'_1 \parallel [S] B'_2 \text{ oraz } B_2 \parallel [S] B_1 \xrightarrow{e} B'_2 \parallel [S] B'_1$$

znajdujemy się w sytuacji podobnej do analizowanej, to znaczy kolejne zdarzenie czasowe, które mogą wygenerować wyrażenia $B'_1 \parallel [S] B'_2$ oraz $B'_2 \parallel [S] B'_1$ są takie same, czyli $B'_1 \parallel [S] B'_2 \sim B'_2 \parallel [S] B'_1$. Zatem $B_1 \parallel [S] B_2 \sim B_2 \parallel [S] B_1$.

Druga równość, wyrażająca łączność

$$B_1 \parallel [S] (B_2 \parallel [R] B_3) \sim (B_1 \parallel [S] B_2) \parallel [R] B_3$$

zachodzi, podobnie jak równość pierwsza, na mocy tych samych reguł tranzycji: jako pierwsze jest zawsze wybierane zdarzenie czasowe najwcześniejsze spośród zdarzeń oferowanych przez składowe B_1, B_2, B_3 , a po jego zajściu osiągnięte nowe wyrażenia oferują te same zbiory kolejnych zdarzeń czasowych.

Trzecia równość

$$\text{exit } [[S]] \text{ stop} \sim \text{stop}$$

jest oczywista.

Dla języka RT-LOTOS nie zachodzą natomiast inne równoważności, na przykład dla akcji wewnętrznych, takie jak:

$$g[\Delta_g] ; i[\Delta_i] ; B = g[\Delta_g] ; B$$

$$B [] i[\Delta_i] ; B = i[\Delta_i] ; B$$

$$g[\Delta_g] ; (B_1 [] i[\Delta_i] ; B_2) [] g[\Delta_g] ; B_2 \approx^c g[\Delta_g] ; (B_1 [] i[\Delta_i] ; B_2)$$

gdzie symbol $=$ oznacza dowolną z relacji $\sim, \approx^c, \approx$. Podobnie dla akcji **exit**

$$\text{exit } >> B \approx^c i ; B$$

Dla języka RT-LOTOS nie zachodzi, niestety, także twierdzenie o ekspansji, to znaczy złożenie wyrażeń w sekwencyjnej postaci normalnej nie zawsze daje się sprowadzić do wyrażenia w sekwencyjnej postaci normalnej. Warto rozpatrzeć ilustrujący to przykład.

Niech będą dwa wyrażenia behawioralne w normalnej postaci sekwencyjnej:

$$P \equiv a[\Delta_a] ; P_1 [] b[\Delta_b] ; P_2$$

$$Q \equiv c[\Delta_c] ; P_3 [] b[\Delta_b] ; P_4$$

Dla wyrażenia postaci $P [] b[] Q$ twierdzenie o ekspansji, stanowiące odpowiednik twierdzenia o ekspansji dla języka LOTOS, miałoby w tym przypadku postać:

$$P [] b[] Q \sim a[\Delta_a] ; (P_1 [] b[] Q) [] c[\Delta_c] ; (P_3 [] b[] P) [] b[\Delta_b] ; (P_2 [] b[] P_4)$$

Tymczasem rozpatrzmy przejścia do nowego wyrażenia; możliwe są następujące sytuacje:

1. Jako pierwsze zachodzi zdarzenie czasowe a , które nie jest związane z bramką synchronizującą P z Q .
2. Jako pierwsze zachodzi zdarzenie czasowe b na bramce synchronizującej P z Q .
3. Jako pierwsze zachodzi zdarzenie czasowe c , które nie jest związane z bramką synchronizującą P z Q .

W przypadku pierwszym, po rozpoczęciu akcji czasowej $a[\Delta_a]$, następnymi akcjami są nie tylko zakończenie akcji a , ale także – jeszcze przed zakończeniem akcji a – rozpoczęcie akcji c . Przedstawiona postać twierdzenia o ekspansji tej ostatniej możliwości nie przewiduje.

9.7. Język RT-LOTOS

Język RT-B-LOTOS został wprowadzony przede wszystkim w celu zwartego przedstawienia prawdziwie równoległej semantyki, bez uwzględnienia typów danych, które nie mają zasadniczego wpływu na definicję semantyki. Przyjmując definicję języka RT-B-LOTOS, można łatwo – w oczywisty sposób – rozszerzyć ją po dołączeniu typów danych do języka. W zastosowaniach praktycznych wymagamy zwykle języka dysponującego danymi, dlatego na podstawie definicji (9.8) dokonamy oczywistego rozszerzenia składni czasowych wyrażeń behawioralnych. Zakładamy przy tym, że definicja i semantyka typów danych są zgodne z odpowiednimi definicjami podanymi w rozdziale 4.

Zbiór wyrażeń behawioralnych, oznaczany $RT-TBeh$, jest zdefiniowany, za pomocą notacji BNF, w sposób następujący:

$$\begin{aligned}
 B ::= & \text{stop} \mid \text{exit} \mid g \alpha_1 \dots \alpha_n [\Delta]; B_1 \mid i[\Delta]; B_1 \mid \text{timeout}(\delta); B_1 \mid B_1 >> B_2 \mid \\
 & B_1 [] B_2 \mid \text{hide } R \text{ in } B_1 \mid B_1 [> B_2 \mid B_1 |[R]| B_2 \mid P[R] \mid (B_1) \mid [c] \rightarrow B_1
 \end{aligned} \tag{9.17}$$

gdzie: B, B_1, B_2 są metazmiennymi, $\alpha_1 \dots \alpha_n$ ($n \geq 0$) są elementami komunikacyjnymi zdefiniowanymi tak jak dla akcji strukturalnych (5.2), c jest predykatem – wyrażeniem logicznym, czyli termem rodzaju *Bool*.

Postać specyfikacji w języku RT-LOTOS jest zgodna ze składnią języka LOTOS i nie wymaga oddzielnej definicji.

Nieprzedstawianą tutaj formalnie semantykę specyfikacji można łatwo wyprowadzić na podstawie definicji języka LOTOS, przedstawionej w rozdziale 5., i definicji języka RT-B-LOTOS, przedstawionej w bieżącym rozdziale.

W przedstawionych definicjach abstrahujemy od sposobu wyboru czasu wykonania akcji. W zastosowaniach, jak wspominaliśmy, przyjmuje się różne mechanizmy wyboru. Ze względu na potrzeby dalej rozważanego przykładu specyfikacji założymy tu sposób wynikający z podziału akcji komunikacyjnych na dwie kategorie: akcje czynne i akcje bierne [Bernardo, Gorrieri 1998]. Akcje czynne to akcje, z którymi jest związany zbiór dopuszczalnych czasów wykonania (tak samo jak dotychczas rozważane akcje czasowe). Akcje bierne nie mają natomiast określonego z góry zbioru dopuszczalnych czasów wykonania. Czas wykonania akcji biernej jest określany przez czas akcji czynnej, z którą akcja bierna synchronizuje się podczas komunikacji.

W celu odróżnienia akcji czynnych od biernych, akcja bierna będzie oznaczana

$$g \alpha_1 \dots \alpha_n [*]$$

gdzie: $*$ jest dodatkowym wyróżnionym symbolem, a akcja czynna jest oznaczana jak dotychczas

$$g \alpha_1 \dots \alpha_n [\Delta]$$

Para synchronizujących się akcji, na przykład

$$g ! 10\{2, 3\} \qquad g ? x : \text{int}[*]$$

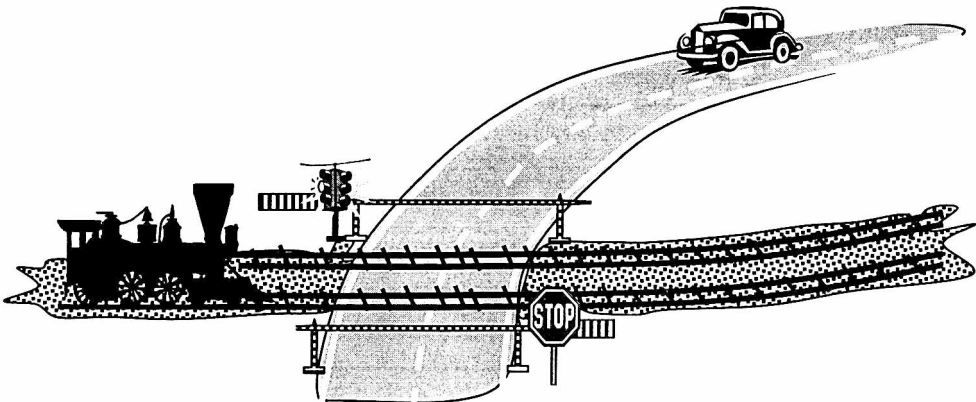
będzie się wykonywać 2 albo 3 jednostki czasu.

Podział na akcje czynne i bierne ma dobre odniesienie do wielu sytuacji praktycznych, kiedy komunikacja przebiega pomiędzy dwoma procesami, z których jeden jest usługodawcą, a drugi usługobiorcą. Czas komunikacji w takich przypadkach wyznacza zwykle usługodawca, a usługobiorca czas ten akceptuje. W przypadku wielu synchronizujących się akcji co najwyżej jedna może być akcją czynną.

9.8. Przykład specyfikacji w RT-LOTOSie

Rozważanym problemem jest sporządzenie specyfikacji systemu czasu rzeczywistego na podstawie następującego opisu tekstowego [Heitmeyer, Mandrioli 1996]:

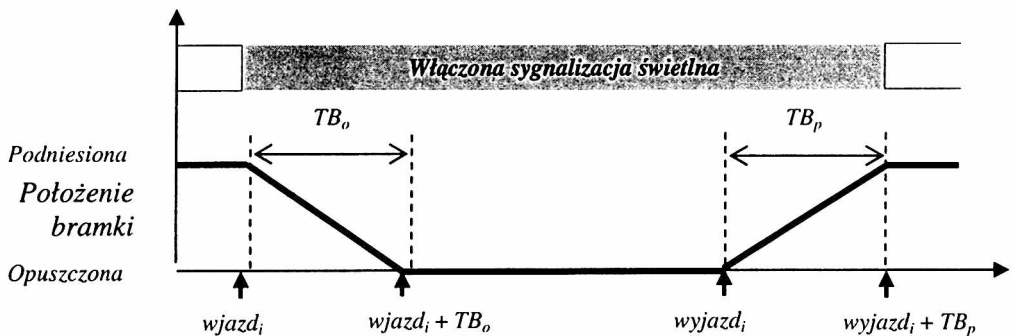
Zadanie polega na wyspecyfikowaniu funkcji systemu komputerowego sterującego ruchem pojazdów na drodze przez przejazd z n torami kolejowymi ($n > 0$). Na strzeżonym przejeździe kolejowym są rogatki, które – opuszczone lub podniesione – pozwalają na bezpieczny przejazd. Decyzję o opuszczeniu i podniesieniu bramek ma podejmować system sterujący przejazdem, na podstawie informacji otrzymywanych z czujników rozmieszczonych na torach w rejonie skrzyżowania drogi z torami. Zakłada się, że przez dany tor pociągi mogą przejeżdżać w jednym kierunku. Przy wjeździe na tor k ($k = 1, \dots, n$), w rejonie skrzyżowania, znajduje się czujnik rejestrujący wjazd pociągu (sensor typu *Wjazd*), a przy wyjeździe – czujnik rejestrujący wyjazd (sensor typu *Wyjazd*).



Rys. 9.1. Skrzyżowanie drogi z przejazdem kolejowym

System sterujący rozpoczyna pracę, gdy w rejonie skrzyżowania nie ma pociągów. Następnie oczekuje na sygnał o pojawieniu się pociągu. Po otrzymaniu sygnału z sensora typu *Wjazd*, na torze k , system włącza sygnalizację świetlno-dźwiękową oraz opuszcza bramki. Po odbiorze sygnału z sensora typu *Wjazd*, system komputerowy oczekuje na sygnał z sensora typu *Wyjazd*, dla tego samego toru k . Jeżeli to nastąpi, i w tym czasie nie pojawi się nowy sygnał wjazdu, to bramka się podnosi i wyłącza sygnalizację świetlno-dźwiękową. W przeciwnym razie, to znaczy gdy przed wyjazdem pociągu na torze k wjedzie inny pociąg na inny tor, bramka pozostaje opuszczona aż do momentu, gdy pojawią się sygnały o wyjeździe wszystkich pociągów z rejonu skrzyżowania.

Niech ($wjazd_i$, $wyjazd_i$) oznacza i -ty przedział czasu ($i = 1, 2, \dots$), w którym przejazd jest zajęty, tzn. $wjazd_i$ jest momentem, w którym nastąpił wjazd pierwszego pociągu w rejon skrzyżowania, gdy uprzednio było ono wolne, $wyjazd_i$ zaś jest momentem, w którym ostatni pociąg opuścił rejon skrzyżowania, gdy uprzednio było ono zajęte (rys. 9.2).



Rys. 9.2. Diagram czasowy włączania sygnalizacji i położenia bramek na przejeździe

Opuszczanie i podnoszenie rogatek może następować z pewnym opóźnieniem w stosunku do momentów, w których odnotowuje się wjazd i wyjazd pociągów. Niech TB_o oznacza okres, po którym powinno nastąpić opuszczenie bramek, od momentu zgłoszenia pierwszego sygnału z sensora typu *Wjazd*, w momencie $wjazd_i$, oraz niech TB_p oznacza okres, po którym powinno nastąpić podniesienie bramek, od momentu zgłoszenia ostatniego sygnału z sensora typu *Wyjazd*, w momencie $wyjazd_i$. Od systemu sterowania wymaga się własności bezpieczeństwa i użyteczności.

Własność **bezpieczeństwa** oznacza, że:

- dla każdej chwili z przedziału $[wjazd_i + TB_o, wyjazd_i]$ bramki muszą być opuszczone.

Własność *użytkowalności* oznacza, że:

- dla każdej chwili spoza przedziału ($wjazd_i$, $wjazd_i + TB_p$) bramki muszą być podniesione.

Ustalmy założenia dotyczące komunikacji specyfikowanego systemu z jego otoczeniem.

Zapora jest urządzeniem, do którego system sterowania, przez bramki o nazwach *podnieś*, *opuść*, będzie kierować polecenia opuszczenia i podnoszenia bramki.

Specyfikacja zachowania zapory jest następująca:

```
process Zapora [opuść, podnieś] : noexit :=
  opuść[ $\{t_{opuszczanie}\}$ ];
  podnieś[ $\{t_{podnoszenie}\}$ ];
  Zapora[opuść, podnieś]
endproc
```

Wartości $t_{opuszczanie}$ oraz $t_{podnoszenie}$ mogą być dowolne pod warunkiem, że są mniejsze od zadanych wartości TB_o i TB_p . Z przedstawionej specyfikacji wynika, że nie można przerywać operacji podnoszenia lub opuszczania zapory – operacja raz rozpoczęta musi się zakończyć.

Sygnalizacja jest urządzeniem, do którego system, przez bramki *włącz*, *wyłącz*, będzie kierować polecenia włączenia i gaszenia świateł sygnalizacyjnych. Jej specyfikacja jest podobna do specyfikacji zapory i ma postać:

```
process Sygnalizacja [włącz, wyłącz] : noexit :=
  włącz[ $\{t_{włączanie}\}$ ];
  wyłącz[ $\{t_{wyłączanie}\}$ ];
  Sygnalizacja[włącz, wyłącz]
endproc
```

Wartości $t_{włączanie}$ oraz $t_{wyłączanie}$ są oczywiście mniejsze od czasów opuszczania i podnoszenia zapory, a tym samym od zadanych wartości TB_o i TB_p .

Łącznie, co wynika z dalej przedstawionej postaci specyfikacji, wymaga się, aby:

$$t_{włączanie} + t_{opuszczanie} \leq TB_o \quad (9.18)$$

$$t_{wyłączanie} + t_{podnoszenie} \leq TB_p. \quad (9.19)$$

Nawet w tym prostym przykładzie uwidacznia się niejednorodność stylu specyfikowania. Zapora i sygnalizacja są zasobami, których obecność w specyfikacji jest odzwierciedleniem stylu specyfikacji ukierunkowanego na zasoby. Struktura natomiast

treści specyfikacji systemu sterowania zaporą odzwierciedla styl ukierunkowany na ograniczenia (rozdz. 7.) i ma postać podaną poniżej.

Stanowiące treść specyfikacji wyrażenie behawioralne jest złożeniem równoległym czterech procesów: *Bezpieczeństwo*, *Użyteczność*, *Sygnalizacja* i *Zapora*:

```

specification WymaganiaPodstawowe[wjazd, wyjazd] : noexit :=
behaviour
hide włacz, wyłącz, opuść, podnieś in
    Sygnalizacja [włacz, wyłącz]
    |[włacz, wyłącz]|
    ( Bezpieczeństwo [wjazd, wyjazd, opuść, włacz](0)
      |[wjazd, wyjazd]|
      Użyteczność [wjazd, wyjazd, podnieś, wyłącz](0)
    )
    |[opuść, podnieś]|
    Zapora [opuść, podnieś]
where
    process Zapora [opuść, podnieś] : noexit :=
        .....
    endproc
    process Sygnalizacja [włacz, wyłącz] : noexit :=
        .....
    endproc
    process Bezpieczeństwo[wjazd, wyjazd, opuść, włacz](n : nat) : noexit :=
        .....
    endproc
    process Użyteczność[wjazd, wyjazd, podnieś, wyłącz](n : nat) : noexit :=
        .....
    endproc
endspec

```

Składowe procesy specyfikacji są zdefiniowane poniżej.

Proces *Bezpieczeństwo* został zdefiniowany tak, aby odzwierciedlać warunek bezpieczeństwa:

```

process Bezpieczeństwo[wjazd, wyjazd, opuść, włacz](n : nat) : noexit :=
    [n = 0] ->
        wjazd[*];
        włacz[*];

```

```

opuść[*];
-- Ze względu na semantykę języka i założenie (9.18), akcja kończy
-- się nie później niż po upływie odcinka czasu  $TB_o$  od chwili
-- zakończenia pierwszej akcji wjazd[*] w sytuacji, gdy wcześniej
-- przejazd był pusty.
-- Oznacza to spełnienie własności bezpieczeństwa.

```

Bezpieczeństwo[*wjazd*, *wyjazd*, *opuść*, *włłącz*]($n + 1$)

[] [$n \geq 1$] ->

wjazd[*];

Bezpieczeństwo[*wjazd*, *wyjazd*, *opuść*, *włłącz*]($n + 1$)

[] [$n \geq 1$] ->

wyjazd[*];

Bezpieczeństwo[*wjazd*, *wyjazd*, *opuść*, *włłącz*]($n - 1$)

endproc

Proces *Użyteczność* odzwierciedla warunek użyteczności:

process *Użyteczność*[*wjazd*, *wyjazd*, *podnieś*, *wyłłącz*]($n : nat$) : **noexit** :=

wjazd[*];

Użyteczność[*wjazd*, *wyjazd*, *podnieś*, *wyłłącz*]($n + 1$)

[] [$n > 1$] ->

wyjazd[*];

Użyteczność [*wjazd*, *wyjazd*, *podnieś*, *wyłłącz*]($n - 1$)

[] [$n = 1$] ->

wyjazd[*];

podnieś[*];

wyłłącz[*];

```

-- Ze względu na semantykę języka i założenie (9.19), akcja kończy
-- się nie później niż po upływie odcinka czasu  $TB_p$  od chwili
-- zakończenia ostatniej akcji wyjazd[*] w sytuacji, gdy wcześniej
-- przejazd był zajęty.
-- Oznacza to spełnienie własności użyteczności.

```

Użyteczność[*wjazd*, *wyjazd*, *podnieś*, *wyłłącz*](0)

endproc

Warto zauważyć, że w komunikacji na bramkach *wjazd* i *wyjazd* uczestniczą dwa procesy składowe specyfikacji: *Bezpieczeństwo* i *Użyteczność* oraz otoczenie specyfikacji. Czas wykonywania akcji z udziałem tych bramek jest określony przez otoczenie specyfikacji. W komunikacji na pozostałych bramkach wewnętrznych uczestniczą tylko pary procesów.

Rozpatrywany przykład jest nie tylko ilustracją zastosowania języka RT-LOTOS, ale jest również ciekawy z tego względu, że dotyczy systemów związanych z bezpieczeństwem. Przedstawiona specyfikacja *WymaganiaPodstawowe* przedstawia funkcjonowanie systemu po jego uruchomieniu w warunkach normalnych.

Każdy system zwykle wymaga operatora, który system uruchamia i zatrzymuje, przeprowadza testowanie jego działania, wspomaga system w niektórych sytuacjach awaryjnych. Sytuacje awaryjne mogą powstać na skutek różnych okoliczności, gdy zachowanie się elementów składowych systemu, to jest zapory i sygnalizacji, a także zachowanie się pociągów przejeżdżających przez rejon skrzyżowania, mogą odbiegać od założonych [Huzar 2001]. Na przykład akcje opuszczania lub podnoszenia zapory mogą się nie zakończyć w założonym okresie, pociąg może się zatrzymać i cofnąć w rejonie skrzyżowania itp. Rozpatrzenie różnych możliwych okoliczności i określenie odpowiedniego postępowania wymaga oddzielnej analizy związanej z bezpieczeństwem [Magott 2005]. Analiza ma zidentyfikować potencjalne zagrożenia, a następnie należy ustalić sposoby postępowania zapobiegające ewentualnym skutkom zagrożeń. Wynikiem jest zwykle znaczna rozbudowa specyfikacji, w której uwzględnia się dodatkowe zachowanie.

Rozważmy na przykład tylko jeden przypadek, gdy opuszczanie zapory nie zakończy się przed upływem zadanego odcinka czasu t_{awaria} i przy założeniu, że światła sygnalizacyjne pracują poprawnie. Aby ostrzec pojazdy na drodze przed potencjalnym niebezpieczeństwem, można – zamiast stałych świateł – włączyć światła pulsujące. Rezultatem będzie rozbudowa specyfikacji, a głównie procesu *Bezpieczeństwo*. Nowy proces *Bezp* ma definicję

```

process Bezp[wjazd, wyjazd, opuść, włącz, wyłącz, koniec](n : nat) : noexit :=
  [n = 0] ->
    wjazd[*];
    włącz[*];
    ( opuść[*];
      exit
    [> timeout( $t_{awaria}$ );
      ŚwiatłaPulsujące[włącz, wyłącz, koniec]
    )
  >> Bezp[wjazd, wyjazd, opuść, włącz, wyłącz, koniec](n + 1)
[] [n ≥ 1] ->
  wjazd[*];
  Bezp[wjazd, wyjazd, opuść, włącz, wyłącz, koniec](n + 1)
[] [n ≥ 1] ->
  wyjazd[*];

```

Bezp[wjazd, wjazd, opuść, włłącz, wyłącz, koniec](n - 1)

where

process *ŚwiatłaPulsujące*[włłącz, wyłącz, koniec] : **exit** :=

wyłłącz[*];

i[{*t_{wyłłącz}*}];

-- okres wyłączenia świateł

włłącz[*];

i[{*t_{włłącz}*}];

-- okres włączenia świateł

ŚwiatłaPulsujące[włłącz, wyłącz, koniec]

[] *koniec*[*]; **exit**

endproc

Złożenie deaktywujące

opuść[*]; **exit** [**>** **timeout**(*t_{awaria}*)]; *ŚwiatłaPulsujące*[włłącz, wyłącz, koniec]

spowoduje, że jeśli akcja czasowa *opuść* nie zakończy się przed upływem odcinka czasu *t_{awaria}*, nastąpi jej przerwanie i zostanie zainicjowany proces postępowania awaryjnego *ŚwiatłaPulsujące*. Jednocześnie należy zauważyć, że sama akcja czasowa *opuść*, jeśli się już zaczęła, nie zostanie przerwana.

W definicji procesu pojawiła się nowa bramka *koniec*, która ma służyć zakończeniu postępowania awaryjnego. Zakłada się tu, że komunikacja na tej bramce, po usunięciu awarii zapory, zachodzi na skutek zewnętrznej interwencji operatora, którego istnienie jest konieczne, a dotychczas nie było uwzględniane.

9.9. Uwagi końcowe

W tym rozdziale przedstawiono i porównano dwa podejścia do czasowego rozszerzenia języka LOTOS. Oba podejścia są równoważne w sensie ekspresywności, różnią się natomiast własnościami.

Podejście pierwsze, zakładające zachowanie akcji natychmiastowych, było rozwijane między innymi w pracach: [Leduc, Leonard 1992], [Quemada, Fernandez 1987], [Quemada, Azcorra, Frutos 1990], [Quemada, Frutos, Miguel 1993]. Przedstawiany w tym rozdziale język TE-B-LOTOS stał się podstawą definicji języka E-LOTOS [ISO/IEC FDIS 15437, 2001]. Dwoma zasadniczymi własnościami języka E-LOTOS są bowiem rozszerzenie czasowe i nowy sposób definiowania typów danych.

Drugie podejście, oparte na wprowadzeniu akcji czasowych, było rozwijane w pracach: [Huzar, Magott 1995a, 1995b], [Huzar, Magott 1996], [Huzar, Magott 1997a, 1997b, 1997c, 1997d]. Opisywany w bieżącym rozdziale język RT-B-LOTOS jest

uogólnieniem tych prac. Polega to na przypisaniu akcji czasowych zbioru dopuszczalnych czasów wykonania i abstrahowaniu mechanizmu uzgadniania czasu trwania wspólnie realizowanych akcji.

Wadą drugiego z podejść, w stosunku do podejścia pierwszego, jest niezachowanie wszystkich własności języka LOTOS. Zaletą natomiast jest to, że wersja z akcjami czasowymi – bardziej naturalna – bezpośrednio odpowiada mechanizmom komunikacji, jakie spotykamy w językach programowania czasu rzeczywistego, na przykład w języku Ada [Huzar, Fryźlewicz, Dubielewicz, Hnatkowska, Waniczek 1998]. Warto zauważyć, że modelowanie akcji czasowych języka RT-B-LOTOS wymaga użycia w języku TE-B-LOTOS nie tylko dwóch akcji natychmiastowych, ale także wyrażenia powiązania przyczynowo-skutkowego obu tych akcji.

Wydaje się także, że podejście z akcjami czasowymi jest dobrym kandydatem do rozszerzenia języka do wersji probabilistycznej.

Przedstawiono dwie wersje języka opartego na założeniu akcji czasowych – wersję bazową RT-B-LOTOS i wersję pełną RT-LOTOS. W wersji pełnej wprowadzono mechanizm ustalania czasu wspólnie realizowanych akcji komunikujących się procesów, wykorzystujący akcje czynne i bierne [D’Argenio, Hermanns, Katoen 1998], [Bernardo, Gorrieri 1998]. Możliwości modelowania języka RT-LOTOS pokazano na przykładzie systemu sterowania przejazdem kolejowym, zaczerpniętego z [Heitmeyer, Mandrioli 1996] oraz [Huzar 2001]. Przykład ten jest dodatkowo ilustracją stylów specyfikacji omówionych w rozdziale 7.

10. Rozszerzenia wydajnościowe LOTOSa

10.1. Wstęp

Przedstawione w poprzednim rozdziale czasowe rozszerzenia języka LOTOS charakteryzuje duży stopień niedeterminizmu. W przypadku języka TE-LOTOS odnosi się to do przedziału czasów, w których może zachodzić akcja, a w przypadku języka RT-LOTOS – do czasu trwania akcji czasowej. W obu językach wspólnym źródłem niedeterminizmu jest otoczenie specyfikowanego systemu, które dokonuje wyboru jednej z oferowanych akcji – jest to niedeterminizm zewnętrzny, oraz niedeterministyczny wybór jednej spośród oferowanych akcji wewnętrznych – jest to niedeterminizm wewnętrzny.

Często, przed przystąpieniem do implementacji na podstawie specyfikacji, specyfikacja jest przedmiotem odrębnych badań. Jednym z rodzajów takich badań jest badanie własności wydajnościowych. Badania wydajnościowe są badaniami ilościowymi, które rzadko można przeprowadzać na modelach niedeterministycznych, można je natomiast skutecznie prowadzić na modelach deterministycznych lub probabilistycznych. W tym rozdziale przedstawiono rozszerzenie języka LOTOS, które polega na wprowadzeniu probabilistycznej charakterystyki czasów trwania akcji oraz na eliminacji z języka mechanizmów niedeterministycznych i zastąpieniu ich priorytetami i mechanizmami probabilistycznymi. Priorytety mają służyć, w pierwszej kolejności, do wyznaczania jednej akcji, spośród akcji, w danym stanie, kandydujących do wykonania. Gdy priorytety nie rozstrzygają o wyborze akcji, wyboru dokonuje się na podstawie mechanizmu probabilistycznego.

Przyjmuje się, że obowiązują następujące zasady przydzielania priorytetów: Zakładamy, że mamy dwa rodzaje akcji: akcje czasowe i natychmiastowe. (Podział taki wynika z pragmatyki: akcje, których czas trwania jest znikomym krótki względem pozostałych, traktuje się jako natychmiastowe, akcje zaś, których czas trwania – w kontekście konkretnej sytuacji – jest zauważalny, traktuje się jako akcje czasowe.) Akcjom natychmiastowym przypisuje się jako priorytety dodatnie liczby naturalne. Akcjom czasowym przypisuje się natomiast jednakowy, najniższy priorytet zero. Wynika to z przyjęcia zasady wyścigów podczas wykonywania akcji czasowych: najpierw wykonują się akcje o najwcześniejszych chwilach rozpoczęcia wykonywania.

Charakterystyki probabilistyczne można wprowadzać różnie, tu chodzi o taki sposób, aby na podstawie specyfikacji w LOTOSie można w jednoznaczny sposób otrzymać pewien skończony łańcuch Markowa. Łańcuchy Markowa są wygodnym modelem probabilistycznym, dla którego są znane metody obliczania interesujących charakterystyk [Czachórski 1999], [Bronsztajn, Siemiendiajew, Musiol, Mühling 2004]. W tym celu przyjmuje się, że czasy wykonywania akcji czasowych charakteryzuje rozkład wykładniczy, a akcje natychmiastowe są wykonywane w zerowym czasie.

Dwoma źródłami niedeterminizmu są wybór i złożenie równoległe wyrażeń behawioralnych. Wyrażeniom tym przypisuje się pewne, dalej omówione, parametry probabilistyczne.

Przedstawiane dalej rozszerzenie języka LOTOS jest nazywane MB-LOTOS. Wpływa to z faktu, że rozszerzenie jest rozszerzeniem markowowskim, a ponadto – podobnie jak w poprzednim rozdziale – prezentacja ogranicza się tylko do części bazowej, czyli bez uwzględniania komunikacji danych. Dodatkowo prezentacja ma charakter techniczny, co oznacza, że używane dalej oznaczenia akcji są rozbudowane tak, aby wygodnie wyrażać pewne własności, co nie jest wymagane w przypadku posługiwania się językiem do specyfikowania systemów.

10.2. Składnia i nieformalna semantyka języka MB-LOTOS

Wprowadźmy oznaczenia:

G	zbiór nazw bramek obserwowalnych,
$L = G \cup \{\text{exit}\}$	zbiór nazw akcji obserwowalnych rozszerzony o akcję exit , wykonywaną na bramce nieobserwowalnej <i>exit</i> ,
$A = L \cup \{i\}$	zbiór nazw wszystkich akcji obserwowalnych i nieobserwowalnych,
Nat	zbiór liczb naturalnych,
$AP = Nat \cup \{\infty\}$	zbiór priorytetów; symbol ∞ oznacza najwyższy priorytet,
$PR = [0, 1]$	zbiór prawdopodobieństw,
$AR = R_+ \setminus \{0\} \cup \{\infty\}$	zbiór wartości parametrów intensywności rozkładów wykładniczych, gdzie R_+ jest zbiorem nieujemnych liczb rzeczywistych,
$Proc$	zbiór nazw procesów,
$Time = R_+$	dziedzina chwil czasowych.

Jak wspomniano, akcje będą przedstawiane w rozszerzonej postaci

$$(g, \alpha, \lambda, \pi) \quad (10.1)$$

gdzie:

$g \in A$	oznacza nazwę akcji,
$\alpha \in Nat$	oznacza indeks jednoznacznie identyfikujący każde wystąpienie akcji,
$\lambda \in AR$	oznacza parametr czasowy akcji; jeżeli $\lambda \neq \infty$, to λ oznacza akcję czasową o rozkładzie wykładniczym czasu wykonania, w przypadku przeciwnym, gdy $\lambda = \infty$, oznacza akcję natychmiastową,
$\pi \in AP$	oznacza priorytet akcji; jeżeli $\pi = 0$, oznacza to akcję czasową, $\pi > 0$ oznacza akcję natychmiastową.

Indeksy wystąpień akcji wprowadza się tylko w celu prezentacji formalnej semantyki. Ze względu na zastosowanie języka są one zbędne. Szczegółowe powody ich wpro-

wadzenia są dwa: Po pierwsze – idzie o jednoznaczne rozróżnienie wystąpień tej samej akcji w tym samym stanie. Drugi powód, ze względów technicznych, jest nawet ważniejszy, gdyż idzie o wygodną i jednoznaczną identyfikację akcji złożonych, to jest takich, które są wynikiem synchronizacji wielu akcji na tej samej bramce. Dalej będziemy zakładać, że jest stosowany pewien mechanizm jednoznacznego indeksowania akcji. Szczegółowo mechanizm taki jest przedstawiony w pracach [Huzar, Magott 1997b] i [Huzar, Magott 1997d], jest także używany w pracach innych autorów, np. [D’Argenio, Hermanns, Katoen 1998].

Wystąpienie akcji jest jednoznacznie identyfikowane przez swój indeks, stąd wynikają oznaczenia: jeżeli akcja ma postać $(g, \alpha, \lambda, \pi)$, to z definicji

$$\text{name}(\alpha) = g, \text{rate}(\alpha) = \lambda, \text{prior}(\alpha) = \pi \quad (10.2)$$

Zbiór n akcji składowych, synchronizujących się na bramce $g \in L$, nazwany dalej akcją zagregowaną, będzie reprezentowany przez ciąg ich indeksów, oznaczany przez

$$\Phi = \alpha_1 \dots \alpha_n \quad (10.3)$$

gdzie $\alpha_i \neq \alpha_j$, dla $i \neq j$ ($i, j = 1, \dots, n$). Oczywiście

$$\text{name}(\alpha_1) = \dots = \text{name}(\alpha_n) = g \quad (10.4)$$

oraz z definicji:

$$\text{name}(\Phi) = \text{name}(\alpha_i) \text{ dla } i = 1, \dots, n \quad (10.5)$$

Zakładamy, że priorytet akcji zagregowanej reprezentowanej przez (10.3) jest równy

$$\text{prior}(\Phi) = \text{prior}(\alpha_1) = \dots = \text{prior}(\alpha_n) \quad (10.6)$$

co wynika z założenia, że priorytety akcji składowych wykonywanych na tych samych bramkach mają być równe. Nie oznacza to, że bramki wyznaczają priorytety akcji, ale że jeśli akcje się synchronizują na danej bramce, to mają takie same priorytety, co nie wyklucza, że na danej bramce mogą się synchronizować różne zbiory akcji o różnych priorytetach.

Założenie dotyczące priorytetów przyjęto tylko dla uproszczenia rozważań. Możliwe są oczywiście różne sposoby obliczania priorytetu akcji zagregowanej, na przykład priorytet akcji zagregowanej mógłby być sumą priorytetów akcji składowych.

Składnia zbioru wyrażeń behawioralnych *MBeh* języka MB-LOTOS przedstawia się następująco:

$$\begin{aligned} Q ::= & \text{stop} \mid (g, \alpha, \lambda, \pi); Q \mid (i, \alpha, \lambda, \pi); Q \mid (\text{exit}, \alpha, \infty, \infty) \mid Q_1 [p] Q_2 \mid \\ & Q_1 [R]_{s,r} Q_2 \mid \text{hide } R \text{ in } Q \mid Q_1 >> Q_2 \mid Q_1 [> Q_2] [p[R]] (Q) \end{aligned} \quad (10.7)$$

Symbole Q, Q_1, Q_2 są metazmiennymi, oznaczającymi wyrażenia behawioralne, $g \in G$ jest bramką, $R \in SeqG$ jest ciągiem niepowtarzających się bramek, $\lambda \in AR$ parametrem

intensywności, $\pi \in AP$ priorytetem oraz $p, s, r \in PR$ są prawdopodobieństwami. Kolejność wprowadzenia operatorów odpowiada malejącej kolejności ich priorytetów.

Dalej, podobnie jak we wcześniejszych rozdziałach, zakładamy, że rozważamy tylko wyrażenia dozorowane, regularnie zbudowane.

Znaczenia poszczególnych wyrażeń są następujące:

Wyrażenia **stop** oraz (**exit**, α , ∞ , ∞) mają takie same znaczenia, jak we wszystkich poprzednio rozważanych wersjach języka. O akcji **exit** zakładamy dodatkowo, że jest akcją natychmiastową o najwyższym priorytecie. Podobne, jak poprzednio, znaczenie mają wyrażenia prefiksowania akcją obserwowalną (g , α , λ , π); Q oraz wewnętrzną (i , α , λ , π); Q .

Bardziej złożone jest znaczenie wyrażenia wyboru $Q_1 [p] Q_2$. Wyrażenie oferuje do wykonania te akcje, które są przygotowane w wyrażeniach Q_1 oraz Q_2 . Wybór jednej spośród nich jest dokonywany dwufazowo. W pierwszej fazie jest wyznaczany spośród tych akcji zbiór akcji o najwyższym priorytecie, a w drugiej fazie następuje wybór jednej z nich. Jeżeli akcjami o najwyższym priorytecie są akcje natychmiastowe i wśród nich jest co najmniej jedna akcja z wyrażenia Q_1 oraz co najmniej jedna akcja z wyrażenia Q_2 , to do wykonania, z prawdopodobieństwem $(1 - p)$, może być wybrana dokładnie jedna akcja należąca do Q_1 , albo – z prawdopodobieństwem p – dokładnie jedna akcja należąca do Q_2 . Jeżeli akcjami o najwyższym priorytecie są akcje czasowe, to wybór jednej z nich następuje zgodnie z zasadą wyścigów: wybierana jest ta, która rozpoczyna się najwcześniej. Po wyborze akcji do wykonania dalsze zachowanie wyrażenia $Q_1 [p] Q_2$ jest takie same jak w poprzednich wersjach języka.

Równie złożona jest semantyka wyrażenia złożenia równoległego $Q_1 ||R||_s Q_2$. Wybór akcji do wykonania także przebiega dwufazowo: najpierw wyznacza się zbiór akcji o najwyższym priorytecie, oferowanych przez wyrażenia Q_1 oraz Q_2 , a następnie wybiera się z nich jedną akcję. Wybrana ostatecznie akcja może być akcją należąca albo do wyrażenia Q_1 , albo Q_2 , albo też może być akcją złożoną z synchronizujących się akcji należących do Q_1 i Q_2 . Jeżeli wśród akcji o najwyższym priorytecie są akcje należące do wyrażenia Q_1 oraz do wyrażenia Q_2 , to – z prawdopodobieństwem s – są wybierane akcje synchronizujące się na wspólnych bramkach R oraz akcje **exit**, akcje natomiast, które nie synchronizują się na bramkach R , inne od **exit**, są wybierane z prawdopodobieństwem $(1 - s)$. W przypadku wyboru akcji niesynchronizujących się na bramkach R , spośród nich – z prawdopodobieństwem r – zostaje wybrana akcja należąca do wyrażenia Q_2 , a z prawdopodobieństwem $(1 - r)$ – akcja należąca do Q_1 .

Znaczenie pozostałych wyrażeń behawioralnych: przesłonięcia **hide** R **in** Q , złożenia aktywującego $Q_1 >> Q_2$, złożenia deaktywującego $Q_1 [> Q_2$ oraz wywołania procesu $p[R]$ są takie same jak w poprzednio omawianej wersji języka. Zakładamy przy tym, że akcje wewnętrzne, będące wynikiem przesłonięcia, mają takie same własności czasowe jak akcje przesłanianie.

Tak samo jak w poprzednich wersjach są określane deklaracje procesów, ich funkcjonalność oraz specyfikacja.

W celu przybliżenia mechanizmów probabilistycznego wyboru akcji rozpatrzmy dwa przykłady wyboru akcji natychmiastowych.

Przykład 10.1

Niech dane będzie wyrażenie behawioralne Q :

$$(g_1, \alpha_1, \infty, \pi_1); Q_1 \\ [p] ((g_2, \alpha_2, \infty, \pi_2); Q_2 [q] (g_3, \alpha_3, \infty, \pi_3); Q_3))$$

W skład wyrażenia wchodzi tylko akcja natychmiastowa. Jeżeli ich priorytety π_1, π_2, π_3 są różne, to wartości prawdopodobieństw p oraz q są nieistotne, a jako pierwsza – pod warunkiem, że otoczenie wyrażenia Q będzie gotowe – zostanie wybrana i wykonana akcja o najwyższym priorytecie. Jeżeli priorytety są równe, wybór akcji jest probabilistyczny. Z prawdopodobieństwem p następuje wybór prawej strony wyrażenia Q

$$((g_2, \alpha_2, \infty, \pi_2); Q_2 [q] (g_3, \alpha_3, \infty, \pi_3); Q_3)),$$

a z prawdopodobieństwem $(1 - p)$ – wybór lewej strony wyrażenia Q

$$(g_1, \alpha_1, \infty, \pi_1); Q_1$$

Pod warunkiem wyboru prawej strony akcja o indeksie α_2 będzie wybrana z prawdopodobieństwem $(1 - q)$, akcja zaś o indeksie α_3 z prawdopodobieństwem q . Prawdopodobieństwo zatem wyboru akcji o indeksie α_2 wynosi $p(1 - q)$, a akcji o indeksie α_3 wynosi pq . Prawdopodobieństwo natomiast wyboru akcji o indeksie α_1 wynosi $(1 - p)$.

Przykład 10.2

Niech dane będzie wyrażenie behawioralne Q postaci:

$$((g_1, \alpha_1, \infty, \pi_1); Q_1 [p] (g_2, \alpha_2, \infty, \pi_2); Q_2) \\ |\{g_2\}|_{s,r} \\ ((g_3, \alpha_3, \infty, \pi_3); Q_3 [q] (g_2, \alpha_4, \infty, \pi_4); Q_4)$$

Założmy, że priorytety wszystkich akcji są takie same. Akcje natychmiastowe o indeksach α_2 i α_4 synchronizują się na bramce g_2 . Oznacza to, że

$$\text{prior}(\alpha_1) = \text{prior}(\alpha_2) = \text{prior}(\alpha_3) = \text{prior}(\alpha_4) = \text{prior}(\alpha_2 \alpha_4)$$

Pod warunkiem, że otoczenie wyrażenia Q nie stwarza własnych ograniczeń, możliwe są trzy wybory akcji: jednej pary synchronizujących się akcji o indeksach α_2 i α_4 oraz jednej z dwóch akcji o indeksach α_1 i α_3 , niesynchronizujących się na wspólnej bramce g_2 .

W podwyrażeniu

$$((g_1, \alpha_1, \infty, \pi_1); Q_1 [p] (g_2, \alpha_2, \infty, \pi_2); Q_2)$$

akcja o indeksie α_1 jest wybierana w tym wyrażeniu z prawdopodobieństwem $(1-p)$, a akcja o indeksie α_2 z prawdopodobieństwem p .

Podobnie, w podwyrażeniu

$$((g_3, \alpha_3, \infty, \pi_3); Q_3 [q] (g_2, \alpha_4, \infty, \pi_4); Q_4)$$

akcja o indeksie α_3 jest wybierana z prawdopodobieństwem $(1-q)$, akcja o indeksie α_2 z prawdopodobieństwem q .

Selekcja akcji w równoległym złożeniu obu podwyrażeń odbywa się dwufazowo. W pierwszej fazie z prawdopodobieństwem s są wybierane akcje α_2, α_4 synchronizujące się na bramce g_2 , a w drugiej fazie są wybierane akcje niesynchronizujące się na tej bramce: akcja α_1 z prawdopodobieństwem $(1-s)*(1-r)$, oraz akcja α_3 z prawdopodobieństwem $(1-s)*r$.

Oprócz opisanego mechanizmu probabilistycznego wyboru akcji, drugim elementem charakteryzującym język MB-LOTOS jest mechanizm wyznaczania czasu trwania synchronizujących się akcji. Pojedyncze akcje czasowe mają wykładniczy rozkład prawdopodobieństwa czasu realizacji. Synchronizujące się akcje mają również rozkład wykładniczy, ale parametr intensywności tego rozkładu jest wyznaczany w specyficzny sposób.

Tabela 10.1

$r_g((g', \alpha, \lambda, 0); Q) = \begin{cases} \lambda & \text{gdy } g = g' \\ 0 & \text{w przypadku przeciwnym} \end{cases}$	
$r_g(Q_1 [p] Q_2) = r_g(Q_1) + r_g(Q_2)$	
$r_g(Q_1 \mid R \mid_{s,r} Q_2) = \begin{cases} \min(r_g(Q_1), r_g(Q_2)) & \text{gdy } g \in \text{set}(R) \\ r_g(Q_1) + r_g(Q_2) & \text{gdy } g \in (G \cup \{i\}) \setminus \text{set}(R) \end{cases}$	
$r_g(Q_1 [> Q_2) = r_g(Q_1) + r_g(Q_2)$	
$r_g(Q_1 >> Q_2) = r_g(Q_1)$	
$r_g(\text{hide } R \text{ in } Q) = \begin{cases} r_g(Q) & \text{gdy } g \in G \setminus \text{set}(R) \\ 0 & \text{gdy } g \in \text{set}(R) \\ \sum_{g' \in \text{set}(R)} r_{g'}(Q) + r_i(Q) & \text{gdy } g = i \end{cases}$	
$r_g(P[g_1, \dots, g_m]) = r_g(Q[g_1 ::= h_1, \dots, g_m ::= h_m]) \quad \text{gdzie } Q \text{ jest treścią procesu } P.$	

W przypadku szczególnym, gdy rozważamy dwie synchronizujące się akcje na bramce $g \in G$, na przykład

$$(g, \alpha_1, \lambda_1, \pi) \text{ oraz } (g, \alpha_2, \lambda_2, \pi)$$

parametrem rozkładu prawdopodobieństwa, charakteryzującym wspólny czas ich wykonania, będzie wartość $\min(\lambda_1, \lambda_2)$.

Ogólnie wyznaczanie parametru intensywności akcji czasowych w wyrażeniu behawioralnym Q przedstawia się następująco: Dla akcji o nazwach $g \in G \cup \{i\}$ w wyrażeniu Q definiuje się parametr intensywności akcji czasowych $r_g(Q)$. Definicja parametru jest rekursywna (tab. 10.1).

Przedstawiony mechanizm oparto na pracy [Hilston 1996], w której opisuje się język PEPA, stanowiący czasowe uogólnienie języka CCS. Mechanizm ten umożliwia użycie rozważanych dalej własności kompozycyjności.

10.3. Semantyka formalna języka MB-LOTOS

Akcje kandydujące do wykonania

Semantyka języka jest wyrażana – podobnie jak wcześniejsze wersje języka – przez etykietowany system przejść. W odróżnieniu od języków przedstawianych w poprzednim rozdziale, tu semantyka jest przeplotowa, co jest możliwe dzięki przyjęciu markowskiego modelu czasu wykonywania akcji czasowych. Do zdefiniowania semantyki potrzebne są funkcje pomocnicze. Pierwsza z nich, funkcja o sygnaturze

$$F : MBeh \rightarrow 2^{Nat^*} \quad (10.8)$$

służy do wyznaczania zbioru indeksów akcji przygotowanych do wykonania w danej konfiguracji. Jest ona zdefiniowana rekursywnie w sposób pokazany w tabeli 10.2.

Druga funkcja pomocnicza, o sygnaturze

$$M : 2^{Nat} \rightarrow 2^{Nat} \quad (10.9)$$

służy do wyznaczania w danym zbiorze indeksów akcji, podzbioru indeksów akcji o najwyższym priorytecie. Jej definicja jest następująca: niech $\Theta \subseteq Nat$ będzie zbiorem indeksów akcji oraz niech $mprior(\Theta) = \max\{prior(\Phi) \mid \Phi \in \Theta\}$, wówczas

$$M(\Theta) = \{ \Phi \in \Theta \mid prior(\Phi) = mprior(\Theta) \}. \quad (10.10)$$

Jeżeli $\Theta = F(Q)$, to $M(F(Q)) \subseteq F(Q)$ jest podzbiorem tych akcji wyrażenia Q przygotowanych do wykonania, które mają najwyższy priorytet. Jest to zbiór akcji wyrażenia Q kandydujących do wykonania.

Zbiór akcji kandydujących może zawierać albo tylko akcje natychmiastowe, albo tylko akcje czasowe, nie może natomiast zawierać obu rodzajów akcji, gdyż akcje czasowe mają priorytet zerowy, akcje natychmiastowe zaś mają priorytet większy od zera. Wyra-

żenia behawioralne, dla których zbiór $M(F(Q))$ zawiera tylko indeksy akcji natychmiastowych, nazywa się wyrażeniem zanikającym, wyrażenia zaś, dla których zbiór $M(F(Q))$ zawiera tylko indeksy akcji czasowych, nazywa się wyrażeniem uchwytym.

Tabela 10.2

$$F(\text{stop}) = \emptyset$$

$$F((g, \alpha, \lambda, \pi); Q) = \{ \alpha \}$$

$$F(Q_1 [p] Q_2) = F(Q_1) \cup F(Q_2)$$

$$F(Q_1 |R|_{s,r} Q_2) = FA(Q_1, R) \cup FA(Q_2, R) \cup FS(Q_1, Q_2, R)$$

$$\text{gdzie: } FA(Q_i, R) = \{ \Phi \in F(Q_i) \mid \text{name}(\Phi) \notin R \cup \{\text{exit}\} \} \quad (i = 1, 2)$$

$$FS(Q_1, Q_2, R) = \{ \Phi \Psi \mid \Phi \in F(Q_1) \wedge \Psi \in F(Q_2) \wedge \text{prior}(\Phi) = \text{prior}(\Psi) \wedge \text{name}(\Phi) = \text{name}(\Psi) \in R \cup \{\text{exit}\} \}$$

$$F(\text{hide } R \text{ in } Q) = F(Q [\alpha_1 ::= \beta_1, \dots, \alpha_n ::= \beta_n])$$

$$\text{gdzie: } \{ \alpha_1, \dots, \alpha_n \} = \{ \alpha \mid \text{name}(\alpha) \in R \},$$

$\{ \beta_1, \dots, \beta_n \}$ jest zbiorem nowych indeksów innych od indeksów występujących w Q , a $Q[\alpha_1 ::= \beta_1, \dots, \alpha_n ::= \beta_n]$ jest tekstowym zastąpieniem indeksów $\alpha_1, \dots, \alpha_n$ przez indeksy $\beta_1, \dots, \beta_n$ w wyrażeniu Q . Nowe indeksy odróżniają akcje wykonywane na bramkach R wewnątrz wyrażenia **hide** R **in** Q od tych, które są widoczne na zewnątrz tego wyrażenia jako jego akcje wewnętrzne. Nowe akcje o indeksach β_i ($i = 1, \dots, n$) mają następujące własności:

$$\text{name}(\beta_i) = \mathbf{i}, \text{rate}(\beta_i) = \text{rate}(\alpha_i), \text{and } \text{prior}(\beta_i) = \text{prior}(\alpha_i)$$

$$F(Q_1 >> Q_2) = F(Q_1)$$

$$F(Q_1 [> Q_2) = F(Q_1) \cup F(Q_2)$$

$$F(P[h_1, \dots, h_m]) = F(Q[g_1 ::= h_1, \dots, g_m ::= h_m])$$

$$\text{gdzie } Q \text{ jest treścią procesu } P[g_1, \dots, g_m].$$

Jeżeli zbiór akcji kandydujących jest jednoelementowy, to wyznacza on jednoznacznie akcję do wykonania, jeżeli natomiast jest większy, to wybór akcji jest probabilistyczny.

Etykietowany system przejść

Semantykę języka przedstawiono dwuetapowo. W pierwszym etapie zdefiniowano etykietowany system przejść, który dla danej konfiguracji wyznacza możliwe przejścia do nowej konfiguracji, inaczej: wyznacza zakres niedeterminizmu. W drugim etapie określono prawdopodobieństwa przejścia do nowych konfiguracji, przy czym

oddzielnie rozważano przejścia realizowane w wyniku zajścia akcji czasowych oraz natychmiastowych.

Dla specyfikacji $MSpec = \langle \{Q, \{D_1, \dots, D_n\}\rangle$ etykietowany system przejść jest zdefiniowany jako

$$MTS(MSpec) = \langle MBeh, MEvent, MTrans, Q \rangle \quad (10.11)$$

gdzie:

$MEvent = \{(g, \alpha, \lambda, \pi) \mid g \in A \wedge \alpha \in Nat \wedge \lambda \in AR \wedge \pi \in AP\}$ jest zbiorem zdarzeń,

$MTrans = \{ \xrightarrow{e} \subseteq MBeh \times MBeh \mid e \in MEvent \}$ jest zbiorem relacji przejść.

Zbiór relacji przejść jest definiowany rekursywnie w sposób następujący:

Proces pusty stop – brak aksjomatów

Proces zakończenia $Q \equiv \mathbf{exit}$

$$Q \xrightarrow{(exit, \alpha, \lambda, \pi)} \mathbf{stop} \quad (A\text{-exit})$$

Prefiksowanie akcją $Q \equiv (g, \alpha, \lambda, \pi); Q_1$

$$Q \xrightarrow{(g, \alpha, \lambda, \pi)} Q_1 \quad (A\text{-pre})$$

Wybór probabilistyczny $Q \equiv Q_1 [p] Q_2$

$$\frac{Q_i \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_i}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_i} \quad (R\text{-choice})$$

gdzie $\Phi \in M(F(Q))$ dla $i = 1, 2$.

Złożenie równoległe $Q \equiv Q_1 \parallel [R]_{s,r} Q_2$

$$\frac{Q_1 \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1 \parallel [R]_{s,r} Q_2} \quad (R\text{-par}_1)$$

$$\frac{Q_2 \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_2}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} [R]_{s,r} Q'_2} \quad (R\text{-par}_2)$$

gdzie $\Phi \in M(F(Q))$ oraz $g \notin \text{set}(R) \cup \{\mathbf{exit}\}$,

$$\frac{\frac{Q_1 \xrightarrow{(g, \Phi, \lambda_1, \pi)} Q'_1}{Q_2 \xrightarrow{(g, \Psi, \lambda_2, \pi)} Q'_2}}{Q \xrightarrow{(g, \Phi\Psi, \lambda_{12}, \pi)} Q'_1 \parallel [R]_{s,r} Q'_2} \quad (R\text{-par}_{12})$$

gdzie $\Phi\Psi \in M(F(Q))$ oraz $g \in \text{set}(R) \cup \{\mathbf{exit}\}$

gdzie

$$\lambda_{12} = \begin{cases} \frac{\lambda_1}{r_g(Q_1)} \frac{\lambda_2}{r_g(Q_2)} \min(r_g(Q_1), r_g(Q_2)) & \text{gd } \pi = 0 \\ \infty & \text{gd } \pi > 0 \end{cases}$$

Złożenie deaktywujące $Q \equiv Q_1 [> Q_2$

$$\frac{Q_1 \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1 [> Q_2]} \quad (R\text{-dis}_1)$$

dla $\Phi \in M(F(Q))$ oraz $g \neq \text{exit}$

$$\frac{Q_2 \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_2}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_2} \quad (R\text{-dis}_2)$$

dla $\Phi \in M(F(Q))$

$$\frac{Q_1 \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1} \quad (R\text{-dis}_3)$$

dla $\Phi \in M(F(Q))$ oraz $g = \text{exit}$

Złożenie aktywujące $Q \equiv Q_1 >> Q_2$

$$\frac{Q_1 \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1 >> Q_2} \quad (R\text{-accept}_1)$$

dla $\Phi \in M(F(Q))$ oraz $g \neq \text{exit}$

$$\frac{Q_1 \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q_2} \quad (R\text{-accept}_1)$$

dla $\Phi \in M(F(Q))$ oraz $g = \text{exit}$

Przestonięcie $Q \equiv \text{hide } R \text{ in } Q_1$

$$\frac{Q_1 \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} \text{hide } R \text{ in } Q'_1} \quad (R\text{-hide}_1)$$

dla $\Phi \in M(F(Q))$ oraz $g \notin \text{set}(R)$

$$\frac{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q'_1}{Q \xrightarrow{(1, \Phi', \lambda, \pi)} \text{hide } R \text{ in } Q'_1} \quad (R\text{-hide}_2)$$

dla $\Phi \in M(F(Q))$ oraz $g \in \text{set}(R)$

gdzie Φ' jest nowym indeksem różnym od wszystkich indeksów występujących w Q .

Instantacja procesu $Q \equiv p[g_1, \dots, g_n]$

$$\frac{Q_1[h_1 ::= g_1, \dots, h_n ::= g_n] \xrightarrow{(g, \Phi, \lambda, \pi)} Q'}{Q \xrightarrow{(g, \Phi, \lambda, \pi)} Q'} \quad (R-inst)$$

dla $\Phi \in M(F(Q[h_1 ::= g_1, \dots, h_n ::= g_n]))$, gdzie Q_1 jest treścią procesu p .

Probabilistyczny wybór akcji do wykonania

W celu przedstawienia probabilistycznego wyboru akcji do wykonania będziemy się posługiwać algebrą indeksów akcji. Niech

$$D = \{ \langle \Phi_1, x_1 \rangle, \dots, \langle \Phi_n, x_n \rangle \} \quad (10.12)$$

gdzie Φ_i oraz Φ_j są parami rozłącznych sekwencji indeksów akcji, oraz $x_i \in R_+$ dla $i = 1, \dots, n$. Liczby x_i będą interpretowane jako prawdopodobieństwa albo jako parametry intensywności. Interpretowane jako prawdopodobieństwa liczby będą wyznaczać do wykonania akcje natychmiastowe, a jako parametry intensywności – będą wyznaczać akcje czasowe.

Jeżeli $0 \leq x_i \leq 1$, dla $i = 1, \dots, n$, oraz jest spełniony warunek

$$\sum_{i=1}^n x_i = 1 \quad (10.13)$$

to zbiór postaci (10.12) nazywamy rozkładem prawdopodobieństwa.

Dla zbiorów postaci (10.12) definiujemy następujące operacje:

- *Mnożenie przez stałą*. Jeżeli $p \in PR$, to

$$p * D =_{\text{def}} \{ \langle \Phi_1, p \cdot x_1 \rangle, \dots, \langle \Phi_n, p \cdot x_n \rangle \} \quad (10.14)$$

Jeżeli $D = \emptyset$, to $p * D =_{\text{def}} \emptyset$.

- *Normalizacja*. Jeżeli zbiór D nie jest rozkładem prawdopodobieństwa, to może być sprowadzony do rozkładu prawdopodobieństwa za pomocą operacji $norm(D)$, zdefiniowanej następująco

$$norm(D) =_{\text{def}} \left(\sum_{i=1}^n x_i \right)^{-1} * D \quad (10.15)$$

- *Obcięcie bramkowe*. Jeżeli $F \subseteq L$, to obcięcie bramkowe jest zdefiniowane jako

$$D|_F =_{\text{def}} \{ \langle \Phi_i, x_i \rangle \mid \langle \Phi_i, x_i \rangle \in D \wedge name(\Phi_i) \in F \} \quad (10.16)$$

- *Suma*. Dla zbiorów D_1, D_2 postaci (10.12) ich suma jest określona jako suma mnogościowa.

- **Mnożenie.** Niech będą dane zbiory D_1, D_2 :

$$D_1 = \{ \langle \Phi_1, x_1 \rangle, \dots, \langle \Phi_n, x_n \rangle \}$$

$$D_2 = \{ \langle \Psi_1, y_1 \rangle, \dots, \langle \Psi_m, y_m \rangle \}$$

Ich iloczynem $D_1 \otimes D_2$ będzie zbiór postaci

$$D_1 \otimes D_2 =_{\text{def}} \{ \langle \Phi_i \Psi_j, x_i \cdot y_j \rangle \mid \langle \Phi_i, x_i \rangle \in D_1 \wedge \langle \Psi_j, y_j \rangle \in D_2 \wedge \text{name}(\Phi_i) = \text{name}(\Psi_j) \wedge \text{prior}(\Phi_i) = \text{prior}(\Psi_j) \} \quad (10.17)$$

- **Agregacja parametrów intensywności.** Niech będą dane zbiory D_1, D_2 :

$$D_1 = \{ \langle \Phi_1, \lambda_1 \rangle, \dots, \langle \Phi_n, \lambda_n \rangle \}$$

$$D_2 = \{ \langle \Psi_1, \mu_1 \rangle, \dots, \langle \Psi_m, \mu_m \rangle \}$$

gdzie λ_i, μ_j są parametrami intensywności. Agregacją parametrów intensywności, oznaczaną $D_1 \otimes D_2$, będzie zbiór postaci

$$D_1 \otimes D_2 =_{\text{def}} \left\{ \langle \Phi_i \Psi_j, \frac{\lambda_i}{ar_g(D_1)} \frac{\mu_j}{ar_g(D_2)} \min(ar_g(D_1), ar_g(D_2)) \rangle \mid \langle \Phi_i, \lambda_i \rangle \in D_1 \wedge \langle \Psi_j, \mu_j \rangle \in D_2 \wedge \text{name}(\Phi_i) = \text{name}(\Psi_j) = g \right\} \quad (10.18)$$

gdzie $ar_g(D) = \sum_{\langle \Phi, \lambda \rangle \in D \mid g} \lambda$ dla $g \in G$.

- **i-przeindeksowanie.** Operacja jest określona dla zbioru D postaci (10.12), dla którego $\text{name}(\Phi_i) \neq i$ dla $i = 1, \dots, n$. Wynikiem jest zbiór

$$reind(D) =_{\text{def}} \{ \langle \Phi'_1, \lambda_1 \rangle, \dots, \langle \Phi'_n, \lambda_n \rangle \} \quad (10.19)$$

gdzie Φ'_i zastępują Φ_i oraz $\text{name}(\Phi'_i) = i$.

Rozpatrzmy najpierw prawdopodobieństwa wyboru akcji natychmiastowych. Jeżeli dla danego wyrażenia Q zbiór $M(F(Q))$ zawiera więcej niż jeden element, to przejście do nowego wyrażenia jest wyznaczone przez pewien rozkład prawdopodobieństwa określony na zbiorze $M(F(Q))$. Rozkład ten jest zdefiniowany rekursywnie w sposób następujący:

$$PD(\text{stop}) = \emptyset$$

$$PD((g, \alpha, \infty, \pi); Q) = \{ \langle \alpha, 1 \rangle \}$$

$$PD(Q_1 [p] Q_2) = fact_1 * PD(Q_1) \cup fact_2 * PD(Q_2)$$

gdzie $fact_1, fact_2 \in PR$ są współczynnikami, których wartość zależy od prawdziwości formuł

$$empty_i =_{\text{def}} M(F(Q_i)) \cap M(F(Q_1 [p] Q_2)) = \emptyset \quad \text{dla } i = 1, 2.$$

Współczynniki te zdefiniowano w tabeli 10.3 (symbole T oraz F oznaczają odpowiednio prawdę i fałsz).

Tabela 10.3

$empty_1$	$empty_2$	$fact_1$	$fact_2$
F	F	$1-p$	p
F	T	1	0
T	F	0	1
T	T	0	0

Łatwo sprawdzić, że dla każdej wartości formuł $empty_1$ oraz $empty_2$, jeśli $PD(Q_1)$ oraz $PD(Q_2)$ są rozkładami prawdopodobieństwa, to również wyrażenie

$$fact_1 * PD(Q_1) \cup fact_2 * PD(Q_2)$$

jest rozkładem prawdopodobieństwa

$$PD(Q_1 | R|_{s,r} Q_2) = fact_1 * PD_1 \cup fact_{12} * PD_{12} \cup fact_2 * PD_2$$

gdzie:

$$PD_1 = norm(PD(Q_1)|_{A \setminus set(R')})$$

$$PD_{12} = norm(PD(Q_1)|_{set(R')}) \otimes PD(Q_2)|_{set(R')}$$

$$PD_2 = norm(PD(Q_2)|_{A \setminus set(R')})$$

$$set(R') = set(R) \cup \{\mathbf{exit}\}$$

$$FA(Q) = \{ \langle g, \pi \rangle \mid \Phi \in F(Q) \wedge g = name(\Phi) \wedge \pi = prior(\Phi) \}$$

$fact_1, fact_{12}, fact_2 \in PR$ są natomiast współczynnikami, których wartości zależą od prawdziwości następujących formuł:

$$empty_i =_{\text{def}} M(F(Q_i))|_{A \setminus set(R')} \cap M(F(Q_1 | R|_{s,r} Q_2)) = \emptyset \quad \text{dla } i = 1, 2,$$

$$empty_{12} = empty_{21} =_{\text{def}} M(F(Q_1 | R|_{s,r} Q_2))|_{set(R')} = \emptyset$$

Wartości współczynników określono w tabeli 10.4.

Tabela 10.4

$empty_1$	$empty_{12}$	$empty_2$	$fact_1$	$fact_{12}$	$fact_2$
T	F	F	0	s	$1-s$
T	F	T	0	1	0
T	T	F	0	0	1
T	T	T	0	0	0
F	F	F	$(1-s)*(1-r)$	s	$(1-s)*r$
F	F	T	$1-s$	s	0
F	T	F	$1-r$	0	r
F	T	T	1	0	0

$$PD(\text{hide } R \text{ in } Q) = PD(Q [\alpha_1 ::= \beta_1, \dots, \alpha_n ::= \beta_n])$$

gdzie zbiór indeksów $\{\alpha_1, \dots, \alpha_n\} = \{\alpha \mid \text{name}(\alpha) \in \text{set}(R)\}$ jest zastąpiony przez zbiór indeksów $\{\beta_1, \dots, \beta_n\}$, różnych od wszystkich indeksów występujących w Q .

$$PD(Q_1 >> Q_2) = PD(Q_1)$$

$$PD(Q_1 [> Q_2) = PD(Q_1 | \emptyset|_{0,1} Q_2) = PD(Q_1 [d] Q_2)$$

$$PD(P[h_1, \dots, h_m]) = PD(Q[g_1 ::= h_1, \dots, g_m ::= h_m])$$

gdzie Q jest treścią procesu $p[g_1, \dots, g_m]$, a $Q[g_1 ::= h_1, \dots, g_m ::= h_m]$ tekstowym zastąpieniem bramek $g_1, \dots, g_m$ przez $h_1, \dots, h_m$ w wyrażeniu Q .

Rozpatrzmy teraz prawdopodobieństwa wyboru akcji czasowych, gdy dla danego wyrażenia Q zbiór $M(F(Q))$ zawiera więcej niż jeden element. W celu wyznaczenia rozkładu prawdopodobieństwa określonego na zbiorze $M(F(Q))$ wprowadzimy funkcję pomocniczą $RATE(Q)$. Funkcja jest zdefiniowana rekursywnie w sposób następujący:

$$RATE(\text{stop}) = \emptyset$$

$$RATE((g, \alpha, \infty, \pi); Q) = \{<\alpha, \lambda>\}$$

$$RATE(Q_1 [p] Q_2) = RATE(Q_1) \cup RATE(Q_2)$$

$$RATE(Q_1 |R|_{s,r} Q_2) = RATE_1 \cup RATE_{12} \cup RATE_2$$

gdzie:

$$RATE_1 = RATE(Q_1)|_{A\text{set}(R')}$$

$$RATE_{12} = RATE(Q_1)|_{\text{set}(R')} \circledast RATE(Q_2)|_{\text{set}(R)}$$

$$RATE_2 = RATE(Q_2)|_{A\text{set}(R)}$$

$$RATE(\text{hide } R \text{ in } Q) = RATE(Q)|_{A\text{set}(R)} \cup \text{reind}(RATE(Q)|_{\text{set}(R)})$$

$$RATE(Q_1 >> Q_2) = RATE(Q_1)$$

$$RATE(Q_1 [> Q_2) = RATE(Q_1) \cup RATE(Q_2)$$

$$RATE(p[h_1, \dots, h_m]) = RATE(Q[g_1 ::= h_1, \dots, g_m ::= h_m])$$

Po zdefiniowaniu funkcji $RATE$ rozkład prawdopodobieństwa akcji czasowej jest określony wyrażeniem

$$PD(Q) = \text{norm}(RATE(Q)) \quad (10.20)$$

10.4. Wyprowadzanie łańcuchów Markowa

Specyfikację w języku MB-LOTOS można przetransformować w łańcuch Markowa z czasem ciągłym. Jeżeli wchodzące w skład specyfikacji wyrażenia behawioralne spełniają warunek regularności (rozdz. 2.), to łańcuch ma skończoną liczbę stanów.

Skończony jednorodny łańcuch Markowa jest dyskretnym procesem stochastycznym, określonym przez skończony zbiór stanów $S = \{s_1, \dots, s_n\}$ oraz prawdopodobieństwa przejść pomiędzy stanami, które nie są zależne od czasu. Takiemu łańcuchowi odpowiada stała macierz przejść

$$P = \begin{bmatrix} p_{11} & \dots & p_{1n} \\ \vdots & \vdots & \vdots \\ p_{n1} & \dots & p_{nn} \end{bmatrix} \quad (10.21)$$

gdzie p_{ij} oznacza prawdopodobieństwo przejścia ze stanu s_i do stanu s_j , dla $i, j = 1, \dots, n$.

Zastosowane podejście do transformacji opiera się na metodzie przedstawionej w pracy [Ajmone Marsan, Balbo, Conte 1984] dla uogólnionych stochastycznych sieci Petriego.

Przypomnijmy, że wyrażenia behawioralne, dla których zbiór akcji o najwyższym priorytecie zawiera tylko indeksy akcji natychmiastowych, nazywa się wyrażeniem *zanikającym*, wyrażenia zaś, dla których ten zbiór zawiera tylko indeksy akcji czasowych, nazywa się wyrażeniem *uchwytnym*.

Dalej zakładamy, że reprezentujące specyfikację wyrażenie Q jest komunikacyjnie zamknięte (rozdz. 2.). Gdy tak nie jest, to znaczy specyfikacja reprezentuje pewien system współpracujący z otoczeniem, dla zbudowania łańcucha Markowa konieczne jest przyjęcie pewnego modelu tego otoczenia w taki sposób, aby złożenie równoległe wyrażen reprezentujących system i otoczenie stało się zamkniętym wyrażeniem behawioralnym.

Budowany łańcuch Markowa reprezentuje bezpośrednio tylko wyrażenia uchwytnie. Wyrażenia znikające nie są bezpośrednio reprezentowane, gdyż przejścia natychmiastowe pomiędzy wyrażeniami znikającymi będą łączone z przejściami czasowymi.

Rozważając przejścia pomiędzy wyrażeniami uchwytymi, należy rozpatrzyć dwa przypadki:

- bezpośrednio, jednokrokowe przejście z wyrażenia uchwytne Q do innego wyrażenia uchwytne Q_1

$$Q \xrightarrow{e} Q_1 \quad (10.22)$$

gdzie $rate(e) \in R_+ \setminus \{0\}$,

- pośrednie, wielokrokowe przejście z wyrażenia uchwytne Q , poprzez przejścia pośrednie pomiędzy wyłącznie wyrażeniami znikającymi $Q_1, \dots, Q_k$, do innego wyrażenia uchwytne Q_{k+1}

$$Q \xrightarrow{e_1} Q_1 \xrightarrow{e_2} \dots \xrightarrow{e_k} Q_k \xrightarrow{e_{k+1}} Q_{k+1} \quad (10.23)$$

gdzie $rate(e_1) \in R_+ \setminus \{0\}$ oraz $rate(e_i) = \infty$, dla $i = 2, \dots, k+1$.

W pierwszym przypadku wyrażenie Q_1 jest bezpośrednim następnikiem wyrażenia Q , w drugim natomiast – jest następnikiem pośrednim.

Dla wyrażeń zanikających i wyrażeń uchwytnych odpowiednie rozkłady prawdopodobieństwa $PD(Q)$, określone odpowiednio na zbiorach akcji natychmiastowych i czasowych, były przedstawione w poprzedniej części rozdziału.

Na tej podstawie przedstawimy najpierw prawdopodobieństwo przejścia z wyrażenia zanikającego Q_1 , będącego bezpośrednim następnikiem wyrażenia uchwytnego Q , do wyrażenia uchwytnego Q_{k+1} . Wszystkie te przejścia odbywają się natychmiastowo.

Niech $B(Q)$ oznacza zbiór wszystkich wyrażeń behawioralnych osiągalnych z wyrażenia Q . Zbiór ten jest skończony, co wynika z założenia o regularności budowy rozpatrywanych wyrażeń. Zbiór $B(Q)$ dzieli się na dwa rozłączne podzbiory, oznaczane $VB(Q)$ oraz $TB(Q_0)$, oznaczające odpowiednio podzbiór wyrażeń zanikających i uchwytnych osiągalnych z wyrażenia Q , czyli spełniających własności

$$VB(Q) \cup TB(Q) = B(Q) \text{ oraz } VB(Q) \cap TB(Q) = \emptyset \quad (10.24)$$

W celu określenia prawdopodobieństw przejść z wyrażeń zanikających (będących bezpośrednimi następnikami wyrażeń uchwytnych) do wyrażeń uchwytnych wykorzystamy prawdopodobieństwa przejść z wyrażeń zanikających do zanikających oraz z wyrażeń zanikających do uchwytnych. Prawdopodobieństwa takie można zestawić w dwie macierze: V oraz T o wymiarach $n \times n$ oraz $n \times m$, gdzie $n = \text{card}(VB(Q))$ i $m = \text{card}(TB(Q))$. Elementy v_{ij} oraz t_{ij} obu macierzy są zdefiniowane następująco:

$$v_{ij} = \begin{cases} p_{ij} & \text{gdy } Q_i \xrightarrow{e_{ij}} Q_j \text{ z prawdopodobieństwem } p_{ij}, \text{ dla } Q_i, Q_j \in VB(Q_0), \\ 0 & \text{gdy nie istnieje przejście z } Q_i \text{ do } Q_j \end{cases} \quad (10.25)$$

oraz

$$t_{ij} = \begin{cases} p_{ij} & \text{gdy } Q_i \xrightarrow{e_{ij}} Q_j \text{ z prawdopodobieństwem } p_{ij}, \\ & \text{dla } Q_i \in VB(Q), Q_j \in TB(Q), \\ 0 & \text{gdy nie istnieje przejście z } Q_i \text{ do } Q_j \end{cases} \quad (10.26)$$

Sposób obliczania prawdopodobieństw stanowiących elementy obu macierzy został przedstawiony w poprzedniej części rozdziału.

Niech $V^{k+1} = V^k * V$, dla $k = 0, 1, \dots$, gdzie $*$ oznacza symbol mnożenia macierzy, oraz niech v_{ij}^k będzie elementem macierzy V^k . Element v_{ij}^k jest prawdopodobieństwem zajścia sekwencji przejść o długości k , prowadzących z zanikającego wyrażenia Q_i do zanikającego wyrażenia Q_j .

Ze względu na założenie, że rozważane są tylko wyrażenia behawioralne regularnie zbudowane, a więc także dozorowane, nie są możliwe nieskończenie długie sekwencje przejść wyłącznie pomiędzy wyrażeniami zanikającymi. Istnieje zatem pewna liczba k_0 taka, że V^k jest macierzą zerową dla $k > k_0$.

Element v_{ij}^* macierzy

$$V^- = \sum_{k=0}^{k_0} V^k \quad (10.27)$$

wyraża więc prawdopodobieństwo przejścia z zanikającego wyrażenia behawioralnego Q_i do zanikającego wyrażenia Q_j przez zajście wszystkich możliwych sekwencji tranzycji, w tym sekwencji pustej, prowadzących wyłącznie poprzez wyrażenia zanikające.

Ostatecznie element vt_{ij} macierzy

$$V^- * T = \sum_{k=0}^{k_0} V^k * T \quad (10.28)$$

wyraża prawdopodobieństwo przejścia z wyrażenia zanikającego Q_i do wyrażenia uchwytanego Q_j przez zajście wszystkich możliwych sekwencji przejść wyłącznie poprzez wyrażenia zanikające.

Dalsze rozważania są prowadzone przy założeniu, że analizowane wyrażenie Q jest wyrażeniem uchwytanym.

Niech $A_{m \times m}^T$, gdzie $m = \text{card}(TB(Q))$, będzie macierzą intensywności przejść λ_{ij} dla bezpośredniego przejścia z uchwytanego wyrażenia Q_i do uchwytanego wyrażenia Q_j .

Niech $A_{m \times n}^V$, gdzie $n = \text{card}(VB(Q))$, będzie macierzą intensywności przejść λ_{ij} dla bezpośredniego przejścia z uchwytanego wyrażenia Q_i do zanikającego wyrażenia Q_j .

Zdefiniujmy teraz macierz $A_{m \times m}^-$, która będzie macierzą intensywności przejścia z wyrażenia uchwytanego do uchwytanego poprzez dowolną liczbę wyrażeń zanikających. Macierz ta wyraża się przez

$$A^- = A^T + A^V * V^- * T \quad (10.29)$$

i stanowi macierz definiującą intensywności przejść w łańcuchu Markowa z czasem ciągłym, wyprowadzonym z etykietowanego sytemu przejść dla początkowego wyrażenia Q .

Macierz A^- jest podstawą do obliczenia macierzy P prawdopodobieństwa przejść pomiędzy wyrażeniami uchwytanymi oraz do określenia średniego czasu pozostawania w danym wyrażeniu uchwytanym.

Element p_{ij} macierzy P , określający prawdopodobieństwo przejścia z uchwytnej wyrażenia Q_i do uchwytnej wyrażenia Q_j , jest zdefiniowany wzorem

$$p_{ij} = \frac{\lambda_{ij}}{\sum_{k=1}^m \lambda_{ik}} \quad (10.30)$$

w którym λ_{ij} są elementami macierzy A^- .

Średni natomiast czas τ_i pozostawania w wyrażeniu uchwytnej Q_i oblicza się z zależności

$$\tau_i = \frac{1}{\sum_{k=1}^m \lambda_{ik}} \quad (10.31)$$

W celu ilustracji przedstawionych wyżej rozważań rozpatrzmy przykład prostego systemu obsługi. System składa się z serwera, reprezentowanego procesem S , oraz dwóch klientów, reprezentowanych procesami C_1 oraz C_2 . Każdy z klientów działa cyklicznie, na przemian wykonując obliczenia lokalne i obliczenia we współpracy z serwerem. Definicje procesów klientów, dla $i = 1, 2$, mają postać

```

process  $C_i[loc, work, req, serv]$  : noexit :=
    ( $loc, \infty, \pi$ );
    ( $work, \lambda_{work}, 0$ );
    ( $req, \infty, \pi$ );
    ( $serv, \lambda_{serv}, 0$ );
     $C_i[loc, work, req, serv]$ 
endproc

```

W celu uproszczenia dalszych zapisów akcje nie będą jawnie indeksowane. Natychmiastowe akcje loc oraz req rozpoczynają odpowiednio fazę obliczeń lokalnych i obliczeń wspólnych, akcje czasowe $work$ oraz $serv$ modelują zaś realizację tych faz.

Definicja procesu serwera ma postać

```

process  $S[req, serv]$  : noexit :=
    ( $req, \infty, \pi$ );
    ( $serv, \lambda_{serv}, 0$ );
     $S[req, serv]$ 
endproc

```

Akcja natychmiastowa req rozpoczyna fazę współpracy, a akcja czasowa $serv$ modeluje okres współpracy. Współpraca pomiędzy procesami klientów a serwerem jest oparta na następujących zasadach:

1. Dostęp klientów do serwera jest rozłączny, przy czym prawdopodobieństwo dostępu obu procesów do serwera, w przypadku jednoczesnego ubiegania się o taki dostęp, jest jednakowe.
2. Jeżeli jeden z procesów klientów zamierza rozpocząć fazę pracy lokalnej, a drugi – współpracy z serwerem, to prawdopodobieństwo wyboru współpracy z serwerem jest trzykrotnie większe od wyboru pracy lokalnej.

Specyfikacja całego systemu spełniającego podane zasady przyjmuje postać

specification *System*[]() : **noexit** :=

behaviour

hide *req, serv, loc, work in*

($C_1[loc, work, req, serv] \mid \emptyset \mid_{*, 0.5} C_2[loc, work, req, serv]$)

$\mid [req, serv] \mid_{0.75, *}$

$S[req, serv]$

endspec

Symbol * w miejscu przeznaczonym na umieszczenie prawdopodobieństwa oznacza, że wartość prawdopodobieństwa, w danym kontekście, jest nieistotna. Rzeczywiście, symbol gwiazdki w złożeniu równoległym procesów C_1 oraz C_2 wynika z faktu, że procesy te nie synchronizują się ze sobą, a zatem nie nastąpi sytuacja, w której należy podejmować decyzję o wyborze akcji synchronizujących się. Druga z gwiazdek wynika z faktu, że proces S nie wykonuje akcji niezależnych od swego otoczenia, a zatem nigdy nie będzie podejmowana decyzja o wyborze tego rodzaju akcji.

Dla wyrażenia behawioralnego, stanowiącego treść specyfikacji, wyznaczymy zbiór wyrażeń osiągalnych. W celu skrócenia dalszych zapisów wyrażenie to i wyrażenia od niego pochodne będą zapisywane bez operatora przestąpienia. Dodatkowo przed każdym wyrażeniem będzie wstawiona etykieta (liczba naturalna) jednoznacznie je identyfikująca. Wyrażenie początkowe będzie mieć postać

1: $(C_1[loc, work, req, serv]$

$\mid \emptyset \mid_{*, 0.5} C_2[loc, work, req, serv]$)

$\mid [req, serv] \mid_{0.75, *} S[req, serv]$

Z wyrażenia o etykiecie 1 są osiągalne następujące wyrażenia:

2: $((work, \lambda_{work}, 0); (req, \infty, \pi); (serv, \lambda_{serv}, 0); C_1[loc, work, req, serv]$

$\mid \emptyset \mid_{*, 0.5} C_2[loc, work, req, serv]$)

$\mid [req, serv] \mid_{0.75, *} S[req, serv]$

3: $(C_1[loc, work, req, serv]$

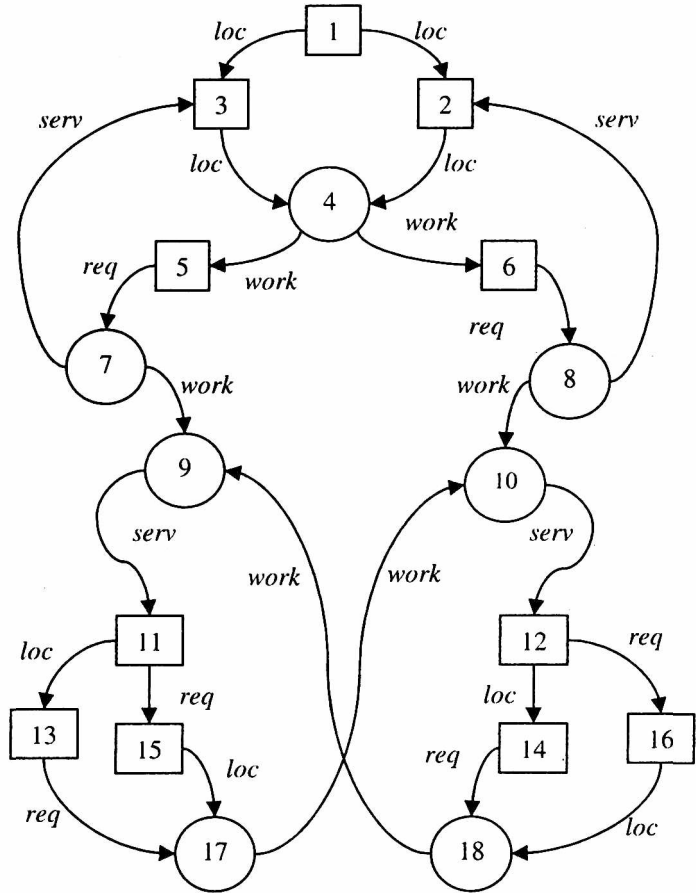
$\mid \emptyset \mid_{*, 0.5} (work, \lambda_{work}, 0); (req, \infty, \pi); (serv, \lambda_{serv}, 0); C_2[loc, work, req, serv]$)

$\mid [req, serv] \mid_{0.75, *} S[req, serv]$

4: $((work, \lambda_{work}, 0); (req, \infty, \pi); (serv, \lambda_{serv}, 0); C_1[loc, work, req, serv]$

- 16: ((serv, λ_{serv} , 0); $C_1[loc, work, req, serv]$
 $|\emptyset|^{*, 0.5} C_2[loc, work, req, serv]$)
 $[[req, serv]]_{0.75, *} (serv, \lambda_{serv}, 0); S[req, serv]$
- 17: ((work, λ_{work} , 0); (req, ∞ , π); (serv, λ_{serv} , 0); $C_1[loc, work, req, serv]$
 $|\emptyset|^{*, 0.5} (serv, \lambda_{serv}, 0); C_2[loc, work, req, serv]$)
 $[[req, serv]]_{0.75, *} (serv, \lambda_{serv}, 0); S[req, serv]$
- 18: ((serv, λ_{serv} , 0); $C_1[loc, work, req, serv]$
 $|\emptyset|^{*, 0.5} (work, \lambda_{work}, 0); (req, \infty, \pi); (serv, \lambda_{serv}, 0); C_2[loc, work, req, serv]$)
 $[[req, serv]]_{0.75, *} (serv, \lambda_{serv}, 0); S[req, serv]$

Zbiór wyrażeń uchwytnych $TB(Q)$, gdzie Q jest wyrażeniem o etykietce 1, składa się z {4, 7, 8, 9, 10, 17, 18}, a zbiór wyrażeń zanikających $VB(Q)$ składa się z {1, 2, 3, 5, 6, 11, 12, 13, 14, 15, 16}. Na grafie tranzycji na rysunku 10.1 wyrażenia zanikające są przedstawiane w postaci kwadratów, a uchwytne – w postaci kółek.



Rys. 10.1. Graf tranzycji
dla przykładowego wyrażenia

Macierz prawdopodobieństw $V_{11 \times 11}$ jest macierzą, której jedynymi niezerowymi elementami są:

$$v_{1,2} = v_{1,3} = v_{11,13} = v_{11,15} = v_{12,14} = v_{12,16} = 0,5$$

Dolne indeksy elementów tej i następnych macierzy są etykietami wyrażeń.

Zauważmy, że V^2 jest macierzą zerową, dlatego $V^- = V$.

Podobnie, jedynymi niezerowymi elementami macierzy $T_{11 \times 7}$ są:

$$t_{2,4} = t_{3,4} = t_{5,7} = t_{6,8} = t_{13,17} = t_{15,17} = t_{16,18} = t_{14,18} = 1$$

Macierz $V^- * T$, wyrażająca prawdopodobieństwa przejścia z wyrażenia zanikającego do wyrażenia uchwytne, przez zajście wszystkich możliwych sekwencji przejść wyłącznie poprzez wyrażenia zanikające, ze względu na prostotę systemu, ma tylko elementy 0 oraz 1. Elementami o wartościach 1 są:

$$vt_{1,4} = vt_{2,4} = vt_{3,4} = vt_{5,7} = vt_{6,8} = vt_{11,17} =$$

$$vt_{13,17} = vt_{15,17} = vt_{12,18} = vt_{16,18} = vt_{14,18} = 1$$

Macierz intensywności przejść dla bezpośredniego przejścia z wyrażenia uchwytne do zanikającego $\lambda_{7 \times 1}^V$ ma następujące elementy niezerowe:

$$\lambda_{7,3}^V = \lambda_{8,2}^V = \lambda_{serv}$$

$$\lambda_{4,5}^V = \lambda_{4,6}^V = \lambda_{9,11}^V = \lambda_{10,12}^V = \lambda_{work}$$

Macierz $\lambda_{7 \times 7}^-$ wyraża intensywności przejścia z wyrażeń uchwytne do uchwytne poprzez dowolną liczbę wyrażeń zanikających. Niezerowymi jej elementami są:

$$\lambda'_{4,7} = \lambda'_{4,8} = \lambda'_{7,9} = \lambda'_{8,10} = \lambda'_{17,10} = \lambda'_{18,9} = \lambda_{work}$$

$$\lambda'_{9,17} = \lambda'_{10,18} = \lambda'_{7,4} = \lambda'_{8,2} = \lambda_{serv}$$

Niezerowymi elementami macierzy P prawdopodobieństwa przejść pomiędzy wyrażeniami uchwytymi są:

$$p_{4,7} = p_{4,8} = 0,5$$

$$p_{17,10} = p_{18,9} = p_{9,17} = p_{10,18} = 1$$

$$p_{7,9} = p_{8,10} = \frac{\lambda_{work}}{\lambda_{work} + \lambda_{serv}}$$

$$p_{7,4} = p_{8,2} = \frac{\lambda_{serv}}{\lambda_{work} + \lambda_{serv}}$$

10.5. Silna bisymulacja markowska

Silna rozszerzona bisymulacja markowska wyznacza klasy równoważności na zbiorze wyrażeń behawioralnych, które mają te same własności funkcjonalne i nie-funkcjonalne – czasowe i probabilistyczne. W celu zdefiniowania tej relacji wprowadzimy pomocnicze oznaczenia i definicje.

Jeżeli R jest relacją równoważności na zbiorze X , to przez $X|_R$ będzie oznaczany zbiór ilorazowy wyznaczany przez R , a przez $[x]$ będzie oznaczana klasa abstrakcji generowana przez $x \in X$.

Przez $\{|x_1, \dots, x_n|\}$ będziemy oznaczać wielozbiór nad zbiorem liczb rzeczywistych. Wyrażenie $\sum \{|x_1, \dots, x_n|\}$ będzie sumą wszystkich elementów tego wielozbioru. W szczególnym przypadku, dla wielozbioru pustego, z definicji $\sum \emptyset = 0$.

Definicja 10.1

Zagregowanym prawdopodobieństwem dla wyrażenia behawioralnego Q względem zbioru wyrażeń C będziemy nazywać funkcję częściową $AProb$ o sygnaturze

$$AProb : BEH \times A \times AP \times 2^{BEH} \rightarrow PR \quad (10.32)$$

zdefiniowaną przez wyrażenie

$$\begin{aligned} & AProb(Q, g, \pi, C) \\ &= \sum \{|p| \langle \Phi, p \rangle \in PD(Q) \wedge Q \xrightarrow{(g, \Phi, \infty, \pi)} Q' \wedge Q' \in C\} \end{aligned} \quad (10.33)$$

Zagregowane prawdopodobieństwo jest sumą prawdopodobieństw przejścia z wyrażenia Q do dowolnego wyrażenia ze zbioru C , w wyniku wykonania akcji natychmiastowych o nazwie $g \in A$ i priorytecie π .

Definicja 10.2

Zagregowaną intensywnością dla wyrażenia behawioralnego Q względem zbioru wyrażeń C będziemy nazywać funkcję częściową $ARate$ o sygnaturze

$$ARate : BEH \times A \times \mathbb{P}(BEH) \rightarrow R_+ \quad (10.34)$$

zdefiniowaną przez wyrażenie

$$\begin{aligned} & ARate(Q, g, C) \\ &= \sum \{| \lambda | \langle \Phi, \lambda \rangle \in RATE(Q) \wedge Q \xrightarrow{(g, \Phi, \lambda, 0)} Q' \wedge \lambda \in \mathbb{R}_+ \wedge Q' \in C\} \end{aligned} \quad (10.35)$$

Zagregowana intensywność jest sumaryczną intensywnością przejścia z wyrażenia Q do dowolnego wyrażenia ze zbioru C w wyniku wykonania akcji czasowej o nazwie $g \in A$.

Definicja 10.3

Relacja równoważności $B \subseteq BEH \times BEH$ jest silną rozszerzoną bisymulacją markowską (*EMB*), wtedy i tylko wtedy, gdy jeśli $\langle Q_1, Q_2 \rangle \in B$, to dla wszystkich $g \in A$, $\pi \in AP$, i dowolnej klasy $C \in BEH|_B$ zachodzą związki:

$$AProb(Q_1, g, \pi, C) = AProb(Q_2, g, \pi, C), \quad (10.36)$$

$$ARate(Q_1, g, C) = ARate(Q_2, g, C). \quad (10.37)$$

Fakt, że wyrażenia Q_1 oraz Q_2 są w relacji *EMB* będzie zapisywany w postaci $Q_1 \sim Q_2$.

Lemat 10.1

Niech $\sim_{EMB}$ będzie mnogościową sumą wszystkich silnych rozszerzonych bisymulacji markowskich. Relacja $\sim_{EMB}$ jest największą silną rozszerzoną bisymulacją markowską (największą *EMB*).

Dowód. Analogiczny dowód jest pokazany w pracy (Bernardo&Gorieri, 1998).

Relacja $\sim_{EMB}$ jest również kongruencją. Pokazuje to poniższe twierdzenie, które sprostawa się do wykazania, że relacja jest zachowywana przez wszystkie operatory języka oraz przez instancje procesów.

Twierdzenie 10.2

Niech $Q_1, Q_2 \in BEH$. Jeżeli $Q_1 \sim_{EMB} Q_2$, to:

1. Dla dowolnych $g \in A$, $\alpha \in \mathbb{N}$, $\lambda \in AR$, $\pi \in AP$

$$(g, \alpha, \lambda, \pi); Q_1 \sim_{EMB} (g, \alpha, \lambda, \pi); Q_2 \quad (10.38)$$

2. Dla dowolnych $Q \in BEH$, $p \in PR$

$$Q_1[p]Q \sim_{EMB} Q_2[p]Q \quad \text{oraz} \quad Q[p]Q_1 \sim_{EMB} Q[p]Q_2 \quad (10.39)$$

3. Dla dowolnych $Q \in BEH$, $r, s \in PR$, $\Gamma \subseteq G$

$$Q_1|[R]|_{s,r}Q \sim_{EMB} Q_2|[R]|_{s,r}Q \quad \text{oraz} \quad Q|[R]|_{s,r}Q_1 \sim_{EMB} Q|[R]|_{s,r}Q_2 \quad (10.40)$$

4. Dla dowolnego $Q \in BEH$

$$Q_1[p > Q] \sim_{EMB} Q_2[p > Q] \quad \text{oraz} \quad Q[p > Q_1] \sim_{EMB} Q[p > Q_2] \quad (10.41)$$

5. Dla dowolnego $Q \in BEH$

$$Q_1 >> Q \sim_{EMB} Q_2 >> Q \quad \text{oraz} \quad Q >> Q_1 \sim_{EMB} Q >> Q_2 \quad (10.42)$$

6. Dla dowolnego $R \in SeqG$

$$\text{hide } R \text{ in } Q_1 \sim_{EMB} \text{hide } R \text{ in } Q_2 \quad (10.43)$$

Dowód – zob. Dodatek 1.

W celu pokazania, że relacja $\sim_{EMB}$ jest zachowana przez instancje procesów, wprowadzimy pewne definicje pomocnicze.

Proces niemający definicji w danym zbiorze definicji jest określany jako *wolny względem tego zbioru*.

Wyrażenie Q jest *częściowo dozorowane względem danego zbioru definicji procesów*, jeśli dowolna nazwa procesu P , występująca w Q , jest albo wolna względem danego zbioru definicji, albo proces ten ma definicję postaci $P[h_1, \dots, h_m] := Q'$ oraz każdy proces o nazwie P' , który występuje w Q' , jest dozorowany.

Wprowadźmy pojęcie rekursywnego podstawienia wyrażenia Q za wywołanie procesu $p[R]$ w wyrażeniu Q_1 . Oznaczenie

$$Q_1[<P[R] ::= Q>]$$

ma następującą definicję:

$$\text{stop}[<P[g_1, \dots, g_n] ::= Q>] \equiv \text{stop}$$

$$(\text{exit}, \alpha, \infty, \infty)[<P[R] ::= Q>] \equiv (\text{exit}, \alpha, \infty, \infty)$$

$$((g, \alpha, \lambda, \pi); Q_1)[<P[R] ::= Q>] \equiv (g, \alpha, \lambda, \pi); Q_1[<P[R] ::= Q>]$$

$$(Q_1 [p] Q_2)[<P[R] ::= Q>] \equiv Q_1[<P[R] ::= Q>] [p] Q_2[<P[R] ::= Q>]$$

$$(Q_1 [S]_{s,r} Q_2)[<P[R] ::= Q>] \equiv Q_1[<P[R] ::= Q>] [S]_{s,r} Q_2[<P[R] ::= Q>]$$

$$(\text{hide } R \text{ in } Q_1)[<P[R] ::= Q>] \equiv \text{hide } R \text{ in } Q_1[<P[R] ::= Q_p>]$$

$$(Q_1 >> Q_2)[<P[R] ::= Q>] \equiv Q_1[<P[R] ::= Q>] >> Q_2[<P[R] ::= Q>]$$

$$(Q_1 [> Q_2])[<P[R] ::= Q>] \equiv Q_1[<P[R] ::= Q>] [> Q_2[<P[R] ::= Q>]]$$

$$(p[R'])[<P[R] ::= Q>] \equiv$$

$$\begin{cases} Q & \text{gdy } p[R'] \equiv P[R] \\ p[R'] & \text{gdy } \neg(p[R'] \equiv P[R]) \wedge p[R'] \text{ wolny w zbiorze definicji} \\ Q_p[<P[R] ::= Q>] & \text{gdy } \neg(p[R'] \equiv P[R]) \wedge Q_p \text{ jest treścią } p[R'] \end{cases}$$

$$(Q_1)[<P[R] ::= Q>] \equiv Q_1[<P[R] ::= Q>]$$

Twierdzenie 10.3

Niech $Q_1, Q_2 \in BEH$ będą częściowo dozorowanymi wyrażeniami względem danego zbioru definicji oraz niech $P[g_1, \dots, g_n]$ będzie jedynym wolnym procesem występującym w Q_1 oraz Q_2 . Niech dane będą dwie rekursywne definicje procesów:

$$\begin{aligned} P_1[g_1, \dots, g_n] &:= Q_1[<P[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n]>] \\ P_2[g_1, \dots, g_n] &:= Q_2[<P[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n]>] \end{aligned} \quad (10.44)$$

Jeżeli $Q_1 \sim_{EMB} Q_2$, to $P_1[g_1, \dots, g_n] \sim_{EMB} P_2[g_1, \dots, g_n]$.

Dowód – zob. Dodatek 1.

10.6. Uwagi końcowe

W bieżącym rozdziale przedstawiono wydajnościowe rozszerzenie języka LOTOS, które opiera się przede wszystkim na wcześniejszych pracach: [Huzar, Magott 1996], [Huzar, Magott 1997b], [Huzar, Magott 2001]. Istotnymi cechami język MB-LOTOS są akcje natychmiastowe i czasowe, z wykładniczym rozkładem prawdopodobieństwa, oraz probabilistyczne operatory wyboru i złożenia równoległego. Wybór akcji do wykonania następuje, w pierwszej kolejności, na podstawie priorytetów, a następnie na wyborze probabilistycznym. Mechanizmy te, eliminując niedeterminizm, umożliwiają tym samym prowadzenie analizy probabilistycznej specyfikacji wyrażonych w tym języku. Wprowadzone mechanizmy probabilistycznego wyboru akcji są zależne od otoczenia. Jest to rozwinięcie języka w stosunku do wcześniejszych prac: [Huzar, Magott 1999a], [Huzar, Magott 1996], [Huzar, Magott 197a], a także prac innych autorów, na przykład: [Schieferdecker 1995], [Miguel, Fernandez, Vidaller 1993], [Hermanns, Rettelbach 1996].

Dla przedstawionego rozszerzenia języka zdefiniowano relację silnej bisymulacji markowskiej i pokazano, że jest kongruencją, co oznacza kompozycyjność języka. Pokazano też jak dla specyfikacji wyrażonej w języku MB-LOTOS wyprowadzać łańcuchy Markowa. Opisane wyprowadzanie łańcuchów Markowa było zastosowane między innymi w pracach: [Babczyński, Huzar, Magott 1999], [Huzar, Magott 2000], [Babczyński, Huzar, Magott 2000].

Algebry procesowe, oparte na językach CCS, CSP, ACP, LOTOS i innych, były przedmiotem wielu prac, na przykład: [Reed, Roscoe 1986] [Bolognesi, Lucidi 1992], [Fidge 1992], [Hermanns, Rettelbach 1994], [Brinksma, Katoen, Latella 1995], [Schieferdecker 1995], [Rettelbach 1995], [Hermanns, Rettelbach 1996], [Hillston 1996], [Bravetti, Bernardo 2000], [Hermanns, Herzog, Katoen 2002]. Wśród nich warto wyróżnić: [Nicollin, Sifakis 1992], [Bernardo, Gorrieri 1998] i [Hermanns, Herzog, Katoen 2002], ze względu na zawarty w nich obszerny przegląd literatury.

Można stwierdzić, że prace nad algebrami procesowymi przerodziły się w oddzielny nurt poszukiwania modeli formalnych i narzędzi programowych wspomagających ich stosowanie. Na przykład dla LOTOSa pierwsze rozszerzenie stochastyczne, oparte na semitańcuchach Markowa przedstawiono w pracy [Rico, von Bochmann 1991] oraz w pracy [Valderrutten, Hjej, Benzekri Gazal 1992], pokazującej wyprowadzanie systemów kolejkowych na podstawie rozszerzonych specyfikacji. Stochastyczne rozszerzenie LOTOSa z dowolnymi rozkładami, ale ograniczoną kompozycyjnością, jest przedstawione w pracy [Ajmone Marsan, Balbo, Conte, Donatelli, Franceschinis 1994]. Język TIPP [Götz, Herzog, Rettelbach 1993], początkowo pomyślany jako algebra z dowolnymi rozkładami czasów wykonywania akcji, skupił się ostatecznie na rozkładach wykładniczych, podobnie jak PEPA: [Hilston 1996] i EMPA [Bernardo, Gorrieri 1998]. Prace [Brinksma E., Katoen J.-P., Latella D., 1995] i [Katoen,

Brinksma, Latella, Langerak 1996] są propozycją stosowania semantyki nieprzepłotowej, umożliwiającej rozważanie bardziej ogólnych rozkładów prawdopodobieństwa. Inną propozycją podobnego podejścia jest praca [Priami 1996], dotycząca stochastycznej wersji rachunku π , oraz praca [Herzog 1996]. Propozycje niemarkowskich modeli zawiera między innymi praca [D'Argenio, Hermanns, Katoen 1999].

Do budowania modeli wydajnościowych sięga się także po inne metody formalne. Przykładem są materiały warsztatów na temat systemów czasu rzeczywistego [Gnesi, Schieferdecker, Rennoch 2000]. Wśród różnych metod formalnych ważną rolę odgrywają sieci Petriego. Do przykładowych prac należą tu między innymi: [Murphy 1991], [Juanole, Atamna 1991], [Magott 2005].

Krótkie omówienie innych metod formalnych przedstawiono w następnym rozdziale.

11. Zakończenie

11.1. LOTOS a inne techniki formalne

Omawiany LOTOS jest jedną z wielu technik formalnych wykorzystywanych w procesie wytwarzania systemów informatycznych. Poniżej omówiono krótko inne metody formalne i półformalne, które zasługują na uwagę ze względów historycznych lub stopień ich upowszechnienia. Do metod wspólnie rozwijanych z LOTOSEm należą, wspomniane wcześniej, standardy SDL [ITU-T 1999] i ESTELLE [ISO 9074]. LOTOS, SDL i ESTELLE były przeznaczone do definiowania i badania standardów dotyczących sieci komputerowych i sieci telekomunikacyjnych. Oprócz norm ISO i wielu odrębnych opisów, jak np. pozycje [Ellsberg, Hogrefe, Sarma 1997], [Szmuc 1998] dotyczące SDL, czy prace [Budkowski, Dembiński 1987], [Budkowski, Dembiński 1989], dotyczące ESTELLE, łączny opis tych trzech technik zawiera książka [Turner 1993].

Oprócz nich były rozwijane inne metody, przeznaczone ogólnie do wspomagania procesów specyfikowania i projektowania systemów rozproszonych [ISO/ODP 10746]. Należą do nich między innymi metody i związane z nimi narzędzia programowe, jak SPIN i UPPAL, rozwijane przede wszystkim w środowiskach akademickich [Babbich, Deontio 2002]. Oddzielną grupę stanowią metody wykorzystujące sieci Petriego, a w ostatnim okresie metodyki oparte na języku UML czy też modele kolejkowe.

Wspólną cechą LOTOSa, SDL, ESTELLE, SPIN i UPPAL jest to, że służą do konstruowania specyfikacji wykonywalnych i wspólnie odwołują się do pojęcia, różnie definiowanej, maszyny stanowej.

SDL [ITU-T 1999] jest w zasadzie językiem półformalnym, przeznaczonym do opisu systemów reaktywnych i systemów czasu rzeczywistego. Specyfikacja w SDL przedstawia pewną rozszerzoną maszynę stanową. Struktura specyfikacji jest określona przez zbiór równoległych komunikujących się procesów. Komunikacja odbywa się w sposób asynchroniczny, za pośrednictwem komunikacyjnych kanałów (kolejek FIFO skończonej długości). Zachowanie procesów jest określane oddzielnymi diagramami aktywności, ukazującymi reakcje procesów na przychodzące pobudzenia. Poszczególne aktywności mogą być opisywane w języku naturalnym lub w języku programowania. Opis zachowania dopuszcza niedeterminizm. Procesy operują na własnej przestrzeni danych. Początkowo SDL (podobnie jak LOTOS) opierał się wyłącznie na abstrakcyjnych typach danych, później dołączono typy danych z języka programowania C.

Dzięki intuicyjnej, graficznej postaci specyfikacji uzyskał dużą popularność zarówno w środowiskach akademickich, jak i przemysłowych. (Warto dodać, że powszechnie stosowanej notacji graficznej zawdzięcza swą popularność także język UML – de facto standard w wytwarzaniu systemów oprogramowania.)

Głównym obszarem zastosowań SDL są systemy telekomunikacyjne, dlatego wśród komercyjnych narzędzi programowych wspomagających posługiwanie się tym językiem jest *Telelogic Tau Suite* – produkt firm Swedish Telelogic oraz Object-Geode, umożliwiające generowanie kodu w C i C++, testowanie i wykonywanie aplikacji na podstawie specyfikacji w SDL. Część testująca narzędzia wykorzystuje język TTCN – standard ISO definiowania testów dla systemów telekomunikacyjnych.

W ogólnej ocenie języka SDL pozytywnymi cechami są: modularne podejście do tworzenia specyfikacji, z wyraźnym oddzieleniem definiowania struktury od definicji zachowań. Język jest zorientowany na szybkie tworzenie aplikacji, stąd wynika powiązanie z językami programowania C oraz C++, a ze względu na obszar zastosowań, również powiązanie z notacją ASN.1 do zapisu składni abstrakcyjnej typów danych oraz do kodowania danych transmitowanych w sieciach telekomunikacyjnych. Badanie własności prototypu wygenerowanego na podstawie specyfikacji w SDL wymaga pełnego testowania. Walidacja samej specyfikacji wymaga natomiast jej transformacji do innego formalizmu, na przykład do LOTOSa.

ESTELLE, podobnie jak SDL, bazuje na rozszerzonych maszynach stanowych i korzysta z notacji języka Pascal do operowania na danych. Dzięki temu specyfikacje w ESTELLE są podstawą szybkiego generowania wykonywalnego prototypu. ESTELLE, bardziej niż LOTOS, jest ukierunkowany na implementację.

Specyfikacja składa się z modułów, które mogą się ze sobą komunikować asynchronicznie za pośrednictwem dwukierunkowych kanałów komunikacyjnych. Moduł dysponuje własnym zbiorem danych, na których można wykonywać określone operacje. Moduły mogą być zagnieżdżane. ESTELLE ma dobrze określone podstawy teoretyczne i znalazł zastosowanie w licznych projektach; ocenia się jednak, że dla zastosowań przemysłowych wymagane byłoby opracowanie odpowiednio przyjaznych środowisk wspierających [Babbich, Deotio 2002].

SPIN (*Simple ProMeLa Interpreter*) jest narzędziem przeznaczonym do specyfikacji, symulacji i walidacji protokołów komunikacyjnych. SPIN korzysta z notacji *ProMeLa* (*Process Meta Language*), zbliżonej do języka C, co ułatwia szybkie generowanie wykonywalnego kodu. SPIN dostarcza ponadto mechanizmów weryfikacji modelowej, m.in. przez badanie inwariantów i formuł logiki temporalnej. Podstawowymi komponentami specyfikacji są procesy. Komunikacja pomiędzy procesami odbywa się przez kanały komunikacyjne – kolejki FIFO o skończonej długości. Gdy kolejki mają długość zerową, komunikacja jest synchroniczna – odbywa się w trybie *rendez vous*, natomiast w przypadku przeciwnym komunikacja jest asynchroniczna. Zachowanie procesu jest opisywane przez zbiór możliwych tranzycji pomiędzy stanami procesu. Tranzycja może nastąpić, gdy jest prawdziwy odpowiedni dozór – wyrażenie logiczne określone na wartościach zmiennych lokalnych procesu lub zmiennych globalnych, oraz gdy zajdzie odpowiednia akcja wejścia-wyjścia. Możliwa jest sytuacja niedeterministyczna, gdy możliwe są do wykonania przynajmniej dwie tranzycje.

Dzięki otwartemu, bezpłatnemu dostępowi SPIN należy do najbardziej popularnych i najczęściej stosowanych narzędzi do specyfikacji i weryfikacji protokołów, co wyraża się między innymi organizacją cyklu warsztatów poświęconych rozwojowi i zastosowaniom tego narzędzia. Jednym z interesujących zastosowań jest wykorzystanie SPIN do badania własności specyfikacji zapisanych w SDL. Osiąga się to dzięki prostej transformacji specyfikacji w SDL na specyfikację w SPIN.

UPPAL (skrót od *UPPsala and Aalborg* – miast, w których mieszczą się współpracujące ze sobą uniwersytety) jest narzędziem służącym do modelowania, symulacji i weryfikacji automatów czasowych. Strukturę specyfikacji wyznacza zbiór synchronicznie komunikujących się procesów. Dopuszcza się, że komunikacja może trwać skończony odcinek czasu. Głównym obszarem zastosowania narzędzia jest specyfikowanie i badanie protokołów komunikacyjnych czasu rzeczywistego.

Pełniejsze omówienie i porównanie wymienionych wyżej technik można znaleźć w pracy [Babich, Deotto 2002]. Oprócz wymienionych, istnieje również wiele innych, opartych na automatach czasowych i maszynach stanowych, które dostarczają narzędzi do specyfikowania i badania różnych własności, na przykład VerICS [Demiński, Janowska, Janowski, Penczek, Półrola, Szreter, Woźna, Zbrzezny 2003].

Sieci Petriego stanowią oddzielną, bardzo szeroką klasę technik, z którą są związane liczne narzędzia wspomagające. Aktualny przegląd w tym zakresie zawiera książka [Girault 2003], przykładów wykorzystania sieci Petriego dostarczają prace [Szmuc 1998], [Szmuc, Motet 1998], [Szpyrka 1999], [Magott 2005], ciekawe związki pomiędzy sieciami Petriego a rachunkiem CCS analizuje Kułakowski [2004]. Równie szeroką klasę, rozwijaną od dziesiątków lat, stanowią modele kolejkowe, przykładem ich zastosowań do systemów komputerowych jest książka [Czachórski 1999].

Oprócz omawianych, należy wspomnieć o technikach, które wyrosły z języków programowania czasu rzeczywistego w środowiskach przemysłowych. Należą do nich takie języki i związane z nimi środowiska wspomagające, jak: Esterel [Esterel Technologies, 2005], Signal [Le Guernic, Gautier, Le Borgne, Le Maire 1991] i Lustre [Halbwachs, Caspi, Raymond, Pilaud 1991], stosowane między innymi w europejskim przemyśle lotniczym i awionice. Omówienie tych języków i ich zastosowań zawiera praca [Benveniste, Caspi, Edwards, Halbwachs, Le Guernic, De Simone 2003]. Do tej samej grupy technik można zaliczyć język Ada i związane z nim środowiska projektowe. Obecnie trwają prace nad ustanowieniem standardu ISO wersji Ada 2005, stanowiącej rozwinięcie poprzedniej wersji Ada 1995 [Huzar, Fryźlewicz, Dubielewicz, Hnatkowska, Waniczek 1998].

11.2. LOTOS a UML

Współczesne metody wytwarzania systemów oprogramowania opierają się na paradygmacie modelowania obiektowego – cykl wytwarzania oprogramowania jest po-

strzegany jako proces budowy ciągu powiązanych ze sobą modeli [Hnatkowska, Huzar, Tuzinkiewicz 2001], a podstawowym językiem modelowania jest obecnie UML [UML 2003], [Booch, Rumbaugh, Jacobson 1999]. UML jest językiem półformalnym – składnia bezkontekstowa (opis jest wyrażony w podzbiorze języka UML) i kontekstowa (opisana w języku OCL – *Object Constraint Language*) są opisane formalnie, semantyka natomiast jest opisana w języku naturalnym.

LOTOS natomiast nie jest ani językiem projektowania obiektowego, ani też nie może być uważany za język programowania obiektowego. Uzasadnienie tego faktu wynika z następującego rozumowania:

Zakładając, że LOTOS jest językiem programowania obiektowego, należałoby przyjąć, że instancja procesu jest odpowiednikiem obiektu, a definicja procesu jest odpowiednikiem klasy. Podobieństwo polega na tym, że instancja procesu komunikuje się ze swym otoczeniem przez dobrze określony interfejs, chociaż ograniczony tylko do wskazania bramek, bez jawnego określenia typów komunikowanych danych, i stanowi hermetyzację danych, na których proces operuje. Na tym jednak kończą się podobieństwa z językiem programowania obiektowego, gdyż pomiędzy procesami nie ma mechanizmu dziedziczenia (dziedziczenie jest ograniczone w LOTOSie tylko do typów danych) ani też mechanizmu polimorfizmu [Górski 1999].

Zakładając zaś, że LOTOS jest językiem projektowania obiektowego, należałoby oczekiwać dodatkowych własności: po pierwsze – definiowania asocjacji pomiędzy klasami (definicjami procesów), wraz z odpowiednimi ograniczeniami licznosciowymi, oraz – po drugie – możliwości tworzenia powiązanych instancji obiektów (instancji procesów) spełniających te ograniczenia. W języku UML własności te są wyrażane przez diagramy klas i diagramy obiektów – instancje diagramów klas. LOTOS może definiować dowolny zbiór procesów, ale – poza zagnieżdżeniami – nie może określać żadnych związków pomiędzy ich definicjami, struktura powiązań pomiędzy instancjami procesów jest natomiast ograniczona tylko do tych możliwości, jakie dają operatory składania procesów – w zasadzie chodzi tylko o operator złożenia równoległego. Wprawdzie tworzenie różnych struktur powiązań pomiędzy instancjami procesów jest możliwe, ale wymaga to oddzielnego postępowania [Haj-Hussein, Logrippo 1991], prowadzącego do znacznej rozbudowy wyrażen behawioralnych.

Analiza możliwości stosowania języka LOTOS w kontekście obiektowego podejścia do wytwarzania oprogramowania była rozważana jeszcze w latach dziewięćdziesiątych ubiegłego wieku, na przykład [Gibson 1993], [Hnatkowska 1996], ale ostatecznie praktyka przesądziła o dominującej roli języka UML. Nie oznacza to, że LOTOS stał się niepotrzebny, ale że może pełnić inną rolę. Tą rolę, w stosunku do języka UML, jest użycie LOTOSa do definiowania behawioralnego aspektu semantyki UML. Przykładami takiego zastosowania LOTOSa są prace: [Clark, Moreira 2000], [Hnatkowska, Huzar 2000], [Hnatkowska, Huzar 2001], [de Saqui-Sannes, Aprville, Lohr, Sénac, Courtiat 2002], [Cichoń, Huzar 2005], [Cichoń, Huzar 2006] oraz [Walkowiak 2006].

11.3. Diagramy stanów UML a LOTOS

W tym podrozdziale przedstawiono propozycję reprezentacji diagramów stanów języka UML w postaci wyrażeń behawioralnych LOTOSa, opartą na pracy [Hnatkowska, Huzar 2001].

Diagramy stanów w UML są graficzną reprezentacją maszyn stanowych i opierają się na koncepcji map stanów Harela [Harel 1987], których istotnym mechanizmem jest zagnieżdżanie stanów. Zagnieżdżanie oznacza, że we wnętrzu stanu może być zawarta inna maszyna stanowa, reprezentowana nowym diagramem stanów. Zagnieżdżanie stanów pozwala ograniczać eksplozję stanów, która jest zasadniczym problemem związanym ze stosowaniem metod operacyjnych.

Stany i tranzycje pomiędzy stanami są reprezentowane odpowiednio w postaci wierzchołków i łuków grafu. Wyróżnia się dwie podstawowe kategorie stanów:

- Pseudostany, na które składają się stan początkowy i końcowy, łączniki historyczne (płytki i głęboki), rozgałęzienie, rozwidlenie i złączenie równoległe.
- Stany właściwe, krótko stany, wśród których wyróżnia się stany proste i złożone, a te ostatnie dzieli się na złożone sekwencyjnie i równoległe. W stanie prostym nie są zagnieżdżone inne stany. W stanie złożonym, „rodzicielskim”, są zagnieżdżone inne stany, stany „potomne”, nazywane też podstanami. Podstan jest bezpośrednim podstanem danego stanu, jeśli nie jest zagnieżdżony w innym stanie, a przeciwnym razie jest podstanem przechodnim tego stanu.

Diagram stanów można przedstawić jako etykietowany graf. Graf diagramu stanów jest definiowany jako szóstka

$$S = \langle \text{Box}N, \text{child}B, \text{type}B, \text{default}B, \text{Arc}N, \text{Arc} \rangle \quad (11.1)$$

gdzie:

$\text{Box}N$ jest skończonym zbiorem nazw wierzchołków, reprezentujących stany właściwe; nazwa wierzchołka jest utożsamiana z nazwą stanu.

$\text{child}B \subseteq \text{Box}N \times \text{Box}N$ jest relacją hierarchii stanów: $\langle b_1, b_2 \rangle \in \text{child}B$ oznacza, że b_2 jest bezpośrednim „potomkiem” stanu „rodzicielskiego” b_1 . Zbiór $\text{Box}N$ oraz relacja hierarchii $\text{child}B$ definiują drzewo. Korzeń tego drzewa r nie ma „rodziców”, a jego liście nie mają „potomków”. Zwrotne i przechodnie domknięcie relacji $\text{child}B$ jest oznaczane przez $\text{child}B^*$.

$\text{type}B : \text{Box}N \rightarrow \{\text{PRIM}, \text{XOR}, \text{AND}, \text{FIN}\}$ jest funkcją określającą typ danego wierzchołka. Korzeń r , z definicji jest typu XOR , liście są typu PRIM lub FIN , pozostałe zaś wierzchołki mogą być albo typu XOR (sekwencyjnie złożony podstan), albo AND (równoległe złożony podstan). Liście typu FIN reprezentują stany końcowe.

$\text{default}B : \text{Box}N \rightarrow 2^{\text{Box}N}$ jest funkcją, która dla danego stanu złożonego określa te jego podstany, które są stanami początkowymi. Dla wierzchołka typu XOR jest to

zawsze zbiór jednoelementowy, dla wierzchołka typu *AND* jest to zbiór co najmniej dwuelementowy. Dla wierzchołka typu *PRIM* zbiór podstawów początkowych jest pusty. Rozszerzeniem funkcji *defaultB* jest funkcja $DefaultB : BoxN \rightarrow 2^{BoxN}$, określona przez następujące własności: $b \in DefaultB(b)$ oraz dla wierzchołków $b' \in BoxN$, takich że $\langle b, b' \rangle \in childB^*$, wymaga się, aby $b' \in DefaultB(b)$ wtedy i tylko wtedy, gdy $defaultB(b') \subseteq DefaultB(b)$.

$ArcN$ jest skończonym zbiorem nazw łuków: $BoxN \cap ArcN = \emptyset$.

$Arc \subseteq BoxN \times ArcN \times BoxN$ jest zbiorem łuków. Łuk $\alpha \in Arc$ jest trójką $\langle b_1, a, b_2 \rangle$ z wierzchołkiem początkowym $source(\alpha) = b_1$, końcowym $target(\alpha) = b_2$ oraz nazwą $name(\alpha) = a$.

Zarówno wierzchołki, jak i łuki grafu diagramu stanów S mają swoje etykiety.

Wierzchołki (stany) są etykietowane przez pięć elementów:

- akcje (niepodzielne i nieprzerywalne czynności obliczeniowe) wykonywane bezpośrednio po wejściu do stanu,
- ciągi akcji (aktywności) wykonywane podczas przebywania w danym stanie; ciągi te mogą być przerywane w dowolnej chwili po wykonaniu każdej z akcji,
- akcje wykonywane bezpośrednio przed wychodzeniem ze stanu,
- tranzycje wewnętrzne, określone przez dwa elementy: zdarzenia, jakie mogą zachodzić podczas przebywania w stanie, i przez akcje, które są wykonywane po zajściu tych zdarzeń,
- zdarzenia odroczone – zdarzenia, których zajście jest tylko rejestrowane i przekazywane do kolejnych stanów.

Każdy z wymienionych elementów jest opcjonalny (może nie występować); w dalszych rozważaniach uwzględnimy tylko trzy pierwsze elementy – akcje wejściowe, ciągi akcji wewnętrznych i akcje wyjściowe.

Łuki grafu diagramu stanów są etykietowane trzema elementami:

- zdarzeniem wyzwalającym tranzycję ze stanu, od którego łuk wychodzi, do stanu, do którego łuk prowadzi,
- dozorem – wyrażeniem logicznym, określonym przez wartości atrybutu obiektu (ogólniej instancji klasyfikatora), którego zachowanie jest opisywane diagramem stanów; prawdziwość dozoru w momencie zajścia zdarzenia wyzwalającego jest warunkiem koniecznym tranzycji,
- akcją, która jest wykonywana podczas tranzycji ze stanu do stanu (akcja jest wykonywana po wykonaniu akcji wyjściowej w stanie, z którego następuje tranzycja, a przed wykonaniem akcji wejściowej stanu, do którego następuje tranzycja).

Każdy z wymienionych elementów jest opcjonalny, przy czym jawny brak zdarzenia wyzwalającego w etykiecie łuku oznacza, że zakończenie ciągu akcji wewnętrznych w danym stanie jest interpretowane jako zdarzenie wyzwalające tranzycję. Wyróżnia

się cztery rodzaje zdarzeń wyzwalających: zdarzenie odbioru sygnału, wywołania operacji, zdarzenie czasowe i zdarzenie zmiany wartości. Sygnały i wywołania operacji mogą przenosić wartości.

Diagram stanów jest określony jako trójka

$$MS = \langle S, labB, labA \rangle \quad (11.2)$$

gdzie:

S jest grafem diagramu,

$labB$ oraz $labA$ są funkcjami etykietującymi wierzchołki oraz łuki.

Funkcja $labB$ każdemu wierzchołkowi $b \in BoxN$ przypisuje piętkę

$$labB(b) = \langle entry(b), do(b), exit(b), deferrable(b), internal(b) \rangle \quad (11.3)$$

gdzie:

$entry(b)$, $do(b)$, $exit(b)$ są ciągami akcji,

$deferrable(b)$ jest ciągiem par: zdarzenie, akcja,

$internal(b)$ jest ciągiem zdarzeń.

Zakładamy, że dla wierzchołka-korzenia $labB(r)$ jest puste.

Funkcja $labA$ każdemu łukowi $a \in ArcN$ przypisuje trójkę

$$labA(a) = \langle trigger(a), effect(a), guard(a) \rangle \quad (11.4)$$

gdzie:

$trigger(a)$ jest zdarzeniem wyzwalającym tranzycję po łuku a ; gdy element ten jest pusty – oznacza to, że tranzycja może nastąpić w wyniku zakończenia aktywności wewnętrznej,

$effect(a)$ jest akcją; gdy element ten jest pusty – oznacza to brak akcji,

$guard(a)$ jest warunkiem logicznym; gdy element ten jest pusty – oznacza to, że warunek jest tożsamościowo prawdziwy.

W dalszych rozważaniach będzie wykorzystywana następująca funkcja pomocnicza

$$TypeA : ArcN \rightarrow \{EV-LAB, TO-LAB, UN-LAB\} \quad (11.5)$$

która ma odróżniać jedną z trzech sytuacji, gdy zdarzenie wyzwalające tranzycję (a) jest przesłaniem sygnału lub wywołaniem operacji – *EV-LAB*, (b) zdarzeniem czasowym – *TO-LAB* lub (c) zdarzenie nie jest jawnie określone – *UN-LAB*.

Semantykę diagramu stanów opisuje się w terminach ciągów konfiguracji, w których może się znajdować. Standard UML zmiany konfiguracji opisuje w sposób nieformalny, dalej pokazujemy jak zmiany te można wyrazić w sposób formalny dzięki zastosowaniu LOTOSa.

Konfigurację diagramu stanów w danym momencie określa się przez zbiór stanów aktywnych. Konfiguracją diagramu stanów S jest podzbiór stanów $B \subseteq BoxN$, taki że

$r \in B$ oraz dla każdego wierzchołka $b \in B$, jeżeli jego typ $typeB(b) = AND$, to wszystkie jego potomne wierzchołki należą do B , a jeżeli $typeB(b) = XOR$, to dokładnie jeden jego potomek należy do B . Hierarchia wierzchołków $childB$, ograniczona do zbioru B , stanowi poddrzewo $\langle B, childB \cap B \times B \rangle$ drzewa $\langle BoxN, childB \rangle$. Początkowa konfiguracja diagramu S jest określona jako $B_{ini} = defaultB(r)$.

Niech $lca(b_1, b_2)$ oznacza najmniejszy wspólny poprzednik wierzchołków b_1 oraz b_2 w drzewie $\langle BoxN, childB \rangle$, to znaczy $\langle lca(b_1, b_2), b_i \rangle \in childB^*$ dla $i = 1, 2$, oraz nie istnieje inny wierzchołek b taki, że $\langle b, b_i \rangle \in childB^*$ oraz $\langle lca(b_1, b_2), b \rangle \in childB^*$.

Dwa wierzchołki $b_1, b_2 \in BoxN$ są niezależne w sensie równoczesnego wykonywania aktywności – krótko: są niezależne, jeżeli żaden z nich nie jest poprzednikiem drugiego i $type(lca(b_1, b_2)) = AND$. Dwa łuki $\langle b_1, a_1, b'_1 \rangle$ oraz $\langle b_2, a_2, b'_2 \rangle$ są niezależne, jeżeli wierzchołki $lca(b_1, b'_1)$ oraz $lca(b_2, b'_2)$ są niezależne.

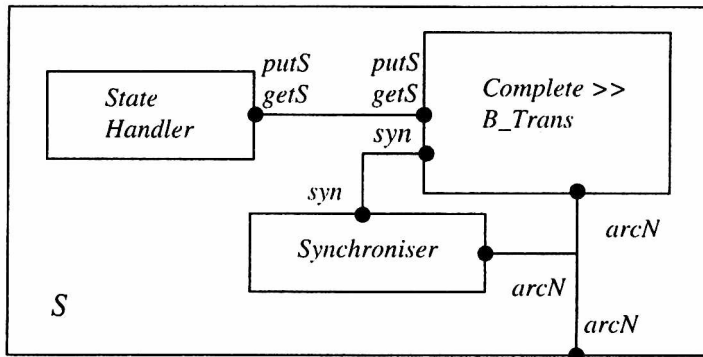
Tranzycja T z konfiguracji B do konfiguracji B' jest określona jako największy zbiór nazw wzajemnie niezależnych łuków, których wierzchołki początkowe należą do zbioru B .

Zmiana konfiguracji następuje w wyniku zajścia zdarzenia wyzwalającego. Tranzycja diagramu stanów zależy od konfiguracji i od otoczenia diagramu, które jest źródłem zdarzeń wyzwalających. Wyróżnia się dwa rodzaje tranzycji: tranzycje niskopoziomowe pomiędzy stanami prostymi i tranzycje wysokopoziomowe, związane z przechodzeniem od lub do stanu złożonego. Tranzycje wysokopoziomowe mają priorytet nad tranzycjami niskopoziomowymi.

W przedstawianej formalizacji przyjmuje się następujące założenia ograniczające:

- Rozważamy wyłącznie tak zwane strukturalne diagramy stanów, to znaczy takie, które nie zawierają łuków przecinających granice stanów. Oznacza to, że jeżeli $\langle b_1, a, b_2 \rangle \in Arc$, to istnieje taki $b' \in B$ typu XOR , że $\langle b', b_1 \rangle, \langle b', b_2 \rangle \in childB$ (inaczej: wierzchołki b_1, b_2 mają wspólnego rodzica typu XOR). Ograniczenie to ma na celu nie tylko uproszczenie rozważań, ale jest ono także zaleceniem metodycznym; niestosowanie się do tego zalecenia jest analogonem całkowicie swobodnego wykorzystywania instrukcji skoku bezwzględnego w programowaniu.
- Rozważamy tylko stany normalne oraz tylko dwa pseudostany: początkowy i końcowy. Ograniczenie to ze względów praktycznych nie jest istotne, gdyż bardzo rzadko wykorzystuje się inne rodzaje pseudostanów, a z teoretycznego punktu widzenia siła ekspresji diagramów stanów nie ulega tu ograniczeniu.
- Funkcja etykietująca wierzchołki $labB$ jest ograniczona tylko do trzech pierwszych elementów, pomijamy tranzycje wewnętrzne i zdarzenia odraczane. Wynika to z chęci uproszczenia rozważań, gdyż uwzględnienie zwłaszcza zdarzeń odraczanych wprowadziłoby istotną rozbudowę przedstawianych transformacji.
- Dodatkowo przyjmujemy, że nazwy wszystkich akcji są unikatowe, a zbiory nazw akcji, wierzchołków, łuków i zdarzeń są parami rozłączne.

Omawiana transformacja każdemu diagramowi stanów przyporządkowuje wyrażenie behawioralne. Transformacja polega na przypisaniu wyrażenia behawioralnego każdemu wierzchołkowi (stanowi) diagramu stanów, a następnie połączeniu wyrażeń składowych w jedno wyrażenie końcowe. Strukturę wyrażenia końcowego, złożonego z trzech komunikujących się procesów, pokazano na rysunku 11.1.



Rys. 11.1. Struktura specyfikacji diagramu stanów

Proces *StateHandler* obejmuje stany aktywne, odpowiadające stanom złożonym sekwencyjnie, to jest stanom typu *XOR*.

Proces *Synchronizer*, przez bramkę *syn*, blokuje odbiór zdarzeń, przez bramkę *arcN*, w stanie, do którego nastąpi tranzycja dopóty, dopóki nie stanie się on stanem stabilnym, to znaczy do chwili zakończenia wykonywania akcji wejściowej w tym stanie i we wszystkich jego stanach zagnieżdżonych. Blokowanie jest ograniczone do najmniejszego wspólnego następnika stanów, z którego i do którego nastąpiła tranzycja.

Proces *Complete* jest odpowiedzialny za wykonanie zagnieżdżonych akcji wejściowych i wyjściowych stanów złożonych. Po nadejściu kolejnego zdarzenia proces wykonuje następujące działania:

1. Dla danego łuku a , pod warunkiem, że $typeB(source(a)) - XOR$, pobiera przez bramkę *getS* od procesu *StateHandler* aktywny stan (zbiór nazw stanów).
2. Wykonuje, być może zagnieżdżone, akcje wyjściowe stanu, z którego następuje tranzycja.
3. Wykonuje, być może zagnieżdżone, akcje wejściowe stanu, do którego następuje tranzycja.
4. Ustala nowy stan aktywny i komunikuje go do procesu *StateHandler* przez bramkę *putS*.
5. Odblokowuje odbiór zdarzeń w najmniejszym wspólnym następniku stanów, od którego i do którego następuje tranzycja.

Główną częścią transformacji diagramu stanów jest funkcja $B_Trans(b)$, która wyznacza wyrażenia behawioralne dla dowolnego $b \in BoxN$. Wyrażenie takie reprezentuje konfigurację początkową poddiagramu stanów o korzeniu b . Wierzchołek-korzeń r reprezentuje, oczywiście, konfigurację początkową całego diagramu stanów. Funkcja ta wykorzystuje dwie funkcje pomocnicze $procD^*(b)$ oraz $procD(b)$, generujące definicje procesów, reprezentujących odpowiednio wierzchołki typu *AND* oraz *XOR*.

W definicji funkcji $B_Trans(b)$ zastosowano następujące oznaczenia pomocnicze: równoległe i alternatywne złożenia skończonego zbioru wyrażeń behawioralnych P_i , dla $i \in I$, są oznaczane odpowiednio $\prod_{i \in I} P_i$ oraz $\sum_{i \in I} P_i$, czyli

$$\prod_{i=1, \dots, n} P_i \equiv P_1 [] P_2 [] \dots [] P_n \text{ oraz } \sum_{i=1, \dots, n} P_i \equiv P_1 ||| P_2 ||| \dots ||| P_n \quad (11.6)$$

Dalej zakładamy, że wszystkie tranzycje etykietowane zdarzeniem czasowym są unikalnie numerowane, oraz że z danego stanu wychodzi co najwyżej jeden łuk etykietowany zdarzeniem czasowym, czyli $card\{a \in leaving(b) \mid typeA(a) = TO-LAB\} \leq 1$.

$$B_Trans(b) = \begin{cases} b[ExtArcN] & \text{jeśli } typeB(b) = PRIM \text{ lub } typeB(b) = FIN \\ B_Trans(defaultB(b)) [> b[ExtArcN] & \text{jeśli } typeB(b) = XOR \\ & \text{oraz dla wszystkich } b' \in childB(b) \text{ jest generowana} \\ & \text{definicja nowego procesu przez funkcję } procD(b') \\ (\prod_{b' \in childB(b)} B_Trans(b')) [> b[ExtArcN] & \text{jeśli } typeB(b) = AND \\ & \text{oraz dla wszystkich } b' \in childB(b) \text{ jest generowana} \\ & \text{definicja nowego procesu przez funkcję } procD^*(b') \end{cases}$$

gdzie:

$$ExtArcN = list(\{putS, getS, syn\} \cup ActionNames \cup ArcN) \quad (11.7)$$

$$ActionNames = \bigcup_{b \in BoxN} (AS_Names(entry(b)) \cup AS_Names(exit(b)) \cup AS_Names(do(b))) \quad (11.8)$$

$$EventArcN = \{a \in ArcN \bullet typeA(a) = EV-LAB\} \quad (11.9)$$

Funkcja $list(S)$ transformuje zbiór S na listę zawierającą wszystkie elementy ze zbioru S , a funkcja $AS_Names(a)$ dany ciąg akcji a transformuje na zbiór nazw akcji należących do tego ciągu.

Definicja procesu reprezentującego wierzchołek-korzeń r jest niezależna od diagramu stanów i ma postać

$$\text{process } r[\text{ExtArcN}]: \text{noexit} := \text{stop endproc} \quad (11.10)$$

Definicja funkcji procD^* , dla danego stanu b , przedstawia się następująco:

$$\text{procD}^*(b) = \text{process } b[\text{ExtArcN}]: \text{noexit} := \text{stop endproc} \quad (11.11)$$

Funkcja procD^* jest bardziej złożona, jej definicja, dla danego stanu b , przedstawia się następująco:

1. Jeżeli $\text{typeB}(b) = \text{FIN}$, to

$$\text{procD}(b) = \text{process } b[\text{ExtArcN}]: \text{noexit} := \text{stop endproc} \quad (11.12)$$

2. Jeżeli $\text{typeB}(b) \neq \text{FIN}$, to

$$\begin{aligned} \text{ProcD}(b) = & \text{process } b[\text{ExtArcN}]: \text{noexit} := \\ & (\quad \text{Do_Trans}(b) \text{ Time_Out}(\text{leaving}(b)) \\ & \quad \text{Disabling_Part}(b) \\ &) \quad \text{After_Time_Out}(\text{leaving}(b)) \\ & \text{endproc} \end{aligned} \quad (11.13)$$

Funkcje, które występują w definicji funkcji procD , są określone w sposób następujący:

$$\begin{aligned} \text{Do_Trans}(b) = & \begin{cases} \text{do}(b); & \text{gdy } \text{do}(b) \text{ jest niepustą sekwencją akcji} \\ \text{puste} & \text{w przypadku przeciwnym} \end{cases} \\ \text{Disabling_Part}(b) = & \begin{cases} [> (\sum_{\langle b, a, b' \rangle \in \text{Arc}} (\text{trigger}(a); \text{exit})) \\ >> \text{Complete}[\text{ExtArcN}](a) \\ >> \text{B_Trans}(b') \\ & \text{gdy istnieje } x \in \text{leaving}(b) \text{ taki, że } \text{typeA}(x) = \text{EV-LAB} \\ \text{puste} & \text{w przypadku przeciwnym} \end{cases} \end{aligned}$$

$$\text{Time_Out}(A) = \begin{cases} \text{stop} & \text{gdy dla dowolnego } a \in A \text{ typeA}(a) = \text{EV-LAB} \\ \text{trigger}(a); \text{exit} & \text{gdy istnieje } a \in A \text{ taki, że } \text{typeA}(a) = \text{TO-LAB} \end{cases}$$

$$\text{After_Time_Out}(A) = \begin{cases} >> \text{Complete}[\text{ExtArcN}](a) \\ >> \text{B_Trans}(\text{target}(a)) \\ & \text{gdy istnieje } a \in A \text{ taki, że } \text{typeA}(a) = \text{TO-LAB} \\ \text{puste} & \text{w przypadku przeciwnym} \end{cases}$$

Pełna definicja funkcji transformującej SM_Trans jest dosyć złożona; pełną jej postać przedstawiono w Dodatku 2.

Poprawność zdefiniowanej funkcji została zbadana przez testowanie. Najpierw przygotowano zestaw testowych diagramów stanów, które ręcznie, zgodnie ze zdefiniowaną funkcją transformacji, przekształcono na specyfikacje w LOTOSie. Dla testowych diagramów stanów opracowano też scenariusze zachowań – ciągi zdarzeń. Specyfikacje te następnie badano za pomocą pakietu programowego LOLA [Pavon, Larrabeiti, Rabay 1995]. Badanie polegało na wykonaniu i porównaniu scenariuszy zachowań diagramów stanów i ich odpowiedników w LOTOSie.

Inny sposób wyrażania semantyki diagramów stanów UML, nieodwołujący się do języka LOTOS, przedstawiono w pracy [Huzar, Magott 2000], a wykorzystanie tej semantyki w badaniu wydajności systemów zawarto w pracach: [Babczyński, Huzar, Magott 2001], [Babczyński, Huzar, Magott 2003].

11.4. Perspektywy technik formalnych

LOTOS jest jedną z technik formalnych wykorzystywaną w procesie wytwarzania oprogramowania. Nasuwa się pytanie: jaka jest obecnie rola technik formalnych w przemysłowym wytwarzaniu oprogramowania? Opinie na ten temat są zróżnicowane, ale wydaje się, że dominuje przeświadczenie, iż stosowanie metod formalnych znacznie spowalnia proces wytwarzania oprogramowania, nie dając przy tym istotnych korzyści. W stosunku do wszystkich współcześnie stosowanych metod formalnych wysuwa się następujące zastrzeżenia [Logrippo 2000]:

- Żadna z metod oddzielnie nie ma zastosowania w pełnym cyklu wytwarzania oprogramowania.
- Znajomość metod formalnych wśród analityków i projektantów jest ograniczona, a w przypadku ich znajomości nie ma dominującego standardu.

Można postawić tezę, że jedną z przyczyn wymienionych słabości jest skoncentrowanie uwagi rozwijanych metod formalnych tylko na obliczeniowej i informacyjnej perspektywie modelowania, czyli skupienie uwagi wyłącznie na wymaganiach funkcjonalnych, podczas gdy od metod oczekuje się także możliwości modelowania innych perspektyw, w tym inżynierskiej i technologicznej [ISO/IEC 10746-1, 1995].

Należy też dodać, że narzędzia programowe, wspomagające stosowanie technik formalnych, są ciągle niedojrzałe. Na przykład dla języka LOTOS najbardziej zaawansowane środowisko CADP [Garavel, Lang, Mateescu 2001] można uważać za narzędzie o charakterze półprzemysłowym, a pierwsze narzędzie – pakiet LOLA [Pavon, Larrabeiti, Rabay 1995] ma charakter akademicki.

Zastrzeżenia te nie oznaczają, że nie dostrzega się zalet metod formalnych (precyzja, jednoznaczność, możliwość wsparcia programowego). Wynikiem krytyki i jednocześnie doceniania metod formalnych jest rozwój metod pośrednich – półformalnych. Obecnie UML, jako najbardziej rozpowszechniony język specyfikacji, modelowania

i dokumentowania projektów informatycznych, jest wyrazem takiego pośredniego podejścia.

Nie oznacza to jednak, że metody formalne nie są wcale używane. Są one bowiem stosowane w wytwarzaniu oprogramowania związanego z bezpieczeństwem. Podczas projektowania tych systemów jednym z wymagań jest dowiedzenie poprawności wytworzonego systemu. Samo testowanie, chociaż niezbędne, nie jest wystarczające – wymaga się weryfikacji zbudowanego systemu względem specyfikacji albo zagwarantowania jego poprawności na mocy poprawności konstrukcji transformujących specyfikację w implementację. W tym przypadku koszt i trud projektowania są rekompensowane uzyskaniem wysokiego, chociaż nieabsolutnego, stopnia przekonania o poprawności i wiarygodności zbudowanego systemu.

Dążenie do formalizacji znajduje też częściowe odzwierciedlenie w obecnych tendencjach rozwoju metod wytwarzania oprogramowania. Współczesne podejście do wytwarzania oprogramowania jest co raz częściej oparte na MDA (*Model Driven Architecture*) [MDA 2003], co oznacza, że proces wytwarzania wyraża się jako ciąg transformacji modeli. Zdefiniowanie transformacji modeli, podobnie jak i samych modeli, wymaga posługiwania się metodami formalnymi. Opracowany nowy standard grupy OMG [MOF QVT 2005] dotyczy właśnie języka definiowania transformacji.

Niezależnie od stosowania metod półformalnych lub formalnych warunkiem koniecznym zastosowania każdej metody jest wsparcie narzędziowe w postaci kompleksowych środowisk projektowo-implementacyjnych. Chodzi o narzędzia programistyczne, wspierające uczestników procesu wytwarzania przynajmniej od fazy wymagań, przez fazy analizy i projektowania, aż do fazy implementacji, a często również fazy analizy dziedziny. W obrębie każdej fazy od narzędzia wspomagającego oczekuje się, oprócz możliwości edycji modelu, badania jego spójności wewnętrznej, a w przechodzeniu pomiędzy kolejnymi fazami oczekuje się mechanizmów transformacji elementów jednego modelu w kolejny model lub możliwości sprawdzenia zgodności pomiędzy modelami [Huzar, Kuźniarz, Reggio, G., Sourrouille 2005]. Spełnienie wymienionych oczekiwań jest możliwe wyłącznie za pomocą formalizacji modeli i ich transformacji.

Dodatek 1

Dowód twierdzenia 10.2

Dowód poprowadzono według następującego schematu: Zakładając, że relacja D jest silną EMB , taką że $\langle Q_1, Q_2 \rangle \in D$, konstruuje się pewną relację B – rozszerzenie relacji D , która jest również silną EMB , taką że $\langle Q'_1, Q'_2 \rangle \in B$, gdzie wyrażenia Q'_1 oraz Q'_2 są w jednej postaci (10.)–(10.) Ponieważ relacja $\sim_{EMB}$ jest sumą wszystkich relacji będących silnymi bisymulacjami, dlatego wystarczy pokazać, że jeśli para $\langle Q'_1, Q'_2 \rangle \in B$, to również $\langle Q'_1, Q'_2 \rangle \in \sim_{EMB}$, dlatego w poszczególnych wyróżnionych przypadkach, wykorzystując indukcję strukturalną, pokazano, że jeśli $Q_1 \sim_{EMB} Q_2$, to $\langle Q'_1, Q'_2 \rangle \in B$, gdzie B jest relacją EMB , a Q'_1 oraz Q'_2 są w jednej postaci (10.)–(10.).

W celu pokazania, że $\langle Q'_1, Q'_2 \rangle \in B$, należy dowieść, że:

- jeżeli Q'_1, Q'_2 są wyrażeniami behawioralnymi takimi, że $mprior(M(F(Q'_i))) > 0$, dla $i = 1, 2$, czyli że $M(F(Q'_i))$ zawiera tylko akcje natychmiastowe, to

$$AProb(Q'_1, g, \pi, C) = AProb(Q'_2, g, \pi, C) \quad (D.1)$$

dla dowolnych $g \in A$, $\pi \in AP$ oraz $C \in BEH|_B$,

- jeżeli Q'_1, Q'_2 są wyrażeniami behawioralnymi takimi, że $mprior(M(F(Q'_i))) = 0$, dla $i = 1, 2$, czyli że $M(F(Q'_i))$ zawiera tylko akcje czasowe, to

$$ARate(Q'_1, g, C) = ARate(Q'_2, g, C) \quad (D.2)$$

dla dowolnych $g \in A$, $\pi \in AP$ oraz $C \in BEH|_B$.

Poniżej podano dowody tylko dla równości dotyczących zagregowanych prawdopodobieństw. Dowody odpowiednich równości dla zagregowanych intensywności są podobne i dlatego je pominięto.

Dalej rozważono tylko przypadki, gdy otoczenie wyrażenia Q jest gotowe do uczestniczenia w akcjach na bramce g , gdyż w przeciwnym razie z semantyki języka wynika, że $AProb(Q, g, \pi, C) = 0$.

1. Niech $D \subseteq BEH \times BEH$ będzie relacją EMB taką, że $\langle Q_1, Q_2 \rangle \in D$. Zdefiniujmy relację B w sposób następujący:

$$B = (D \cup \{ \langle (g, \alpha, \infty, \pi); Q_1, (g, \alpha, \infty, \pi); Q_2 \rangle, \langle (g, \alpha, \infty, \pi); Q_2, (g, \alpha, \infty, \pi); Q_1 \rangle \})^+ \quad (D.3)$$

Relacja B jest tranzytywnym domknięciem sumy relacji równoważności, a zatem jest także relacją równoważności. Rozpatrzmy dwa przypadki:

Przypadek 1

Jeśli

$$\langle (g, \alpha, \infty, \pi); Q_1, (g, \alpha, \infty, \pi); Q_2 \rangle \in D$$

to

$$B = D$$

czyli B jest silną EMB , taką że

$$\langle (g, \alpha, \infty, \pi); Q_1, (g, \alpha, \infty, \pi); Q_2 \rangle \in B \quad (D.4)$$

Przypadek 2

Niech $\langle (g, \alpha, \infty, \pi); Q_1, (g, \alpha, \infty, \pi); Q_2 \rangle \notin D$.

Zachodzi następujący związek

$$\begin{aligned} BEH|_B &= (BEH|_D \setminus \{[(g, \alpha, \infty, \pi); Q_1]_D, [(g, \alpha, \infty, \pi); Q_2]_D\}) \\ &\cup \{[(g, \alpha, \infty, \pi); Q_1]_D \cup [(g, \alpha, \infty, \pi); Q_2]_D\} \end{aligned} \quad (D.5)$$

Rozpatrzmy teraz dowolne elementy $\langle P_1, P_2 \rangle \in B$, $h \in A$, $\rho \in PR$ oraz $C \in BEH|_B$. Mamy do rozważenia trzy podprzypadki:

Podprzypadek 1

Jeśli $\langle P_1, P_2 \rangle \in D$ oraz $C \in BEH|_D \setminus \{[(g, \alpha, \infty, \pi); Q_1]_D, [(g, \alpha, \infty, \pi); Q_2]_D\}$, to

$$AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C) \quad (D.6)$$

Podprzypadek 2

Jeśli $\langle P_1, P_2 \rangle \in D$ oraz $C = \{[(g, \alpha, \infty, \pi); Q_1]_D \cup [(g, \alpha, \infty, \pi); Q_2]_D\}$,

to dla $i \in \{1, 2\}$ zachodzi

$$\begin{aligned} &AProb(P_i, h, \rho, C) \\ &= AProb(P_i, h, \rho, [(g, \alpha, \infty, \pi); Q_1]_D) \\ &\quad + AProb(P_i, h, \rho, [(g, \alpha, \infty, \pi); Q_2]_D) \end{aligned} \quad (D.7)$$

Ponieważ $AProb(P_1, h, \rho, C') = AProb(P_2, h, \rho, C')$ dla dowolnego C' , zatem

$$AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C) \quad (D.8)$$

Podprzypadek 3

Jeśli $\langle P_1, P_2 \rangle \in B \setminus D$, co oznacza, że $P_1 \in [(g, \alpha, \infty, \pi); Q_1]_D$ oraz $P_2 \in [(g, \alpha, \infty, \pi); Q_2]_D$, to dla $i \in \{1, 2\}$ mamy

$$AProb(P_i, h, \rho, C) = \begin{cases} 1 & \text{gdy } C = [Q_i]_B \\ 0 & \text{w przeciwnym przypadku} \end{cases} \quad (D.9)$$

Ponieważ $[Q_1]_B = [Q_2]_B$, zatem $AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C)$.

2. Niech $D \subseteq BEH \times BEH$ będzie silną relacją *EMB* taką, że $\langle Q_1, Q_2 \rangle \in D$. Relacja B , zdefiniowana jako

$$B = (D \cup \{\langle Q_1[p] Q, Q_2[p] Q \rangle, \langle Q_2[p] Q, Q_1[p] Q \rangle\})^+ \quad (D.10)$$

jest relacją równoważności. Dalszy dowód jest podobny jak w punkcie 1. Przeanalizujemy tylko przypadek 2., czyli gdy $\langle Q_1[p] Q, Q_2[p] Q \rangle \notin D$. Zachodzi następujący związek

$$\begin{aligned} BEH|_B &= (BEH|_D \setminus \{[Q_1[p] Q]_D, [Q_2[p] Q]_D\}) \\ &\cup \{[Q_1[p] Q]_D \cup [Q_2[p] Q]_D\} \end{aligned} \quad (D.11)$$

Rozpatrzmy $\langle P_1, P_2 \rangle \in B$, $p \in PR$, $h \in A$, $\rho \in PR$ oraz $C \in BEH|_B \subseteq L$.

Podprzypadki 1. oraz 2. są podobne do analogicznych podprzypadków w punkcie 1. Rozpatrzmy tylko podprzypadek 3., gdy $\langle P_1, P_2 \rangle \in B \setminus D$, co oznacza, że $P_1 \in [Q_1[p] Q]_D$ oraz $P_2 \in [Q_2[p] Q]_D$, wówczas dla $i \in \{1, 2\}$ mamy

$$\begin{aligned} AProb(P_i, h, \rho, C) &= AProb(Q_i[p] Q, h, \rho, C) \\ &= fact_{i1} * AProb(Q_i, h, \rho, C) + fact_{i2} * AProb(Q, h, \rho, C) \end{aligned} \quad (D.12)$$

gdzie $fact_{i1}$ oraz $fact_{i2}$ są zdefiniowane tak, jak w tabeli 10.1 dla wyrażeń behawioralnych $Q_i[p] Q$.

Zauważmy, że $fact_{11} = fact_{21}$ oraz $fact_{12} = fact_{22}$. Wartości współczynników $fact_{i1}$ zależą od wartości p i od prawdziwości formuł

$$M(F(Q_i)) \cap M(F(Q_i[p] Q)) = \emptyset \quad \text{dla } i = 1, 2 \quad (D.13)$$

Gdyby zachodziło

$$M(F(Q_1)) \cap M(F(Q_1[p] Q)) = \emptyset$$

oraz

$$M(F(Q_2)) \cap M(F(Q_2[p] Q)) \neq \emptyset \quad (D.14)$$

przeczyłoby to założeniu, że $Q_1 \sim_{EMB} Q_2$. Istotnie, oznaczałoby to, że istnieje zbiór indeksów Φ taki, że $\Phi \in M(F(Q_2)) \cap M(F(Q_2[p] Q))$ oraz $\Phi \notin M(F(Q_1)) \cap M(F(Q_1[p] Q))$, co dla $g = name(\Phi)$ dawałoby nierówność

$$AProb(Q_1, g, \rho, C) \neq AProb(Q_2, g, \rho, C) \quad (D.15)$$

Podobnie można pokazać równość współczynników $fact_{12}$ oraz $fact_{22}$.

Rozpatrzmy dwie dalsze możliwości ze względu na postać C .

Możliwość 1

Niech $C \in BEH|_D \setminus \{[Q_1[p] Q_2]_D, [Q_2[p] Q_1]_D\}$. Ponieważ $\langle Q_1, Q_2 \rangle \in D$, zachodzi

$$AProb(Q_1, h, \rho, C) = AProb(Q_2, h, \rho, C) \quad (D.16)$$

stąd

$$AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C) \quad (D.17)$$

Możliwość 2

Jeśli $C = [Q_1[p] Q]_D \cup [Q_2[p] Q]_D$, to dla $i \in \{1, 2\}$ zachodzi

$$\begin{aligned} AProb(Q_i, h, \rho, C) \\ = AProb(Q_i, h, \rho, [Q_1[p] Q]_D) + AProb(Q_i, h, \rho, [Q_2[p] Q]_D) \end{aligned} \quad (D.18)$$

Ponieważ $\langle Q_1, Q_2 \rangle \in D$, więc

$$AProb(Q_1, h, \rho, C) = AProb(Q_2, h, \rho, C) \quad (D.19)$$

i ostatecznie

$$AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C) \quad (D.20)$$

3. Pokażemy tylko, że dla dowolnych $Q \in BEH$, $R \in SeqG$, oraz $s, r \in PR$ zachodzą następujące związki:

$$Q_1 |[R]|_{s,r} Q \sim_{EMB} Q_2 |[R]|_{s,r} Q \quad (D.21)$$

Dowód dla $Q |[R]|_{s,r} Q_1 \sim_{EMB} Q |[R]|_{s,r} Q_2$ jest podobny. Rozpatrzmy relację

$$B = I \cup D \quad (D.22)$$

gdzie I jest relacją jednostkową na zbiorze BEH , a

$$D = \{\langle Q_1 |[R]|_{s,r} Q, Q_2 |[R]|_{s,r} Q \rangle \mid Q_1 \sim_{EMB} Q_2\} \quad (D.23)$$

Pokażemy, że relacja B jest silną EMB .

Łatwo sprawdzić, że B jest relacją równoważności. Zauważmy, że dla klasy równoważności C relacji B , dla danych R, r i s , albo każdy element klasy C ma operator złożenia równoległego $[R]|_{s,r}$ jako najbardziej zewnętrzny operator, albo żaden element tej klasy operatora tego nie ma.

Rozpatrzmy dowolne $\langle P_1, P_2 \rangle \in B$, $h \in A$, $\rho \in AP$, $C \in BEH|_B$, $R \in SeqG$ oraz $r, s \in PR$. Należy przeanalizować dwa przypadki:

Jeśli $\langle P_1, P_2 \rangle \in I$, to oczywiście $AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C)$.

Jeżeli natomiast $\langle P_1, P_2 \rangle \in D$, to istnieją takie $R_1, R_2 \in BEH$, że $R_1 \sim_{EMB} R_2$, $P_1 \in [R_1 |[R]|_{s,r} Q]_B$ oraz $P_2 \in [R_2 |[R]|_{s,r} Q]_B$. Rozpatrzmy teraz dwa podprzypadki:

Podprzypadek 1

Żadne z wyrażeń behawioralnych w klasie C nie ma $[[R]]_{s,r}$ jako najbardziej zewnętrznego operatora. W tej sytuacji

$$AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C) = 0 \quad (D.24)$$

Podprzypadek 2

Wszystkie wyrażenia behawioralne w klasie C mają $[[R]]_{s,r}$ jako najbardziej zewnętrzny operator, dlatego istnieją takie wyrażenia behawioralne E oraz F , że $C = \{E' \mid [[R]]_{s,r} F \mid E' \sim_{EMB} F\}$. W tej sytuacji, dla $P_i \equiv R_i \mid [[R]]_{s,r} Q$, gdy $i \in \{1, 2\}$

$$AProb(P_i, h, \rho, C) = AProb(R_i \mid [[R]]_{s,r} Q, h, \rho, C). \quad (D.25)$$

Jeśli $h \notin \text{Set}(R) \cup \{\text{exit}\}$, to

$$\begin{aligned} AProb(P_i, h, \rho, C) &= \text{fact}_{1_i} * AProb(R_i, h, \rho, [E']_{\sim_{EMB}}) \\ &+ \text{fact}_{2_i} * AProb(Q, h, \rho, \{F\}) \end{aligned} \quad (D.26)$$

gdzie fact_{1_i} oraz fact_{2_i} są zdefiniowane zgodnie z tabelą 2. dla wyrażeń behawioralnych $R_i \mid [[R]]_{s,r} Q$.

W sposób podobny jak w punkcie 2., możemy pokazać, że $\text{fact}_{1_i} = \text{fact}_{1_j}$ oraz $\text{fact}_{2_i} = \text{fact}_{2_j}$.

Ponieważ $R_1 \sim_{EMB} R_2$, dlatego

$$AProb(R_1, h, \rho, [E']_{\sim_{EMB}}) = AProb(R_2, h, \rho, [E']_{\sim_{EMB}}) \quad (D.27)$$

oraz

$$AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C) \quad (D.28)$$

Jeśli $h \in \text{Set}(R) \cup \{\text{exit}\}$, to rozważmy tranzycje, dla $i \in \{1, 2\}$, postaci

$$R_i \mid [[R]]_{s,r} Q \xrightarrow{(h, \Phi\Psi, \infty, \rho)} R'_i \mid [[R]]_{s,r} Q' \quad (D.29)$$

gdzie:

$$\begin{aligned} R_i \xrightarrow{(h, \Phi, \infty, \rho)} R'_i \text{ oraz } [R'_i]_{\sim_{EMB}} &= [R'_2]_{\sim_{EMB}} \\ Q \xrightarrow{(h, \Psi, \infty, \rho)} Q' \text{ oraz } R'_i \mid [[R]]_{s,r} Q' &\in C \end{aligned} \quad (D.30)$$

Wystarczy przeanalizować przypadek, gdy $C = \{E' \mid [[R]]_{s,r} Q' \mid E' \sim_{EMB} E\}$, ponieważ w przeciwnym razie, gdy $F \neq Q'$

$$AProb(R_i \mid [[R]]_{s,r} Q, h, \rho, C) = 0 \quad (D.31)$$

Niech

$$\begin{aligned} & MFS(R_i, Q, R, h, \rho) \\ &=_{\text{def}} \{ \Psi \in F(Q) \mid \exists \Phi \in F(R_i) \bullet \Phi \Psi \in M(F(R_i \parallel [R]_{s,r} Q)) \\ & \quad \wedge \text{name}(\Psi) = h \wedge \text{prior}(\Psi) = \rho \} \end{aligned} \quad (\text{D.32})$$

Łatwo zauważyć, że $MFS(R_1, Q, R, h, \rho) = MFS(R_2, Q, R, h, \rho)$, ponieważ w przeciwnym razie otrzymuje się sprzeczność, z założeniem, że $R_1 \sim_{EMB} R_2$, stąd

$$\begin{aligned} & AProb(R_i \parallel [R]_{s,r} Q, h, \rho, C) \\ &= \text{fact}_{12_i} * \sum \langle |p_\Phi * p_\Psi| < \Phi \Psi, p_\Phi * p_\Psi \rangle \\ & \in \text{norm}(PD(R_i, FA(Q))|_{\text{set}(R')} \otimes PD(Q, FA(R_i))|_{\text{set}(R')}) \\ & \wedge R_i \parallel [R]_{s,r} Q \longrightarrow (h, \Phi \Psi, \infty, \rho) \rightarrow R'_i \parallel [R]_{s,r} S \wedge \\ & R'_i \parallel [R]_{s,r} S \in C \} \end{aligned} \quad (\text{D.33})$$

gdzie: $\text{set}(R') = \text{set}(R) \cup \{\text{exit}\}$ oraz

$$PD(Q, FA(R_1))|_{\text{set}(R')} = PD(Q, FA(R_2))|_{\text{set}(R')} \quad (\text{D.34})$$

Dalej, niech $p_\Psi \in PD(Q, FA(R_1))|_{\text{set}(R')}$, wówczas

$$\begin{aligned} & AProb(R_i \parallel [R]_{s,r} Q, h, \rho, C) \\ &= \text{fact}_{12_i} * n_i * \sum_{\Psi \in MFS(R_i, Q, \Gamma, h, \rho)} p_\Psi \\ & * \sum \langle |p_\Phi| < \Phi, p_\Phi \rangle \in PD(R_i, FA(Q))|_{\text{set}(R')} \\ & \wedge R_i \longrightarrow (h, \Phi, \infty, \rho) \rightarrow R'_i \wedge R'_i \in [R']_{-EMB} \\ & \wedge R_i \parallel [R]_{s,r} Q \longrightarrow (h, \Phi \Psi, \infty, \rho) \rightarrow R'_i \parallel [R]_{s,r} S \} \\ &= \text{fact}_{12_i} * \sum_{\Psi \in MFS(R_i, Q, \Gamma, h, \rho)} p_\Psi * \sum AProb(R_i, h, \rho, [R'_i]) \end{aligned} \quad (\text{D.35})$$

gdzie n_i ($i = 1, 2$) jest współczynnikiem normalizującym.

Jak poprzednio można pokazać, że $\text{fact}_{12_1} = \text{fact}_{12_2}$.

Ponieważ

$$AProb(R_1, g, \rho, [R'_1]) = AProb(R_2, g, \rho, [R'_2]) \quad (\text{D.36})$$

oraz

$$MFS(R_1, Q, R, h, \rho) = MFS(R_2, Q, R, h, \rho) \quad (\text{D.37})$$

stąd jednakowe są współczynniki normalizujące n_1 oraz n_2 , dlatego

$$AProb(R_1 \parallel [R]_{s,r} Q, g, \rho, C) = AProb(R_2 \parallel [R]_{s,r} Q, g, \rho, C) \quad (\text{D.38})$$

4. Dowód można sprowadzić do szczególnego przypadku punktu 3. twierdzenia, ponieważ semantyka wyrażenia $Q_i [p > Q]$ jest równoważna semantyce wyrażenia $Q_i [\Box]_{0,p} Q$.
5. Dowód prowadzi się w sposób podobny, jak w punkcie 3.

Najpierw pokażemy, że dla dowolnego $Q \in BEH$ zachodzi następujący związek

$$Q_1 >> Q \sim_{EMB} Q_2 >> Q \quad (D.39)$$

Przeanalizujemy relację

$$B = I \cup D \quad (D.40)$$

gdzie

$$D = \{ \langle Q_1 >> Q, Q_2 >> Q \mid Q_1 \sim_{EMB} Q_2 \} \quad (D.41)$$

Relacja B – jak łatwo sprawdzić – jest relacją równoważności. Jeśli $\langle P_1, P_2 \rangle \in D$, to istnieją takie $R_1, R_2 \in BEH$, że $R_1 \sim_{EMB} R_2$, $P_1 \in [R_1 >> Q]_B$ oraz $P_2 \in [R_2 >> Q]_B$.

Rozpatrzmy tylko przypadek, gdy wszystkie wyrażenia w klasie C mają $>>$ jako najbardziej zewnętrzny operator. Istnieją zatem takie wyrażenia behawioralne E oraz F , że $C = \{ E' >> F \mid E' \sim_{EMB} F \}$.

Dla $P_i \equiv Q_i >> Q$, $i \in \{1, 2\}$, jeśli

$$Q_i >> Q \xrightarrow{(h, \alpha, \infty, \rho)} Q'_i >> Q \quad \text{dla } h \in A \quad (D.42)$$

to

$$\begin{aligned} AProb(P_i, h, \rho, C) \\ = AProb(P_i, h, \rho, [R_i >> Q]_B) = AProb(Q_i, h, \rho, [R_i]_{\sim_{EMB}}) \end{aligned} \quad (D.43)$$

Ponieważ $AProb(Q_1, h, \rho, [R_1]_{\sim_{EMB}}) = AProb(Q_2, h, \rho, [R_2]_{\sim_{EMB}})$, więc

$$AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C) \quad (D.44)$$

Dowód, że $Q >> Q_1 \sim_{EMB} Q >> Q_2$ dla dowolnego $Q \in BEH$ zachodzi w taki sam sposób, jak dla $Q_1 >> Q \sim_{EMB} Q_2 >> Q$.

6. Ponownie zdefiniujemy relację równoważności

$$B = I \cup D \quad (D.45)$$

gdzie

$$D = \{ \langle \text{hide } R \text{ in } Q_1, \text{hide } R \text{ in } Q_2 \rangle \mid Q_1 \sim_{EMB} Q_2 \} \quad (D.46)$$

Rozpatrujemy tylko przypadek, gdy $\langle P_1, P_2 \rangle \in D$, wówczas istnieją takie wyrażenia $R_1, R_2 \in BEH$ takie, że $R_1 \sim_{EMB} R_2$ oraz

$$P_1 \equiv \text{hide } R \text{ in } Q_1, P_2 \equiv \text{hide } R \text{ in } Q_2$$

Rozważmy tylko podprzypadek, gdy wszystkie wyrażenia w klasie C relacji B mają **hide** R **in** jako najbardziej zewnętrzny operator, dlatego istnieje takie wyrażenie Q , że $C = \{\text{hide } R \text{ in } Q' \mid Q' \sim_{EMB} Q\}$. Dla wyrażeń P_i , gdzie $i \in \{1, 2\}$, zachodzi

$$AProb(P_i, h, \rho, C) = AProb(Q_i, h, \rho, C) \quad \text{gdy } h \in L \setminus set(R) \quad (D.47)$$

oraz

$$AProb(P_i, h, \rho, C) = 0 \quad \text{gdy } h \in set(R) \quad (D.48)$$

Ponieważ $Q_1 \sim_{EMB} Q_2$, więc $AProb(P_1, h, \rho, C) = AProb(P_2, h, \rho, C)$.

Dowód twierdzenia 10.3

Dowód polega na pokazaniu, że dla dowolnej relacji $\sim_{EMB}$ (spełniającej twierdzenie 10.3) można skonstruować jej rozszerzenie, które będzie spełniać tezę twierdzenia. W tym celu wystarczy pokazać, że relacja $B' = B \cup B^{-1}$, gdzie

$$\begin{aligned} B &= \{ \langle Q_1, Q_2 \rangle \mid Q_1 \equiv Q[\langle S[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n] \rangle] \\ &\quad \wedge Q_2 \equiv Q[\langle S[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n] \rangle] \\ &\quad \wedge S[g_1, \dots, g_n] \text{ jest co najwyżej jedynym wolnym procesem względem} \\ &\quad \text{danego zbioru definicji procesów} \} \end{aligned} \quad (D.49)$$

jest silną rozszerzoną relacją bisymulacji markowskiej względem $\sim_{EMB}$.

Relacja $B' = B \cup B^{-1}$ jest relacją równoważności. B' jest symetryczna z definicji i jest zwrotna, ponieważ jeśli Q nie zawiera wolnych zmiennych procesowych lub $P_1 \equiv P_2$, to $Q_1 \equiv Q_2 \equiv Q$. Relacja jest również przechodnia, gdyż istnieje co najwyżej jedna zmienna procesowa wolna w S .

Zauważmy, że złożenie $\sim_{EMB} \circ B' \circ \sim_{EMB}$ relacji równoważności $\sim_{EMB}$ oraz relacji równoważności B' jest również relacją równoważności.

Aby pokazać tezę, wystarczy dowieść, że B' jest silną EMB względem relacji $\sim_{EMB}$, to znaczy że dla danych $g \in A$, $\pi \in AP$ oraz $C \in BEH|_{\sim_{EMB} \circ B' \circ \sim_{EMB}}$

$$AProb(Q_1, g, \pi, C) = AProb(Q_2, g, \pi, C) \quad (D.50)$$

oraz

$$ARate(Q_1, g, C) = ARate(Q_2, g, C) \quad (D.51)$$

Będziemy rozpatrywać klasy równoważności zbioru ilorazowego $BEH|_{\sim_{EMB} \circ B' \circ \sim_{EMB}}$.

Dla danych $g \in A$, $\pi \in AP$ oraz $C \in BEH|_{\sim_{EMB} \circ B' \circ \sim_{EMB}}$, pokażemy przez indukcję strukturalną na Q tylko dla (10.37a). Dowód dla (10.37b) przebiega w podobny sposób.

Dowód pokażemy tylko dla najbardziej reprezentatywnych konstrukcji: prefiksowania akcją, wyboru i złożenia równoległego.

1. Dla $Q \equiv (g, \alpha, \infty, \pi)$; Q' oraz $i = 1, 2$, mamy

$$Q_i \equiv (g, \alpha, \infty, \pi); Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle]$$

Ponieważ

$$\langle Q'[\langle S[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n] \rangle], Q'[\langle S[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n] \rangle] \in B \quad (D.52)$$

więc

$$\begin{aligned} C &= [Q'[\langle S[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n] \rangle] \sim_{EMB} B' \circ \sim_{EMB} \\ &= [Q'[\langle S[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n] \rangle] \sim_{EMB} B' \circ \sim_{EMB} \end{aligned} \quad (D.53)$$

Dla tego przypadku

$$AProb(Q_1, g, \pi, C) = AProb(Q_2, g, \pi, C) = 1 \quad (D.54)$$

2. Dla $Q \equiv Q' [p] Q''$ oraz $i = 1, 2$, mamy

$$\begin{aligned} Q_i &\equiv Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle] [p] \\ &Q''[S[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle]] \end{aligned} \quad (D.55)$$

Hipoteza indukcyjna stanowi, że

$$\begin{aligned} AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n] \rangle], g, \pi, C) \\ = AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n] \rangle], g, \pi, C) \end{aligned} \quad (D.56)$$

oraz

$$\begin{aligned} AProb(Q''[\langle S[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n] \rangle], g, \pi, C) \\ = AProb(Q''[\langle S[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n] \rangle], g, \pi, C) \end{aligned} \quad (D.57)$$

Ponieważ

$$\begin{aligned} AProb(Q_i, g, \pi, C) \\ = (1 - p) * AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, C) \\ + p * AProb(Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, C) \end{aligned} \quad (D.58)$$

zatem

$$AProb(Q_1, g, \pi, C) = AProb(Q_2, g, \pi, C) \quad (D.59)$$

3. Dla $Q \equiv Q' [R]_{s,r} Q''$ oraz $i = 1, 2$, mamy

$$\begin{aligned} Q_i &\equiv Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle] \\ &[R]_{s,r} Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle]. \end{aligned} \quad (D.60)$$

Rozważmy dwa przypadki dla $g \in A$.

Przypadek 1

Jeśli $g \notin \text{set}(R)$, to albo $Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle]$ przechodzi do S'_i , gdzie

$$S'_i \in C'_i = [S'_i \mid [R]]_{s,r} Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle] \sim_{EMB \circ B' \circ \sim_{EMB}} \quad (D.61)$$

albo $Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle]$ przechodzi do S''_i , gdzie

$$S''_i \in C''_i = [\langle S[g_1, \dots, g_n] ::= Q'[P_i[g_1, \dots, g_n]] \rangle \mid [R]]_{s,r} S''_i \sim_{EMB \circ B' \circ \sim_{EMB}} \quad (D.62)$$

Dla danego $C \in BEH|_{\sim_{EMB \circ B' \circ \sim_{EMB}}}$ oraz C'_i (podobnie dla C''_i) albo $C'_i \subseteq C$, albo $C'_i \cap C = \emptyset$, zatem odpowiednio albo

$$\begin{aligned} AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, C) \\ = AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, S'_i) \end{aligned} \quad (D.63)$$

albo

$$AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, C) = 0 \quad (D.64)$$

Z hipotezy indukcyjnej

$$\begin{aligned} AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n] \rangle], g, \pi, S) \\ = AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n] \rangle], g, \pi, S) \end{aligned} \quad (D.65)$$

oraz

$$\begin{aligned} AProb(Q''[\langle S[g_1, \dots, g_n] ::= \langle P_1[g_1, \dots, g_n] \rangle \rangle], g, \pi, S) \\ = AProb(Q''[\langle P_2[g_1, \dots, g_n] / S[g_1, \dots, g_n] \rangle], g, \pi, S) \end{aligned} \quad (D.66)$$

dla dowolnego $S \in BEH|_{\sim_{EMB \circ B' \circ \sim_{EMB}}}$, oraz z formuły

$$\begin{aligned} AProb(Q_i, g, \pi, S) \\ = (1-r) * (1-s) * AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, C) \\ + r * (1-s) * AProb(Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, C) \end{aligned} \quad (D.67)$$

wynika, że $AProb(Q_1, g, \pi, S) = AProb(Q_2, g, \pi, S)$.

Przypadek 2

Jeśli $g \in \text{set}(R)$, to $Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle]$ przechodzi do

$S' \in C' \in BEH|_{\sim_{EMB \circ B' \circ \sim_{EMB}}}$ z prawdopodobieństwem

$$AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, S') \quad (D.68)$$

oraz $Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle]$ przechodzi do $S'' \in C'' \in BEH|_{-EMB \circ B' \circ -EMB}$ z prawdopodobieństwem

$$AProb(Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, S'') \quad (D.69)$$

zatem

$$Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle] \llbracket [R] \rrbracket_{s,r} Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle] \quad (D.70)$$

przechodzi do $S \in BEH|_{-EMB \circ B' \circ -EMB}$ z prawdopodobieństwem

$$\begin{aligned} & AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle] \llbracket [R] \rrbracket_{s,r} Q''[\langle S[g_1, \dots, g_n] \\ & ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, S) \\ & = s * AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, S) * \\ & AProb(Q''[\langle S[g_1, \dots, g_n] ::= P_i[g_1, \dots, g_n] \rangle], g, \pi, S) \end{aligned} \quad (D.71)$$

Z hipotezy indukcyjnej:

$$\begin{aligned} & AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n] \rangle], g, \pi, S) \\ & = AProb(Q'[\langle S[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n] \rangle], g, \pi, S) \end{aligned} \quad (D.72)$$

oraz

$$\begin{aligned} & AProb(Q''[\langle S[g_1, \dots, g_n] ::= P_1[g_1, \dots, g_n] \rangle], g, \pi, S) \\ & = AProb(Q''[\langle S[g_1, \dots, g_n] ::= P_2[g_1, \dots, g_n] \rangle], g, \pi, S) \end{aligned} \quad (D.73)$$

wynika, że $AProb(Q_1, g, \pi, S) = AProb(Q_2, g, \pi, S)$.

Dodatek 2

Definicja funkcji *SM_Trans*

W przedstawionej definicji funkcji transformacyjnej *SM_Trans* przyjmuje się, że nazwy funkcji pisane kursywą oznaczają funkcje transformacyjne, natomiast funkcje pisane czcionką prostą oznaczają tekst w języku LOTOS.

Główna część definicji funkcji *SM_Trans* ma postać

SM_Trans(*ms*) =

specification *S*[*list*(*ActionNames* $\cup$ *EventArcN*)] : **noexit**

type *ArcNames* **is**

sorts *arcN*

opns

$a_i : \rightarrow \text{arcN}$ (* dla $a_i \in \text{ArcN}$ takich, że $\text{typeA}(a_i) = \text{EV-LAB}$ *)

$\text{trigger}(tm_i) : \rightarrow \text{arcN}$ (* dla $tm_i \in \text{ArcN}$ takich, że $\text{typeA}(tm_i) = \text{TO-LAB}$ *)

default : $\rightarrow \text{arcN}$ (* wskazuje stan początkowy *)

endtype

type *BoxNames* **is**

sorts *boxN*

opns

$b_i : \rightarrow \text{boxN}$ (* dla $b_i \in \text{BoxN}$ *)

endtype

behaviour

hide *putS*, *getS*, *syn*, *list*($\{\text{trigger}(tm_i)\}$) **in**

(* dla $tm_i \in \text{ArcN}$ takich, że $\text{typeA}(tm_i) = \text{TO-LAB}$ *)

(

StateHandler[*putS*, *getS*]

$[[\text{putS}, \text{getS}]]$

(*Complete*[*ExtArcN*](**default**) $\gg$

B_Trans(*r*) (* *r* jest nazwą stanu początkowego *)

)

$[[\text{syn}, \text{list}(\text{ArcN})]]$

Synchroniser[*syn*, *list*(*ArcN*)](*TrueList*)

)

where

(* definicje trzech głównych procesów składowych *)

process *Synchroniser*[*syn*, *list*(*ArcN*)](*BoxList*): **noexit** :=

```

LetExpression(r)          (*r jest nazwą stanu początkowego *)
(
   $\sum_{b \in \text{BoxN} \bullet \text{typeB}(b) = \text{PRIM} \text{ or } \text{typeB}(b) = \text{FIN}}$  (
    syn !boxN(b); Synchroniser[syn, list(ArcN)](ChangeBoxList(b)) )
[]
   $\sum_{a \in \text{ArcN} \bullet b = \text{source}(a)}$  ( [b]  $\rightarrow$ 
    trigger(a); Synchroniser[syn, list(ArcN)](ChangeBoxList(target(a))) )
)
endproc (* Synchroniser *)

process StateHandler[putS, getS] : noexit :=
   $\prod_{b \in \text{BoxN} \bullet \text{typeB}(b) = \text{XOR}}$  b[putS, getS](default(b))
where
  process bi[putS, getS](s: boxN): noexit :=
    (* rodzina procesów dla  $b_i \in \text{BoxN}$  takich, że  $\text{typeB}(b_i) = \text{XOR}$  *)
    getS !bi !s; bi[putS, getS](s)
    []  $\sum_{a \in \text{childB}(b_i)}$  putS !bi !a; bi[putS, getS](a)
  endproc (* bi *)
endproc (* StateHandler *)

process Complete[ExtArcN](s: arcN): exit :=
  ([s = default]  $\rightarrow$ 
    Entry_r[ExtArcN];
    SetDefaultBox(r);          (*r jest nazwą stanu początkowego *)
    exit )
  []  $\sum_{a \in \text{ArcN}}$  ( ([s = a]  $\rightarrow$ 
    (* przyjęcie zdarzenia *)
    (* przejście do stanu niestabilnego *)
    Exit_source(a)[ExtArcN]; (* wykonanie akcji wejściowej *)
    Entry_target(a)[ExtArcN]; (* wykonanie akcji wyjściowej *)
    SetDefaultBox(target(a)); (* wyznaczenie nowego stanu aktywnego *)
    SetStableBox(target(a));  (* odblokowanie odbioru zdarzeń *)
    (* przejście do stanu stabilnego *)

    exit )
  )
endproc (* Complete *)
..... (* definicje procesów składowych są podane niżej *)
endspec

Definicje procesów składowych występujących w głównych procesach składowych są
podzielone na kilka grup.

Definicje procesów wykonujących akcje wejściowe w danym stanie bi:

process Entry_bi[ExtArcN]: exit :=      (* dla  $b_i \in \text{BoxN}$  takich, że  $\text{typeB}(b_i) = \text{PRIM}$  *)
  entry(bi) exit
  (* funkcja entry zwraca ciąg akcji separowanych i zakończonych średnikiem *)
endproc

```



```

process Entry_ $b_i$ [ExtArcN]: exit :=      (* dla  $b_i \in \text{Box}N$  takich, że  $\text{type}B(b_i)=\text{FIN}$  *)
  exit
endproc

process Entry_ $b_i$ [ExtArcN]: exit :=      (* dla  $b_i \in \text{Box}N$  takich, że  $\text{type}B(b_i)=\text{XOR}$  *)
  entry( $b_i$ ) Entry_default( $b_i$ )
  (* funkcja entry zwraca ciąg akcji separowanych i zakończonych średnikiem *)
endproc

process Entry_ $b_i$ [ExtArcN]: exit :=      (* dla  $b_i \in \text{Box}N$  takich, że  $\text{type}B(b_i)=\text{AND}$  *)
  entry( $b_i$ )
  ( $\prod_{a \in \text{child}B(b_i)} \text{Entry}_a[\text{ExtArcN}]$ )
  (* funkcja entry zwraca ciąg akcji separowanych i zakończonych średnikiem *)
endproc

```

Definicje procesów wykonujących akcje wyjściowe w danym stanie b_i .

```

process Exit_ $b_i$ [ExtArcN]: exit :=      (* dla  $b_i \in \text{Box}N$  takich, że  $\text{type}B(b_i)=\text{PRIM}$  *)
  exit( $b_i$ ) exit
  (* funkcja exit zwraca ciąg akcji separowanych i zakończonych średnikiem *)
endproc

process Exit_ $b_i$ [ExtArcN]: exit :=      (* dla  $b_i \in \text{Box}N$  takich, że  $\text{type}B(b_i)=\text{FIN}$  *)
  exit
endproc

process Exit_ $b_i$ [ExtArcN]: exit :=      (* dla  $b_i \in \text{Box}N$  takich, że  $\text{type}B(b_i)=\text{XOR}$  *)
  (* oraz  $b_i \neq r$  *)
  getS ! $b_i$  ? $s$  : boxN;
  ( $\sum_{a \in \text{child}B(b_i)} ([s = a] \rightarrow \text{Exit}_a[\text{ExtArcN}])$ ); exit( $b_i$ ) exit
  (* funkcja exit zwraca ciąg akcji separowanych i zakończonych średnikiem *)
endproc

process Exit_ $b_i$ [ExtArcN]: exit :=      (* dla  $b_i \in \text{Box}N$  takich, że  $\text{type}B(b_i)=\text{AND}$  *)
  ( $\prod_{a \in \text{child}B(b_i)} \text{Exit}_a[\text{ExtArcN}]$ ); exit( $b_i$ ) exit
  (* funkcja exit zwraca ciąg akcji separowanych i zakończonych średnikiem *)
endproc

```

Do następnej grupy należy definicja procesu początkowego, reprezentującego stan r , oraz definicje procesów generowanych przez funkcje $\text{proc}D$ oraz $\text{proc}D^*$, które były zdefiniowane w rozdziale 11.

Funkcje pomocnicze, które wystąpiły w definicji funkcji transformacji, są określone następująco:

$$\text{SetDefaultBox}(b) = \begin{cases}
 \text{putS!}b'!b; & \text{gdy } \text{typeB}(b) = \text{PRIM} \text{ lub } \text{typeB}(b) = \text{FIN} \\
 & \text{oraz } \exists b' \in \text{BoxN} \bullet b \in \text{childB}(b') \text{ oraz} \\
 & \text{typeB}(b') = \text{XOR} \\
 \\
 \text{putS!}b'!b; & \text{gdy } \text{typeB}(b) = \text{XOR} \text{ oraz} \\
 \text{SetDefaultBox}(\text{default}(b)) & \exists b' \in \text{BoxN} \bullet b \in \text{childB}(b') \text{ oraz} \\
 & \text{typeB}(b') = \text{XOR} \\
 \\
 \text{SetDefaultBox}(\text{default}(b)) & \text{gdy } \text{typeB}(b) = \text{XOR} \text{ oraz} \\
 & \neg \exists b' \in \text{BoxN} \bullet b \in \text{childB}(b') \text{ oraz} \\
 & \text{typeB}(b') = \text{XOR} \\
 \\
 \text{putS!}b'!b; & \text{gdy } \text{typeB}(b) = \text{AND} \text{ oraz } b_i \in \text{childB}(b) \\
 \text{SetDefaultBox}(b_1) \dots & \text{oraz } \exists b' \in \text{BoxN} \bullet b \in \text{childB}(b') \text{ oraz} \\
 \quad \text{SetDefaultBox}(b_k) & \text{typeB}(b') = \text{XOR} \\
 \\
 \text{SetDefaultBox}(b_1) \dots & \text{gdy } \text{typeB}(b) = \text{AND} \text{ oraz } b_i \in \text{childB}(b) \\
 \quad \text{SetDefaultBox}(b_k) & \text{oraz } \neg \exists b' \in \text{BoxN} \bullet b \in \text{childB}(b') \text{ oraz} \\
 & \text{typeB}(b') = \text{XOR}
 \end{cases}$$

$$\text{SetStableBox}(b) = \begin{cases}
 \text{syn !}b; & \text{gdy } \text{typeB}(b) = \text{PRIM} \\
 & \text{lub } \text{typeB}(b) = \text{FIN} \\
 \text{SetStableBox}(\text{default}(b)) & \text{gdy } \text{typeB}(b) = \text{XOR} \\
 \text{SetStableBox}(b_1) \dots & \text{gdy } \text{typeB}(b) = \text{AND} \\
 \quad \text{SetStableBox}(b_k) & \text{oraz } b_i \in \text{childB}(b)
 \end{cases}$$

$$\text{LetExpression}(b) = \begin{cases}
 \text{LetExpression}(b_1) \dots \text{LetExpression}(b_k) \\
 \text{Let } b : \text{bool} = (b_1 \text{ and } \dots \text{ and } b_k) \text{ in} & \text{gdy } \text{typeB}(b) = \text{XOR} \text{ lub} \\
 & \text{typeB}(b) = \text{AND} \text{ oraz} \\
 & b_i \in \text{childB}(b) \\
 \text{puste} & \text{w przypadku przeciwnym}
 \end{cases}$$

$TrueList = true^1, \dots, true^k$ $k = card(\{b \in BoxN \mid typeB(b)=PRIM \text{ lub } typeB(b)=FIN\})$

$BoxList = b_1: bool, \dots, b_k: bool$ gdzie $b_1, \dots, b_k = list(\{b \in BoxN \mid typeB(b)=PRIM \text{ lub } typeB(b)=FIN\})$

$ChangeBoxList(b) = if-not(b, b_1), \dots, if-not(b, b_k)$ gdzie $b_1, \dots, b_k = list(\{b \in BoxN \mid typeB(b)=PRIM \text{ lub } typeB(b)=FIN\})$

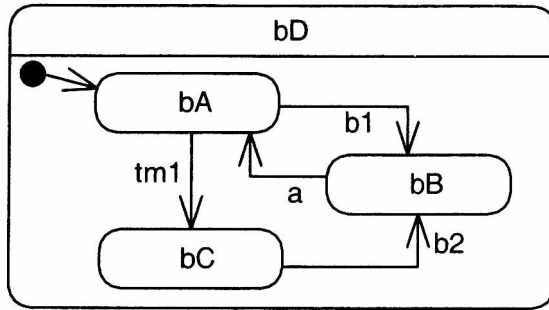
$if - not(b_1, b_2) = \begin{cases} b_2 & \text{gd}y \ b_1 \neq b_2 \text{ oraz } (b_2 \notin childB^*(b_1) \\ & \text{lub } (b_2 \in childB(b_1) \text{ oraz } defaultB(b_1) \neq b_2)) \\ not(b_2) & \text{gd}y \ b_1 = b_2 \\ if-not(default(b_1), b_2) & \text{gd}y \ typeB(b_1) = XOR \text{ oraz } \\ & b_2 \in childB^*(b_1) \\ if-not(b_3, b_2) & \text{gd}y \ typeB(b_1) = AND \text{ oraz } \\ & b_3 \in childB(b_1) \\ & \text{and } b_2 \in childB^*(b_3) \end{cases}$

Funkcje *SetDefaultBox* oraz *SetStableBox* słuŹą do generowania tekstów stanowiących fragmenty wyraŹeń behawioralnych w obrębie definicji procesu *Complete*.

Pozostałe funkcje są związane z definicją procesu *Synchroniser*. Ich parametry formalne i aktualne są określone odpowiednio przez funkcje *BoxList* oraz *TrueList*. Liczba parametrów formalnych jest łączną liczbą stanów typu *PRIM* i *FIN*. Zawarte w wyrażeniu *LetExpression* zmienne $b_1, \dots, b_k$ rodzaju *bool* reprezentują podstany odpowiednich stanów. Wartość *true* zmiennej b_i oznacza, że odpowiedni stan jest stabilny, a w przypadku przeciwnym, że jest niestabilny. Stan złożony jest stabilny tylko wtedy, gdy stabilne są wszystkie jego podstany. Wartości tych zmiennych są obliczane na podstawie parametrów aktualnych procesu *Synchroniser*.

Specyfikacja przykładowego diagramu stanów

Przedstawiana dalej specyfikacja w LOTOSie jest wynikiem zastosowania funkcji *SM_Trans* do diagramu stanów przedstawionego na rysunku D2.1.



Rys. D2.1. Przykładowy diagram stanów

Diagram składa się ze stanów bA, bB, bC, bD. Stany bA, bB, bC są stanami prostymi, a stan bD jest złożony sekwencyjnie. Pomiędzy stanami są przejścia, z których jedno, oznaczone tm1, jest przejściem czasowym.

Zakładamy, że każdy stan prosty ma akcje wejściowe i akcje wyjściowe – są one reprezentowane przez odpowiednie bramki. Przyjęto konwencję: bramka *ia* odpowiada akcji wejściowej do stanu A – pierwsza litera *i* oznacza wejście, druga *a* odnosi się do stanu A; bramka *ea* odpowiada akcji wyjściowej do stanu A – pierwsza litera *e* oznacza wyjście, druga – jak poprzednio – odnosi się do stanu.

Specification S[ia,ea,ib,eb,ic,ec,b1,b2,a]: **noexit**
library Boolean **endlib**

```

type ArcNames is
sorts arcN
opns
    b1,b2,a :-> arcN
    tm1 :-> arcN
    default :-> arcN
endtype

```

```

type BoxNames is
sorts boxN
opns bA, bB, bC, bD:->boxN
endtype

```

behaviour

```

hide putS, getS, syn, tm1 in
(* przejścia czasowe 'time-out' są ukryte *)
(
    StateHandler[putS, getS]

```

```

| [putS, getS] |
(
  Complete[ putS, getS, syn, ia, ea, ib, eb, ic,
    ec, b1, b2, a, tm1] (default) >>
  ( bA[putS, getS, syn, ia, ea, ib, eb, ic, ec, b1, b2, a, tm1]
    [> bD[putS, getS, syn, ia, ea, ib, eb, ic, ec, b1, b2, a, tm1]
  )
)
| [syn, b1, b2, a, tm1] |
  Synchroniser[syn, b1, b2, a, tm1] (true, true, true)
)

```

where

```

process Synchroniser[syn, b1, b2, a, tm1] (A, B, C: bool):
noexit :=
let bD: bool = (A and B and C) in
  (
    syn !bA; Synchroniser[syn, b1, b2, a, tm1] (not(A), B, C)
  []
  syn !bB; Synchroniser[syn, b1, b2, a, tm1] (A, not(B), C)
  []
  syn !bC; Synchroniser[syn, b1, b2, a, tm1] (A, B, not(C))
  []
  [A]->b1; Synchroniser[syn, b1, b2, a, tm1] (A, not(B), C)
  []
  [A]->tm1; Synchroniser[syn, b1, b2, a, tm1] (A, B, not(C))
  []
  [B]->a; Synchroniser[syn, b1, b2, a, tm1] (not(A), B, C)
  []
  [C]->b2; Synchroniser[syn, b1, b2, a, tm1] (A, not(B), C)
  )
endproc (* Synchroniser *)

```

```

process StateHandler[putS, getS]: noexit :=
  bD[putS, getS] (bA)
where

```

```

  process bD[putS, getS] (s: boxN): noexit :=
    getS !bD !s; bD[putS, getS] (s)
  []
    putS !bD !bA; bD[putS, getS] (bA)
  []
    putS !bD !bB; bD[putS, getS] (bB)
  []
    putS !bD !bC; bD[putS, getS] (bC)
endproc (* bD *)
endproc (* StateHandler *)

```

```

process Complete[putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml](s: arcN) : exit :=

( [s=default] ->
EntrybD[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> putS !bD !bA; exit
)
[]
( [s=b1] ->
ExitbA[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> EntrybB[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> putS !bD !bB; syn !bB; exit
)
[]
( [s=b2] ->
ExitbC[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> EntrybB[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> putS !bD !bB; syn !bB; exit
)
[]
( [s=a] ->
ExitbB[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> EntrybA[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> putS !bD !bA; syn !bA; exit
)
[]
( [s=tml] ->
ExitbA[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> EntrybC[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
>> putS !bD !bC; syn !bC; exit )
endproc

(* ----- akcje wejścia 'entry' i wyjścia 'exit' -----
--*)

process EntrybA[ putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml]: exit :=
    ia; exit
endproc

process ExitbA[putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml]: exit :=
    ea; exit
endproc

process EntrybB[ putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml]: exit :=
    ib; exit
endproc

```

```

process ExitbB[putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml]: exit :=
    eb; exit
endproc

```

```

process EntrybC[ putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml]: exit :=
    ic; exit
endproc

```

```

process ExitbC[putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml]: exit :=
    ec; exit
endproc

```

```

process EntrybD[ putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml]: exit :=
    EntrybA[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
endproc

```

```

(* ----- definicje procesów ----- *)
process bA[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]: noexit
:=
    ( (* DoTrans is empty *)
    tml; exit (* Time Out *)
    [> (* Disabling Part *)
    ( b1; exit)
    >> Complete[      putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml](b1)
    >> bB[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
    )
    ) (* After Time-Out *)
    >> Complete[      putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml](tml)
    >> bC[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
endproc

```

```

process bB[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]: noexit
:=
    (      stop (* przeterminowanie 'time out' *)
    [>    (* część deaktywująca 'disabling' *)
    (      (a; exit)
    >> Complete[      putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tml](a)
    >> bA[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tml]
    )
    ) (* puste po zdarzeniu przeterminowania *)
endproc

```

```
process bC[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tm1]: noexit
:=
( stop (* Time Out *)
  [> (* cześć deaktywująca 'disabling' *)
  ( b2; exit)
>> Complete[      putS,getS,syn,ia,ea,ib,eb,ic,
ec,b1,b2,a,tm1](b2)
      >> bB[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tm1]
)
) (*puste po zdarzeniu przeterminowania *)
endproc

process bD[putS,getS,syn,ia,ea,ib,eb,ic,ec,b1,b2,a,tm1]: noexit
:=
  stop (* root *)
endproc
endspec
```


Literatura

- Ajmone Marsan M., Balbo G., Conte G., 1984, *A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems*, ACM Trans. Computer Systems, Vol. 2, 93–122.
- Ajmone Marsan M., Balbo G., Conte G., Donatelli S., Franceschinis G., 1994, *A LOTOS extension for the performance analysis of distributed systems*, IEEE/ACM Trans. Networking, Vol. 2 (2), 151–164.
- Apt K., Olderog E.-R., 1991, *Verification of Sequential and Concurrent Programs*, Springer.
- Apvrille L., Courtiat J.-P., Lohr C., de Saqui-Sannes P., 2004, *TURTLE: A Real-Time UML profile supported by a formal validation toolkit*, IEEE Transactions on software Engineering, Vol. 30, No. 7, 473–486.
- Babczyński T., Huzar Z., Magott J., 1999, *Algebraic semantics for Markovian statecharts*, in: Proc. 15th Annual UK Performance Engineering Workshop, University of Bristol, July 22–23, 105–120.
- Babczyński T., Huzar Z., Magott J., 2000, *Derivation of Markov processes from the UML statecharts*, Archiwum Informatyki Teoretycznej i Stosowanej, t. 12, z. 3, 141–171.
- Babczyński T., Huzar Z., Magott J., 2001, *Performance modelling and evaluation of continuous media transport protocol using statecharts*, Archiwum Informatyki Teoretycznej i Stosowanej, t. 13, z. 4, 329–342.
- Babczyński T., Huzar Z., Magott J., 2003, *Performance evaluation in analysis of ATM network signalling*, in: Proc. IASTED Int. Conference On Modelling, Identification and Control, Innsbruck, 341–346.
- Babich F., Deonto L., 2002, *Formal Methods for Specification and Analysis of Communication Protocols*, IEEE Communications Surveys, The Electronic Magazine of Original Peer-Reviewed Survey articles, Vol. 3, No. 1. <http://www.comsoc.org/pubs/surveys>.
- Baeten J.C.M., 2004, *A brief history of process algebra*, Rapport CRS 0402, Vakgroep Informatica, Technische Universiteit Eindhoven.
- Baeten J.C.M., Bergstra S.A., Smolka S.A., 1995, *Axiomatizing probabilistic processes: ACP with generative probabilities*, Information and Computation, 121, 234–255.
- Baeten J.C.M., Bergstra S.A., 1991, *Real Time Process Algebra*, Formal Aspects of Computing, 3 (2), 183–235.
- Baeten J.C.M., Middelburg C.A., 2002, *Process Algebra with Timing*, Springer.
- Behforooz A., Hudson F.J., 1996, *Software Engineering Fundamentals*, Oxford University Press.
- Berard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen P., and McKenzie P., 2001, *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer.

- Belina F., Hogrefe D., 1989, *The CCITT – Specification and Description Language SDL*, Computer Networks and ISDN Systems, Vol. 16, 311–341.
- Benveniste A., Caspi P., Edwards S., Halbwachs N., Le Guernic P., De Simone R., 2003, *The synchronous languages 12 years later*, Proceedings of the IEEE, Vol. 91, No. 1, 66–83.
- Bergstra J.A., Klop J.W., 1985, *Algebra for Communicating Processes with Abstraction*, Theoretical Computer Science, Vol. 37, 77–121.
- Bernardo M., Gorrieri R., 1998, *A Tutorial on EMPA: A Theory of Concurrent Processes with Non-determinism, Priorities, Probabilities and Time*, Theoretical Computer Science, 201, 1–54.
- Bilski E., Dubielewicz I., 1991, *Model odniesienia dla współdziałania systemów otwartych*, t. 1, Wydawnictwo Politechniki Wrocławskiej.
- Bilski E., Dubielewicz I., 1993, *Model odniesienia dla współdziałania systemów otwartych*, t. 2, Wydawnictwo Politechniki Wrocławskiej.
- Bolognesi T., Smolka S.A., 1987, *Fundamental results for the verification of observational equivalence: a survey*, in: *Protocol Specification and Verification, VII*, Rudin H., West C.H. (eds.), Elsevier Science Publishers, IFIP, 165–179.
- Bolognesi T., Lucidi F., Trigila S., 1990, *From Timed Petri Nets to Timed LOTOS*, in: *Protocol Specification, Testing and Verification, X*, Logrippo L., Probert R., Ural H. (eds.), North-Holland, 395–408.
- Bolognesi T., Lucidi F., 1992, *LOTOS-like process algebra with urgent and timed interactions*, in: *Formal Description Techniques, IV*, Parker K., Rose G. (eds.), North-Holland, 249–264.
- Bolognesi T., Lucidi F., Trigilla S., 1995, *Converging towards timed LOTOS standard*, Computer Standards Interfaces, 16, 87–118.
- Bolognesi T., Brinksma E., 1989, *Introduction to ISO Specification Language LOTOS*, in: Eijk P.H.J. van, Vissers C.A., Diaz M. (eds.), *The formal Description Technique LOTOS*, North-Holland, 23–73.
- Booch G., Rumbaugh J., Jacobson I., 1999, *The Unified Modelling Language User Guide*, Addison-Wesley.
- Bortnik E., Trčka N., Wijs A.J., Luttik S.P., van de Mortel-Fronczak J.M., Baeten J.C.M., Fokkink W.J., Rooda J.E., 2004, *Analyzing a Model of a Turntable System using Spin, CADP and Uppaal*, Technical Report, Department of Software Engineering, GB Amsterdam.
- Börger E., Stärk R., 2003, *Abstract State Machines. A Method for High-Level System Design and Analysis*, Springer.
- Bravetti M., Bernardo M., 2000, *Compositional Asymmetric Cooperations for process algebras with probabilities, priorities, and time*, Electronic Notes for Theoretical Computer Science, Vol. 39, No. 3, 34.
- Brinksma E., Scollo G., 1986, *Formal notions of implementation and conformance in LOTOS*, Technical Report, Computer Science Department, Twente University.

- Brinksma E. (ed.), 1988, *Information processing systems – Open systems interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*, International Standard ISO 8807.
- Brinksma E., 1989, *Constraint-oriented specification in a constructive formal description technique*, in: de Bakker J.W., de Roever W.-P., Rozenberg G. (eds.), *Stepwise Refinement of Distributed Systems, Models Formalism, Correctness*, LNCS 430, Springer, 130–152.
- Brinksma E., Katoen J.-P., Latella D., 1995, *A stochastic causality-based process algebra*, The Computer Journal, Vol. 38, 552–565.
- Bronsztejn I.N., Siemiendiajew K.A., Musiol G., Mühlig H., 2004, *Nowoczesne kompendium matematyki*, PWN.
- Budkowski S., 1992, *ESTELLE Development Toolset (EDT)*, Computer Networks and ISDN Systems, 23 (5), North-Holland.
- Budkowski S., Dembiński P., 1987, *An introduction to ESTELLE: a specification language for distributed systems*, Computer Networks and ISDN Systems, 14 (1), North-Holland, 3–23.
- Budkowski S., Dembiński P., 1989, *The specification language ESTELLE*, in: Diaz M. et al. (eds.), *The Formal Description Technique ESTELLE*, North-Holland.
- Burstall R.M., Goguen J.A., 1982, *Algebras, Theories and Freeness: An Introduction for Computer Scientists*, Proc. 1981 Marktoberdorf NATO Summer School, Reidel, 329–348.
- Cardelli L., Gordon A.D., 1998, *Mobile Ambients*, in: Nivat M. (ed.), *Proceedings of the First international Conference on Foundations of Software Science and Computation Structure*, Lecture Notes In Computer Science, Vol. 1378, 140–155.
- CCITT, 1992, *Specification and Description Language*, CCITT Z100, International Consultative Committee on Telegraph and Telephony, Geneva.
- Cichoń P., Huzar Z., 2005, *Wyrażanie???*, w: Kwiecień A., Gaj P. (red.), *Systemy informatyczne z ograniczeniami czasowymi*, WKiŁ.
- Cichoń P., Huzar Z., 2006, *Wyrażanie synchronizacji zdarzeń w języku UML*, w: Kwiecień A., Gaj P. (red.), *Systemy informatyczne z ograniczeniami czasowymi*, WKiŁ, 13–22.
- Clarke E., Emerson E.A., Sistla A.P., 1983, *Automatic verification of finite-state concurrent systems using temporal logic*, in: 10th Annual Symposium on Principles of Programming Languages, ACM.
- Clark R., Moreira A., 2000, *Use of E-LOTOS in adding formality to UML*, Journal of Universal Computer Science, Springer Pub. Co., Vol. 6 (11), 1071–1087.
- Comer D.E., 1997, *Sieci komputerowe TCP/IP. Zasady, protokoły i architektura*, WNT.
- Czachórski T., 1999, *Modele kolejkowe w ocenie efektywności sieci i systemów komputerowych*, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego.
- D’Argenio P.R., Hermanns H., Katoen J.-P., 1999, *On generative parallel composition*, in: Proc. On Probabilistic Methods in Verification, ENTCS, Vol. 22, Elsevier.

- Esterel Technologies, 2005, *The Esterel v7 Reference Manual. Version v7_30 – initial IEEE standarization proposal*.
- Dembiński P., 1997, *Semantics of Timed Concurrent Systems*, Fundamenta Informaticae, 29, 27–50.
- Dembiński P., Janowska A., Janowski P., Penczek W., Pórola A., Szreter M., Woźna B., Zbrzezny A., 2003, *A Tool for Verifying Timed Automata and Estelle Specifications*, Proceedings of TACAS'03, 2619, Springer.
- Dijkstra E.W., 1978, *Umiejętność programowania*, WNT.
- Douglass B.P., 1998, *Real-Time UML*, Addison-Wesley.
- Douglass B.P. 1999, *Doing Hard Real. Developing Real-Time Systems with UML*, Addison-Wesley.
- Dubielewicz I., Hnatkowska B., Huzar Z., Tuzinkiewicz L., 2006, *An Approach to Software Quality Specification and Evaluation*, Sacha K. (ed.), *Software engineering Techniques: Design for Quality*, Springer, 155–166.
- Ehrig H., Mahr B., 1985, *Fundamentals of Algebraic Specifications I*, EATCS Monographs on Theoretical Computer Science, Vol. 6, Springer.
- van Eijk P.H.J., Vissers C.A., Diaz M. (eds.), 1989, *The Formal Description Technique LOTOS*, North-Holland.
- Ellsberg J., Hogrefe D, Sarma A., 1997, *SDL Formal Object-Oriented Language for communication systems*, Prentice Hall.
- Fencott C., 1996, *Formal Methods for Concurrency*, International Thomson Computer Press.
- Fernandez J.-C., Mounier L., 1995, *A Local Checking Algorithm for Boolean Equation Systems*. Rapport SPECTRE 95-07, VERIMAG, Grenoble.
- Garavel H., Sifakis J., 1990, *Compilation and Verification of LOTOS Specifications*, in: Logrippo L., Probert R.L., Ural H. (eds.), *Protocol Specification, Testing, and Verification*, North-Holland.
- Garavel H., Mateescu R., 1996a, *French-Romanian proposal for capture of requirements and expression of properties in E-LOTOS modules*, Rapport SPECTRE 96-04, VERIMAG, Grenoble, May 1996. Input document (KC4) to the ISO/IEC JTC1/SC21/WG7 Meeting on Enhancements to LOTOS (1.21.20.2.3), Kansas City, Missouri, USA, May 12–21, 1996.
- Garavel H., Sighireanu M., 1996b, *French-Romanian integrated proposal for the user language of E-LOTOS*, Rapport SPECTRE 96-05, VERIMAG, Grenoble, May 1996. Input document (KC3) to the ISO/IEC JTC1/SC21/WG7 Meeting on Enhancements to LOTOS (1.21.20.2.3), Kansas City, Missouri, USA, May 12–21, 1996.
- Garavel H., 1998, *OPEN/CAESAR: An Open Software architecture for Verification, Simulation and Testing*, in: Steffen B. (ed.), *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS Vol. 1384, Springer, 68–84.

- Garavel H., Lang F., Mateescu R., 2001, *An overview of CADP 2001*, INRIA, Rapport technique, No. 0254, December.
- Garavel H., Hermanns H., 2001, *On Combining Functional Verification and Performance Evaluation using CADP*, Rapport de Recherche 4492, INRIA.
- Gibson J.P., 1993, *Formal Object Oriented Development of Software Systems using LOTOS*, PH. D. Dissertation, Department of Computing Science, University of Stirling.
- Girault C., Valk R., 2003, *Petri Nets for Systems Engineering: a Guide to Modelling, Verification and Applications*, Springer.
- Gnesi S., Schieferdecker I., Rennoch A. (eds.), 2000, 5th International ERCIM Workshop on Formal Methods for Industrial Critical Systems, Proceedings of FMICS'2000, April 3–4, Berlin, Report GDM – Forschungszentrum Informationstechnik.
- Götz N., Herzog U., Rettelbach M., 1993, *Multiprocessor and distributed system design: the integration of functional and performance analysis using stochastic process algebras*, in: Donatiello L., Nelson R. (eds.), *Performance Evaluation of Computer and Communication Systems*, LNCS 729, Springer, 121–146.
- Górski J., 2001, *Inżynieria oprogramowania w projekcie informatycznym*, Mikom.
- Groote J.F., 1990, *Specification and Verification of Real Time Systems in ACP*, in: Logrippo L., Probert R., Ural H. (eds.), *Protocol Specification, Testing and Verification, X*, North-Holland, 262–274.
- Groote J.F., 1990, *Transition system specifications with negative premises*, in: Baeten J.C.M., Klop J.W. (eds.), *CONCUR '90*, LNCS, Vol. 458, Springer, 332–341.
- Le Guernic P., Gautier P., Le Borgne M., Le Maire C., 1991, *Programming real-time applications with SIGNAL*, Proc. IEEE, Vol. 79, 1321–1336.
- Haj-Hussein M., Logrippo L., 1991, *Specifying Distributed Algorithms in LOTOS*, in: Komorowski W. (ed.), Proceedings of the Conference 'Computer Networks', Wydawnictwo Politechniki Wrocławskiej, 50–62.
- Halbwachs N., Caspi P., Raymond P., Pilaud D., 1991, *The synchronous dataflow programming language*, Proceedings of the IEEE, Vol. 79 (9), 1305–1320.
- Hanson H., Jonsson B., 1990, *A calculus for communicating systems with time and probabilities*, Proc. 11th IEEE Real-Time Systems Symposium, 278–287.
- Harel D., 1979, *First Order Dynamic Logic*, Springer.
- Harel D., 1987, *Statecharts: A visual formalism for complex systems*, Science of Computer Programming, Vol. 8, No. 3, 231–274.
- Harel D., Kozen D., Tiuryn J., 2000, *Dynamic Logic*, MIT.
- Harel D., Marelly R., 2003, *Come, Let's Play*, Springer.
- Heitmeyer C., Mandrioli D. (eds.), 1996, *Formal Methods for Real-Time Computing*, Wiley.
- Hennessy M.C.B., Plotkin G.D., 1979, *Full abstraction for a simple programming language*, LNCS 74, Springer, 108–120.

- Hennessy M.C.B., Plotkin G.D., 1980, *A term model for CCS*, LNCS 88, Springer, 261–274.
- Hennessy M.C.B., Milner R., 1985, *Algebraic laws for nondeterminism and concurrency*, Journal of the ACM, Vol. 32, 137–161.
- Hennessy M.C.B., 1988, *Algebraic Theory of Processes*, The MIT Press.
- Hennessy M.C.B., Reagan T., 1991, *A temporal process algebra*, in: Formal Description Techniques, III, Quemada J., Manas J., Vazquez (eds.), North-Holland, 33–48.
- Herrmanns H., Rettelbach M., 1994, *Syntax, Semantics, Equivalences for MTIPP*, in: Herzog U., Rettelbach M. (eds.), Proc. of 2nd Workshop on Process Algebras and Performance Modelling.
- Herrmanns H., Rettelbach M., Weiss T., 1995, *Formal characterisation of immediate actions in SPA with nondeterministic branching*, The Computer Journal, Vol. 38, 530–541.
- Herrmanns H., Rettelbach M., 1996, *A superset of Basic LOTOS for performance prediction*, in: Ribaud M. (ed.) Proceedings of the 4th Workshop on Process Algebras and Performance Modelling, CLUT, 77–94.
- Herrmanns H., Herzog U., Katoen J.-P., 2002, *Process Algebra for Performance Evaluation*, Theoretical Computer Science, 274, 43–87.
- Herzog U. 1996, *A concept for graph-based stochastic process algebras, generally distributed times*, in: Ribaud M. (ed.) Proceedings of the 4th Workshop on Process Algebras and Performance Modelling, CLUT, 1–20.
- Hillston J., 1996, *Compositional Approach to Performance Modelling*, Cambridge University Press.
- Hnatkowska B., 1998, *Obiektowy język specyfikacji systemów reaktywnych – O-LOTOS*, rozprawa doktorska, Wydziałowy Zakład Informatyki, Politechnika Wrocławska.
- Hnatkowska B., Huzar Z., 2000, *Extending the UML with a multicast synchronization*, Proceedings of Workshop on Rigorous Object Oriented Methods, York, UK, January 2000.
- Hnatkowska B., Huzar Z., 2001, *Transformation of dynamic aspect of UML models into LOTOS behaviour expressions*. Int. J. Appl. Math. Comput. Sci., Vol. 11, No. 2, 537–556.
- Hnatkowska B., Huzar Z., Tuzinkiewicz L., 2001, *Związki pomiędzy modelami w metodyce USDP*, III krajowa konferencja Inżynierii oprogramowania, Mikom, 103–112.
- Hoare C.A.R., 1978, *Communicating sequential processes*, Communications of ACM, Vol. 21, 666–677.
- Hoare C.A.R., 1985, *Communicating sequential processes*, Prentice Hall International.
- Huzar Z., 1989, *Programowanie procesów komunikujących się w czasie rzeczywistym*, Centrum Obliczeniowe, Monografia 6/1, Wydawnictwo Politechniki Wrocławskiej.
- Huzar Z., 1990, *Specyfikacja formalna usług warstwy sesji dla poczty elektronicznej*. Raporty Centrum Obliczeniowego Politechniki Wrocławskiej, Ser. SPR nr 4,
- Huzar Z., 1991, *An operational approach to semantics of real-time programming language*, Computers and Artificial Intelligence, Vol. 10, 239–254.

- Huzar Z., 1991, *Real Time LOTOS*, in: Proceedings of the Computer Networks '91 Conference, Komorowski W. (ed.), 63–77, Computer Centre, Technical University of Wrocław.
- Huzar Z., 1996, *Real-Time CCS**, *Archiwum Informatyki Teoretycznej i Stosowanej*, t. 8, z. 1–2, 147–167.
- Huzar Z., 2001, *Zastosowanie UML do projektowania systemów czasu rzeczywistego – metodyka ROPES*, w: *Systemy czasu rzeczywistego. Wykłady zaproszone*, Szmuc T. (red.), VII Konferencja, Kraków, 25–28 września 2000, Katedra Automatyki AGH, 3–60.
- Huzar Z., 2002, *Elementy logiki dla informatyków*, Oficyna Wydawnicza Politechniki Wrocławskiej.
- Huzar Z., Fryźlewicz Z., Dubielewicz I., Hnatkowska B., Waniczek J., 1998, *Ada 95*, Helion.
- Huzar Z., Kuźniarz L., 1990a, *Język LOTOS – specyfikacje typów danych*, *Informatyka*, r. 25, nr 1, 19–22.
- Huzar Z., Kuźniarz L. 1990b, *Język LOTOS – specyfikacje procesów*. *Informatyka*, r. 25, nr 2, 21–24.
- Huzar Z., Kuźniarz L. 1990c, *Język LOTOS – specyfikacja systemów*. *Informatyka*, r. 25, nr 3, 26–28.
- Huzar Z., Kuźniarz L., 1993, *Hiding deadlocks in LOTOS semantics*, *Archiwum Informatyki Teoretycznej i Stosowanej*, t. 5, z. 3–4, 409–427.
- Huzar Z., Kuźniarz L., Łach M., 1997, *A software tool for deadlock detection in basic Lotos expressions*, *Raporty Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej*, ser. SPR nr 16.
- Huzar Z., Kuźniarz L., Reggio G., Sourrouille J.-L., 2005, *Consistency Problems in UML-based Software Development*, in: Nunes N.J. et al. (eds.), <<UML>> Satellite Activities, LNCS 3297, Springer, 1–12.
- Huzar Z., Kuźniarz L., Spławski Z., Magott J., 1995, *Specification of synchronization requirements in multimedia systems*, in: Proc. 2nd Conference on Real Time Systems, Szklarska Poręba, 20–23 września, Oficyna Wydawnicza Politechniki Wrocławskiej, 122–131.
- Huzar Z., Magott J., 1995a, *Real-time and performance evaluation extensions of specification language LOTOS*, in: Proc. MASCOTS '95 – IEEE and ACM International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, January 18–20, 1995, Durham, USA, 382–386.
- Huzar Z., Magott J., 1995b, *Timing requirement specification in a real-time LOTOS extension*, in: Proc. Conference on Real-Time Systems, Paris, January 11–13, 5–20.
- Huzar Z., Magott J., 1996, *Reduction method for transition system of a performance evaluation extension of LOTOS*, in: Ribaud M. (ed.), Proc. 4th Workshop on Process Algebras and Performance Modelling, Torino, July 5–7, CLUT, 95–120.
- Huzar Z., Magott J., 1997a, *Syntax and semantics of a real-time and performance evaluation extension of LOTOS*, *Fundamenta Informaticae*, Vol. 29, 77–96.

- Huzar Z., Magott J., 1997b, *Performance modelling extension of TE-LOTOS*, *Archiwum Informatyki Teoretycznej i Stosowanej*, t. 9, z. 1–4, 3–31.
- Huzar Z., Magott J., 1997c, *Markovian Basic LOTOS with Priorities and Probabilities*, in: *Proc. 5th Workshop on Process Algebras and Performance Modelling*, Twente, June 26–27.
- Huzar Z., Magott J., 1997d, *Probabilistic extension of TE-LOTOS*, in: *Proc. 2nd Workshop on Applied Formal Methods*, Zagreb, 89–99.
- Huzar Z., Magott J., 2000, *New semantics for Markovian Statecharts*, in: *Proc. 16th Annual UK Performance Engineering Workshop*, Durham, July, 115–128.
- Huzar Z., Magott J., 2001, *Markovian Extension of Basic LOTOS with priorities and probabilities*, in: *Proc. 17th Annual UK Performance Engineering Workshop*, Leeds, July, 167–178.
- Huzar Z., Szałowski Z., 1989, *Tworzenie oprogramowania sieciowego – problemy metodologiczne i teoretyczne*, *Prace Naukowe Centrum Obliczeniowego Politechniki Wrocławskiej* 9/4, *Sieci komputerowe – projektowanie, zastosowania, eksploatacja*, Wydawnictwo Politechniki Wrocławskiej, 132–148.
- ISO 7498, 1984, *Information Processing systems – Open Systems Interconnection – Basic Reference Model*.
- ISO 8870, 1989, *Information Processing systems – Open Systems Interconnection – LOTOS: A Formal Description Technique based on Temporal Ordering of Observation Behaviour*.
- ISO 9074, 1989, *Information Processing systems – Open Systems Interconnection – Estelle: A Formal Description Technique based on Extended State Transition Model*.
- ISO/IEC 10746-2, 1995, *Open Distributed Processing – Reference Model – Part 2: Foundations*.
- ISO/IEC 10746-3, 1995, *Open Distributed Processing – Reference Model – Part 3: Architecture*.
- ISO/IEC 15437, 2001, *Information Technology – Enhancements to LOTOS (E-LOTOS)*.
- ISO/IEC 9621-1:2000, *Software engineering – Product quality – Part 1: Quality model*.
- ISO/IEC TR 9621-2:2002, *Software engineering – Product quality – Part 2: External metrics*.
- ISO/IEC 14598-3:2000, *Software engineering – Product evaluation – Part 3: Process for developers*.
- ISO/IEC TR 9621-3:2002, *Software engineering – Product quality – Part 4: Internal metrics*.
- ISO/IEC TR 9621-4:2002, *Software engineering – Product quality – Part 4: Quality in use metrics*.
- ISO/IEC 10746-1, 1995, *Information technology – Open Systems Interconnections – Data Management and Open Distributed Processing – Overview*.

- ISO/IEC 10746-2, 1995, *Information technology – Open Systems Interconnections – Data Management and Open Distributed Processing – Basic reference model of open distributed processing: Descriptive model*.
- ISO/IEC 10746-3, 1995, *Information technology – Open Systems Interconnections – Data Management and Open Distributed Processing – Basic reference model of open distributed processing: Prescriptive model*.
- ISO/IEC 10746-4, 1995, *Information technology – Open Systems Interconnections – Data Management and Open Distributed Processing – Architectural semantics, specification techniques and formalisms*.
- ISO/IEC 9646-3, 1998, *Information technology – Open Systems Interconnections – Tree and Tabular Combined Notation*.
- ITU-T, 1999, Recommendation Z.100, *Specification and Description Language*.
- Jaszkiewicz A., 1997, *Inżynieria oprogramowania*, Helion.
- Juanole G., Atamna Y., 1991, *Dealing with arbitrary time distributions with the Stochastic Timed Petri Net model – Applications to queuing networks*, in: Proc. Workshop Petri Nets and Performance Models, Melbourne, December 1991, IEEE Computer Society Press, 32–41.
- Katoen J.-P., Brinksma E., Latella D., Langerak R., 1996, *Stochastic simulation of event structures*, in: Ribaud M. (ed.), Proc. 4th Workshop on Process Algebra and Performance Modelling, CLUT Press, 21–40.
- Kosmulska-Bochenek E., 2002, *Wymiana informacji w heterogenicznych systemach sieciowych – język ASN.1*, Oficyna Wydawnicza Politechniki Wrocławskiej.
- Koymans R., Shyamasundar R.K., De Roeper W.P., Gerth R., Arun-Kumar S., 1985, *Compositional semantics for real-time distributed computing*, LNCS 193, 167–189, Springer.
- Kuśkowski K., 2004, *Konstrukcja i analiza sterowników wspomagana metodami formalnymi*, rozprawa doktorska, Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki, Akademia Górniczo-Hutnicza.
- Leduc G., 1992, *A methodology for the design of large LOTOS specifications and its application to ISO 8073*, University de Liege, Esprit Project 5341/ Sector OBS, SART 92/19/05.
- Leduc G., Leonard L., 1992, *A timed LOTOS supporting a dense time domain and including new timed operators*, in: Diaz M., Groz R. (eds.), *Formal Description Techniques*, Participants' Proceedings, 99–114.
- Leduc G., Leonard L., 1995, *An introduction to ET-LOTOS for the description of time-sensitive systems*, Computer Networks and ISDN Systems, 29 (3), 271–292.
- Linger R.C., Lipson H.F., McHugh J., Mead N.R., Sledge C.A., 2002, *Life-Cycle Models for Survivable Systems*, Technical Report, Carnegie Mellon Software Engineering Institute – 2002-TR-026, ESC-TR-2002-026.
- Logrippo L., Probert R.L., Ural H. (eds.), 1990, *Protocol Specification, Testing, and Verification*, North-Holland.

- Logrippo L., 2000, *Immaturity and potential of formal methods: A personal view*, Proceedings of the 6th Six International Workshop on Feature Interactions in Telecommunications and Software Systems, IOS Press, 9–13.
- Magott J., 2005, *Techniki opisu formalnego systemów informatycznych czasu rzeczywistego*, WKiŁ.
- Manna Z., Pnueli A., 1992, *A Temporal Logic of Reactive and Concurrent Systems*, Springer.
- Maciaszek L.A., 2004, *Requirements Analysis and System Design*, Pearson, Addison-Wesley.
- MDA, 2003, Guide Version 1.0.1, Document OMG: omg/2003-06-01, OMG.
- MOF QVT, 2005, *Final Adopted Specification*, Document OMG: ptc/05-11-01.
- Mouhoub M., Sadaoui S., 2005, Parkway W., *Improving Lotos Simulation Using Constraint Propagation*, Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence, IEEE.
- Miguel C., Fernandez A., Vidaller L., 1993, *LOTOS extended with probabilistic behaviour*, Formal Aspects of Computing, Vol. 5, 253–281.
- Milner R., 1980, *A Calculus of Communicating Systems*, Springer.
- Milner R., 1983, *Calculi for Synchrony and Asynchrony*, Theoretical Computer Science, Vol. 25, 267–310.
- Milner R., 1989, *Communication and Concurrency*, Prentice-Hall.
- Milner R., 1999, *Communicating and Mobile Systems: the π -calculus*, Cambridge University Press.
- Milner R., Tofte M., Harper R., 1990, *The Definition of Standard ML*, MIT Press.
- Moller F., Tofts C., 1989, *A Temporal Calculus of Communicating Systems*, Technical Report ECS-LFCS-89-104, Laboratory for the Foundations of Computer Science, Edinburgh University.
- De Nicola R., Hennessy M.C.B., 1984, *Testing Equivalence for Processes*, Theoretical Computer Science, Vol. 34, No. 1, 2, 83–133.
- De Nicola R., Vaandrager F.W., 1990, *Action versus state based logics for transition systems*, LNCS 469, Springer, 407–419.
- Nicollin X., Sifakis J., 1992, *An Overview and Synthesis on Timed Process Algebras*, in: de Bakker J.W. et al. (eds.), *Real-Time: Theory in Practice*, LNCS, Vol. 600, Springer, 526–548.
- Parrow J.G., 1985, *Fairness Properties in Process Algebra*, PhD Thesis, DoCS 85/03, Computer Science Department, Uppsala University.
- Pavon S., Larrabeiti D., Rabay G., 1995, *LOLA – LOTOS Laboratory, User Manual*, Departamento de Ingenieria Telematica, Universidas Politecnica de Madrid, LOLA/NS/V10.
- Paige R., Tarjan R.E., 1987, *Three Partition Refinement Algorithm*, SIAM Journal of Computing, Vol. 16 (6), 973–989.

- Pelc L., 2004, *Specyfikacja i walidacja protokołów komunikacyjnych czasu rzeczywistego*, rozprawa doktorska, Politechnika Wrocławska.
- Priami C., 1996, *Stochastic π -calculus with general distributions*, in: Ribaud M. (ed.), Proc. 4th Workshop on Process Algebra and Performance Modelling, CLUT Press, 41–57.
- Plotkin G., 1981, *A structural approach to operational semantics*, Technical Report DAIMI-FN 19, Department of Computer Science, Aarhus University.
- Queille J.-P., Sifakis J., 1983, *Fairness and related properties in transition systems – a temporal logic to deal with fairness*, Acta Informatica, Vol. 19, 195–220.
- Quemada J., Fernandez A., 1987, *Introduction to quantitative relative time into LOTOS*, in: Protocol Specification and Verification, VII, Rudin H., West C.H. (eds.), Elsevier Science Publishers, IFIP, 105–121.
- Quemada J., Pavón S., Fernandez Z., 1989, *State exploration by transformation with LOLA*, Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, June.
- Quemada J., Azcorra A., Frutos D., 1990, *TIC: A timed calculus for LOTOS*, in: Formal Description Techniques, II, Voung S.T. (ed.), North-Holland, 195–209.
- Quemada J., Frutos D., Miguel C., 1993, *The design of timed systems*, in: Proc. 1st AMAST Workshop on Real-Time Systems, Nov. 3–4, 1993, Iowa City, USA.
- Reed G.M., Roscoe A.W., 1986, *A timed model for communicating sequential processes*, Proceedings of ICALP '86, LNCS 226, Springer, 314–324.
- Reed G.M., Roscoe A.W., 1999, *The timed failure-stability model for CSP*, Theoretical Computer Science, 11 (1–2), 85–127.
- Reisig W., 1988, *Sieci Petriego*, WNT.
- Rettelbach M., 1995, *Probabilistic branching in Markovian process algebras*, The Computer Journal, 38, 590–599.
- Rico N., von Bochmann G., 1991, *Performance description and analysis for distributed systems using a variant of LOTOS*, in: Jonsson B. et al. (eds.), *Protocol Specification, Testing and Verification IX*, North-Holland, 199–213.
- Roscoe A.W., 1998, *The Theory and Practise of Concurrency*, Prentice Hall.
- Salaün G., Serwe W., 2005, *Translating Hardware Process Algebras into Standard Process Algebras — Illustration with CHP and LOTOS*, Rapport de Recherche 5666, INRIA.
- Salwicki A., Müldner T., 1981, *On the algorithmic properties of concurrent programs*, LNCS 125, Springer, 169–197.
- de Saqui-Sannes P., Apvrille L., Lohr C., Sénac P., Courtiat J.-P., 2002, *UML and RT-LOTOS An Integration for Real-Time System Validation*, European Journal of Automation (JESA), Hermès, Vol. 36, 1029–1042.
- Sawyer P., Kotonaya G., 2001, *Software Requirements*, in: SWEBOK, 2.1–2.26.

- Schieferdecker I.K., 1995, *Performance-oriented specification of communication protocols and verification of deterministic bounds of QoS characteristics*, Ph. D. Dissertation, GMD-Bericht Nr. 242, R. Oldenbourg Verlag.
- Schneider S., 1995, *An operational semantics for timed CCS*, Information and Computing, 116, 193–213.
- Schneider J.M., 1996, *Protocol Engineering. A Rule-Based Approach*, Springer.
- Schneider K., 2004, *Verification of Reactive Systems. Formal Methods and Algorithms*, Springer.
- Schieferdecker I.K., 1995, *Performance-oriented specification of communication protocols and verification of deterministic bounds of QoS characteristics*, PhD Thesis, GMD-Bericht Nr. 242, R. Oldenbourg Verlag.
- Sjödin P., 1991, *From LOTOS Specifications to Distribute Implementations*, PhD Thesis, DoCS 91/31, Department of Computer Science, Uppsala University.
- Splawski Z., 1997, *Synchronization mechanisms for multimedia streams and their specification in timed LOTOS*, New Frontiers of Information Technology – Proceedings of the 23th Euromicro Conference, IEEE, 456–463.
- SWEBOK – *Guide to Software Engineering Body of Knowledge*, 2001, IEEE Trial Version 1.0, May.
- Szejko S. (red.), 2002, *Metody wytwarzania oprogramowania*, MIKOM.
- Szmuc T., 1998, *Zaawansowane metody tworzenia oprogramowania czasu rzeczywistego*, CCATIE 15, Krakowskie Centrum Informatyki Stosowanej.
- Szmuc T., Motet G., 1998, *Specyfikacja i projektowanie oprogramowania czasu rzeczywistego*, CCATIE 6, Krakowskie Centrum Informatyki Stosowanej.
- Szpyrka M., 1999, *Wspomaganie tworzenia oprogramowania systemów reaktywnych z zastosowaniem kolorowanych sieci Petriego*, rozprawa doktorska, Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki, Akademia Górniczo-Hutnicza.
- Tatcher J., Wagner E., Wright J.B., 1982, *Data Type Specification: Parameterization and The Power of Specification Techniques*. Trans. Prog. Lang. and Systems, 4 (4), 711–732.
- Tiuryn J., 2003, *Wstęp do teorii mnogości i logiki*, Uniwersytet Warszawski, Wydział Matematyki, Informatyki i Mechaniki.
- Tofts C., 1989, *Timing Concurrent Processes*, Technical Report ECS-LFCS-89-103, Laboratory for the Foundations of Computer Science, Edinburgh University.
- Turner K.J. (ed.), 1993, *Using Formal Description Techniques*, Wiley.
- Turner K.J., 2000, *Realising Architectural Feature Descriptions using LOTOS. Parallel Computers, Networks and Distributed Systems* (Calculateurs Paralleles, Reseaux et Systemes Repartis), Hermes.
- UML, 1999, *Unified Modeling Language*, version 1.3, Object Management Group.
- UML, 2003, *Unified Modeling Language: Superstructure*, version 2.0, Final Adopted Specification, ptc/03-08-02, Object Management Group.

- Valderrutten A., Hjej O., Benzekri A., Gazal D., 1992, *Deriving queuing networks performance models from annotated LOTOS specifications*, in: Pooley R., Hillston J. (eds.), *Computer Performance Evaluation – Modelling Techniques and Tools*, Edinburgh University Press, 167–178.
- Vissers C.A., Scollo G., van Sinderen M., 1988, *Architecture and specification style in formal descriptions of distributed systems*, in: Aggarwal S., Sabnani (eds.), *Protocol Specification, Testing and Verification*, North-Holland, 189–204.
- Vissers C.A., Scollo G., van Sinderen M., Brinksma E., 1991, *Specification Styles in Distributed System Design and Verification*, Theoretical Computer Science, 89, 179–206.
- Walkowiak A., 2006, *Transformacja diagramów sekwencji języka UML 2.0 w wyrażenia behawioralne języka specyfikacji formalnych LOTOS*, w: Kwiecień A., Gaj P. (red.), *Systemy informatyczne z ograniczeniami czasowymi*, WKiŁ, 41–56.
- Wallace D., Recker L., 2001, *Software Quality*, in: SWEBOK, 11.1–11.19.
- Wang Y., 1991, *CCS + Time = an Interleaving Model for Real Time Systems*, in: Albert L.J., Mounier B., Artalego M.R. (eds.), *Automata, Languages and Programming*, LNCS 510, Springer, 217–228.

W monografii przedstawiono język LOTOS, jego zastosowania, a także jego rozszerzenia, które umożliwiają wykorzystanie języka w procesie specyfikacji i projektowania systemów czasu rzeczywistego. Omówiono dwa rodzaje rozszerzeń: rozszerzenia czasowe, które pozwalają na specyfikowanie systemów czasu rzeczywistego z silnymi ograniczeniami czasowymi, oraz rozszerzenia wydajnościowe, które umożliwiają analizę wydajnościową systemów czasu rzeczywistego ze słabymi ograniczeniami czasowymi. Przedstawiono metodykę stosowania LOTOSa, a zwłaszcza jego wykorzystanie w specyfikacji usług i protokołów sieciowych jako głównego zakresu zastosowania języka.

Omówiono również podstawy matematyczne, na których opiera się LOTOS - są nimi: koncepcja algebraicznej specyfikacji abstrakcyjnych typów danych oraz koncepcja algebraicznych specyfikacji behawioralnych, oparta na rachunku komunikujących się procesów. W monografii poświęcono dużo uwagi definiowaniu i analizie semantyki języka, zwłaszcza w kontekście jego czasowych i wydajnościowych rozszerzeń.



Wydawnictwa Politechniki Wrocławskiej są do nabycia w księgarni „TECH”
plac Grunwaldzki 13, 50-377 Wrocław, budynek D-I PWr., tel. 071 320 29 35
Prowadzimy sprzedaż wysyłkową

ISBN 978-83-7493-335-3