Prace Naukowe Centrum Obliczeniowego
Politechniki Wrocławskiej
Seria: Monografie

8
1

Zbigniew Huzar

Programowanie procesów komunikujących się w czasie rzeczywistym

Wrocław 1989

PRACE NAUKOWE POLITECHNIKI WROCŁAWSKIEJ

Scientific Papers of the Computer Centre
No. 8 of the Technical University of Wrocław No. 8

Monographs No. 1 1989

Zbigniew HUZAR

Programming of real-time communicating processes

Prace Naukowe Centrum Obliczeniowego
Politechniki Wrocławskiej 8

Seria:
Monografie 1

Zbigniew Huzar

Programowanie procesów komunikujących się w czasie rzeczywistym

Wydawnictwo Politechniki Wrocławskiej · Wrocław 1989

Recenzenci
Jacek BAŃKOWSKI
Juliusz L. KULIKOWSKI
Antoni MAZURKIEWICZ

Redaktor naukowy
Witold KOMOROWSKI

Opracowanie redakcyjne
Janina SURMACZYŃSKA

Korekta
Małgorzata NAGAŃSKA-PILAK

Praca została częściowo wykonana w ramach CPBR 8. 13 "Budowa Krajowej Akademickiej Sieci Komputerowej", cel badawczy nr 52 "Formalne metody specyfikacji i konstrukcji oprogramowania sieciowego"

WYDAWNICTWO POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISSN 0860-7311

Nakład 150 + 60 egz. Ark. wyd. 11,25. Ark. druk. 9¾. Papier offset. kl. III, 70 g, B1.
Oddano do druku w grudniu 1988 r. Druk ukończono w styczniu 1989 r.
Zakład Graficzny Politechniki Wrocławskiej. Zam. nr 4059/88. Cena zł 300,-

Prace Naukowe Centrum Obliczeniowego
Nr 8 Politechniki Wrocławskiej Nr 8
Monografie Nr 1 1989

*Programowanie współbieżne,
programowanie w czasie rzeczywistym,
język programowania, semantyka
operacyjna, system dowodzenia
programów*

Zbigniew HUZAR*

PROGRAMOWANIE PROCESÓW KOMUNIKUJĄCYCH SIĘ W CZASIE RZECZYWISTYM

Celem pracy jest zbudowanie, na przykładzie prostego języka programowania, modelu opisu semantyki oraz systemu dowodzenia własności programów czasu rzeczywistego. Punktem wyjścia jest definicja języka programowania sekwencyjnych procesów komunikujących się w czasie rzeczywistym, nazywanego RTCSP (Real Time CSP), stanowiącego modyfikację języka CSP (Communicating Sequential Processes) Hoare'a. Semantykę języka definiuje się wykorzystując strukturalne podejście operacyjne zaproponowane przez Plotkina. Dowodzenie własności programów prowadzi się w oparciu o logikę Hoare'a. Wprowadza się pojęcie schematu systemu dowodzenia programów oraz dowodzi jego niesprzeczności i relatywnej zupełności. Pokazuje się także sposób wyznaczania konkretnego systemu dowodzenia i ilustruje jego zastosowanie na przykładowym, nietrywialnym programie. Zasadnicze wyniki pracy sprowadzają się więc do dwóch, ściśle powiązanych elementów - zbudowania modelu definiowania semantyki języka programowania sekwencyjnych procesów komunikujących się w środowisku czasu rzeczywistego oraz - ścisłego sformułowania problemu poprawności i systemu dowodzenia własności programów czasu rzeczywistego.

1. WSTĘP

1.1. Przedmiot pracy

Ostatnie piętnastolecie przyniosło ogromny postęp w teorii i praktyce programowania. Osiągnięcia technologii układów mikroprocesorowych, powiązanie komputerów ze środkami telekomunikacyjnymi i budowa różnych typów sieci komputerowych stworzyły możliwości nowych rodzajów zastosowań, a równocześnie stały się impulsem poszukiwania nowych koncepcji

* Centrum Obliczeniowe Politechniki Wrocławskiej, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

przetwarzania informacji, wykraczających poza ramy nakreślone tradycyjnym, sekwencyjnym podejściem neumanowskim. Powstało pilne zapotrzebowanie na dostarczenie programiście odpowiednich narzędzi programistycznych oraz racjonalnych metod ich użytkowania.

Obecnie rozwijane koncepcje przetwarzania informacji można podzielić na dwie grupy. Do pierwszej zalicza się podejście polegające na uogólnieniu klasycznej koncepcji von Neumana - podejścia te nazywa się imperatywnymi. Drugą grupę stanowią tzw. podejścia nieimperatywne. Obecnie są tu rozwijane głównie koncepcje programowania aplikatywnego i logicznego. Koncepcje te omówiono w podrozdz. 1.2. Ponieważ rozważania autora opierają się na podejściu pierwszym, w podrozdziale tym podano przegląd podstawowych koncepcji imperatywnych i tylko ogólnie omówiono koncepcje nieimperatywne.

Przyjęcie pewnej koncepcji przetwarzania informacji jest podstawą zdefiniowania języka programowania, tzn. wyboru odpowiednich mechanizmów i konstrukcji językowych oraz precyzyjnego określenia ich znaczenia - czyli semantyki języka. Istnieje bardzo bogaty dorobek w zakresie różnych proponowanych mechanizmów i konstrukcji językowych. Najszersze ich zestawienie w zakresie programowania niesekwencyjnego zawiera praca [7], bogate przeglądy dają także prace [5], [108], [119].

W pracy definiuje się pewien język programowania procesów sekwencyjnych komunikujących się w czasie rzeczywistym. Do opisu jego semantyki wybrano podejście operacyjne. Krótki przegląd różnych metod opisu semantyki języków programowania imperatywnego przedstawiono w podrozdz. 1.3. Rozwój prac nad metodami opisu semantyki języków niesekwencyjnych jest tak gwałtowny, że trudno byłoby wskazać pozycję literaturową dokonującą aktualnego i wyczerpującego przeglądu badań w tym zakresie. Spośród wielu prac o charakterze częściowo przeglądowym można wymienić [9], [14], [15], [29], [113], [119].

Mając narzędzie, czyli język programowania, może programista rozwiązywać problem, czyli pisać program. Punktem wyjścia przy rozwiązywaniu problemu jest jego specyfikacja, ostatecznym zaś rezultatem - tekst programu. W ustalaniu zamierzonego związku między specyfikacją a programem spotyka się dwa podejścia [182]. Jedno podejście, nazywane konstruktywnym lub analitycznym, polega na rozwijaniu i uszczegółowianiu specyfikacji początkowej, co w kolejnych krokach prowadzi do tekstu programu, np. [52], [57]. Drugie podejście, nazywane syntetycznym, polega na napisaniu pewnego tekstu programu, a następnie na zbadaniu jego zgodności ze specyfikacją, np. [145], [156]. Istota tych podejść i zakres ich stosowalności są zasadniczym przedmiotem inżynierii oprogramowania, np. [78], [207], [208], natomiast ich elementem wspólnym jest pewien formalny system, nazywany systemem dowodzenia własności programów, który słu-

ży do sprawdzania zgodności programu ze specyfikacją, a także może być podstawą formułowania reguł rozwijania specyfikacji.

W pracy skupiono uwagę na zdefiniowaniu i zbadaniu własności logicznych systemów wnioskowania o własnościach programów pisanych w uprzednio wspomnianym języku programowania procesów sekwencyjnych komunikujących się w czasie rzeczywistym. System wnioskowania jest oparty na logice Hoare'a.

Przedmiotem pracy jest zatem język programowania współbieżnego w czasie rzeczywistym. Badania nad językami programowania współbieżnego dotyczą trzech grup zagadnień:

- formalizacji semantyki języków,
- translacji i implementacji języków,
- konstrukcji i weryfikacji programów.

Rozważania w pracy dotyczą pierwszego i trzeciego z wymienionych zagadnień. Istotną i oryginalną cechą podjętych rozważań jest uwzględnienie upływu czasu wykonania programu na jego semantykę. Kolejne podrozdz. 1.2 oraz 1.3, stanowiąc przegląd koncepcji języków programowania oraz metod opisu semantyki, wiążą tematykę pracy z aktualnie prowadzonymi badaniami.

1.2. Przegląd koncepcji języków programowania

Jak wspomniano, przedstawiona praca mieści się w nurcie koncepcji imperatywnych. Rozwój tych koncepcji polegał na przejściu od programowania sekwencyjnego do niesekwencyjnego. Pojęcie programowania niesekwencyjnego wiąże się z wyobrażeniem równolegle działających procesów sekwencyjnych, które w pewien sposób współpracując ze sobą, wykonują obliczenia cząstkowe składające się na wspólny wynik.

Nieformalnie proces obrazuje sekwencję pewnych działań - niepodzielnych czynności obliczeniowych, nazywanych też akcjami. Akcje mogą dotyczyć tylko zmian stanu obiektów lokalnych danego procesu bądź też mogą wiązać się z oddziaływaniem z innymi procesami, czyli z wymianą informacji między procesami.

Spośród różnych klasyfikacji procesów warto zwrócić uwagę co najmniej na następujące dwie klasyfikacje.

Pierwsza, historycznie najstarsza i dotycząca tylko programowania imperatywnego, wiąże się z dwoma rodzajami środowiska, w którym działają procesy. Wyróżnia się środowiska skupione i rozproszone. Środowisko skupione oznacza, że procesy współpracują ze sobą dzięki dostępowi do wspólnej pamięci. Różne mechanizmy programowe współpracy w takim środowisku omawiają m.in. prace [7], [106], [108], [119], przykładami zaś odpowiednich języków programowania są m.in. Concurrent Pascal [106], [108], Edison [34], Chill [42], Modula [214].

W środowisku rozproszonym współpraca procesów odbywa się wyłącznie poprzez przesyłanie komunikatów bezpośrednio między parami komunikujących się procesów. Różne mechanizmy programowe współpracy w takim środowisku omawiają m.in [7], [106], [108],[210], są one też dyskutowane w podrozdz. 2.5. Przykładami odpowiednich języków programowania są m.in. DP [33], Star Mod [50], Small-talk [75], CSP [96], Maxwell [102], Conic [122], [198], Occam [203]; inne rozwiązania przedstawiają też prace [63], [93], [139-142], [189-193], [196], [197].

Rozważania przedstawione w pracy dotyczą programowania współbieżnego w środowisku rozproszonym, gdyż definiowany w pracy język programowania jest modyfikacją wymienionego wyżej języka CSP Hoare'a.

Druga z klasyfikacji procesów [20] wiąże się z upływem czasu podczas wykonywania akcji. Dokładniej, można wyróżnić dwa kryteria odnoszące się do sposobu upływu czasu podczas wykonywania akcji autonomicznych w danym procesie oraz podczas wykonywania akcji komunikacji procesu z innymi procesami. Ze względu na sposób wykonania akcji autonomicznych mówi się o wariantach:

- synchronicznym, gdy akcje równoległych procesów są wykonywane współbieżnie z tą samą szybkością,
- asynchronicznym, gdy akcje te przebiegają z własnymi, niezależnymi szybkościami.

Podobnie, ze względu na sposób wykonywania akcji komunikacyjnych, można mówić o komunikacji:

- synchronicznej, gdy oba komunikujące się procesy wykonują równocześnie odpowiednie akcje,
- asynchronicznej, gdy wykonanie akcji komunikacji w jednym z procesów nie wymaga wykonania odpowiedniej akcji w drugim z procesów.

Przedstawione kryteria wyznaczają cztery kategorie procesów. Ponieważ klasyfikacja druga jest bardziej abstrakcyjna niż klasyfikacja pierwsza i odnosi się nie tylko do programowania imperatywnego, w podanym niżej zestawieniu przykładów języków należących do poszczególnych kategorii są wymienione także języki nie będące językami programowania imperatywnego.

Do pierwszej kategorii - procesów wykonujących się i komunikujących się synchronicznie - można zaliczyć m.in. SCCS i ASCCS Milnera [158], [159] oraz ASP Bergstry i Klopa [19].

Kategoria procesów wykonujących się synchronicznie, a komunikujących się asynchronicznie nie jest, jak dotychczas, przedmiotem zainteresowania i badań, dlatego nie można wskazać tu odpowiednich przykładów.

Do kolejnej kategorii procesów wykonujących się asynchronicznie i komunikujących się synchronicznie należą m.in.: CCS [163], CSP [96], CCSP [169], [170], TCSP [168], Ada [107], sieci Petri [38], [41], [70], [71] oraz inne prezentowane np. w [16], [17], [37].

Ostatnia kategoria procesów wykonywanych i komunikujących się asynchronicznie grupuje m.in. Chill [42], języki sterowane przepływem danych [1], [204], [205], języki "aktorskie" [88], [89], [90], [91], [92], [93].

Przyjmując za punkt odniesienia przedstawioną klasyfikację, tematyka pracy należy do trzeciej z wymienionych kategorii.

Jak wspomniano, rozwojowi nurtu imperatywnego towarzyszą poszukiwania koncepcji nieimperatywnych. Ponieważ tematyka pracy nie należy do tej grupy koncepcji, więc ich omówienie ogranicza się tylko do wskazania referencji. Można tu wyróżnić dwa kierunki prac określane jako programowanie aplikatywne oraz programowanie logiczne. Koncepcję pierwszego z tych kierunków omawiają m.in. [1], [204], [205], [208]. W ramach drugiego kierunku dominują prace oparte na logice klasycznej [121], lecz należy także podkreślić poszukiwania nowych dróg, m.in. na podstawie logiki konstruktywnej [48], [49], [149].

Charakterystycznym wyróżnikiem pracy jest jawne włączenie upływu czasu podczas prowadzonych obliczeń. Oznacza to wejście w dziedzinę programowania nazywaną programowaniem w czasie rzeczywistym. Pojęcie to nie zawsze jest rozumiane jednoznacznie. Według Le Lanna za systemy czasu rzeczywistego uważa się takie systemy, które są zdolne do sterowania (poprzez układy wyjściowe) oraz do monitorowania (poprzez układy wejściowe) obiektów bądź procesów, które podlegają własnym, zadanym a priori zasadom [133].

Tradycyjnie zalicza się do takiego programowania tworzenie oprogramowania systemów komputerowych, taki np. jak: prowadzących zbieranie danych radarowych, sterujących pracą silników odrzutowych, sterujących obiektami latającymi czy - ogólniej - systemów sterujących procesami technologicznymi lub produkcyjnymi. Nie uważa się na ogół za systemy czasu rzeczywistego klasycznych wielodostępnych systemów komputerowych lub sieci komputerowych, mimo występujących w nich jawnych reakcji na upływający czas, np. odcinki czasu przeterminowania. Systemy takie, choć nie są uważane za systemy czasu rzeczywistego, są jednocześnie obiektem pomiarów ich dynamicznych własności, co wskazuje na pewne niekonsekwencje w traktowaniu czasu. Niekonsekwencje te są w dużej mierze świadome, gdyż wynikają z przeczucia znacznych trudności w wyrażaniu własności obliczeń z jawnym uwzględnieniem czasu [214].

Podobnie trudno jednoznacznie określić, co to jest język programowania w czasie rzeczywistym [24], [61]. Analizując bogaty przegląd rozwoju języków czasu rzeczywistego zawarty w [74], trudno wskazać na takie koncepcyjne cechy, które jednoznacznie odróżniają języki czasu rzeczywistego od innych współbieżnych języków programowania. Różnice sprowadzają się do cech drugorzędnych z punktu widzenia filozofii programowania

współbieżnego, należą do nich przede wszystkim:

- mechanizmy reakcji na upływ czasu,
- rozbudowane mechanizmy komunikacji z obiektami sprzężonymi z komputerem.

Zgodnie natomiast podkreśla się potrzebę efektywnej implementacji języka.

Współczesne języki programowania współbieżnego, np. Ada, Chill, mogą równie dobrze być uważane za języki programowania ogólnego, jak i programowania w czasie rzeczywistym, np. [76]. Wiele starszych języków, w powiązaniu z udogodnieniami systemów operacyjnych, w których są implementowane (np. Pascal w systemie R-32), także można wykorzystywać jako narzędzia programowania w czasie rzeczywistym.

Próbując uchwycić istotę programowania w czasie rzeczywistym, należy brać pod uwagę trzy elementy:

- specyfikację problemu,
- język programowania,
- środowisko wykonawcze programów.

Problem należy uznać za problem czasu rzeczywistego wtedy, gdy w jego sformułowaniu jawnie występują związki pomiędzy tym, co program oblicza a tym, w jakim czasie są te obliczenia wykonywane. Często w specyfikacji programu, mimo że ma stanowić rozwiązanie problemu czasu rzeczywistego, związki te nie są uwidocznione. Wynika to na ogół z przekonania, czy też tylko nadziei, że program napisany w odpowiednim języku programowania, zgodnie z wymogami sztuki programistycznej, będzie na tyle efektywny, iż nie naruszy nie wyrażonych jawnie ograniczeń czasowych. W tym przypadku niejawnie zakłada się również, że środowisko wykonawcze programu (implementacja języka) ma wystarczająco dobre charakterystyki dynamiczne.

Zatem rozstrzygając o tym, czy dany problem czasu rzeczywistego ma poprawne rozwiązanie w danym języku programowania, nie sposób tego uczynić bez poczynienia ustaleń dotyczących własności dynamicznych środowiska wykonawczego. W pracy podjęto próbę powiązania tych elementów w taki sposób, aby możliwe było wykazanie, że dany program w danym środowisku spełnia zadaną specyfikację.

1.3. Przegląd koncepcji opisu semantyki

Rozwojowi koncepcji i mechanizmów programowania współbieżnego towarzyszą poszukiwania jednoznacznych, wygodnych sposobów wyrażania ich semantyki. Poszukiwania te czerpią inspiracje z dwóch źródeł: jednym są powstające konkretne mechanizmy programowe lub języki programowania, drugim - abstrakcyjne wyobrażenia procesów obliczeniowych, nie wiążące

się bezpośrednio ze znanymi, powszechnie akceptowanymi konstrukcjami programowymi. Przykładami prac wyrastających z pierwszego źródła są [8], [10], [26], [35], [65], [67], przykładami zaś prac wyrastających z drugiego źródła są [19], [20], [22], [23], [37], [45], [46], [114], [115], [120], [123], [150], [154], [151], [152], [153], [159], [160], [162], [170], [182], [211], [212], [213].

Charakteryzując związek pomiędzy wymienionymi dwoma grupami prac, można powiedzieć, że druga z nich zajmuje się tworzeniem i badaniem własności pewnych formalnych modeli - języków formalnych, które znajdują zastosowanie w pierwszej grupie prac zajmujących się opisem semantyki konkretnych języków programowania. Należy podkreślić, że okres ostatnich kilku lat cechuje wyraźny wzrost liczby publikacji z tego zakresu. Dlatego przytaczane dalej referencje są dalekie od wyczerpującego przeglądu.

Ograniczając się tylko do prac związanych z modelem procesów wykonujących się asynchronicznie i komunikujących się synchronicznie (podrozdz. 1.2), można wskazać na następujące zasadnicze modele prezentacji zachowań systemów współbieżnych [37].

Systemy przejść. Model ten, wyrastający z uogólnienia automatu niedeterministycznego, został zaproponowany przez Kellera [114], a następnie rozwinięty i zastosowany przez Plotkina i Hennessy [83], [178], [179] do zdefiniowania operacyjnej semantyki CSP, CCS. Model ten szeroko wykorzystują inne prace m.in. [8], [9], [10], [11], [12], [16], [36], [157], [158], [159], [168], [169], [199]. Także w niniejszej pracy adoptuje się pojęcie systemu przejść do definicji operacyjnej semantyki wprowadzonego języka czasu rzeczywistego.

Drzewa synchronizacji. Model został wprowadzony przez Milnera [157] i stanowi rozwinięcie systemów przejść. Model ten omawia m.in. praca [212]. Również Milner rozpatrywał algebrę termów jako model dla CCS [157].

Obszerna grupa modeli opiera się na wykorzystaniu sieci Petri [3], [38], [70], [71], [184], [185] lub modeli z nimi związanych. Do grupy tej można zaliczyć:

- struktury zdarzeniowe Winskela [212], [213],
- teorię śladów Mazurkiewicza [151], [152], [153], [154],

a także inne prace, np. [2], [37], [38], [41], [45], [46], [53], [111], [119], [120], [162], [163], [168], [169], [175], [201].

Oddzielna grupa modeli opiera się na wykorzystaniu języka logiki do specyfikacji systemów współbieżnych [135], [180]. Wśród wielu podejść można wskazać m.in. na zastosowanie klasycznych logik Hoare'a [9], [14] i różnych jej uogólnień, np. [12], [36], [109], [136], [137], [199], logiki dynamicznej Pratta [80], logiki algorytmicznej [18], [186], a zwła-

szcza logik modalnych. Te ostatnie, a zwłaszcza pewna ich podklasa tzw. logiki temporalne, są rozwijane bardzo intensywnie np. [15] ,[81], [126], [128], [135], [146], [147], [148], [161].

Opis semantyki konkretnych języków programowania opiera się na wyborze podejścia denotacyjnego, algebraicznego, operacyjnego lub aksjomatycznego [56], [174]. W obrębie każdego z tych podejść istnieje duża różnorodność prac, charakteryzująca się wyborem i sposobem użycia różnych formalizmów. I tak, w obrębie podejść denotacyjnych można wyróżnić m.in.:

- metodę VDM [20], która - opracowana początkowo przez Bjørnera i Jonesa dla języków sekwencyjnych - znalazła później szerokie zastosowanie w definicji, a także w implementacji języków programowania współbieżnego m.in. Edison, Ada, Chill, a także współbieżnej wersji Prologu [26], [27], [28], [31], [32], [65];

- metodę równań stałopunktowych, zastosowaną m.in. do opisu semantyki języków CSP oraz Occam [22], [35], [39], [62], [155];

- zastosowanie tzw. struktur metrycznych oraz dziedzin uporządkowanych [176], [177]; metody te omawia bliżej praca przeglądowa [11].

Podejście algebraiczne, w najczystszej postaci, polega na zdefiniowaniu algebry abstrakcyjnej definiującej dany język. Przykładem aksjomatycznej definicji takich algebr są prace [19], [20]. W szerszym sensie do podejść algebraicznych zalicza się te, które używają języka algebry do zdefiniowania semantyki. W takim sensie do tego nurtu można zaliczyć prace np. [21], [40], [84], [86], [157], [158], [164], [211], [212], [213].

Wśród podejść operacyjnych można wyróżnić trzy kierunki:

- tradycyjny, bazujący na wykorzystaniu, opracowanej pod koniec lat sześćdziesiątych, metody VDL [56], [174], zastosowanej po raz pierwszy do opisu semantyki języka PL/I, a później m. in. do opisu semantyki języka czasu rzeczywistego Tomal [82];

- nowszy, powstały na początku lat osiemdziesiątych, wywodzący się od zaproponowanej przez Plotkina abstrakcyjnej metody operacyjnej [83], [178], [179]; podejście Plotkina okazało się bardzo atrakcyjne, co znalazło wyraz w licznych zastosowaniach m.in. [4], [10], [12], [17], [36], [114], [138], [159], [160], [168], [169], [170], stając się w ten sposób współczesnym standardem operacyjnej semantyki;

- najmłodszym jest chyba kierunek oparty na modelu sieci Petri, w tym sensie, że dopiero ostatnie lata przyniosły wiele prac, w których proponuje się wyrażanie semantyki języków modelowych typu CCS oraz CSP, a także pokrewnych np. Cosy, poprzez zastosowanie wcześniej rozwiniętych metod sieci Petri. Można w tej grupie wymienić wiele prac np. [22], [23], [41], [45], [46], [119], [120], [151-154], [162], [163], [168],

[169], [170], [175], warto też zwrócić uwagę na potencjalne szerokie możliwości zastosowania czasowych sieci Petri [9] do opisu semantyki języków czasu rzeczywistego.

Semantyka aksjomatyczna, zainicjowana pracami Floyda [64] i Hoare'a [94] w końcu lat sześćdziesiątych, stanowi obecnie nieodłączny element w specyfikacji i konstrukcji oprogramowania. Podejście Hoare'a, oparte na logice klasycznej, stało się standardem, a prace [8], [9], [36], [43], [73], [125], [127], [171], [172], [173], [183], [199] stanowią wycinkowy fragment publikacji dotyczących tego podejścia. Wśród innych podejść należy wskazać na poprzednio wymienianą logikę dynamiczną, logikę algorytmiczną oraz liczne odmiany logik modalnych. Należy podkreślić, że są prowadzone liczne poszukiwania nowych nieklasycznych logik, np. logiki konstruktywnej, nie tylko do specyfikacji, ale i do programowania, za przykład mogą służyć prace [48], [49], [149].

Spośród przedstawionych koncepcji i narzędzi opisu semantyki, rozważania prezentowane w niniejszej pracy wykorzystują podejście operacyjne i podejście aksjomatyczne. Podejście operacyjne służy zdefiniowaniu pierwotnej semantyki wprowadzonego języka programowania czasu rzeczywistego, podejście zaś aksjomatyczne służy budowie systemu wnioskowania o własnościach programów w tym języku.

Narzędziem opisu semantyki operacyjnej jest zmodyfikowany system przejść, uogólniający podejście Plotkina. Istotna zaleta tego podejścia polega na tym, że wykorzystuje się, znaną wcześniej w podejściu denotacyjnym, zasadę strukturalizacji semantyki (znaczenie programu wyraża się poprzez złożenie znaczeń składowych części programu). Podejście operacyjne do definiowania semantyki jest stosunkowo proste - pojęciowo znacznie mniej złożone niż podejście denotacyjne, oraz może w bezpośredni sposób wyznaczać implementację języka.

W celu zbudowania systemu wnioskowania o własnościach programów wykorzystuje się w przedstawianej pracy również semantykę aksjomatyczną. Ten sposób wykorzystania semantyki aksjomatycznej jest zresztą podstawowym zakresem jej zastosowania. Metoda aksjomatyczna jest na ogół komplementarna i wtórna w stosunku do metody denotacyjnej lub operacyjnej, rzadko bowiem jest stosowana jako bazowa metoda definiowania semantyki języka programowania. Wprowadzana semantyka akcjomatyczna bazuje na logice Hoare'a [95], zmodyfikowanej stosownie do cech definiowanego języka programowania procesów sekwencyjnych komunikujących się w czasie rzeczywistym. Przesłanki dokonanych modyfikacji były inspirowane przede wszystkim pracami Apta [8], [10], [12], [14] oraz Levina [136], [137], a częściowo pracą Soundararajana [199].

Pojęcie czasu używane w programowaniu rozproszonym lub programowaniu w czasie rzeczywistym opiera się na modelu substancjalnym lub ewen-

tystycznym [116], [124], [132], [133]. Model substancjalny traktuje czas jako czas globalny (absolutny), wspólny dla wszystkich procesów składowych programu [116]. Model ewentystyczny [124] zakłada, że każdy proces składowy dostrzega upływ czasu poprzez ciąg obserwowanych zdarzeń. Przy odpowiednich założeniach, dotyczących sposobu przenoszenia informacji pomiędzy procesami, każdy z komponentów programu obserwuje tak samo uporządkowane ciągi obserwowalnych przez siebie zdarzeń [132]. W niniejszej pracy przyjmuje się model czasu absolutnego. Zakłada się ponadto, że upływ czasu dokonuje się skokowo, co pewien (jednakowej długości) odcinek czasu.

1.4. Cel i wyniki pracy

Poprzednia część rozdziału stanowiła prezentację przedmiotu pracy na tle badań w zakresie rozwoju koncepcji programowania oraz metod opisu semantyki. W prezentacji tej starano się uwypuklić problemy związane z uwzględnieniem upływu czasu rzeczywistego na wykonanie programu. Analiza literatury przedmiotu wskazuje na to, że języki programowania czasu rzeczywistego nie mają jeszcze ostatecznie ustalonych i powszechnie akceptowanych modeli opisu semantyki. Wniosek taki wypływa na przykład z obszernego przeglądu Andrewsa i Schneidera [7], dotyczącego głównie mechanizmów języków niesekwencyjnego programowania imperatywnego i częściowo metod opisu ich semantyki, bądź też ze zbioru artykułów zebranych przez Glassa [74], obejmującego przegląd wszystkich zagadnień związanych z tworzeniem oprogramowania czasu rzeczywistego, czy też z autorytatywnego artykułu Le Lanna [133] będącego przeglądem problemów związanych ze specyfikacją i projektowaniem rozproszonych systemów czasu rzeczywistego. Bardziej dojrzała jest sytuacja w zakresie mechanizmów języków programowania w czasie rzeczywistym: istnieje wiele prac, które przedstawiają takie same lub podobne mechanizmy komunikacji programu z otoczeniem, a szczególnie mechanizmy reakcji na upływ czasu podczas oczekiwania na komunikację [48], [107], [122], [131], [134], [144], [189], [193], [198], [203].

Brak formalnych i pełnych metod opisu semantyki języków programowania w czasie rzeczywistym odbija się w metodologii tworzenia oprogramowania czasu rzeczywistego; metody specyfikacji, konstruowania, testowania i weryfikacji tego rodzaju oprogramowania są jeszcze bardzo słabo rozwinięte. Tymczasem, jak wspomniano w podrozdz. 1.2, oprogramowanie czasu rzeczywistego występuje także w systemach, które jawnie nie są za takie uważane, np. systemy operacyjne klasycznych systemów wielodostępnych czy sieci komputerowych.

Nakreślona sytuacja uzasadnia wybór tematyki, a także nasuwa cel pracy. Celem pracy jest zbudowanie na przykładzie prostego języka mode-

lu opisu semantyki języka programowania czasu rzeczywistego, umożliwiającego zdefiniowanie i zweryfikowanie oczekiwanych własności programów.

Zasadnicze wyniki pracy sprowadzają się do dwóch, ściśle powiązanych ze sobą elementów.

Pierwszym jest zbudowanie modelu do definiowania semantyki języka programowania sekwencyjnych procesów komunikujących się w środowisku czasu rzeczywistego. Model ten jest adaptacją idei abstrakcyjnej semantyki operacyjnej Plotkina.

Drugim jest ścisłe sformułowanie problemu poprawności programów wykonywanych w środowisku czasu rzeczywistego oraz zbudowanie, w oparciu o logikę Hoare'a, systemu dowodzenia własności takich programów. Stało się to możliwe dzięki pierwszemu rezultatowi - pełnemu zdefiniowaniu semantyki użytego języka programowania.

Punktem wyjścia prowadzonych w pracy rozważań jest definicja prostego języka czasu rzeczywistego. Prezentację i dyskusję podstawowych mechanizmów tego języka, stanowiącego modyfikację języka CSP Hoare'a, przedstawia rozdział 2. Szczególny nacisk położono w nim na mechanizmy komunikacji procesów w środowisku czasu rzeczywistego.

Rozdział 3, dając formalny opis semantyki ustalonego przykładowego języka programowania, prezentuje tym samym nowe podejście do opisu semantyki dowolnego języka programowania czasu rzeczywistego, co stanowi pierwszy wynik pracy.

Rozdziały 4, 5, 6 przedstawiają drugi zasadniczy rezultat pracy i omawiają kolejno problemy dowodzenia własności programów wykonywanych w środowisku czasu rzeczywistego (rozdział 4) oraz badają poprawność i zupełność wprowadzonych schematów dowodzenia własności programów (rozdziały 5, 6).

Rozdział 7 ilustruje, na przykładzie prostego programu, problemy i wprowadzony sposób postępowania przy dowodzeniu programu.

Ostatni rozdział 8 jest krótkim podsumowaniem wyników pracy i oceną ich przydatności oraz wskazuje na kierunki dalszych prac.

* * *

Pragnę złożyć serdeczne podziękowania Panu Profesorowi Jerzemu Bromirskiemu i Panu Docentowi Jerzemu Battkowi - pierwszym Czytelnikom pracy - za stałą zachętę i wsparcie podczas jej realizacji. Dziękuję recenzentom Panu Profesorowi Jackowi Bańkowskiemu, Panu Profesorowi Juliuszowi Kulikowskiemu oraz Panu Profesorowi Antoniemu Mazurkiewiczowi - ich uwagi i sugestie pozwoliły na znaczne ulepszenie tekstu. Dziękuję również kolegom: Doktorowi Józefowi Goetzowi oraz Doktorowi Janowi Magottowi za udostępnienie bogatego materiału bibliograficznego z zakresu sieci Petriego.

2. PODSTAWOWE KONCEPCJE JĘZYKA RTCSP

2.1. Wprowadzenie

Większość używanych języków programowania w czasie rzeczywistym (uwarunkowanego czasem rzeczywistym) ma naturę sekwencyjną. Przykłady takich języków jak RTL, Real Time Fortran i inne przedstawiają [24], [61], [74].

Spośród języków o naturze współbieżnej część z nich była specjalnie projektowana z myśla o zastosowaniach uwarunkowanych czasem rzeczywistym, np. Modula [214], Tomal [82], Conic [198], Occam [203]. Inne natomiast, jak np. Ada [107], projektowane z myślą o zastosowaniach uniwersalnych, okazały się [16] również przydatne jako języki programowania w czasie rzeczywistym. Pomijając przegląd najistotniejszych konstrukcji czasu rzeczywistego, wystarczy ograniczyć się tutaj do stwierdzenia, że wymagają one jeszcze badań [74]. W dalszej części rozdziału zostanie dla ilustracji tego stwierdzenia przeprowadzone porównanie pomiędzy proponowaną konstrukcją komunikacji w czasie rzeczywistym a analogiczną konstrukcją w języku Ada.

W rozdziale przedstawia się propozycję prostego języka czasu rzeczywistego, nazywanego RTCSP (Real Time Communicating Sequential Processes), który jest modyfikacją języka CSP Hoare'a [96]. Uzasadnienie wyboru języka CSP jako bazy do tworzenia języka czasu rzeczywistego wynika z roli jaką język ten, a dokładniej proponowany mechanizm komunikacji pomiędzy równolegle wykonywanymi procesami, odgrywa we współcześnie projektowanych językach programowania.

Mechanizm komunikacji z CSP znajduje odbicie w języku Ada [107], a także w innych, np. Star Mod [50], Maxwell [102], Occam [203]. Język CSP wraz z pewnymi rozszerzeniami jest analizowany pod kątem możliwości różnych zastosowań np. [58], [59], [191], [192], [196], [197].

Opis języka RTCSP, przedstawiony w niniejszym rozdziale, nie jest kompletny, formalnie została tu podana syntaktyka i semantyka statyczna, natomiast formalny opis semantyki właściwej (dynamicznej) jest przeniesiony do następnego rozdziału.

2.2. Syntaktyka

Syntaktyka języka RTCSP jest parametryzowana następującymi rozłącznymi, przeliczalnymi zbiorami obiektów:

Procid - zbiór identyfikatorów procesów; pojedyncze procesy będą oznaczane symbolami P, Q, R z ewentualnymi indeksami.

Portid - zbiór identyfikatorów portów komunikacyjnych; pojedyncze porty będą oznaczane symbolem W z ewentualnymi indeksami.

Clockid - zbiór identyfikatorów zegarów procesów ,

Varid - zbiór identyfikatorów zmiennych; pojedyncze zmienne będą oznaczane symbolami x, y, z z ewentualnymi indeksami.

Exp - zbiór wyrażeń liczbowych, zawierający zbiór liczb całkowitych $Int = \{...,-1,0,1,2,...\}$; pojedyncze wyrażenie będzie oznaczone symbolem e, z ewentualnymi indeksami, liczby całkowite zaś będą oznaczone symbolami v z ewentualnymi indeksami.

Bexp - zbiór wyrażeń logicznych, zawierający zbiór wartości logicznych $Log = \{tt, ff\}$; pojedyncze wyrażenie będzie oznaczone symbolem b z ewentualnymi indeksami, natomiast wartości logiczne będą również oznaczane symbolem v.

Przez Val będzie się oznaczać sumę zbiorów Int ∪ Log.

Przyjmuje się standardowe rozumienie pojęcia zmiennych wolnych i związanych w wyrażeniach [181]. Zakłada się, że wszystkie wyrażenia e oraz b mają skończone zbiory zmiennych wolnych FV(e) oraz FV(b). W standardowy sposób określa się również podstawienie wyrażenia e' pod wszystkie wolne wystąpienia zmiennej x w wyrażeniu e lub b, co będzie zapisywane w postaci $e[e'/x]$, $b[e'/x]$. Podobnie określa się jednoczesne podstawienie wyrażeń e', e'' pod wszystkie wolne wystąpienia różnych zmiennych x', x'' w wyrażeniu e, co będzie zapisywane w postaci $e[e'/x', e''/x'']$.

Przyjmuje się również, że obliczenie wartości wyrażeń nie wywołuje żadnych efektów ubocznych. Sposób obliczania wartości wyrażeń może być określony metodą denotacyjną lub operacyjną. Ponieważ sposób obliczenia wartości wyrażeń, z punktu widzenia prowadzonych rozważań, nie jest istotny, więc ograniczamy się tutaj do przyjęcia oznaczenia $[\![e]\!]$, $[\![b]\!]$ symbolizującego wartość wyrażeń e, b. Wartość wyrażeń b, e zależy od wartościowania występujących w nich zmiennych wolnych oraz od przyjętej interpretacji wyrażeń. Przyjmuje się, że wartościowanie zmiennych, nazywane też stanem programu, będzie opisywane funkcją częściową

$$s: Varid \rightharpoonup Val.$$

Przyjmuje się, że przez cały ciąg rozważań obowiązują ustalone, dowolnie przyjęte interpretacje J wyrażeń. Dlatego, jeśli nie będzie to konieczne, interpretacja J nie będzie w tekście wymieniana jawnie, a więc napisy $[\![e]\!]_s$, $[\![b]\!]_s$ będą oznaczać wartość wyrażeń e, b w interpretacji J przy wartościowaniu s.

Gdy dla danego wartościowania s wartość wyrażenia e jest nieokreślona, będzie to zaznaczone w postaci $[\![e]\!]_s = \underline{error}$.

Syntaktyka języka RTCSP zawiera następujące kategorie obiektów. Ka-

tegorie te są przedstawiane w konwencji stanowiącej oczywiste rozszerzenie notacji BNF.

Gcm - zbiór instrukcji dozorowanych; pojedyncza instrukcja dozorowana, oznaczana przez GC, jest zdefiniowana następująco

$$GC::= \mathop{\Box}_{i=1..n} b_i \rightarrow AC_i, \qquad (n > 0),$$

gdzie AC_i jest dowolną instrukcją, zdefiniowaną dalej.

Ccm - zbiór instrukcji komunikacyjnych; pojedyncza instrukcja, oznaczona CM, jest zdefiniowana następująco

CM::= SC|RC,

gdzie SC jest instrukcją wyjścia, RC - instrukcją wejścia, i mają one postać

SC::= P!W(e),
RC::= P?W(x).

Gcc - zbiór dozorowanych instrukcji komunikacyjnych; pojedyncza instrukcja, oznaczana CC, jest zdefiniowana przez

$$CC::= \mathop{\Box}_{i=1..n} b_i;CM_i \rightarrow AC_i, \qquad (n > 0).$$

Com - zbiór instrukcji; pojedyncza instrukcja, oznaczana AC, jest zdefiniowana równaniem

AC::= skip | abort | x:=e | rcl x | AC;AC | if GC end |
do GC end | through V wait CC later AC end .

Portd - zbiór deklaracji portów komunikacyjnych; pojedyncza deklaracja portu komunikacyjnego, oznaczana PD, jest zdefiniowana następująco

PD::= empty |W| PD, PD,

gdzie empty oznacza, że deklaracja PD może być pusta.

Proc - zbiór procesów; pojedynczy proces, oznaczany PC, jest zdefiniowany przez

PC::= P : PD; AC.

Prog - zbiór programów; pojedynczy program, oznaczany PR, ma definicję

$$PR::= \mathop{\|}_{i=1..n} PC_i, \qquad (n > 0).$$

Podana syntaktyka abstrahuje od szczegółów, które nie są istotne z punktu widzenia analizy komunikacji i współbieżności; pominięto deklarację zmiennych, wprowadzanie różnych typów, zrezygnowano z zagnieżdżania procesów. Zbiór instrukcji strukturalnych języka to zbiór instruk-

cji sekwencyjnych języka programowania Dijkstry [57] z dodaną czasowo uwarunkowaną dozorowaną instrukcją komunikacji (through).

W dalszych zapisach programów będą często wykorzystywane następujące konwencje. Zamiast

$$\square_{i=1..n}\ b_i \longrightarrow AC_i$$

$$\square_{i=1..n}\ b_i;CM_i \longrightarrow AC_i$$

będzie się pisać

$$b_1 \longrightarrow AC_1 \square \ \ldots \ \square b_n \longrightarrow AC_n$$

$$b_1;CM_1 \longrightarrow AC_1 \square \ \ldots \ \square b_n;CM_n \longrightarrow AC_n$$

oraz zamiast

$$\|_{i=1..n}\ P_i : PD_i;AC_i$$

będzie się pisać

$$P_1 : PD_1;AC_1 \| \ldots \| P_n : PD_n;AC_n.$$

Również zamiast through V wait CC later AC end będzie używana skrócona forma th V wt CC lt AC end.

2.3. Semantyka statyczna

Syntaktyka języka RTCSP przedstawiona w poprzednim punkcie jest bezkontekstowa. Język RTCSP, podobnie jak prawie wszystkie języki programowania, ma pewne uwarunkowania kontekstowe. Przed zdefiniowaniem semantyki języka konieczna jest gwarancja, że konstrukcje programowe, których definicja ta dotyczy, są prawidłowo zbudowane w sensie zachowania odpowiednich wymogów kontekstowych. Często wymogi takie określa się mianem semantyki statycznej [138]. Zależności kontekstowe w konstrukcjach programowych można wyrazić w różny sposób, używając w tym celu, na przykład, gramatyk dwupoziomowych lub atrybutowych [56]. W definicji semantyki statycznej języka RTCSP zastosowano formalizm pochodzący od Plotkina [178], [179]. Podejście to jest interesujące, ponieważ wykorzystuje schemat dedukcyjny oparty na indukcji strukturalnej. W przypadku języka RTCSP zasady tworzenia programów prawidłowo zbudowanych są bardzo proste, ze względu jednak na walory elegancji i zwartości warto wyrazić je we wspomnianej konwencji.

Niech Synt będzie zbiorem wszystkich konstrukcji syntaktycznych, tzn.

$$Synt = Gcm \cup Ccm \cup Gcc \cup Com \cup Portd \cup Proc \cup Prog.$$

Jeżeli ρ jest prawidłowo zbudowanym elementem ze zbioru Synt, to będzie to zapisywane w postaci $\vdash \rho$. Wprowadza się następujące zbiory pomocnicze:

PDE(ρ) - zbiór portów komunikacyjnych zadeklarowanych w ρ,

FV(ρ) - zbiór zmiennych wolnych występujących w ρ,

PSC(ρ) - zbiór par (nazwa procesu, nazwa portu) występujących w instrukcjach wyjścia zawartych w ρ,

PRC(ρ) - zbiór par (nazwa procesu, nazwa portu) występujących w instrukcjach wejścia zawartych w ρ.

Wymienione zbiory są zdefiniowane metodą indukcji strukturalnej przez następujące tabele:

	empty	W	PD1, PD2
PDE	$\emptyset$	$\{W\}$	PDE(PD1) $\cup$ PDE(PD2)

	$\square_{i=1..n}\ b_i \longrightarrow AC_i$	P!W(e)	P?W(x)
FV	$\bigcup_{i=1..n} (FV(b_i) \cup FV(AC_i))$	FV(e)	$\{x\}$
PSC	$\bigcup_{i=1..n} PSC(AC_i)$	$\{(P,W)\}$	$\emptyset$
PRC	$\bigcup_{i=1..n} PRC(AC_i)$	$\emptyset$	$\{(P,W)\}$

	$\square_{i=1..n}\ b_i;CM_i \longrightarrow AC_i$	skip, abort	rcl x
FV	$\bigcup_{i=1..n} (FV(b_i) \cup FV(CM_i) \cup FV(AC_i))$	$\emptyset$	$\{x\}$
PSC	$\bigcup_{i=1..n} (PSC(CM_i) \cup PSC(AC_i))$	$\emptyset$	$\emptyset$
PRC	$\bigcup_{i=1..n} (PRC(DM_i) \cup PRC(AC_i))$	$\emptyset$	$\emptyset$

	x:=e	AC1; AC2	if GC end	do GC end
FV	$\{x\} \cup$ FV(e)	FV(AC1) $\cup$ FV(AC2)	FV(GC)	FV(GC)
PSC	$\emptyset$	PSC(AC1) $\cup$ PSC(AC2)	PSC(GC)	PSC(GC)
PRC	$\emptyset$	PRC(AC1) $\cup$ PRC(AC2)	PRC(GC)	PRC(GC)

	th ∨ wt CC lt AC end	P : PD;CD;AC	$\|\|_{i=1..n} PC_i$
FV	FV(CC) ∪ FV(AC)	FV(AC)	$\bigcup_{i=1..n} FV(PC_i)$
PSC	PSC(CC) ∪ PSC(AC)	PSC(AC)	$\bigcup_{i=1..n} PSC(PC_i)$
PRC	PRC(CC) ∪ PRC(AC)	PRC(AC)	$\bigcup_{i=1..n} PRC(PC_i)$

Aksjomaty i reguły tworzenia programów prawidłowo zbudowanych przedstawia podane niżej zestawienie. Zapis w postaci $\frac{A,B}{C,D}$ jest regułą, która oznacza, że A i B implikują C i D, natomiast $\vdash$ AC jest aksjomatem, który oznacza, że AC jest prawidłowo zbudowaną instrukcją.

Instrukcje dozorowane

$$\frac{\vdash AC_i \ (i=1..n)}{\vdash \ \square_{i=1..n} \ b_i \longrightarrow AC_i}$$

Instrukcje komunikacyjne

1. $\vdash$ P!W(e) 2. $\vdash$ P?W(x)

Dozorowane instrukcje komunikacyjne

$$\frac{\vdash CM_i, \ \vdash AC_i \ \ (i=1..n)}{\vdash \ \square_{i=1..n} \ b_i;CM_i \longrightarrow AC_i}$$

Instrukcje

1. $\vdash$ skip 2. $\vdash$ abort 3. $\vdash$ x:=e

4. $\vdash$ rcl x 5. $\dfrac{\vdash AC1, \ \vdash AC2}{\vdash AC1; \ AC2}$

6. $\dfrac{\vdash GC}{\vdash \text{if GC end}}$ 7. $\dfrac{\vdash GC}{\vdash \text{do GC end}}$

8. $\dfrac{v \geqslant 0, \ \vdash CC, \ \vdash AC}{\vdash \text{th } V \text{ wt CC lt AC end}}$

Deklaracje portów komunikacyjnych

1. $\vdash$ empty 2. $\vdash$ W

3. $\dfrac{\vdash PD1, \ \vdash PD2, \ PDE(PD1) \cap PDE(PD2) = \emptyset}{\vdash PD1, \ PD2}$

Procesy

$$\frac{\vdash PD,\ \vdash AC,\ (\forall (Q,W) \in PSC(AC))(Q \neq P),\ (\forall (Q,W) \in PRC(AC))(Q \neq P \wedge W \in PDE(PD))}{\vdash P : PD,AC}$$

Programy

$$\frac{\vdash PC_i \quad (i=1..n),\ FV(PC_i) \cap FV(PC_j) = \emptyset \quad \text{dla } i \neq j \quad (i,j=1..n)}{\vdash \|_{i=1..n} PC_i}$$

gdzie PC_i jest postaci $P_i:PD_i;AC_i$

Spośród przedstawionych reguł komentarza może wymagać tylko reguła dotycząca procesów. Reguła orzeka, że proces jest zbudowany prawidłowo, jeżeli instrukcje komunikacji występujące w treści procesów są takie, że użyte w nich identyfikatory procesów są różne od identyfikatorów danego procesu (proces nie może komunikować się sam ze sobą), a ponadto w przypadku instrukcji wejścia występujące w nich identyfikatory portów komunikacyjnych muszą należeć do zbioru portów zadeklarowanych w tym procesie.

Przyjmuje się, że zbiory zmiennych wolnych w różnych procesach są rozłączne; założenie to ma tylko techniczny charakter. W dalszym ciągu zakłada się, że mamy do czynienia tylko z prawidłowo zbudowanymi programami.

2.4. Semantyka nieformalna

Program w języku RTCSP składa się z pewnej liczby równolegle pracujących procesów. Proces rozpocząwszy swe obliczenia, dokonuje etapowego przekształcenia wartości swych zmiennych. W trakcie obliczeń proces ma dostęp wyłącznie do swych zmiennych lokalnych, natomiast z innymi procesami może współdziałać, przesyłając lub odbierając komunikaty. Obliczenia wykonywane przez proces można podzielić na pewne elementarne, dalej już niepodzielne, kroki. Każdy z takich kroków wymaga do swej realizacji pewnego odcinka czasu. Czas trwania takich odcinków zależy od własności środowiska wykonującego obliczenia programu. Oznacza to, że z każdym procesem P_i jest związany niejawnie pewien licznik (zegar), który progresywnie, w miarę upływu czasu, zwiększa swą zawartość. Przyjmuje się, że wszystkie zegary wskazują ten sam czas kalendarzowy (substancjalne rozumienie czasu), przy czym zmieniają dyskretnie swą zawartość nie w momencie upływu jednej (wspólnej dla wszystkich procesów) jednostki czasu, lecz w momencie rozpoczęcia (zakończenia) przez proces elemen-

tarnego kroku obliczeniowego. Procesy rozpoczynają swe obliczenia w pewnym ustalonym momencie czasu. Stan zegarów procesów będzie opisany funkcją

$$c:Clockid \rightarrow Int.$$

Przyjęcie powyższych założeń dotyczących sposobu odmierzania czasu w poszczególnych procesach jest świadomym uproszczeniem ogólniejszych sytuacji, jakie można rozważać. Problemy synchronizacji czasu w rozproszonych procesach są poruszone m.in. w pracach [116], [124].

Zasady wykonywania poszczególnych instrukcji są następujące.

Instrukcja skip jest instrukcją pustą, niezmieniającą wartości zmiennych, trwającą zerowy odcinek czasu. Instrukcja abort jest instrukcją zrywającą obliczenia. Pozostawia ona nieokreślone wartości zmiennych procesu i trwa nieokreślony odcinek czasu. Instrukcja podstawienia x:=e jest instrukcją niepodzielną trwającą skończony odcinek czasu. Jej efektem jest zmiana wartościowania zmiennych procesu. Niech s będzie wartościowaniem zmiennych programu, tzn. s jest funkcją częściową $s:Varid \rightarrow Val$, w momencie rozpoczęcia instrukcji. Nowym wartościowaniem zmiennych programu, po zakończeniu instrukcji podstawienia, jest $s' = s(v/x)$, gdzie v jest wartością wyrażenia e przy wartościowaniu s, tzn. $v = [\![e]\!]_s$, natomiast $s(v/x)$ jest funkcją częściową taką, że

$$s(v/x)(x') = \begin{cases} s(x') & \text{gdy } x' \neq x, \\ v & \text{gdy } x' = x. \end{cases}$$

Instrukcja rcl x odczytuje wartości zegara i podstawia je pod zmienną x, czyli stan s zostaje zmieniony na $s(v/x)$, gdzie v oznacza wartość pokazywaną przez zegar procesu. Zakłada się, że instrukcja realizuje się natychmiastowo.

Sekwencyjne założenie instrukcji AC1, AC2 jest realizowane w taki sposób, że najpierw jest wykonywana instrukcja AC1, a następnie - - jeśli nie nastąpi zerwanie obliczeń podczas wykonywania AC1 - wykonuje się AC2. Chwila zakończenia AC1 jest momentem rozpoczęcia AC2.

Instrukcja alternatywy if $\square_{i=1..n} b_i \rightarrow AC_i$ end jest wykonywana zgodnie ze schematem:

1. Oblicza się wartości wyrażeń logicznych $b_1, \ldots, b_n$. Obliczenie trwa pewien skończony odcinek czasu.

2. Jeżeli istnieją pewne wyrażenia prawdziwe, tzn. takie, że $[\![b_i]\!]_s = tt$, to jedno z nich, załóżmy b_i, zostaje wybrane (niedeterministycznie) i wykonuje się instrukcję AC_i.

3. Jeżeli nie istnieją wyrażenia prawdziwe, tzn. $[\![b_i]\!]_s = ff$ dla $i = 1..n$, to następuje zerwanie obliczeń.

Instrukcję iteracji **do** $\square_{i=1..n}\ b_i \rightarrow AC_i$ **end** wykonuje się zgodnie z zasadami:

1. Oblicza się wartości wyrażeń logicznych $b_1, \ldots, b_n$. Obliczenie to trwa skończony odcinek czasu.

2. Jeżeli istnieją pewne wyrażenia prawdziwe, to wybiera się jedno z nich, załóżmy b_i, i wykonuje instrukcję AC_i, a po jej zakończeniu powtarza się czynności z punktu 1.

3. Jeżeli nie ma wyrażeń prawdziwych, to następuje zakończenie obliczeń instrukcji.

Wykonanie czasowo uwarunkowanej dozorowanej instrukcji komunikacyjnej **th** $\vee$ **wt** $_{1=1..n}\square b_i$; $CM_i \rightarrow AC_i$ **it** AC **end** przebiega następująco:

1. Oblicza się wartości wyrażeń logicznych $b_1, \ldots, b_n$. Jak poprzednio, obliczenie to trwa skończony odcinek czasu. Alternatywy, dla których odpowiadające im wyrażenia logiczne są prawdziwe nazywa się alternatywami otwartymi.

2. Jeżeli zbiór alternatyw otwartych nie jest pusty, to wyznacza on zbiór instrukcji komunikacji, które oczekują na wymianę informacji z innymi procesami. Spośród nich zostanie wybrana ta, dla której w odcinku czasu o długości v, poczynając od chwili obliczenia zbioru alternatyw otwartych, zostanie spełniony warunek jej realizacji. Jeżeli warunek taki w tej samej chwili zostanie spełniony dla kilku instrukcji komunikacji, to zostaje wybrana (niedeterministycznie) jedna z nich. Po wybraniu instrukcji komunikacji, załóżmy CM_i, oraz po jej wykonaniu wykonuje się AC_i, co kończy wykonanie całej instrukcji.

3. Jeżeli zbiór alternatyw otwartych jest pusty, to następuje **zerwanie obliczeń.**

4. Jeżeli, w przypadku niepustego zbioru alternatyw otwartych, w odcinku czasu przeterminowania o długości v nie nastąpi, dla żadnej otwartej alternatywy, rozpoczęcie realizacji instrukcji komunikacji, to po upływie tego odcinka czasu rozpoczyna się realizacja instrukcji AC.

Podane zasady wymagają uzupełnienia o reguły opisujące wykonanie instrukcji komunikacji CM_i. Instrukcje komunikacji służą przekazywaniu informacji pomiędzy parą procesów: procesem - nadawcą P, który posługuje się w tym celu instrukcją wyjścia postaci Q!W(e) oraz procesem - odbiorcą Q, który wykorzystuje w tym celu instrukcję wejścia postaci P?W(x).

Instrukcja wyjścia (!) wskazuje odbiorcę informacji Q, jego odbiorczy port komunikacyjny W oraz określa przesyłaną informację - jest nią wartość $v = [\![e]\!]_s$, gdzie s jest wartościowaniem zmiennych w momencie wykonywania instrukcji.

Instrukcja wejścia (?) wskazuje nadawcę informacji P, odbiorczy port komunikacyjny W oraz podaje zmienną lokalną x procesu - odbiorcy Q, pod którą będzie podstawiona odbierana wartość v.

Aby taka para syntaktycznie skojarzonych ze sobą instrukcji wejścia/wyjścia - stanowiących elementy dwóch czasowo-uwarunkowanych instrukcji komunikacji w dwóch różnych procesach - mogła być wykonana, musi nastąpić ich zsynchronizowanie (skojarzenie dynamiczne), tzn. musi nastąpić taka sytuacja, że w tym samym momencie czasu w obu mających skomunikować się ze sobą procesach sterowanie dopuści ich realizację.

W przypadku zajścia skojarzenia dynamicznego następuje równoczesna realizacja pary skojarzonych instrukcji wejścia; ich wykonanie jest równoważne instrukcji podstawienia x:=e, przy czym x jest zmienną lokalną procesu odbiorczego, a e wyrażeniem w procesie nadawczym. Zatem instrukcja uwarunkowanej czasowo komunikacji dla zbioru alternatyw otwartych wyznacza zbiór instrukcji komunikacji A, które przez zadany odcinek czasu oczekują aż inny proces również rozpocznie (lub rozpoczął wcześniej) wykonywanie takiej uwarunkowanej czasowo instrukcji komunikacji, która w swoim zbiorze instrukcji komunikacji B, odpowiadających alternatywom otwartym, będzie zawierać pewną instrukcję syntaktycznie skojarzoną z jedną spośród instrukcji zbioru A. Jeżeli oczekiwanie przez zadany odcinek czasu nie doprowadzi do skojarzenia żadnej instrukcji ze zbioru A z inną odpowiadającą jej instrukcją ze zbioru B, to pc upływie tego odcinka czasu zostanie wykonana alternatywa z instrukcją AC.

2.5. Dyskusja

Jak wynika z podanego opisu, język RTCSP jest językiem bardzo prostym. Wszystkie instrukcje strukturalne - oprócz uwarunkowanej czasowo instrukcji komunikacji - są instrukcjami Dijkstry [57] i stanowią już klasyczny element nowo proponowanych języków. Natomiast uwarunkowana czasowo instrukcja komunikacji jest oryginalnym elementem języka, chociaż zawiera znany, wprowadzony przez Hoare'a [96], mechanizm wymiany informacji pomiędzy parami komunikujących się procesów. Instrukcja odczytu zegara rcl (x) jest typową instrukcją spotykaną w większości języków programowania uwarunkowanego czasowo.

Języki programowania współbieżnego w środowisku rozproszonym cechuje duża różnorodność stosowanych mechanizmów wymiany informacji pomiędzy komunikującymi się komponentami - procesami. Spotykane mechanizmy, a przynajmniej większą ich część, można scharakteryzować według następujących kryteriów. Dla wygody opisu przyjmuje się, że procesy są połączone abstrakcyjnym, niezawodnym kanałem łączności, przez który mogą przesyłać sobie komunikaty.

K 1. Proces - nadawca może podczas wysyłania komunikatu do odbiorcy przyjmować następujące wzorce postępowania:

1. Proces kompletuje wiadomości i wysyła w odpowiedni kanał, po czym, nie oczekując na jakąkolwiek reakcję odbiorcy, kontynuuje swe obliczenia.

2. Proces po skompletowaniu wiadomości oczekuje

a) nieskończenie długo,

b) przez zadany odcinek czasu, aż do momentu, gdy proces - odbiorca będzie w stanie przyjąć wiadomości.

3. Proces po skompletowaniu wiadomości oczekuje

a) nieskończenie długo,

b) przez zadany odcinek czasu, aż do momentu, gdy proces - odbiorca po przyjęciu wiadomości skompletuje i prześle w odpowiedzi wiadomość zwrotną.

K 2. Proces - odbiorca może zachowywać się następująco:

1. Proces ma bufor wejściowy

a) nieograniczonej,

b) ograniczonej pojemności, gromadzący napływające wiadomości.

2. Proces nie mając bufora wejściowego, oczekuje

a) nieskończenie długo,

b) przez zadany odcinek czasu, na nadejście kierowanej do niego wiadomości, po czym kontynuuje swe obliczenia.

3. Proces nie mając bufora wejściowego, oczekuje

a) nieskończenie długo,

b) przez zadany odcinek czasu, na nadejście kierowanej do niego wiadomości, po czym - w przypadku odbioru - przygotowuje i wysyła wiadomość zwrotną, a następnie kontynuuje swe obliczenia.

K 3. Kwestią związaną z komunikacją jest sposób identyfikacji procesów. Proces - nadawca zawsze musi znać identyfikator procesu, do którego chce wysłać wiadomości, natomiast proces - odbiorca może bezpośrednio

1) nie otrzymywać,

2) otrzymywać informację o identyfikatorze procesu - nadawcy odebranego komunikatu.

Poniżej podana tabela zestawia charakterystyki mechanizmów komunikacji w wybranych językach programowania współbieżnego i rozproszonego.

Język programowania	Kryteria			Literatura
	K 1	K 2	K 3	
1	2	3	4	5
NP	1	1a	1	[112]
DP	3a	3a	1	[33]
CSP	2a	2a	2	[96]
SR	1, 3a	1a, 3a	1	[6]
ADA	2a, 3a	2a, 2b, 3a, 3b	1	[107]

1	2	3	4	5
Chill	1, 2a	1b, 2	1, 2	[42]
Argus	1, 3b	1b, 3a	2	[142]
Concurrent C	2a, 2b	2a, 2b	1, 2	[206]
Conic	1, 3b	2a, 2b, 3a, 3b	2	[198]
RTCSP	2b	2b	2	

Przegląd mechanizmów komunikacji i synchronizacji przedstawiają m.in. prace [7], [200] oraz praca [210], która dodatkowo rozważa zagadnienia implementacji tych mechanizmów.

Wybór mechanizmów komunikacji w języku RTCSP wymaga krótkiego uzasadnienia. Kierowano się przyjęciem możliwie prostego i uniwersalnego mechanizmu. Łatwo przekonać się, że wybrane mechanizmy są istotnie uniwersalne. Po pierwsze, należy zauważyć, że mając mechanizm K 1 - 2b, przez wprowadzenie wyróżnionej, nieskończonej wartości dla odcinka czasu przeterminowania uzyskuje się mechanizm K 1 - 2a. Po drugie, bezpośrednio widać w jaki sposób z mechanizmów K 1 - 2a, 2b uzyskać mechanizmy K 1 - 3a, 3b. Po trzecie, zaimplementowanie mechanizmu K 1 - 1 jest możliwe przez wprowadzenie dodatkowego procesu symulującego działanie kanału łączności (bufora), przyjmującego wszystkie wiadomości kierowane na jego wejście. Wszystkie wiadomości przygotowane przez proces - nadawcę są odbierane przez proces symulujący kanał, chociaż bezpośredni interfejs między tymi procesami może być jednym z poprzednio wymienionych mechanizmów.

Podobne rozważania mogą przekonać o tym, że mechanizm K 2 - 2b jest wystarczający do implementacji pozostałych mechanizmów zachowań procesu odbiorczego.

Wybrany mechanizm K 3 - 2 jest, oczywiście, ogólniejszy od K 3 - 1, przy czym nie przysparza on żadnych kłopotów implementacyjnych.

Dokonany wybór mechanizmów jest w znacznym stopniu arbitralny; można by przytaczać wiele innych argumentów. Dyskusja tego zagadnienia nie byłaby jednak istotna z punktu widzenia całości rozważań. Warto tu jednak zwrócić uwagę na to, że np. twórcy języka Concurrent C [206], przeznaczonego do implementacji oprogramowania systemowego, zdecydowali się na przyjęcie zbioru mechanizmów komunikacji w synchronizacji zawierającego mechanizmy języka RTCSP; podobnie ewoluują poglądy na dobór tych mechanizmów prezentowane przez Liskov, kierującą realizacją projektu Argus [140], [141], [142].

Warto również skomentować ogólną postać uwarunkowanej czasowo instrukcji komunikacji na tle innej podobnej konstrukcji z języka Ada. Konstrukcją tą jest instrukcja selekcji o następującej, nieformalnie

przedstawionej, postaci:

```
select
   when b1 ⟹ AD1; S1 or
   when b2 ⟹ AD2; S2 or
   ..................
   when bn ⟹ ADn; Sn
   else S
end select
```

gdzie b_1, ..., b_n są wyrażeniami logicznymi, AD_1, ..., AD_n są instrukcjami akceptacji lub opóźnienia, S_1, ..., S_n, S są dowolnymi ciągami instrukcji. Konstrukcja else S może nie występować, jest ona opcją w instrukcji selekcji.

Instrukcja akceptacji jest **odpowiednikiem** instrukcji komunikacji; ogólnie ma ona postać:

accept W (x:in T_1, y:out T_2) do S end W,

gdzie W jest nazwą portu komunikacyjnego (entry) procesu (task), x, y są parametrami wejściowymi i wyjściowymi typów T_1, T_2, S jest dowolnym ciągiem instrukcji. Realizacja instrukcji polega na oczekiwaniu aż pewien proces wywoła port W z aktualnymi parametrami a, b. Gdy to nastąpi, wtedy pod x podstawia się wartość a, oblicza się S i pod b podstawia wartość y. Mamy tu więc wcześniej opisany mechanizm K 2 - 3a komunikacji między procesem wywołującym port (procesem - nadawcą) a procesem zawierającym instrukcję akceptacji (procesem - odbiorcą).

Instrukcja opóźnienia delay (v) powoduje opóźnienie wykonania procesu o v jednostek czasu.

Powracając do instrukcji selekcji, jej realizację wyznaczają następujące reguły [107]:

1. Oblicza się wartości wyrażeń logicznych b_1, ..., b_n i określa się zbiór alternatyw otwartych.

2. Otwarta alternatywa rozpoczynająca się instrukcją akceptacji może być wykonana, jeżeli zachodzi skojarzenie z odpowiadającym jej wywołaniem portu w innym procesie.

3. Otwarta alternatywa rozpoczynająca się instrukcją opóźnienia może być wykonana, jeżeli przed upływem odcinka czasu wskazanego przez tę instrukcję nie została wykonana żadna inna alternatywa.

4. Jeżeli żadna z otwartych alternatyw nie może być wykonana natychmiast i istnieje opcja else, to zostaje wykonany ciąg instrukcji S. Jeżeli opcja else nie istnieje, to oczekuje się, aż jedna z otwartych alternatyw będzie mogła być wykonana dzięki spełnieniu warunków opisanych regułą 2 lub 3.

5. Jeżeli zbiór alternatyw otwartych jest pusty i istnieje opcja else, to wykonuje się ciąg instrukcji S. Jeżeli opcja else nie występuje, to obliczenia zostają zerwane (dokładniej: sygnalizowane jest powstanie sytuacji wyjątkowej (exception)).

W stosunku do instrukcji selekcji można zgłosić co najmniej dwa zastrzeżenia. Po pierwsze: wprowadza ona pewien niepożądany (i prawdopodobnie przez twórców języka nie oczekiwany) rodzaj niedeterminizmu w sytuacji, gdy w zestawie instrukcji występują przynajmniej dwie instrukcje typu delay, o różnych wartościach czasu opóźnienia. Gdy, na przykład, instrukcja ma postać

```
select when true ⟹ accept W.... end W; S1 or
       when true ⟹ delay (5); S2 or
       when true ⟹ delay (10); S3
end select
```

możliwe są następujące - zgodnie z wcześniej podanymi regułami - warianty jej realizacji:

1. Wykonanie pierwszej alternatywy, gdy w odcinku czasu [0, 5], od momentu rozpoczęcia obliczeń instrukcji, nastąpi wywołanie portu W.

2. Wykonanie drugiej alternatywy, po upływie 5 jednostki czasu, gdy wcześniej nie nastąpiło wywołanie portu W.

3. Wykonanie pierwszej alternatywy, gdy w odcinku czasu [0, 10], od momentu rozpoczęcia obliczeń instrukcji, nastąpi wywołanie portu W.

4. Wykonanie trzeciej alternatywy, po upływie 10 jednostek czasu, gdy wcześniej nie nastąpiła realizacja pierwszej lub drugiej alternatywy. (Podobne zastrzeżenie można zgłosić pod adresem języka Conic [122], używającego konstrukcji analogicznej do instrukcji select.)

Drugie zastrzeżenie dotyczy przypadku, gdy jedna z alternatyw rozpoczyna się instrukcją delay i jednocześnie występuje opcja else. W tej sytuacji alternatywa rozpoczynająca się instrukcją delay nie będzie nigdy wykonywana.

Podane wady wskazują na słabość syntaktyczną konstrukcji select - konstrukcja stwarza niepotrzebne pułapki dla programisty. Łatwo stwierdzić, że proponowana konstrukcja through jest wolna od tego typu niebezpieczeństw. W obecnej postaci instrukcja uwarunkowanej czasowo komunikacji jest pewną modyfikacją instrukcji wprowadzanej w [100], natomiast pierwotnym źródłem inspiracji była budowa oprogramowania komunikacyjnego systemu minikomputerowego [68].

3. OPIS SEMANTYKI JĘZYKA RTCSP

3.1. Wprowadzenie

Do opisu semantyki języka RTCSP wykorzystuje się podejście operacyjne. Alternatywnym, możliwym do rozważenia podejściem przy definicji języka jest tylko podejście denotacyjne, bowiem podejście aksjomatyczne ma, w stosunku do podejść poprzednich, charakter uzupełniający; rozstrzygnięcia niesprzeczności i zupełności opisu aksjomatycznego można dokonać tylko według innego opisu semantyki języka. Przy omawianiu języka programowania współbieżnego i dodatkowo po uwzględnieniu upływu czasu wybór podejścia operacyjnego jest nie tylko naturalny, lecz również znacznie prostszy - efektywniejszy w zastosowaniu. Opinia taka o omawianej klasie języków jest w zasadzie powszechna [72], [82], [138].

Istotą podejścia operacyjnego jest formalne przedstawienie modelu interpretacji dowolnego programu, tzn. ciągów akcji (elementarnych, niepodzielnych kroków obliczeniowych) specyfikowanych przez program [56], [174]. Model ten sprowadza się na ogół do zdefiniowania pewnej maszyny abstrakcyjnej, w której wykonanie pojedynczej akcji jest równoważne z wykonaniem pojedynczej instrukcji abstrakcyjnej. Wadą podejścia operacyjnego jest to, że wprowadzając maszynę abstrakcyjną, często wprowadza się jednocześnie wiele nieistotnych jej elementów. Aby uniknąć tej niedogodności, a przynajmniej ograniczyć do osiągalnego minimum, w pracy wykorzystuje się nowe podejście operacyjne - strukturalne podejście operacyjne - zaproponowane i rozwinięte przez Plotkina [83], [178], [179]. Podejście to w istotny sposób wykorzystuje tzw. schematy przejść znakowanych (labelled transition schemata) wprowadzone przez Kellera [114]. Przedstawione poniżej pojęcia są modyfikacją pojęć oryginalnych, wynikającą z konieczności uwzględnienia czasu rzeczywistego oraz własności dynamicznych środowiska, w którym przebiegają obliczenia.

Definicja 3.1. Dynamicznym schematem przejść znakowanych, krótko - - schematem przejść, będzie nazywana czwórka

$$(\Sigma, \Sigma_f, \Lambda, \rightarrow),$$

gdzie Σ, Σ_f, są zbiorami nazywanymi odpowiednio zbiorem konfiguracji programu i zbiorem konfiguracji finalnych $\Sigma_f \subseteq \Sigma$; Λ jest zbiorem oddziaływań międzyprocesorowych; symbol $\rightarrow$ oznacza relację $\rightarrow \subseteq \Sigma \times \Lambda \times \Sigma$ zwaną relacją zmian konfiguracji. ■

Pojęcie konfiguracji, dokładnie określone dalej, można rozumieć jako zestaw istotnych informacji w programie w momencie rozpoczynania lub

zakończenia elementarnego, niepodzielnego kroku obliczeń programu. Wykonanie programu może być opisane pewnym ciągiem konfiguracji. Relacja zmian konfiguracji służy do wyznaczenia kolejnych konfiguracji jakie może przyjąć program. Przejście programu z jednej konfiguracji do następnej może być uwarunkowane albo tym, co dzieje się w pojedynczym procesie, albo przejście do następnej konfiguracji odbywa się w wyniku interakcji (komunikacji) pomiędzy parą procesów. W pierwszym przypadku mówi się o braku oddziaływań międzyprocesowych - będzie to zaznaczane przez tzw. puste oddziaływanie $\varepsilon \in \Lambda$, w drugim przypadku będzie zachodzić pewne niepuste oddziaływanie międzyprocesowe $\lambda \in \Lambda$. Nieformalnie przez oddziaływanie λ należy rozumieć zestaw tych informacji, które opisują komunikat i okoliczności, w których zostały one wymienione między procesami.

Element $(\sigma_1, \lambda, \sigma_2)$ relacji $\longrightarrow$ będzie zapisywany w postaci

$$\sigma_1 \xrightarrow{\lambda} \sigma_2.$$

Jeżeli $\lambda_1, \ldots, \lambda_n \in \Lambda$, $\sigma_0, \ldots, \sigma_n \in \Sigma$ i zachodzi

$$\sigma_0 \xrightarrow{\lambda_1} \sigma_1 \xrightarrow{\lambda_2} \ldots \xrightarrow{\lambda_n} \sigma_n$$

to będzie się zapisywać to w postaci

$$\sigma_0 \xrightarrow{\lambda_1 \ldots \lambda_n} + \sigma_n.$$

Niech Λ^+ oznacza zbiór niepustych ciągów oddziaływań $\lambda \in \Lambda$. Jeżeli dla konfiguracji σ_1, σ_2 istnieje niepusty ciąg oddziaływań $\lambda_1 \ldots \lambda_n \in \Lambda^+$ taki, że

$$\sigma_1 \xrightarrow{\lambda_1 \ldots \lambda_n} + \sigma_2,$$

to będzie się to zapisywać w postaci

$$\sigma_1 \longrightarrow + \sigma_2.$$

Jeżeli konfiguracje σ_1, σ_2 są identyczne lub $\sigma_1 \longrightarrow + \sigma_2$, to będzie się taki fakt zapisywać w postaci

$$\sigma_1 \longrightarrow * \sigma_2.$$

Bezpośrednimi następnikami konfiguracji σ będzie nazywany zbiór konfiguracji

$$\Delta(\sigma) = \left\{\sigma' \mid \sigma \xrightarrow{\lambda} \sigma',\ \lambda \in \Lambda\right\}$$

Definicja 3.2. Obliczeniem w schemacie przejść $(\Sigma, \Sigma_f, \Delta, \rightarrow)$ rozpoczynającym się w konfiguracji σ_0 nazywa się ciąg nieskończony

$$\sigma_0 \xrightarrow{\lambda_1} \sigma_1 \xrightarrow{\lambda_2} \ldots \xrightarrow{\lambda_n} \sigma_n \xrightarrow{\lambda_{n+1}} \ldots$$

lub taki ciąg skończony

$$\sigma_0 \xrightarrow{\lambda_1} \sigma_1 \xrightarrow{\lambda_2} \ldots \xrightarrow{\lambda_n} \sigma_n$$

że $\Delta(\sigma_n) = \emptyset$.

Obliczenie skończone, takie że $\sigma_n \in \Sigma_f$ nazywa się obliczeniem właściwym. Obliczenie skończone, takie że $\sigma_n \notin \Sigma_f$ nazywa się obliczeniem zablokowanym.

Jeżeli dla konfiguracji σ istnieje pewne obliczenie nieskończone, to będzie to zapisywane w postaci

$$\sigma \longrightarrow *\underline{\text{infinity}} ,$$

jeżeli zaś dla σ istnieje pewne obliczenie zablokowane, to będzie to zapisywane w postaci

$$\sigma \longrightarrow *\underline{\text{deadlock}} .$$

■

3.2. Środowisko wykonawcze

Semantyka programów uwarunkowanych czasowo w istotny sposób zależy od własności dynamicznych środowiska, w którym program jest przetwarzany. Obliczenie procesu jest ciągiem wykonań elementarnych, niepodzielnych kroków obliczeniowych. Od własności środowiska wykonawczego zależy czas trwania takiego kroku, przy czym zależy on także od:

1) rodzaju danego elementarnego kroku obliczeń,

2) stanu programu w momencie rozpoczynania tego kroku.

Przyjmuje się natomiast, że nie ma zależności od wartości chwili, w której rozpoczyna się obliczenie danego kroku, czyli od stanu zegarów programu (zakłada się jednorodność czasu trwania obliczeń elementarnych kroków). Dalej zakłada się, że omawiana zależność ma charakter niedeterministyczny, tzn. czas trwania danego kroku obliczeniowego, dla zadanego stanu programu, nie musi być wyznaczony jednoznacznie, lecz może mu odpowiadać pewien skończony zbiór czasów trwania obliczeń. Zbiór ten dla danego kroku obliczeniowego będzie wyznaczony przez pewien predykat, zapisywany w postaci D (dur), gdzie dur (od duration) oznacza zmienną, której wartością jest czas trwania obliczeń tego kroku. Zmienna dur jest różna od zmiennych programowych, tzn. dur $\notin$ Varid. Załóżmy, że przez D[v/dur] będziemy oznaczać podstawienie wartości v pod zmienną dur w predykacie D. Konwencja ta jest w zależności od kontekstu ła-

two odróżnialna od podobnego oznaczenia użytego na podstawienie wyrażenia pod zmienną wolną. Predykt D jest prawdziwy dla wartości v zmiennej dur i danego stanu s, co jest oznaczone w postaci $[\![D[v/dur]]\!]_s = tt$, wtedy i tylko wtedy, gdy $v \geqslant 0$ jest dopuszczalnym czasem realizacji rozważanego kroku. Zatem zbiorem dopuszczalnych czasów realizacji danego kroku obliczeniowego, przy ustalonym stanie początkowym s, jest zbiór

$$\{v \mid [\![D[v/dur]]\!]_s = tt\}.$$

Z przyjęciem klasy predykatów opisujących własności środowiska wykonawczego wiąże się przyjęcie ich interpretacji. Tutaj zakłada się, że może to być dowolnie ustalona interpretacja, zgodna z interpretacją J, przyjętą dla zbiorów Exp i Bexp. Zatem w dalszym ciągu pojęcie interpretacji J będzie wspólnie odnoszone do wszystkich wymienionych klas obiektów.

Elementarne kroki w realizacji procesów oraz ich opis przedstawiają się następująco:

1. Wykonanie instrukcji pustej skip nie zależy od stanu początkowego i odbywa się natychmiastowo.

2. Dla instrukcji zerwania abort czas wykonania jest nieokreślony.

3. Wykonanie instrukcji podstawienia x:=e odbywa się w jednym, niepodzielnym kroku, którego czas trwania wyznacza predykat ASS_e.

4. Wykonanie instrukcji alternatywy (IF) $\underline{\text{if}} \; \square_{i=1..n} \; b_i \longrightarrow AC_i \; \underline{\text{end}}$ rozpoczyna się obliczeniem wartości wyrażeń $b_1, \ldots, b_n$ i wyborem jednej z otwartych alternatyw lub zerwaniem obliczeń. Czas trwania tego kroku, jeśli nie nastąpi zerwanie obliczeń, wyznacza predykat $TEST_{IF}$. Następnymi krokami, jeśli nie nastąpiło zerwanie obliczeń, są kolejne kroki odpowiadające realizacji wybranej instrukcji AC_i.

5. Wykonanie instrukcji iteracji (DO) $\underline{\text{do}} \; \square_{i=1..n} \; b_i \longrightarrow AC_i \; \underline{\text{end}}$ rozpoczyna się obliczeniem wartości wyrażeń $b_1, \ldots, b_n$ i wyborem jednej z otwartych alternatyw lub zakończeniem instrukcji. Czas trwania tego kroku wyznacza predykat $TEST_{DO}$. Gdy instrukcja nie zostanie zakończona, następnymi krokami jej realizacji są kolejne kroki realizacji złożenia instrukcji AC_i; $\underline{\text{do}} \; \square_{i=1..n} \; b_i \longrightarrow AC_i \; \underline{\text{end}}$.

6. Pierwszym krokiem wykonania instrukcji uwarunkowanej czasowo komunikacji (TH) $\underline{\text{th}} \vee \underline{\text{wt}} \; \square_{i=1..n} \; b_i ; CM_i \longrightarrow AC_i \; \underline{\text{lt}} \; AC \; \underline{\text{end}}$ jest również obliczenie wartości wyrażeń $b_1, \ldots, b_n$ i wyznaczenie zbioru alternatyw otwartych lub zerwanie obliczeń. Czas trwania tego kroku, jeżeli nie nastąpiło zerwanie obliczeń, wyznacza predykat $TEST_{TH}$. Następnym krokiem, jeśli nie nastąpiło zerwanie obliczeń, jest krok polegający na oczekiwaniu na zsynchronizowanie się z pewnym procesem przez najwyżej v jednostek czasu; czas jego trwania wyznacza predykat $WAIT_v$, którego

definicja ma oczywistą postać:

$$0 \leqslant dur \leqslant v \ .$$

Następnym krokiem po zsynchronizowaniu się z pewnym procesem jest wykonanie pary skojarzonych instrukcji wejścia/wyjścia P?W(x) i Q!W(e); przyjmuje się, że czas realizacji tego kroku wyznacza predykat $SEND_e$.

Dalszymi krokami są w tym przypadku kolejne kroki realizacji instrukcji AC_i odpowiadające wybranej alternatywie.

Gdy brak jest synchronizacji w odcinku czasu o długości v, w następnej jednostce czasu, po upływie tego odcinka, rozpoczyna się obliczenie pierwszego kroku instrukcji AC.

Definicja 3.3. Środowisko wykonawcze programu jest określone przez rodzinę predykatów ENV

$$ENV = \{ASS_e \mid e \in Exp\} \cup \{TEST_{IF} \mid IF \in Cif\} \cup$$
$$\{TEST_{DO} \mid DO \in Cdo\} \cup \{TEST_{TH} \mid TH \in Cth\} \cup$$
$$\{WAIT_v \mid v \in Int\} \cup \{SEND_e \mid e \in Exp\} \ ,$$

gdzie Cif, Cdo, Cth są podzbiorami Com oznaczającymi zbiory instrukcji alternatywy, iteracji oraz czasowo uwarunkowanej komunikacji. ■

3.3. Konfiguracje procesów

Konfiguracjami roboczymi (niekońcowymi) programów będą trójki $\langle \rho, s, c \rangle$, gdzie ρ oznacza bieżące instrukcje programu, które mają być wykonane, s jest stanem zmiennych programu, natomiast c jest stanem zegarów programu. Formalne określenie zbioru konfiguracji wymaga wprowadzenia pomocniczych konstrukcji syntaktycznych.

Synch - zbiór konstrukcji synchronizujących; pojedyncze konstrukcje synchronizujące, oznaczone przez SY, są zdefiniowane następująco:

SY::=BS|AS
BS::=v wt CC lt AC
AS::=CC syn P

Ecom - zbiór złożeń pomocniczych; pojedynczy element EC tego zbioru jest zdefiniowany następująco:

EC::=GC|SY|GC;AC|SY;AC|AC .

Eproc - zbiór procesów pomocniczych; pojedynczy proces EP jest zdefiniowany przez:

EP::=P:PD;EC .

Eprog - zbiór programów pomocniczych; pojedynczy program EG jest zdefiniowany przez:

$$EG ::= \mathop{||}_{i=1..n} EP_i \ .$$

Oczywiście zachodzą inkluzje:

$$Com \subset Ecom \ ,$$
$$Proc \subset Eproc \ ,$$
$$Prog \subset Eprog \ ,$$

a także

$$Gcm \subset Ecom \ ,$$
$$Synch \subset Ecom \ .$$

Prawidłowość budowy pomocniczych konstrukcji syntaktycznych określają reguły:

1. $$\frac{v \geqslant 0, \ \Vdash CC, \ \Vdash AC}{\Vdash v \ \underline{wt} \ CC \ \underline{lt} \ AC}$$

2. $$\frac{\Vdash CC, \ (\exists W)((P,W) \in PSC(CC) \cup PRC(CC))}{\Vdash CC \ \underline{syn} \ P}$$

3. $$\frac{\Vdash GC, \ \Vdash AC}{\Vdash GC;AC}$$

4. $$\frac{\Vdash SY, \ \Vdash AC}{\Vdash SY;AC}$$

5. $$\frac{\Vdash PD, \ \Vdash EC, \ (\forall (Q,W) \in PSC(EC))(Q \neq P), \ (\forall (Q,W) \in PRC(EC))(Q \neq P \wedge W \in PDE(PD))}{\Vdash P:PD;EC}$$

6. $$\frac{\Vdash EP_i \quad (i=1..n), \ FV(EP_i) \cap FV(EP_j) = \emptyset \quad (i \neq j, \ i,j=1..n)}{\Vdash \mathop{||}_{i=1..n} EP_i}$$

Występujące w tych regułach zbiory FV, PSC, PRC mają oczywiste definicje:

	v $\underline{wt}$ CC $\underline{lt}$ AC	CC $\underline{syn}$ P	GC;AC	SY;AC	P:PD;EC
FV	FV(CC) ∪ FV(AC)	FV(CC)	FV(GC) ∪ FV(AC)	FV(SY) ∪ FV(AC)	FV(EC)
PSC	PSC(CC) ∪ PSC(AC)	PSC(CC)	PSC(GC) ∪ PSC(AC)	PSC(SY) ∪ PSC(AC)	PSC(EC)
PRC	PRC(CC) ∪ PRC(AC)	PRC(CC)	PRC(GC) ∪ PRC(AC)	PRC(SY) ∪ PRC(AC)	PRC(EC)

Niech Esynt będzie następującym zbiorem konstrukcji syntaktycznych

$$Esynt = Ccm \cup Gcc \cup Ecom \cup Eproc \cup Eprog .$$

Definicja 3.4. Zbiór konfiguracji programów jest określony następująco:

$$\Sigma = \{\langle \rho, s, c\rangle \mid \rho \in Esynt,\ s \in States,\ c \in Clocks\} \cup \{\langle s, c\rangle \mid s \in States,\ C \in Clocks\} \cup \{abortion\},$$

gdzie States jest zbiorem funkcji częściowych - stanów zmiennych programu - postaci s: Varid $\cdot\!\!\rightarrow$ Val; Clocks jest zbiorem funkcji - stanów zegarów programu - postaci c: Clockid $\rightarrow$ Int. Zbiór konfiguracji końcowych programu jest określony przez:

$$\Sigma_f = \{\langle s, c\rangle \mid s \in States,\ c \in Clocks\} \cup \{abortion\}$$ ■

Przejście z konfiguracji $\langle \rho, s, c\rangle$ do konfiguracji $\langle \rho', s', c'\rangle$ oznacza realizację pewnego elementarnego kroku obliczeń programu, w którym ρ oznacza instrukcję jaką ma program wykonać przed rozpoczęciem tego kroku, natomiast ρ' - instrukcję jaka pozostaje jeszcze do wykonania po zakończeniu obliczeń tego kroku; podobnie s, c oraz s', c' oznaczają początkowe i końcowe stany zmiennych oraz zegarów programu podczas realizacji tego kroku. Gdy wykonywany krok jest ostatnim krokiem obliczeń programu, wtedy po jego zakończeniu mogą powstać dwie sytuacje. Sytuacja po prawidłowym zakończeniu będzie opisywana konfiguracją $\langle s', c'\rangle$, natomiast po nieprawidłowym - konfiguracją abortion (zerwanie programu).

Wyjaśnienia wymaga przyjęta w dalszym ciągu konwencja dotycząca identyfikacji zegarów. Przyjmuje się, że każdy proces, czyli wszystkie jego instrukcje, dysponuje własnym czasomierzem, oznaczonym pewnym identyfikatorem, niedostępnym programiście, który będzie wyznaczony przez funkcję częściową cl: Esynt\Eprog$\cdot\!\!\rightarrow$ Clockid. Funkcja ta ma następujące własności:

1. Każdemu procesowi EP i wszystkim jego instrukcjom składowym odpowiada ten sam zegar oznaczony przez cl EP. Zatem jeżeli np. EP::=P:PD;AC|P:PD;SY;AC , to cl EP = cl AC = cl SY, itd.

2. Różnym procesom EP_i, EP_j w programie $EG = \underset{k=1..n}{\|} EP_k$ odpowiadają różne zegary, tzn. cl $EP_i \neq$ cl EP_j dla $i \neq j$.

Niech $\sigma = \langle \rho, s, c\rangle$ będzie konfiguracją taką, że $\rho \in Eprog$, tzn. $\rho = \underset{i=1..n}{\|} EP_i$, gdzie

$$EP_i = P_i : PD_i; BS_i$$

lub

$$EP_i = P_i : PD_i; BS_i; AC_i$$

oraz

$$BS_i = v_i \ \underline{wt} \ \square_{l=1..L_i} \ b_{il}; CM_{il} \longrightarrow AC_{il} \ \underline{lt} \ AC_{iO}.$$

Dalej, niech $c_i = c(\underline{cl}\ EP_i)$.

W formułowanych niżej definicjach 3.5-3.11 przyjmuje się ustaloną powyżej postać konfiguracji oraz oznaczenia jej elementów składowych.

Definicja 3.5. Proces EP_i (i=1..n) jest otwarty w stanie s do komunikacji z procesem o nazwie P wtedy i tylko wtedy, gdy istnieją b_{il}, CM_{il} takie, że $[\![b_{il}]\!]_s = tt$ oraz $CM_{il} = P!\ W(x)$ lub $CM_{il} = P?\ W(e)$. Fakt ten będzie zapisywany w postaci prawdziwości predykatu

$$[\![OPEN(EP_i,P)]\!]_s = tt. \qquad \blacksquare$$

Definicja 3.6. Procesy EP_i, EP_j (i,j=1..n) są otwarte w stanie s do komunikacji wtedy i tylko wtedy, gdy istnieje w BS_i, BS_j taka para syntaktycznie skojarzonych instrukcji komunikacji CM_{il}, $CM_{jl'}$, dla których

$$[\![b_{il}]\!]_s = tt \quad \text{oraz} \quad [\![b_{jl'}]\!]_s = tt.$$

Fakt ten będzie zapisywany w postaci prawdziwości predykatu

$$[\![COM\_OPEN\ (EP_i,\ EP_j)]\!]_s = tt. \qquad \blacksquare$$

Definicja 3.7. Procesy EP_i, EP_j mogą synchronizować się ze sobą w stanie programu s i stanie zegarów c wtedy i tylko wtedy, gdy zachodzą następujące warunki:

1) $[\![COM\_OPEN\ (EP_i,\ EP_j)]\!]_s = tt$,

2) $\max(c_i, c_j) \leqslant \min(c_i + v_i,\ c_j + v_j)$.

Fakt ten będzie zapisywany w postaci prawdziwości predykatu

$$[\![POSYN\ (EP_i,\ EP_j)]\!]_{s,c} = tt\ . \qquad \blacksquare$$

Definicja 3.8. Proces EP_i nie może synchronizować się w stanie programu s i stanie zegarów c, wtedy i tylko wtedy, gdy dla każdego $EP_j (j \neq i)$ zachodzi

$$[\![POSYN\ (EP_i,\ EP_j)]\!]_{s,c} = ff.$$

Fakt ten będzie zapisywany w postaci prawdziwości predykatu

$$[\![NOPOS\ (EP_i)]\!]_{s,c} = tt\ . \qquad \blacksquare$$

Definicja 3.9. Procesy EP_i, EP_j, takie że $[\![POSYN\ (EP_i, EP_j)]\!]_{s,c} = tt$, mogą synchronizować się ze sobą w konfiguracji σ wtedy i tylko wtedy, gdy są spełnione warunki:

1) dla każdej innej pary procesów EP_k, EP_l, różnej od pary EP_i, EP_j takiej, że $[\![POSYN(EP_k, EP_l)]\!]_{s,c} = tt$, zachodzi:

$$\max (c_i, c_j) \leqslant \max (c_k, c_l),$$

2) dla każdego procesu EP_k, różnego od EP_i, EP_j i takiego, że $[\![NOPOS\ (EP_k)]\!]_{s,c} = tt$ zachodzi:

$$\max (c_i, c_j) \leqslant c_k + v_k ,$$

3) dla pozostałych procesów EP_k

$$\max (c_i, c_j) \leqslant c_k .$$

Fakt ten będzie zapisywany w postaci prawdziwości predykatu

$$[\![SYNCH\ (EP_i, EP_j)]\!]_\sigma = tt .$$

■

Definicja 3.10. Proces EP_i, taki że $[\![NOPOS\ (EP_i)]\!]_{s,c} = tt$, nie może synchronizować się w konfiguracji σ wtedy i tylko wtedy, gdy są spełnione następujące warunki:

1) dla każdej pary procesów EP_k, EP_l różnych od EP_i, takiej że $[\![POSYN\ (EP_k, EP_l)]\!]_{s,c} = tt$, zachodzi $c_i + v_i < \max (c_k, c_l)$,

2) dla każdego procesu EP_j, różnego od EP_i, takiego że $[\![NOPOS\ (EP_j)]\!]_{s,c} = tt$, zachodzi: $c_i + v_i \leqslant c_j + v_j$,

3) dla pozostałych procesów EP_j zachodzi: $c_i + v_i < c_j$.

Fakt ten będzie zapisywany w postaci prawdziwości predykatu: $[\![NOSYN\ (EP_i)]\!]_\sigma = tt$.

Niech $\sigma = \langle \rho, s, c \rangle$ będzie teraz dowolną konfiguracją, taką że $\rho \in Eprog$.

Definicja 3.11. Procesy EP_i, EP_j są skojarzone ze sobą dynamicznie w konfiguracji σ wtedy i tylko wtedy, gdy są spełnione następujące warunki:

1) procesy EP_i, EP_j są postaci $EP_i = P_i : PD_i ; AS_i$ lub $EP_i = P_i : PD_i ; AS_i ; AC_i$, $EP_j = P_j : PD_j ; AS_j$ lub $EP_j = P_j : PD_j ; AS_j ; AC_j$, gdzie: $AS_i = CC_i$ syn P_j, $AS_j = CC_j$ syn P_i,

2) $c_i = c_j$.

Fakt ten będzie się zapisywać w postaci prawdziwości predykatu: $[\![MATCH\ (EP_i, EP_j)]\!]_\sigma = tt$.

■

3.4. Oddziaływania międzyprocesowe

Niech będą ustalone dwie, syntaktycznie skojarzone, instrukcje wejścia/wyjścia

P? W(x) oraz Q! W(e),

dla których, w pewnym momencie czasu t, zostanie stworzona taka sytuacja, że mogą one jednocześnie wykonać się, tzn. wartość v wyrażenia e nadana przez instrukcję wyjścia zostanie podstawiona pod zmienną x w instrukcji wejścia. Informację, jaka jest podczas tej realizacji obserwowalna przez instrukcje wejścia, można ująć w szóstkę:

(*, P, W, ?, v, t),

co oznacza, że w momencie t wartość v przesłana od procesu P została odebrana w porcie W procesu, w którym jest umieszczona instrukcja wejścia; znak * symbolizuje nieznaną dla instrukcji wejścia nazwę tego procesu. Podobnie to, co można zaobserwować z punktu widzenia instrukcji wyjścia, będzie zestawem informacji:

(*, Q, W, !, v, t).

Gdy z instrukcjami wejścia/wyjścia są jawnie związane identyfikatory procesów zawierających te instrukcje, wtedy zamiast symbolu * wystąpi identyfikator tego procesu; a więc odpowiednie oddziaływanie przyjmie postać

(Q, P, W, ?, v, t)

oraz

(P, Q, W, !, v, t) .

Podobne oddziaływania są informacjami obserwowalnymi z punktu widzenia poszczególnych procesów. Z punktu widzenia obserwatora zewnętrznego w stosunku do tych procesów będą dostrzegane te same informacje. Będą one natomiast zapisywane w bardziej zwartej postaci. Jeżeli procesy P, Q jednocześnie dostrzegają **zachodzące oddziaływania, co oznaczamy**

(P, Q, W, ?, v, t) || (Q, P, W, !, v, t) ,

to z punktu widzenia zewnętrznego obserwatora jest to równoważne z oddziaływaniem

(P, Q, W, v, t) ,

gdzie proces P stojący na pierwszym miejscu oznacza zawsze nadawcę, natomiast proces Q - na drugim miejscu - odbiorcę komunikatu.

Niech teraz będzie ustalona pewna instrukcja uwarunkowanej czasowo komunikacji. Podczas jej realizacji można zaobserwować dwie sytuacje, z których jedna polega na tym, że w okresie odcinka czasu wyznaczonego przez czas przeterminowania nastąpi zsynchronizowanie się z pewnym procesem oraz druga, gdy w tym odcinku czasu nie nastąpi synchronizacja.

W pierwszym przypadku informacją obserwowalną przez dany proces jest

$$(*, P, t)$$

lub

$$(Q, P, t),$$

jeśli identyfikator procesu Q jest jawnie związany z instrukcją uwarunkowanej czasowo komunikacji. Symbol P oznacza identyfikator procesu, z którym nastąpiła synchronizacja, natomiast t oznacza chwilę zsynchronizowania się obu procesów.

W drugim przypadku, jeżeli proces, oczekując do pewnej chwili t, nie doczeka się zsynchronizowania się z żadnym innym procesem, to obserwację zarejestrowaną przez proces będzie się zapisywać w postaci oddziaływania:

$$(*, t)$$

lub

$$(P, t),$$

jeśli identyfikator procesu P jest jawnie związany z instrukcją uwarunkowanej czasowo komunikacji.

W przypadku oddziaływań (P, Q, t) i (Q, P, t) ich jednoczesne wystąpienie

$$(P, Q, t) \,||\, (Q, P, t)$$

jest, oczywiście, równoważne każdemu z nich.

Definicja 3.12. Zbiór oddziaływań międzyprocesowych Λ_{loc}, obserwowalnych lokalnie przez pojedynczy proces jest określony następująco:

$$\begin{aligned}\Lambda_{loc} = {} & \{(P,Q,W,?,v,t) \mid P \in Procid \cup \{*\},\ Q \in Procid,\\ & W \in Portid,\ v \in Val,\ t \in Int\} \cup\\ & \{(P,Q,W,!,v,t) \mid P \in Procid \cup \{*\},\ Q \in Procid,\\ & W \in Portid,\ v \in Val,\ t \in Int\} \cup\\ & \{(P,Q,T) \mid P \in Procid \cup \{*\},\ Q \in Procid,\ t \in Int\} \cup\\ & \{(P,t) \mid P \in Procid \cup \{*\},\ t \in Int\} \cup \{\varepsilon\},\end{aligned}$$

gdzie ε jest oddziaływaniem pustym. ■

Definicja 3.13. Oddziaływaniem komplementarnym $\bar{\lambda}$ do danego oddziaływania $\lambda \in \Lambda_{loc}$ jest oddziaływanie następujące:

$$\bar{\lambda} = \begin{cases} (P,Q,W,?,v,t) & \text{dla } \lambda = (Q,P,W,!,V,t), \\ (P,Q,W,!,v,t) & \text{dla } \lambda = (Q,P,W,?,v,t), \\ (P,Q,t) & \text{dla } \lambda = (Q,P,t), \\ \text{niezdefiniowane} & \text{dla pozostałych } \lambda. \end{cases}$$

■

Definicja 3.14. Zbiór oddziaływań międzyprocesowych Λ_{glob}, obserwowalnych globalnie w programie jest określony następująco:

$$\Lambda_{glob} = \{(P,Q,W,v,t) \mid P,Q \in Procid,\ W \in Portid,\ v \in Val,\ t \in Int\} \cup \{(P,Q,t) \mid P,Q \in Procid,\ t \in Int\} \cup \{(P,t) \mid P \in Procid,\ t \in Int\} \cup \{\varepsilon\}.$$

■

Definicja 3.15. Operacja złożenia równoległego $\lambda \| \bar{\lambda} \in \Lambda_{glob}$ jest zdefiniowana dla oddziaływań komplementarnych λ, $\bar{\lambda} \in \Lambda_{loc}$ w sposób następujący:

$$\lambda \| \bar{\lambda} = \begin{cases} (P,Q,W,v,t) & \text{dla } \lambda = (P,Q,W,!,v,t) \\ & \text{lub } \lambda = (Q,P,W,?,v,t), \\ (P,Q,t) & \text{dla } \lambda = (P,Q,t) \\ & \text{lub } \lambda = (Q,P,t), \\ \text{niezdefiniowane} & \text{dla pozostałych } \lambda. \end{cases}$$

■

Warto zauważyć, że każda para lokalnych oddziaływań komplementarnych z Λ_{loc} wyznacza pewne oddziaływanie globalne, a także odwrotnie: każde globalne oddziaływanie z Λ_{glob} wyznacza parę lokalnych oddziaływań komplementarnych.

Dla wygody w operowaniu poszczególnymi elementami składowymi oddziaływań wprowadza się następującą konwencję oznaczeń. Dla $\lambda \in \Lambda_{loc}$:

- λ.obs - oznacza, jeśli jest określony, identyfikator procesu obserwującego oddziaływanie λ,
- λ.part - oznacza, jeśli jest określony, identyfikator procesu skojarzonego z procesem obserwującym λ,
- λ.port - oznacza, jeśli jest określony, identyfikator portu komunikacyjnego w λ,
- λ.dir - oznacza, jeśli jest określony, kierunek transmisji względem procesu obserwującego oddziaływanie λ; jest to nadawanie (!) lub odbiór (?)
- λ.val - oznacza, jeśli jest określona, wartość przesyłaną pomiędzy procesami w oddziaływaniu λ,
- λ.time - oznacza, jeśli jest określony, czas oddziaływania λ.

Ponadto dla $\lambda \in \Lambda_{glob}$:

λ.send - oznacza, jeśli jest określony, identyfikator procesu nadającego komunikat,

λ.receiv - oznacza, jeśli jest określony, identyfikator procesu odbierającego komunikat.

Dla rozpoznania rodzaju oddziaływania $\lambda \in \Lambda_{loc} \cup \Lambda_{glob}$ zapisuje się:

$$\lambda.type = \begin{cases} comm & gdy \ \lambda = (P,Q,W,!,v,t) \\ & \quad \lambda = (P,Q,W,?,v,t) \\ & \quad \lambda = (P,Q,W,v,t), \\ syn & gdy \ \lambda = (P,Q,t), \\ nosyn & gdy \ \lambda = (P,t), \\ empty & gdy \ \lambda = \varepsilon. \end{cases}$$

Podczas obliczeń programu $PR = \underset{i=1..n}{||} PC_i$ każdy z procesów składowych PC_i (i=1..n) obserwuje lokalnie pewien ciąg oddziaływań $hloc_i = \lambda_{i1}, \lambda_{i2}, \ldots, \lambda_{i,n(i)}$, gdzie $\lambda_{ij} \in \Lambda_{loc}$ dla $j = 1...n(i)$, zwany lokalną historią.

Definicja 3.16. Rodzina lokalnych historii $hloc_i$ (i=1..n) jest rodziną zgodną wtedy i tylko wtedy, gdy spełnia następujące warunki:

1) jeżeli

$$hloc_i = \lambda_{i1}, \lambda_{i2}, \ldots, \lambda_{i,n(i)},$$

to

$$\lambda_{i,j}.time \leqslant \lambda_{i,j+1}.time \quad (j=1..n(i)-1),$$

2) jeżeli λ będące elementem historii $hloc_i$ jest takim oddziaływaniem, dla którego jest określone oddziaływanie komplementarne $\bar{\lambda}$, to oddziaływanie $\bar{\lambda}$ jest elementem historii $hloc_j$ procesu P_j takiego, że $P_j = \lambda.part$. ■

Podczas obliczeń programu $PR = \underset{i=1..n}{||} PC_i$ można prowadzić obserwację zachodzących oddziaływań z punktu widzenia obserwatora zewnętrznego. Zarejestrowany ciąg oddziaływań

$$hglob = \lambda_1, \lambda_2, \ldots, \lambda_N,$$

gdzie $\lambda_j \in \Lambda_{glob}$, dla j=1..N, nazywa się historią globalną. Oczywiście

$$\lambda_{ij}.time \leqslant \lambda_{i,j+1}.time$$

dla j=1..N-1.

Zgodny zbiór historii lokalnych $hloc_1, \ldots, hloc_n$ wyznacza w oczywisty sposób pewną historią globalną hglob, oraz odwrotnie: globalna historia hglob wyznacza zgodny zbiór historii lokalnych. Zwią-

zek ten będzie notowany w postaci

$$hglob = \prod_{i=1..n}^{||} hloc_i .$$

Definicja 3.17. Jeżeli $h_1, \ldots, h_n$ jest ciągiem historii lokalnych (globalnych) w procesie (programie), to ciąg ten będzie nazywany łańcuchem w procesie (programie), jeżeli istnieją takie $\lambda_i \in \Lambda_{loc} \setminus \{\varepsilon\}(\Lambda_{glob} \setminus \{\varepsilon\})$, że

$$h_{i+1} = h_i \cap \lambda_i \qquad (i=1,\ldots,n-1)$$

gdzie symbol $\cap$ oznacza konkatenację.
Jeżeli h_i, h_j są elementami pewnego łańcucha i $i < j$, to będzie to zapisywać się w postaci $h_i \prec h_j$. ■

3.5. Relacja zmiany konfiguracji

Relację zmiany konfiguracji $\longrightarrow$ opisuje zbiór aksjomatów lub reguł związanych z każdą produkcją syntaktyczną. Zespół reguł i aksjomatów jest dobrany tak, aby dla dowolnej konfiguracji $\sigma \in \Sigma \setminus \Sigma_f$ było możliwe wyznaczenie obliczenia rozpoczynającego się w tej konfiguracji.
W definicjach przedstawionych poniżej przyjmuje się następujące oznaczenia:

Zapis reguły w postaci

$$\frac{A \longrightarrow B_1 \mid \ldots \mid B_n}{A' \longrightarrow B_1' \mid \ldots \mid B_n'}$$

jest skróconą formą zapisu n reguł:

$$1.\ \frac{A \longrightarrow B_1}{A' \longrightarrow B_n'} \quad \ldots \quad n.\ \frac{A \longrightarrow B_n}{A' \longrightarrow B_n'}$$

Dla konstrukcji, w których występują warunki logiczne jako dozory, a więc dla GC, CC, IF, DO, TH jest zdefiniowany, predykat BOOL. Dla GC, i podobnie dla pozostałych konstrukcji, predykat BOOL jest określony następująco:

$$[\![BOOL\ (GC)]\!]_s = tt \Longleftrightarrow \exists_{i=1..n} ([\![b_i]\!]_s = tt),$$

gdzie $b_1, \ldots, b_n$ występują jako dozory, czyli

$$GC = \prod_{i=1..n}^{\square} b_i \longrightarrow AC_i .$$

Gcm	Instrukcje dozorowane
$GC = \Box_{i=1..n}\ b_i \longrightarrow AC_i$	$\dfrac{[\![b_i]\!]_s = tt,\ \langle AC_i,s,c\rangle \xrightarrow{\lambda} \sigma}{\langle GC,s,c\rangle \xrightarrow{\lambda} \sigma}$
Ccm	**Instrukcja komunikacyjna**
$SC = P!W(e)$	1. $\dfrac{[\![e]\!]_s=v,\ [\![SEND_e[t'/dur]]\!]_s=tt,\ t=c(\underline{cl}\ SC)}{\langle SC,s,c\rangle \xrightarrow{(\ast,P,W,!,v,t)} \langle s,c(t+t')/clSC)\rangle}$
	2. $\dfrac{[\![e]\!]_s = \underline{error}}{\langle SC,s,c\rangle \xrightarrow{\varepsilon} \underline{abortion}}$
$RC = P?W(x)$	$\dfrac{[\![e]\!]_s=v,\ [\![SEND_e[t'/dur]]\!]_s=tt,\ t=c(\underline{cl}\ RC)}{\langle RC,s,c\rangle \xrightarrow{(\ast,P,W,?,v,t)} \langle s(v/x),c((t+t')/\underline{cl}PC)\rangle}$
Gcc	**Dozorowane instrukcje komunikacyjne**
$CC = \Box_{i=1..n}\ b_i;CM_i \longrightarrow AC_i$	$\dfrac{[\![b_i]\!]_s = tt,\ \langle CM_i,s,c\rangle \xrightarrow{\lambda} \langle s',c'\rangle}{\langle CC,s,c\rangle \xrightarrow{\lambda} \langle AC_i,s',c'\rangle}$
Com	**Instrukcje**
$AC = \underline{skip}$	$\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle s,c\rangle$
$AC = \underline{abort}$	$\langle AC,s,c\rangle \xrightarrow{\varepsilon} \underline{abortion}$
$AC = x:=e$	1. $\dfrac{[\![e]\!]_s=v,\ [\![ASS_e[t/dur]]\!]_s=tt}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle s(v/x),c(c(\underline{cl}AC)+t/\underline{cl}AC)\rangle}$
	2. $\dfrac{[\![e]\!]_s = \underline{error}}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \underline{abortion}}$
$AC = \underline{rcl}\ x$	$\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle s(c(\underline{cl}AC)/x),c\rangle$
$AC = AC1;AC2$	$\dfrac{\langle AC1,s,c\rangle \xrightarrow{\varepsilon} \langle AC1',s',c'\rangle \mid \langle s',c'\rangle \mid \underline{abortion}}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle AC1',AC2',s',c'\rangle\ \langle AC2,s',c'\rangle \mid \underline{abortion}}$
$AC = \underline{if}\ CC\ \underline{end}$	1. $\dfrac{[\![BOOL(AC)]\!]_s=tt,\ [\![TEST_{AC}[t/dur]]\!]_s=tt}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle GC,s,c((c(\underline{cl}AC)+t)/\underline{cl}AC)\rangle}$

Com	Instrukcje
AC:= do GC end	2. $\dfrac{[\![BOOL(AC)]\!]_s \neq tt}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \underline{abortion}}$ 1. $\dfrac{[\![BOOL(AC)]\!]_s = tt,\ [\![TEST_{AC}[t/dur]]\!]_s = tt}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle GC;AC,s,c(c(\underline{cl}AC)/\underline{cl}AC)\rangle}$ 2. $\dfrac{[\![BOOL(AC)]\!]_s = ff,\ [\![TEST_{AC}[t/dur]]\!]_s = tt}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle s,c((c(\underline{cl}AC)+t/\underline{cl}AC)\rangle}$
AC = th v wt CC lt AC1 end	1. $\dfrac{[\![BOOL(AC)]\!]_s = tt,\ [\![TEST_{AC}[t/dur]]\!]_s = tt}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle v\ \underline{wt}CC\underline{lt}AC1,s,c((c(\underline{cl}AC)+t)/\underline{cl}AC)\rangle}$ 2. $\dfrac{[\![BOOL(AC)]\!]_s \neq tt}{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \underline{abortion}}$
Proc	**Procesy**
PC = P:PD;AC	$\dfrac{\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle AC',s',c'\rangle \mid \langle s',c'\rangle \mid \underline{abortion}}{\langle PC,s,c\rangle \xrightarrow{\varepsilon} \langle PC',s',c'\rangle \mid \langle s',c'\rangle \mid \underline{abortion}}$ gdzie $PC' = P:PD;AC'$
Prog	**Programy**
$PR = \underset{i=1..n}{\Vert} PC_i$	$\dfrac{\langle PC_i,s,c\rangle \xrightarrow{\varepsilon} \langle PC_i',s',c'\rangle \mid \langle s',c'\rangle \mid \underline{abortion}}{\langle PR,s,c\rangle \xrightarrow{\varepsilon} \langle PR',s',c'\rangle \mid \langle PR'',s',c'\rangle \mid \underline{abortion}}$ gdzie $PR' = \underset{k=1..n}{\Vert} PC_k'$ oraz $PC_k = PC_k'$ dla $k \neq i$, $PR'' = \underset{\substack{k=1..n \\ k\neq i}}{\Vert} PC_k$
Synch	**Konstrukcje synchronizujące**
SY = v wt CC lt AC	1. $\dfrac{[\![OPEN(SY,P)]\!]_s = tt,\ c(\underline{cl}SY) \leqslant t \leqslant c(\underline{cl}SY)+v}{\langle SY,s,c\rangle \xrightarrow{(*,P,t)} \langle CC\underline{syn}P,s,c(t/\underline{cl}SY)\rangle}$ 2. $\dfrac{t=c(\underline{cl}\ SY)}{\langle SY,s,c\rangle \xrightarrow{(*,t+v+1)} \langle AC,s,c((t+v+1)/\underline{cl}SY)\rangle}$

Synch	Konstrukcje synchronizujące
SY = CC <u>syn</u> P	$\dfrac{\langle CC,s,c\rangle \xrightarrow{\lambda} \sigma}{\langle SY,s,c\rangle \xrightarrow{\lambda} \sigma}$
Ecom	**Złożenia pomocnicze**
EC = SY;AC	$\dfrac{\langle SY,s,c\rangle \xrightarrow{\lambda} \langle SY',s',c'\rangle \mid \langle AC',s',c'\rangle}{\langle EC,s,c\rangle \xrightarrow{\lambda} \langle SY';AC,s',c'\rangle \mid \langle AC',AC,s',c'\rangle}$
EC = GC;AC	$\dfrac{\langle GC,s,c\rangle \xrightarrow{\varepsilon} \langle AC',s',c'\rangle}{\langle EC,s,c\rangle \xrightarrow{\varepsilon} \langle AC',AC,s',c'\rangle}$
Eproc	**Procesy pomocnicze**
EP =P:PD;EC	$\dfrac{\langle EC,s,c\rangle \xrightarrow{\lambda} \langle EC',s',c'\rangle \mid \langle AC,s',c'\rangle,\ \Psi_p(\lambda)\downarrow}{\langle EP,s,c\rangle \xrightarrow{\Psi_p(\lambda)} \langle P:PD;EC',s',c'\rangle \mid \langle P:PD;AC,s',c'\rangle}$ gdzie Ψ_p jest funkcją częściową $\Psi_p : \Delta_{loc}^{*} \to \Delta_{loc}$ zdefiniowaną następująco: $\Psi_p(\lambda) = \begin{cases} (P,Q,W,!,v,t) & \text{gdy } \lambda = (\mathrm{x},Q,W,!,v,t), \\ (P,Q,W,?,v,t) & \lambda = (\mathrm{x},Q,W,?,v,t), \\ (P,Q,t) & \lambda = (\mathrm{x},Q,t), \\ (P,t) & \lambda = (\mathrm{x},t), \\ \lambda & \lambda = \varepsilon, \end{cases}$ zaś $\Psi_p(\lambda)\downarrow$ oznacza, że Ψ_p jest określona dla λ.
Eprog	**Programy pomocnicze**
EG = $\|\!\|_{i=1..n}$ EP_i	1. $\dfrac{\langle EP_i,s,c\rangle \xrightarrow{\varepsilon} \langle EP_i',s',c'\rangle \mid \langle s',c'\rangle \mid \underline{\text{abortion}}}{\langle EG,s,c\rangle \xrightarrow{\varepsilon} \langle EG',s',c'\rangle \mid \langle EG'',s',c'\rangle \mid \underline{\text{abortion}}}$ gdzie $EG' = \|\!\|_{k=1..n} EP_k'$ oraz $EP_k' = EP_k$ dla $k \neq i$, $EG'' = \|\!\|_{\substack{k=1..n \\ k\neq i}} EP_k$. $[\![SYNCH(EP_i,EP_j)]\!]\ (EG,s,c) = tt,$

Eprog	Programy pomocnicze

$\langle EP_i, s, c\rangle \xrightarrow{\lambda} \langle EP_i', s, c'\rangle$,

$\langle EP_j, s, c\rangle \xrightarrow{\bar{\lambda}} \langle EP_j', s, c''\rangle$,

$t' = c'(\underline{cl} EP_i), t'' = c''(\underline{cl} EP_j)$,

2. $$\frac{\lambda.time = \max(c(\underline{cl}\ EP_i,\ c(\underline{cl}\ EP_j)}{\langle EG, s, c\rangle \xrightarrow{\lambda \| \bar{\lambda}} \langle EG', s, c(t'/\underline{cl} EP_i, t''/\underline{cl} EP_j)\rangle}$$

gdzie $EG' = \prod_{k=1..n}^{\|} EP_k'$ oraz $EP_k' = EP_k$ dla $k \neq i,j$.

$[\![MATCH(EP_i, EP_j)]\!]\ \langle EG, s, c\rangle = tt$,

$\langle EP_i, s, c\rangle \xrightarrow{\lambda} \langle EP_i', s', c'\rangle$,

$\langle EP_j, s, c\rangle \xrightarrow{\bar{\lambda}} \langle EP_j', s, c''\rangle$,

3. $$\frac{t' = c'(\underline{cl}\ EP_i),\ t'' = c''(\underline{cl}\ EP_j)}{\langle EG, s, c\rangle \xrightarrow{\lambda \| \bar{\lambda}} \langle EG', s', c(t'/\underline{cl} EP_i, t''/\underline{cl} EP_j)\rangle}$$

gdzie $EG' = \prod_{k=1..n}^{\|} EP_k'$ oraz $EP_k' = EP_k$ dla $k \neq i,j$.

$[\![NOSYN(EP_i)]\!]\ \langle EG, s, c\rangle = tt$,

$\langle EP_i, s, c\rangle \xrightarrow{\lambda} \langle EP_i, s', c'\rangle$,

4. $$\frac{\lambda.type = timeout}{\langle EG, s, c\rangle \xrightarrow{\lambda} \langle EG', s', c'\rangle}$$

gdzie $EG' = \prod_{k=1..n}^{\|} EP_k'$, $EP_k' = EP_k$ dla $k \neq i$.

Do przedstawionych aksjomatów i reguł definiujących relację zmiany konfiguracji $\longrightarrow$ można dołączyć następujące komentarze.

Instrukcje dozorowane. Relacja zmiany konfiguracji jest zdefiniowana tylko dla tych przypadków, gdy istnieje co najmniej jedna alternaty-

wa otwarta. Pierwszym krokiem realizacji instrukcji jest realizacja pierwszego kroku instrukcji w otwartej alternatywie. Instrukcja GC będzie realizowana w programie zawsze jako fragment instrukcji IF lub DO. W definicji relacji $\longrightarrow$ dla IF oraz DO są uwzględnione przypadki, gdy nie istnieją alternatywy otwarte. Gdy tak się dzieje, wówczas nigdy nie dochodzi do realizacji odpowiedniej instrukcji GC. Stąd wynika uzasadnienie obszaru określoności relacji $\longrightarrow$ dla instrukcji GC.

Instrukcje komunikacyjne. Wykonanie instrukcji wyjścia realizuje się pomyślnie, jeżeli jest określona wartość wyrażenia e w stanie s. Pomyślne wykonanie polega na realizacji pojedynczego kroku zakończonego przesłaniem wartości wyrażenia e do wskazanego portu W wybranego procesu P. Czas trwania kroku wyznacza predykat $SEND_e$. Wykonanie instrukcji wejścia przebiega analogicznie, przy czym czas jej realizacji zależy od wyrażenia e w procesie nadawczym.

Dozorowane instrukcje komunikacyjne. Relacja zmiany konfiguracji jest określona tylko wtedy, gdy istnieje alternatywa otwarta. Dla tego przypadku pierwszym krokiem realizacji instrukcji jest wykonanie dowolnej instrukcji komunikacji odpowiadającej otwartej alternatywie.

Instrukcje. Wykonanie instrukcji skip, abort, x:=e, rcl x oraz sekwencyjnego złożenia instrukcji nie wymaga komentarza.

Pierwszym krokiem wykonania instrukcji alternatywy jest obliczenie wartości dozorów. Czynność ta, nie zmieniając stanu programu, wprowadza tylko pewne opóźnienie przed wykonaniem wybranej otwartej alternatywy. Jeżeli alternatywa taka nie istnieje, następuje zerwanie obliczenia całej instrukcji (a w konsekwencji całego programu). Pierwszy krok realizacji instrukcji iteracji jest taki jak dla instrukcji alternatywy, z tą różnicą, że brak otwartej alternatywy kończy poprawnie całą instrukcję.

Podobnie do instrukcji alternatywy przebiega wykonanie pierwszego kroku uwarunkowanej czasowo instrukcji komunikacji. Jeżeli istnieją alternatywy otwarte, to krok ten kończy się rozpoczęciem oczekiwania na zsynchronizowanie się z pewnym procesem; w przypadku przeciwnym następuje zerwanie instrukcji.

Procesy. Dla procesów jako elementów zbioru Proc jest zdefiniowana relacja $\longrightarrow$ tylko dla tych przypadków, gdy oddziaływanie $\lambda = \varepsilon$. Elementarnymi krokami realizacji procesów są kroki odpowiadające realizacji treści procesów.

Programy. Programy jako elementy zbioru Prog mają również zdefiniowaną relację $\longrightarrow$ tylko dla tych przypadków, gdy $\lambda = \varepsilon$. Pierwszym krokiem realizacji takich programów jest dowolny pierwszy krok realizacji jednego z procesów składowych.

Konstrukcje synchronizujące. Są one wprowadzone po to, aby opisywać mechanizm synchronizacji procesów. Pierwsza z tych konstrukcji podaje zasady oczekiwania procesu na synchronizację z innym procesem. Jest ona wykonywana w pojedynczym kroku, podczas którego może nastąpić synchronizacja - oddziaływanie (ж,P,t), gdzie t mieści się w zadanym odcinku czasu oczekiwania, natomiast P jest procesem, z którym można skomunikować się poprzez jedną z otwartych alternatyw instrukcji CC - - bądź też nastąpi upływ odcinka czasu oczekiwania na synchronizację.

Wykonanie drugiej konstrukcji synchronizującej jest równoważne z takim wykonaniem dozorowanej instrukcji komunikacji, któremu towarzyszy oddziaływanie z procesem P.

Złożenia pomocnicze nie wymagają komentarzy.

Procesy pomocnicze. Wykonanie kroku obliczeniowego procesu pomocniczego jest równoważne z wykonaniem kroku treści tego procesu, z tym że nieznany - z punktu widzenia instrukcji - identyfikator procesu zostaje zastąpiony jawnie określonym identyfikatorem.

Programy pomocnicze. Reguła 1 stwierdza, że w przypadku oddziaływania pustego, krok realizacji pojedynczego procesu składowego pociąga za sobą wykonanie kroku całego programu. Reguła 2 określa warunki, w których może nastąpić synchronizacja dwóch procesów składowych, które oczekują się wzajemnie, a gdy zostaną spełnione, podaje zmianę konfiguracji całego programu. Zmiana konfiguracji programu jest, w tym przypadku, związana z równoczesną zmianą konfiguracji tych procesów, które zostały ze sobą zsynchronizowane. Warunek synchronizacji określa predykat $SYNCH(EP_i, EP_j)$. Z definicji tego predykatu (definicja 3.9) wynika, że EP_i oraz EP_j są takie, że λ oraz $\bar{\lambda}$ występujące w przesłankach reguły mają postać typu (P,Q,t), czyli są oddziaływaniami synchronizującymi. Reguła 3 odnosi się do sytuacji, gdy istnieją dwa dynamiczne skojarzone ze sobą procesy, tzn. spełniające predykat $MATCH(EP_i, EP_j)$. Z definicji predykatu (definicja 3.11) wynika, że λ oraz $\bar{\lambda}$ występujące w przesłankach reguły są oddziaływaniami typu (P,Q,W,!,v,t) lub (P,Q,W,?,v,t), czyli są oddziaływaniami komunikacyjnymi. Zatem zmiana konfiguracji całego programu polega na równoczesnym wykonaniu dwóch, jednakowo długo trwających kroków obliczeniowych w skojarzonym dynamicznie procesie. Reguła 4 dotyczy przypadku, gdy dany proces EP, oczekujący na synchronizację z innym procesem, nie ma możliwości jej uzyskania w zadanym odcinku czasu, tzn. gdy jest prawdziwy predykat $NOSYN(EP_i)$ (definicja 3.10). Realizacja kroku obliczeniowego procesu EP_i pociąga za sobą zmianę konfiguracji całego programu.

Na zakończenie tego punktu należy jeszcze dołączyć następującą uwagę. Podstawą tworzenia opisu semantyki języka RTCSP jest dynamiczny

schemat przejść (definicja 3.1)

$$(\Sigma, \Sigma_f, \Lambda, \rightarrow).$$

Zbiór konfiguracji Σ oraz konfiguracji terminalnych Σ_f określa definicja 3.4. Zbiór oddziaływań

$$\lambda = \Lambda_{loc} \cup \Lambda_{glob}$$

określają definicje 3.12 oraz 3.14. Relację zmiany konfiguracji definiuje zbiór aksjomatów i reguł przedstawiony w tym podrozdziale. W definicjach tych aksjomatów i reguł wykorzystuje się w istotny sposób cechy środowiska wykonawczego ENV (definicja 3.3). Środowisko wykonawcze ENV stanowi zatem niejawny parametr schematu przejść, w podobnym sensie jak zbiory wyrażeń Exp i Bexp.

3.6. Własności relacji zmiany konfiguracji

Przyjmując definicję relacji zmiany konfiguracji podaną w poprzednim podrozdz., należy być pewnym, że definicja nie pomija pewnych przypadków bądź nie prowadzi do sprzeczności. Przedstawione poniżej własności przekonują, że przypadki takie nie istnieją.

Niech Syn $\subset$ Esynt. Wprowadza się następujące oznaczenia podzbiorów konfiguracji

$$\Sigma_{Syn} = \{\langle \rho, s, c \rangle \mid \rho \in Syn,\ s \in States,\ c \in Clocks\},$$

$$\Sigma_{abort} = \{\underline{abortion}\},$$

$$\Sigma_{term} = \{\langle s, c \rangle \mid s \in States,\ c \in Clocks\}.$$

Lemat 3.1

Poniżej przedstawiony diagram pokazuje dopuszczalne przejścia pomiędzy podzbiorami konfiguracji. Oznaczenia występujących oddziaływań mają znaczenie następujące:

λ_s oznacza, że $\lambda \cdot type = syn$,

λ_{ns} oznacza, że $\lambda \cdot type = nosyn$,

λ_c oznacza, że $\lambda \cdot type = comm$.

■

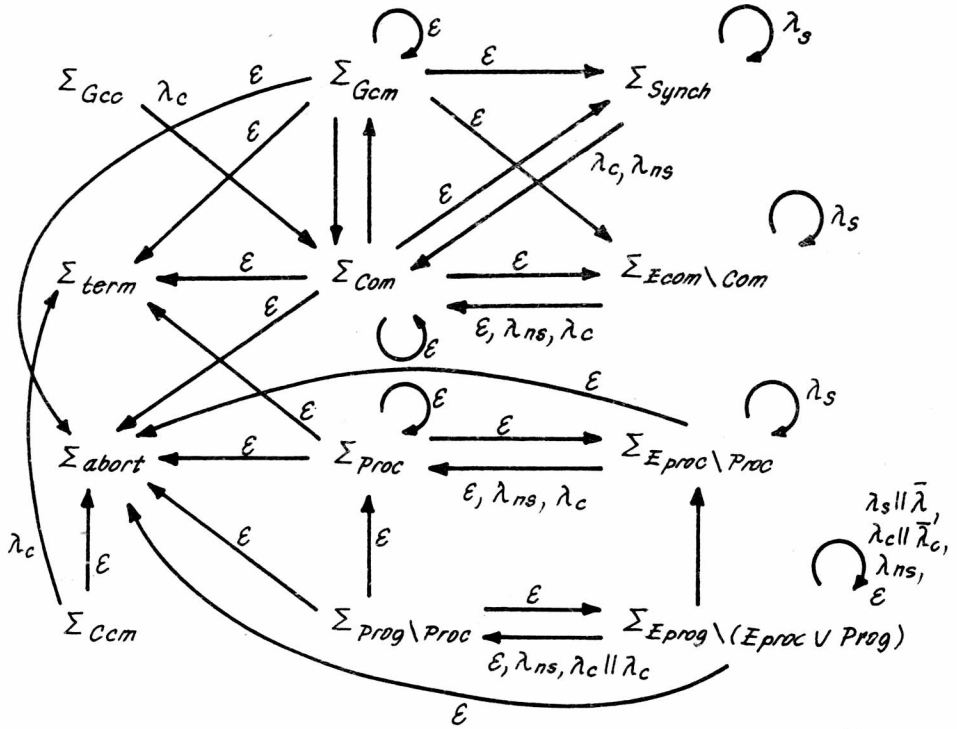
Dowód

Dowód jest przedstawiony w załączniku na końcu pracy.

Twierdzenie 3.1

Niech $\sigma = \langle \rho, s, c \rangle$, $\sigma' = \langle \rho', s', c' \rangle$ oraz $\sigma \xrightarrow{\lambda} \sigma'$ dla $\lambda \in \Lambda$. Jeżeli ρ jest poprawnie zbudowaną konstrukcją syntaktyczną, to także ρ' jest poprawnie zbudowaną konstrukcją syntaktyczną.

■



Rys. 3.1. Diagram przejść między podzbiorami konfiguracji
Fig. 3.1. Diagram of transitions between configuration subsets

Dowód

Dowód prowadzi się metodą indukcji strukturalnej.

1. Jeżeli ρ = AC1;AC2, to ρ' = AC1',AC2 | AC2. Z założenia $\vdash\rho$, zatem również $\vdash$AC1, $\vdash$AC2, a na mocy hipotezy indukcyjnej $\vdash$AC1'. Stąd na mocy reguły dla sekwencyjnego złożenia instrukcji $\vdash$AC1';AC2.

2. Jeżeli ρ = GC = $\square_{i=1..n} b_i \longrightarrow AC_i$, to ρ' = AC_i' (i=1..n). Ponieważ z założenia $\vdash$GC, stąd również $\vdash AC_i$, na mocy hipotezy indukcyjnej $\vdash AC_i'$, czyli $\vdash\rho'$.

3. Jeżeli ρ = CC lub ρ = IF, to analiza jest podobna do analizy p. 1.

4. Jeżeli ρ = DO = do GC end, to ρ' = GC;DO. Z założenia $\vdash$DO, a stąd $\vdash$GC. Na mocy reguły złożenia sekwencyjnego $\vdash$GC; DO.

5. Jeżeli ρ = TH = th t wt CC lt AC end, to ρ' = t wt CC lt AC. Z $\vdash$TH wynika, że $\vdash$CC oraz $\vdash$AC, a stąd, na mocy reguły poprawności budowy konstrukcji synchronizujących $\vdash$t wt CC lt AC.

6. Jeżeli ρ = PC = P : PD;AC, to ρ' = PC' = P:PD;AC', gdzie $\langle AC,s,c\rangle \xrightarrow{\varepsilon} \langle AC',s',c'\rangle$. Z założenia $\vdash$PC, stąd $\vdash$AC, a z p. 1, 3, 4, 5 wynika, że $\vdash$AC'. Ponieważ PSC(AC') $\leqslant$ PSC(AC), PRS(AC') $\leqslant$ PRS(AC), zatem zachodzi reguła poprawności budowy dla PC', czyli $\vdash\rho'$.

7. Jeżeli $\rho = PR = \|_{i=1..n} PC_i$, to $\rho' = PR' = \|_{i=1..n} PC_i'$, gdzie $PC_i' = PC_i$ dla $i \neq k$ oraz $PC_k' = P_k : PD; AC_k$ i $\langle AC_k, s, c\rangle \xrightarrow{\varepsilon} \langle AC_k', s', c'\rangle$ albo $\rho' = PR' = \|_{\substack{i=1..n \\ i\neq k}} PC_i$.

W pierwszym przypadku (z analizy p. 6) wynika, że $\vdash PC_k'$, a ponieważ $FV(PC_k') \leq FV(PC_k)$, więc z reguły poprawności budowy procesów $\vdash PR'$.

W drugim przypadku ponieważ $\vdash PC_i$ dla $i \neq k$, i z założenia $\vdash PR$, więc $\vdash \|_{\substack{i=1..n \\ i\neq k}} PC_i$.

8. Jeżeli $\rho = SY = t$ wt CC lt AC, to $\rho' = CC$ syn P lub $\rho' = AC$, gdzie proces P jest taki, że $[\![OPEN(SY,P)]\!]_s = tt$. Ponieważ $\vdash SY$, więc również $\vdash CC$ oraz $\vdash AC$. Natomiast jeżeli dla pewnego stanu s prawdziwy jest predykat $OPEN(SY,P)$, to wynika stąd, na podstawie definicji tego predykatu, że istnieje W takie, że $(P,W) \in PSC(SY) \cup PRC(SY)$. Zatem $\vdash CC$ syn P.

9. Jeżeli $\rho = SY;AC$ lub $\rho = GC;AC$, to prawidłowość ρ' pokazuje się tak jak w p. 1.

10. Jeżeli $\rho = EP$, to prawidłowość ρ' pokazuje się tak jak w p. 6.

11. Jeżeli $\rho = EG$, to prawidłowość ρ' pokazuje się podobnie jak w p. 7, 8.

Lemat 3.2

Niech $\sigma = \langle \rho, s, c\rangle$ będzie konfiguracją, że $\rho = \|_{k=1..n} \rho_i$. Jeżeli ρ jest prawidłowo zbudowane oraz spełnione są warunki:

1) jeżeli $\rho_i = P_i : PD_i; AS_i$ lub $\rho_i = P_i : PD_i; AS_i; AC_i$, to istnieje ρ_j takie, że $\rho_j = P_j : PD_j; AS_j$ lub $\rho_j = PD_j : PD_j; AS_j; AC_j$ oraz $[\![MATCH(\rho_i,\rho_j)]\!]_\sigma = tt$;

2) jeżeli $\rho_i = P_i : PD_i;$ GC lub $\rho_i = P_i = PD_i;$ GC; AC, to $[\![BOOL(GC)]\!]_s = tt$.

to istnieją $\lambda \in \Lambda$ oraz $\sigma' \in \Sigma$ takie, że $\sigma \xrightarrow{\lambda} \sigma'$.

Dowód ■

Dowód jest przedstawiony w załączniku.

Lemat 3.3

Niech $\sigma = \langle \rho, s, c\rangle$ będzie konfiguracją taką, że ρ jest prawidłowo zbudowane i $\rho \in Prog$.

Jeżeli istnieje obliczenie takie, że $\sigma \longrightarrow^* \sigma'$, gdzie $\sigma' = \langle \rho', s', c'\rangle$, to:

1) jeżeli $\rho' = \|_{k=1..n} \rho_k'$ oraz pewne ρ_i' ma postać $P_i : PD_i; AS_i$ lub $P_i : PD_i; AS_i; AC_i'$, to istnieje dokładnie jedno ρ_j' takiej samej postaci oraz $[\![MATCH(\rho_i',\rho_j')]\!]_{\sigma'} = tt$,

2) jeżeli $\rho' = \|_{k=1..n} \rho'_k$ oraz pewne ρ'_i ma postać P_i: $PD_i;GC_i$ lub P_i: $PD_i;GC_i;AC_i$, to zachodzi $[\![BOOL(GC_i)]\!]_{s'} = tt$. ■

Dowód

Dowód jest przedstawiony w załączniku.

Z lematów 3.2, 3.3 wynika bezpośrednio twierdzenie

Twierdzenie 3.2

Obliczenie prawidłowo zbudowanego programu w języku RTCSP nie blokuje się (definicja 3.2). ■

Twierdzenie 3.3

Dla dowolnego obliczenia programu w języku RTCSP

$$\sigma_0 \xrightarrow{\lambda_1} \sigma_1 \xrightarrow{\lambda_2} \sigma_2 \xrightarrow{\lambda_3} \cdots$$

jeżeli λ_i, λ_j są takie, że $i < j$ oraz $\lambda_i.type$, $\lambda_j.type \in \{syn, nosyn\}$, to $\lambda_i.time \leq \lambda_j.time$. ■

Dowód

Prosty dowód prowadzony indukcyjnie ze względu na długość obliczenia z wykorzystaniem definicji 3.9 oraz 3.10 pomija się.

3.7. Przykład obliczenia

Niech będzie dany następujący prosty program złożony z dwóch procesów.

```
P : W;
    i:=1;                                                    ]A1
    s:=false;                                                ]A2
    do
     ¬s ∧ i ≤ 2 ⟶                                           ]
        th 3 wt                                              ]
          true; Q?W(x) ⟶                                     ]A3
                s:=true                                      ]A7
          lt                                                 ]A8
                i:=i+1                                       ]A6
        end                                                  ]A5
    end                                                      ]A4
```

```
||
Q : j:=1;                                                          ] B1
    q:=false;                                                      ] B2
    do
     ¬q ∧ j ≤ 2 →
        th 2 wt
          true; P!W(j) →                                  ] B6     ] B3
                q:=true                          ] B7     ]  B5 B4 ]
          lt
                j:=j+1                           ] B8     ]
        end
    end
```

Symbole umieszczone po prawej stronie opisu procesów służą do identyfikacji wskazywanych fragmentów tekstu. Będą one używane dalej w celu uzyskania zwartości zapisów. Dodatkowo używanymi symbolami będą

A5 ^{syn} na oznaczenie 2 **wt** A6 **lt** A8,

B5 ^{syn} na oznaczenie 2 **wt** B6 **lt** B8,

A6 ^{Q} na oznaczenie A6 **syn** Q,

B6 ^{P} na oznaczenie B6 **syn** P.

Przykładowy program będzie wykonywany w środowisku ENV, którego własności dynamiczne określa następująco zdefiniowany zbiór predykatów

$$ASS_e(dur) \Longleftrightarrow (dur=1 \vee dur=2)$$

$$TEST_{DO}(dur) \Longleftrightarrow TEST_{TH}(dur) \Longleftrightarrow (dur=0 \vee dur=1)$$

$$SEND_e(dur) \Longleftrightarrow (dur=2 \vee dur=3)$$

dla dowolnych $e \in Exp$; $DO, TH \in Com$.

Niech stan początkowy programu będzie nieokreślony, a początkową chwilą rozpoczęcia obu procesów będzie chwila 0. Poniżej przedstawia się pewne obliczenie tego programu.

1. $\langle P{:}W;A1;A2;A3 \,||\, Q{:}B1;B2;B3,(\,),(0,0)\rangle$
   $\downarrow \varepsilon$
2. $\langle P{:}W;A2;A3 \,||\, Q{:}B1;B2;B3,(i=1),(2,0)\rangle$
   $\downarrow \varepsilon$
3. $\langle P{:}W;A3 \,||\, Q{:}B1;B2;B3,(i=1,\ s=false),(4,0)\rangle$
   $\downarrow \varepsilon$
4. $\langle P{:}W;A4;A3 \,||\, Q{:}B1;B2;B3,(i=1,\ s=false),(5,0)\rangle$
   $\downarrow \varepsilon$
5. $\langle P{:}W;A5^{syn};A3 \,||\, Q{:}B1;B2;B3,(i=1,\ s=false),(6,0)\rangle$
   $\downarrow \varepsilon$

6. $\langle P{:}W;A5^{syn};A3 \,\|\, Q{:}B2;B3,(i{=}1,\ j{=}1,\ s{=}false),(6,1)\rangle$
   $\downarrow \varepsilon$
7. $\langle P{:}W;A5^{syn};A3 \,\|\, Q{:}B3,(i{=}1,\ j{=}1,\ s{=}false,\ q{=}false),(6,2)\rangle$
   $\downarrow \varepsilon$
8. $\langle P{:}W;A5^{syn};A3 \,\|\, Q{:}B4;B3,(i{=}1,\ j{=}1,\ s{=}false,\ q{=}false),(6,2)\rangle$
   $\downarrow \varepsilon$
9. $\langle P{:}W;A5^{syn};A3 \,\|\, Q{:}B5^{syn};B3,(i{=}1,\ j{=}1,\ s{=}false,\ q{=}false),(6.2)\rangle$
   $\downarrow (Q,4)$
10. $\langle P{:}W;A5^{syn};A3 \,\|\, Q{:}B8;B3,(i{=}1,\ j{=}1,\ s{=}false,\ q{=}false),(6,4)\rangle$
   $\downarrow \varepsilon$
11. $\langle P{:}W;A5^{syn};A3 \,\|\, Q{:}B3,(i{=}1,\ j{=}2,\ s{=}false,\ q{=}false),(6,5)\rangle$
   $\downarrow \varepsilon$
12. $\langle P{:}W;A5^{syn};A3 \,\|\, Q{:}B4,B3,(i{=}1,\ j{=}2,\ s{=}false,\ q{=}false),(6,5)\rangle$
   $\downarrow \varepsilon$
13. $\langle P{:}W;A5^{syn};A3 \,\|\, Q{:}B5^{syn};B3,(i{=}1,\ j{=}2,\ s{=}false,\ q{=}false),(6,5)\rangle$
   $\downarrow (P,Q,7) \,\|\, (Q,P,7)$
14. $\langle P{:}W;A6\underline{syn}Q;A3 \,\|\, Q{:}B6\underline{syn}P;B3,(i{=}1,\ j{=}2,\ s{=}false,\ q{=}false),(7,7)\rangle$
   $\downarrow (P,Q,W,?,2,7)\quad (Q,P,!,W,2,7)$
15. $\langle P{:}W;A7;A3 \,\|\, Q{:}B7;B3,(i{=}1,\ j{=}2,\ x{=}2,\ s{=}false,\ q{=}false),(9,9)\rangle$
   $\downarrow \varepsilon$
16. $\langle P{:}W;A3 \,\|\, Q{:}B7;B3,(i{=}1,\ j{=}2,\ x{=}2,\ s{=}true,\ q{=}false),(11,9)\rangle$
   $\downarrow \varepsilon$
17. $\langle Q{:}B7;B3,(i{=}1,\ j{=}2,\ x{=}2,\ s{=}true,\ q{=}false),(12,9)\rangle$
   $\downarrow \varepsilon$
18. $\langle Q{:}B3,(i{=}1,\ j{=}2,\ x{=}2,\ s{=}true,\ q{=}true),(12,10)\rangle$
   $\downarrow \varepsilon$
19. $\langle (i{=}1,\ j{=}2,\ x{=}2,\ s{=}true,\ q{=}true),(12,10)\rangle$

Obliczenie, jak łatwo sprawdzić, było tworzone zgodnie z zasadą, że każdy elementarny krok obliczeniowy w procesie P jest wykonywany w najdłuższym dopuszczalnym odcinku czasu, natomiast krok elementarny w Q - w najkrótszym. Obliczenie zawiera wszystkie możliwe rodzaje oddziaływań między procesami. Relacja zmiany konfiguracji umożliwia quasi-równoległą symulację współbieżnej pracy procesów. Konfiguracje 1, ..., 5 ilustrują postęp obliczeń procesu P. Warto zwrócić uwagę, że w konfiguracji 5 nie jest możliwy dalszy postęp obliczeń tego procesu, konieczne jest oczekiwanie na odpowiedni postęp obliczeń procesu Q. Dalszy postęp obliczeń procesu P staje się możliwy dopiero w konfiguracji 13, gdy następuje synchronizacja obu procesów. Natomiast w konfiguracji 9 możliwy jest postęp obliczeń procesu Q, gdyż wiadomo, że nie zsynchronizuje się on z procesem P. W konfiguracji 13 następuje zsynchronizowanie procesów, natomiast w 14 przesłanie odpowiedniej informacji. Dalsze konfiguracje są końcową fazą obliczenia, które kończy się właściwie w konfiguracji 19.

4. DOWODZENIE POPRAWNOŚCI PROGRAMÓW

4.1. Wprowadzenie

Formalne dowodzenie własności programów czasu rzeczywistego jest w embrionalnej fazie. O ile, poczynając od końca lat siedemdziesiątych, pojawiło się wiele prac na temat dowodzenia programów współbieżnych i rozproszonych, o tyle w odniesieniu do programów czasu rzeczywistego literatura jest raczej skromna. Przykładem interesującej próby jest praca Taylora [202]. Jest ona pierwszą, znaną autorowi, próbą jawnego wprowadzenia czasu do predykatów orzekających o stanie programu. Przyczyną tego stanu jest to, że nie zostały jeszcze ugruntowane formalne metody opisu semantyki takich programów, bowiem tylko w przypadku istnienia dobrze zdefiniowanej semantyki metodą operacyjną lub denotacyjną można wprowadzić sensowny logicznie system dowodzenia własności programów. Przedstawiony dalej system dowodzenia własności programów w języku RTCSP wykorzystuje cechy różnych systemów dowodzenia zbudowanych dla różnych wariantów języka CSP Hoare'a, a szczególnie podejście zaproponowane przez Levina [136], [137]. Wykorzystuje się również cechy systemów dowodzenia budowanych dla języków z wbudowanymi mechanizmami niedeterministycznymi - przegląd problemów dowodzenia tej klasy programów zawiera praca Apta [14].

Sformułowanie własności programu wymaga wprowadzenia języka specyfikacji. Tak jak we wszystkich niemal pracach, przyjmuje się, że dany jest pewien język teorii sformalizowanej pierwszego rzędu, nazywany ASSERT, który będzie służyć do wyrażania asercji użytych do specyfikacji. Przyjmuje się, że zbiór termów (Term) języka ASSERT zawiera zbiór wyrażeń Exp, a zbiór formuł (Form) tego języka zawiera zbiór wyrażeń boolowskich Bexp oraz zbiór predykatów określających środowisko wykonawcze programów ENV.

Zakłada się, że dziedzina interpretacji języka ASSERT zawiera zbiory wartości całkowitoliczbowych Int oraz wartości logicznych Log. Przyjmuje się również, że jest ustalona pewna interpretacja J języka ASSERT, która każdemu nielogicznemu symbolowi języka przypisuje pewną funkcję lub relację określoną na produkcie kartezjańskim odpowiedniej krotności nad dziedzinę interpretacji. Interpretacja języka ASSERT stanowi więc rozszerzanie interpretacji wyrażeń Exp, Bexp oraz predykatów ENV, o których była mowa w rozdziale 3.

Przez α, β, γ będą oznaczane formuły języka ASSERT. Jeśli ustalona jest interpretacja J i ustalony stan s zmiennych programu, praw-

dziwość formuły α będzie oznaczana przez $\models_J \alpha(s)$ lub $\models \alpha(s)$, gdy wiadomo z kontekstu, że chodzi o wcześniej ustaloną interpretację.

Przez formuły specyfikacyjne, krótko specyfikacje, będzie rozumieć się konstrukcje postaci

$$\{\alpha\}\ S\ \{\beta\}$$

gdzie α, β są asercjami - formułami języka ASSERT, natomiast S jest prawidłowo zbudowanym programem w języku RTCSP lub fragmentem takiego programu, tzn. $S \in Esynt$. Formuły specyfikacyjne będą oznaczane symbolem Φ z ewentualnymi indeksami. Zbiór formuł specyfikacyjnych stanowi pewien język specyfikacji, nazywany SPECIF.

Interpretacja formuł języka SPECIF jest pewnym rozszerzeniem interpretacji J przyjmowanej dla języka ASSERT, przy czym możliwe są różne rozszerzenia tej interpretacji, w zależności od rodzaju badanych własności programu. Poprawność programów współbieżnych może wiązać się z następującymi własnościami:

1. Dla dowolnego stanu początkowego programu S, spełniającego zadaną asercję α, jeżeli obliczenie programu zakończy się pomyślnie (nie nastąpi zerwanie, zapętlenie lub blokada obliczeń), to osiągnięty stan końcowy spełnia zadaną asercję β.

2. Dla dowolnego stanu początkowego programu S, spełniającego zadaną asercję α, obliczenie programu zakończy się (nie nastąpi zapętlenie).

3. Dla dowolnego stanu początkowego programu S, spełniającego zadaną asercję α, obliczenie programu nie zostanie zerwane.

4. Dla dowolnego stanu początkowego programu S, spełniającego zadaną asercję α, obliczenie programu nie zablokuje się.

Jeśli chodzi o programy współbieżne czasu rzeczywistego, można badać te same rodzaje własności, przy czym sformułowanie "dla dowolnego stanu początkowego programu" należy zastąpić sformułowaniem rozszerzonym "dla dowolnego stanu początkowego zmiennych i zegarów programu".

W zależności od wybranych własności używa się różnych pojęć poprawności programu. Program, który spełnia tylko własność 1 przyjmuje się określać programem częściowo poprawnym, program, który spełnia własności 1, 2 - programem słabo poprawnym, program, który spełnia własności 1, 2, 3 - programem silnie poprawnym, a program, który spełnia własności 1, 2, 3, 4 - programem całkowicie poprawnym.

Należy przypomnieć, że obliczenia w języku RTCSP nie blokują się, co oznacza, że programy w tym języku zawsze spełniają własność 4.

Zdefiniowanie pojęcia poprawności programu wymaga wprowadzenią semantyk pomocniczych

$$Sem_p[\![PR]\!](\langle s,c\rangle) = \{\langle s',c'\rangle \mid \langle PR,s,c\rangle \longrightarrow^{*} \langle s',c'\rangle\},$$

$$Sem_w[\![PR]\!](\langle s,c\rangle) = Sem_p[\![PR]\!](\langle s,c\rangle) \cup \{\underline{infinity} \mid \langle PR,s,c\rangle \longrightarrow^* \underline{infinity}\},$$

$$Sem_s[\![PR]\!](\langle s,c\rangle) = Sem_w[\![PR]\!](\langle s,c\rangle) \cup \{\underline{abortion} \mid \langle PR,s,c\rangle \longrightarrow^* \underline{abortion}\},$$

$$Sem_t[\![PR]\!](\langle s,c\rangle) = Sem_s[\![PR]\!](\langle s,c\rangle) \cup \{\underline{deadlock} \mid \langle PR,s,c\rangle \longrightarrow^* \underline{deadlock}\}.$$

Symbole abortion, infinity, deadlock będą tu traktowane jako specjalnie wyróżnione stany, dla których z definicji $\not\models_J \alpha(\underline{abortion})$, $\not\models_J \alpha(\underline{inifinity})$, $\not\models_J \alpha(\underline{deadlock})$, dla dowolnej asercji α i dowolnej interpretacji J. Przez $[\alpha]_J$ będzie oznaczony zbiór wszystkich stanów zmiennych i zegarów $\langle s,c\rangle$, które spełniają α w interpretacji J, tzn. $\models_J \alpha(\langle s,c\rangle)$. Więc z definicji

$$\underline{abortion} \notin [\alpha]_J, \quad \underline{infinity} \notin [\alpha]_J, \quad \underline{deadlock} \notin [\alpha]_J.$$

Niech dla $d \in \{p,w,s,t\}$

$$Sem_d[\![PR]\!]([\alpha]_J) = \bigcup_{\langle s,c\rangle \in [\alpha]_J} Sem_d[\![PR]\!](\langle s,c\rangle).$$

Definicja 4.1. Interpretacja formuł specyfikacyjnych $\{\alpha\}$ PR $\{\beta\}$ jest następująca:
Dla $d \in \{p,w,s,t\}$

$$\models_{J,d} \{\alpha\} PR \{\beta\}$$

wtedy i tylko wtedy, gdy

$$Sem_d[\![PR]\!]([\alpha]_J) \subseteq [\beta]_J.$$

Oznacza to, że program jest częściowo poprawny, gdy

$$\models_{J,p} \{\alpha\} PR \{\beta\},$$

słabo poprawny, gdy

$$\models_{J,w} \{\alpha\} PR \{\beta\},$$

silnie poprawny, gdy

$$\models_{J,s} \{\alpha\} PR \{\beta\},$$

oraz całkowicie poprawny, gdy

$$\models_{J,t} \{\alpha\} PR \{\beta\}.$$

4.2. Własności systemów dowodzenia

Formalne dowodzenie d-poprawności programu polega na zbudowaniu odpowiedniego dowodu wyprowadzalnego w ramach pewnego systemu dedukcyjnego. Systemy takie, nazywane systemami dowodzenia odpowiedniego rodzaju poprawności programów, składają się ze skończonego zbioru aksjomatów A_d oraz ze skończonego zbioru reguł R_d, gdzie $d \in \{p,w,s,t\}$. Każdy z aksjomatów wyznacza pewną klasę formuł specyfikacyjnych - elementów języka SPECIF, które uznaje się za d-poprawne. Natomiast pojedyncza reguła na podstawie pewnego zestawu przesłanek - są nimi d-poprawne formuły specyfikacyjne lub prawdziwe asercje języka ASSERT - pozwala rozstrzygać o poprawności pewnej formuły specyfikacyjnej będącej wnioskiem reguły. Zatem, jeśli interpretacja J jest ustalona, system PS_d dowodzenia d-poprawności programów można traktować jako czwórkę

$$PS_d = (Tr_J, \text{SPECIF}, A_d, R_d) \qquad d \in \{p,w,s,t\},$$

gdzie Tr_J oznacza zbiór asercji prawdziwych w interpretacji J. Istotną cechą tak rozumianego systemu dowodzenia jest istnienie pewnego mechanizmu pomocniczego (wyroczni), który pozwala rostrzygać o prawdziwości dowolnej asercji języka ASSERT. Takie podejście, nazywane zabiegiem Cooka [51], pozwala skupić uwagę na samych zasadach tekstowej transformacji specyfikacji i pozostawić na boku cały aparat matematyki, który jest wykorzystany w konstrukcjach języka ASSERT i w badaniu ich prawdziwości. W dalszym ciągu przyjmuje się stałe założenie, że język ASSERT zawiera rachunek kwantyfikatorów i arytmetykę liczb całkowitych.

Ogólnie konstrukcja dowodu dla specyfikacji Φ w systemie dowodzenia PS polega na znalezieniu ciągu formuł $\Phi_1, \Phi_2, \ldots, \Phi_n$ takiego, że

1) Φ_n jest zadaną specyfikacją Φ,

2) dla $i = 1..n-1$ Φ_i jest aksjomatem bądź wnioskiem jednej z reguł zastosowanej do pewnego podciągu ciągu $\Phi_1, \Phi_2, \ldots, \Phi_{i-1}$, ewentualnie uzupełnionego pewnymi asercjami ze zbioru Tr_J jako przesłankami tej reguły. Fakt, że specyfikacja Φ ma dowód w systemie PS będzie zapisywany w postaci

$$\vdash_{PS} \Phi .$$

Systemy dowodzenia własności programów współbieżnych mają specyficzną cechę, powodującą że struktura dowodów odbiega od opisanej wyżej. Mianowicie wykorzystuje się, pochodzącą od Owickiej [171], ideę rozbicia procesu dowodzenia na dwa etapy. Pierwszy etap - dowodzenia sekwencyjnego - polega na zbudowaniu oddzielnych dowodów poprawności dla procesów składowych programu. Drugi etap - złożenia dowodów sekwencyjnych

- polega na sprawdzeniu "zgodności" dowodów sekwencyjnych i w przypadku zachodzenia takiej "zgodności", złożeniu ich w jeden wspólny dowód poprawności całego programu. Dowodząc poprawności pojedynczego procesu, nie można, bez znajomości zakresu współpracy z innymi procesami, rozstrzygnąć do końca, czy proces będzie działać poprawnie we wszystkich oczekiwanych okolicznościach. Dlatego dowody sekwencyjne pojedynczych procesów zawierają pewne "luki" wyrażające uzależnienie danego procesu od pozostałych procesów. "Luki" te są usuwane w drugim etapie dowodzenia, podczas składania dowodów sekwencyjnych w jeden wspólny dowód. Składanie takie dokonuje się na podstawie odpowiedniej reguły równoległego złożenia dowodów sekwencyjnych, która zwykle - w sensie logicznym - jest metaregułą.

Druga specyficzna cecha dowodzenia programów współbieżnych wiąże się z użyciem zmiennych pomocniczych. Zmienne pomocnicze, to takie zmienne, których obecność w programie nie zmienia przepływu sterowania oraz oryginalnego przepływu danych. Pojęcie zmiennych pomocniczych zostało wprowadzone przez Habermanna [79], a potem używane wielokrotnie, między innymi w systemach dowodzenia [8], [10], [47], [136], [137], [171], [172], [173], [183], [199]. Także w niniejszej pracy używa się zmiennych pomocniczych. Będą nimi tzw. zmienne historyczne służące do zapamiętywania lokalnych historii oddziaływań w procesach. Na zmiennych tych będą wykonywane proste operacje, np. konkatenacji historii, testowanie elementu historii itp. Istotny jest fakt, że pojedyncze oddziaływania, a także skończone ciągi oddziaływań można kodować liczbami całkowitymi [10], [12], [195]. Zatem wprowadzenie tego typu obiektów wraz z dowolnie określonymi operacjami wykonanymi na nich nie rozszerza przyjętego języka asercji ASSERT, gdyż zawiera on arytmetykę liczb całkowitych. Zmienne historyczne będą oznaczane symbolem χ z ewentualnymi indeksami procesów. Ponadto będzie się wykorzystywać zmienne zegarowe, oznaczone symbolem τ z ewentualnymi indeksami, służące do zapamiętywania aktualnego stanu czasomierza procesu. Przyjmuje się założenie, że zmienne pomocnicze χ_i, τ_i będą używane tylko w asercjach dowodu sekwencyjnego danego procesu P_i (i=1..n). Innymi zmiennymi występującymi jako zmienne wolne w asercjach dowodu danego procesu są tylko zmienne lokalne tego procesu.

Z powodu wprowadzenia zmiennych pomocniczych rozszerzeniu ulega pojęcie stanu programu. Stanem s programu $\|_{i=1..n} PC_i$ będzie więc funkcja częściowa

$$s: \mathrm{Varid} \cup \{\tau_1, \ldots, \tau_n\} \cup \{\chi_1, \ldots, \chi_n\} \cdot \longrightarrow \mathrm{Val}.$$

W tej formie zapisu uwypukla się fakt, że zmiennym historycznym przypisuje się jako wartości liczby całkowite, stanowiące kod ciągów od-

działywań przypisywanych tym zmiennym. Dla czytelności historie oddziaływań będą zapisywane w asercjach w postaci niezakodowanej; $s(\chi_i)$ będzie formalnie traktowana jako liczbowa wartość kodu historii zapisanej w χ_i.

Zbiór wszystkich rozszerzonych stanów będzie oznaczany przez Astates.

Wprowadzenie zmiennych czasowych i rozszerzenie znaczenia stanu pociąga za sobą konieczność drobnej modyfikacji wprowadzonych poprzednio semantyk Sem_d $d \in \{p,w,s,t\}$. Konieczność tej modyfikacji wypływa z przyjęcia roli, jaką mają odgrywać zmienne zegarowe: wartość zmiennej τ_i w procesie PC_i ma być w każdym momencie obliczenia równa wartości wskazywanej przez zegar $\underline{cl}\ PC_i$. Jeśli $s(\tau_i)$ w dowolnej rozpatrywanej sytuacji $\langle s,c\rangle$ jest określone, to $s(\tau_i) = c(\underline{cl}\ PC_i)$. Sytuacje spełniające ten warunek będą nazywane sytuacjami prawidłowymi.

Definicja 4.2. Predykat PROP($\langle s,c\rangle$) jest prawdziwy w dowolnej interpretacji wtedy i tylko wtedy, gdy: jeżeli $s(\tau_i)$ jest określone, to $s(\tau_i) = c(\underline{cl}\ PC_i)$ dla $i=1,\ldots,n$. ■

Stąd wyprowadza się następujące nowe semantyki pomocnicze:

$$Asem_p[\![PR]\!](s) = \{s' \mid (\exists c)(\langle PR,s,c\rangle \rightarrow\!\!\!\!\ast\ \langle s',c'\rangle \wedge PROP(\langle s,c\rangle))\},$$

$$Asem_w[\![PR]\!](s) = Asem_p[\![PR]\!](s) \cup \{\underline{infinity} \mid (\exists c)(\langle PR,s,c\rangle \rightarrow\!\!\!\!\ast\ \underline{infinity} \wedge PROP(\langle s,c\rangle))\},$$

$$Asem_s[\![PR]\!](s) = Asem_w[\![PR]\!](s) \cup \{\underline{abortion} \mid (\exists c)(\langle PR,s,c\rangle \rightarrow\!\!\!\!\ast\ \underline{abortion} \wedge PROP(\langle s,c\rangle))\},$$

$$Asem_t[\![PR]\!](s) = Asem_s[\![PR]\!](s) \cup \{\underline{deadlock} \mid (\exists c)(\langle PR,s,c\rangle \rightarrow\!\!\!\!\ast\ \underline{deadlock} \wedge PROP(\langle s,c\rangle))\},$$

oraz dla $d \in \{p,w,s,t\}$

$$Asem_d[\![PR]\!]([\alpha]_J) = \bigcup_{s \in [\alpha]_J} Asem_d[\![PR]\!](s)$$

Definicja 4.3. Interpretacja formuł specyfikacyjnych $\{\alpha\}$ PR $\{\beta\}$ jest następująca:

$$\vdash_{J,d} \{\alpha\} PR \{\beta\} \qquad d \in \{p,w,s,t\}$$

wtedy i tylko wtedy, gdy

$$Asem_d [\![PR]\!] ([\alpha]_J) \subseteq [\beta]_J.$$ ■

Trzecia omawiana tu cecha systemów dowodzenia własności programów wynika z faktu, że obliczenia przebiegają w czasie rzeczywistym. Klasa obliczeń programów współbieżnych nie uwzględniających czasu rzeczywistego jest szersza niż klasa obliczeń programów współbieżnych uwzględniających upływ czasu rzeczywistego. Jest to spowodowane tym, że te wszystkie syntaktycznie skojarzone pary instrukcji komunikacji, które mogą kojarzyć się ze sobą dynamicznie podczas obliczeń programów nie uwzględniających upływu czasu, nie muszą się kojarzyć dynamicznie, gdy uwzględni się jego upływ. Problem wyznaczenia zbioru obliczeń programu czasu rzeczywistego jest najtrudniejszym do rozwiązania zagadnieniem podczas dowodzenia własności programów. W celu uniknięcia, przynajmniej częściowo, tych problemów proponuje się tutaj, aby zamiast dokładnego wyznaczania takich zbiorów obliczeń wprowadzać i wyznaczać pewne ich nadzbiory. Efektem takiego zabiegu jest uproszczenie dowodów własności, kosztem osłabienia zupełności systemu dowodzenia własności. Sposób uproszczeń, o których jest tu mowa, może zależeć od konkretnego programu i własności, których należy dowieść, jednak całe postępowanie musi przebiegać według określonego schematu.

Dlatego w dalszej części rozdziału wprowadza się, obok pojęcia systemu dowodzenia własności, pojęcie schematu systemu dowodzenia własności programów. O schemacie systemu będzie można dowieść, że jest on niesprzeczny i zupełny, podczas gdy o konkretnym systemie będzie można dowieść tylko jego niesprzeczności.

Schemat systemu dowodzenia, podobnie jak system dowodzenia, będzie składać się z tego samego zbioru aksjomatów oraz reguł, natomiast różnica między nimi będzie dotyczyć postaci reguły równoległego składania dowodów sekwencyjnych, a dokładniej postaci tzw. warunków zgodności.

4.3. Schemat systemu dowodzenia poprawności częściowej

4.3.1. Aksjomaty

Zbiór aksjomatów A_p schematu systemu dowodzenia poprawności częściowej PS_p zawiera aksjomaty:

A1. Aksjomat instrukcji zostaw

$\{\alpha\}$ **skip** $\{\alpha\}$.

A2. Aksjomat instrukcji zerwij

$\{\alpha\}$ **abort** $\{$**false**$\}$.

A3. Aksjomat instrukcji podstawienia

$\{\alpha\}\ x:=e\ \{\beta\}$

gdzie α jest postaci

$$(\forall t)(ASS_e[t/dur] \Longrightarrow \beta\ [e/x, \tau + t/\tau]).$$

A4. Aksjomat instrukcji odczytu zegara

$\{\alpha[\tau/x]\}$ rcl $x\{\alpha\}$.

A5. Aksjomat zachowawczy

Jeżeli asercja β nie zawiera zmiennej wolnej modyfikowanej przez instrukcję elementarną AC, różną od instrukcji komunikacji, to

$\{\alpha\}$ AC $\{\beta\}$,

gdzie α jest postaci

$$(\forall t)(D_{AC}[t/dur] \Longrightarrow \beta\ [\tau + t/\tau]),$$

zaś D_{AC} jest predykatem opisującym czas realizacji instrukcji AC.

A6. Aksjomat instrukcji komunikacji

$\{\alpha\}$ CM $\{\beta\}$.

Zbiór instrukcji komunikacji rozszerza się przy tym o tzw. pustą instrukcję komunikacji oznaczoną symbolem nosyn. ■

Przedstawione aksjomaty mają następujące uzasadnienie.

Aksjomat A1 wyraża fakt, że efekt działania instrukcji skip jest pusty: nie modyfikuje ona zmiennych programu, a jej realizacja odbywa się natychmiastowo.

Aksjomat A2 stwierdza, że wykonanie instrukcji abort przynosi efekt nieokreślony.

Aksjomat A3 jest uogólnieniem klasycznego aksjomatu dla instrukcji podstawienia. Ponieważ w języku RTCSP uwzględnia się upływ czasu podczas wykonywania instrukcji, zatem efekt ten wyraża się w zmianie wartości zmiennych czasowych występujących w asercjach. Uzasadnienie postaci aksjomatu wynika z następujących rozważań. Należy przypomnieć, że każdy elementarny krok obliczeniowy jest wykonywany w skończonym odcinku czasu. Zatem dla ustalonego stanu s zbiór

$$\{t \mid [\![ASS_e[t/dur]]\!]_s = tt\}$$

jest również skończony. Niech $t_1, \ldots, t_N$ będą wszystkimi elementami tego zbioru. Jeżeli instrukcja x:=e rozpoczyna swą realizację w sytuacji $\langle s,c \rangle$, to efekt jej działania można wyrazić równoważnie przez instrukcję Dijkstry [44]

$$\underline{\text{if}}_{\langle s\rangle} \bigsqcup_{i=1\ldots N} \underline{\text{true}} \longrightarrow x:=e;\ \tau:=\tau+t_i\ \underline{\text{fi}}$$

Oznaczenie $\underline{\text{if}}_{\langle s\rangle}$ ... $\underline{\text{fi}}$ podkreśla fakt, że postać instrukcji zależy od wartości stanu s w początkowej sytuacji $\langle s,c\rangle$. Z definicji zmiennej pomocniczej τ zachodzi równość $s(\tau) = c(\underline{\text{cl}}\ x:=e)$. Zawartością instrukcji są ciągi zwykłych instrukcji podstawienia, z których realizacją nie wiąże się pojęcie upływu czasu. Instrukcja wybierając niedeterministycznie jedną z alternatyw, prawidłowo modeluje upływ czasu podczas wykonywania instrukcji podstawienia. Łatwo stwierdzić, że zgodnie z podanymi przez Dijkstrę zasadami obliczania najsłabszych warunków wstępnych, dla danej asercji końcowej β, najsłabszy warunek wstępny ma postać

$$(\forall i=1\ldots N)\ \beta\,[e/x,\ \tau+t_i/\tau],$$

przy czym t_i są takie, że $[\![ASS_e[t_i/dur]]\!]_s = tt$, co dla dowolnego s jest równoważne z formułą

$$(\forall t)(ASS_e[t/dur] \Longrightarrow \beta\,[e/x,\ \tau+t/\tau]).$$

Aksjomat A4 nie wymaga odrębnego uzasadnienia: wynika on bezpośrednio stąd, że instrukcja $\underline{\text{rcl}}$ x odczytując stan zegara, realizuje się natychmiastowo.

Aksjomat A5, stanowiąc szczególną postać aksjomatu A3, jest oczywisty: jeżeli instrukcja AC nie modyfikuje żadnej ze zmiennych wolnych swej asercji końcowej, to - będąc instrukcją wykonywaną w pojedynczym kroku obliczeniowym, różną od instrukcji komunikacji - może tylko wpłynąć na uwzględnienie upływu czasu. Powodem wprowadzenia tego aksjomatu jest konieczność użycia w dalej przedstawionych regułach wnioskowania pewnych instrukcji pomocniczych, których jedynym efektem działania jest tylko upływ czasu. Instrukcjami tymi będą:

$\underline{\text{test}}$ (IF), $\underline{\text{test}}$ (DO), $\underline{\text{test}}$ (TH),

$\underline{\text{wait}}$ (t), $\underline{\text{later}}$ (t).

Czas trwania tych instrukcji wyznaczają odpowiednio predykaty:

$TEST_{IF}$, $TEST_{DO}$, $TEST_{TH}$,

$WAIT_t$, $LATER_t$,

gdzie $LATER_t \Longleftrightarrow dur = t+1$, dla dowolnego stanu s.

Aksjomat A6, jako wyizolowany obiekt, może budzić zdumienie, gdyż dopuszcza całkowicie dowolne przekształcenie sytuacji początkowej, przed wykonaniem instrukcji komunikacji CM, w sytuację końcową. Takiej dowolności jednak nie będzie, gdyż aksjomat zawiera świadomie pozo-

stawioną "lukę", którą wypełni reguła kojarząca dowody sekwencyjne dla pojedynczych procesów we wspólny dowód dla całego programu.

Dołączenie do zbioru instrukcji komunikacyjnych pustej instrukcji komunikacyjnej nosyn wynika z konieczności jej użycia w regule wnioskowania związanej z czasowo uwarunkowanymi instrukcjami komunikacyjnymi. Zakłada się, że instrukcja nosyn realizuje się natychmiastowo i nie modyfikuje zmiennych programu (podobnie jak instrukcja skip). Jedynym efektem jej wykonania będzie modyfikacja pomocniczej zmiennej historycznej procesu, w którym jest wykonywana. Modyfikacja ta będzie polegać na zarejestrowaniu oddziaływania symbolizującego brak zsynchronizowania się danego procesu z innym procesem.

4.3.2. Reguły

Zbiór reguł R_p schematu systemu dowodzenia PS_p zawiera następujące reguły:

R1. Reguła złożenia sekwencyjnego instrukcji

$$\frac{\{\alpha\}\ AC1\ \{\beta\},\ \{\beta\}\ AC2\ \{\gamma\}}{\{\alpha\}\ AC1;\ AC2\ \{\gamma\}}$$

R2. Reguła instrukcji alternatywy

$$\frac{\{\alpha \wedge b_i\}\ \underline{test}\ (IF);\ AC_i\ \{\beta\}\ (i=1,\ldots,n)}{\{\alpha\}\ \underline{if}\ \underset{i=1\ldots n}{\Box}\ b_i \longrightarrow AC_i\ \underline{end}\ \{\beta\}}$$

R3. Reguła instrukcji iteracji

$$\frac{\{\alpha \wedge b_i\}\ \underline{test}\ (DO);\ AC_i\ \{\alpha\}\ (i=1,\ldots,n),\ \{\alpha \wedge \neg\, BOOL\ (DO)\}\ \underline{test}\ (DO)\ \{\beta\}}{\{\alpha\}\ \underline{do}\ \underset{i=1\ldots n}{\Box}\ b_i \longrightarrow AC_i\ \underline{end}\ \{\beta\}}$$

R4. Reguła czasowo-uwarunkowanej instrukcji komunikacji

$$\frac{\{\alpha \wedge b_i\}\ \underline{test}\ (TH);\ \underline{wait}\ (v);\ CM_i;\ AC_i\ \{\beta\}\ (i=1,\ldots,n),\ \{\alpha \wedge BOOL(TH)\}\ \underline{test}\ (TH);\ \underline{later}\ (v);\ \underline{nosyn};\ AC\ \{\beta\}}{\{\alpha\}\ \underline{th}\ v\ \underline{wt}\ \underset{i=1\ldots n}{\Box}\ b_i;\ CM_i \longrightarrow AC_i\ \underline{lt}\ AC\ \underline{end}\ \{\beta\}}$$

R5. Reguła konsekwencji

$$\frac{\alpha \Longrightarrow \alpha_1,\ \{\alpha_1\}\ AC\ \{\beta_1\},\ \beta_1 \Longrightarrow \beta}{\{\alpha\}\ AC\ \{\beta\}}$$

R6. Reguła zastąpienia

$$\frac{\{\alpha\}\ PR\ \{\beta\},\ tr \in Term,\ x \notin FV(PR) \cup FV(\beta),\ x \in FV(\alpha)}{\{\alpha[tr/x]\}\ PR\ \{\beta\}}$$

R7. Reguła złożenia równoległego

$$\frac{\text{Dowody } \{\alpha_i\}\ PC_i\ \{\beta_i\} \text{ są zgodne dla } i=1..n.}{\{\alpha_1 \wedge \ldots \wedge \alpha_n\}\ \underset{i=1\ldots n}{\|}\ PC_i\ \{\beta_1 \wedge \ldots \wedge \beta_n\}}$$

Reguła R1 ma znaną postać: jak w przypadku programowania sekwencyjnego przyjmuje się, że sytuacja wynikowa po zakończeniu instrukcji AC1 staje się sytuacją początkową przed wykonaniem instrukcji AC2.

Reguła R2 mówi, że obliczenie wartości wyrażeń logicznych wymaga upływu pewnego odcinka czasu, co dodatkowo wyraża pomocnicza instrukcja test (IF), której odpowiada predykat $TEST_{IF}$. Instrukcja ta poprzedza wykonanie każdej instrukcji alternatywnej AC_i.

W regule R3 także występuje instrukcja pomocnicza test (DO), poprzedzająca wykonanie każdej z alternatyw AC_i, a także poprzedza ona zakończenie instrukcji DO. W stosunku do reguły dla instrukcji iteracji, wykonywanej bez uwzględniania upływu czasu rzeczywistego, występuje nowa przesłanka

$$\{\alpha \wedge \neg BOOL\ (DO)\}\ \underline{test}\ (DO)\ \{\beta\}$$

która podkreśla fakt, że ostatni krok obliczeniowy, pomimo braku modyfikacji zmiennych programowych, powoduje pewne opóźnienie czasowe.

Reguła R4 przypomina swym kształtem regułę instrukcji alternatywy. Przesłanki w pierwszym wierszu odpowiadają tym przypadkom, w których, w ciągu zadanego odcinka czasu przeterminowania, następuje oddziaływanie procesu zawierającego instrukcję TH z innym procesem. W przesłankach tych występują dwie pomocnicze instrukcje, z których pierwsza - test (TH) - symuluje upływ czasu podczas obliczenia wyrażeń logicznych, druga zaś - wait (v) - symuluje upływ czasu podczas oczekiwania na skojarzenie z pewnym procesem. Przesłanka w drugim wierszu odpowiada brakowi oddziaływań danego procesu z innymi procesami. Występująca tu instrukcja pomocnicza later (v) modeluje upływ odcinka czasu o długości v. Natomiast pusta instrukcja komunikacyjna nosyn będzie wykorzystywana przy kojarzeniu dowodów sekwencyjnych. Można ją tu traktować jako instrukcję symbolizującą zajście oddziaływania w procesie, polegającego na braku zsynchronizowania się danego procesu z jakimkolwiek innym procesem.

Reguła R5 ma znaną postać. Dwie implikacje zawarte w zbiorze jej przesłanek reprezentują zewnętrzny, w stosunku do systemu dowodzenia, mechanizm wyrokujący o ich prawdziwości.

Reguła zastąpienia R6 ma prosty intuicyjny sens: jeżeli x jest zmienną,która nie występuje ani w programie, ani w asercji końcowej, to zastąpienie tej zmiennej w asercji α zmienną o innej nazwie - lub ogólnie zastąpienie dowolnym termem $tr \in Term$ - nie wprowadza zmiany zbioru stanów wyznaczanych przez asercję początkową.

Reguła R7 podaje zasady składania dowodów sekwencyjnych dla procesów w jeden wspólny dowód dla całego programu. Wykorzystywane w niej pojęcie zgodności dowodów sekwencyjnych wymaga oddzielnego omówienia.

4.3.3. Zgodność dowodów sekwencyjnych

Zgodność dowodów sekwencyjnych polega na zachowaniu odpowiednich warunków przez asercje poprzedzające i następujące po instrukcjach komunikacji, które występują w tych dowodach. Chodzi zatem o instrukcje komunikacji, które zawierają teksty poszczególnych procesów, a także o puste instrukcje komunikacyjne wprowadzone do dowodów jako obiekty pomocnicze.

Dowody sekwencyjne będą zgodne, jeżeli dla każdej syntaktycznie skojarzonej pary instrukcji komunikacji będzie spełniony tzw. warunek skojarzenia, a dla każdej pustej instrukcji komunikacyjnej - tzw. warunek przeterminowania.

W dalszym ciągu przez pojęcie instrukcji będzie rozumiany dowolny element zbioru

$$Ccm \cup Com \cup Acm$$

gdzie: $Acm = \{\underline{test}\ (AC) \mid AC \in Cif \cup Cdo \cup Cth\} \cup$
$\{\underline{wait}\ (v) \mid v \in Int\} \cup$
$\{\underline{later}\ (v) \mid v \in Int\} \cup \{\underline{nosyn}\}$

Jeżeli AC jest dowolną instrukcją, to przez pre(AC) oraz post(AC) będą oznaczane asercje, które występują w dowodach sekwencyjnych bezpośrednio przed i po danej instrukcji AC.

Jeżeli CM jest instrukcją wejścia/wyjścia, w jednym procesie, to syntaktycznie skojarzoną z nią instrukcję wejścia/wyjścia w drugim procesie będzie oznaczać się przez $\overline{CM}$.

Warunek skojarzenia

Warunek skojarzenia ma opisywać związek między dowolną parą CM,$\overline{CM}$ syntaktycznie skojarzonych instrukcji wejścia/wyjścia. Należy zaznaczyć, że wykonanie takiej pary może nastąpić tylko wtedy, gdy sterowanie w obu procesach osiągnie w pewnej chwili (tej samej dla zegarów obu procesów) instrukcje CM, $\overline{CM}$. Należy też dodać, że jeżeli sterowanie osiągnie takie położenie, to dotychczas zaobserwowane lokalnie w obu procesach historie oddziaływań są zgodne w sensie definicji 3.16. Również bezpośrednio po wykonaniu pary instrukcji CM, $\overline{CM}$ czasomierze obu

procesów będą wskazywać tę samą chwilę, nowe zaś lokalne historie oddziaływań pozostaną zgodne.

Zatem bezpośrednio przed i bezpośrednio po wykonaniu pary instrukcji CM w procesie P i skojarzonej z nią syntaktycznie instrukcji $\overline{CM}$ w procesie Q będzie prawdziwy predykat

$$inv\ (P,Q) \iff \tau_P = \tau_Q \wedge \chi_{PQ} = \overline{\chi}_{QP}$$

gdzie τ_P, τ_Q są pomocniczymi zmiennymi czasowymi procesów P, Q; χ_{PQ} jest podciągiem ciągu χ_P złożonym ze wszystkich tych elementów λ zmiennej historycznej χ_P, dla których λ.part = Q; natomiat χ_{QP} jest otrzymywane z podciągu χ_{QP} w taki sposób, że każdy jego element λ jest zastąpiony oddziaływaniem komplementarnym $\overline{\lambda}$.

Predykat inv(P,Q) będzie nazywany niezmiennikiem skojarzeń procesów P, Q.

Załóżmy, że dana jest pewna reguła, która dla danego programu $PR = \|_{i=1,\ldots,n} PC_i$ i założonych warunków początkowych $pre(PC_i)$ (i = =1,...,n) rozstrzyga, czy istnieje obliczenie programu PR takie, że syntaktycznie skojarzone instrukcje CM, $\overline{CM}$ kojarzą się dynamicznie w stanie s. Niech reguła ta będzie wyrażona w postaci asercji

$$possmatching\ (CM,\overline{CM}),$$

która dla danego stanu s przyjmuje wartość prawdziwą wtedy i tylko wtedy, gdy takie obliczenie istnieje, Wtedy między asercjami początkowymi i końcowymi instrukcjami CM, $\overline{CM}$ powinien zachodzić następujący warunek skojarzenia:

Definicja 4.4. Para syntaktycznie skojarzonych instrukcji wejścia /wyjścia

$$CM = P!W(e), \quad \overline{CM} = Q?W(x)$$

spełnia warunek skojarzenia, jeżeli zachodzi formuła:

$$pre(CM) \wedge pre(\overline{CM}) \wedge inv(P,Q) \wedge possmatching\ (CM,\overline{CM}) \Longrightarrow$$
$$\Longrightarrow (\forall t)(SEND[t/dur] \Longrightarrow$$
$$((post(CM) \wedge post(\overline{CM}) \wedge inv(P,Q))[\tau_P+t/\tau_P, \tau_Q+t/\tau_Q])$$
$$[e/x, \lambda_P \cap \lambda/\chi_P, \chi_Q \cap \overline{\lambda}/\chi_Q]),$$

gdzie

$$\lambda = (Q,P,W,!,e,\tau_P),$$
$$\overline{\lambda} = (P,Q,W,?,e,\tau_Q),$$

zaś symbol $\cap$ oznacza tu i dalej konkatenację ciągów. ■

Podana definicja warunku skojarzenia ma następujące uzasadnienie. Realizacja pary syntaktycznie skojarzonych instrukcji $CM, \overline{CM}$ w stanie początkowym s jest w efekcie równoważna z sekwencyjnym wykonaniem następujących instrukcji podstawienia:

1. $x := e$

przy czym x jest zmienną procesu P, natomiast e wyrażeniem w procesie Q:

2. $\chi_P := \chi_P \cap \lambda$

   $\chi_Q := \chi_Q \cap \bar{\lambda}$

gdzie: $\lambda = (Q,P,W,!,v,\tau_P)$, $\bar{\lambda} = (P,Q,W,?,v,\tau_Q)$ i oczywiście $v = [\![e]\!]_s$.

3. $\tau_P := \tau_P + t$,

   $\tau_Q := \tau_Q + t$

gdzie t oznacza czas realizacji, oczywiście $[\![SEND_e(t)]\!]_s = tt$.

Niech post(CM) w procesie Q oraz post(CM) w procesie P będą asercjami, które mają być prawdziwe po zakończeniu pary skojarzonych instrukcji. Zatem asercję

$$post(CM) \wedge post(\overline{CM}) \wedge inv(P,Q)$$

można uważać za warunek końcowy dla podanego wyżej zestawu instrukcji podstawienia. Ze znanego aksjomatu dla instrukcji podstawienia (nie uwarunkowanej czasowo) [44] wynika, że

$$((post(CM) \wedge post(\overline{CM}) \wedge inv(P,Q))[\tau_P+t/\tau_P, \tau_Q+t/\tau_Q])$$
$$[e/x, \chi_P \cap \lambda / \chi_P, \chi_Q \cap \bar{\lambda} / \chi_Q]$$

jest asercją, która powinna być spełniona w momencie rozpoczęcia realizacji instrukcji. Asercja ta powinna być, oczywiście, prawdziwa dla dowolnego t spełniającego warunek $[\![SEND_e[t/dur]]\!]_s = tt$, czyli zachodzi asercja A:

$$(\forall t)(SEND_e\ [t/dur] \Longrightarrow ((post(CM) \wedge post(\overline{CM}) \wedge inv(P,Q))$$
$$[\tau_P+t/\tau_P, \tau_Q+t/\tau_Q])[e/x, \chi_P \cap \lambda / \chi_P, \chi_Q \cap \bar{\lambda} / \chi_Q)]$$

określająca zbiór wszystkich sytuacji początkowych, które prowadzą do pożądanych sytuacji końcowych. Ponieważ z założenia początkowe sytuacje, jakie zachodzą przed rozpoczęciem skojarzonych instrukcji, określa asercja B:

$$pre(CM) \wedge pre(\overline{CM}) \wedge inv(P,Q) \wedge possmatching\ (CM, \overline{CM})$$

więc musi zachodzić implikacja $B \Longrightarrow A$, czego właśnie wymaga definicja 4.4.

Warunek przeterminowania

Warunek przeterminowania opisuje związek jaki zachodzi pomiędzy początkową i końcową asercją pustej instrukcji komunikacyjnej umieszczonej w dowodzie każdej czasowo uwarunkowanej instrukcji komunikacji TH postaci

$$\underline{th} \;\; v \;\; \underline{wt} \;\; \Box_{i=1...n} \; b_i ; CM_i \longrightarrow AC_i \;\; \underline{lt} \;\; AC \;\; \underline{end}$$

Asercja pre(nosyn) określa zbiór tych stanów, które byłyby osiągnięte, w procesie P zawierającym instrukcję TH, gdyby proces P osiągał instrukcję nosyn. Osiągnięcie tej instrukcji jest uwarunkowane tym, aby w odcinku czasu przeterminowania nie zachodziła komunikacja procesu P z żadnym innym procesem. Uwarunkowanie to powinna opisywać inna asercja. Niech timeout (TH) będzie asercją określającą zbiór tych stanów, dla których jest możliwe, że nastąpi upływ odcinka przeterminowania - dokładniej: timeout (TG)(s) = tt winno oznaczać, że istnieją takie obliczenia programu, że proces P osiąga instrukcję nosyn w stanie s. Jeżeli timeout (TH) ma taką własność, to postać warunku przeterminowania jest następująca.

Definicja 4.5. Pusta instrukcja komunikacyjna nosyn umieszczona w dowodzie instrukcji TH

$$\underline{th} \;\; v \;\; \underline{wt} \;\; \Box_{i=1...n} \; b_i ; CM_i \longrightarrow AC_i \;\; \underline{lt} \;\; AC \;\; \underline{end}$$

wchodzącej w skład procesu P, spełnia warunek przeterminowania, jeżeli jest prawdziwa implikacja

$$pre(\underline{nosyn}) \wedge timeout(TH) \Longrightarrow post(\underline{nosyn})[\chi_P \cap \lambda / \chi_P],$$

gdzie $\lambda = (P, \tau_P)$ jest oddziaływaniem oznaczającym upływ czasu przeterminowania. ■

4.3.4. Uwagi o nieinterferencji dowodów sekwencyjnych

Konstrukcja dowodu sekwencyjnego dla pojedynczego procesu, w sposób niezależny od treści pozostałych procesów, oznacza przyjęcie założenia, że realizacja pozostałych procesów nie modyfikuje wartości zmiennych lokalnych danego procesu - z wyjątkiem tych sytuacji, w których zachodzi wymiana komunikatów przez wykonanie instrukcji komunikacji - oraz nie zmienia prawdziwości asercji występujących w dowodzie. Założenie to znajduje formalne odbicie w pojęciu nieinterferencji dowodów sekwencyjnych. Pojęcie to zostało wprowadzone przez Owicką [171], do języków zaś klasy CSP było zaadaptowane przez Levina [136].

Niech α będzie asercją użytą w dowodzie sekwencyjnym procesu P, natomiast AC niech będzie dowolną instrukcją, różną od instrukcji komunikacji, w procesie Q. Fakt, że instrukcja AC nie ma wpływu na prawdziwość α oznacza, że powinna być prawdziwa specyfikacja

$$\{\alpha \wedge pre(AC)\}\ AC\ \{\alpha\}$$

gdzie pre(AC) jest asercją stojącą przed AC w dowodzie sekwencyjnym procesu Q.

Gdy dana jest para syntaktycznie skojarzonych instrukcji komunikacji AC = R!W(e) i $\overline{AC}$ = Q?W(x), powinna zachodzić asercja

$$(\alpha \wedge pre(AC) \wedge pre(\overline{AC}) \wedge inv(R,Q)) \Longrightarrow (\forall t)(SEND_e[t/dur] \Longrightarrow$$

$$(inv(R,Q) \wedge \alpha[e/x,\ \tau_Q+t/\tau_Q,\ \tau_R+t/\tau_R,\ \chi_Q \cap \lambda/\chi_Q,\ \chi_R \cap \bar{\lambda}/\chi_R])$$

Postać tej asercji wynika z warunku skojarzenia, gdy zamiast post(AC) $\wedge$ post($\overline{AC}$) zostanie podstawiona asercja α, od której wymaga się, aby pozostała prawdziwa po wykonaniu pary AC, $\overline{AC}$.

Jeśli dowody sekwencyjne spełniają dwa wyżej podane warunki, mówi się o ich nieinterferencji.

Pojęcie nieinterferencji jest ważne wówczas, gdy w dowodach używa się zmiennych pomocniczych. Jak stwierdzono w podrozdz. 4.2, jedynymi zmiennymi pomocniczymi są zmienne historyczne χ_P oraz zmienne zegarowe τ_P. Przyjmuje się założenie, że w dowodzie procesu P używa się wyłącznie zmiennych historycznych i czasowych danego procesu P. Ponadto zakłada się, że innymi zmiennymi używanymi w asercjach dotyczących danego procesu są tylko zmienne lokalne tego procesu.

Z tak przyjętych ustaleń wynika, że budowane dowody sekwencyjne nie interferują ze sobą, w związku z czym nie zachodzi dodatkowa konieczność sprawdzania warunków nieinterferencji.

4.3.5. Przykłady

Dla ilustracji dowodu poprawności częściowej, a zwłaszcza problemów wynikających z konieczności sprawdzenia zgodności dowodów sekwencyjnych, rozpatruje się następujący elementarnie prosty program złożony z dwóch procesów:

```
P: th T1 wt   Q!W(x) ⟶ x:=1
         lt    x:=2
   end
   ||
Q: W;
   th T2 wt   P?W(y) ⟶ skip
         lt    y:=2
   end
```

Środowisko wykonawcze programu definiuje następujący zbiór predykatów:

$$ASS_e(d) \iff 1 \leqslant d \leqslant 2$$

$$TEST_{TH}(d) \iff 0 \leqslant d \leqslant 1$$

$$SEND_e(d) \iff 1 \leqslant d \leqslant 2$$

dla dowolnych $e \in Exp$, $TH \in Cth$, $s \in States$.

Niech

$$x = 0 \wedge \tau_P = a \wedge \chi_P = ( )$$

oraz

$$\tau_Q = b \wedge \chi_Q = ( )$$

będą początkowymi asercjami dla procesów P, Q. $\chi_P = ( )$ oraz $\chi_Q = ( )$ oznacza, że historie χ_P, χ_Q są puste.

Przyjmując różne wartości dla stałych a, b oraz T1, T2, można otrzymać różne obliczenia procesów. Możliwe są tu przypadki:

1) pomiędzy P i Q skojarzenie dynamiczne zawsze zachodzi,
2) może (ale nie musi) zajść,
3) nigdy nie zachodzi.

Łatwo przekonać się, że np. przypadki te zachodzą odpowiednio dla następujących zestawów wartości stałych:

1) a = 0, b = 5, T1 = 8, T2 = 5,
2) a = 0, b = 9, T1 = 8, T2 = 5,
3) a = 0, b = 10, T1 = 8, T2 = 5.

Poniżej przedstawia się dowody programu odpowiadające tym zestawom.

Przypadek 1

Dowody sekwencyjne

P: $\{x=0 \wedge \tau_P=0 \wedge \chi_P=( )\}$

th 8 wait

$\{x=0 \wedge 0 \leqslant \tau_P \leqslant 9 \wedge \chi_P=( )\}$

Q!W(x) $\longrightarrow$

$\{x=0 \quad (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 1 \leqslant \tau_P \leqslant 11 \wedge \chi_P=(P,Q,W,!,0,\tau_P-t_1))$

x:=1

$\{x=1 \wedge (\exists t_1)(\exists t_2)(1 \leqslant t_1 \leqslant 2 \wedge 1 \leqslant t_2 \leqslant 2 \wedge 2 \leqslant \tau_P \leqslant 13 \wedge$

$\chi_P=(P,Q,W,!,0,\tau_P-t_1-t_2))\}$

later

$\{x=0 \wedge 9 \leqslant \tau_P \leqslant 10 \wedge \chi_P=( )\}$

nosyn {false}

x:=2

{false}

end

$\{x=1 \wedge (\exists t_1)(\exists t_2)(1 \leqslant t_1 \leqslant 2 \wedge 1 \leqslant t_2 \leqslant 2 \wedge 2 \leqslant \tau_P \leqslant 13 \wedge \chi_P=(P,Q,W,!,0,\tau_P-t_1-t_2))\}$

Q: $\{\tau_Q=5 \wedge \chi_Q=(\,)\}$

th 5 wait

$\{5 \leqslant \tau_Q \leqslant 11 \wedge \chi_Q=(\,)\}$

P?W(y) →

$\{y=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 6 \leqslant \tau_Q \leqslant 13 \wedge \chi_Q=(Q,P,Q,?,0,\tau_Q-t_1))\}$

skip

$\{y=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 6 \leqslant \tau_Q \leqslant 13 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1))\}$

later

$\{11 \leqslant \tau_Q \leqslant 12 \wedge \tau_Q=(\,)\}$

nosyn {false}

y:=2

{false}

end

$\{y=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 6 \leqslant \tau_Q \leqslant 13 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1))\}$

Warunek skojarzenia

Niech

$$pre \Longleftrightarrow pre(Q!W(x)) \wedge pre(P?W(y)) \wedge inv(P,Q),$$
$$post \Longleftrightarrow post(Q!W(x)) \wedge post(P?W(y)) \wedge inv(P,Q)$$

czyli

$$pre \Longleftrightarrow x=0 \wedge 5 \leqslant \tau_P \leqslant 9 \wedge \chi_P=(\,) \wedge \chi_Q=(\,) \wedge \tau_P=\tau_Q\,,$$
$$post \Longleftrightarrow x=0 \wedge y=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 1 \leqslant \tau_P \leqslant 11 \wedge$$
$$\chi_P=(P,Q,W,!,0,\tau_P-t_1)) \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 6 \leqslant \tau_Q \leqslant 13 \wedge$$
$$\chi_Q=(Q,P,W,?,0,\tau_Q-t_1)) \wedge \tau_P=\tau_Q\,.$$

Prawa strona implikacji w warunku skojarzenia ma postać:

$$(\forall t)(1 \leqslant t \leqslant 2 \Longrightarrow post\,[\tau_P+t/\tau_P,\ \tau_Q+t/\tau_Q])[x/y,\ \chi_P \cap \lambda/\chi_P,$$
$$\chi_Q \cap \bar{\lambda}/\chi_Q]$$

gdzie: $\lambda = (P,Q,W,!,x,\tau_P)$.

Po dokonaniu potrzebnych przekształceń otrzymuje się:

$$(post\,[\tau_P+1/\tau_P,\ \tau_Q+1/\tau_Q])\ [x/y,\ \chi_P \cap \lambda/\chi_P,\ \chi_Q \cap \bar{\lambda}/\chi_Q] \wedge$$
$$(post\,[\tau_P+2/\tau_P,\ \tau_Q+2/\tau_Q])\ [x/y,\ \chi_P \cap \lambda/\chi_P,\ \chi_Q \cap \bar{\lambda}/\chi_Q] \Longleftrightarrow$$
$$x=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 0 \leqslant \tau_P \leqslant 10 \wedge \chi_P=(P,Q,W,!,0,\tau_P-t_1+1)) \wedge$$

$$(\exists t_1)(1 \leq t_1 \leq 2 \wedge 5 \leq \tau_Q \leq 12 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1+1)) \wedge$$
$$(\exists t_1)(1 \leq t_1 \leq 2 \wedge -1 \leq \tau_P \leq 9 \wedge \chi_P=(P,Q,W,!,0,\tau_P-t_1+2)) \wedge$$
$$(\exists t_1)(1 \leq t_1 \leq 2 \wedge 4 \leq \tau_Q \leq 11 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1+2)) \wedge$$
$$\tau_P = \tau_Q \Longleftrightarrow$$

$$x=0 \wedge 0 \leq \tau_P \leq 9 \wedge \chi_P=(\,) \wedge 5 \leq \tau_Q \leq 11 \wedge \chi_Q=(\,) \wedge \tau_P = \tau_Q \Longleftrightarrow \text{pre}.$$

Ponieważ, jak łatwo zauważyć, bezpośrednio z semantyki programu

$$\text{possmatching}\ (Q!W(x),\ P?W(y)) \Longleftrightarrow x=0 \wedge \chi_P=(\,) \wedge \chi_Q=(\,) \wedge$$
$$\tau_P = \tau_Q \wedge 5 \leq \tau_P \leq 6$$

zachodzi więc również warunek skojarzenia

$$\text{possmatching}\ (Q!W(x),\ P?W(y)) \wedge \text{pre} \Longrightarrow \text{pre}.$$

Warunek przeterminowania

Warunek przeterminowania jest, oczywiście, spełniony, gdyż, jak wynika z semantyki programu, asercja

$$\text{timeout}\ (TH1) \Longleftrightarrow \text{timeout}\ (TH2) \Longleftrightarrow \underline{\text{false}},$$

gdzie TH1, TH2 są czasowo uwarunkowanymi instrukcjami komunikacji w procesach P, Q.

Przypadek 2

Dowody sekwencyjne

P: $\{x=0 \wedge \tau_P=0 \wedge \chi_P=(\,)\}$

th 8 wait

$\{x=0 \wedge 0 \leq \tau_P \leq 9 \wedge \chi_P=(\,)\}$

$Q!W(x) \longrightarrow$

$\{x=0 \wedge (\exists t_1)(1 \leq t_1 \leq 2 \wedge 1 \leq \tau_P \leq 11 \wedge \chi_P=(P,Q,W,!,0,\tau_P-t_1))\}$

x:=1

$\{x=1 \wedge (\exists t_1)(\exists t_2)(1 \leq t_1 \leq 2 \wedge 1 \leq t_2 \leq 2 \wedge 2 \leq \tau_P \leq 13 \wedge$
$\chi_P=(P,Q,W,!,0,\tau_P-t_1-t_2)\}$

later

$\{x=0 \wedge 9 \leq \tau_P \leq 10 \wedge \chi_P=(\,)\}$

nosyn

$\{x=0 \wedge 9 \leq \tau_P \leq 10 \wedge \chi_P=(P,\tau_P)\}$

x:=2

$\{x=2 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 10 \leqslant \tau_P \leqslant 12 \wedge \chi_P=(P, \tau_P-t_1))\}$

end

$\{x=1 \wedge (\exists t_1)(\exists t_2)(1 \leqslant t_1 \leqslant 2 \wedge 1 \leqslant t_1 \leqslant 2 \wedge 2 \leqslant \tau_P \leqslant 13 \wedge \chi_P=(P,Q,W,!,\tau-t_1-t_2)) \vee x=2 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 10 \leqslant \tau_P \leqslant 12 \wedge \chi_P=(P, \tau_P-t_1))\}$

$Q:\{\tau_Q=9 \wedge \chi_Q=(\,)\}$

th 5 **wait**

$\{9 \leqslant \tau_Q \leqslant 15 \wedge \chi_Q=(\,)\}$

$P?W(y) \longrightarrow$

$\{y=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 10 \leqslant \tau_Q \leqslant 17 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1))\}$

skip

$\{y=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 10 \leqslant \tau_Q \leqslant 17 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1))\}$

later

$\{15 \leqslant \tau_Q \leqslant 16 \wedge \chi_Q=(\,)\}$

nosyn

$\{15 \leqslant \tau_Q \leqslant 16 \wedge \chi_Q=(Q,\tau_Q)\}$

$y:=2$

$\{y=2 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 16 \leqslant \tau_Q \leqslant 18 \wedge \chi_Q=(Q,\tau_Q-t_1))\}$

end

$\{y=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 10 \leqslant \tau_Q \leqslant 17 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1) \vee y=2 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 16 \leqslant \tau_Q \leqslant 18 \wedge \chi_Q=(Q,\tau_Q-t_1))\}$

Warunek skojarzenia

Przyjmując oznaczenia pre, post jak dla przypadku 1, otrzymujemy ciąg przekształceń:

$(\forall t)(1 \leqslant t \leqslant 2 \Longrightarrow \text{post}\,[\tau_P+t/\tau_P,\ \tau_Q+t/\tau_Q])\,[x/y,\ \chi_P\ \lambda/\chi_P,\ \chi_Q \cap \bar{\lambda}/\chi_Q] \Longleftrightarrow$

$x=0 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 0 \leqslant \tau_P \leqslant 10 \wedge \chi_P=(P,Q,W,!,0,\tau_P-t_1+1)) \wedge$
$(\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge -1 \leqslant \tau_P \leqslant 9 \wedge \chi_P=(P,Q,W,!,0,\tau_P-t_1+2)) \wedge$
$(\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 9 \leqslant \tau_Q \leqslant 17 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1+1)) \wedge$
$(\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge 8 \leqslant \tau_Q \leqslant 16 \wedge \chi_Q=(Q,P,W,?,0,\tau_Q-t_1+2)) \wedge$

$\tau_P = \tau_Q \Longleftrightarrow$

$x=0 \wedge 0 \leqslant \tau_P \leqslant 9 \wedge \chi_P=(\,) \wedge 9 \leqslant \tau_Q \leqslant 16 \wedge \chi_Q=(\,) \wedge \tau_P=\tau_Q \Longleftrightarrow \text{pre}.$

Oznacza to, że warunek skojarzenia jest spełniony, gdyż

$$\text{possmatching}\,(Q!W(x),\ P?W(y)) \Longleftrightarrow x=0 \wedge \chi_P=(\,) \wedge \chi_Q=(\,) \wedge \tau_P = \tau_Q = 9$$

Warunek przeterminowania

Należy zauważyć, że

$$\text{timeout}\,(TH_1) \Longleftrightarrow 9 \leqslant \tau_P \leqslant 10 \wedge x=0 \wedge \chi_P=(\,)$$

oraz

$$\text{timeout}\,(TH_2) \Longleftrightarrow 15 \leqslant \tau_Q \leqslant 16 \wedge \chi_Q=(\,).$$

Dla procesu P

$$\text{pre}(\underline{\text{nosyn}}) \wedge \text{timeout}\,(TH1) \Longleftrightarrow x=0 \wedge 9 < \tau_P < 10 \wedge \chi_P=(\,)$$

oraz

$$\text{post}(\underline{\text{nosyn}})\,[\chi_P \hat{} (P,\tau_P)/\chi_P] \Longleftrightarrow x=0 \wedge 9 < \tau_P < 10 \wedge \chi_P=(\,).$$

Zatem prawdziwa jest implikacja

$$\text{pre}(\underline{\text{nosyn}}) \wedge \text{timeout}\,(TH1) \Longrightarrow \text{post}(\underline{\text{nosyn}})\,[\chi_P \hat{} (P,\tau_P)/\chi_P].$$

Podobna implikacja zachodzi także dla procesu Q, co oznacza, że warunek przeterminowania jest spełniony.

Przypadek 3

Dowody sekwencyjne

P: $\{x=0 \wedge \tau_P=0 \wedge \chi_P=(\,)\}$

th 8 wait

$\{x=0 \wedge 0 \leqslant \tau_P \leqslant 9 \wedge \chi_P=(\,)\}$

$Q!W(x) \longrightarrow$

$\{\underline{\text{false}}\}$

$x:=1$

$\{\underline{\text{false}}\}$

later

$\{x=0 \wedge 9 \leqslant \tau_P \leqslant 10 \wedge \chi_P=(\,)\}$

nosyn

$\{x=0 \wedge 9 \leqslant \tau_P \leqslant 10 \wedge \chi_P=(P,\tau_P)\}$

$x:=2$

$\{x=2 \wedge 10 \leqslant \tau_P \leqslant 12 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge \chi_P=(P,\tau_P-t_1))\}$

end

$\{x=2 \wedge 10 \leqslant \tau_P \leqslant 12 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge \chi_P=(P,\tau_P-t_1))\}$

Q: $\{\tau_Q=10 \wedge \chi_Q=(\ )\}$

th 5 **wait**

$\{10 \leqslant \tau_Q \leqslant 16 \wedge \chi_Q=(\ )\}$

P?W(y) $\longrightarrow$

{**false**}

skip

{**false**}

later

$\{16 \leqslant \tau_Q \leqslant 17 \wedge \chi_Q=(\ )\}$

nosyn

$\{16 \leqslant \tau_Q \leqslant 17 \wedge \chi_Q=(Q, \tau_Q)\}$

y:=2

$\{y=2 \wedge 17 \leqslant \tau_Q \leqslant 19 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge \chi_Q=(Q, \tau_Q-t_1))\}$

end

$\{y=2 \wedge 17 \leqslant \tau_Q \leqslant 19 \wedge (\exists t_1)(1 \leqslant t_1 \leqslant 2 \wedge \chi_Q=(Q, \tau_Q-t_1))\}$

Warunek skojarzenia

Łatwo zauważyć, że

possmatching (Q!W(x), P?W(y)) $\Longleftrightarrow$ **false**

z czego natychmiast wynika prawdziwość warunku skojarzenia. Warto też zauważyć, że w tym przypadku także asercja pre, określona dla przypadku 1, jest również tożsamościowo fałszywa

$$\text{pre} \Longleftrightarrow x=0 \wedge 0 \leqslant \tau_P \leqslant 9 \wedge \chi_P=(\ ) \wedge 10 \leqslant \tau_Q \leqslant 16 \wedge \chi_Q=(\ ) \wedge \tau_P=\tau_Q \Longleftrightarrow \text{false}.$$

Warunek przeterminowania

Z semantyki procesów wynika, że

$$\text{timeout (TH1)} \Longleftrightarrow 9 \leqslant \tau_P \leqslant 10 \wedge x=0 \wedge \chi_P=(\ )$$

oraz

$$\text{timeout (TH2)} \Longleftrightarrow 16 \leqslant \tau_P \leqslant 17 \wedge \chi_Q=(\ ),$$

co, podobnie jak w przypadku 2, prowadzi do stwierdzenia, że warunek przeterminowania jest spełniony.

Na podstawie analizy przedstawionych przypadków można przedstawić dowód poprawności programu przy założeniu dowolnych wartości stałych a, b. Postać dowodów sekwencyjnych będzie następująca:

P: $\{x=0 \wedge \tau_P=a \wedge \chi_P=(\,)\}$

th 8 wait

$\{x=0 \wedge a \leq \tau_P \leq a+9 \wedge \chi_P=(\,)\}$

$Q!W(x) \longrightarrow$

$\{(\text{allowed1} \wedge (\exists t_1)(1 \leq t_1 \leq 2 \wedge a+1 \leq \tau_P \leq a+11 \wedge$

$\chi_P=(P,Q,W,!,0,\tau_P-t_1))\}$

$x:=1$

$\{\text{allowed1} \wedge (\exists t_1)(\exists t_2)(1 \leq t_1 \leq 2 \wedge 1 \leq t_2 \leq 2 \wedge a+2 \leq \tau_P \leq a+13 \wedge$

$\chi_P=(P,Q,W,!,0,\tau_P-t_1-t_2))\}$

later

$\{x=0 \wedge a+9 \leq \tau_P \leq a+10 \wedge \chi_P=(\,)\}$

nosyn

$\{\text{allowed2} \wedge a+9 \leq \tau_P \leq a+10 \wedge \chi_P=(P,\tau_P)\}$

$x:=2$

$\{\text{allowed2} \wedge a+10 \leq \tau_P \leq a+12 \wedge (\exists t_1)(1 \leq t_1 \leq 2 \wedge \chi_P=(P,\tau_P-t_1)\}$

end

$\{\text{allowed1} \wedge a+2 \leq \tau_P \leq a+13 \wedge (\exists t_1)(\exists t_2)(1 \leq t_1 \leq 2 \wedge 1 \leq t_2 \leq 2 \wedge$

$\chi_P=(P,Q,W,!,0,\tau_P-t_1-t_2)) \vee \text{allowed2} \wedge a+10 \leq \tau_P \leq a+12 \wedge$

$(\exists t_1)(\exists t_2)(1 \leq t_1 \leq 2 \wedge 1 \leq t_2 \leq 2 \wedge \chi_P=(P,\tau_P-t_1-t_2))\}$

Q: $\{\tau_Q=b \wedge \chi_Q=(\,)\}$

th 5 wait

$\{b \leq \tau_Q \leq b+6 \wedge \chi_Q=(\,)\}$

$P?W(y) \longrightarrow$

$\{\text{allowed1} \wedge (\exists t_1)(1 \leq t_1 \leq 2 \wedge b+1 \leq \tau_Q \leq b+8 \wedge \chi =(Q,P,Q,?,0,\tau_Q-$

$-t_1))\}$

skip

$\{\text{allowed1} \wedge (\exists t_1)(1 \leq t_1 \leq 2 \wedge b+1 \leq \tau_Q \leq b+8 \wedge \chi_Q=(Q,P,Q,?,0,\tau_Q-$

$-t_1))\}$

later

$\{b+6 \leq \tau_Q \leq b+7 \wedge \chi_Q=(\,)\}$

nosyn

$\{\text{allowed2} \wedge b+6 \leq \tau_Q \leq b+7 \wedge \chi_Q = (Q, \tau_Q)\}$

y:=2

$\{\text{allowed2} \wedge y=2 \wedge b+7 \leq \tau_Q \leq b+9 \wedge (\exists t_1)(1 \leq t_1 \leq 2 \wedge \chi_Q = (Q, \tau_Q - t_1))\}$

end

$\{\text{allowed1} \wedge b+1 \leq \tau_P \leq b+8 \wedge (\exists t_1)(1 \leq t_1 \leq 2 \wedge \chi_Q = (Q, \tau_Q - t_1)) \vee$

$\text{allowed2} \wedge y=2 \wedge b+7 \leq \tau_Q \leq b+9 \wedge (\exists t_1)(1 \leq t_1 \leq 2 \wedge \chi_Q = (Q, \tau_Q - t_1))\}$

gdzie dodatkowe asercje allowed1, allowed2 są zdefiniowane następująco:

$$\text{allowed1} \Longleftrightarrow \max(a,b) \leq \min(a+9, b+6),$$
$$\text{allowed2} \Longleftrightarrow \max(a,b) > \min(a+8, b+5).$$

Łatwo sprawdzić, że po przyjęciu wartości a, b ustalonych dla analizowanych przypadków otrzymuje się poprzednio otrzymane dowody.

Kolejny przykład ma zwrócić uwagę na sytuacje, w których mimo prawdziwości asercji typu

$$\text{pre}(CM) \wedge \text{pre}(\overline{CM}) \wedge \text{inv}(P,Q)$$

skojarzenie dynamiczne pomiędzy CM, $\overline{CM}$ zajść nie może.

Niech będzie dany następujący program złożony z trzech procesów

```
P1:: x:=0, y:=1;
     th 3 wt P2!W(x) ⟶ skip
          □ P3!U(y) ⟶ skip
     lt skip end
   ||
P2:: W;
     u:=1; v:=0;
     th 3 wt P1?W(u) ⟶ skip
          □ P3?W(v) ⟶ skip
     lt skip end
   ||
P3:: U;
     th 6 wt P1?U(z) ⟶ skip
          □ P2?W(1) ⟶ skip
     lt z:=0 end
```

Przyjmując dla poszczególnych procesów warunki początkowe postaci

$$\text{pre}(P_i) \Longleftrightarrow \tau_{P_i} = 0 \qquad (i = 1,2,3)$$

i zakładając środowisko wykonawcze takie jak w poprzednim przykładzie, łatwo można obliczyć, że

$$\text{pre}\ (P2!W(x)) \Longleftrightarrow \text{pre}\ (P3!U(y)) \Longleftrightarrow x=0 \wedge y=1 \wedge 2 \leqslant \tau_{P_1} \leqslant 8,$$

$$\text{pre}\ (P1!W(u)) \Longleftrightarrow \text{pre}\ (P3?W(v)) \Longleftrightarrow u=1 \wedge v=0 \wedge 2 \leqslant \tau_{P_2} \leqslant 8,$$

$$\text{pre}\ (P1?U(z)) \Longleftrightarrow \text{pre}\ (P2!W(1)) \Longleftrightarrow 0 \leqslant_{P_3} \leqslant 5$$

Dla każdej z trzech skojarzonych syntaktycznie par istnieją takie stany, że asercje poprzednio wymienionego typu są prawdziwe. Jednak, jak wynika z semantyki procesów, skojarzenie dynamiczne pomiędzy P2!W(x) a P1?W(u) niegdy nie nastąpi. Zachodzą bowiem asercje

$$\text{possmatching}\ (P2!W(x), P1?W(u)) \Longleftrightarrow \underline{\text{false}},$$

$$\text{possmatching}\ (P3!U(y), P1?U(z)) \Longleftrightarrow x=0 \wedge y=1 \wedge 2 \leqslant \tau_{P1} \leqslant 5,$$

$$\text{possmatching}\ (P3?W(v), P2!W(1)) \Longleftrightarrow u=1 \wedge v=0 \wedge 2 \leqslant \tau_{P2} \leqslant 5.$$

4.4. Systemy dowodzenia poprawności częściowej

Konkretny system dowodzenia poprawności częściowej PS_p wymaga, zgodnie z ustaleniami w podrozdz. 4.2, 4.3, przedstawienia analitycznej postaci asercji będących odpowiednikami asercji possmatching oraz timeout, używanych w dowodzeniu warunków zgodności. Konkretny system dowodzenia będzie zatem składać się ze zbioru aksjomatów A1-A6, reguł R1-R7 oraz asercji pmatching i tout zdefiniowanych w taki sposób, że dla dowolnego programu, dowolnych warunków początkowych oraz dla każdej pary skojarzonych syntaktycznie w tym programie instrukcji komunikacji CM, $\overline{CM}$ zachodzi implikacja

$$\text{possmatching}\ (CM,\overline{CM}) \Longrightarrow \text{pmatching}\ (\overline{CM},CM)$$

a dla dowolnej, czasowo uwarunkowanej, instrukcji komunikacji TH zachodzi implikacja

$$\text{timeout}\ (TH) \Longrightarrow \text{tout}\ (TH).$$

Takie określenie konkretnego systemu dowodzenia umożliwia wprowadzanie i stosowanie różnych systemów, a także umożliwia porównywanie systemów.

Jeżeli PS_p1, PS_p2 są dwoma systemami, to PS_p1 będzie lepszym niż PS_p2 wtedy, gdy zachodzą implikacje

$$\text{pmatching1}\ (CM,\overline{CM}) \Longrightarrow \text{pmatching2}\ (CM,\overline{CM}),$$

$$\text{tout1}\ (TH) \Longrightarrow \text{tout2}\ (TH).$$

Sens użytej tutaj "dobroci" systemu sprowadza się do tego, że wszystkie twierdzenia, które daje się o dowolnym programie dowieść w systemie PS_p2, daje się także dowieść w systemie PS_p1, natomiast nie zawsze zachodzi wynikanie odwrotne. Zatem najlepszym - dalej pokażemy, że

zupełnym - jest system, dla którego

$$\text{pmatching}(CM,\overline{CM}) \Longleftrightarrow \text{possmatching}(CM,\overline{CM})$$
$$\text{tout}(TH) \Longleftrightarrow \text{timeout}(TH),$$

a najgorszym taki, że

$$\text{pmatching}(CM,\overline{CM}) \Longleftrightarrow \underline{\text{true}}$$
$$\text{tmatching}(CM,\overline{CM}) \Longleftrightarrow \underline{\text{true}}.$$

Asercje possmatching oraz timeout, jeśli dałoby się przedstawić w analitycznej formie, muszą uwzględnić semantykę zachodzenia skojarzeń bądź upływu czasu przeterminowania, wyrażaną przez definicje 3.10 oraz 3.11. Zatem w postaci tych asercji powinny potencjalnie znaleźć odzwierciedlenie postacie wszystkich asercji poprzedzających instrukcje komunikacji w programie. Tak silne powiązania pomiędzy wymienionymi asercjami budzą wątpliwość w sens poszukiwania postaci asercji possmatching oraz timeout, zamiast tego łatwiej poszukiwać - a co ważniejsze - znacznie łatwiej stosować, asercji zastępczych pmatching oraz tout na podstawie przesłanek heurystycznych. Poniżej przedstawia się przykład takiego heurystycznego podejścia przy wyznaczeniu postaci warunku przeterminowania.

Obliczenia, dla których może nie nastąpić komunikacja z innymi procesami w instrukcji

$$TH = \underline{\text{th}} \text{ v } \underline{\text{wt}} \; \Box_{i=1..n} \; b_i ; \; CM_i \longrightarrow AC_i \; \underline{\text{lt}} \; AC \; \underline{\text{end}}$$

w procesie P dzielą się na dwie grupy.

Pierwszą grupę stanowią obliczenia, w których odcinek czasu v oczekiwania na komunikację jest położony rozłącznie względem innych odcinków oczekiwania na komunikację we wszystkich procesach, które zawierają synktaktycznie skojarzone instrukcje komunikacji. Drugą grupę stanowią te obliczenia, w których, pomimo koincydencji takich odcinków czasu, komunikacja nie zachodzi w procesie P, ponieważ procesy z nim skojarzone są równocześnie skojarzone z innymi procesami i właśnie z nimi prowadzą komunikację.

Niech CM_i będzie instrukcją komunikacji, stanowiącą element czasowo uwarunkowanej instrukcji komunikacji TH w procesie P. Przez $\overline{CM_i}$ będzie oznaczana skojarzona z nią instrukcja komunikacji, a przez $CM_i \cdot part$ będzie oznaczany identyfikator procesu zawierającego $\overline{CM_i}$.

Wprowadza się następującą asercję pomocniczą

$$\text{match}(CM_i,\overline{CM_i}) \Longleftrightarrow (\exists t)(\tau_P - v - 1 \leqslant t \leqslant \tau_P - 1 \wedge$$
$$\text{pre}(CM_i) \wedge \text{pre}(\overline{CM_i}) \wedge \text{inv}(P, CM_i \cdot part))\ [t/\tau_P]$$

Jeżeli s jest takim stanem, że pre (nosyn)(s) = tt, to asercja

match $(CM_i, \overline{CM}_i)(s) = tt$ wtedy, gdy w odcinku czasu $[s(\tau_P)-v-1, s(\tau_P)-1]$ może zajść skojarzenie dynamiczne pomiędzy CM_i a $\overline{CM}_i$.

Jeśli założy się, że każda instrukcja komunikacji może być wykonana co najwyżej jeden raz, asercja tout1(TH) dla danego stanu s będzie prawdziwa wtedy i tylko wtedy, gdy dla każdej alternatywy instrukcji TH jest spełniony jeden z warunków:

1. $\vDash \neg b_i(s)$
2. $\vDash b_i(s)$

oraz nie istnieje taka instrukcja wejścia/wyjścia $\overline{CM}_i$ w pewnym procesie Q, że zachodzi

$$\text{match } (CM, \overline{CM}) \Longrightarrow \text{pre } (\underline{\text{nosyn}})$$

3. Jeżeli $\vDash b_i(s)$ i istnieje $\overline{CM}_i$, o której jest mowa w warunku 2, to spełniona jest alternatywa

$$(\text{before } (CM_i, \overline{CM}_i) \vee \text{after } (CM_i, \overline{CM}_i) \vee \text{during } (CM_i, \overline{CM}_i))(s)$$

gdzie asercje before, after, during są zdefiniowane następująco:

before $(CM_i, \overline{CM}_i) \Longleftrightarrow$

$$(\exists t > 0)(\forall t')(\tau_P - v - \bar{v} - t - 1 \leqslant t' \leqslant \tau_P - v - t - 1 \Longrightarrow \text{pre } (\overline{CM}_i)\ [t'/\tau_Q])$$

after $(CM_i, \overline{CM}_i) \Longleftrightarrow$

$$(\exists t \geqslant 0)(\forall t')(\tau_P + t \leqslant t' \leqslant \tau_P + \bar{v} + t \Longrightarrow \text{pre } (\overline{CM}_i)\ [t'/\tau_Q])$$

during $(CM_i, \overline{CM}_i) \Longleftrightarrow$

$$(\exists CM_j, \overline{CM}_j)(\exists t)(\tau_P - v - \bar{v} - 1 \leqslant t \leqslant \tau_P - v + 1 \Longrightarrow \text{match } (CM_j, \overline{CM}_j)\ [t/\tau_Q]) \vee$$

$$(\exists CM_J, \overline{CM}_J)(\exists t)(\tau_P - v - 1 \leqslant t \leqslant \tau_P - 1 \Longrightarrow \text{match } (CM_j, \overline{CM}_j) \wedge \text{pre } (\underline{\text{wait}}\ (\bar{v}))\ [t/\tau_Q])$$

natomiast v, $\bar{v}$ są odcinkami czasu przeterminowania związanymi z CM_i, $\overline{CM}_i$; CM_j jest elementem tej samej instrukcji $\overline{TH}$ w procesie Q, która zawiera instrukcję $\overline{CM}_i$; $\underline{\text{wait}}(\bar{v})$ jest instrukcją pomocniczą użytą w dowodzie w instrukcji $\overline{TH}$.

Kształt asercji tout1 (TH) wynika z następujących rozważań. Jeżeli s jest ustalonym stanem, spełniającym asercję pre (nosyn), to czas przeterminowania upłynie tylko wtedy, gdy dla każdej spośród otwartych alternatyw, tzn. takich, że $[\![b_i]\!]_s = tt$, może nie nastąpić komunikacja z innym procesem. Stwierdzenie to wyjaśnia istnienie pierwszego z warunków. Kiedy jednak komunikacja procesu poprzez instrukcję CM_i

może nie nastąpić? Oczywiście wtedy, gdy nie istnieje taka instrukcja $\overline{CM}_i$, że dla stanu s prawdziwa jest asercja match $(CM_i, \overline{CM}_i)$. Implikacja w warunku 2 oznacza, że chodzi tylko o te stany s, które również spełniają asercję pre (nosyn). Jeżeli natomiast istnieje taka instrukcja $\overline{CM}_i$, to także jest możliwe, że komunikacja pomiędzy CM_i a $\overline{CM}_i$ nie nastąpi z powodu zajścia jednej z trzech sytuacji w trakcie obliczeń programu:

- odcinek czasu przeterminowania związany z instrukcją $\overline{CM}_i$ poprzedza odcinek czasu związany z CM_i i jest z nim rozłączny (opisuje to asercja before),
- ten sam odcinek czasu następuje po odcinku czasu przeterminowania związanym z CM_i i również jest z nim rozłączny (opisuje to asercja before),
- gdy odcinki te nie są rozłączne, istnieje instrukcja CM_j będąca elementem tej samej czasowo uwarunkowanej instrukcji komunikacji $\overline{TH}$, która zawiera $\overline{CM}_i$, taka że CM_j jest skojarzona z pewną instrukcją $\overline{CM}_j$ w procesie innym od procesu P, a przy tym skojarzenie to następuje nie później niż skojarzenie CM_i z $\overline{CM}_i$ (opisuje to asercja during).

Natomiast wyjaśnienie kształtu asercji before, after, during jest następujące.

Niech s będzie stanem spełniającym asercję pre (nosyn). Stan ten wyznacza pewną wartość dla zmiennej τ_p oraz odcinek czasu $[\tau_p\text{-}v\text{-}1, \tau_p\text{-}1]$, w którym trwało oczekiwanie na komunikację z innym procesem. Jeżeli $\overline{CM}_i$ jest instrukcją taką, że match $(CM_i, \overline{CM}_i)(s) = tt$, to oznacza tylko potencjalną możliwość komunikacji pomiędzy CM_i, $\overline{CM}_i$, tzn. istnieje takie obliczenie, że asercja pre (CM_i) $[t/\tau_Q(s)] = tt$ dla pewnej chwili t należącej do podanego wyżej odcinka czasu. Nie wyklucza to jednak, że proces Q może rozpocząć i zakończyć oczekiwanie na komunikację przed chwilą $\tau_p\text{-}v\text{-}1$. Zatem jeżeli proces Q może oczekiwać na komunikację w dowolnym przedziale $[\tau_p\text{-}v\text{-}\bar{v}\text{-}t\text{-}1, \tau_p\text{-}v\text{-}t\text{-}1]$, gdzie t jest dowolną wartością dodatnią, to istnieje obliczenie, dla którego CM_i nie będzie zrealizowane. To, że proces Q może oczekiwać na komunikację w podanym wyżej odcinku czasu oznacza, że asercja pre (CM_i) $[t'/\tau_Q](s)$ musi być prawdziwa dla każdego t' z tego przedziału, co wyjaśnia kształt asercji before $(CM_i, \overline{CM}_i)$.

Do asercji after stosuje się analogiczne rozumowanie, ale rozważa się położenie odcinka czasu oczekiwania przez proces Q na prawo od chwili $\tau_p\text{-}1$.

Postać asercji during opisuje takie obliczenia, gdy odcinki czasu oczekiwania na komunikację przez CM_i, $\overline{CM}_i$ mają część wspólną, lecz w instrukcji $\overline{TH}$, obok $\overline{CM}_i$, występuje instrukcja CM_j, która dynami-

cznie kojarzy się z instrukcją $\overline{CM}_j$ w procesie $R = CM_j \cdot part$, różnym od procesu P, przy czym skojarzenie to następuje nie później niż następuje skojarzenie CM_i z $\overline{CM}_i$. Należy tu rozważyć dwa możliwe przypadki: pierwszy - gdy proces Q rozpocznie oczekiwanie na komunikację wcześniej niż proces P oraz drugi - gdy proces P rozpocznie oczekiwanie przed procesem Q. W pierwszym przypadku, pomimo że istnieje wspólna chwila dla obu odcinków oczekiwań, czyli zachodzi $match(CM_i, \overline{CM}_i)$, może nie dojść do komunikacji, gdyż w odcinku czasu $[\tau_P\text{-}v\text{-}\bar{v}\text{-}1,\ \tau_P\text{-}v\text{-}1]$ proces Q kojarzy się dynamicznie z procesem R. Możliwość taka istnieje, gdy prawdziwa jest asercja:

$$(\exists CM_j, \overline{CM}_j)(\exists t)(\tau_P\text{-}v\text{-}\bar{v}\text{-}1 \leqslant t \leqslant \tau_P\text{-}v\text{-}1 \wedge match\ (CM_j, \overline{CM}_j)\ [t/\tau_Q])$$

Drugi przypadek odzwierciedla sytuację, gdy w odcinku czasu $[\tau_P\text{-}v\text{-}1, \tau_P\text{-}1]$, dla pewnej chwili t z tego przedziału, rozpoczyna oczekiwanie na komunikację proces Q, lecz do komunikacji z udziałem CM_i, $\overline{CM}_i$ nie dochodzi, gdyż dla tej samej chwili t instrukcja $\overline{CM}_j$ jest dynamicznie skojarzona z instrukcją CM_j w procesie R. Możliwość taka istnieje, gdy prawdziwa jest asercja:

$$(\exists CM_j, \overline{CM}_j)(\exists t)(\tau_P\text{-}v\text{-}1 \leqslant t \leqslant \tau_P\text{-}1 \wedge match\ (CM_j, \overline{CM}_j) \wedge pre\ (\underline{wait}\ (\bar{v}))[t/\tau_Q])$$

gdzie asercja pre (wait ($\bar{v}$)) jest asercją stojącą bezpośrednio przed pomocniczą instrukcją wait ($\bar{v}$) w dowodzie procesu Q. Asercja pre (wait ($\bar{v}$)) określa zbiór stanów, dla których proces Q rozpoczyna oczekiwanie na komunikację, natomiast pre (wait ($\bar{v}$)) $[t/\tau_Q]$, dla t z przedziału $[\tau_P\text{-}v\text{-}1, \tau_P\text{-}1]$, oznacza zbiór tych stanów, dla których oczekiwanie procesu Q rozpoczyna się po rozpoczęciu oczekiwania na komunikację przez proces P.

Wyjaśnienia dotyczące kształtu asercji tout1 (TH) były przeprowadzone przy założeniu, że każda czasowo uwarunkowana instrukcja komunikacji może być wykonana, w trakcie realizacji programu, co najwyżej jeden raz. Jeżeli zrezygnuje się z tego założenia, to uwzględniająca to asercja tout2 (TH) staje się bardziej złożona. Wynika to stąd, że biorąc pod uwagę instrukcję TH oraz pewien stan s spełniający asercję pre (nosyn), możemy tylko wtedy stwierdzić, że dla stanu s może nastąpić przeterminowanie instrukcji TH (tzn. wykonanie nosyn), gdy - z punktu widzenia każdej instrukcji składowej CM_i - istnieje takie obliczenie programu, dla którego przy wielokrotnym wykonywaniu instrukcji TH nie nastąpi skojarzenie dynamiczne.

Niech $h_1, \ldots, h_n$ będzie łańcuchem historii lokalnych w procesie P oraz niech α będzie pewną asercją w dowodzie procesu P.

Definicja 4.6. Łańcuch historii lokalnych (definicja 3.17) ch = $= h_1, \ldots, h_n$ w procesie P nazywa się łańcuchem względem asercji α w stanie s, jeżeli są spełnione następujące warunki:

1) $\alpha\,[h_i/\chi_P](s) = tt$ dla $i=1,\ldots,n$

2) dla dowolnych h', h'' takich, że

$$h' \prec h_i \quad \text{oraz} \quad h_n \prec h''$$

zachodzi

$$\alpha\,[h'/\chi_P](s) = ff \quad \text{oraz} \quad \alpha\,[h''/\chi_P](s) = ff$$

3) dla dowolnego h takiego, że

$$h_i \prec h \prec h_{i+1} \quad \text{dla} \quad i=1,..,n-1$$

zachodzi

$$\alpha\,[h/\chi_P](s) = ff.$$

Fakt, że ch jest takim łańcuchem w procesie P będzie zapisywany w postaci prawdziwości asercji

$$chain_{\alpha,P}(ch)(s)$$

a to, że historia h_i jest elementem łańcucha ch, w postaci

$$h_i \in ch. \qquad \blacksquare$$

Jeżeli rozważy się stan s spełniający asercję pre (nosyn) w procesie P, to pewien łańcuch historii lokalnych $h_1, \ldots, h_n$ (być może pusty) w procesie Q względem asercji match $(CM_i, \overline{CM_i})$ w stanie s odzwierciedla takie obliczenie programu, dla którego n-krotnie nastąpi sytuacja, że CM_i oraz $\overline{CM_i}$ mogą być ze sobą skojarzone dynamicznie. Stąd znajdują uzasadnienie następujące definicje.

Asercja tout2 (TH) jest prawdziwa dla danego stanu s wtedy i tylko wtedy, gdy dla każdej alternatywy instrukcji TH jest spełniony jeden z warunków 1, 2, takich jak w definicji asercji tout1, oraz - zamiast warunku 3 - ma być spełniony warunek

3a) jeżeli $\vDash b_i(s)$ oraz istnieje $\overline{CM_i}$, o którym mówi warunek 2, to istnieje taki łańcuch historii ch w procesie Q, zawierającym instrukcję $\overline{CM_i}$, że

$$\vDash chain_{match\ (CM_i,\overline{CM_i}),Q}\ (ch)(s)$$

oraz dla każdego $h \in ch$ zachodzi

$$bad\ (CM_i,\overline{CM_i},h)(s)$$

gdzie:

$$\text{bad}\ (CM_i, \overline{CM}_i, h) \Longleftrightarrow$$

$$(\text{before}\ (CM_i, \overline{CM}_i) \vee \text{after}\ (CM_i, \overline{CM}_i) \vee \text{during}\ (CM_i, \overline{CM}_i))\ [h/\chi_Q].$$

Zachodzi, oczywiście, implikacja

$$\text{tout2(TH)} \Longrightarrow \text{tout1(TH)}$$

dla dowolnej instrukcji TH. Łatwo podać przykłady, że asercja tout2(TH) określa warunki wystarczające, ale nie konieczne, w których zachodzi przeterminowanie, czyli

$$\text{timeout (TH)} \Longrightarrow \text{tout2 (TH)}.$$

4.5. Dowodzenie poprawności słabej i silnej

Wykazanie słabej poprawności programu wymaga udowodnienia, że program jest częściowo poprawny, a ponadto, że zawsze się kończy. Zakończenie programu może nastąpić z powodu zerwania obliczeń lub prawidłowego osiągnięcia końca programu. System dowodzenia poprawności słabej PS_w będzie różnić się od systemu dowodzenia poprawności częściowej PS_0 tylko dodatkową regułą dotyczącą instrukcji iteracji.

R8. Reguła zakończenia instrukcji iteracji

$$\frac{\{\delta(n) \wedge n>0 \wedge b_i\}\ \underline{\text{test}}\ (DO); AC_i \{(\exists m<n)\,\delta(m)\}\quad (i=1..n),\quad \delta(n) \wedge n>0 \Longrightarrow \text{BOOL(DO)},\ \delta(0) \Longrightarrow \neg\text{BOOL(DO)}}{\{(\exists n)\,\delta(n)\}\ \underline{\text{do}}\ \square_{i=1..n}\ b_i \rightarrow AC_i\ \underline{\text{end}}\ \{\delta(0)\}}$$

W regule R8 występuje dodatkowa asercja δ ze zmienną wolną n, która jest różna od pozostałych zmiennych występujących w programie i która przyjmuje wartości całkowite nieujemne.

Zatem schemat systemu PS_w będzie składać się ze zbioru aksjomatów A1-A6 oraz zbioru reguł R1-R8.

Silna poprawność programu dodatkowo wymaga stwierdzenia, że zakończenie programu nie nastąpi w wyniku zerwania obliczeń. Powodem zerwania obliczeń może być wykonanie instrukcji zerwania, alternatywy lub uwarunkowanej czasowo instrukcji komunikacji. Po przyjęciu założenia, że w programie nie używa się instrukcji zerwania, silną poprawność zapewniają dwie dodatkowe reguły stanowiące modyfikację reguł R2 oraz R4.

R2a. Reguła instrukcji alternatywy

$$\frac{\alpha \Longrightarrow \text{BOOL(IF)},\quad \{\alpha \wedge b_i\}\ \underline{\text{test}}\ (IF);\ AC_i\ \{\beta\}\ \ (i=1..n)}{\{\alpha\}\ \underline{\text{if}}\ \square_{i=1..n}\ b_i \rightarrow AC_i\ \underline{\text{end}}\ \{\beta\}}$$

R4a. Reguła czasowo uwarunkowanej instrukcji komunikacji

$$\frac{\alpha \Longrightarrow \mathrm{BOOL}\ (TH)\quad \{\alpha \wedge b_i\}\ \underline{test}\ (TH);\ \underline{wait}\ (t); CM_i; AC_i\ \{\beta\}\ (i=1..n),\quad \{\alpha\}\ \underline{test}\ (TH);\ \underline{later}\ (t);\ \underline{nosyn};\ AC\ \{\beta\}}{\{\alpha\}\ \underline{th}\ t\ \underline{wt}\ \Box_{i=1..n}\ b_i; CM_i \longrightarrow AC_i\ \underline{lt}\ AC\ \underline{end}\ \{\beta\}}$$

Zatem schemat systemu PS_s dowodzenia silnej poprawności programów będzie składać się ze zbioru aksjomatów A1-A6 oraz zbioru reguł R1, R2a, R3, R4a, R6-R8.

Dowodzenie słabej i silnej poprawności programów w języku RTCSP nie wnosi, jak widać, nowych problemów w stosunku do dowodzenia odpowiednich własności dla programów sekwencyjnych.

4.6. Uwagi w dowodzeniu poprawności całkowitej

Program silnie poprawny, którego obliczenia nie blokują się, jest programem całkowicie poprawnym. Zgodnie z twierdzeniem 3.2, obliczenia programu w języku RTCSP nie blokują się, a zatem program silnie poprawny jest jednocześnie programem całkowicie poprawnym. Nie ma więc konieczności rozbudowy systemu dowodzenia PS_s. Warto natomiast rozważyć konsekwencje modyfikacji uwarunkowanej czasowo instrukcji komunikacji, polegającej na wprowadzeniu nieograniczonego oczekiwania na zsynchronizowanie się procesów. Po przyjęciu, że do zbioru liczb całkowitych dołącza się nowy element **infinite** (nieskończony) i dopuszcza się następujące jego użycie

$$\underline{th}\ \underline{infinite}\ \underline{wt}\ \Box_{i=1..n}\ b_i; CM_i \longrightarrow AC_i\ \underline{lt}\ AC_0\ \underline{end}$$

- czyli

$$\underline{th}\ \underline{infinite}\ \underline{wt}\ \Box_{i=1..n}\ b_i; CM_i \longrightarrow AC_i\ \underline{end}.$$

gdyż alternatywa AC_0 nie będzie nigdy realizowana, powstaje możliwość występowania blokad podczas obliczeń. Istnienie blokady będzie charakteryzować się tym, że wystąpi co najmniej jeden proces oczekujący na komunikację, pozostałe procesy zakończą swoje działanie.

Analiza możliwości powstania blokad dla języka CSP była prowadzona przez Apta [8] oraz Levina [141], [142]. Idea analizy została w obu przypadkach zaczerpnięta z pracy Owickiej [171] i polega na znalezieniu zbioru potencjalnie możliwych konfiguracji programu, w których następuje oczekiwanie procesów na zsynchronizowanie się z innymi procesami. Wykazanie, że program nie zablokuje się polega na pokazaniu, że konfiguracje takie nie mogą powstać albo też, że w przypadku ich powstania istnieje możliwość postępu obliczeń, tzn. że istnieje co najmniej para syn-

chronizujących się procesów. Istotne jest to, że wykazanie możliwości powstania lub braku blokady jest odnoszone do zbudowanego już dowodu częściowej poprawności programu.

Łatwo zauważyć, że (w przypadku języka RTCSP) fakt, iż dana instrukcja czasowo uwarunkowanej komunikacji nie będzie mogła zostać zsynchronizowana przed upływem czasu przeterminowania z innym procesem opisuje asercja timeout (TH). Zatem jeżeli timeout (TH) nie jest asercją tożsamościowo fałszywą, to oznacza - przy założeniu, że czas przeterminowania w instrukcji TH jest nieograniczony - iż istnieje możliwość powstania blokady.

5. NIESPRZECZNOŚĆ SCHEMATU SYSTEMU DOWODZENIA POPRAWNOŚCI CZĘŚCIOWEJ

Niesprzeczność jest podstawową cechą, którą powinien się charakteryzować każdy system dowodzenia poprawności programów. Niesprzeczność schematu systemu dowodzenia poprawności częściowej programów PS_p oznacza, że dowolna specyfikacja Φ, która jest twierdzeniem w PS_p, tzn. ma dowód w PS_p, jest prawdziwa w ustalonej interpretacji J, czyli: jeżeli

$$\vdash_{PS_p} \Phi, \text{ to } \models_J \Phi.$$

Aby dowieść niesprzeczności schematu systemu dowodzenia, wystarczy pokazać, że każdy aksjomat jest prawdziwy w ustalonej interpretacji J oraz że każda reguła dowodzenia jest niesprzeczna, tzn. dla przesłanek prawdziwych w interpretacji J wynika prawdziwość wniosku w interpretacji J.

Bezpośrednie, wykorzystanie powyższego stwierdzenia przy dowodzeniu niesprzeczności schematu systemu PS_p nie jest możliwe, gdyż reguła złożenia równoległego R7 jest w istocie metaregułą, a aksjomat dla instrukcji komunikacji A6 może być użyty tylko w tych dowodach sekwencyjnych, które są przesłankami reguły R7. Należy bowiem zauważyć, że nie można wyłącznie na podstawie aksjomatu A6 sensownie przypisywać jakiegokolwiek znaczenia instrukcjom komunikacji. Aby pokonać te trudności, zostanie wykorzystany zabieg, zaproponowany przez Apta [12], polegający na transformacji schematu systemu PS_p w pewien równoważny system PS_p^*. Transformacja polega na zastąpieniu aksjomatu A6 i metareguły R7 jedną nową regułą R7*. Zatem można bezpośrednio sprawdzić prawdziwość i niesprzeczność wszystkich aksjomatów i reguł schematu systemu dowodzenia PS_p, oprócz A6 oraz R7.

Lemat 5.1

W schemacie systemu dowodzenia PS_p aksjomaty A1-A5 są prawdziwe, a reguły R1-R6 są niesprzeczne w dowolnej interpretacji J. ■

Dowód

Dowód jest przedstawiony w załączniku.

Wprowadzenie nowej reguły R7* poprzedza lemat, który ustala własności asercji będących elementami dowodu sekwencyjnego.

Lemat 5.2

Niech J będzie ustaloną interpretacją, PC procesem składowym programu. Specyfikacja $\{\alpha\}$ PC $\{\beta\}$ ma dowód sekwencyjny w schemacie systemu PS_p wtedy i tylko wtedy, gdy dla każdej instrukcji składowej AC procesu PC istnieją asercje pre (AC), post (AC) takie, że w interpretacji J są prawdziwe następujące formuły:

1. $\alpha \Longrightarrow$ pre (PC), post (PC) $\Longrightarrow \beta$.
2. Jeżeli AC = skip, to

   pre (AC) $\Longrightarrow$ post (AC).
3. Jeżeli AC = abort, to

   post (AC) $\Longrightarrow$ true.
4. Jeżeli AC = x:=e, to

   pre (AC) $\Longrightarrow (\forall t)(ASS_e[t/dur] \Longrightarrow$ post (AC) $[e/x, \tau+t/\tau])$.
5. Jeżeli AC = rcl x, to

   pre (AC) $\Longrightarrow$ post (AC) $[\tau/x]$.
6. Jeżeli AC = AC1;AC2, to

   pre (AC) $\Longrightarrow$ pre (AC1),
   post (AC1) $\Longrightarrow$ pre(AC2),
   post (AC2) $\Longrightarrow$ post (AC).
7. Jeżeli AC jest pomocniczą instrukcją test, wait lub later, to:

   pre (AC) $\Longrightarrow (\forall t)(D_{AC}[t/dur] \longrightarrow$ post (AC) $[\tau+t/\tau])$,

gdzie D_{AC} stanowi predykat opisujący czas realizacji instrukcji AC.

8. Jeżeli AC = if $\Box_{i=1..n}\ b_i \longrightarrow AC_i$ end, to

   pre (AC) $\wedge\ b_i \Longrightarrow$ pre (test AC; AC_i) (i=1..n)
9. Jeżeli AC = do $\Box_{i=1..n}\ b_i \longrightarrow AC_i$ end, to

   pre (AC) $\wedge\ b_i \Longrightarrow$ pre (test AC; AC_i) (i=1..n),

post (test (AC); AC_i) $\Longrightarrow$ pre (AC) (i=1..n),

pre (AC) $\wedge$ $\neg$BOOL (AC) $\Longrightarrow$
$(\forall t)(TEST_{AC}[t/dur] \Longrightarrow$ post (AC) $[\tau+t/\tau])$.

10. Jeżeli AC = th v wt $\Box_{i=1..n}$ $b_i, CM_i \longrightarrow AC_i$ lt AC_0 end, to

pre (AC) $\wedge$ $b_i \Longrightarrow$ pre (test (AC); wait (v); $CM_i; AC_i$) (i=1..n),

post (test (AC); wait (v); $CM_i; AC_i$) $\Longrightarrow$ post (AC) (i=1..n),

pre (AC) $\wedge$ BOOL (AC) $\Longrightarrow$ pre (test (AC); later (v); nosyn; AC_0),

post (test (AC); later (v); nosyn; AC_0) $\Longrightarrow$ post (AC).

■

Dowód

Dowód jest przedstawiony w załączniku.

Definicja 5.1 Niech dla danego dowodu sekwencyjnego $\{\alpha_i\}$ PC_i $\{\beta_i\}$, SC($\{\alpha_i\}$ PC_i $\{\beta_i\}$) oznacza zbiór wszystkich asercji, które spełniają warunki 1-10 lematu 5.2 (i=1..n). Dalej niech PC($\{\alpha_i\}$ PC_i $\{\beta_i\}$, i=1..n) oznacza zbiór warunków zgodności dla dowodów sekwencyjnych $\{\alpha_i\}$ PC_i $\{\beta_i\}$ (i=1..n). Zmodyfikowana reguła złożenia równoległego R7* ma postać

$$\frac{SC(\{\alpha_i\}\ PC_i\ \{\beta_i\})\ \ (i+1..n),\ PC(\{\alpha_i\}\ PC_i\ \{\beta_i\},\ i=1..n)}{\{\alpha_1 \wedge .. \wedge \alpha_n\}\ \|_{i=1..n}\ PC_i\ \{\beta_1 \wedge .. \wedge \beta_n\}}$$

■

Schemat systemu dowodzenia PS_p złożony z aksjomatów A1-A6 i reguł R1-R7 oraz systemu PS_p^* złożony z aksjomatów A1-A5 i reguł R1-R6, R7* są równoważne. Specyfikacja Φ ma dowód w schemacie systemu PS_p wtedy i tylko wtedy, gdy Φ ma dowód w systemie PS_p^*. Twierdzenie to łatwo dowieść z lematu 5.2. Dlatego w dalszym ciągu zamiast dowodzić niesprzeczności schematu systemu PS_p wystarczy dowieść niesprzeczności systemu PS_p^*, co w tym miejscu oznacza konieczność pokazania niesprzeczności reguły R7*.

W kolejnym lemacie będą wykorzystywane dwie pomocnicze konstrukcje after i before. Niech AC' będzie instrukcją składową w procesie PC = = PD;AC. Nieformalnie konstrukcja after (AC', AC) oznacza tę część procesu PC, która może być wykonana po zrealizowaniu instrukcji AC'. Natomiast before (AC', AC) oznacza tę część procesu PC, która pozostaje do wykonania bezpośrednio przed wykonaniem AC'. Zatem before (AC', AC) = AC'; after (AC', AC). Formalnie konstrukcje te definiowane są indukcyjnie ze względu na strukturę procesu.

Definicja 5.2 Niech PC = PD; AC, wtedy

1. after (AC', AC) = skip

2. Jeżeli AC = **if** $\square_{i=1..n}$ $b_i \longrightarrow AC_i$ **end** oraz

a) AC' = **test** (AC), to

after $(AC', AC) = AC_i$ $(i=1..n)$,

b) AC' jest instrukcją składową pewnej instrukcji AC_i, to

after (AC', AC) = **after** (AC', AC_i).

3. Jeżeli AC = **do** $\square_{i=1..n}$ $b_i \longrightarrow AC_i$ **end** oraz

a) AC' = **test** (AC), to

after $(AC', AC) = AC_i ; AC$ $(i=1..n)$

lub

after (AC', AC) = **skip**

b) AC' jest instrukcją składową pewnej instrukcji AC_i, to

after (AC', AC) = **after** (AC', AC);AC.

4. Jeżeli AC = **th** v **wt** $\square_{i=1..n}$ $b_i ; CM_i \longrightarrow AC_i$ **lt** AC_0 **end** oraz

a) AC' = **test** (AC), to

after (AC', AC) = **wait** (v); CM_i; AC_i $(i=1..n)$

lub

after (AC', AC) = **later** (v); **nosyn**; AC_0,

b) AC' = **wait** (v), to

after $(AC', AC) = CM_i ; AC_i$ $(i=1..n)$,

c) AC' = **later** (v), to

after (AC', AC) = **nosyn**; AC_0,

d) AC' = CM_i, $(i=1..n)$, to

after $(AC', AC) = AC_i$,

e) AC' = **nosyn**, to

after $(AC', AC) = AC_0$,

f) AC' jest instrukcją składową w AC_i $(i=0..n)$, to

after (AC', AC) = **after** (AC', AC_i).

5. Jeżeli AC = AC1;AC2 oraz

a) AC' jest instrukcją składową w AC1, to

after (AC', AC) = **after** $(AC', AC1; AC2)$,

b) AC' jest instrukcją składową w $AC2$, to

$$\underline{\text{after}}\ (AC', AC) = \underline{\text{after}}\ (AC', AC2). \qquad \blacksquare$$

Lemat 5.3

Niech $hglob = \|_{i=1..n} hloc_i$ będzie historią oddziaływań taką, że

$$\langle \|_{i=1..n} AC_i,\ s,\ c\rangle \xrightarrow{hglob} * \langle \|_{i=1..n} \overline{AC}_i,\ \bar{s},\ \bar{c}\rangle.$$

Ponadto niech dla asercji początkowych α_i $(i=1..n)$ zachodzi

$$\vDash_J (\alpha_1 \wedge \ldots \wedge \alpha_n)(s).$$

1. Jeżeli $\overline{AC}_i$ jest takie, że dla pewnej instrukcji AC', będącej składową w AC_i zachodzi

$$\overline{AC}_i = \underline{\text{before}}\ (AC', AC_i),$$

to dla asercji $pre\ (AC')$ zachodzi

$$\vDash_J pre\ (AC')(\bar{s}).$$

2. Jeżeli $\overline{AC}_i$ jest takie, że dla pewnej instrukcji AC', będącej składową w AC_i zachodzi

$$\overline{AC}_i = \underline{\text{after}}\ (AC', AC_i),$$

to dla asercji $post\ (AC')$ zachodzi

$$\vDash_J post\ (AC')(\bar{s}). \qquad \blacksquare$$

Dowód

Dowód jest przedstawiony w załączniku.

Bezpośrednio z przedstawionego lematu wynika, że reguła $R7^*$ jest niesprzeczna. Istotnie, jeżeli

$$\vDash_J (\alpha_1 \wedge \ldots \wedge \alpha_n)(s).$$

to zgodnie z lematem

$$\vDash_J post\ (AC_i)(\bar{s}) \quad (i=1..n),$$

gdyż $\underline{\text{after}}\ (AC_i, AC_i) = \underline{\text{skip}}$. Ponieważ formuły

$$post\ (AC_i) \Longrightarrow \beta_i \quad (i=1..n)$$

są przesłankami reguły $R7^*$, prawdziwymi w interpretacji J, zatem

$$\vDash_J (\beta_1 \wedge \ldots \wedge \beta_n)(\bar{s}),$$

a stąd

$$\models_J \{(\alpha_1 \wedge \ldots \wedge \alpha_n)(s)\} \;\|_{i=1..n}\; AC_i \;\{(\beta_1 \wedge \ldots \wedge \beta_n)(\bar{s})\},$$

co oznacza niesprzeczność reguły. W ten sposób pokazano twierdzenie.

Twierdzenie 5.1

Schemat systemu PS_p dowodzenia poprawności częściowej jest niesprzeczny. ■

6. ZUPEŁNOŚĆ SCHEMATU SYSTEMU DOWODZENIA POPRAWNOŚCI CZĘŚCIOWEJ

Zgodnie z rozważaniami z podrozdz. 4.2, zupełność schematu systemu dowodzenia jest rozumiana w sensie relatywnej zupełności Cooka. Zupełność oznacza, że jeżeli dowolna specyfikacja Φ jest prawdziwa w interpretacji J, to specyfikacja ta jest twierdzeniem w PS_p, czyli: jeżeli $\models_J \Phi$, to $\vdash_{PS_p} \Phi$. Okazuje się jednak, że - w odróżnieniu od niesprzeczności - schemat systemu PS_p nie jest zupełny dla dowolnej interpretacji J. Określenie klasy interpretacji, dla której system PS_p jest zupełny, wymaga wprowadzenia dodatkowych pojęć [12]. Niech

$$sp_{J,p}(\alpha, PR) = Asem_p\, [\![PR]\!]\, ([\alpha]_J),$$

$$wp_{J,p}(PR, \beta) = \{s \mid Asem_p\, [\![PR]\!]\, (s) \leqslant [\beta]_J\}.$$

Łatwo zauważyć, że zbiory te, nazywane zbiorem najsilniejszych warunków końcowych i najsłabszych warunków początkowych, spełniają następujące równoważności

$$\models_{J,p} \{\alpha\}\ PR\ \{\beta\} \Longrightarrow [\alpha]_J \subseteq wp_{J,p}(PR,\beta) \Longleftrightarrow sp_{J,p}(\alpha, PR).$$

Definicja 6.1 Język asercji ASSERT jest wyrażalny względem interpretacji J, języka programowania RTCSP i środowiska wykonawczego ENV, jeżeli dla dowolnej asercji α i programu PR istnieje taka asercja $\beta \in$ ASSERT, że

$$[\beta]_J = sp_{J,p}(\alpha, PR).$$

Jeżeli interpretacja J jest taka, że język ASSERT jest wyrażalny względem J, RTCSP oraz ENV, to będzie to zapisywane w postaci

$$J \in Ex\ (ASSERT, RTCSP, ENV).$$ ■

Należy zatem pokazać, że schemat systemu PS_p jest relatywnie zupełny w klasie interpretacji wyrażalnych.

Dla przejrzystości rozważań dowód relatywnej zupełności schematu systemu PS_p wygodnie będzie podzielić na dwie części: pierwsza pokazuje relatywną zupełność sekwencyjnej części schematu (dowodzenie własności procesów nie zawierających instrukcji komunikacji), druga - równoległej części schematu.

Lemat 6.1

Schemat systemu dowodzenia częściowej poprawności składowych sekwencyjnych programów w języku RTCST (procesy bez instrukcji komunikacji), złożony z aksjomatów A1-A5 oraz reguł wnioskowania R1-R3, R5, R6, jest relatywnie zupełny dla dowolnej interpretacji

$$J \in Ex\ (ASSERT, RTCSP, ENV).$$ ■

Dowód

Dowód jest przedstawiony w załączniku.

Dowód zupełności schematu systemu PS_p ma polegać na pokazaniu, że jeżeli $\models_J \{\alpha\}\ PR\ \{\beta\}$, to $\overline{PS_p} \vdash \{\alpha\}\ PR\ \{\beta\}$. Krokiem wstępnym do takiego dowodu będzie pokazanie, że jeżeli $\models_J \alpha^*\ PR\ \{\beta\}$, to $\overline{PS_p} \vdash \{\alpha^*\}\ PR\ \{\beta\}$, gdzie α^* jest pewną modyfikacją asercji α. Zmodyfikowana asercja α^* ma postać

$$\alpha^* \iff iks_1 = x_1 \wedge \ldots \wedge iks_N = x_N \wedge \alpha \wedge$$

$$chi_1 = \chi_1 \wedge \ldots \wedge chi_n = \chi_n \wedge tau_1 = \tau_1 \wedge \ldots \wedge tau_n = \tau_n$$

gdzie $\{x_1, \ldots, x_N\} = FV(PR)$, $\chi_1, \ldots, \chi_n$ oraz $\tau_1, \ldots, \tau_n$ są zmiennymi pomocniczymi historycznymi i czasowymi w procesach składowych programu $PR = \|_{i=1..n} PC_i$, natomiast $iks_1, \ldots, iks_N$, $chi_1, \ldots, chi_n$ $tau_1, \ldots, tau_n$ są nowo wprowadzonymi zmiennymi, różnymi od zmiennych wymienionych poprzednio.

Łatwo zauważyć, że jeżeli $\models_J \{\alpha\}\ PR\ \{\beta\}$, to także $\models_J \{\alpha^*\}\ PR\ \{\beta\}$, gdyż $\alpha^* \Longrightarrow \alpha$. Jeżeli pokaże się $\overline{PS_p} \vdash \{\alpha^*\}\ PR\ \{\beta\}$, to stąd, na podstawie reguły zastąpienia R6, wynika, że $\overline{PS_p} \vdash \{\alpha\}\ PR\ \{\beta\}$, a zatem otrzyma się dowód zupełności.

Definicja 6.2. Niech AC będzie dowolną instrukcją składową procesu PC_j w programie $PR = \|_{i=1..n} PC_i$. Dla takich instrukcji AC definiuje się asercje pre (AC) oraz post (AC) w sposób następujący:

$$\models_J pre\ (AC)(s) \iff$$

$$(\exists s', s'' \in Astates)(\exists c', c'' \in Clocks)(\exists\ \|_{i=1..n} hloc_i)$$

$$(\models_J \alpha^*(s') \wedge PROP\ (s', c') \wedge$$

$$\langle \|_{i=1..n} PC_i, s', c' \rangle \xrightarrow{\|_{i=1..n} hloc_i}{}^{*} \langle \|_{i=1..n} PC_i', s'', c'' \rangle \wedge$$

$$\text{PROP}\ (s'',c'')\ \wedge\ S''\,|_{FV(PC_j)\cup\{\tau_j,\chi_j\}} = s|_{FV(PC_j)\cup\{\tau_j,\chi_j\}}\ \wedge$$

$$s(\chi_j) = hloc_j\ \wedge\ PC_j''' = \underline{\text{before}}\ (AC,PC_j)),$$

$$\vDash_J \text{post}\ (AC)(s) \iff$$

$$(\exists s',s'' \in \text{Astates})(\exists c',c''' \in \text{Clocks})(\exists\ {\scriptstyle i=1\,||\,..n}\ hloc_i)$$

$$(\vDash_J \alpha^*(s')\ \wedge\ \text{PROP}\ (s',c')\wedge$$

$$\langle {\scriptstyle i=1..n}\ PC_i,s',c'\rangle \xrightarrow{\ {\scriptstyle i=1\,||\,..n}\ hloc_i\ } {}^*\langle {\scriptstyle i=1\,||\,..n}\ PC_i,s''',c'''\rangle\wedge$$

$$\text{PROP}\ (s''',c''')\wedge s'''|_{FV(PC_j)\cup\{\tau_j,\chi_j\}} = s|_{FV(PC_j)\cup\{\tau_j,\chi_j\}}\wedge$$

$$s(\chi_j) = hloc_j\ \wedge PC_i'' = \underline{\text{after}}\ (AC,PC_j)).\qquad \blacksquare$$

Asercje pre(AC) oraz post (AC) są dobrze zdefiniowane, to znaczy, że są one elementami języka ASSERT i są wyrażalne względem interpretacji J. Aby przekonać się o tym, można przeprowadzić następujące rozważania. Niech

$$PR = {\scriptstyle i=1\,||\,..n}\ PC_i$$

będzie ustalonym programem. Zbiór instrukcji składowych AC dla dowolnego procesu PC_i (i=1..n) jest skończony. Skończony jest zatem również zbiór PRS, którego elementami są ${\scriptstyle i=1\,||\,..n}\ PC_i'$, gdzie PC_j' jest równe $\underline{\text{after}}\ (AC,PC_j)$ lub $\underline{\text{before}}\ (AC,PC_j)$ dla pewnej instrukcji składowej AC w ustalonym procesie PC_j.

Wystarczy zauważyć, że wyżej zdefiniowane asercje pre (AC), post (AC), gdzie AC jest składową instrukcją w wybranym procesie PC_j, są równoważne następującym definicjom

$$[\text{pre}\ (AC)]_J \iff \bigcup_{PR\in PRS_b} sp_{J,p}(\alpha^*,PR),$$

$$[\text{post}\ (AC)]_J \iff \bigcup_{PR\in PRS_a} sp_{J,p}(\alpha^*,PR),$$

gdzie $PRS_b = \{PR \in PRS \mid PR = {\scriptstyle i=1\,||\,..n}\ PC_i' \wedge PC_j' = \underline{\text{before}}\ (AC,PC_j)\}$,

$PRS_a = \{PR \in PRS \mid PR = {\scriptstyle i=1\,||\,..n}\ PC_i' \wedge PC_j' = \underline{\text{after}}\ (AC,PC_j)\}$.

Ponieważ $\alpha \in$ ASSERT, więc również $\alpha^* \in$ ASSERT na mocy przyjętych założeń, że język ASSERT zawiera rachunek kwantyfikatorów i arytmetykę liczb całkowitych. Z definicji 6.1, wynika, że dla $PR \in PRS_a \cup PRS_b$

istnieją asercje wyrażające zbiory $sp_{J,p}(\alpha^*,PR)$, a stąd także skończone sumy takich zbiorów.

Należy teraz pokazać, że tak zdefiniowane asercje pre i post mogą być przesłankami reguły R7*. Sprawdzenie, że spełniają one warunki 1-10 lematu 5.2 jest proste, dlatego zostaje tu pominięte, natomiast trudniejsze jest sprawdzenie, że spełniają one warunek skojarzenia (definicja 4.4) oraz warunek przeterminowania (definicja 4.5). Jeżeli pokaże się, że asercje określone w definicji 6.2 mogą być przesłankami reguły R7*, będzie to oznaczać, że istnieje dowód

$$\vdash_{\overline{PS}_p} \alpha^* \{PR\} \{\beta\},$$

co na mocy poprzednich ustaleń zakończy dowód zupełności schematu systemu PS_p.

Definicja 6.3. Niech s będzie stanem w programie $PR = \|_{i=1..n} PC_i$, natomiast AC_i niech będzie instrukcją składową procesu PC_i dla $i \in A$, gdzie A jest pewnym niepustym podzbiorem zbioru $\{1, \ldots, n\}$. Zbiór instrukcji $\{AC_i \mid i \in A\}$ nazywa się s-osiągalny, gdy $(\exists s', s'' \in Astates)$ $(\exists c', c'' \in Clocks)(\exists \|_{i=1..n} hloc_i)$

$$(\vdash_J \alpha^*(s') \wedge PROP(s',c')$$

$$\langle \|_{i=1..n} PC_i, s', c' \rangle \xrightarrow{\|_{i=1..n} hloc_i} * \langle \|_{i=1..n} PC_i'', s'', c'' \rangle \wedge$$

$$(PC_i'' = AC_i \text{ dla } i \in A) \wedge PROP(s'',c'') \wedge$$

$$s'' \mid FV(PC_i) \cup \{\tau_j, \chi_j\} = s \mid FV(PC_i) \cup \{\tau_j, \chi_i\} \text{ dla } i \in A).$$

■

Lemat 6.2

Jeżeli pojedyncza instrukcja AC_i, dla $i \in A$, jest s-osiągalna, to zbiór instrukcji $\{AC_i \mid i \in A\}$ jest również s-osiągalny.

Dowód

Dowód jest przedstawiony w załączniku.

Lemat 6.3

Asercje pre i post wprowadzone przez definicję 6.2 spełniają warunek skojarzenia określony przez definicję 4.4.

■

Dowód

Dowód jest przedstawiony w załączniku.

Lemat 6.4

Asercje pre i post, wprowadzone przez definicję 6.2, spełniają warunek przeterminowania określony definicją 4.5. ■

Dowód

Dowód jest przedstawiony w załączniku.

Z dotychczasowych rozważań wynika, że dla dowolnej składowej AC w procesie PC_i (i=1..n) asercje pre i post, wprowadzone w definicji 4.2, są takie, że

$$\vdash_J \{pre\ (AC)\}\ AC\ \{post\ (AC)\},$$

przy czym AC jest instrukcją różną od instrukcji komunikacji oraz od pustej instrukcji komunikacji. Natomiast dla instrukcji komunikacji są spełnione warunki zgodności (definicje 4.4 i 4.5). Ponadto

$$\alpha^* \Longrightarrow \alpha$$

oraz po przyjęciu założenia, że $\vdash_J \{\alpha\}\ \|_{i=1..n}\ PC_i\ \{\beta\}$,

$$(post\ (PC_1) \wedge \ldots \wedge post\ (PC_n)) \Longrightarrow \beta.$$

Zbiór asercji pre i post, spełniając warunki lematu 6.1 oraz warunki zgodności, stanowi zbiór asercji wymaganych w systemie PS_p^* do udowodnienia częściowej poprawności programu $\|_{i=1..n}\ PC_i$, czyli

$$\vdash_{PS_p^*} \{\alpha\}\ \|_{i=1..n}\ PC_i\ \{\beta\}.$$

Stąd wynika twierdzenie.

Twierdzenie 6.1

Schemat systemu PS_p dowodzenia poprawności częściowej programów jest relatywnie zupełny dla interpretacji $J \in Ex(ASSERT, RTCSP, ENV)$. ■

7. POPRAWNOŚĆ CZĘŚCIOWA PRZYKŁADOWEGO PROGRAMU

7.1. Przykładowy program

Przykładowy program rozważany w niniejszym rozdziale można uważać za odmianę klasycznego zadania producenta i konsumenta. Składa się on z dwóch procesów, z których jeden jest nadawcą pewnych komunikatów, drugi zaś ich odbiorcą. Treść programu przedstawia poniższy tekst.

```
P1::NAD;
    lnd:=0;stop1:=false;
    do ¬stop1 ∧ lnd<n →
        prod(x);nad1:=false;l:=0;
        do ¬nad1 ∧l1<2 →
           through 5 wait
             true;P2!ODB(x) → nad1:=true;lnd:=lnd+1
             later
```

```
            l1:=l1+1
          end
        end;
        if l1=2 → stop1:=false
         □ l1≠2 → odb1:=false;k1:=0;
          do ¬ odb1 ∧ k1<2 →
             through 5 wait
               true;P2?NAD( ) → odb1:=true
               later
               k1:=k1+1
             end
          end;
          if k1=2 → stop1:=true
           □ k1≠2 → skip
          end
        end
    end
  ||
P2::ODB;
   lod:=0;stop2:=false;
    do ¬ stop2 →
       odb2:=false;l2:=0;
       do ¬ odb2 ∧ l2<3 →
          through 5 wait
            true;P1?ODB(y) → odb2:=true
            later
            l2:=l2+1
          end
       end;
       if l2=3 → stop2:=true
        □ l2≠3 → nad2:=false;k2:=0;kons(y);
          do ¬ nad2 ∧ k2<2 →
             through 5 wait
               true;P1!NAD( ) → nad2:=true;lod:=lod+1
               later
               k2:=k2+1
             end;
             if k2=3 → stop2:=true
              □ k2≠3 → skip
             end
          end
       end
    end
```

W tekście procesu P_1 występuje wywołanie procedury prod (x), a w tekście P_2 - procedury kons (y). O procedurze prod (x) zakłada się, że jedynym efektem jej wywołania jest wygenerowanie pewnej wartości całkowitoliczbowej i podstawienie pod zmienną x, natomiast o procedurze kons (y) zakłada się, że w pewien sposób konsumuje wartość y, nie modyfikując przy tym wartości zmiennych występujących w procesie. O obu tych procedurach zakłada się, że czas ich wykonania jest niedeterministyczny - taki sam jak dla instrukcji podstawienia. Procesy wymieniają między sobą dwa rodzaje komunikatów: jeden - to liczby całkowite wysyłane z P_1, poprzez port ODB, do konsumpcji w P_2, drugi - to komunikaty puste, odbierane w P_1, poprzez port NAD, i stanowiące potwierdzenie przez P_2 o skonsumowaniu ostatnio otrzymanej liczby całkowitej.

Środowisko wykonawcze programu definiuje następujący zbiór predykatów:

$$ASS_e\ (dur) \Longleftrightarrow 1 \leqslant dur \leqslant 2,$$

$$TEST_{IF}\ (dur) \Longleftrightarrow dur=0,$$

$$TEST_{DO}\ (dur) \Longleftrightarrow dur=0,$$

$$TEST_{TH}\ (dur) \Longleftrightarrow dur=0,$$

$$SEND_e\ (dur) \Longleftrightarrow dur=2.$$

Ze względu na dalsze potrzeby wprowadza się następujące oznaczenia pomocnicze. Jeżeli χ jest zmienną historyczną, to len (χ) oznacza długość historii χ, natomiast $len_{comm}(\chi)$ oznacza długość podhistorii χ składającą się ze wszystkich oddziaływań λ takich, że λ.type= = comm. Jeżeli len (χ) > 0, to przez last (χ) będzie oznaczany ostatni element historii χ; podobnie jeżeli $len_{comm}(\chi) > 0$, to $last_{comm}(\chi)$ będzie oznaczać ostatnie oddziaływanie λ w historii χ takie, że λ.type = comm.

W dowodzie procesu P_1 będą wykorzystane następujące predykaty

$$PT1(a,b) \Longleftrightarrow len_{comm}(\chi_1) > 0 \Longrightarrow$$

$$last_{comm}(\chi_1).time+a \leqslant \tau_1 < last_{comm}(\chi_1).time+b,$$

$$TO1(a,b) \Longleftrightarrow last\ (\chi_1).type=nosyn \wedge a \leqslant last\ (\chi_1).time \leqslant b,$$

$$NN(i,j,k) \Longleftrightarrow \neg nad1 \wedge lnd=i \wedge \neg stop1 \wedge l1=j \wedge len_{comm}(\chi_1)=k,$$

$$NA(i,j,k) \Longleftrightarrow nad1 \wedge lnd=i \wedge \neg stop1 \wedge l1=j \wedge len_{comm}(\chi_1)=k,$$

$$NO(i,j,k) \Longleftrightarrow \neg odb \wedge lnd=i \wedge \neg stop \wedge k1=j \wedge len_{comm}(\chi_1)=k,$$

$$ND(i,j,k) \Longleftrightarrow odb \wedge lnd=i \wedge \neg stop \wedge k1=j \wedge len_{comm}(\chi_1)=k,$$

gdzie τ_1, χ_1 są zmiennymi pomocniczymi w procesie P_1. Natomiast w dowodzie procesu P_2 występować będą predykaty:

$PT2(a,b) \Longleftrightarrow len_{comm}(\chi_2) > 0 \Longrightarrow$

$last_{comm}(\chi_2)\cdot time+a \leqslant \tau_2 \leqslant last_{comm}(\chi_2)\cdot time+b,$

$TO2(a,b) \Longleftrightarrow last\ (\chi_2)\cdot type=nosyn \wedge a \leqslant last\ (\chi_2)\cdot time \leqslant b,$

$OO(i,j,k) \Longleftrightarrow \neg odb2 \wedge lod=i \wedge \neg stop2 \wedge 12=j \wedge len_{comm}(\chi_2)=k,$

$OD(i,j,k) \Longleftrightarrow odb2 \wedge lod=i \wedge \neg stop2 \wedge 12=j \wedge len_{comm}(\chi_2)=k,$

$ON(i,j,k) \Longleftrightarrow \neg nad2 \wedge lod=i \wedge \neg stop \wedge k2=j \wedge len_{comm}(\chi_2)=k,$

$OA(i,j,k) \Longleftrightarrow nad \wedge lod=i \wedge \neg stop \wedge k2=j \wedge len_{comm}(\chi_2)=k,$

gdzie τ_2, χ_2 są zmiennymi pomocniczymi w procesie P_2.

Pokażemy, że dowody sekwencyjne

$\{(\tau_1=0 \wedge len\ (\chi_1)=0\}$

P_1:: ... treść jak wyżej ...

$\{lnd=m \wedge \neg stop \wedge len_{comm}(\chi_1)=2m \wedge PT1(3,4)\}$

oraz

$\{\tau_2=0 \wedge len\ (\chi_2)=0\}$

P_2:: ... treść jak wyżej ...

$\{lod=m \wedge stop \wedge PT2(27,34)\}$

są zgodne ze sobą, a zatem że cały program jest częściowo poprawny względem specyfikacji

$\{\tau_1=\tau_2\}\ P_1$:: ... || P_2:: ... $\{lod=lnd\}$

Skomentowany tekst procesu P_1 przedstawia się następująco:

P_1:: $\{\tau_1=0 \wedge len\ (\chi_1)=0\}$
lnd:=0;stop1:=false;
$\{lnd=0 \wedge \neg stop1 \wedge 2 \leqslant \tau_1 \leqslant 4 \wedge len\ (\chi_1)=0 \vee$
$(\exists i)(0<i\leqslant m \wedge lnd=i \wedge \neg stop \wedge len_{comm}(\chi_1)=2i \wedge PT1(3,4))\}$
do $\neg stop1 \wedge lnd<m \rightarrow$
$\{lnd=0 \wedge \neg stop \wedge 2 \leqslant \tau_1 \leqslant 4 \wedge len\ (\chi_1)=0 \vee$
$(\exists i)(0<i\leqslant m \wedge lnd=i \wedge \neg stop \wedge len_{comm}(\chi_1)=2i \wedge PT1(3,4))\}$
prod (x);nad1:=false;l1:=0;
$\{NN(0,0,0) \wedge 5 \leqslant \tau_1 \leqslant 10 \wedge len\ (\chi_1)=0 \vee$
$(\exists i)(0<i<m \wedge NN(i,0,2i) \wedge PT1(6,10)) \vee$
$(\exists j)(0\leqslant j\leqslant 1 \wedge NA(1,j,1) \wedge PT1(4,6)) \vee$
$(\exists i)(1<i\leqslant m \wedge NA(i,0,2i-1) \wedge PT1(4,6))\}$
do $\neg nad1 \wedge l1<2 \rightarrow$

```
{NN(0,0,0) ∧ 5 ≤ τ1 ≤ 10 ∧ len (χ1)=0 ∨
(∃i)(0 < i < m ∧ NN(i,0,2i) ∧ PT1(6,10))}
through 5 wait
   {NN(0,0,0) ∧ 5 ≤ τ1 ≤ 15 ∧ len (χ1)=0 ∨
   (∃i)(0 < i < m ∧ NN(i,0,2i) ∧ PT(6,15))}
   P2!ODB(x) →
      {NN(0,0,1) ∧ PT1(2,2) ∨
       (∃i)(0 < i < m ∧ NN(i,0,2i+1) ∧ PT1(2,2))}
       nad1:=true; lnd:=lnd+1;
         {(∃j)(0 ≤ j ≤ 1 ∧ NA(1,j,1) ∧ PT1(4,6)) ∨
          (∃i)(0 < i < m ∧ NA(i+1,0,2i+1) ∧ PT1(4,6))}
     later
      {NN(0,0,0) ∧ 11 ≤ τ1 ≤ 16 ∧ len (χ1)=0
       (∃i)(0 < i < m ∧ NN(i,0,2i) ∧ PT1(12,16))}
       nosyn1;
       {false}
       l1:=l1+1;
       {false}
  end
   {(∃j)(0 ≤ j ≤ 1 ∧ NA(1,j,1) ∧ PT1(4,6)) ∨
    (∃i)(0 < i < m ∧ NA(i+1,0,2i+1) ∧ PT1(4,6))}
end;
{(∃j)(0 ≤ j ≤ 1 ∧ NA(1,j,1) ∧ PT1(4,6)) ∨
 (∃i)(1 < i ≤ m ∧ NA(i,0,2i-1) ∧ PT1(4,6))}
if l1=2 → stop1:=false
   {false}
□ l1≠2 → odb1:=false; k1:=0
   {(∃i)(1 ≤ i ≤ m ∧ NO(i,0,2i-1) ∧ PT1(6,10)) ∨
    (∃i)(1 ≤ i ≤ m ∧ ND(i,0,2i) ∧ PT1(3,4))}
    do ¬ odb1 ∧ k1 < 2 →
      {(∃i)(1 ≤ i ≤ m ∧ NO(i,0,2i-1) ∧ PT1(6,10))}
       through 5 wait
          {(∃i)(1 ≤ i ≤ m ∧ NO(i,0,2i-1) ∧ PT1(6,15))}
          P2?NAD( )
            {(∃i)(1 ≤ i ≤ m ∧ NO(i,0,2i) ∧ PT1(2,2))}
            odb1:=true;
            {(∃i)(1 ≤ i ≤ m ∧ ND(i,0,2i) ∧ PT1(3,4))}
         later
            {(∃i)(1 ≤ i ≤ m ∧ NO(i,0,2i-1) ∧ PT1(12,16))}
            nosyn2;
            {false}
            k1:=k1+1
```

{false}
end;
{(∃i)(1 ≤ i ≤ m ∧ND(i,0,2i) ∧PT1(3,4))}
end;
{(∃i)(1 ≤ i ≤ m ∧ND(i,0,2i) ∧PT1(3,4))}
if k1=2 → stop1:=true;
{false}
□ k1≠2 → skip
{(∃i)(1 ≤ i ≤ m ∧ND(i,0,2i) ∧PT1(3,4))}
end
{(∃i)(1 ≤ i ≤ m ∧ND(i,0,2i) ∧PT1(3,4))}
end
{(∃i)(1 ≤ i ≤ m ∧ND(i,0,2i) ∧PT1(3,4))}
end
{lnd=m ∧ ¬stop ∧ $len_{comm}(\chi_1)$=2m ∧PT1(3,4)}

Skomentowany tekst procesu P_2 przedstawia się następująco:

P_2:: {τ_2=0 ∧ len (χ_2)=0}
lod:=0; stop2:=false;
{lod=0 ∧ ¬stop2 ∧ 2 ≤ τ_2 ≤ 4 ∧ len (χ_2) =0 ∨
(∃i)(0 < i ≤ m ∧lod=i ∧ ¬stop2 ∧PT2(3,4)) ∨
lod=m ∧stop2 ∧PT2(27,34)}
do ¬ stop2 →
{lod=0 ∧ ¬stop2 ∧ 2 ≤ τ_2 ≤ 4 ∧ len (χ_2) =0 ∨
(∃i)(0 < i ≤ m ∧lod=i ∧ ¬stop2 ∧PT2(3,4))}
odb2:=false; l2:=0;
{00(0,0,0) ∧ 4 ≤ τ_2 ≤ 8 ∧ len (χ_2) =0 ∨
00(0,1,0) ∧ 11 ≤ τ_2 ≤ 12 ∧T02(10,10) ∧ $len_{comm}(\chi_2)$ =0 ∨
(∃i)(0 < i < m ∧00(i,0,2i) ∧PT2(5,8)) ∨
(∃j)(0 ≤ j ≤ 3 ∧00(m,j,2m) ∧PT2(5+7j,8+8j)) ∨
0D(0,0,1) ∧PT2(3,4) ∨
0D(0,1,1) ∧PT2(3,4) ∨
(∃i)(0 < i < m ∧0D(i,0,2i) ∧PT2(3,4))}
do ¬odb2 ∧l2 < 3 →
{00(0,0,0) ∧ 4 ≤ τ_2 ≤ 8 ∧len (χ_2) =0 ∨
00(0,1,0) ∧ 11 ≤ τ_2 ≤ 12 ∧T02(10,10) ∧ $len_{comm}(\chi_2)$ =0 ∨
(∃i)(0 < i < m ∧00(i,0,2i) ∧PT2(5,8)) ∨
(∃j)(0 ≤ j < 3 ∧00(m,j,2m) ∧PT2(5+7j,8+8j))}
through 5 wait
{00(0,0,0) ∧ 4 ≤ τ_2 ≤ 13 ∧len (χ_2) =0 ∨
00(0,1,0) ∧ 11 ≤ τ_2 ≤ 17 ∧T02(10,10) ∧ $len_{comm}(\chi_2)$ =0 ∨

```
    (∃i)(0<i<m ∧ OO(i,0,2i) ∧ PT2(5,13)) ∨
    (∃j)(0≤j<3 ∧ OO(m,j,2m) ∧ PT2(5+7j,8+8j))}
  P1?ODB(y) ⟶
      {OO(0,0,1) ∧ PT2(2,2) ∨
       OO(0,1,1) ∧ PT2(2,2) ∨
       (∃i)(0<i<m ∧ OO(i,0,2i+1) ∧ PT2(2,2))}
      odb2:=true;
      {OD(0,0,1) ∧ PT2(3,4) ∨
       OD(0,1,1) ∧ PT2(3,4) ∨
      (∃i)(0<i<m ∧ OD(i,0,2i+1) ∧ PT2(3,4))}
  later
      {OO(0,0,0) ∧ 10≤τ2 ≤ 14 ∧ len(χ2) =0 ∨
       OO(0,1,0) ∧ 17≤τ2 ≤18 ∧ TO2(10,10) ∧ len_comm(χ2) =0 ∨
       (∃i)(0<i<m ∧ OO(i,0,2i) ∧ PT2(11,14)) ∨
       (∃j)(0≤j<3 ∧ OO(m,j,2m) ∧ PT2(11+7j,14+8j))}
       nosyn3;
       {OO(0,0,0) ∧ τ2=10 ∧ TO2(10,10) ∧ len_comm(χ2) =0 ∨
        (∃j)(0≤j<3 ∧ OO(m,j,2m) ∧ PT2(11+7j,14+8j))}
       12:=12+1
       {OO(0,1,0) ∧ 11≤τ2≤12 ∧ TO2(10,10) ∧ len_comm(χ2) =0 ∨
        (∃j)(1≤j≤3 ∧ OO(m,j,2m) ∧ PT2(5+7j,8+8j))}
  end
  {OO(0,1,0) ∧ 11≤τ2≤12 ∧ TO2(10,10) ∧ len_comm(χ2) =0 ∨
   OD(0,0,1) ∧ PT2(3,4) ∨
   OD(0,1,1) ∧ PT2(3,4) ∨
   (∃i)(0<i<m ∧ OD(i,0,2i+1) ∧ PT2(3,4)) ∨
   (∃j)(1≤j≤3 ∧ OO(m,j,2m) ∧ PT2(5+7j,8+8j))}
end
{OO(m,3,2m) ∧ PT2(26,32) ∨
 OD(0,0,1) ∧ PT2(3,4) ∨
 OD(0,1,1) ∧ PT2(3,4) ∨
 (∃i)(0<i<m ∧ OD(i,0,2i+1) ∧ PT2(3,4))}
if 12=3 ⟶ stop2:=true
    {¬odb2 ∧ lod=m ∧ stop2 ∧ PT2(27,34)}
 □12≠3 ⟶ nad2:=false; k2:=0; kons(y);
   {(∃i)(0≤i≤m ∧ ON(i,0,2i+1) ∧ PT2(6,10)) ∨
    (∃i)(0<i≤m ∧ OA(i,0,2i) ∧ PT2(4,6))}
   do ¬nad2 ∧ k2<2 ⟶
      {(∃i)(0≤i<m ∧ ON(i,0,2i+1) ∧ PT2(6,10))}
   through 5 wait
      {(∃i)(0≤i<m ∧ ON(i,0,2i+1) ∧ PT2(6,15))}
     P1!NAD( ) ⟶
```

```
                {(∃i)(0≤i<m ∧ON(i,0,2i+2) ∧PT2(2,2))}
              nad2:=true; lod:=lod+1
                {(∃i)(0≤i<m ∧OA(i+1,0,2i+2) ∧PT2(4,6))}
            later
                {(∃i)(0≤i<m ∧ON(i,0,2i+1) ∧PT2(12,16))}
              nosyn4;
                {false}
              k2:=k2+1
                {false}
          end;
            {(∃i)(0≤i≤m ∧OA(i,0,2i) ∧PT2(4,6)}
          if k2=3 ⟶ stop2:=true
                {false}
           □ k2≠3 ⟶ skip
                {(∃i)(0<i≤m ∧OA(i,0,2i) ∧PT2(4,6))}
          end
                {(∃i)(0<i≤m ∧OA(i,0,2i) ∧PT2(4,6))}
      end
      {¬odb2 ∧lod=m ∧stop2 ∧PT2(27,34) ∨
       (∃i)(0<i≤m ∧OA(i,0,2i) ∧PT2(4,6))}
end
 {lod=m ∧stop2 ∧PT2(27,34)}
```

7.2. System dowodzenia

Zgodnie z rozważaniami w podrozdz. 4.4 ustalenie systemu dowodzenia polega na wyborze postaci asercji pmatch oraz tout. W ustaleniu systemu dowodzenia wykorzystuje się istotną cechę analizowanego przykładu, mianowicie to, że w każdej czasowo uwarunkowanej instrukcji komunikacji TH występuje tylko jedna instrukcja wejścia/wyjścia. Fakt ten znacznie upraszcza postać asercji pmatch oraz tout.

Przyjmuje się jeszcze jedno założenie upraszczające, że odcinki czasu przeterminowania - oznaczane przez v_i - w procesie P_i są jednakowej długości.

Niech h_i będzie pewną historią lokalną w procesie P_i (i=1..n), i niech h_i', h_i'' będą takimi historiami, że

$$h_i = h_i' \cap h_i''$$

oraz

$$last_{comm}(h_i) = last\ (h_i),$$

czyli wszystkie oddziaływania λ należące do h_i'' są takie, że $\lambda.type$ = nosyn. Dla oddziaływania $\lambda \in h_i''$ niech $[a_i^{\lambda}, b_i^{\lambda}]$ będzie

odcinkiem czasu przeterminowania odpowiadającym zajściu tego oddziaływania, tzn.

$$a_i^{\lambda} = \lambda.\text{time} - v_i - 1,$$

$$b_i^{\lambda} = \lambda.\text{time} - 1.$$

Definiuje się dwie następujące asercje pomocnicze.

Definicja 7.1. Mówi się, że dwie zgodne historie lokalne h_1, h_2 nie kojarzą się ze sobą końcami, co oznacza prawdziwość asercji

$$\text{nomatchtails}\ (h_1, h_2),$$

wtedy i tylko wtedy, gdy

$$\text{last}\ (h_1') = \overline{\text{last}\ (h_2')},$$

oraz dla dowolnego $\lambda_1 \in h_1''$ i dowolnego $\lambda_2 \in h_2''$

$$\left[a_1^{\lambda_1}, b_1^{\lambda_1}\right] \cap \left[a_2^{\lambda_2}, b_2^{\lambda_2}\right] = \emptyset\ .$$ ■

Definicja 7.2. Mówi się, że chwila t jest poza końcem historii h_i, co oznacza prawdziwość asercji

$$\text{offtail}\ (h_i, t),$$

wtedy i tylko wtedy, gdy

$$t \notin \left[a_i^{\lambda}, b_i^{\lambda}\right]$$ ■

dla dowolnego $\lambda \in h_i''$.

Ostatecznie postać asercji pmatching oraz tout wyznaczają następujące definicje.

Definicja 7.3. Jeżeli instrukcje CM, $\overline{\text{CM}}$ są elementami procesów P_1, P_2, to

$$\text{pmatching}\ (\text{CM}, \overline{\text{CM}}) \Longleftrightarrow \text{nomatchtails}\ (\chi_1, \chi_2)$$ ■

Definicja 7.4. Dla instrukcji TH postaci

th v_1 try b;CM → AC lt AC_0 end

w procesie P_1 asercja tout TH jest prawdziwa dla danego stanu s wtedy i tylko wtedy, gdy spełniony jest jeden z podanych niżej warunków:

1. $\vdash \neg b(s)$,
2. $\vdash b(s)$ i nie istnieje syntaktycznie skojarzona instrukcja wejścia/wyjścia $\overline{\text{CM}}$ taka, że

$$\vdash \text{match}\ (\text{CM}, \overline{\text{CM}})(s) \Longrightarrow \text{pre}\ (\underline{\text{nosyn}})(s)$$

3. jeżeli $\vDash b(s)$ i istnieje $\overline{CM}$ taka, że zachodzi implikacja w warunku 2, to istnieje łańcuch historii ch w procesie P_2 zawierającym instrukcję $\overline{CM}$

$$\vDash chain_{match(CM,\overline{CM}),P_2}(ch)(s)$$

taki, że dla każdego elementu h tego łańcucha zachodzi

$$\vDash (nomatch\ (\chi_1,h) \wedge ba(CM,\overline{CM})\ [h/\chi_2])(s),$$

gdzie $chain_{match(CM,\overline{CM}),P_2}(ch)$ określa definicja 4.6, natomiast

$$ba(CM,\overline{CM}) \Longleftrightarrow bef\ (CM,\overline{CM}) \vee aft\ (CM,\overline{CM})$$

oraz

$$bef\ (CM,\overline{CM}) \Longleftrightarrow$$
$$(\exists t > 0)(\forall t')(\tau_1 - v_1 - v_2 - t - 1 \leqslant t' \leqslant \tau_1 - v_1 - t - 1 \Longrightarrow$$
$$pre\ (\overline{CM})\ [t'/\tau_2] \wedge offtail\ (\chi_1,t')),$$
$$aft\ (CM,\overline{CM}) \Longleftrightarrow$$
$$(\exists t \geqslant 0)(\forall t')(\tau_1 + t \leqslant t' \leqslant \tau_1 + v_2 + t \Longrightarrow$$
$$pre\ (\overline{CM})\ [t/\tau_2]).$$

■

Wprowadzone asercje pmatching oraz tout wyznaczają niesprzeczny system dowodzenia poprawności częściowej programów. Wynika to z twierdzenia.

Twierdzenie 7.1.

Zachodzą następujące implikacje:

$$possmatching\ (CM,\overline{CM}) \Longrightarrow pmatching\ (CM,\overline{CM})$$

dla dowolnych syntaktycznie skojarzonych instrukcji wejścia/wyjścia CM, $\overline{CM}$, oraz

$$timeout\ (TH) \Longrightarrow tout\ (TH)$$

dla dowolnej czasowo uwarunkowanej instrukcji komunikacji TH postaci

$$\underline{th}\ v\ \underline{try}\ b;\ CM \longrightarrow AC\ \underline{lt}\ AC_0\ \underline{end}.$$

■

Dowód.

Ze względu na ograniczenie przyjęte na początku tego podrozdziału można przyjąć bez naruszenia ogólności, że program składa się z dwóch procesów P_1, P_2.

Zgodnie z określeniem wprowadzonym w punkcie 4.3.3 asercja possmatching $(CM,\overline{CM})$ jest prawdziwa dla danego stanu s, gdy istnieje obliczenie takie, że $CM,\overline{CM}$ kojarzą się dynamicznie w stanie s. Niech h_1, h_2 będą lokalnie obserwowalnymi historiami oddziaływań towarzyszą-

cymi obliczeniu, które prowadzi do skojarzenia dynamicznego CM, $\overline{CM}$ w stanie s. Oddziaływania h_1, h_2 są oczywiście zgodne (w sensie definicji 3.16), a dla każdego oddziaływania λ_1 należącego do h_1 oraz λ_2 należącego do h_2 takich, że λ_1.type = λ.type = nosyn zachodzą warunki definicji 3.10, co oznacza, że związane z nimi odcinki czasów przeterminowania są rozłączne. Wynika z tego, że nomatchtails(h_1,h_2)(s) = tt, a ponieważ $s(\chi_1) = h_1$, $s(\chi_2) = h_2$, zatem i pmatching(CM,$\overline{CM}$)(s) = tt.

Niech timeout (TH)(s) = tt dla instrukcji TH w procesie P_1 oraz, oczywiście, $s(\tau_1) \leqslant s(\tau_2)$. Zgodnie z określeniem w punkcie 4.3.3, oznacza to, że istnieje obliczenie, reprezentowane przez lokalne historie oddziaływań h_1, h_2 takie, że $s(\chi_1) = h_1$, $s(\chi_2) = h_2$. Ponadto dla instrukcji TH w konfiguracji wyznaczonej przez położenie sterowania w procesie P_1 bezpośrednio przed instrukcją nosyn, zaś w procesie P_2 - bezpośrednio przed pewną instrukcją różną od instrukcji wejście/wyjście bądź bezpośrednio po ostatniej instrukcji tego procesu są spełnione warunki definicji 3.10.

Jeżeli w stanie s warunek b jest fałszywy, to zachodzi tout (TH)(s) = tt. Jeżeli warunek b jest prawdziwy w stanie s oraz jeżeli nie zachodzi implikacja

$$\text{match (CM,}\overline{\text{CM}}\text{)(s)} \Longrightarrow \text{pre (nosyn)(s)}$$

dla żadnej instrukcji CM w procesie P_2, skojarzonej syntetycznie z CM, to oznacza, że nie jest możliwe w tym stanie skojarzenie dynamiczne CM z jakąkolwiek instrukcją. Zgodnie z definicją tout (TH) jest asercją prawdziwą w takim stanie s. Jeżeli natomiast implikacja taka zachodzi dla pewnego CM, to łatwo wykazać, że istnieje łańcuch historii o własnościach wymaganych przez warunek 3 definicji 7.4. Istotnie, jeżeli zachodzi implikacja, to

$$\text{match (CM,}\overline{\text{CM}}\text{)(s)} = \text{tt}$$

oraz

$$\text{match (CM,}\overline{\text{CM}}\text{) } [h_2/\chi_2](s) = \text{tt}$$

bo $s(\chi_2) = h_2$, a więc istnieje łańcuch ch, gdyż h_2 jest jego, co najmniej jednym, elementem.

Z definicji obliczenia reprezentowanego przez h_1, h_2 wynika, że pomiędzy CM a $\overline{CM}$ w czasie od $s(\tau_1)-v_1-1$ do $s_1(\tau_1)-1$ nie zachodzi skojarzenie dynamiczne. Oczywiście, spełniona jest asercja nomatch (λ_1,h_2) w stanie s, a także musi być spełniona asercja ba(CM,$\overline{CM}$) $[h_2/\chi_2](s)$, gdyż w przeciwnym razie z faktu, że

$$\text{bef (CM,}\overline{\text{CM}}\text{) } [h_2/\chi_2] \text{ (s)} = \text{tt}$$

oraz

$$\text{aft } (CM,\overline{CM})\ [h_2/\chi_2]\ (s) = tt$$

wynika, że - wbrew założeniu - w pewnej chwili odcinka czasu $[s(\tau_1)-v\ -1,\ s(\tau_1)-1]$ zachodzi skojarzenie pomiędzy CM a $\overline{CM}$. Podobne rozumowanie jest prawdziwe dla każdego $h < h_2$ takiego, że

$$\text{match } (CM,\overline{CM})\ [h/\chi_2]\ (s) = tt\ .$$

Zatem każdy element łańcucha ch spełnia własności określone przez definicję 7.4, co oznacza, że również w tym przypadku tout (TH)(s) = tt. ■

7.3. Dowody zgodności

Sprawdzenie, że asercje przedstawione w dowodach procesów spełniają odpowiednie aksjomaty i reguły - oprócz reguły R7 - nie nastręcza żadnych trudności, dlatego w dalszej części przykładu ograniczono się do pokazania, że zachodzą wymagane warunki skojarzenia dla instrukcji komunikacji oraz warunki przeterminowania dla instrukcji pomocniczych nosyn.

W programie występują dwie pary syntaktycznie skojarzonych instrukcji komunikacji:

1. P_2!ODB(x) , P_1?ODB(y)
2. P_2?NAD( ) , P_1!NAD( )

Dla instrukcji z pierwszej pary dane są asercje:

$$\text{pre } (P_2!ODB(x)) \Longleftrightarrow NN(0,0,0) \wedge 5 \leqslant \tau_1 \leqslant 15 \wedge \text{len}(\chi_1)=0 \vee (\exists i)(0<i<m \wedge NN(i,0,2i) \wedge PT1(6,15))$$

$$\text{post } (P_2!ODB(x)) \Longleftrightarrow NN(0,0,1) \wedge PT1(2,2) \vee (\exists i)(0<i<m \wedge NN(i,0,2i+1) \wedge PT1(2,2))$$

$$\text{pre } (P_1?ODB(y)) \Longleftrightarrow OO(0,0,0) \wedge 4 \leqslant \tau_2 \leqslant 13 \wedge \text{len}(\chi_2)=0 \vee OO(0,1,0) \wedge 11 \leqslant \tau_2 \leqslant 17 \wedge T02(10,10) \wedge \text{len}_{comm}(\chi_2)=0 \vee (\exists i)(0<i<m \wedge OO(i,0,2i) \wedge PT2(5,13)) \vee (\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(5+7j,8+8j))$$

$$\text{post } (P_1?ODB(y)) \Longleftrightarrow OO(0,0,1) \wedge PT2(2,2) \vee OO(0,1,1) \wedge PT2(2,2) \vee (\exists i)(0<i<m \wedge OO(i,0,2i+1) \wedge PT2(2,2))$$

Prowadząc obliczenia zgodnie z definicją 4.4, otrzymuje się:

$$\text{pre } (P_2!ODB(x)) \wedge \text{pre } (P_1?ODB(y)) \wedge \text{inv } (P_1, P_2) \Longleftrightarrow$$

$$NN(0,0,0) \wedge OO(0,0,0) \wedge len(\chi_1)=0 \wedge len(\chi_2)=0 \wedge 5 \leqslant \tau_1,\tau_2 \leqslant 13 \vee$$

$$NN(0,0,0) \wedge OO(0,1,0) \wedge len(\chi_1)=0 \wedge len_{comm}(\chi_2)=0 \wedge 11 \leqslant \tau_1,\tau_2 \leqslant 15 \wedge$$

$$T02(10,10) \vee$$

$$(\exists i)(0<i<m \wedge NN(i,0,2i) \wedge OO(i,0,2i) \wedge PT1(6,13) \wedge PT2(6,13))$$

Ponieważ $SEND_e(dur) \Longleftrightarrow dur=2$, zatem prawa strona implikacji warunku skojarzenia dla $\lambda = (P_1P_2,!,x,\tau_1)$ ma postać

$$((post\ (P_2!ODB(x)) \wedge post\ (P_1!ODB(y)) \wedge$$

$$\wedge inv\ (P_1,P_2))[\tau_1+2/\tau_1,\tau_2+2/\tau_2])$$

$$[x/y, \chi_1^\frown\lambda/\chi_1, \chi_2^\frown\bar{\lambda}/\chi_2] \Longleftrightarrow$$

$$((NN(0,0,1) \wedge OO(0,0,1) \vee$$

$$NN(0,0,1) \wedge OO(0,1,1) \vee$$

$$(\exists i)(0<i<m \wedge NN(i,0,2i+1) \wedge OO(i,0,2i+1) \wedge PT1(0,0) \wedge PT2(0,0))$$

$$[x/y, \chi_1^\frown\lambda/\chi_1, \chi_2^\frown\bar{\lambda}/\chi_2] \Longleftrightarrow$$

$$NN(0,0,0) \wedge OO(0,0,0) \vee$$

$$NN(0,0,0) \wedge OO(0,1,0) \vee$$

$$(\exists i)(0<i<m \wedge NN(i,0,2i) \wedge OO(i,0,2i))$$

Ostatnia równoważność wynika stąd, że zachodzi

$$NN(i,j,k)[\chi_1^\frown\lambda/\chi_1] \Longleftrightarrow NN(i,j,k-1)$$

oraz

$$PT1(0,0)[\chi_1^\frown\lambda/\chi_1] \Longleftrightarrow true$$

i podobne równoważności zachodzą dla predykatów OO i PT2. Stąd wynika ostatecznie, że

$$pre\ (P_2!ODB(x)) \wedge pre\ (P_1?ODB\ (y)) \wedge inv\ (P_1,P_2) \Longrightarrow$$

$$((post\ (P_2!ODB(x)) \wedge post\ (P_2?ODB(y)) \wedge inv\ (P_1,P_2))$$

$$[\tau_1+2/\tau_1,\tau_2+2/\tau_2])\ [x/y, \chi_1^\frown\lambda/\chi_1, \chi_2^\frown\bar{\lambda}/\chi_2],$$

co oznacza zachodzenie warunku skojarzenia dla pierwszej pary skojarzonych syntaktycznie instrukcji komunikacji. Z drugą parą instrukcji są związane asercje:

$$pre\ (P_2?NAD()) \Longleftrightarrow (\exists i)(1 \leqslant i \leqslant m \wedge NO(i,0,2i-1) \wedge PT1(6,15))$$

$$post\ (P_2?NAD()) \Longleftrightarrow (\exists i)(1 \leqslant i \leqslant m \wedge NO(i,0,2i) \wedge PT1(2,2))$$

$$pre\ (P_1!NAD()) \Longleftrightarrow (\exists i)(0 \leqslant i < m \wedge ON(i,0,2i+1) \wedge PT2(6,15))$$

$$post\ (P_1!NAD()) \Longleftrightarrow (\exists i)(0 \leqslant i < m \wedge ON(i,0,2i+2) \wedge PT2(2,2))$$

Prowadząc, jak poprzednio, odpowiednie obliczenia, otrzymuje się:

$$\text{pre}\ (P_2?NAD()) \wedge \text{pre}\ (P_1!NAD()) \wedge \text{inv}\ (P_1,P_2) \Longleftrightarrow$$
$$(\exists i)(1 \leqslant i \leqslant m-2 \wedge NO(i,0,2i-1) \wedge ON(i-1,0,2i-1) \wedge PT(6,15) \wedge PT2(6,15))$$

oraz dla $\lambda = (P_1,P_2,?,(\ ),\tau_1)$

$$((\text{post}\ (P_2?NAD()) \wedge \text{post}\ (P_1!NAD()) \wedge \text{inv}\ (P_1,P_2))[\tau_1+2/\tau_1,\tau_2+2/\tau_2])$$
$$[\chi_1 \frown \lambda/\chi_1, \chi_2 \frown \bar{\lambda}/\chi_2] \Longleftrightarrow$$
$$(\exists i)(1 \leqslant i \leqslant m-2 \wedge NO(i,0,2i) \wedge ON(i-1,0,2i) \wedge PT1(0,0) \wedge PT2(0,0))$$
$$[\chi_1 \frown \lambda/\chi_1, \chi_2 \frown \bar{\lambda}/\chi_2] \Longleftrightarrow$$
$$(\exists i)(1 \leqslant i \leqslant m-2 \wedge NO(i,0,2i-1) \wedge ON(i-1,0,2i-1))$$

co oznacza, że także dla tej pary zachodzi warunek skojarzenia.

W programie występują cztery instrukcje pomocnicze pustej komunikacji, oznaczone w tekście skomentowanego programu przez nosyn1, ..., nosyn4. Są z nimi związane następujące asercje:

$$\text{pre}\ (\underline{\text{nosyn1}}) \Longleftrightarrow NN(0,0,0) \wedge 11 \leqslant \tau_1 \leqslant 16 \wedge \text{len}(\chi_1)=0 \vee$$
$$(\exists i)(0 < i < m \wedge NN(i,0,2i) \wedge PT1(12,16))$$

$$\text{post}\ (\underline{\text{nosyn1}}) \Longleftrightarrow \text{false}$$

$$\text{pre}\ (\underline{\text{nosyn2}}) \Longleftrightarrow (\exists i)(1 \leqslant i \leqslant m \wedge NO(i,0,2i-1) \wedge PT1(12,16))$$

$$\text{post}\ (\underline{\text{nosyn2}}) \Longleftrightarrow \text{false}$$

$$\text{pre}\ (\underline{\text{nosyn3}}) \Longleftrightarrow OO(0,0,0) \wedge 10 \leqslant \tau_2 \leqslant 14 \wedge \text{len}(\chi_2)=0 \vee$$
$$OO(0,1,0) \wedge 17 \leqslant \tau_2 \leqslant 18 \wedge T02(10,10) \wedge$$
$$\text{len}_{comm}(\chi_2)=0 \vee$$
$$(\exists i)(0 < i < m \wedge OO(i,0,2i) \wedge PT2(11,14)) \vee$$
$$(\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(11+7j,14+8j))$$

$$\text{post}\ (\underline{\text{nosyn3}}) \Longleftrightarrow OO(0,0,0) \wedge \tau_2=10 \wedge T02(10,10) \wedge \text{len}_{comm}(\chi_2)=0 \vee$$
$$(\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(11+7j,14+8j))$$

$$\text{pre}\ (\underline{\text{nosyn4}}) \Longleftrightarrow (\exists i)(0 \leqslant i < m \wedge ON(i,0,2i+1) \wedge PT2(12,16))$$

$$\text{post}\ (\underline{\text{nosyn4}}) \Longleftrightarrow \text{false}$$

W obliczeniach warunków przeterminowania dla instrukcji nosyn1 oraz nosyn3 występuje asercja match $(P_2!ODB(x),P_1?ODB(y))$, a dla instrukcji nosyn2 oraz nosyn4 - asercja match $(P_2?NAD(\ ),P_1!NAD())$. Asercje te mają następującą postać:

$$\text{match}\ (P_2!ODB(x),P_1?ODB(y)) \Longleftrightarrow$$
$$(\exists t)(\tau_1-6 \leqslant t \leqslant \tau_1-1 \wedge (\text{pre}\ (P_2!ODB(x)) \wedge \text{pre}\ (P_1?ODB(y)) \wedge$$

$$\text{inv}\ (P_1,P_2))\ [t/\tau_1] \Longleftrightarrow$$

$$(\exists t)(\tau_1-6 \leqslant t \leqslant \tau_1-1 \wedge$$

$$NN(0,0,0) \wedge OO(0,0,0) \wedge len(\chi_1)=0 \wedge len(\chi_2)=0 \wedge 5 \leqslant t \leqslant 13 \vee$$

$$NN(0,0,0) \wedge OO(0,1,0) \wedge len(\chi_1)=0 \wedge len_{comm}(\chi_2)=0 \wedge 11 \leqslant t \leqslant 15$$

$$\wedge T02(10,10) \vee$$

$$(\exists i)(0<i<m \wedge NN(i,0,2i) \wedge OO(i,0,2i) \wedge PT1(6,13)[t/\tau_1])) \Longleftrightarrow$$

$$NN(0,0,0) \wedge OO(0,0,0) \wedge len(\chi_1)=0 \wedge len(\chi_2)=0 \wedge 6 \leqslant \tau_1 \leqslant 19$$

$$NN(0,0,0) \wedge OO(0,1,0) \wedge len(\chi_1)=0 \wedge len_{comm}(\chi_2)=0 \wedge T02(10,10) \wedge$$

$$16 \leqslant \tau_1 \leqslant 21$$

$$(\exists i)(0<i<m \wedge NN(i,0,2i) \wedge OO(i,0,2i) \wedge PT1(7,19))$$

$$\text{match}\ (P_2?NAD(\ ),P_1!NAD()) \Longleftrightarrow$$

$$(\exists t)(\tau_1-6 \leqslant t \leqslant \tau_1-1 \wedge (pre\ (P_2?NAD()) \wedge pre\ (P_1!NAD())) \wedge$$

$$\text{inv}\ (P_1,P_2))[t/\tau_1]) \Longleftrightarrow$$

$$(\exists t)(\tau_1-6 \leqslant t \leqslant \tau_1-1 \wedge$$

$$(\exists i)(1 \leqslant i \leqslant m \wedge NO(i,0,2i-1) \wedge ON(i-1,0,2i-1) \wedge PT1(6,15)[t/\tau_1])) \Longleftrightarrow$$

$$(\exists i)(1 \leqslant i \leqslant m \wedge NO(i,0,2i-1) \wedge ON(i-1,0,2i-1) \wedge PT1(7,21))$$

Należy dodać, że match $(CM,\overline{CM})$ jest równoważny match $(CM,\overline{CM})$ po zastąpieniu τ_1 przez τ_2.

Na podstawie powyższych wzorów można wyliczyć zbiory wszystkich łańcuchów historii:

- $chain_{match(P_2!ODB(x),P_1?ODB(y)),P_2}$

zawiera

h^1_{21} = ( ) dla $NN(0,0,0) \wedge len(\chi_1)=0 \wedge 6 \leqslant \tau_1 \leqslant 15$,

h^2_{21} = ( ), h^2_{22} = $(P_2,10)$ dla $NN(0,0,0) \wedge len(\chi_1)=0 \wedge 16 \leqslant \tau_1 \leqslant 19$,

h^3_{21} = $(P_2,10)$ dla $NN(0,0,0) \wedge len(\chi_1)=0 \wedge 20 \leqslant \tau_1 \leqslant 21$,

h^4_{21} - historię o długości $len_{comm}(h^4_{21})=2i$ $(0<i<m)$

dla $NN(i,0,2i) \wedge PT1(7,19)$

- $chain_{match(P_2?NAD(\ ),P_1!NAD()),P_2}$

zawiera

h^1_{21} - historię o długości $len_{comm}(h^1_{21})=2i-1$ $(1 \leqslant i \leqslant m)$

dla $NO(i,0,2i-1) \wedge PT1(7,21)$

- $chain_{match(P_1?ODB(y),P_2!ODB(x)),P_1}$

zawiera

$$h^1_{11} = (\ ) \quad \text{dla} \quad OO(0,0,0) \wedge len(\chi_2)=0 \wedge 6 \leqslant \tau_2 \leqslant 19 \vee$$
$$OO(0,0,0) \wedge len_{comm}(\chi_2)=0 \wedge$$
$$TO2(10,10) \wedge 16 \leqslant \tau_2 \leqslant 21$$

h^2_{11} - historię o długości $len_{comm}(h^2_{11})=2i \quad (0<i<m)$
dla $OO(i,0,2i) \wedge PT2(7,19)$

- $chain_{match}(P_1!NAD(\ ), P_2?NAD()),P_1$

zawiera

h^1_{11} - historię o długości $len_{comm}(h^1_{11})=2i-1 \quad (1 \leqslant i \leqslant m)$
dla $ON(i-1,0,2i-1) \wedge PT2(7,21)$

Łatwo można sprawdzić, że wymienione historie nie kojarzą się końcami z odpowiadającymi im historiami w procesach partnerskich. Na przykład dla historii generowanych przez

$$chain_{match}(P_2!ODB(x),P_1?ODB(y)),P_2$$

historia h^1_{21} nie kojarzy się końcami z $\chi_1 = (\ )$, h^2_{22} z $\chi_1 = (\ )$, h^4_{21} o długości $len_{comm}(h^4_{21}) = 2i \quad (0<i<m)$ z χ_1 o takiej samej długości itd.

Zatem wynika stąd wniosek, że asercję nomatchtail, stanowiącą element asercji tout (TH), można w dalszych obliczeniach pomijać. Obliczenia dla kolejnych instrukcji TH1, ..., TH4 przedstawiają się następująco:

$$bef\ (P_2!ODB(x),P_1?ODB(y)) \Longleftrightarrow$$
$$(\exists t>0)(\forall t')(\tau_1-11-t \leqslant t' \leqslant \tau_1-6-t \Longrightarrow pre\ (P_1?ODB(y))[t'/\tau_2] \wedge$$
$$offtail\ (\chi_2,t')$$

a stąd wynika:

$$bef\ (P_2!ODB(x),P_1?ODB(y))\ [h^1_{21}/\chi_2] \Longleftrightarrow$$
$$(\exists t>0)(\forall t')(\tau_1-11-t \leqslant t' \leqslant \tau_1-6-t \Longrightarrow$$
$$OO(0,0,0) \wedge 4 \leqslant t' \leqslant 13) \Longleftrightarrow$$
$$OO(0,0,0) \wedge 16 \leqslant \tau_1.$$

$$bef\ (P_2!ODB(x),P_1?ODB(y))\ [h^2_{21}/\chi_2] \Longleftrightarrow$$
$$OO(0,0,0) \wedge 16 \leqslant \tau_1,$$

$$bef\ (P_2!ODB(x),P_1?ODB(y))\ [h^2_{22}/\chi_2] \Longleftrightarrow$$
$$(\exists t>0)(\forall t')(\tau_1-11-t \leqslant t' \leqslant \tau_1-6-t \Longrightarrow$$

$$OO(0,1,0) \wedge 11 \leqslant t' \leqslant 17) \Longleftrightarrow$$

$$OO(0,1,0) \wedge 23 \leqslant \tau_1,$$

$$\text{bef } (P_2!ODB(x), P_1?ODB(y))\ [h^3_{21}/\chi_2] \Longleftrightarrow$$

$$OO(0,1,0) \wedge 23 \leqslant \tau_1,$$

$$\text{bef } (P_2!ODB(x), P_1?ODB(y))\ [h^4_{21}/\chi_2] \Longleftrightarrow$$

$$(\exists t > 0)(\forall t')(\tau_1 - 11 - t \leqslant t' \leqslant \tau_1 - 6 - t \Longrightarrow$$

$$OO(i,0,2i) \wedge \text{last } (h^4_{21}).time + 5 \leqslant t' \leqslant$$

$$\text{last } (h^4_{21}).time + 13 \Longleftrightarrow$$

$$OO(i,0,2i) \wedge \text{last } (h^4_{21}).time + 17 \leqslant \tau_1$$

dla h^4_{21} o długości $len_{comm}(h^4_{21}) = 2i$ $(0 < i < m)$.

Podobnie oblicza się

$$\text{aft } (P_2!ODB(x), P_1?ODB(y)) \Longleftrightarrow$$

$$(\exists t \geqslant 0)(\forall t')(\tau_1 + t \leqslant t' \leqslant \tau_1 + 5 + t \Longrightarrow \text{pre } (P_1?ODB(y))\ [t'/\tau_2])$$

a stąd

$$\text{aft } (P_2!ODB(x), P_1?ODB(y))\ [h^1_{21}/\chi_2] \Longleftrightarrow$$

$$(\exists t \geqslant 0)(\forall t')(\tau_1 + t \leqslant t' \leqslant \tau_1 + 5 + t \Longrightarrow$$

$$OO(0,0,0) \wedge 4 \leqslant t' \leqslant 13 \Longleftrightarrow$$

$$OO(0,0,0) \wedge \tau_1 \leqslant 8,$$

$$\text{aft } (P_2!ODB(x), P_1?ODB(y))\ [h^2_{21}/\chi_2] \Longleftrightarrow$$

$$OO(0,0,0) \wedge \tau_1 \leqslant 8,$$

$$\text{aft } (P_2!ODB(x), P_1?ODB(y))\ [h^2_{22}/\chi_2] \Longleftrightarrow$$

$$OO(0,1,0) \wedge \tau_1 \leqslant 12,$$

$$\text{aft } (P_2!ODB(x), P_1?ODB(y))\ [h^4_{21}/\chi_2] \Longleftrightarrow$$

$$OO(i,0,2i) \wedge \tau_1 \leqslant \text{last } (h^4_{21}).time + 8$$

Ponieważ warunki 1, 2 definicji 7.4 nie zachodzą, zatem wynika stąd postać asercji

$$\text{tout } (TH1) \Longleftrightarrow$$

$$(\exists ch)(chain_{match}(P_1!ODB(x), P_2(ODB(y)), P_2^{(ch)} \wedge$$

$$(\forall h \in ch)(ba(P_2!ODB(x), P_1?ODB(y))[h/\chi_2])) \Longleftrightarrow$$

$$NN(0,0,0) \wedge len(\chi_1) = 0 \wedge 6 \leqslant \tau_1 \leqslant 15 \wedge OO(0,0,0) \wedge$$

$$(\tau_1 \geqslant 16 \vee \tau_1 \leqslant 8) \vee$$

$$NN(0,0,0) \wedge len(\chi_1)=0 \wedge 16 \leqslant \tau_1 \leqslant 19 \wedge 00(0,0,0) \wedge (\tau_1 \geqslant 16 \vee \tau_1 \leqslant 8) \wedge$$
$$00(0,1,0) \wedge (\tau_1 \geqslant 23 \vee \tau_1 \leqslant 12) \vee$$
$$NN(0,0,0) \wedge len(\chi_1)=0 \wedge 20 \leqslant \tau_1 \leqslant 21 \wedge 00(0,1,0) \wedge (\tau_1 \geqslant 23 \vee \tau_1 \leqslant 12) \vee$$
$$(\exists i)(0<i<m \wedge NN(i,0,2i) \wedge PT1(7,19) \wedge$$
$$00(i,0,2i) \wedge (\tau_1 \geqslant last(h^4_{21}).time+17 \vee \tau_1 \leqslant last(h^4_{21}).time+8))$$
$$\Longleftrightarrow$$
$$NN(0,0,0) \wedge len(\chi_1)=0 \wedge 00(0,0,0) \wedge 6 \leqslant \tau_1 \leqslant 7 \vee$$
$$(\exists i)(0<i<m \wedge NN(i,0,2i) \wedge 00(i,0,2i) \wedge$$
$$(PT1(7,7) \vee PT1(17,19)))$$

Zatem

$$pre\ (\underline{nosyn}1) \wedge tout\ (TH1) \Longleftrightarrow$$
$$NN(0,0,0) \wedge 11 \leqslant \tau_1 \leqslant 16 \wedge len(\chi_1)=0 \wedge 00(0,0,0) \wedge 6 \leqslant \tau_1 \leqslant 7 \vee$$
$$(\exists i)(0<i<m \wedge NN(i,0,2i) \wedge 00(i,0,2i) \wedge PT1(12,16) \wedge$$
$$(PT1(7,7) \vee PT1(17,19)) \Longleftrightarrow false$$

co dowodzi warunku przeterminowania dla nosyn1.

Dla instrukcji TH2 oblicza się:

$$bef\ (P_2?NAD(\ ),P_1!NAD()) \Longleftrightarrow$$
$$(\exists t>0)(\forall t')(\tau_1-11-t \leqslant t' \leqslant \tau_1-6-t \Longrightarrow pre\ (P_1!NAD())[t'/\tau_2] \wedge$$
$$offtail\ (\chi_2,t'))$$
$$bef\ (P_2?NAD(\ ),P_1!NAD())\ [h^1_{21}/\chi_2] \Longleftrightarrow$$
$$ON(i,0,2i) \wedge last(h^1_{21}).time+18 \leqslant \tau_1$$

dla h^1_{21} o długości $len_{comm}(h^1_{21})=2i-1 \quad (1 \leqslant i \leqslant m)$,

$$aft\ (P_2?NAD(\ ),P_1!NAD()) \Longleftrightarrow$$
$$(\exists t \geqslant 0)(\forall t')(\tau_1+t \leqslant t' \leqslant \tau_1+5+t \Longrightarrow pre\ (P_1!NAD())[t/\tau_2]),$$
$$aft\ (P_2?NAD(\ ),P_1!NAD())\ [h^1_{21}/\chi_2] \Longleftrightarrow$$
$$ON(i-1,0,2i-1) \wedge last(h^1_{21}).time+10 \geqslant \tau_1,$$

dla h^1_{21} o długości $len_{comm}(h^1_{21})=2i-1 \quad (1 \leqslant i \leqslant m)$.

Ponieważ nie zachodzą warunki 1, 2 definicji 7.4, więc postać asercji

$$tout\ (TH2) \Longleftrightarrow$$
$$(\exists i)(1 \leqslant i \leqslant m \wedge NO(i,0,2i-1) \wedge PT1(7,21) \wedge$$
$$ON(i-1,0,2i-1) \wedge (last(h^1_{21}).time+18 \leqslant \tau_1 \vee$$
$$last(h^1_{21}).time+10 \geqslant \tau_1)).$$

Zatem

$$\text{pre } (\underline{\text{nosyn}}2) \wedge \text{tout } (TH2) \Longleftrightarrow$$

$$(\exists i)(1 \leqslant i \leqslant m \wedge NO(i,0,2i-1) \wedge ON(i-1,0,2i-1) \wedge PT1(12,16) \wedge$$

$$(PT1(18,21) \vee PT1(7,10))) \Longleftrightarrow \text{false}$$

co również dowodzi warunku przeterminowania dla $\underline{\text{nosyn}}2$.

Jeśli chodzi o instrukcję TH3, należy zauważyć, że warunek 2 definicji 7.4 zachodzi dla stanów spełniających asercję

$$(\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(11+7j,14+8j)).$$

Dla pozostałych stanów oblicza się:

$$\text{bef } (P_1?ODB(y),P_2!ODB(x)) \Longleftrightarrow$$

$$(\exists t > 0)(\forall t')(\tau_2 - 11 - t \leqslant t' \leqslant \tau_2 - 6 - t \Longrightarrow \text{pre } (P_2!ODB(x))[t/\tau_1] \wedge$$

$$\text{offtail } (\chi_2,t))$$

$$\text{bef } (P_1?ODB(y),P_2!ODB(x))[h^1_{11}/\chi_1] \Longleftrightarrow$$

$$NN(0,0,0) \wedge \tau_2 \geqslant 22,$$

$$\text{bef } (P_1?ODB(y),P_2!ODB(x))[h^2_{11}/\chi_1] \Longleftrightarrow$$

$$NN(i,0,2i) \wedge \tau_2 \geqslant \text{last}(h^2_{11}).\text{time}+18$$

dla h^2_{11} o długości $\text{len}_{comm}(h^2_{11})=2i \quad (0<i<m)$,

$$\text{aft } (P_1?ODB(y),P_2!ODB(x)) \Longleftrightarrow$$

$$(\exists t \geqslant 0)(\forall t')(\tau_2 + t \leqslant t' \leqslant \tau_2 + 5 + t \Longrightarrow \text{pre } (P_2!ODB(x))[t/\tau_1],$$

$$\text{aft } (P_1?ODB(y),P_2!ODB(x))\ [h^1_{11}/\chi_1] \Longleftrightarrow$$

$$NN(0,0,0) \wedge \tau_2 \leqslant 10,$$

$$\text{aft } (P_1?ODB(y),P_2!ODB(x))[h^2_{11}/\chi_1] \Longleftrightarrow$$

$$NN(i,0,2i) \wedge \tau_2 \leqslant \text{last}(h^2_{11}).\text{time}+10$$

dla h^2_{11} o długości $\text{len}_{comm}(h^2_{11})=2i \quad (0<i<m)$.

Stąd postać asercji

$$\text{tout } (TH3) \Longleftrightarrow (\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(11+7j,14+8j)) \vee$$

$$(OO(0,0,0) \wedge \text{len}(\chi_2)=0 \wedge 6 \leqslant \tau_2 \leqslant 19 \vee$$

$$OO(0,1,0) \wedge \text{len}_{comm}(\chi_2)=0 \wedge TO2(10,10) \wedge 16 \leqslant \tau_2 \leqslant 21) \wedge$$

$$NN(0,0,0) \wedge (\tau_2 \geqslant 22 \vee \tau_2 \leqslant 10) \vee$$

$$(\exists i)(0 < i < m \wedge OO(i,0,2i) \wedge PT2(7,19) \wedge NN(i,0,2i) \wedge$$

$$(\tau_2 \geqslant \text{last}(h^2_{11}).\text{time}+18 \vee$$

$$\tau_2 \leqslant \text{last}(h^2_{11}).\text{time}+10))) \Longleftrightarrow$$

$$(\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(11+7j,14+8j)) \vee$$

$$(OO(0,0,0) \wedge len(\chi_2)=0 \wedge 6 \leqslant \tau_2 \leqslant 10 \wedge NN(0,0,0) \vee$$

$$(\exists i)(0<i<m \wedge OO(i,0,2i) \wedge NN(i,0,2i) \wedge$$
$$(PT2(18,19) \vee PT2(7,10))$$

Zatem

$$pre\ (\underline{nosyn3}) \wedge tout\ (TH3) \Longleftrightarrow$$
$$(\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(11+7j,14+8j)) \vee$$
$$OO(0,0,0) \wedge NN(0,0,0) \wedge len(\chi_2)=0 \wedge 6 \leqslant \tau_2 < 10 \vee$$
$$(\exists i)(0<i<m \wedge OO(i,0,2i) \wedge NN(i,0,2i) \wedge$$
$$(PT2(7,10) \vee PT2(18,19))) \wedge$$
$$(OO(0,0,0) \wedge 10 \leqslant \tau_2 \leqslant 14 \wedge len(\chi_2)=0 \vee$$
$$OO(0,1,0) \wedge 17 \leqslant \tau_2 \leqslant 18 \wedge T02(10,10) \wedge len_{comm}(\chi_2)=0 \vee$$
$$(\exists i)(0<i<m \wedge OO(i,0,2i) \wedge PT2(11,14)) \vee$$
$$(\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(11+7j,14+8j)))$$

Biorąc pod uwagę, że

$$post\ (\underline{nosyn3})[\chi_2 \cap \lambda_{ns}/\chi_2] \Longleftrightarrow$$
$$OO(0,0,0) \wedge \tau_2=10 \wedge \lambda_{ns}.time=10 \wedge len_{comm}(\chi_2)=0 \vee$$
$$(\exists j)(0 \leqslant j < 3 \wedge OO(m,j,2m) \wedge PT2(11+7j,14+8j))$$

widać, że zachodzi implikacja

$$pre\ (\underline{nosyn3}) \wedge tout\ (TH3) \Longrightarrow post\ (\underline{nosyn3})[\chi_2\ \lambda_{ns}/\chi_2]$$

gdzie: $\lambda_{ns} = (P_2, \tau_2)$.

Dla instrukcji TH4 oblicza się

$$bef\ (P_1!NAD(\ ), P_2?NAD()) \Longleftrightarrow$$
$$(\exists t>0)(\forall t')(\tau_2-11-t \leqslant t' \leqslant \tau_2-6-t \Longrightarrow$$
$$pre(P_2?NAD())[t'/\tau_1] \wedge offtail\ (\chi_2, t'))$$

$$bef\ (P_1!NAD(\ ), P_2?NAD())[h^1_{11}/\chi_1] \Longleftrightarrow$$
$$NO(i,0,2i-1) \wedge \tau_2 \geqslant last(h^1_{11}).time+18$$

dla h^1_{11} o długości $len_{comm}(h^1_{11})=2i-1 \quad (1 \leqslant i \leqslant m)$.

$$aft(P_1!NAD(\ ), P_2?NAD()) \Longleftrightarrow$$
$$(\exists t \geqslant 0)(\forall t')(\tau_2+t \leqslant t' \leqslant \tau_2+5+t \Longrightarrow pre\ (P_2?NAD())[t'/\tau_1])$$

$$aft\ (P_1!NAD(), P_2?NAD())[h^1_{11}/\chi_1] \Longleftrightarrow$$
$$NO(i,0,2i-1) \wedge \tau_2 \leqslant last(h^1_{11}).time+10$$

dla h^1_{11} o długości $len_{comm}(h^1_{11})=2i-1 \quad (1 \leqslant i \leqslant m)$.

Ponieważ nie zachodzą warunki 1, 2 definicji 7.4, stąd postać asercji

$$tout\ (TH4) \Longleftrightarrow$$
$$(\exists\, 1 \leqslant i \leqslant m)(ON(i-1,0,2i-1) \wedge PT2(7,21) \wedge NO(i,0,2i-1) \wedge$$

$$(\tau_2 \geqslant last(h_{11}^1).time+18 \vee$$

$$\tau_2 \leqslant last(h_{11}^1).time+10)) \Longleftrightarrow$$

$$(\exists i)(1 \leqslant i \leqslant m \wedge ON(i-1,0,2i-1) \wedge NO(i,0,2i-1) \wedge$$
$$(PT2(7,10) \vee PT2(18,21))$$

Zatem

$$pre\ (\underline{nosyn}4) \wedge tout\ (TH4) \Longleftrightarrow false$$

co dowodzi zachodzenia warunku przeterminowania dla TH4, a tym samym kończy dowód warunków zgodności.

8. ZAKOŃCZENIE

Zasadnicze wyniki pracy sprowadzają się do przedstawienia metody opisu semantyki programów współbieżnych wykonywanych w środowisku czasu rzeczywistego oraz metody dowodzenia własności dla takich programów. Jak wspomniano we wstępie, oba te problemy nie były dotąd przedmiotem szczegółowych badań, a nawet nie było ogólnych ich sformułowań. Rezultatem pracy jest więc oryginalne sformułowanie przedmiotu badań. Osiągnięte wyniki wymagają skomentowania a ponadto należy wskazać na obszary wymagające dalszych badań.

1. W pracy został zdefiniowany prosty język czasu rzeczywistego RTCSP bazujący na języku CSP Hoare'a. Najistotniejszym elementem języka jest czasowo uwarunkowana instrukcja komunikacji. Poza tym, że tego rodzaju konstrukcja okazała się bardzo dogodna do rozwiązywania wielu problemów czasu rzeczywistego np. [102], to równie istotną konsekwencją jej użycia jest brak blokad podczas wykonywania programu. Patrząc z punktu widzenia potrzeb praktycznych, wydaje się, że warto w dalszych badaniach uwzględnić również inną czasowo uwarunkowaną instrukcję komunikacji

$$\underline{till}\ v\ \underline{try}\ \Box_{i=1..n}\ b_i;CM_i \longrightarrow AC_i\ \underline{later}\ AC\ \underline{end},$$

a więc o postaci podobnej do instrukcji TH, natomiast różniącej się od niej tym, że wartość v oznacza pewien konkretny moment czasu (a nie długość pewnego odcinka czasu). Przed upływem tego momentu należy oczekiwać na zajście jednej z instrukcji wejścia/wyjścia CM_i (i=1..n), a po jego upływie - realizować instrukcję AC. Zbadanie tej instrukcji i określenie semantyki oraz podanie odpowiedniej reguły o systemie dowodzenia nie stanowi istotnego problemu; nie poruszano go w pracy, aby nie zwiększać jej objętości. Trudniejszym natomiast problemem byłoby uwzględnienie możliwości zagnieżdżania procesów - problem ten pozostaje otwarty do dalszych badań.

2. Środowisko wykonawcze, wprowadzone oryginalnie w pracy, charakteryzuje ograniczony niedeterminizm (wynikający z przyjęcia założenia, że czas realizacji elementarnych kroków obliczeniowych jest skończony). Wydaje się, że charakterystyka ta jest całkowicie wystarczającą do opisu środowiska wykonawczego działającego bezawaryjnie i nie ma potrzeby jej rozszerzania. Sensowne natomiast byłoby rozważenie, oprócz przyjętego w pracy modelu czasu absolutnego, modeli ewentystycznych. Należy natomiast podkreślić, że przedstawiony aparat bez żadnych kłopotów daje się zastosować, gdy chód lokalnych zegarów w procesach składowych programów jest różny. Przedstawienie semantyki języka programowania z takimi zegarami wymaga w zasadzie tylko modyfikacji definicji 3.5-3.11 i drobnych modyfikacji definicji relacji zmiany konfiguracji. Z wprowadzaniem zegarów lokalnych o różnym chodzie pojawiają się jednak nowe problemy: mechanizmy i metody synchronizacji zegarów.

3. W opisie semantyki języka czasu rzeczywistego wykorzystano i odpowiednio rozszerzono podejście operacyjne z metodą indukcji strukturalnej Plotkina. Możliwość pełnego ujęcia semantyki tak złożonych obiektów, jakimi są programy czasu rzeczywistego, wskazuje na niezwykle dużą siłę ekspresji tego podejścia. Dodatkową zaletą podejścia Plotkina jest to, że daje się równie dogodnie stosować do wyrażania semantyki języków nieimperatywnych.

4. Praca jest pierwszą publikacją, w której formułuje się system dowodzenia własności programów współbieżnych czasu rzeczywistego. Wykorzystano tu i odpowiednio przystosowano klasyczną logikę Hoare'a. Uzyskany efekt można oceniać dwojako.

Z jednego punktu widzenia należy oceniać efektywność stosowania opracowanego systemu dowodzenia. Przeanalizowane proste przykłady wskazują na niezmierną złożoność zagadnienia - bogactwo sytuacji, które należy brać pod uwagę przy weryfikacji programu. Stąd wynika, że niezależnie od tego jakie podejście byłoby stosowane, to przy założeniu, że nie wprowadza się istotnych uproszczeń, cała złożoność zagadnienia przenosi się do weryfikacji. Wypływa stąd wniosek, że praktyczne stosowanie proponowanego podejścia może pociągać konieczność wprowadzania uproszczeń. Możliwość taka istnieje dzięki wprowadzeniu pojęcia schematu systemu dowodzenia, co pozwala na wprowadzenie uproszczeń w miejscu najbardziej istotnym - tzn. w postaci warunków zgodności. Druga istotna możliwość uproszczeń, nie poruszona w pracy i wymagająca dalszych rozwinięć, tkwi w zastępowaniu środowisk: sensowne byłoby niekiedy rozpatrywanie własności programów w środowiskach zastępczych. Sens tego zastąpienia sprowadzałby się do zbadania, czy jeżeli program ma pewne udowodnione własności w środowisku zastępczym, to ma je także w środowisku oryginalnym [104].

Z drugiego punktu widzenia można pytać, czy istnieją potencjalnie inne podejścia do rozwiązywania problemu weryfikacji programów współbieżnych czasu rzeczywistego. Pytanie jest sensowne m.in. dlatego, że w przedstawionej pracy uzyskano niejako kres tego co można uzyskać bazując na logice Hoare'a. Nie ma podstaw do oczekiwań, iż zachowując podejście Hoare'a, da się otrzymać obliczeniowo prostsze systemy weryfikacji. Dlatego nasuwa się kwestia wykorzystania innych podejść. Aktualnie wśród nich można wymienić tylko jednego potencjalnego kandydata - jest nim logika temporalna. Należy jednak zwrócić uwagę na dwie okoliczności. Po pierwsze: dotąd słabo jest jeszcze rozpoznany związek pomiędzy formułami logiki temporalnej a semantyką języków programowania. Z dotychczasowych prac nasuwa się jednak wniosek, że związki takie będą łatwiejsze do przedstawienia dla języków nieimperatywnych, np. CCS, LOTOS niż dla języków imperatywnych, a więc takich jak RTCSP. Po drugie: dotychczasowe prace w zakresie logiki temporalnej nie uwzględniają czasu rzeczywistego, tak jak robi się to w niniejszej pracy.

A zatem przedstawione w pracy podejście pozostaje jedynym, które - pomijając problem złożoności obliczeniowej - daje rozwiązanie zagadnienia dowodzenia własności programów współbieżnych w czasie rzeczywistym, natomiast dopiero przyszłość może rozstrzygnąć o przewadze innych podejść.

ZAŁĄCZNIK - DOWODY LEMATÓW

Dowód lematu 3.1

Dowód prowadzi się metodą indukcji strukturalnej. Oddzielnego rozważenia wymagają przejścia z każdego spośród wymienionych podzbiorów konfiguracji. Ze względu na podobieństwo prowadzenia analizy dowód ogranicza się do pokazania prawdziwości lematu dla przejścia z $\Sigma_{Eprog \setminus (Eproc \cup Prog)}$. Oznacza to, że należy pokazać prawdziwość tez:

Jeżeli $\sigma = \langle \rho, s, c \rangle$, $\rho \in Eprog \setminus (Eproc \cup Prog)$ i $\sigma \xrightarrow{\lambda} \sigma'$, gdzie $\sigma' = \langle \rho', s', c' \rangle$ i $\lambda \in \Lambda$, to:

1) jeżeli $\lambda = \varepsilon$, to $\rho' \in Eprog \setminus (Eproc \cup Prog) \cup Prog \setminus Proc$,
2) jeżeli $\lambda = \lambda_s \| \bar{\lambda}_s$, to $\rho' \in Eprog \setminus (Eproc \cup Prog)$,
3) jeżeli $\lambda = \lambda_c \| \bar{\lambda}_c$, to $\rho' \in Eprog \setminus (Eproc \cup Prog) \cup Prog \setminus Proc$,
4) jeżeli $\lambda = \lambda_{ns}$, to $\rho' \in Eprog \setminus (Eproc \cup Prog) \cup Prog \setminus Proc$.

Fakt, że $\rho \in Eprog \setminus (Eproc \cup Prog)$ oznacza, że ρ składa się co najmniej z dwóch procesów, zapiszmy $\rho = \|_{k=1..n} \rho_k$, $n \geqslant 2$ oraz przynajmniej jeden proces $\rho_i \in Eproc$, i ma postać

$$P_i : PD_i ; EC_i$$

gdzie: $EC_i ::= GC \mid GC;AC \mid SY \mid SY;AC$.

Dla tych procesów składowych ρ_j, które są elementami zbioru Proc możliwe są tylko takie przejścia, które są pokazane przy wierzchołku Σ_{Proc} na diagramie. Wszystkim tym przejściom towarzyszy oddziaływanie $\lambda = \varepsilon$. Jeśli nie następuje przejście do Σ_{abort}, to wszystkie pozostałe przejścia zachowują potrzebne własności.

Istotnie, rozważając wszystkie te przejścia, otrzymuje się:

a) Jeżeli

$$\langle \rho_j, s, c \rangle \xrightarrow{\varepsilon} \langle s', c' \rangle,$$

to

$$\langle \rho, s, c \rangle \xrightarrow{\varepsilon} \langle \rho', s', c' \rangle$$

gdzie: $\rho' = \|_{\substack{k=1..n \\ k \neq j}} \rho_k$. Ponieważ $n \geqslant 2$ oraz istnieje co najmniej jeden $\rho_j \in Eproc \setminus Proc$, zatem gdy $n = 2$, wtedy $\rho' \in Eproc \setminus Proc$, a gdy $n > 2$, wtedy $\rho' \in Eprog \setminus (Eproc \cup Prog)$.

b) Jeżeli

$$\langle \rho_j, s, c \rangle \xrightarrow{\varepsilon} \langle \rho', s', c' \rangle,$$

gdzie na mocy hipotezy indukcyjnej $\rho'_j \in Proc$, to

$$\rho' = \|_{k=1..n} \rho'_k, \quad \rho_k = \rho'_k \text{ dla } k \neq j,$$

należy do $Eprog \setminus (Eproc \cup Prog)$, ponieważ $\rho_j \in Proc$.

c) Jeżeli

$$\langle \rho_j, s, c \rangle \xrightarrow{\varepsilon} \langle \rho'_j, s', c' \rangle,$$

gdzie na mocy hipotezy indukcyjnej $\rho'_j \in Eproc \setminus Proc$, to oczywiście

$$\rho' = \|_{k=1..n} \rho'_k,$$

gdzie $\rho'_k = \rho_k$ dla $k \neq j$, również należy do $Eprog \setminus (Eproc \cup Prog)$.

Dla tych procesów składowych ρ_i, które są elementami zbioru $Eproc$ należy rozpatrzeć następujące przypadki:

Jeżeli $EC_i = GC$, to możliwe są tylko przejścia pod wpływem oddziaływania $\lambda = \varepsilon$. Zbiór tych przejść pokazują łuki wychodzące z wierzchołka Σ_{Gcm} na diagramie. Stwarzają one konieczność analizy dalszych pięciu podprzypadków przejść, którym nie towarzyszy zerwanie obliczeń.

d) Jeżeli

$$\langle GC, s, c \rangle \xrightarrow{\varepsilon} \langle s', c' \rangle,$$

to

$$\langle \rho, s, c \rangle \xrightarrow{\varepsilon} \langle \rho', s', c' \rangle,$$

gdzie: $\rho' = \|_{\substack{k=1..n \\ k \neq i}} \rho_k$, i jeżeli ρ_i był jedynym procesem należącym do zbioru $Eproc$, to $\rho' \in Prog \setminus Proc$.

e) Jeżeli

$$\langle GC, s, c \rangle \xrightarrow{\varepsilon} \langle AC', s', c' \rangle,$$

to

$$\langle \rho, s, c \rangle \xrightarrow{\varepsilon} \langle \rho', s', c' \rangle,$$

gdzie: $\rho' = \|_{k=1..n} \rho'_k$, $\rho'_k = \rho_k$ dla $k \neq i$ oraz $\rho_i \in Proc$ na mocy hipotezy indukcyjnej, wtedy $\rho' \in Prog \setminus Proc$.

f) Jeżeli

$$\langle GC, s, c \rangle \xrightarrow{\varepsilon} \langle GC', s', c' \rangle,$$

to oczywiście z hipotezy indukcyjnej $\rho'_i \in Eproc$ i $\rho \in Eprog \setminus (Eproc \cup Prog)$.

g) Jeżeli

$$\langle GC, s, c \rangle \xrightarrow{\varepsilon} \langle SY', s', c' \rangle,$$

to $\rho'_i \in Eproc$ i $\rho \in Eprog \setminus (Eproc \cup Prog)$.

h) Jeżeli

$$\langle GC,s,c\rangle \xrightarrow{\varepsilon} \langle SY',AC',s',c'\rangle,$$

to przypadek jest analogiczny do g).

Jeżeli $EC_i = GC;AC$, to analiza przypadków jest prawie taka sama jak wyżej dla $EG_i = GC$.

Jeżeli $EC_i = SY$, to potrzebna jest analiza następujących przypadków:

i) Istnieje tylko jeden proces $\rho_j \in Eproc$ lub istnieje więcej procesów ze zbioru Eproc, lecz możliwe jest tylko przejście pod wpływem oddziaływania λ_{ns}, wtedy:

$$\langle SY,s,c\rangle \xrightarrow{\lambda_{ns}} \langle AC',s',c'\rangle$$

co jest równoważne przypadkowi e).

j) Istnieje więcej niż jeden proces ze zbioru Eproc oraz pomiędzy procesem ρ_j a ρ_k zachodzi możliwość synchronizacji, wtedy

$$\langle SY,s,c\rangle \xrightarrow{\lambda_s} \langle SY',s,c'\rangle \text{ w procesie } \rho_j$$

oraz analogiczne przejście jest możliwe w procesie ρ_k pod wpływem $\bar{\lambda}_s$. Łącznie zachodzi zatem oddziaływanie $\lambda_s \| \bar{\lambda}_s$, przy czym w każdym z procesów składowych $\rho_j', \rho_k' \in Eproc$ na mocy hipotezy indukcyjnej, zatem $\rho \in Eprog \setminus (Eproc \cup Prog)$.

k) Istnieje więcej niż jeden proces ze zbioru Eproc oraz pomiędzy parą takich procesów ρ_j, ρ_k zachodzi skojarzenie dynamiczne. Wtedy w obu procesach są możliwe oddziaływania λ_c oraz $\bar{\lambda}_c$, a łącznie zachodzi oddziaływanie $\lambda_c \| \bar{\lambda}_c$. Zatem

$$\langle SY,s,c\rangle \xrightarrow{\lambda_c} \langle AC',s',c'\rangle,$$

w procesie ρ_j oraz podobnie w procesie ρ_k. Oznacza to, że $\rho_j', \rho_k' \in Proc$, a więc na pewno $\rho \in Prog \cup Proc$.

Jeżeli $EC_i = SY;AC$, to analiza jest taka sama jak dla $EC_i = SY$.

Zatem po zbilansowaniu przypadków a), ..., h) otrzymuje się tezę 1, przypadek i) daje tezę 4, przypadek j) - tezę 2 oraz przypadek k) tezę 3. ■

Dowód lematu 3.2

Dowód prowadzi się metodą indukcji strukturalnej. Spośród przypadków wymagających rozpatrzenia wystarczy ograniczyć się do najbardziej istotnego, gdy $\rho \in Eprog \setminus (Eproc \cup Prog)$, bowiem w pozostałych przypadkach teza wynika bezpośrednio z definicji relacji zmiany konfiguracji.

Fakt, że $\rho \in Eprog \setminus (Eproc \cup Prog)$ oznacza, że ρ składa się co najmniej z dwóch procesów oraz przynajmniej jeden proces należy do zbioru Eproc. Zatem $n \geqslant 2$. Rozpatrzenia wymagają następujące przypadki:

1. Istnieje $\rho_i \in Proc$, wtedy z hipotezy indukcyjnej wynika, że istnieją ρ_i', s', c' takie, że

$$\langle \rho_i, s, c \rangle \xrightarrow{\varepsilon} \langle \rho_i', s', c' \rangle \mid \langle s', c' \rangle \mid \underline{abortion}.$$

Oznacza to również istnienie odpowiedniego przejścia z konfiguracji $\langle \rho, s, c \rangle$.

2. Nie istnieje $\rho_i \in Proc$, czyli $\rho_i \in Eproc \setminus Proc$, i istnieje wśród nich ρ_i postaci $P_i : PD_i ; GC$ lub $P_i : PD_i ; GC ; AC$. Zgodnie z założeniem 2, istnieje alternatywa otwarta w GC w stanie s. Zatem istnieje przejście

$$\langle \rho_i, s, c \rangle \xrightarrow{\varepsilon} \langle \rho_i', s', c' \rangle,$$

gdzie: $\rho_i' \in Eproc$, a stąd również odpowiednie przejście z konfiguracji $\langle \rho, s, c \rangle$.

3. Jeżeli $\rho_i \in Eproc \setminus Proc$ oraz nie istnieje ρ_i postaci jak w p. 2, to oznacza, że ρ_i są postaci $P_i : PD_i ; SY_i$ lub $P_i : PD_i ; SY_i ; AC_i$

a) Niech wszystkie SY_i są postaci $BS_i = t_i \ \underline{wt} \ CC_i \ \underline{lt} \ AC_i$

Możliwe są tu dwie sytuacje:

- istnieje para ρ_i, ρ_j taka, że

$$[\![SYNCH\ (\rho_i, \rho_j)]\!]_\sigma = tt.$$

W tym przypadku istnieją oddziaływania λ_s, $\bar{\lambda}_s$, dla których są możliwe przejścia z $\langle \rho_i, s, c \rangle$ pod wpływem λ_s oraz z $\langle \rho_j, s, c \rangle$ pod wpływem $\bar{\lambda}_s$, a tym samym istnieje odpowiednie przejście z konfiguracji $\langle \rho, s, c \rangle$ pod wpływem $\lambda_s \| \bar{\lambda}_s$.

- nie istnieje para procesów, które synchronizują się ze sobą. Niech ρ_i będzie tym spośród procesów składowych, dla których jest spełniony warunek

$$c(\underline{cl}\ \rho_i) + t_i \leqslant c(\underline{cl}\ \rho_j) + t_j, \quad \text{dla} \quad j \neq i.$$

Dla tak wybranego procesu zachodzi oczywiście

$$[\![NOSYN\ (\rho_i, \rho_j)]\!]_\sigma = tt \quad \text{dla} \quad j \neq i,$$

a zatem istnieje przejście z konfiguracji $\langle \rho_i, s, c \rangle$ pod wpływem oddziaływania λ_{ns}, a stąd także odpowiednie przejście z konfiguracji $\langle \rho, s, c \rangle$ pod wpływem tego samego oddziaływania.

b) Jeżeli istnieje SY_i postaci AS_i, to z założenia 1 istnieje także ρ_j takie, że

$$[\![MATCH\ (\rho_i, \rho_j)]\!]_\sigma = tt,$$

czyli możliwe są przejścia z $\langle \rho_i, s, c \rangle$ pod wpływem pewnego λ_c oraz z $\langle \rho_j, s, c \rangle$ pod wpływem pewnego $\bar{\lambda}_c$, czyli również przejście z $\langle \rho, s, c \rangle$ pod wpływem oddziaływania $\lambda_c \| \bar{\lambda}_c$. ∎

Dowód lematu 3.3

Dowód własności 1 jest następujący. Niech ρ' będzie takie jak określa własność 1. Z definicji relacji zmiany konfiguracji wynika, że przejście do konfiguracji, w której składowa ρ'_i przyjęła zakładaną postać, mogło nastąpić tylko w wyniku przejścia od pewnej konfiguracji $\sigma'' = \langle \rho'', s'', c'' \rangle$, w której składowa ρ''_i, przy założeniu, że $\rho'' = \|_{k=1..n} \rho''_k$, ma postać $P_i : PD_i ; BS_i$ lub $P_i : PD_i ; BS_i ; AC''_i$. Także z definicji relacji zmiany konfiguracji wynika, że przejście takie mogło nastąpić tylko z jednoczesną zmianą pewnej składowej ρ''_j, która ma postać tego samego rodzaju co ρ''_i. Zatem przejście $\sigma \longrightarrow^+ \sigma'$ można zdekomponować na

$$\sigma \longrightarrow^* \sigma'', \quad \sigma'' \xrightarrow{\lambda_s \| \bar{\lambda}_s} \sigma''', \quad \sigma''' \longrightarrow^* \sigma'$$

przy czym

$$\sigma''' = \langle \|_{k=1..n} \rho'''_k, s''', c''' \rangle \text{ jest taka, że}$$

$$s''' = s''$$

$$c'''(\underline{cl}\ \rho'''_i) = c'''(\underline{cl}\ \rho'''_j)$$

a ponadto

$$\rho'''_i = \rho'_i, \qquad \rho'''_j = \rho'_j.$$

Ponieważ zmienna wolna i zegary procesów są rozłączne oraz

$$FV(\rho''_i) \subseteq FV(\rho_i), \qquad FV(\rho''_j) \subseteq FV(\rho'_j),$$

zatem

$$s'' \,|\, FV(\rho'_i) \cup FV(\rho'_j) = s' \,|\, FV(\rho'_i) \cup FV(\rho'_j)$$

a także

$$s'(\underline{cl}\ \rho'_i) = s'(\underline{cl}\ \rho'_j).$$

Ponieważ warunkiem przejścia $\sigma'' \xrightarrow{\lambda_s \| \bar{\lambda}_s} \sigma'''$ była prawdziwość predykatu

$$[\![SYNCH\ (\rho''_i, \rho''_j)]\!]_{\sigma''} \doteq tt$$

stąd z definicji predykatów SYNCH, MATCH wynika, że

$$[\![\text{MATCH } (\rho_i', \rho_j')]\!]_{\sigma'} = tt.$$

Prawdziwość własności 2 dowodzi się podobnie. ∎

Dowód lematu 5.1

Prawdziwość aksjomatów A1, A2 jest oczywista. Niech s będzie takim stanem, że dla dowolnego t takiego, że

$$[\![ASS_e\,[t/dur]]\!]_e = tt$$

prawdziwa jest asercja

$$\vdash_J \beta\,[e/x, \tau+t/\tau]\ (s).$$

Jeśli obliczenie instrukcji podstawienia x:=e kończy się prawidłowo, to początkowy stan s - zgodnie z definicją relacji zmiany konfiguracji - ponieważ $c(\underline{cl}\ x:=e)=\tau$, jest przekształcony w

$$s' = s([\![e]\!]_s/x, [\![t+\tau]\!]_s/\tau)$$

gdzie t również spełnia $[\![ASS_e\,[t/dur]]\!]_s = tt$.

Jeżeli

$$\vdash_J \beta\,[e/x, \tau+t/\tau]\ (s),$$

to oczywiste także

$$\vdash_J \beta(s([\![e]\!]_s/x,\ [\![t+\tau]\!]_s/\tau)$$

dla dowolnego t. Stąd wynika prawdziwość aksjomatu A3:

$$\vdash_J \{(\forall t)(ASS_e[\![t/dur]\!] \Longrightarrow \beta\,[e/x, \tau+t/t]\}\ x:=e\ \{\beta\}$$

Aksjomaty A4, A5 są szczególnymi przypadkami aksjomatu A3.

Niesprzeczność reguł R1, R5, R6 jest oczywista. Dowód niesprzeczności reguły R2 jest szczególnym przypadkiem dowodu dla reguły R4. Stąd pokazuje się tylko niesprzeczność R3, R4.

Niech przy ustalonej interpretacji J, dla instrukcji DO postaci $\underline{do} \bigsqcup_{i=1\ldots n} b_i \longrightarrow AC_i\ \underline{end}$, będą prawdziwe przesłanki reguły R3:

$$\vdash_J \{\alpha \wedge b_i\}\ \underline{test}\ (DO);\ AC_i\ \{\alpha\}\quad (i=1..n),$$

$$\vdash_J \{\alpha \wedge \neg BOOL\ (DO)\}\ \underline{test}\ (DO)\ \{\beta\},$$

czyli

$$Asem_p\,[\![\underline{test}\ (DO); AC_i]\!]\ ([\alpha \wedge b_i]_J) \subseteq [\alpha]_J \quad (i=1..n),$$

$$Asem_p\,[\![\underline{test}\ (DO)]\!]\ ([\alpha \wedge \neg BOOL\ (DO)]_J) \leqslant [\beta]_J.$$

Niech $\bar{s} \in Asem_p[\![DO]\!]([\alpha]_J)$. Wtedy istnieje taki s, że $\bar{s} \in Asem_p[\![DO]\!]\ (s)$. Z definicji $Asem_p$ otrzymuje się

$$\langle DO, s_0, c_0\rangle \rightarrow^* (DO, s_1, c_1\rangle \rightarrow^* \ldots \rightarrow^* \langle DO, s_m, c_m\rangle \rightarrow \langle s_{m+1}, c_{m+1}\rangle,$$

gdzie $\langle s_0,c_0\rangle = \langle s,c\rangle$, $\langle s_{m+1},c_{m+1}\rangle = \langle s,c\rangle$ oraz, dla i=0..n-1, $s_i \in [b_{k_i}]_J$ oraz $s_{i+1} \in Asem_p [\![\underline{test}\ (DO);AC_{k_i}]\!](s_i)$ dla pewnego $k_i \in \{1, \ldots, n\}$, oraz $s_m \notin [BOOL\ (DO)]_J$ oraz $s_m = s_{m+1}, c_{m+1} = c_m(c_m(\underline{cl}\ DO) + t/\underline{cl}\ DO)$ dla pewnego t spełniającego

$$[\![TEST_{DO}[t/dur]]\!]_{s_m} = tt.$$

Zatem $s_0 \in [\alpha]_J$ oraz jeżeli $s_i \in [\alpha]_J$, to

$$s_{i+1} \in Asem_p [\![\underline{test}\ (DO);AC_i]\!]([\alpha \wedge b_i]_J) \subseteq \{\alpha\}_J$$

Stąd dla wszystkich i=0..m $\langle s_i,c_i\rangle \in [\alpha]_J$. Szczególnie $s_m \in [\alpha]_J$, stąd $s_m \in [\alpha \wedge \neg BOOL\ (DO)]_J$. Zatem

$$Asem_p [\![\underline{test}\ DO]\!](s_m) \subseteq [\beta]_J$$

co kończy dowód prawdziwości reguły R3.

Niech teraz, przy ustalonej interpretacji J, będą prawdziwe przesłanki reguły R4 dla instrukcji TH postaci

$$\underline{th}\ t\ \underline{wt}\ \square_{i=1..n}\ b_i;CM_i \longrightarrow AC_i\ \underline{lt}\ AC_0\ \underline{end},\ \text{tzn.}$$

$$\models_J \{\alpha \wedge b_i\}\ \underline{test}\ (TH);\ \underline{wait}\ (t);CM_i;AC_i\ \{\beta\}\quad (i=1..n),$$

$$\models_J \{\alpha \wedge BOOL\ (TH)\}\ \underline{test}\ (TH);\ \underline{later}\ (t);\ \underline{nosyn};\ AC_0\ \{\beta\}$$

Z definicji semantyki $Asem_p$ wynika, że

$$Asem_p [\![TH]\!](s) = \bigcup_{i=1..n} Asem_p [\![\underline{test}\ (TH);\ \underline{wait}\ (t);CM_i;AC_i]\!](s)$$

$$\cup\, Asem_p [\![\underline{test}\ (TH);\ \underline{later}\ (t);\ \underline{nosyn};\ AC_0]\!](s).$$

Niech $s \in [\alpha \wedge b_i]_J$, wtedy z założenia

$$Asem_p [\![\underline{test}\ (TH);\ \underline{wait}\ (t);CM_i;AC_i]\!](s) \subseteq [\beta]_J,$$

$$Asem_p [\![\underline{test}\ (TH);\ \underline{later}\ (t);\ \underline{nosyn};AC_0]\!](s) \subseteq [\beta]_J,$$

czyli dla każdego

$$s_1 \in Asem_p [\![TH]\!](s)$$

zachodzi $\models_J \beta(s_1)$, co oznacza, że reguła R4 jest niesprzeczna.

■

Dowód lematu 5.2

Dowód prowadzi się metodą indukcji strukturalnej. Jeżeli dla każdej instrukcji składowej AC procesu PC istnieją asercje pre (AC), post (AC) takie, że w interpretacji J są prawdziwe formuły 1-10, to istnienie odpowiedniego dowodu w systemie PS_p jest oczywiste - wystar-

czy sprawdzić, że dla danego typu instrukcji AC są prawdziwe przesłanki odpowiadającej jej reguły.

W przypadku odwrotnym, gdy istnieje dowód dla procesu PC w systemie PS_p, dowód wynika bezpośrednio z lematu 5.1. ■

Dowód lematu 5.3

Dowód prowadzi się indukcyjnie ze względu na sumaryczną długość historii lokalnych $hloc_1, \ldots, hloc_n$. Niech k_i będzie długością historii $hloc_i$ $(i=1..n)$ oraz niech $k = k_1 + \ldots + k_n$.

Jeżeli $k = 0$, to

$$\langle \overset{||}{}_{i=1..n} AC_i, s, c\rangle = \langle \overset{||}{}_{i=1..n} \overline{AC}_i, \bar{s}, \bar{c}\rangle$$

i wtedy

$$\overline{AC}_i = \underline{\text{before}}\ (AC_i, AC_i)$$

Wszystkie formuły postaci $\alpha \Longrightarrow \text{pre}\ (AC_i)$ są przesłankami reguły $R7^*$, prawdziwymi w interpretacji J. Stąd także

$$\vdash \text{pre}\ (AC_i)(\bar{s}) \qquad (i=1..n).$$

Niech $k > 0$ oraz, z założenia indukcyjnego, niech tezy lematu będą prawdziwe dla każdego $k' < k$. Należy pokazać, że lemat jest prawdziwy także dla k.

Jeżeli zachodzi

$$\sigma = \langle \overset{||}{}_{i=1..n} AC_i, s, c\rangle \xrightarrow{\overset{||}{}_{i=1..n}\ hloc_i} {}^{*}\langle \overset{||}{}_{i=1..n} \overline{AC}_i, \bar{s}, \bar{c}\rangle = \bar{\sigma}$$

to istnieją takie historie $hloc_i'$, $hloc_i''$, że $hloc_i = hloc_i' \cap hloc_i''$ $(i=1..n)$ oraz długość sumaryczna k'' historii $hloc_1'', \ldots, hloc_n''$ spełnia nierówność $1 \leqslant k'' \leqslant 2$, natomiast sumaryczna długość k' historii $hloc_1', \ldots, hloc_n'$ jest mniejsza od k. Zatem istnieją obliczenia

$$\sigma = \langle \overset{||}{}_{i=1..n} AC_i, s, c\rangle \xrightarrow{\overset{||}{}_{i=1..n}\ hloc_i'} {}^{*}\langle \overset{||}{}_{i=1..n} \overline{\overline{AC}}_i, \bar{\bar{s}}, \bar{\bar{c}}\rangle = \bar{\bar{\sigma}}$$

$$\bar{\bar{\sigma}} = \langle \overset{||}{}_{i=1..n} \overline{\overline{AC}}_i, \bar{\bar{s}}, \bar{\bar{c}}\rangle \xrightarrow{\overset{||}{}_{i=1..n}\ hloc_i''} {}^{*}\langle \overset{||}{}_{i=1..n} \overline{AC}_i, \bar{s}, \bar{c}\rangle = \bar{\sigma}.$$

Obliczenia reprezentowane przez historie $hloc_1'', \ldots, hloc_n''$ mogą odbyć się na dwa sposoby:

- w wyniku autonomicznej realizacji pewnego kroku obliczeniowego w pewnej instrukcji $\overline{\overline{AC}}_i$ (wtedy $k'' = 1$, i $\overline{\overline{AC}}_j = \overline{\overline{AC}}_j$ dla $j \neq i$).

- w wyniku realizacji skojarzonej dynamicznie pary instrukcji komunikacyjnych w pewnych $\overline{\overline{AC}}_i$, $\overline{\overline{AC}}_j$ (wtedy $k'' = 2$ i $\overline{\overline{AC}}_l = AC_l$ dla $l \neq i,j$).

Przypadek $k'' = 1$

Dla tego przypadku rozpatruje się, jako reprezentatywne, dwa podprzypadki:

- realizację pierwszego kroku obliczeniowego w instrukcji iteracji AC' postaci

$$\underline{do} \ \square_{j=1..m} \ b_j \longrightarrow AC'_j \ \underline{end}$$

(oddziaływanie puste) dla pewnego AC_i takiego, że

$$\overline{\overline{AC}}_i = \underline{before}\ (AC', AC_i),$$

- realizację kroku polegającego na upływie odcinka czasu przeterminowania w czasowo uwarunkowanej instrukcji AC postaci

$$\underline{th}\ v_i\ \underline{wt}\ \square_{j=1..m}\ b_{ij};\ CM_{ij} \longrightarrow AC_{ij}\ \underline{lt}\ AC_{iO}\ \underline{end}$$

(oddziaływanie niepuste), dla pewnego AC_i takiego, że

$$\overline{\overline{AC}}_i = \underline{before}\ (\underline{later}\ (v);\ \underline{nosyn},\ AC_i).$$

Podprzypadek 1. Z definicji operacji before wynika, że

$$\overline{AC}_i = \underline{after}\ (\underline{test}\ (AC'),\ AC_i),$$

czyli należy rozpatrzyć dwie możliwości:

$$\overline{AC}_i = \underline{before}\ (AC'_j;\ AC',\ AC_i) \qquad (j=1..m),$$

$$\overline{AC}_i = \underline{after}\ (AC',\ AC_i).$$

Należy zauważyć, że

$$\bar{\bar{s}} = \bar{s} \quad \text{oraz} \quad \bar{c} = \bar{\bar{c}}(\bar{\bar{c}}(\underline{cl}\ AC_i) + t/\underline{cl}\ AC_i)$$

gdzie t jest pewną wartością taką, że

$$[\![TEST_{AC}[t/dur]]\!]_{\bar{\bar{s}}} = tt.$$

Z założenia indukcyjnego

$$\vdash_J pre\ (AC')(\bar{\bar{s}}).$$

Ponieważ w zbiorze przesłanek reguły R7* są prawdziwe w interpretacji J formuły:

$$pre\ (AC') \wedge b_j \Longrightarrow pre\ (\underline{test}\ (AC');AC'_j),$$

$$pre\ (\underline{test}\ (AC');\ AC_j) \Longrightarrow pre\ (\underline{test}\ (AC')),$$

$$post\ (\underline{test}\ (AC')) \Longrightarrow pre\ (AC'_j), \qquad (j=1..m)$$

$$pre\ (\underline{test}\ (AC')) \Longrightarrow (\forall t)(TEST_{AC'}[t/dur] \Longrightarrow$$

$$\text{post } (\underline{\text{test}}\ (AC'))\,[\tau_i+t/\tau_i] \ ,$$

więc z prawdziwości

$$\models_J \text{pre } (AC')(\bar{\bar{s}}) \quad \text{oraz} \quad \models_J [\![\text{TEST}_{AC'}[t/dur]]\!]_{\bar{\bar{s}}}$$

wynika, że

$$\models_J \text{pre } (AC'_j)(\bar{\bar{s}}) \quad \text{dla} \quad j=1..m.$$

Ponieważ w zbiorze przesłanek reguły R7* jest również prawdziwa w interpretacji J przesłanka

$$\text{pre } (AC') \wedge \neg \text{BOOL } (AC') \Longrightarrow$$
$$(\forall t)(\text{TEST}_{AC'}[t/dur] \Longrightarrow \text{post } (AC')[\tau_i+t/\tau_i])$$

zatem również

$$\models_J \text{post } (AC')(\bar{\bar{s}}),$$

co kończy analizę przyjętego podprzypadku.

Podprzypadek 2. Niech

$$\overline{\overline{AC}}_i = \underline{\text{before}}\ (\underline{\text{later}}\ (v);\ \underline{\text{nosyn}},\ AC_i).$$

Zgodnie z definicją relacji zmiany konfiguracji (p. 3.5) wynika, że

$$\overline{AC}_i = \underline{\text{after}}\ (\underline{\text{later}}\ (v);\ \underline{\text{nosyn}},\ AC_i) = \underline{\text{after}}\ (\underline{\text{nosyn}},\ AC_i)$$

czyli, że pojedynczy elementarny krok obliczeń polega na wykonaniu dwóch instrukcji pomocniczych: later (v) oraz nosyn. Przyjmijmy, że po wykonaniu later (v), a przed wykonaniem nosyn, program znajduje się w pewnej pomocniczej konfiguracji

$$\bar{\bar{\sigma}} \Big\langle \prod_{i=1..n} \overline{\overline{AC}}_i,\ \bar{\bar{s}},\ \bar{\bar{c}} \Big\rangle$$

takiej, że

$$\overline{\overline{AC}}_i = \underline{\text{after}}\ (\underline{\text{later}}\ (v),\ AC_i),$$
$$\overline{\overline{AC}}_j = \overline{\overline{AC}}_j \quad \text{dla} \quad j \neq i,$$
$$\bar{\bar{s}} = \bar{\bar{s}}(\bar{\bar{s}}(\tau_i) + v_i + 1/\tau_i),$$
$$\bar{\bar{c}} = \bar{\bar{c}}(\bar{\bar{c}}(\underline{\text{cl}}\ AC_i) + v + 1/\underline{\text{cl}}\ AC_i).$$

Postać $\bar{\bar{s}}$ oraz $\bar{\bar{c}}$ wynika z definicji semantyki instrukcji later (v). Z definicji semantyki instrukcji nosyn wynika natomiast, że

$$\overline{AC}_i = \underline{\text{after}}\ (\underline{\text{nosyn}},\ AC_i),$$
$$\overline{AC}_j = \overline{\overline{AC}}_j \quad \text{dla} \quad j \neq i,$$

$$\bar{\bar{s}} = \bar{\bar{s}}(s(\chi_i) \cap \lambda/\chi_i),$$

$$\bar{\bar{c}} = \bar{\bar{c}},$$

gdzie: $\lambda = (P_i, \bar{\bar{s}}(\tau_i))$.
Z założenia indukcyjnego wynika, że

$$\vDash_J \text{pre } (\underline{\text{later}}\ (v))\ (\bar{\bar{s}}),$$

natomiast należy pokazać, że

$$\vDash \text{post } (\underline{\text{nosyn}})(\bar{\bar{s}}).$$

W zbiorze przesłanek reguły R7* jest prawdziwa w interpretacji J formuła (lemat 5.2 p. 7):

$$\text{pre } (\underline{\text{later}}\ (v)) \Longrightarrow ((v_i + 1 = \text{dur})\ [v_i + 1/\text{dur}] \Longrightarrow \text{post } (\underline{\text{later}}\ (v))\ [\tau_i + v_i + 1/\tau_i],$$

czyli

$$\text{pre } (\underline{\text{later}}\ (v)) \Longrightarrow \text{post } (\underline{\text{later}}\ (v))\ [\tau_i + v + 1/\tau_i].$$

Ponieważ

$$\vDash_J \text{pre } (\underline{\text{later}}\ (v))(\bar{\bar{s}}),$$

zatem

$$\vDash_J \text{post } (\underline{\text{later}}\ (v))\ [\tau_i + v + 1/\tau_i](\bar{\bar{s}}),$$

czyli

$$\vDash_J \text{post } (\underline{\text{later}}\ (v))(\bar{\bar{\bar{s}}}).$$

Stąd oraz z lematu 5.2 p. 6 wynika, że

$$\vDash_J \text{pre } (\underline{\text{nosyn}})(\bar{\bar{\bar{s}}}).$$

W zbiorze przesłanek reguły R7* jest prawdziwa w interpretacji J formuła opisująca warunek przeterminowania dla instrukcji nosyn:

$$\text{pre } (\underline{\text{nosyn}}) \wedge \text{timeout } (AC') \Longrightarrow \text{post } (\underline{\text{nosyn}})[\chi_i \cap \lambda/\chi_i]$$

Ponieważ z definicji asercji timeout AC

$$\vDash_J \text{timeout } (AC')(\bar{\bar{\bar{s}}}),$$

więc wynika stąd, że

$$\vDash_J \text{post } (\underline{\text{nosyn}})[\chi_i \cap \lambda/\chi_i](\bar{\bar{\bar{s}}})$$

i dalej, co należy ostatecznie pokazać, że

$$\vDash_J \text{post } (\underline{\text{nosyn}})(\bar{\bar{s}}).$$

Przypadek k = 2

Przypadek ten zachodzi wówczas, gdy istnieją dwa procesy, dla których

$$\overline{\overline{AC}}_i = \underline{before}\ (CM_{ik}, AC_i),$$
$$\overline{\overline{AC}}_j = \underline{before}\ (CM_{jl}, AC_j),$$

gdzie CM_{ik}, CM_{jl} są parą dynamicznie skojarzonych, w stanie $\bar{\bar{s}}$, instrukcji komunikacyjnych. Oznacza to, że

$$\bar{\bar{c}}(\underline{cl}\ AC_i) = \bar{\bar{c}}(\underline{cl}\ AC_j),$$

a także, że $\chi_{ij} = \bar{\chi}_{ji}$, czyli że jest prawdziwy niezmiennik inv (AC_i, AC_j) w stanie $\bar{\bar{s}}$, a ponadto, że possmatching (AC_i, AC_j) jest także asercją prawdziwą w stanie $\bar{\bar{s}}$. Z założenia indukcyjnego wiadomo, że

$$\vDash_J \text{pre}\ (CM_{ik})(\bar{\bar{s}})$$
$$\vDash_J \text{pre}\ (CM_{jl})(\bar{\bar{s}}),$$

zatem również

$$\vDash_J (\text{pre}\ (CM_{ik}) \wedge \text{pre}\ (CM_{jl}) \wedge \text{inv}\ (AC_i, AC_j) \wedge \text{possmatching}\ (AC_i, AC_j))(\bar{\bar{s}})$$

Z wykonywanego kroku obliczeń wiadomo, że

$$\overline{\overline{AC}}_i = \underline{after}\ (CM_{ik}, AC_i),$$
$$\overline{\overline{AC}}_j = \underline{after}\ (CM_{jl}, AC_j),$$

natomiast stan $\bar{s}$ jest taki, że

$$\bar{s} = \bar{\bar{s}}([\![e]\!]_{\bar{\bar{s}}}/x,\ \bar{\bar{s}}(\chi_i) \cap \lambda/\chi_i,\ \bar{\bar{s}}(\chi_j) \cap \bar{\lambda}/\chi_j,$$
$$\bar{\bar{s}}(\tau_i) + t/\tau_i,\ \bar{\bar{s}}(\tau_j) + t/\tau_j),$$

gdzie e jest wyrażeniem, którego wartość jest wysyłana przez jeden z procesów; x - zmienną, pod którą jest zapamiętywana ta wartość; λ, $\bar{\lambda}$ stanowią parę komplementarnych oddziaływań komunikacyjnych takich, że

$$\lambda.\text{val} = [\![e]\!]_{\bar{\bar{s}}},$$
$$\lambda.\text{time} = \bar{\bar{c}}(\underline{cl}\ AC_i);$$

oraz t jest wartością taką, że

$$[\![\text{SEND}_e\ [t/dur]]\!]_{\bar{\bar{s}}} = tt.$$

Stan czasomierzy $\bar{c}$ jest oczywiście równy

$$\bar{c} = \bar{\bar{c}}(\bar{\bar{c}}(\underline{cl}\ AC_i) + t/\underline{cl}\ AC_i),\ \bar{\bar{c}}(\underline{cl}\ AC_j) + t/\underline{cl}\ AC_j).$$

W zbiorze przesłanej reguły R7* jest prawdziwy warunek skojarzenia (definicja 4.4):

$$\models_J pre\ (CM_{ik}) \wedge pre\ (CM_{jl}) \wedge inv\ (AC_i, AC_j) \wedge possmatching\ (AC_i, AC_j) \Longrightarrow$$
$$(\forall t)(SEND_e\ [t/dur] \Longrightarrow ((post\ (CM_{ik}) \wedge post\ (CM_{jl}) \wedge inv\ (AC_i, AC_j))$$
$$[\tau_i + t/\tau_i, \tau_j + t/\tau_j])[e/x, \chi_i \cap \lambda/\chi_i, \chi_j \cap \bar{\lambda}/\chi_j]$$

dla stanu $\bar{\bar{s}}$. Wynika stąd, że

$$\models_J (((post\ (CM_{ik}) \wedge post\ (CM_{jl}) \wedge inv\ (AC_i, AC_j) \wedge possmatching\ (AC_i, AC_j))$$
$$[\tau_i + t/\tau_i, \tau_j + t/\tau_j])[e/x, \chi_i \cap \lambda/\chi_i, \chi_j \cap \bar{\lambda}/\chi_j])(\bar{\bar{s}}),$$

a stąd, że

$$\models_J (post\ (CM_{ik}) \wedge post\ (CM_{jl}) \wedge inv\ (AC_i, AC_j) \wedge possmatching\ (AC_i, AC_j))(\bar{s})$$

czyli, że

$$\models_J post\ (CM_{ik})(\bar{s}),$$
$$\models_J post\ (CM_{jl})(\bar{s}),$$

co należało wykonać. ■

Dowód lematu 6.1

Dowód jest indukcyjny ze względu na strukturę programu.

1. Niech $\models_J \{\alpha\}$ skip $\{\beta\}$. Stąd i z definicji semantyki instrukcji skip wynika, że $\models_J \alpha \Longrightarrow \beta$. Zatem po zastosowaniu aksjomatu A1 i reguły R5 otrzymuje się, że

$$\vdash_{PS_p} \{\alpha\}\ skip\ \{\beta\}.$$

2. Niech $\models_J \{\alpha\}$ abort $\{\beta\}$. Z aksjomatu A2 i reguły R5 wynika

$$\vdash_{PS_p} \{\alpha\}\ abort\ \{\beta\}.$$

3. Niech $\models_J \{\alpha\} x:=e\ \{\beta\}$. Stąd i z definicji semantyki instrukcji podstawienia wynika, że

$$\models_J \alpha \Longrightarrow \beta[e/x, \tau + t/\tau]$$

dla dowolnego t takiego, że

$$[\![ASS_e(t)]\!] = tt.$$

Zatem po zastosowaniu aksjomatu A3 oraz reguły R5 pokazuje się, że

$$\vdash_{PS_p} \{\alpha\}\ x:=e\ \{\beta\}.$$

4. Gdy $\models_J \{\alpha\}\ \underline{rcl}\ x\ \{\beta\}$, dowód jest szczególnym przypadkiem p.3.

5. Jeżeli $\models_J \{\alpha\} AC1;AC2\ \{\beta\}$, to niech γ będzie predykatem takim, że

$$[\gamma]_J = wp_{J,p}(AC2, \beta).$$

Wtedy z definicji zachodzi

$$\models_J \{\alpha\} AC1\ \{\gamma\} \quad \text{oraz} \quad \models_J \{\gamma\} AC2\ \{\beta\}.$$

Stąd na mocy hipotezy indukcyjnej i reguły R1

$$\vdash_{PS_p} \{\alpha\} AC1;AC2\ \{\beta\}.$$

6. Jeżeli

$$\models_J \{\alpha\}\ \underline{if}\ \square_{i=1..n}\ b_i \longrightarrow AC_i\ \underline{end}\ \{\beta\},$$

to z definicji instrukcji alternatywy wynika, że

$$\models_J \{\alpha \wedge b_i\}\ \underline{test}\ (IF);\ AC_i\ \{\beta\} \quad (i=1..n).$$

Stąd z założenia indukcyjnego i reguły R2 wynika, że

$$\vdash_{PS_p} \{\alpha\}\ \underline{if}\ \square_{i=1..n}\ b_i \longrightarrow AC_i\ \underline{end}\ \{\beta\}.$$

7. Jeżeli

$$\models_J \{\alpha\}\ \underline{do}\ \square_{i=1..n}\ b_i \longrightarrow AC_i\ \underline{end}\ \{\beta\},$$

to należy znaleźć asercję niezmienniczą γ taką, że

$$\models_J \{\gamma \wedge b_i\}\ \underline{test}\ (DO);\ AC_i\ \{\gamma\}\ (i=1..n),$$

$$\models_J \alpha \Longrightarrow \gamma,$$

$$\models_J \gamma \wedge \neg BOOL\ (DO) \Longrightarrow \beta.$$

Wtedy na podstawie założenia indukcyjnego, reguły R3 oraz reguły R5 wynika, że

$$\vdash_{PS_p} \{\alpha\}\ \underline{do}\ \square_{i=1..n}\ b_i \longrightarrow AC_i\ \underline{end}\ \{\beta\}.$$

Niech γ będzie asercją taką, że

$$[\gamma]_J = wp_{J,p}(DO,\beta),$$

zatem $\models_J \{\gamma\}\ DO\ \{\beta\}$. Należy zauważyć, że również

$$\models_J \{\gamma \wedge BOOL\ (DO)\}\ \underline{if}\ \square_{i=1..n}\ b_i \longrightarrow AC_i\ \underline{end};\ DO\ \{\beta\}$$

przy założeniu, że równoważne są predykaty $TEST_{DO}$ oraz $TEST_{IF}$, gdzie IF oznacza wyżej użytą instrukcję alternatywy.

Stąd i z definicji asercji γ wynika, że

$$\models_J \{\gamma \wedge \mathrm{BOOL}(DO)\}\ \underline{\mathrm{if}}\ \square_{i=1..n}\ b_i \longrightarrow AC_i\ \underline{\mathrm{end}}\ \{\gamma\}$$

a stąd, że

$$\models_J \{\gamma \wedge b_i\}\ \underline{\mathrm{test}}\ (DO);\ AC_i\{\gamma\} \qquad i=1..n).$$

Podobnie należy zauważyć, że

$$\models_J \{\gamma \wedge \mathrm{BOOL}(DO)\}\ \underline{\mathrm{if}}\ \neg\mathrm{BOOL}(DO) \longrightarrow \underline{\mathrm{skip}}\ \underline{\mathrm{end}}\ \{\beta\}$$

przy założeniu, że predykat $TEST_{DO}$ jest równoważny z predykatem $TEST_{IF}$, gdzie IF jest wyżej użytą instrukcją alternatywy. Stąd wynika natomiast, że

$$\models_J \{\gamma \wedge \neg\mathrm{BOOL}(DO)\}\ \underline{\mathrm{test}}\ (DO)\ \{\beta\}$$

oraz

$$\models_J \gamma \wedge \neg\mathrm{BOOL}(DO) \Longrightarrow \beta.$$

Z faktu, że

$$\models_J \{\alpha\}\,DO\,\{\beta\} \quad \text{oraz} \quad \models_J \{\gamma\}\,DO\,\{\beta\}$$

oraz z definicji γ wynika, że

$$\models_J \alpha \Longrightarrow \gamma,$$

co kończy dowód. ■

Dowód lematu 6.2

Obliczenie programu $PR = \|_{k=1..n}\ PC_k$ generuje pewien skończony lub nieskończony ciąg konfiguracji

$$comp = \langle \rho_0, s_0, c_0\rangle, \langle \rho_1, s_1, c_1\rangle, \ \ldots$$

Elementarny krok obliczeniowy w takim obliczeniu

$$step_i = \langle \rho_i, s_i, c_i\rangle \xrightarrow{\lambda_i} \langle \rho_{i+1}, s_{i+1}, c_{i+1}\rangle \qquad (i=0,1,\ \ldots)$$

angażuje jeden lub dwa procesy. Jeżeli dany krok angażuje proces PC_j, to z definicji przyjmuje się, że

$$\langle \rho_i, s_i, c_i\rangle(j) = \langle PC_j^i,\ s_i\big|_{FV(PC_j)\cup\{\tau_j,\chi_j\}},\ c\big|_{\{\underline{\mathrm{cl}}\ PC_j\}}\rangle,$$

gdzie: $\rho_i = \|_{i=1..n}\ PC_k^i$,

oraz jeżeli krok ten nie angażuje procesu PC_j, to z definicji przyjmuje się, że $\langle \rho_i, s_i, c_i\rangle(j)$ jest puste. Jeżeli dany jest ciąg comp, to ciąg

$$comp(j) = \langle \rho_0, s_0, c_0\rangle(j), \langle \rho_1, s_1, c_1\rangle(j), \ \ldots$$

będzie nazywany j-tą składową obliczenia comp.

Należy dodać, że odwrotnie: na podstawie danego zestawu j-składowych comp (j) (j=1..n), można wyznaczyć obliczenie comp. Fakt ten będzie zapisywany w postaci

$$comp = \mathop{!!}_{j=1..n} comp\ (j).$$

Założenia lematu oznaczają, że dla każdego $i \in A$ istnieją takie obliczenia $comp_i$, że gwarantują dla instrukcji składowej AC_i w procesie PC_i s-osiągalność. Wszystkie te obliczenia rozpoczynają się w tej samej sytuacji $\langle s_0, c_0 \rangle$, gdyż

$$\begin{aligned}
s_0(x_i) &= s_0(iks_i) = s(iks_i) \quad \text{dla } i = 1..N\\
s_0(\chi_i) &= s_0(chi_i) = s(chi_i) \quad \text{dla } i = 1..n\\
s_0(\tau_i) &= s_0(tau_i) = s(tau_i) \quad \text{dla } i = 1..n\\
c_0(\underline{cl}\ PC_i) &= s_0(\tau_i) = s(tau_i) \quad \text{dla } i = 1..n,
\end{aligned}$$

a ponadto $\vdash_J \alpha^*(s_0)$.

Dowód prowadzi się indukcyjnie ze względu na sumaryczną długość historii $hglob_i$ obliczeń $comp_i$ $(i \in A)$. Dokładniej: wystarczy brać pod uwagę długość historii oddziaływań niepustych.

Niech długość K sumarycznej długości historii h_i wynosi zero. Oznacza to, że w żadnym z obliczeń $comp_i$ nie zachodzą oddziaływania pomiędzy procesami. Wtedy, ze względu na rozłączność zmiennych procesów, widać, że poszukiwane obliczenie comp można dobrze zdefiniować następującym zestawem składowych:

$$comp\ (i) = comp_i\ (i) \quad \text{dla } i \in A$$
$$comp\ (i) = comp_k\ (i) \quad \text{dla } i \notin A,$$

gdzie k jest dowolnie wybranym elementem ze zbioru A.

Niech teraz $K > 0$. Z założenia indukcyjnego wynika, że teza zachodzi dla $K' < K$, tzn. jeżeli sumaryczna długość historii $\mathop{!!}_{j=1..n} hloc_j$ obliczeń $comp_i$ wynosi K', to dla każdych podhistorii $hloc'_j$ i odpowiadających im obliczeń $comp'_j$ dla $j \in A$ istnieje obliczenie $comp'$ spełniające tezę lematu.

Niech dla pewnego $k \in A$ $hloc_k = hloc'_k \cap \lambda$ oraz $hloc_j = hloc'_j$ dla $j \neq k$, $j \in A$. Oznacza to, że obliczenie $comp_j$ jest konkatenacją

$$comp_k = comp'_k \frown comp''_k$$

gdzie obliczeniu $comp'_k$ towarzyszy historia oddziaływań $hloc'_k$, natomiast obliczeniu $comp''_k$ - historia λ. Obliczenie $comp'_k$ przeprowadza program ze stanu początkowego s_0 do pewnego stanu s_1, a obliczenie $comp''_k$ do stanu s.

Jeżeli λ w obliczeniu $comp''_k$ jest oddziaływaniem wynikającym z upływu czasu przeterminowania w pewnym procesie P_j, to łatwo zauważyć,

że zmienia się stan zmiennych tylko w procesie P_k, a więc - z rozważanego punktu widzenia - wnosi ono taki sam efekt jak oddziaływanie puste. Poszukiwane obliczenie comp ma więc w tym przypadku postać

$$comp = comp' \cap comp_k''$$

gdzie comp' jest obliczeniem istniejącym na mocy założenia indukcyjnego.

Jeżeli λ jest oddziaływaniem synchronizującym lub komunikacyjnym, w którym uczestniczą dwa procesy PC_i, PC_j, to należy rozpatrzeć następujące przypadki.

Niech $i, j \notin A$. W tym przypadku obliczenie $comp_k$ nie zmienia stanu zmiennych dla procesów PC_l dla $l \in A$ ani też nie zmienia instrukcji AC_l osiągniętej w PC_l. Zgodnie z założeniem indukcyjnym, obliczenie comp' jest takie, że zbiór instrukcji AC_l dla $l \in A$ jest s_1-osiągalny. Ponieważ podczas obliczania $comp_k''$ biorą udział tylko procesy PC_i, PC_j, zatem s_1-osiągalność oznacza również s-osiągalność dla tego zbioru instrukcji.

Niech $i \in A$, $j \notin A$. Podobnie jak w poprzednim przypadku łatwo zauważyć, że obliczenie comp', istniejące na mocy założenia indukcyjnego, jest takie, ze zbiór instrukcji AC_l dla $l \in A \setminus \{i\}$ jest s_1-osiągalny. Należy zauważyć również, że - tak jak poprzednio - zbiór ten jest również s-osiągalny. Ponieważ podczas obliczenia $comp_k''$ proces PC_i zmienia swój stan w taki sposób, że stają się s-osiągalne jego składowe AC_i, więc i w tym przypadku obliczenie

$$comp = comp' \cap comp_k''$$

jest poszukiwanym obliczeniem.

Analiza ostatniego przypadku, gdy $i,j \in A$ jest analogiczna do poprzedniego. ∎

Dowód lematu 6.3

Dowód. Niech $CM_i = P_i!W(e)$ oraz $CM_j = P_j?W(x)$ będą parą skojarzonych syntaktycznie instrukcji komunikacji. Warunek zgodności wymaga, aby

$$\vDash_J pre(CM_i) \wedge pre(CM_j) \wedge inv(P_i,P_j) \wedge possmatching(CM_i,CM_j) \Longrightarrow$$
$$(\forall t)(SEND_e[t/dur] \Longrightarrow ((post(CM_i) \wedge post(CM_j) \wedge inv(P_i,P_j))$$
$$[\tau_{P_i}+t/\tau_{P_i}, \tau_{P_j}+t/\tau_{P_j}])[e/x, \chi_{P_i}\ \lambda/\chi_{P_i}, \chi_{P_j}\ \bar{\lambda}/\chi_{P_j}],$$

gdzie $\lambda = (P_j,P_i,W,!,e,\tau_{P_i})$.

Niech dla pewnego stanu s zachodzi

$$\models_J(\text{pre } (CM_i) \wedge \text{pre } (CM_j) \wedge \text{inv } (P_i,P_j) \wedge \text{possmatching } (CM_i,CM_j))(s)$$

Z definicji asercji pre oraz lematu 6.2 wynika, że

$$(\exists s',s'' \in \text{Astates})(\exists c',c'' \in \text{Clocks})(\exists \overset{||}{_{k=1..n}} \text{hloc}_k)$$

$$(\models_J \alpha^*(s) \wedge \text{PROP } (s',c') \wedge$$

$$\left(\langle \overset{||}{_{k=1..n}} PC_k,s',c'\rangle \xrightarrow{\overset{||}{k=1..n} \text{hloc}_k} {}^*\langle \overset{||}{_{k=1..n}} PC''_k,s'',c''\rangle\right) \wedge$$

$$\text{PROP } (s'',c'') \wedge$$

$$(s'' \mid FV(PC_1) \cup \{\tau_{P_1},\chi_{P_1}\} = s \mid FV(PC_1) \cup \{\chi_{P_1},\tau_{P_1}\} \wedge$$

$$PC''_1 = \underline{\text{before}}\ (CM_1,CP_1) \quad \text{dla} \quad 1 \in \{i,j\}) \wedge$$

$$s''(\tau_{P_i}) = s(\tau_{P_j})).$$

Z definicji semantyki języka wynika, że jeżeli w trakcie obliczenia osiągnięto konfigurację taką, że

$$PC'_i = \underline{\text{before}}\ (CM_i,PC_i),$$

$$PC'_j = \underline{\text{before}}\ (CM_j,PC_j)$$

oraz

$$CM_i = \overline{CM}_j,$$

to kolejnym krokiem obliczenia może być tylko wykonanie pary skojarzonych dynamicznie instrukcji, gdyż

$$\models_J \text{possmatching } (CM_i,CM_j)(s).$$

Niech

$$s''' \in \text{Asem } [\![CM_i \,||\, CM_j]\!]\ (s)$$

oraz

$$s'''' \in \text{Asem } [\![CM_i \,||\, CM_j]\!]\ (s'')$$

Pomiędzy tymi stanami zachodzi związek:

$$s''' \mid FV(PC_1) \cup \{\tau_{P_1},\chi_{P_1}\} = s'''' \mid FV(PC_1) \cup \{\tau_{P_1},\chi_{P_1}\} \quad 1 \in \{i,j\}.$$

Dla dowolnych oddziaływań λ, $\bar{\lambda}$ jakie mogą zajść pomiędzy PC_i, PC_j w trakcie wykonywania CM_i, CM_j otrzymuje się obliczenie:

$$\langle \overset{||}{_{k=1..n}} PC_k,s',c'\rangle \xrightarrow{\overset{||}{k=1..n} \text{hloc}'_k} {}^*\langle \overset{||}{_{k=1..n}} PC''_k,s'''',c''''\rangle,$$

gdzie

$$hloc'_i = hloc_i \cap \lambda, \quad hloc'_j = hloc_j \cap \bar{\lambda}, \quad hloc'_k = hloc_k$$

dla $k \neq i,j$

oraz

$$PC''_l = \underline{after}\ (CM_l, PC_l) \quad \text{dla} \quad l \in \{i,j\},$$

$$PC''_k = PC'_k \quad \text{dla} \quad k \neq i,j.$$

Oczywiście

$$PROP\ (s'''', \sigma'''') = tt \quad i \quad s''''(\tau_{P_i}) = s''''(\tau_{P_j}).$$

Stąd i z definicji asercji post wynika, że

$$\models_J (post\ (CM_i) \wedge post\ (CM_j) \wedge inv\ (P_i,P_j) \wedge possmatching\ (CM_i,CM_j))(s''''),$$

ale również, że

$$\models_J (post\ (CM_i) \wedge post\ (CM_j) \wedge inv\ (P_i,P_j) \wedge possmatching\ (CM_i,CM_j))(s''').$$

Z definicji semantyki wykonania pary instrukcji CM_i, CM_j wynika, że pomiędzy s, s''' zachodzi związek

$$s''' = s(s(\tau_{P_i}) + t/\tau_{P_i},\ s(\tau_{P_j}) + t/\tau_{P_j},\ [\![e]\!]_s/x,$$
$$s(\chi_{P_i}) \cap \lambda / \chi_{P_i},\ s(\chi_{P_j}) \cap \bar{\lambda} / \chi_{P_j})$$

gdzie t jest takie, że

$$[\![SEND_e[t/dur]]\!]_s = tt,$$

natomiast

$$\lambda = (P_j, P_i, W, !, [\![e]\!]_s, s(\tau_{P_i})).$$

Stąd wynika, że prawdziwa jest prawa strona warunku skojarzenia dla stanu s. ■

Dowód lematu 6.4

Niech nosyn będzie pustą instrukcją komunikacji użytą w dowodzie instrukcji TH, w procesie PC_i, postaci:

$$\underline{th} \vee \underline{wt}\ \Box_{k=1..n}\ b_k;\ CM_k \longrightarrow AC_k\ \underline{lt}\ AC_0\ \underline{end}$$

Asercje pre i post mają spełniać warunek

$$\models_J pre\ (\underline{nosyn}) \wedge timeout\ (TH) \Longrightarrow post\ (\underline{nosyn})[\chi_{P_i} \cap \lambda / \chi_{P_i}]$$

gdzie: $\lambda = (P_i, \tau_{P_i})$.

Niech s będzie stanem takim, że

$$\models_J \text{pre } (\underline{\text{nosyn}})(s).$$

Jeżeli $\models_J$ timeout (TH)(s), to, oczywiście, warunek przeterminowania jest spełniony dla dowolnego przebiegu dalszej części obliczenia. Jeżeli natomiast $\models_J$ timeout (TH)(s), to oznacza, że jedynym możliwym krokiem obliczeniowym jest upływ odcinka czasu przeterminowania i realizacja nosyn, z czego - podobnie jak w lemacie poprzednim - wynika, że warunek przeterminowania także zachodzi. ■

LITERATURA

[1] ACKERMANN W. B., Data flow languages, Computer, 1982, vol. 15, (2), 15-25.

[2] AGGARWAL S., BARBARA, METH K. Z., SPANNER: a tool for specification, analysis and evaluation protocols, IEEE Trans. on Software Engineering, 1987, vol. SE-12, 1218-1237.

[3] AJMONE MARSAN M., BALBO G., TRIVEDI K. (eds), International workshop on timed Ptri nets, Torino 1985, Computer Society Press, 1985.

[4] AMERICA P., BAKKER J. W. de, KOK J. N., RUTTEN J., Operational semantics of parallel object-oriented language, Report CS-R 8515, Centre for Mathematics and Computer Science, Amsterdam 1985.

[5] ANDLER S., Synchronization primitives and the verification of concurrent programs, Proceedings of the Second International Symposium on Operating Systems, IRIA, Le Chesnay, 1978.

[6] ANDREWS G. R., The distributed programming language SR- mechanisms, design and implementation, Software Practice and Experience, 1982, vol. 12, 719-754.

[7] ANDREWS G. R., SCHNEIDER F. B., Concepts and notations for concurrent programming, Computing Surveys, 1983, vol. 15, 3-43.

[8] APT K. R., FRANCEZ N., ROEVER W. P. de, A proof system for communicating sequential processes, Report RUU-CS-80-4, Rijksuniversiteit Utrecht, Vakgroep informatica, May 1980.

[9] APT K. R., Ten years of Hoare's Logic: a survey - part I, Transactions on Programming Languages and Systems, 1981, vol. 3, 431-483.

[10] APT K. R., Recursive assertions and parallel programs, Acta Informatica, 1981, vol. 15, 219-232.

[11] APT K. R., PLOTKIN G. D., Countable nondeterminism and random assignment, Technical Report no. 98, Computer Science Department, University of Edinburgh, 1982.

[12] APT K. R., Formal justification of a proof system for communicating sequential processes, Journal of ACM, 1983, vol. 30, 197-216.

[13] APT K. R., OLDEROG E. R., Proof rules and transformations dealing with fairness, Science of Computer Programming, 1983, vol. 3, 65-100.

[14] APT K. R., Ten years of Hoare's logic: a survey - part II: nondeterminism, Theoretical Computer Science, 1984, vol. 28, 83-109.

[15] APT K. R. (ed.), Logics and models of concurrent systems, Springer Verlag, 1985.

[16] BAKKER J. W. de, KOK J. N., MEYER E. R., ZUCKER J. I., Contrasting themes in the semantics of imperative concurrency, Report CS-R 8603, Centre for Mathematics and Computer Science, Amsterdam, 1986.

[17] BAKKER J. W. de, MEYER J. J. Ch., OLDEROG E. R., ZUCKER J. I., Transition systems, metric spaces and ready sets in the semantics of uniform concurrency, Report CS-R 8601, Centre for Mathematics and Computer Science, Amsterdam, 1986.

[18] BANACHOWSKI L., KRECZMAR A., MIRKOWSKA G., RASIOWA H., SAWICKI A., An introduction to algorithmic logic - methematical investigations in the theory of programs [In:] MAZURKIEWICZ A., PAWLAK Z. (eds), Mathematical Foundations of Computer Science, PWN, 1977.

[19] BERGSTRA J. A., KLOP J. W., Process algebra for synchronous communication, Information and Control, 1984, vol. 60, 109-137.

[20] BERGSTRA J. A., KLOP J. W., TUCKER J. V., Process algebra with asynchronous communication mechanisms, Lecture Notes in Computer Science, 1985, vol. 197, 26-95.

[21] BEST E., Relational semantics of concurrent programs [In:] [29], 431-452.

[22] BEST E., Concurrent behaviour: sequences, processes and axioms, Lecture Notes in Computer Science, 1985, vol. 197, 221-245.

[23] BEST E., COSY: its relation to nets and CSP, Lecture Notes in Computer Science, 1987, vol. 255, 416-440.

[24] BIAŁASIEWICZ J., ADEREK A., MALISZEWSKI K., Oprogramowanie podstawowe komputerowych systemów sterowania, Warszawa, WNT, 1979.

[25] BJØRNER D., JONES C. B., The Vienna Development Method: the META Language, Lecture Notes in Computer Science, Springer Verlag, 1978, vol. 61.

[26] BJØRNER D., OEST O. N. (eds), Towards a formal description of Ada, Lecture Notes in Computer Science, Springer Verlag, 1980, vol. 98.

[27] BJØRNER D., JONES C. B., Formal specification and software development, Prentice Hall International, 1982.

[28] BJØRNER D., PREHN S., Software engineering aspects of VDM:the Vienna Development Method [In:] FERRARI D., BOLOGANI M., GOUGEN J. (eds), Theory and practice of software technology, North Holland, Publishing Company, 1983.

[29] BJØRNED D. (ed.), Formal descriptions of programming concepts - II, Proceedings of IFIP Working Conference 1982, North Holland Publishing Company, 1983.

[30] BLIKLE A., Notes on the mathematical semantics of programming languages, Prace IPI PAN, Nr 445, 1981.

[31] BLIKLE A., A metalanguage for naive denotational semantics, Collana Cnet 104, ETS Pisa, 1983.

[32] BLIKLE A., TARLECKI A., Naive denotational semantics, Information Processing 83, IFIP, North Holland Publishing Company, 1983.

[33] BRINCH HANSEN P., Distributed process: a concurrent programming concept, Communications of ACM, 1978, vol. 21, 934-941.

[34] BRINCH HANSEN P., Edison - a multiprocessor language. Software Practice and Experience, 1981, vol. 11, 325-362.

[35] BROOKES S. P., HOARE C. A. R., ROSCOE A. W., A theory of communicating sequential processes, Journal of ACM, 1984, vol. 31, 560-599.

[36] BROOKES S. D., On the axiomatic treatment of concurrency, Lecture Notes in Computer Science, 1985, vol. 197, 1-34.

[37] BROOKES S. D., ROSCOE A. W., WINSKEL G. (eds), Seminar on concurrency, Lecture Notes in Computer Science, Springer Verlag, 1985, vol. 197.

[38] BRAUER W., REISIG W., ROZENBERG G. (eds), Petri nets: applications and relations to other models of concurrency, Lecture Notes in Computer Science, Springer-Verlag, 1987, vol. 255.

[39] BROY M., Fixed point theory for communication and concurrency [In:] [29], 125-146.

[40] BROY M., WIRSING M., On the algebraic specification of finitary infinite communicating sequential processes [In:] [29], 171-196.

[41] BRUNO G., ELIA A., Operational specification of process control systems: execution of PROT nets using OPS5 [In:] [123], 35-40.

[42] CCITT High Level Language (CHILL), Recommendation Z200, CCITT, Geneva, 1981.

[43] CHANDY K., MISRA J., An axiomatic proof technique for networks of communicating processes, Technical Report TR-98, Department of Computer Science, University of Texas at Austin, 1979.

[44] CHEN M. C., Transformaticns of parallel programs in Crystal [In:] [123], 455-462.

[45] CINDIO F. DE, MICHELIS G. DE, POMELLO L., SIMONE C., Equivalence notions for concurrent systems, Lecture Notes in Computer Science, 1988, vol. 266, 29-33.

[46] CINDIO F. DE, MICHELIS G. de, POMELLO L., SIMONE C., Milner's communicating systems and Petri nets, Lecture Notes in Computer Science, 1988, vol. 266, 40-59.

[47] CLINT M., Program proving: coroutines, Acta Informatica, 1973, vol. 2, 50-63.

[48] CONSTABLE R. L., Constructive mathematics and automatic program writes [In:] Freimann C. V. (ed.), Information Processing 71, North Holland Publishing Company, 1972, 229-233.

[49] CONSTABLE R. L., BATES J. L., The nearly ultimate PEARL, Technical Report TR-83-551, Departament of Computer Science, Cornell University, Ithaca, 1984.

[50] COOK R. P., Star Mod - a language for distributed programming, IEEE Transactions on Software Engineering, 1980, vol. SE-6, 563--571.

[51] COOK S. A., Soundness and completeness of an axiom system for program verification, SIAM Journal on Computing, 1978, vol. 7, 70--90.

[52] CULIK K., RIZKI M. M., Logic versus mathematics in computer science education, ACM SIGCSE Bulletin, 1983, vol. 15, (1), 14-20

[53] CZAGA L., Making nets structured and abstract, Lecture Notes in Computer Science, 1985, vol. 222, 181-202.

[54] DAMM W., JOSKO B., A sound and relatively complete Hoare-logic for a language with higher type procedures, Acta Informatica, 1983, vol. 20, 59-101.

[55] DASARATHY B., Timing constrains of real-time systems: constructs for expressing them, methods of validating them, Proceedings of the Real-time Systems Symposium, IEEE, 1982, 197-204.

[56] DEMBIŃSKI P., MAŁUSZYŃSKI J., Matematyczne metody definiowania języków programowania, WNT, 1981.

[57] DIJKSTRA E. W., Umiejętność programowania, WNT, 1978.

[58] DIX T. I., Exceptions and interrupts in CSP, Science of Computer Programming, 1983, vol. 3, 189-204.

[59] DIX T. I., Preemtive guards in CSP, Technical Report 83/5, Department of Computer Science, University of Melbourne, 1983.

[60] DUBIELEWICZ I., FRYŹLEWICZ Z., HUZAR Z., Komunikacja procesów sekwencyjnych w czasie rzeczywistym, Raport PRE 24/81, Centrum Obliczeniowe, Politechnika Wrocławska, 1981.

[61] ELZER P., ROESSLER R., Real-time languages and operating systems [In:] [74], 351-369.

[62] EMERSON E. A., CLARKE E. M., Characterizing correctness properties of parallel programs using fixpoints, Technical Report TR-04-80, Aiken Computation Laboratory, Harvard University, 1980.

[63] FELDMAN, High level programming for distributed computing, Communications of ACM, 1979, vol. 22, 353-368.

[64] FLOYD R. W., Assigning meanings to programs, Proceedings of AMS Symposium in Applied Mathematics, American Mathematical Society, 1967, 19-31.

[65] FOLKJAR P., BJØRNER D., A formal model of a generalized CSP - like language, Information Processing 80, IFIP, North Holland Publishing Company, 1980.

[66] FRANCEZ N., On achieving distributed termination, [In:] [95], Springer Verlag, 1979, 300-315.

[67] FRANCEZ N., HOARE C. A. R., LEHMANN D. J., ROEVER W. P. de, Semantics of nondeterminism, concurrency and communication, Journal of Computer and System Sciences, 1979, vol. 19, 209-308.

[68] FRYŹLEWICZ Z., Emulator stacji wsadowej ICL-7020 - projekt struktury, Komunikat nr 686, Instytut Cybernetyki Technicznej, Politechnika Wrocławska, 1978.

[69] GAUTIER T., LE GUERNIC P., BENVENISTE A., BOURNAI P., Programming real-time with events and data flow [In:] [123], 469-474.

[70] GENRICH H. J., LAUTENBACH K., System modelling with high-level Petri nets, Theoretical Computer Science, 1981, vol. 13, 109-136.

[71] GENRICH H. J., Net theory and application [In:] [123], 823-831.

[72] GERGELY T., URY L., Program behaviour specification through explicit time consideration, Information Processing 80, IFIP, North Holland Publishing Company, 1980, 107-111.

[73] GERTH R., A sound and complete Hoare axiomatisation of the Ada - rendez vouz, Proceedings of the 9-th Colloguim on Automata, Languages and Programming, Lecture Notes in Computer Science, Springer Verlag, 1982, vol. 140, 252-264.

[74] GLASS R. L., Real-time software, Prentice Hall Inc., Englewood Cliffs, 1983.

[75] GOLDBERG A., ROBSON D., Smalltalk - 80 - the language and its implementation, Addison-Wesley, 1983.

[76] GORDON M. E., ROBINSON W. B., Using preliminary Ada in a process control application [In:] [74], 411-431.

[77] GREIF I., Semantics of communicating parallel processes, Ph. D. Dissertation, MAC TR-154, Massachusetts Institute of Technology, 1975.

[78] GRIES D., (ed.), Programming methodology - a collection of articles by members of IFIP WG 2.3, Springer Verlag, 1978.

[79] HABERMANN A. N., Synchronization of communicating processes, Communications of ACM, 1972, vol. 15, 171-176.

[80] HAREL D., Proving the correctness of regular deterministic programs: a unifying survey using dynamic logic, Theoretical Computer Science, 1980, vol. 12, 61-81.

[81] HART S., SHARIR M., Probabilistic propositional temporal logic, Technical Report 84-002, Institute of Computer Science, Tel Aviv University, 1984.

[82] HENNESSY J. L., KIEBURTZ R. B., The formal definition of a real-time language, Acta Informatica, 1981, vol. 16, 309-345.

[83] HENNESSY M. C. B., PLOTKIN G. D., Full abstraction for a simple programming language, Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Springer Verlag, 1979, vol. 74, 108-120.

[84] HENNESSY M. C. B., PLOTKIN G. D., A term model for CCS, Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Springer-Verlag, 1980, vol. 88, 261-274.

[85] HENNESSY M. C. B., Synchronous and asynchronous experiments on processes, Information and Control, 1983, vol. 59, 36-83.

[86] HENNESSY M. C. B., Proving systolic systems correct, Technical Report, Department of Computer Science, University of Edinburgh, 1984.

[87] HENNESSY M. C. B., An algebraic theory of fair asynchronous communicating processes, Report CSR-171-84, Department of Computer Science, University of Edinburgh, 1984.

[88] HEWITT C. E., ATKINSON R. R., Synchronization in actor systems, Proceedings of the Conference Principles of Programming Languages, Los Angeles, 1977.

[89] HEWITT C. E., ATKINSON R. R., Specification and proof techniques for serializers, IEEE Transactions on Software Engineering, 1979, vol. SE-5, 10-23.

[90] HEWITT C. E., ATTARDI G., An axiomatic denotation specification of a concurrent programming language, Working Paper, Computer Science Department, Massachusetts Institute of Technology, May 1978.

[91] HEWITT C. E., ATTARDI G., LIEBERMAN H., Specyfing and proving properties of guardiens for distributed systems. Working Paper, Artificial Intelligence Department, Massachusetts Institute of Technology, October 1978

[92] HEWITT C. E., BAKER H., Actors and continous functionals, Memo 436A, Artificial Intelligence Department, Massachusetts Institute of Technology, 1977.

[93] HEWITT C. E., BAKER H., Laws for communicating parallel processes, Information Processing 77, IFIP, North-Holland Publishing Company, 1977, 987-982.

[94] HOARE C. A. R., An axiomatic basis for computer programming, Communications of ACM, 1969, vol. 12, 576-583.

[95] HOARE C. A. R., Towards a theory of parallel programming [In:] Operating systems techniques (HOARE C. A. R., PERROT R. N. eds), Academic Press, 1972.

[96] HOARE C. A. R., Communicating sequential processes, Communications of ACM, 1978, vol. 21, 666-677.

[97] HOARE C. A. R., A model for communicating sequential processes and supplementary notes, Technical Monograph PRG-22, Computer Laboratory, Oxford University, 1981.

[98] HUZAR Z., Zagadnienia impasów w schematach blokowych programów równoległych, Podstawy Sterowania, 1978, t. 8, z. 1, 69-83.

[99] HUZAR Z., Uogólnione monitory w języku Concurrent Pascal, Podstawy Sterowania, 1980, t. 10, z. 1, 43-55.

[100] HUZAR Z., Semantyka programów z operacjami wejścia-wyjścia, Podstawy Sterowania, 1981, t. 11, z. 1, 33-49.

[101] HUZAR Z., O najsłabszych warunkach wstępnych dla procesów sekwencyjnych komunikujących się w czasie rzeczywistym, Raport PRE 8/81, Centrum Obliczeniowe, Politechnika Wrocławska, 1981.

[102] HUZAR Z., Język programowania współbieżnego Maxwell, Raport SPR 22/82, Centrum Obliczeniowe, Politechnika Wrocławska, 1982.

[103] HUZAR Z., Programowa specyfikacja protokółu transferu deskryptorów, Raport SPR 28/82, Centrum Obliczeniowe, Politechnika Wrocławska, 1982.

[104] HUZAR Z., Nowe podejście do opisu semantyki komunikujących się procesów sekwencyjnych, Studia i Materiały, 2, 273-288, Centrum Obliczeniowe, Politechnika Wrocławska, 1983.

[105] HUZAR Z., KUŹNIARZ L., SPŁAWSKI Z., Przegląd wybranych systemów wspomagania konstruowania programów, Raport SPR 35/85, Centrum Obliczeniowe, Politechnika Wrocławska, 1985.

[106] HUZAR Z., Wstęp do programowania współbieżnego, Skrypt Politechniki Wrocławskiej, 1985.

[107] ICHBIAH J. D., et all, Preliminary Ada reference manual, SIGPLAN Notices, 1979, vol. 14(6).

[108] ISZKOWSKI W., MANIECKI M., Programowanie współbieżne, WNT, 1982.

[109] JACOBS D., GRIES D., General correctness: a unification of partial and total correctuen, Acta Informatica, 1985, vol. 22, 67-83.

[110] JACOBSON I., FDL: a language for desinning large real-time systems [In:] [123], 463-468.

[111] JAHANIAN F., MOK A. K. L., A graph-teoretic approach for timing analysis and its implementation, IEEE Trans. on Computers, 1987, vol. C-36, 961-975.

[112] KAHN G., MAC-QUEEN D. B., Coroutines and networks of parallel processes, Information Processing 77, IFIP, North Holland Publishing Company, 1977, 993-998.

[113] KAHN G. (ed.), Semantics of concurrent computations, Proceedings of the International Symposium, Lecture Notes in Computer Science, Springer Verlag, 1979, vol. 70.

[114] KELLER R. M., Formal verification of parallel programs, Communications of ACM, 1976, vol. 19, 371-384.

[115] KELLER R. M., Semantics and applications of function graphs, Report UUCS-80-112, Department of Computer Science, University of Utah, 1980.

[116] KOPETZ H., Accuracy of time measurement in distributed real-time systems, MARS Report Nr 4/85, Institut für Praktische Informatik Softwaretechnologie und Echtzeitsysteme, Technische Universität Wien, 1985.

[117] KOPETZ H., OCHSENREITER W., Clock synchronization in distributed real-time systems, IEEE Trans. on Computers, 1987, vol. C-36, 933-940.

[118] KOTOV E. V., On basic parallel language, Information Processing 80, IFIP, North Holland Publishing Company, 1980, 229-240.

[119] KOTOV E. V., MIKLOŠKO J. (red.), Algoritmy matematičeskogo obespečenia i architektura mnogoprocessornych vyčislitelnych sistem, Sibirskoe Otdelenie AN SSSR, Izdatelstvo Nauka, 1982.

[120] KOTOV V. E., CHERKASOVA L. A., On structural properties of generalized processes, Lecture Notes in Computer Science 1988, vol. 266, 288-306.

[121] KOWALSKI R., Logic as a fifth generation computer language, [In:] [187], 73-88.

[122] KRAMER J., MAGEE J., SLOMAN M., LISTER A., CONIC: an integrated approach to distributed computer control systems, IEE Proceedings, 1983, vol. 130, Pt. E, No 1, 1-10.

[123] KUGLER H. J. (ed.), Information processing 86, Elsevier Science Publishers B. V. IFIP, 1986 (Participants edition).

[124] LAMPORT L., Time, clocks and the ordering of events in a distributed systems, Communications of ACM, 1978, vol. 21, 558-565.

[125] LAMPORT L., The "Hoare Logic" of concurrent programs, Acta Informatica, 1980, vol. 14, 21-37.

[126] LAMPORT L., "Sometime" is sometimes "not never", Communications of ACM, 1980, vol. 23, 174-185.

[127] LAMPORT L., An assertional correctuess proof of a distributed algorithm, Op. 61, Computer Science Laboratory, SRI International, 1982.

[128] LAMPORT L., What good is temporal logic, Information Processing 83, IFIP, Elsevier Science Publishers. 1983. 657-668.

[129] LAMPORT L., Specyfing concurrent program modules, ACM Transactions on Programming Languages and Systems, 1983, vol. 5, 190-222.

[130] LAMPORT L., SCHNEIDER F. R., The "Hoare Logic" of CSP and all that, ACM Transactions on Programming Languages and Systems, 1984, vol. 6, 281-296.

[131] LAMSWEERDE A. VAN, SINTZOFF M., Formal derivation of strongly correct concurrent programs, Acta Informatica, 1979, vol. 12, 1-31.

[132] LANN G., LE, Distributed systems-towards a formal approach, Information Processing 77, IFIP, North Holland Publishing Company, 1977, 155-160.

[133] LANN G. LE, On real-time distributed computing. Information Processing 83, Elsevier Science Publishers, 1983, 741-753.

[134] LEE I., DAVIDSON S. B., Adding time to synchronous process communications, IEEE Trans. on Computers, 1987, vol. C-36, 941-948.

[135] LEVI G., New research directions in logic specification languages [In:] [123], 1005-1008.

[136] LEVIN G. M., Proof rules for communicating sequential processes. Ph. D. Thesis, Report 80-435, Department of Computer Science, Cornell University, Ithaca, 1980.

[137] LEVIN G. M., GRIES D., A proof technique for communicating sequential processes, Acta Informatica, 1981, vol. 15, 281-302.

[138] LI W., An operational approach to sematics and translation for concurrent programming languages, Report CST-20-83, Ph. D. Thesis, Department of Computer Science, University of Edinburgh, 1983.

[139] LISKOV B., Primitives for distributed computing, Report TM-175, Laboratory of Computer Science, Massachusetts Institute of Technology, 1979.

[140] LISKOV B., On linguistic support for distributed programs, Computer Structures Group Memo 201-1, Laboratory of Computer Science, Massachusetts Institute of Technology, 1981.

[141] LISKOV B., HERLIHY M., Issues in process and communication structure for distributed programs, Programming Mathematics Group Memo 38, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983.

[142] LISKOV B., SCHEIFLER R., Guardian and actions: linguistic support for robust, distributed programs, ACM Transactions on Programming Languages and Systems, 1983, vol. 5, 381-404.

[143] LUBACHEVSKY B. D., An approach to automatic the verification of compact parallel coordination programs - part I, Acta Informatica, 1984, vol. 21, 125-169.

[144] LUCKHAM D. C., HENKE F. W. VON, LARSEN H. J., STEVENSON D. R., Adam: an Ada-based language for multiprocessing, Software Practice and Experience, 1984, vol. 14, 605-641.

[145] MANNA Z., Six lectures on the logic of computer programming, NSF Regional Conference, Rensselaer Polytechnic Institute, Troy, New York, 1978.

[146] MANNA Z., PNUELI A., Verification of concurrent programs, part I: the temporal framework, Report STAN-CS-81-836, Department of Computer Science, Stanford University, 1981.

[147] MANNA Z., PNUELI A., Verification of concurrent programs, part II: temporal proof principles, Report STAN-CS-81-843, Department of Computer Science, Stanford University, 1981.

[148] MANNA Z., WOLPER P., Synthesis of communicating processes from temporal logic specifications, ACM Transactions on Programming Languages and Systems, 1984, vol. 6, 68-93.

[149] MARTIN-LOF D., Constructive mathematics and computer programming, 6th International Congress for Logic, Methodology and Philosophy of Science, PWN and North Holland Publishing Company, 1982, 153--175.

[150] MAZURKIEWICZ A., Invariants of concurrent programs, Proceedings of IFIP-INFOPOL Conference of Information Processing, North Holland Publishing Company, 1977, 353-372.

[151] MAZURKIEWICZ A., Concurrent program schemes and their interpretations, DAIMI PB-78, Aarhus University Publications, 1977.

[152] MAZURKIEWICZ A., Traces, histories, graphs: instances of a process monoid, Lecture Notes in Computer Science, 1984, vol. 176, 115--133.

[153] MAZURKIEWICZ A., Semantics of concurrent systems: a modular fixed -point trace approach, Lecture Notes in Computer Science, 1985, vol. 188, 353-375.

[154] MAZURKIEWICZ A., Trace theory, Lecture Notes in Computer Science, 1987, vol. 255, 279-324.

[155] MEYER J.-J.CH, Fixed points and the arbitrary and fair merge of a fairly simple class of processes. Technical Report IR-89/IR-92, Free University, Amsterdam, 1984.

[156] MILI A., DESHARNAIS J., A system for classifying program verfication methods. Assining meaning to program verification methods, Proceedings of 7th International Conference on Software Engineering, IEEE, 1984, 499-509.

[157] MILNER R., A calculus of communicating systems, Lecture Notes in Computer Science, Springer Verlag, 1980, vol. 92.

[158] MILNER R., Calculi for synchrony and asynchrony, Theoretical Computer Science, 1983, vol. 25, 267-310.

[159] MILNER R., Lectures on a calculus for communicating systems, Lecture Notes in Computer Science, 1985, vol. 197, 197-220.

[160] MILNER R. Process constructors and interpretations [In:] [123], 507-514.

[161] NGUYEN U., DEMERS A., GRIES D., OWICKI S., A model and temporal proof system for networks of processes, Distributed Computing, 1986, vol. 1, 7-25.

[162] NICOLA R. DE, Transition systems and testing preorders: an alternative to Petri nets for systems specifications [In:] [123], 833-836.

[163] NIELSEN M., CCS - and its relationship to net theory, Lecture Notes in Computer Science, 1987, vol. 255, 393-415.

[164] NISHIMURA T., Formalization of concurrent processes, Information Processing 77, IFIP, North Holland Publishing Company, 1977, 929-937.

[165] O'DONNELL M. J., A critique of the foundations of Hoare style programming logics, Communications of ACM, 1982, vol. 25, 927-935.

[166] OLDEROG E.-R., HOARE C. A. R., Specification oriented semantics for communicating processes, Technical Monograph PRG-37, Programming Research Group, Oxford University, 1984.

[167] OLDEROG E.-R., APT K. R., Fairness in parallel programs: the transformational approach, Bericht Nr 8402, Institute für Informatik und Practische Mathematik, Christian-Albrechts-Universität, Kiel, 1984.

[168] OLDEROG E.-R., TCSP: theory of communicating sequential processes, Lecture Notes in Computer Science, 1987, vol. 255, 441-465.

[169] OLDEROG E.-R., Operational Petri net semantics for CCSP, Institute für Informatik und Praktische Mathematik, Christian-Albrechts Universität, Kiel, Bericht, Nr 8704, 1987.

[170] OLDEROG E.-R., Operational Petri net semantic for CCSP, Lecture Notes in Computer Science, 1988, vol. 266, 196-223.

[171] OWICKI S., Axiomatic proof technique for parallel programs, Ph. D. Thesis, Computer Science Department, Cornell University, Ithaca, 1975.

[172] OWICKI S., GRIES D., Verifying properties of parallel programs: an axiomatic approach, Communications of ACM, 1976, vol. 19, 279-285.

[173] OWICKI S., GRIES D., An axiomatic proof technique for parallel programs, Acta Informatica, 1976, vol. 6, 319-340.

[174] PAGAN F. G., Formal specification of programming languages - a panoramic primer. Prentice Hall Incorporation, 1981.

[175] PENG Z., Construction of asynchronous concurrent systems from their behavioral specifications [In:] [123], 859-864.

[176] PLOTKIN G. D., A powerdomain construction, SIAM Journal Computer, 1976, vol. 5, 452-487.

[177] PLOTKIN G. D., Dijkstra's predicate transformers and Smyth's powerdomains, in: Abstract software specification, (BJØRNER D. ed.), Lecture Notes in Computer Science, 1980, vol. 86, 527-553.

[178] PLOTKIN G. D. An structural approach to operational semantics, Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[179] PLOTKIN G. D., An operational semantics for CSP [In:] [29], 199--223.

[180] PNUELI A., Specification and development of reactive systems [In:] [123], 845-858.

[181] POGORZELSKI W. A., Klasyczny rachunek kwantyfikatorów, Zarys teorii, PWN, 1981.

[182] ROEVER DE W.-P., Questions to Robin Milner - a responder's commentary [In:] [123], 515-518.

[183] RONCKEN M., DIEPEN N. VAN, KRAMER M., ROEVER W. DE, A proof system for Brinch Hansen's distributed processes, RUU-CS-81-5, Vakgroep Informatica, Rijksuniversiteit Utrecht, 1981.

[184] ROZENBERG G. (ed.), Advances in Petri nets 1984, Lecture Notes in Computer Science, Springer-Verlag 1985, vol. 188.

[185] ROZENBERG G. (ed.), Advances in Petri Nets 1897, Lecture Notes in Computer Science, 1988, vol. 266.

[186] SALWICKI A., MÜLDNER T., On the algorithmic properties of concurrent programs, [In:] Logic of programs (ENGELER E. ed.), Lecture Notes in Computer Science, Springer Verlag, 1981, vol. 125, 169--197.

[187] SCARROTT G. G. (ed.) The fifth generation computer project, Pergamon Infotech Limited, State of the Art Report 11:1, 1983.

[188] SHAPIRO E., TAKEUCHI A., Object oriented programming in Concurrent Prolog, Report CS 83-08, Department of Applied Mathematics, Weizmann Institute of Science, 1983.

[189] SCHLICHTING R. D., SCHNEIDER F. B., Using message passing for distributed programming: proof rules and disciplines, ACM, Transactions on Programming Languages and Systems, 1984, vol. 6, 402-431.

[190] SCHNEIDER F. B., Synchronization in distributed programs, ACM Transactions on Programming Languages and Systems, 1982, vol. 4, 179-195.

[191] SCHNEIDER F. B., Byzantine generals in action: implementing fail--stop processors, ACM Transactions on Computer Systems, 1984, vol. 2, 145-154.

[192] SCHNEIDER F. B., GIES D., SCHLICHTING R. D., Fault-tolerant broadcast, Science of Computer Programming, 1984, vol. 4, 1-15.

[193] SCHUTZ H. A., On the design of a language for programming real--time concurrent processes, IEEE Transactions on Software Engineering, 1979, vol. SE-5, 248-255.

[194] SHIELDS M. W., WRAY M. J., A CCS specification of the OSI network service, Report CSR-136-83, Department of Computer Science, University of Edinburgh, 1983.

[195] SHOENFIELD J. R., Mathematical logic, Addison-Wesley, 1967.

[196] SILBERSCHATZ A., Communication and synchronization in distributed systems, IEEE Transactions on Software Engineering, 1979, vol. SE-5, 542-546.

[197] SILBERSCHATZ A., Port directed communication, Computer Journal, 1981, vol. 24, 78-81.

[198] SLOMAN M., KRAMER J., MAGEE J., The Conic toolkit for building distributed systems, 6th IFAC Distributed Computer Control System Workshop, Monterey, 1985.

[199] SOUNDARARAJAN N., Correctness proofs of CSP programs, Theoretical Computer Science, 1983, vol. 24, 131-141.

[200] STAUNSTRUP J., Message passing communication versus procedure call communication, Software Practice and Experience, 1982, vol. 12, 223-234.

[201] STOTTS Jr. P. D., A hierarchical graph model of concurrent real-time software systems, Department of Computer Science, University of Maryland, College Park, TR-1669, (Ph. D. Dissertation), 1986.

[202] TAYLOR J. R., Proving correctness for a real-time operating system, Proceedings of the 1974 IFAC-IFIP Workshop on Real-time Programming, Budapest, 1974, 231-246.

[203] TAYLOR R., WILSON P., OCCAM process - oriented language meets demands of distributed programming, Electronics, 1982, vol. 55(23), 89-95.

[204] TRELEAVEN P. C., BROWNBRIDGE D. R., HOPKINS R., Data-driven and demand-driven computer architecture, Computing Survey, 1982, vol. 4(1), 93-143.

[205] TRELEAVEN P. C., GOUVELA LIMA I., Future computers: logic, data-flow, ..., control flow?, Computer, March 1984, 47-58.

[206] TSUINO Y., ANDO M., ARAKI T., TOKURA N., Concurrent C: a programming language for distributed multiprocessor systems, Software Practice and Experience, 1984, vol. 14, 1061-1078.

[207] TURSKI W. M., Design of large programs, Sprawozdanie nr 87, Instytut Informatyki, Uniwersytet Warszawski, 1980.

[208] TURSKI W. M., Metodologia programowania, Wyd. 2, WNT, 1982.

[209] VEGDALL S. R., A survey of proposed architectures for the execution of functional languages, IEEE Transactions on Computers, 1984, vol. C-33, 1050-1071.

[210] WILLIAMSON R., HOROWITZ E., Concurrent communication and synchronization mechanisms, Software Practice and Experience, 1984, vol. 14, 135-151.

[211] WINKOWSKI J., An algebraic approach to concurrency, Lecture Notes in Computer Science, 1979, vol. 74, 523-532.

[212] WINSKEL G., Categories of models for concurrency, Technical Report no 58, Computer Laboratory, University of Cambridge, 1985.

[213] WINSKEL G., Event structures, Lecture Notes in Computer Science, 1987, vol. 255, 325-392.

[214] WIRTH N., Toward a discipline of real-time programming, Communications of ACM, 1977, vol. 20, 577-583.

[215] ZIELONKA W., Proving assertions about parallel programs by means of traces, Prace IPI, PAN, 1980, 424.

PROGRAMMING OF REAL-TIME COMMUNICATING PROCESSES

The subject of the paper is concerned with a concurrent real-time programming. Especially, two topics are considered here:

- the formalization of a programming language semantics, and
- the construction and verification of programs.

The essential feature of the work is taking into account all the aspects of the time during the execution of programs. The formal methods of real-time languages description as well as the methodology of real-time programming are still in an initial stage. The first aim of the work is to propose, on the base of an exemplary language, the model of semantics descriptions. It enables a fulfilment of the second aim: to state formally the notion of a real-time program correctness and to build a proof system for the real-time program correctness.

The starting point of considerations is the definition of a simple real-time programming language, called RTCSP (Real-time CSP). It is a modification of Hoare's CSP (Communicating Sequential Processes) language. RTCSP comprises a typical set of elementary instructions (skip, abort, assignment) and the following structured instructions:

- Dijkstra's alternative and repetitive instructions, and
- a real-time communication instruction.

The last one has the form:

$$
\begin{array}{ll}
\underline{\text{through}} \quad v & \underline{\text{wait}} \\
\quad b_1;\ CM_1 & \longrightarrow\ AC_1 \\
[]\ b_2;\ CM_2 & \longrightarrow\ AC_2 \\
\quad \ldots\ldots\ldots\ldots\ldots & \\
[]\ b_n;\ CM_n & \longrightarrow\ AC_n \\
\underline{\text{later}} \quad AC_{n+1} & \underline{\text{end}}
\end{array}
$$

where v is a constant defining the length of a time-out period, $b_1, \ldots, b_n$ are boolean expressions; $CM_1, \ldots, CM_n$ are input/output commands; $AC_1, \ldots, AC_{n+1}$ are sequences of instructions. Input/output commands have the same form and meaning as in the CSP. A program in the RTCSP consists of a finite number of processes. A sequence of instructions forms the process body; processes can not be nested.

RTCSP programs are assumed to be executed in a real-time environment. The execution of a program carries out as follows. Each component process starts its execution at a given time and performs its consecutive elementary computational steps. The elementary step can not be inter-

rupted and needs a finite period of time for its completion. The length of this period is undetermined and depends on the executive environment. A clock is bound to each process. All clocks refere to the same calendar time, but the clock, bound to the given process, changes its state only at the moment when a consecutive elementary step of this process is terminated. The passage of time is modelled by integers and all processes have the same unit of time.

An informal semantics of the real-time communication instruction, which is one of the most important language constructions, is expressed by the following rules:

1. The boolean expressions are evaluated as the first elementary step. An alternative of the instruction is called open if the respective boolean expression is true.

2. If the set of open alternatives is not empty, then the respective input/output commands are waiting for a communication with another process until the time-out period expires. If such a communication occurs, say via CM_i, then the appropriate sequence AC_i starts to perform, otherwise after the time-out termination the sequence AC_{n+1} starts to perform.

3. If the set of open alternatives is empty, then the computation is aborted.

A formal semantics description of RTCSP resolves itself into a definition of the labelled transition schema:

$$(\Sigma, \Sigma_f, \Lambda, \longrightarrow),$$

where Σ stands for a set of all programs configurations, Σ_f - for a subset of all terminal configurations, Λ - for a set of process interactions, and $\longrightarrow$ - for a transition relation:

$$\longrightarrow \subseteq \Sigma \setminus \Sigma_f \times \Lambda \times \Sigma$$

Informally speaking, the configuration of the program is a triple (ρ, s, c), where ρ is a subprogram, s is a state of program variables, and c is a state of program clocks. A proces interaction λ can be empty if the performed elementary step is executed autonomously by a single process, or represents the so called communication record, if the performed step results in a communication between a pair of synchronized processes. The intuitive meaning of the transition relation $(\rho_1, c_1, s_1) \xrightarrow{\lambda} (\rho_2, c_2, s_2)$ is: executing ρ_1 along one step in states s_1 and c_1 can lead in the result of λ interaction to states s_2 and c_2 with ρ_2 being remainder of ρ_1 still to be executed. The presented approach is an extension of the structured operational one proposed by Plotkin.

The formal semantics is used in the definition of a program correctness. The notions of a partial, weak, strong and total correctness are introduced and analysed. A specification of the program P is written in the classical form $\{\alpha\}\ P\ \{\beta\}$, where α, β are formulae of a first order predicate language. The formulae can compraise program variables and auxillary variables. There are two types of the auxillary variables: clock variables and historical variables. The last ones represent sequences of process interactions.

Hoare's program logics is used to prove program correctness. The notion of proving system schema is introduced. The soundness and relative completness of the schema is proved. The proving procedure follows, due to Owicki, in two phases. In the first one the sequential proofs for composite processes are done and in the second one these sequential proofs are composed into an entire proof of the program. The most crutial point of this procedure is an examination of cooperation conditions for sequential proofs. The settlement of the cooperation conditions and then their application are the most complicated part of the proving procedure. The reason for this complexity arrises from the great number of program configurations which should be considered. This fact is illustrated by the proof of exemplary, nontrivial program.

The essential results of the work resolve themseves into two closely connected elements: firstly, a construction of the formal model for semantics of real-time lenguages, and secondly, a precise definition of the real-time programme correctness and the method of a proof system construction.

ПРОГРАММИРОВАНИЕ СВЯЗЫВАЮЩИХСЯ ПРОЦЕССОВ В РЕАЛЬНОМ ВРЕМЕНИ

Предмет работы связан с согласованным программированием в реальном времени. В особенности рассмотрены две проблемы:

- формализация описания семантики языка программирования,
- конструирование и верификация программ.

Существенной чертой работы является обращение внимания на все аспекты, связанные с истечением времени во время исполнения программ. Как методы формального описания семантики, так и методология программирования в реальном времени находятся в фазе начальных исследований. Первой целью работы является предложение, опирающееся на пример простого языка программирования, модели описания семантики. Это дает возможность реализации второй цели - формального определения правильности программы реального времени и строение системы доказания правильности программ.

Исходной точкой является определение простого языка программирования в реальном времени, называемого RTCSP (Real-time CSP). Он является модификацией языка CSP Hoare'a (Communicating Sequential Processes). RTCSP содержит типичный комплект элементарных операторов (пустой, срыва, подстановки) и следующие структурные операторы:

- суммы и пятли Дийкстры,
- связи в реальном времени.

Последняя имеет форму:

through v wait
b_1; $CM_1 \longrightarrow AC_1$
$\square b_2$; $CM_2 \longrightarrow AC_2$

$\square b_n$; $CM_n \longrightarrow AC_n$
later AC_{n+1} end

где v является определением длины отрезка времени просрочения (time-out); $b_1,\ldots,b_n$ - бодовские выражения; $CM_1,\ldots,CM_n$ - входные/выходные команды; $AC_1,\ldots,AC_{n+1}$ - последовательности операторов. Команды входа/выхода имеют такую же форму и значение как в CSP . Программа в RTCSP состоит из конечного числа процессов, которые не могут быть вложенными.

Предполагается, что программы в RTCSP выполняются в среде реального времени. Выполнение программы проходит следующим образом. Каждый составной процесс начинает действие в определенный момент и выполняет поочередно элементарные расчетные шаги. Расчетный шаг не может быть прекращенным и продолжается конечный отрезок времени. Длина этого отрезка не детерминирована и зависит от свойств исполнительной среды. Все часы относятся к тому же календарному времени, но часы, связанные с данным процессом, меняют свое состояние лишь в момент окончания элементарного

шага расчетов в этом процессе. Истечение времени моделировано целыми числами и все процессы имеют ту же единицу времени.

Неформально семантику оператора связи в реальном времени, которая является самой важной языковой конструкцией, выражают следующие принципы:

I) В первом шагу вычисляется значения бодовских выражений $b_1,\ldots,$ b_n. Сумма, которой отвечает настоящее бодовское выражение, называется открытой.

2) Если множество открытых сумм не является пустым, то отвечающие им команы входа/выхода ждут связи с другим процессом по крайней мере к истечению отрезка времени просрочения. Если такая связь осуществится, скажем, через CM_i, начинается реализация отвечающей последовательности операторов AC_i. В противоположном случае, после истечения отрезка времени просрочения начинает реализацию последовательности операторов AC_{n+1}.

3) Если множество открытых сумм пусто, то вычисление операторов прекращается.

Формальное описание семантики языка выражено посредством определения собтветствующей системы переходов:

$$(\Sigma, \Sigma_f, \Lambda, \longrightarrow)$$

где Σ - множество т.наз. конфигураций программы, Σ_f - подмножество терминальных конфигураций, Λ - множество интеракций процессов и $\longrightarrow$ является отношением изменения конфигурации:

$$\longrightarrow \subseteq \Sigma \setminus \Sigma_f \times \Lambda \times \Sigma.$$

Неформально говоря, конфигурация программы является тройкой (q, s, c), где q - подпрограмма, s - переменное состояние, следовательно c - состояние часов программы. Интеракция процессов λ может быть пустой, если выполняемый элементарный расчетный шаг выполняется автономно через отдельный процесс, или же может представлять т.наз. рекорд связи, если элементарный шаг является результатом связи синхронизированной пары процессов. Интуитивное значение отношения изменения конфигурации $(q_1, s_1, c_1) \xrightarrow{\lambda} (q_2, s_2, c_2)$ следующее: выполнение подпрограммы q_1 в состоянии s_1, c_1 может вести по ходу отдельного расчетного шага, которому сопутствует интеракция λ, к программе q_2, в состоянии s_2, c_2; подпрограммы q_2 составляет "остаток" подпрограммы q_1, которая остается еще для выполнения. Представленный подход составляет расширение структурного операционного подхода Плоткина.

Формальную семантику используют для определения правильности программы. Выделяют частичную, слабую, сильную и полную правильности. Спецификацию программы P записывают в классической форме $\{\alpha\}$ P $\{\beta\}$, где

α, β являются формулами языка предикатов первого порядка. Эти формулы могут содержать программные и вспомогательные переменные. Выделяют два вида вспомогательных переменных: часовые и исторические. Последние представляют последовательности интеракций процессов.

Доказание правильности программ базирует на логике Гоаре. Вводят понятие схемы системы доказания правильности программ и доказывают его непротиворечивость и относительную полноту. Процедура доказания правильности, согласно концепции Овицки, разбивается на две фазы. В первой - образуют секвенционные доказательства для составных процессов программы, во второй - составляют полное доказательство для всей программы. Самым существенным моментом этого действия является испытание согласованности секвенционных доказательств. Установление условий согласования и их применение для конкретного случая составляет наиболее сложную часть доказания правильности программы. Причиной этой сложности является необходимость рассмотрения очень большого числа конфигураций. Этот факт проиллюстрирован доказательством правильности примерной неустойчивой программы. Основные результаты работы сводятся к двум тесно связанным друг с другом элементам: во-первых - конструкции формальной модели для описания семантики языка реального времени, а во-вторых - методам доказательства правильности программ реального времени.

SPIS TREŚCI

CONTENTS

PRACE NAUKOWE CENTRUM OBLICZENIOWEGO
(wydane w latach 1980—1988)

Nr 1,	Studia i Materiały nr 1, *Zagadnienia oprogramowania systemów komputerowego wspomagania dydaktyki*, Wrocław 1980	20,—
Nr 2,	Studia i Materiały nr 2, *Zagadnienia oprogramowania węzła podsieci komunikacyjnej sieci komputerowej MSK*, Wrocław 1983	180,—
Nr 3,	Konferencje nr 1, *Sieci komputerowe. Usługi, protokoły, modele. Część 1*, Wrocław 1985	160,—
Nr 4,	Konferencje nr 2, *Sieci komputerowe. Usługi, protokoły, modele. Część 2*, Wrocław 1986	250,—
Nr 5,	Konferencje nr 3, *Sieci komputerowe. Teoria, technika, zastosowania*, Wrocław 1987	335,—