

Uniwersytet Warszawski

Wydział Matematyki, Informatyki i Mechaniki

Agata Janowska

**Generowanie automatów czasowych dla
systemów czasu rzeczywistego**

Rozprawa doktorska

Promotor rozprawy
doc. dr hab. Wojciech Penczek
Instytut Podstaw Informatyki
Polskiej Akademii Nauk

maj 2007

Oświadczenie autora rozprawy

Oświadczam, że niniejsza rozprawa została napisana przeze mnie samodzielnie.

Data

Podpis autora rozprawy

Oświadczenie promotora rozprawy

Niniejszym oświadczam, że rozprawa jest gotowa do oceny przez recenzentów.

Data

Podpis promotora rozprawy

Streszczenie

Komercyjne systemy czasu rzeczywistego są projektowane głównie w językach wysokiego poziomu, wspomaganym przez narzędzia ułatwiające ich testowanie, symulację, czy generowanie kodu wykonywalnego, ale rzadko wspierające ich formalną weryfikację. Przykładami języków wysokiego poziomu, które pozwalają na wyrażanie zależności czasowych są Estelle i SDL, języki opisu protokołów komunikacyjnych i systemów rozproszonych. Również popularne języki programowania i modelowania są obecnie wzbogacane o elementy umożliwiające definiowanie warunków czasowych. Z drugiej strony duża część badań w tej dziedzinie dotyczy opracowania i rozwijania metod weryfikacji dla modeli takich jak automaty czasowe i sieci Petriego z czasem. Intensywny rozwój przeżywiają zwłaszcza metody weryfikacji modelowej automatów czasowych.

Aby móc zastosować metody i narzędzia weryfikacji modelowej automatów czasowych do sprawdzania poprawności systemów czasowych, opisanych w językach wysokiego poziomu, proponujemy dokonać tłumaczenia z tych języków do automatów czasowych. W tym celu wprowadzamy reprezentację pośrednią, tak zwany język bazowy. Generowanie automatów odbywa się w dwóch krokach. Najpierw opis systemu w danym języku jest tłumaczony do reprezentacji pośredniej, a z niej do automatów czasowych. Na pierwszy krok powinny się składać przekształcenia wyłącznie syntaktyczne. Dlatego język bazowy musi być wystarczająco bogaty, aby umożliwić wyrażanie wszystkich istotnych aspektów komunikacji i synchronizacji systemów współbieżnych z ograniczeniami czasowymi.

W rozprawie zostały przedstawione dwie metody generowania automatów czasowych dla systemu czasowego opisanego w języku bazowym. Pierwsza metoda konstruuje jeden automat czasowy (tak zwany automat globalny) dla całego systemu. Niektóre metody i narzędzia weryfikacji modelowej działają dużo bardziej efektywnie, jeżeli zamiast jednego (najczęściej dużego) automatu mogą operować na zbiorze mniejszych automatów, opisujących poszczególne składowe systemy. Dlatego opracowano drugą metodę budującą zbiór automatów czasowych dla systemu czasowego, w którym poszczególne automaty odpowiadają składowym systemom.

Istotnym problemem w praktycznym wykorzystaniu metod weryfikacji modelowej jest wykładnicza eksplozja liczby stanów modelu. Dlatego, podczas generowania automatów czasowych wykorzystane są metody redukcji modeli. Praca przedstawia zastosowanie analizy statycznej do uzyskania abstrakcyjnego modelu systemu. Proponowana metoda jest oparta na technice cięcia programów. W przedstawionej metodzie abstrakcja systemu zachowuje prawdziwość formuł logiki temporalnej CTL^*_X , to jest CTL^* bez operatora następnego kroku.

Słowa kluczowe: systemy czasowe, automaty czasowe, weryfikacja modelowa, analiza statyczna, metoda cięcia programów

Klasyfikacja tematyczna pracy według ACM Computing Classification System: D.1.3, D.2.1, D.2.4, F.3.1.

Abstract

Formal methods used for a practical design of timed systems are based on the well known specification languages like Estelle, SDL, or Lotos. These formalisms are supported by various development facilities including editing, code and test generating, debugging, or simulating. But, they lack automated verification tools. On the other hand, model checking seems to be one of the leading and the most promising verification techniques for hardware and software. Unfortunately, there is a gap between the above high level specification languages and low level input formalisms (mostly networks of timed automata) for model checkers.

The aim of this thesis is to cover this gap, i.e., to provide a method of generating timed automata from high level specification languages. The main idea is to use an intermediate representation. The process of translation involves two steps — a translation to the intermediate language and from the intermediate language to timed automata. The first step should be merely syntactical, because the intermediate language is rich enough to describe all important aspects of communication and concurrency.

The thesis provides two methods of generating timed automata. The first one constructs one automaton for the whole system whereas the other one — a network of automata, one for each component of the specified system. A description in terms of local components provides a more compact representation and can be exploited by symbolic model checking methods that take advantage of locality to alleviate the state explosion problem.

The thesis proposes also how static analysis can be used to extract an abstract model of a timed system. Our method uses techniques of program slicing to examine syntax of a system modeled in the intermediate language. The method is property driven. The abstraction is exact with respect to all properties expressed in the temporal logic CTL^*_X , i.e., CTL^* without the next step operator. The reduction is performed at the very beginning of the verification process and this makes it beneficial and efficient in handling the state explosion problem.

Keywords: timed systems, timed automata, model checking, static analysis, program slicing

ACM Computing Classification System: D.1.3, D.2.1, D.2.4, F.3.1.

Pamięci mojej Mamy

Podziękowania

Pragnę serdecznie podziękować doc. dr hab. Wojciechowi Penczkowi za wiele cennych uwag i komentarzy przy pisaniu niniejszej rozprawy, a także za dobrą atmosferę podczas całej współpracy. Podziękowania składam również prof. Piotrowi Dembińskiemu za umożliwienie mi pracy w zespole naukowców skupionych w Instytucie Podstaw Informatyki PAN wokół projektu VerICS. Chciałabym podziękować im wszystkim z osobna za wszelką pomoc jakiej mi udzielili w trakcie naszej współpracy, a także za jej miłą atmosferę. Dziękuję więc Agacie Półroli, Bożenie Woźniej, Gizeli Jakubowskiej, Magdzie Kasprzak, Mirkowi Kurkowskiemu, Wojtkowi Nabiałkowi, Arturowi Niewiadomskiemu, Maciejowi Orzechowskiemu, Maciejowi Szreterowi i Andrzejowi Zbrzezemu. Ostatnie, lecz niemniej ważne podziękowania składam mojej Rodzinie, a przede wszystkim Mężowi.

Spis treści

1	Wstęp	3
2	Automaty czasowe	9
2.1	Automaty czasowe	9
2.1.1	Etykietowany system tranzycyjny	9
2.1.2	Składnia	10
2.1.3	Semantyka	11
2.1.4	Złożenie automatów czasowych	12
2.1.5	Model	13
2.1.6	Przykłady	13
2.2	Weryfikacja automatów czasowych	16
2.2.1	Logiki temporalne	16
2.2.2	Weryfikacja modelowa	18
2.2.3	Relacje równoważności systemów	19
3	Język bazowy	21
3.1	Składnia abstrakcyjna	21
3.1.1	Definicje i oznaczenia	23
3.2	Semantyka	26
3.3	Poprawność programu	31
3.4	Przykłady	32
3.4.1	Producent i konsument	32
3.4.2	Protokół Fischera wzajemnego wykluczania	34
3.4.3	Protokół zmieniającego się bitu	35
3.5	Podsumowanie rozdziału	38
4	Generowanie automatów czasowych dla języka bazowego	39
4.1	Zegary	39
4.2	Globalny automat czasowy	41
4.2.1	Definicja automatu globalnego	41
4.2.2	Funkcja wartościująca dla automatu globalnego	42
4.2.3	Przykład	42
4.2.4	Poprawność	43
4.3	Zbiór automatów czasowych	49

4.3.1	Ograniczenia	50
4.3.2	Niezależne zmienne globalne i bufory	51
4.3.3	Etykiety w automatach składowych	51
4.3.4	Automaty składowe dla procesów	52
4.3.5	Automaty składowe dla zmiennych globalnych i buforów	53
4.3.6	Funkcje wartościujące dla automatów składowych	54
4.3.7	Przykład	55
4.3.8	Poprawność	57
4.4	Redukcja liczby zegarów w automatach czasowych	63
4.5	Podsumowanie rozdziału	64
5	Redukcja przestrzeni stanów programu metodą cięcia	69
5.1	Relacje zależności	70
5.1.1	Zależność danych	70
5.1.2	Zależność przepływu sterowania	71
5.1.3	Zależność czasowa	73
5.1.4	Zależność od synchronizacji	81
5.2	Zredukowany program	81
5.2.1	Kryterium cięcia	82
5.2.2	Istotne operacje	82
5.2.3	Istotne stany kontrolne	84
5.2.4	Zredukowany program	86
5.3	Poprawność	91
5.4	Wyniki eksperymentalne	105
5.5	Podsumowanie rozdziału	111
6	Podsumowanie	113

Rozdział 1

Wstęp

Systemy czasu rzeczywistego, nazywane też systemami czasowymi, to systemy, których działanie jest uzależnione od upływu czasu. Weryfikacja systemów czasowych jest ważnym zagadnieniem współczesnej informatyki ze względu na to, że coraz więcej takich systemów ma wpływ na nasze bezpieczeństwo — na przykład systemy sterowania ruchem drogowym i kolejowym, czy oprogramowanie sprzętu medycznego.

Modelowanie systemów współbieżnych, w tym także czasowych, opiera się głównie na trzech podejściach: algebrze procesów [Mil80], komunikujących się automatach [Zie87] i sieciach Petriego [Rei85]. Języki wyższego rzędu opisujące systemy czasowe bazują głównie na powyższych modelach, co umożliwia odpowiednie adaptowanie metod weryfikacji. Duża część badań w tej dziedzinie dotyczy opracowania i rozwijania metod weryfikacji dla modeli systemów współbieżnych z czasem takich jak automaty czasowe [AD90] i sieci Petriego z czasem [Wal83]. Intensywny rozwój przeżywają zwłaszcza metody weryfikacji modelowej (ang. *model checking*) automatów czasowych, czego dowodem jest powstanie i stałe udoskonalanie narzędzi takich jak Kronos [DOTY95], UppAal [PL00], HyTech [HHWT97] a także VerICS [DJJ⁺03a], rozwijany w Instytucie Podstaw Informatyki PAN.

Z drugiej strony, komercyjne systemy czasu rzeczywistego są projektowane głównie w językach wyższego poziomu, wspomaganym przez narzędzia ułatwiające ich testowanie, symulację, czy generowanie kodu wykonywalnego, ale rzadko wspierające ich formalną weryfikację. Przykładami języków wysokiego poziomu, które pozwalają na wyrażanie zależności czasowych są Estelle [ISO97] i SDL [MT01], języki opisu protokołów komunikacyjnych i systemów rozproszonych. Również powszechnie stosowane języki modelowania (takie jak UML), a nawet popularne języki programowania są obecnie wzbogacane o elementy umożliwiające definiowanie warunków czasowych [BG00, DJPV02].

Aby móc zastosować metody i narzędzia weryfikacji modelowej automatów czasowych do sprawdzania poprawności systemów czasowych, opisanych w językach wysokiego poziomu, proponujemy dokonać tłumaczenia z tych języków do automatów czasowych.

Istotnym problemem w praktycznym wykorzystaniu metod weryfikacji modelowej jest wykładnicza eksplozja liczby stanów modelu. Dlatego, podczas generowania automatów czasowych wykorzystane są metody redukcji modeli.

Cele rozprawy

Podstawowym zamierzeniem rozprawy jest dostarczenie metod i narzędzi generowania automatów czasowych na podstawie opisu systemów w językach wysokiego poziomu. W tym celu wprowadzamy reprezentację pośrednią, tak zwany język bazowy. Generowanie automatów odbywa się w dwóch krokach. Najpierw opis systemu w danym języku jest tłumaczony do reprezentacji pośredniej, a z niej do automatów czasowych. Na pierwszy krok powinny się składać przekształcenia wyłącznie syntaktyczne. Dlatego język bazowy musi być wystarczająco bogaty, aby umożliwić wyrażanie wszystkich istotnych aspektów komunikacji i synchronizacji systemów współbieżnych z ograniczeniami czasowymi.

Dla systemu opisanego w języku bazowym opracowano metody konstruowania automatów czasowych, a także metodę redukcji, dzięki której budowane automaty mają mniejsze rozmiary (w sensie liczby zegarów, stanów i tranzycji). Język bazowy i obie metody zostaną krótko omówione w kolejnych punktach, które odpowiadają głównym rozdziałom rozprawy.

Język bazowy

System czasu rzeczywistego opisujemy w języku bazowym jako zbiór procesów, zmiennych całkowitych i buforów komunikacyjnych. Każdy proces jest przedstawiony w sposób automatowo-zorientowany. Procesy wykonują pewne akcje (synchroniczne) wspólnie, a inne (lokalne) na zasadzie przeplotu. Język dostarcza także mechanizmów do komunikacji asynchronicznej — poprzez wymianę wiadomości za pomocą buforów. W języku nie występują jawnie zmienne reprezentujące czas, jednak z każdą akcją może być związane tak zwane dozwolone opóźnienie, które określa czas w jakim akcja może być wykonana.

W języku można również definiować zmienne zdaniowe, które mogą być użyte do budowy formuł logicznych różnych logik temporalnych opisujących własności systemu typowe dla systemów współbieżnych, takie jak własności bezpieczeństwa i żywotności. Na przykład można zdefiniować zmienną, która ma wartość *prawda* zawsze wtedy, gdy proces znajduje się w określonym stanie lub kiedy wartość zmiennej systemu spełnia określony warunek.

W rozprawie przedstawiono składnię i semantykę operacyjną języka bazowego, a także kilka przykładów zastosowania języka do opisu systemów z czasem.

Generowanie automatów czasowych dla języka bazowego

Automaty czasowe zostały zdefiniowane w pracy [AD90] jako skończenie stanowe automaty Büchiego rozszerzone o zmienne rzeczywiste reprezentujące zegary. Stan w automacie czasowym jest zwyczajowo nazywany *lokacją*, aby odróżnić go od stanu w jakim może znajdować się automat, na który oprócz lokacji składa się także wartościowanie zegarów. Najczęściej używana definicja automatów (także w tej rozprawie) pochodzi z pracy [HNSY94b], gdzie zamiast warunków akceptujących Büchiego zostały wprowadzone niezmienniki lokacji.

Sprowadzanie jednego formalizmu do innego jest powszechnie stosowanym rozwiązaniem. Tłumaczenie do automatów czasowych zostało wykonane dla algebry procesów ATP [NSY92], języka ET-LOTOS [DOY94] i języka Esterel wzbogaconego o zależności czasowe [BCP⁺01]. Wspólną cechą tych modeli jest brak mechanizmów komunikacji asynchronicznej. Współdziałanie między procesami opiera się jedynie na synchronicznym wykonywaniu pewnych akcji. W tym przypadku tłumaczenie polega na zbudowaniu osobnego automatu

czasowego dla każdego procesu i naturalnym przeniesieniu metody synchronizacji (w automatach czasowych przejścia o tej samej etykietce wykonywane są wspólnie). Pewne trudności pojawiają się, gdy język opisu systemu czasowego dostarcza mechanizmów komunikacji asynchronicznej, na przykład poprzez wymianę komunikatów, ponieważ takie bezpośrednie tłumaczenie nie jest wtedy możliwe.

W rozprawie zostały przedstawione dwie metody generowania automatów czasowych dla systemu czasowego opisanego w języku bazowym. Pierwsza metoda konstruuje jeden automat czasowy (tak zwany automat globalny) dla całego systemu. Lokacja globalnego automatu czasowego odpowiada konfiguracji systemu określonej przez stany wszystkich jego składowych (procesów, zmiennych i buforów komunikacyjnych). Intuicyjnie, w globalnym automacie czasowym istnieje przejście z jednej lokacji do drugiej, jeżeli system będąc w konfiguracji odpowiadającej pierwszej lokacji może wykonać akcję, w wyniku której system znajdzie się w konfiguracji odpowiadającej drugiej lokacji. Ograniczenia czasowe wykonywanych akcji są modelowane przez odpowiednie warunki na zegarach automatu.

Niektóre metody i narzędzia weryfikacji modelowej działają dużo bardziej efektywnie, jeżeli zamiast jednego (najczęściej dużego) automatu mogą operować na zbiorze mniejszych automatów, opisujących poszczególne składowe systemu. Oczywiście, w czasie weryfikacji analizowane są przebiegi automatu produktowego automatów składowych. Dlatego opracowano drugą metodę budującą zbiór automatów czasowych dla systemu czasowego, w którym poszczególne automaty odpowiadają składowym systemu. Lokacje składowych automatów czasowych odpowiadają konfiguracjom poszczególnych elementów systemu.

W rozprawie zostało wykazane, że dana własność wyrażona jako formuła logiki CTL* jest prawdziwa dla systemu opisanego w języku bazowym wtedy i tylko wtedy, gdy jest prawdziwa dla automatu globalnego zbudowanego dla tego systemu. Analogiczne twierdzenie jest prawdziwe dla zbioru automatów czasowych.

W konstrukcji zwrócono szczególną uwagę na to, aby liczba zegarów w generowanych automatach była jak najmniejsza, ponieważ złożoność problemu weryfikacji modelowej automatów czasowych jest wykładnicza względem liczby użytych zegarów [AD94]. Rozważamy również metody generowania automatów dla pewnych podklas systemów języka bazowego, takich jak systemy nie wykorzystujące mechanizmów komunikacji asynchronicznej.

Przedstawione metody generowania automatów czasowych zostały opublikowane w materiałach konferencji CS&P'02 [DJJ02] oraz zaimplementowane jako moduły systemu weryfikacyjnego VerICS [DJJ+03a, DJJ+03b] dostępnego pod adresem:

<http://verics.ipipan.waw.pl>.

Redukcja przestrzeni stanów metodą cięcia

Głównym problemem w automatycznej weryfikacji modelowej jest wykładnicza eksplozja liczby stanów modelu [Val98], która ogranicza rozmiary systemów, z którymi można się zmierzyć w procesie weryfikacji. Z tego powodu metody ograniczania eksplozji stanów mają duże znaczenie praktyczne. Dla systemów bez czasu zostały opracowane różne metody, które redukują rozmiary przestrzeni stanów użytej w procesie weryfikacji. Metody te są dostosowane do języków specyfikacji, w których wyrażane są weryfikowane własności, takich jak logiki temporalne. Do najważniejszych z nich należą metody redukcji częściowo-

porządkowych [Val89, Pel98, GKPP99], abstrakcje [DGG94], metody wykorzystujące symetrię [EJ93], czy metody symboliczne [Bry86, McM93, BCC⁺99].

Niektóre z powyższych metod zostały uogólnione dla systemów z czasem. Po raz pierwszy metody redukcji częściowo-porządkowych dla systemów modelowanych przez sieci Petriego z czasem stosuje Yoneda [YSSC93]. Kolejne próby dotyczą automatów czasowych [Pag96, DGKK98]. Z kolei metodę abstrakcji dla systemów czasowych przedstawiono między innymi w pracy [DT98].

Odmiennym podejściem jest zastosowanie metod redukcji nie na modelu systemu, ale na jego reprezentacji. Metody takie określa się mianem *analizy statycznej*. Jedyną znaną próbę zastosowania statycznej redukcji częściowo-porządkowej stanowi praca [KLM⁺98], w której redukcji poddano język SDL na etapie tłumaczenia do języka wejściowego weryfikatora COSPAN [AK96].

Dla systemów czasowych jest stosunkowo niewiele opracowań na ten temat. W pracy [BGO02] zdefiniowano pojęcie *wplywu* (ang. *influence*, dualne do pojęcia niezależności) dla systemów czasowych modelowanych przez automaty czasowe rozszerzone o interfejsy. Pozostałymi technikami analizy statycznej dla systemów z czasem są: metoda oparta na eliminacji nieaktywnych zegarów [DY96] i bardziej ogólna metoda obliczania istotnych warunków na zegarach [BBFL03] w automatach czasowych.

W rozprawie przedstawiono metodę konstrukcji abstrakcyjnego modelu opartą na statycznej analizie opisu systemu, a konkretnie na metodzie cięcia (ang. *slicing*). Metoda ta zaproponowana przez Weisera [Wei84] polega na wykorzystaniu statycznej analizy opisu systemu do eliminacji jego nieistotnych fragmentów, czyli w naszym przypadku takich, które nie mają wpływu na weryfikowane własności. Pozwala to na uproszczenie opisu, co z kolei powoduje zmniejszenie przestrzeni stanów potrzebnej do jego weryfikacji. Zaletą metody jest to, że pełna przestrzeń stanów nie musi być generowana, ponieważ można ją skonstruować już dla zredukowanego opisu systemu.

Pierwotnie metoda cięcia była stosowana do upraszczania procesów śledzenia (ang. *debugging*) i symulacji programów. Była także wykorzystywana do redukcji przestrzeni stanów w weryfikacji modelowej systemów bez czasu. Analiza systemów czasowych różni się od analizy systemów bezczasowych, ponieważ ograniczenia nałożone na czas wykonywania akcji systemu wprowadzają nowego rodzaju zależności.

Metoda cięcia zależy od danej formuły logicznej wyrażającej własność, której prawdziwość chcemy sprawdzić, a dokładniej — od zbioru zmiennych zdaniowych, które wchodzi w skład formuły. Punktem wyjściowym jest ustalenie tak zwanego *kryterium cięcia*, czyli zbiorów akcji i stanów, od których bezpośrednio zależą wartości zmiennych zdaniowych z formuły. Następnie, algorytm redukcji bada rekurencyjnie zależności między akcjami i stanami poszczególnych procesów systemu zaczynając od kryterium cięcia.

W systemie czasowym występują cztery rodzaje zależności. Zależność danych i zależność przepływu sterowania są klasycznymi pojęciami metody cięcia [Tip95]. Zależność od synchronizacji jest charakterystyczna dla systemów współbieżnych, które mogą wykonywać pewne akcje synchronicznie. Natomiast pojęcie zależności czasowej jest nowym rodzajem zależności występującym tylko w systemach czasowych, które zostało po raz pierwszy zdefiniowane przez autorkę rozprawy.

W weryfikacji modelowej własności programów są opisywane między innymi przez formuły logik temporalnych będących podzbiorami logiki CTL* (LTL, CTL) lub TCTL (czasowe rozszerzenie CTL_{-X}). Ze względu na charakter proponowanych redukcji (między

innymi eliminację pewnych akcji) w naszych rozważaniach ograniczamy się do podlogik nie zawierających operatora następnego kroku (X). Zachowanie odpowiednich klas własności zapewniamy przez wykazanie odpowiedniej relacji między modelem konkretnym a abstrakcyjnym. Dla logiki CTL^*_X jest to relacja bisymulacji z powtórzeniami (ang. *stuttering bisimulation*).

Artykuły na temat opracowanej przez autorkę rozprawy metody redukcji systemów czasowych zostały opublikowane w materiałach konferencji CS&P'03 [JJ03] i CS&P'06 [JP06b], a także w czasopiśmie *Fundamenta Informaticae* [JJ04]. Metoda cięcia dla języka bazowego została opracowana przez autorkę rozprawy. Współautor wymienionych prac, Paweł Janowski, współuczestniczył w implementacji metody i przeprowadzeniu eksperymentów.

Struktura pracy

Rozdział 2 zawiera wprowadzenie do problematyki rozprawy, w tym podstawowe definicje i pojęcia dotyczące automatów czasowych, a także krótkie omówienie sposobu ich weryfikacji.

W rozdziale 3 przedstawiono składnię i semantykę języka bazowego oraz definicje pojęć dotyczących języka, używanych w dalszych rozdziałach. Pokazano także, w jaki sposób można wyrażać własności systemu opisanego w języku bazowym. Rozdział zawiera również kilka przykładowych specyfikacji systemów czasowych.

Metody tłumaczenie języka bazowego do automatów czasowych (automatu globalnego i zbioru automatów składowych) są omówione i zilustrowane na przykładach w rozdziale 4. Pokazano w nim także, że przedstawione tłumaczenie zachowuje prawdziwość formuł logiki CTL^* . Rozdział zawiera również opis sposobu redukcji liczby zegarów w generowanych automatach.

Rozdział 5 opisuje redukcję przestrzeni stanów modelu za pomocą metody cięcia. Zdefiniowano w nim relacje zależności dla programów w języku bazowym, algorytmy obliczania istotnych operacji i istotnych stanów kontrolnych oraz pokazano jak z istotnych elementów skonstruować zredukowany program. Ważną część rozdziału stanowi wykazanie, że przedstawiona metoda redukcji zachowuje prawdziwość formuł logiki CTL^*_X . Ostatnia część rozdziału zawiera opis wyników eksperymentalnych.

Na końcu pracy umieszczono indeks pojęć oraz spis oznaczeń i symboli używanych w rozprawie.

Rozdział 2

Automaty czasowe

Rozdział ten stanowi wprowadzenie do dalszej części rozprawy. Przedstawiamy w nim podstawowe pojęcia związane z automatami czasowymi.

2.1 Automaty czasowe

2.1.1 Etykietowany system tranzycyjny

Definicja 2.1 Etykietowanym systemem tranzycyjnym *nazywany krotkę*

$$TS = (S, s^0, \Lambda, \longrightarrow)$$

gdzie:

- S jest zbiorem stanów,
- $s^0 \in S$ jest stanem początkowym,
- Λ jest zbiorem etykiet,
- $\longrightarrow \subseteq S \times \Lambda \times S$ jest etykietowaną relacją przejścia.

Elementy zbioru \longrightarrow nazywamy *przejściami* lub *tranzycjami*. Jeżeli $(s, \lambda, s') \in \longrightarrow$, to tranzycję o etykiecie λ ze stanu s do stanu s' oznaczamy przez $s \xrightarrow{\lambda} s'$, a stan s' nazywamy *następnikiem* stanu s . *Ścieżką* w TS ze stanu $s \in S$ jest nieskończony ciąg

$$s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} \dots$$

gdzie $s_i \in S$ i $\lambda_i \in \Lambda$ dla każdego $i \geq 0$ oraz $s_0 = s$. *Wykonaniem* lub *przebiegiem* systemu TS jest każda ścieżka ze stanu początkowego s^0 . Mówimy, że stan $s \in S$ jest *osiągalny* ze stanu $s' \in S$, jeżeli istnieje ścieżka $s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{k-1}} s_k \xrightarrow{\lambda_k} \dots$ w TS taka, że $s_0 = s'$ i $s_k = s$.

Niech PV będzie ustalonym zbiorem zmiennych zdaniowych i niech $\mathcal{V} : S \rightarrow 2^{PV}$ będzie funkcją *wartościującą*, która każdemu stanowi przypisuje podzbiór zbioru PV . Intuicyjnie, zbiór $\mathcal{V}(s)$ jest zbiorem zmiennych zdaniowych prawdziwych w stanie s . Parę $M = (TS, \mathcal{V})$ nazywamy *modelem* (z powodów historycznych nazywana jest także *strukturą Kripkego*).

2.1.2 Składnia

Automaty czasowe zostały zdefiniowane w pracy [AD90] jako skończenie stanowe automaty Büchiego rozszerzone o zmienne rzeczywiste reprezentujące zegary. Najczęściej używana definicja automatów (także w tej rozprawie) pochodzi z pracy [HNSY94b], gdzie zamiast warunków akceptujących Büchiego zostały wprowadzone niezmienniki lokacji.

Niech \mathbb{Z} oznacza zbiór liczb całkowitych, \mathbb{N} — zbiór liczb naturalnych (nieujemnych liczb całkowitych), a \mathbb{R}_+ — zbiór nieujemnych liczb rzeczywistych.

Zegary

Niech X będzie skończonym zbiorem zmiennych zwanych *zegarami*. Zbiór $\Psi(X)$ wszystkich *ograniczeń zegarów* (lub *warunków na zegarach*) ze zbioru X definiujemy indukcyjnie w następujący sposób:

$$\psi ::= true \mid x \sim c \mid x - y \sim c \mid \psi \wedge \psi$$

gdzie $x, y \in X, c \in \mathbb{N}$ oraz $\sim \in \{\leq, <, >, \geq\}$. O zbiorze $\Psi(X)$ powiemy, że *nie zawiera różnic zegarów*, jeżeli można go opisać gramatyką w postaci:

$$\psi ::= true \mid x \sim c \mid \psi \wedge \psi$$

Wartościowanie zegarów

Funkcję całkowitą $\tau : X \rightarrow \mathbb{R}_+$, która każdemu zegarowi przypisuje nieujemną wartość, nazywamy *wartościowaniem zegarów*. Przez $\tau \models \psi$ będziemy oznaczać, że wartościowanie $\tau \in \mathbb{R}_+^X$ spełnia ograniczenie $\psi \in \Psi(X)$. Spełnialność $\tau \models \psi$ definiujemy indukcyjnie:

- $\tau \models true$,
- $\tau \models x \sim c$ wtedy i tylko wtedy, gdy $\tau(x) \sim c$,
- $\tau \models x - y \sim c$ wtedy i tylko wtedy, gdy $\tau(x) - \tau(y) \sim c$,
- $\tau \models \psi_1 \wedge \psi_2$ wtedy i tylko wtedy, gdy $\tau \models \psi_1$ i $\tau \models \psi_2$.

W przyjętym modelu zakładamy, że czas płynie tak samo dla wszystkich zegarów. Dla $\delta \in \mathbb{R}_+$, $\tau + \delta$ oznacza wartościowanie zegarów τ' takie, że $\tau'(x) = \tau(x) + \delta$ dla każdego $x \in X$. Na zbiorze zegarów definiujemy operację *zerowania* zegarów z pewnego podzbioru. Dla $Y \subseteq X$, niech $\tau[Y := 0]$, oznacza wartościowanie zegarów τ' takie, że $\tau'(x) = 0$ dla $x \in Y$ i $\tau'(x) = \tau(x)$ dla $x \in X/Y$. Przez τ^0 oznaczamy *początkowe wartościowanie zegarów* takie, że $\tau^0(x) = 0$ dla każdego $x \in X$.

Automat czasowy

Definicja 2.2 Automatem czasowym nazywamy krotkę $TA = (\Sigma, L, l^0, X, E, \mathcal{I})$, gdzie:

- Σ jest skończonym zbiorem etykiet,
- L jest skończonym zbiorem lokacji,

- $l^0 \in L$ jest lokacją początkową,
- X jest skończonym zbiorem zegarów,
- $E \subseteq L \times \Sigma \times \Psi(X) \times 2^X \times L$ jest relacją przejścia,
- $\mathcal{I} : L \rightarrow \Psi(X)$ jest niezmiennikiem lokacji.

Niezmiennik $\mathcal{I}(l)$ określa warunek jaki muszą spełniać wartości zegarów, aby automat mógł znajdować się w lokacji l . Element zbioru E zapisany jako $l \xrightarrow{\sigma, \psi, Y} l'$ reprezentuje tranzycję z lokacji l (źródłowej) do lokacji l' (docelowej) o etykiecie σ , gdzie ψ określa warunek umożliwienia tranzycji, czyli warunek jaki muszą spełniać wartości zegarów, aby można było wykonać tę tranzycję, a $Y \subseteq X$ jest zbiorem zegarów do wyzerowania w trakcie wykonywania tej tranzycji.

2.1.3 Semantyka

Definicja 2.3 *Semantyką automatu czasowego $TA = (\Sigma, L, l^0, X, E, \mathcal{I})$ jest etykietowany system tranzycyjny $TS_a = (S_a, s_a^0, \Lambda_a, \rightarrow_a)$, gdzie:*

- $S_a = \{(l, \tau) \mid l \in L \wedge \tau \in \mathbb{R}_+^X \wedge \tau \models \mathcal{I}(l)\}$ jest zbiorem stanów,
- $s_a^0 = (l^0, \tau^0) \in S_a$ jest stanem początkowym,
- $\Lambda_a = \Sigma \cup \mathbb{R}_+$ jest zbiorem etykiet,
- $\rightarrow_a \subseteq S_a \times \Lambda_a \times S_a$ jest relacją przejścia, czyli najmniejszą relacją spełniającą warunki:
 - $(l, \tau) \xrightarrow{\sigma} (l', \tau')$ wtedy i tylko wtedy, gdy istnieje tranzycja $l \xrightarrow{\sigma, \psi, Y} l' \in E$ taka, że $\tau \models \psi$, $\tau' = \tau[Y := 0]$ i $\tau' \models \mathcal{I}(l')$,
 - $(l, \tau) \xrightarrow{\delta} (l', \tau')$ wtedy i tylko wtedy, gdy $\delta \geq 0$, $l = l'$, $\tau' = \tau + \delta$ i $\tau' \models \mathcal{I}(l)$.

Stanem automatu czasowego jest para (l, τ) , gdzie $l \in L$ jest lokacją, a τ jest wartościowaniem zegarów spełniającym niezmiennik lokacji l . Zauważmy, że może być nieprzeliczalnie wiele stanów automatu czasowego. Ze stanu (l, τ) automat może przejść do jednego ze stanów:

- (l', τ') (następnik akcyjny) wykonując tranzycję $l \xrightarrow{\sigma, \psi, Y} l' \in E$ (przejście akcyjne), jeżeli jest ona możliwa do wykonania, czyli wartościowanie zegarów τ spełnia ograniczenie ψ a wartościowanie $\tau' = \tau[Y := 0]$ spełnia ograniczenie wyrażone przez niezmiennik lokacji l' ,
- $(l, \tau + \delta)$ (następnik czasowy), wykonując przejście reprezentujące upływ czasu (przejście czasowe) — pod warunkiem, że ograniczenia wyrażone przez niezmiennik związany z lokacją l pozostaną spełnione dla wartościowania $\tau + \delta$.

Przebiegi

s -przebiegiem automatu czasowego \mathcal{TA} jest nieskończony ciąg stanów:

$$s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} \dots$$

gdzie $s_i \in S_a$ i $\lambda_i \in \Sigma \cup \mathbb{R}_+$ dla każdego $i \geq 0$ oraz $s_0 = s$.

Przebieg automatu czasowego nazywamy *postępującym*, jeżeli suma $\sum_{\{i \in \mathbb{N} \mid \lambda_i \in \mathbb{R}_+\}} \lambda_i$ jest nieskończona. Przebieg, który nie jest postępujący jest nazywany *przebiegiem Zenona* od imienia greckiego filozofa z IV w p.n.e. Automat czasowy \mathcal{TA} jest *postępujący*, jeżeli wszystkie jego s^0 -przebiegi są postępujące. Pojęcie postępowości systemów czasowych jest analizowane między innymi w pracach [AL91, GSSAL94, AH97].

2.1.4 Złożenie automatów czasowych

Systemy czasu rzeczywistego często mają budowę modułarną, co znaczy, że można je opisać jako zbiór współpracujących ze sobą składowych. Formalizm automatów czasowych również daje taką możliwość. Elementy systemu można opisać jako oddzielne automaty czasowe. Cały system może być wtedy reprezentowany przez jeden automat czasowy powstały ze złożenia automatów opisujących poszczególne elementy. Automat taki nazywamy *automatem produktowym*.

Złożenie polega na jednoczesnym (synchronicznym) wykonaniu tranzycji o takich samych etykietach przez wiele automatów. Pozostałe przejścia są niezależne i są wykonywane na zasadzie przepłotu. Przyjmujemy, że tranzycję synchroniczną muszą wykonać wszystkie automaty czasowe, które mają etykietę tej tranzycji w swoim zbiorze etykiet (tak zwana *multisynchronizacja*).

Niech $\mathcal{TA}_i = (\Sigma_i, L_i, l_i^0, X_i, E_i, \mathcal{I}_i)$ będzie automatem czasowym dla $1 \leq i \leq n$ oraz niech $\Sigma(\sigma) = \{1 \leq i \leq n \mid \sigma \in \Sigma_i\}$ oznacza zbiór numerów automatów czasowych zawierających etykietę σ .

Definicja 2.4 Niech $\mathcal{TA}_i = (\Sigma_i, L_i, l_i^0, X_i, E_i, \mathcal{I}_i)$ będzie automatem czasowym dla $1 \leq i \leq n$. Produktem n automatów czasowych (oznaczanym przez $\mathcal{TA}_1 \parallel \mathcal{TA}_2 \parallel \dots \parallel \mathcal{TA}_n$) jest automat czasowy $\mathcal{TA} = (\Sigma, L, l^0, X, E, \mathcal{I})$, gdzie:

- $\Sigma = \bigcup_{i=1}^n \Sigma_i$,
- $L = \prod_{i=1}^n L_i$,
- $l^0 = (l_1^0, \dots, l_n^0)$,
- $X = \bigcup_{i=1}^n X_i$,
- *tranzycja* $((l_1, \dots, l_n), \sigma, \bigwedge_{i \in \Sigma(\sigma)} \psi_i, \bigcup_{i \in \Sigma(\sigma)} Y_i, (q'_1, \dots, q'_n)) \in E$ wtedy i tylko wtedy, gdy $(l_i, \sigma_i, \psi_i, Y_i, l'_i) \in E_i$ dla każdego $i \in \Sigma(\sigma)$ oraz $l'_i = l_i$ dla $i \in \{1, \dots, n\} \setminus \Sigma(\sigma)$,
- $\mathcal{I}(l_1, \dots, l_n) = \bigwedge_{i=1}^n \mathcal{I}_i(l_i)$.

2.1.5 Model

Niech PV będzie zbiorem zmiennych zdaniowych. Definiujemy funkcję wartościującą $\mathcal{V}_{TA} : L \rightarrow 2^{PV}$, która każdej lokacji automatu przypisuje podzbiór zbioru PV . Intuicyjnie, jeżeli $p \in \mathcal{V}_{TA}(l)$, to znaczy, że zdanie reprezentowane przez p jest prawdziwe w lokacji l .

Dla danego zbioru automatów czasowych $\mathcal{T}A_1, \mathcal{T}A_2, \dots, \mathcal{T}A_n$ i funkcji $\mathcal{V}_{\mathcal{T}A_i} : L_i \rightarrow 2^{PV}$, funkcję wartościującą $\mathcal{V}_{\mathcal{T}A'} : L' \rightarrow 2^{PV}$ dla produktu automatów $\mathcal{T}A' = \mathcal{T}A_1 \parallel \mathcal{T}A_2 \parallel \dots \parallel \mathcal{T}A_n$ definiujemy w następujący sposób dla $l = (l_1, \dots, l_n) \in L'$:

$$p \in \mathcal{V}_{\mathcal{T}A'}(l) \text{ wtedy i tylko wtedy, gdy istnieje } 1 \leq i \leq n \text{ takie, że } p \in \mathcal{V}_{\mathcal{T}A_i}(l_i).$$

Funkcję wartościującą \mathcal{V}_{TA} możemy w naturalny sposób rozszerzyć na funkcję $\mathcal{V}_a : S_a \rightarrow 2^{PV}$ wartościującą stany systemu tranzycyjnego dla automatu $\mathcal{T}A$. Dla $p \in PV$ i $s = (l, \tau)$, gdzie $l \in L$ i $\tau \in \mathbb{R}_+^X$:

$$p \in \mathcal{V}_a(s) \text{ wtedy i tylko wtedy, gdy } p \in \mathcal{V}_{TA}(l).$$

Modelem dla automatu czasowego $\mathcal{T}A$ jest para

$$\mathcal{M}_a = (\mathcal{T}S_a, \mathcal{V}_a).$$

2.1.6 Przykłady

Produkt automatów

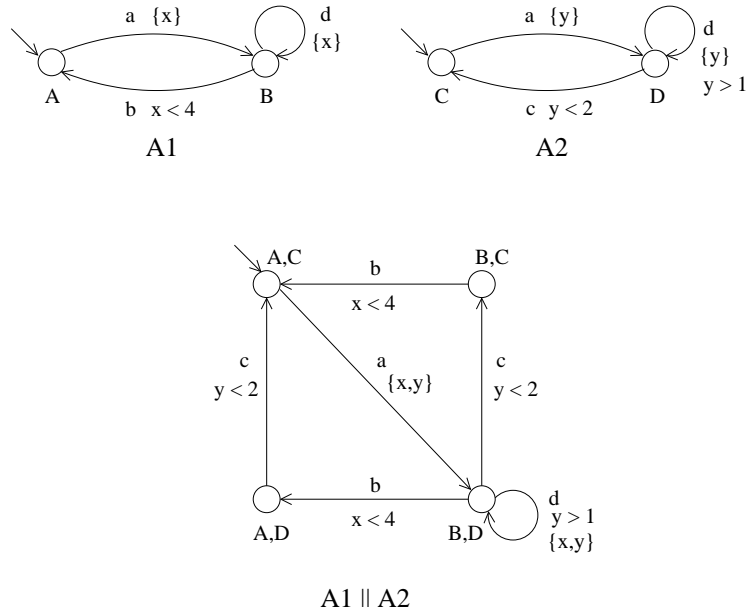
Na rysunku 2.1 przedstawiono dwa automaty czasowe $A1$ i $A2$ oraz ich produkt $A1 \parallel A2$. Automat czasowy $A1$ ma dwie lokacje: A i B , jeden zegar x i zbiór etykiet $\Sigma_1 = \{a, b, d\}$. Lokacją początkową automatu (oznaczoną małą strzałką) jest lokacja A . Relacja przejścia E_1 składa się z następujących elementów: $A \xrightarrow{a, true, \{x\}} B$, $B \xrightarrow{b, x < 4, \emptyset} A$ i $B \xrightarrow{d, true, \{x\}} B$. Niezmienniki w obydwu lokacjach automatu $A1$ są równe $true$ ¹.

Automat czasowy $A2$ ma również dwie lokacje: C i D , jeden zegar y i zbiór etykiet $\Sigma_2 = \{a, c, d\}$. Lokacją początkową automatu $A2$ jest lokacja C . Relacja przejścia E_2 składa się z następujących elementów: $C \xrightarrow{a, true, \{y\}} D$, $D \xrightarrow{c, y < 2, \emptyset} C$ i $D \xrightarrow{d, y > 1, \{y\}} D$. Niezmienniki w obydwu lokacjach automatu $A2$ są równe $true$.

Automat produktowy $A1 \parallel A2$ składa się z czterech lokacji: (A, C) , (A, D) , (B, C) i (B, D) . Automaty $A1$ i $A2$ wykonują przejścia o etykietach a i d wspólnie (synchronicznie), a pozostałe przejścia (o etykietach b i c) na zasadzie przeplotu. Jednym z możliwych s^0 -przebiegów automatu produktowego jest następujący przebieg (para (l, τ) , gdzie $l = (l_1, l_2)$, $l_1 \in L_1$, $l_2 \in L_2$ i $\tau = (\tau(x), \tau(y))$, oznacza stan automatu):

$$\begin{aligned} ((A, C), (0, 0)) &\xrightarrow{a} ((B, D), (0, 0)) \xrightarrow{0.5} ((B, D), (0.5, 0.5)) \xrightarrow{b} ((A, D), (0.5, 0.5)) \xrightarrow{0.5} \\ &((A, C), (1, 1)) \xrightarrow{a} ((B, D), (0, 0)) \xrightarrow{0.5} \dots \end{aligned}$$

¹Na rysunkach pomijamy niezmienniki i warunki równe $true$, a także puste zbiory zegarów do wyzerowania.



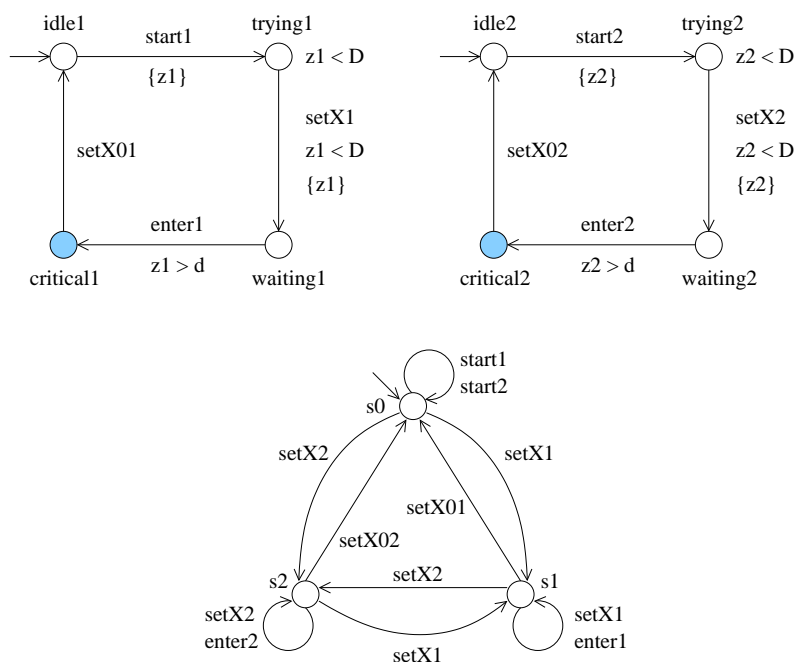
Rysunek 2.1: Produkt automatów czasowych

Protokół Fischera wzajemnego wykluczenia

Od momentu opublikowania w pracy [ACD⁺92] protokół Fischera wzajemnego wykluczenia stał się jednym z najbardziej znanych przykładów automatów czasowych. Rysunek 2.2 przedstawia ten protokół dla dwóch procesów.

Wzajemne wykluczenie procesów polega na tym, że żadne dwa procesy nie mogą jednocześnie wykonywać pewnych czynności, które są określane jako *sekcja krytyczna*. W przykładzie, automat $P1$ nie może przebywać w lokacji *critical1*, jeżeli automat $P2$ znajduje się w tym czasie w lokacji *critical2*. Fischer zaproponował następujące rozwiązanie tego problemu. Dostęp do sekcji krytycznej jest koordynowany przez globalną zmienną modelowaną przez trzeci automat. Każdy proces będąc w stanie beczynności (*idle*) może zgłosić chęć wejścia do sekcji krytycznej pod warunkiem, że zmienna ma wartość 0. Automat $P1$ może wykonać przejście *start1* do lokacji *trying1*, tylko wtedy, gdy automat X jest w lokacji $s0$ (co odpowiada temu, że zmienna ma wartość 0). Co najwyżej D jednostek czasu później automat $P1$ przechodzi do lokacji *waitng1* wykonując tranzycję *setX1* odpowiadającą nadaniu zmiennej wartości 1. Ograniczenia na czas wykonania tranzycji *setX1* są wyrażone przez warunek $z1 < D$, gdzie $z1$ jest zegarem zerowanym przy wykonywaniu tranzycji *start1* i *setX1*. Od tego momentu automat $P1$ jest gotowy do wejścia do sekcji krytycznej (przejście *enter1* do lokacji *critical1*), ale żeby to zrobić musi odczekać w lokacji *waiting1* co najmniej d jednostek czasu. Jest to określone przez warunek $z1 > d$. Proces $P1$ opuszcza sekcję krytyczną wykonując tranzycję *setX01*. Działanie automatu $P2$ jest analogiczne.

Automat X ma trzy lokacje odpowiadające trzem wartościom zmiennej, czyli $s0$, $s1$



Rysunek 2.2: Protokół Fischera dla dwóch procesów

i s_2 . Początkowo zmienna ma wartość 0. Jeżeli proces o numerze i chce wejść do sekcji krytycznej, to ustawia zmienną na i , co oznacza, że automat X przechodzi do lokacji s_i . Następnie zmienna może zostać ustawiona na 0 lub na j , jeżeli proces o numerze j zgłosił żądanie wejścia do sekcji krytycznej.

2.2 Weryfikacja automatów czasowych

2.2.1 Logiki temporalne

Własności systemów czasowych są zazwyczaj wyrażane w postaci formuł logik temporalnych zarówno czasowych, jak i bezczasowych (dobrym wprowadzeniem do logik modalnych i temporalnych są prace [HC84, Gol92]).

W tej części przedstawimy składnię i semantykę logiki CTL*[CES86] oraz jej najważniejsze podlogiki i rozszerzenia czasowe. Innymi ważnymi logikami temporalnymi są: modalny rachunek- μ [Koz83] i jego czasowe rozszerzenie — modalny rachunek- $T\mu$ [HNSY94a], których nie będziemy tu omawiać. Przegląd i porównanie logik temporalnych dla systemów czasowych można znaleźć między innymi w pracach [Eme90, AH92, Pen95].

Składnia CTL*

Jak poprzednio, niech PV będzie zbiorem zmiennych zdaniowych. Zbiór *formuł stanowych* i zbiór *formuł ścieżkowych* definiujemy indukcyjnie w następujący sposób:

- każda zmienna zdaniowa ze zbioru PV jest formułą stanową,
- jeżeli φ i ψ są formułami stanowymi, to $\neg\varphi$ i $\varphi \wedge \psi$ są również formułami stanowymi,
- jeżeli φ jest formułą ścieżkową, to $A\varphi$ jest formułą stanową,
- każda formuła stanowa φ jest również formułą ścieżkową,
- jeżeli φ i ψ są formułami ścieżkowymi, to $\neg\varphi$ i $\varphi \wedge \psi$ są również formułami ścieżkowymi,
- jeżeli φ i ψ są formułami ścieżkowymi, to $X\varphi$ i $U(\varphi, \psi)$ są również formułami ścieżkowymi.

Operatory \neg , \wedge i \vee nazywamy *logicznymi*. Operator alternatywy (\vee) definiujemy następująco:

$$\varphi \vee \psi \stackrel{def}{=} \neg(\neg\varphi \wedge \neg\psi)$$

Mówiąc nieformalnie, operator A , który nazywamy *kwantyfikatorem ścieżkowym*, oznacza “dla wszystkich ścieżek”, czyli $A\varphi$ w danym stanie oznacza, że dla wszystkich ścieżek wychodzących z tego stanu jest prawdziwa formuła φ . Definiujemy również operator E , który jest kwantyfikatorem ścieżkowym dualnym do A , oznaczającym istnienie ścieżki:

$$E\varphi \stackrel{def}{=} \neg A\neg\varphi$$

Operator modalny U (od ang. *until*) nazywamy *stanowym*. $U(\varphi, \psi)$ (czytamy: φ “aż do” ψ) oznacza, że na danej ścieżce istnieje stan, w którym jest prawdziwa formuła ψ i we wszystkich stanach go poprzedzających jest prawdziwa formuła φ . Definiujemy także następujące operatory stanowe:

- R — operator dualny do U . $R(\varphi, \psi)$ oznacza, że dla wszystkich stanów jest prawdziwa formuła ψ lub istnieje taki stan, dla którego jest prawdziwa formuła φ (czytamy: φ “zwalnia” ψ):

$$R(\varphi, \psi) \stackrel{def}{=} \neg U(\neg\varphi, \neg\psi)$$

- G — operator oznaczający “dla wszystkich stanów na ścieżce”:

$$G\varphi \stackrel{def}{=} R(false, \varphi)$$

- F — operator dualny do G , oznaczający istnienie stanu na ścieżce:

$$F\varphi \stackrel{def}{=} U(true, \varphi)$$

Kilka przykładów formuł temporalnych znajduje się w następnym rozdziale rozprawy (punkt 3.4).

Semantyka CTL*

Niech $\mathcal{M} = (\mathcal{TS}, \mathcal{V})$, gdzie $\mathcal{S} = (S, s^0, \Lambda, \longrightarrow)$ będzie modelem, o którym zakładamy, że relacja przejścia \longrightarrow jest całkowita, czyli dla każdego $s \in S$ istnieją $s' \in S$ i $\lambda \in \Lambda$ takie, że $s \xrightarrow{\lambda} s'$ (całkowitość relacji możemy zapewnić dodając do stanów nie mających następnika przejścia do nich samych). Niech $\sigma = s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} \dots$ oznacza nieskończoną ścieżkę w \mathcal{TS} , a $\sigma_i = s_i \xrightarrow{\lambda_i} s_{i+1} \xrightarrow{\lambda_{i+1}} \dots$ niech będzie sufiksem σ .

Definicja 2.5 Prawdziwość formuły φ w stanie s modelu \mathcal{M} (co oznaczamy $\mathcal{M}, s \models \varphi$ ²) definiujemy indukcyjnie w następujący sposób:

- $\mathcal{M}, s \models p$ wtedy i tylko wtedy, gdy $p \in \mathcal{V}(s)$ dla $p \in PV$,
- $\mathcal{M}, s \models \neg\varphi$ wtedy i tylko wtedy, gdy nieprawda, że $\mathcal{M}, s \models \varphi$,
- $\mathcal{M}, s \models \varphi \wedge \psi$ wtedy i tylko wtedy, gdy $\mathcal{M}, s \models \varphi$ i $\mathcal{M}, s \models \psi$,
- $\mathcal{M}, s \models A\varphi$ wtedy i tylko wtedy, gdy $\mathcal{M}, \sigma \models \varphi$ dla każdej ścieżki σ ze stanu s ,
- $\mathcal{M}, \sigma \models \varphi$ wtedy i tylko wtedy, gdy $\mathcal{M}, s_0 \models \varphi$,
- $\mathcal{M}, \sigma \models \neg\varphi$ wtedy i tylko wtedy, gdy nieprawda, że $\mathcal{M}, \sigma \models \varphi$,
- $\mathcal{M}, \sigma \models \varphi \wedge \psi$ wtedy i tylko wtedy, gdy $\mathcal{M}, \sigma \models \varphi$ i $\mathcal{M}, \sigma \models \psi$,
- $\mathcal{M}, \sigma \models X\varphi$ wtedy i tylko wtedy, gdy $\mathcal{M}, \sigma_1 \models \varphi$,
- $\mathcal{M}, \sigma \models U(\varphi, \psi)$ wtedy i tylko wtedy, gdy istnieje $i \geq 0$ takie, że $\mathcal{M}, \sigma_i \models \psi$ oraz $\mathcal{M}, \sigma_j \models \varphi$ dla $0 \leq j < i$.

²Model \mathcal{M} możemy opuścić, jeżeli wynika on jasno z kontekstu. Piszemy wtedy $s \models \varphi$.

Podlogiki CTL*

Poniziej przedstawiamy krótko najpopularniejsze podlogiki CTL*. Podlogikę CTL*, w której:

- nie występują kwantyfikatory egzystencjalne i negacja może występować tylko w podformułach nie zawierających operatorów modalnych, nazywamy ACTL*,
- każdy modalny operator stanowy jest poprzedzony kwantyfikatorem ścieżkowym (czyli kwantyfikatory A i E oraz operatory X, U, R, G i F mogą występować tylko parami — w kombinacjach: AG, AF, EG, EF, AX itd.), nazywamy CTL [EC82],
- formuły są wyłącznie postaci $A\varphi$, gdzie φ nie zawiera kwantyfikatorów ścieżkowych, nazywamy LTL [Pnu77],
- nie występuje operator modalny X, nazywamy CTL*_X (analogicznie definiujemy podlogiki ACTL*_X, CTL_X i LTL_X).

TCTL

Jedną z najbardziej znanych czasowych logik temporalnych jest TCTL [ACD90]. Logika TCTL jest rozszerzeniem czasowym CTL_X, w którym operatorom modalnym towarzyszą przedziały określające ograniczenia czasowe. Zbiór formuł logiki TCTL definiujemy indukcyjnie w następujący sposób:

- każda zmienna zdaniowa ze zbioru PV jest formułą,
- jeżeli φ i ψ są formułami, to $\neg\varphi$ i $\varphi \wedge \psi$ są również formułami,
- jeżeli φ i ψ są formułami, to $AU_J(\varphi, \psi)$ jest formułą, gdzie J jest przedziałem, określonym w zbiorze \mathbb{R}_+ , postaci: (d_1, d_2) , $(d_1, d_2]$, (d_1, ∞) , $[d_1, d_2)$, $[d_1, d_2]$ lub $[d_1, \infty)$ dla $d_1, d_2 \in \mathbb{N}$.

Pozostałe operatory definiujemy w sposób analogiczny do podanego wcześniej. Na przykład formuła $AG_{[0, \infty)}(p_1 \Rightarrow AF_{[0, 10]}p_2)$, gdzie $p_1, p_2 \in PV$ oznacza, że zawsze gdy zachodzi p_1 , to w ciągu 10 jednostek czasu zachodzi p_2 . Semantyka dla TCTL jest przedstawiona między innymi w pracy [ACD93].

2.2.2 Weryfikacja modelowa

Powiemy, że formuła φ logiki temporalnej CTL* jest prawdziwa w modelu \mathcal{M} wtedy i tylko wtedy gdy $\mathcal{M}, s^0 \models \varphi$, czyli φ jest prawdziwa w stanie początkowym modelu \mathcal{M} .

Problem *weryfikacji modelowej* polega na sprawdzeniu, czy dana formuła φ jest prawdziwa w modelu \mathcal{M} . Istnieje wiele metod rozwiązywania tego problemu. Najstarsze opracowane metody wykorzystują technikę *etykietowania stanów* (ang. *state labelling*) formułami, które są prawdziwe w tych stanach. Algorytm etykietowania rozpoczyna od analizy najprostszyc składowych formuły, czyli od zmiennych zdaniowych. W kolejnych krokach stany etykietowane są podformułami o coraz większej długości.

Inne podejście (nazywane *automatowym*) polega na konstrukcji automatu dla negacji formuły, której prawdziwość chcemy sprawdzić. Następnie budowany jest produkt automatu opisującego działanie systemu i automatu dla negacji formuły. Weryfikacja polega na sprawdzeniu, czy język akceptowany przez automat produktowy jest niepusty. Generalnie problem niepustości dla dowolnego automatu czasowego jest nierozstrzygalny, jednak w przypadku weryfikacji modelowej automat produktowy jest specyficznej postaci i problem sprawdzenia niepustości akceptowanego przez niego języka sprowadza się do problemu *jednoliterowej niepustości*, który ma rozwiązanie o złożoności liniowej [DW99].

Kolejną grupę metod stanowią *metody symboliczne*, czyli takie, w których przestrzeń stanów jest reprezentowana w sposób symboliczny, na przykład w postaci różnych diagramów decyzyjnych, takich jak NDD (*Numeric Decision Diagrams*) [ABK⁺97], CDD (*Clock Decision Diagrams*) [Wan00], czy DDD (*Difference Decision Diagrams*) [MLAH99]. Do metod symbolicznych można zaliczyć także techniki oparte na kodowaniu systemów czasowych i ich własności w postaci formuł logiki zdaniowej i wykorzystaniu algorytmów i narzędzi sprawdzających spełnialność takich formuł (ang. *SAT-solvers*). Przegląd metod symbolicznych związanych z techniką SAT przedstawiono w [PBG05].

Podstawową lekturą na temat weryfikacji modelowej jest praca [CGP99]. Natomiast szczegółowy i aktualny przegląd technik weryfikacji modelowej systemów czasowych oraz narzędzi, które je wykorzystują można znaleźć w pracy [PP06].

Chcielśmy podkreślić, że w większości przypadków problem weryfikacji modelowej formuł TCTL rozwiązuje się poprzez przełożenie go na problem weryfikacji modelowej formuł CTL. Dlatego w rozprawie koncentrujemy się na wykazaniu, że prezentowane przez nas metody zachowują prawdziwość formuł bezczasowych logik temporalnych.

2.2.3 Relacje równoważności systemów

Na koniec podamy podstawowe relacje równoważności między systemami, które charakteryzują również systemy czasowe, a mianowicie relację silnej symulacji (ang. *strong simulation*) i symulacji z powtórzeniami (ang. *stuttering simulation*).

Niech $\mathcal{M} = (\mathcal{TS}, \mathcal{V})$ i $\mathcal{M}' = (\mathcal{TS}', \mathcal{V}')$ będą dwiema strukturami takimi, że $\mathcal{TS} = (S, s^0, \Lambda, \longrightarrow)$, $\mathcal{TS}' = (S', s^{0'}, \Lambda', \longrightarrow')$, $\mathcal{V} : \mathcal{TS} \rightarrow 2^{PV}$ i $\mathcal{V}' : \mathcal{TS}' \rightarrow 2^{P'V}$ dla pewnego zbioru zmiennych zdaniowych PV .

Silna bisymulacja

Definicja 2.6 [BCG88] *Relacja $\cong_b \subseteq S \times S'$ jest relacją silnej symulacji pomiędzy dwiema strukturami \mathcal{M} i \mathcal{M}' jeżeli spełnione są następujące warunki:*

1. $s^0 \cong_b s^{0'}$ i
2. jeżeli $s \cong_b s'$, to $\mathcal{V}(s) = \mathcal{V}'(s')$ i dla każdego stanu s_1 w \mathcal{M} takiego, że $s \xrightarrow{\lambda} s_1$, gdzie $\lambda \in \Lambda$, istnieje stan s'_1 w \mathcal{M}' i etykieta $\lambda' \in \Lambda'$ takie, że $s' \xrightarrow{\lambda'} s'_1$ i $s_1 \cong_b s'_1$.

Relacja \cong_b jest relacją silnej bisymulacji, jeżeli ona i relacja do niej odwrotna są silnymi symulacjami.

Twierdzenie 2.7 [BCG88] Niech φ będzie formułą CTL* nad zbiorem zmiennych zdaniowych PV, niech \mathcal{M} i \mathcal{M}' będą dwiema skończonymi strukturami i niech relacja \cong_b będzie relacją silnej bisymulacji pomiędzy \mathcal{M} i \mathcal{M}' . Dla każdej pary stanów s i s' takich, że $s \cong_b s'$ zachodzi $\mathcal{M}, s \models \varphi$ wtedy i tylko wtedy, gdy $\mathcal{M}', s' \models \varphi$.

Bisymulacja z powtórzeniami

Definicja 2.8 [BCG88, GKPP99] Relacja $\cong_{sb} \subseteq S \times S'$ jest relacją symulacji z powtórzeniami pomiędzy dwiema strukturami \mathcal{M} i \mathcal{M}' , jeżeli spełnione są następujące warunki:

1. $s^0 \cong_{sb} s^{0'}$ i
2. jeżeli $s \cong_{sb} s'$, to $\mathcal{V}(s) = \mathcal{V}(s')$ i dla każdej ścieżki σ w \mathcal{M} ze stanu s , istnieje ścieżka σ' w \mathcal{M}' ze stanu s' , podział B_1, B_2, \dots ścieżki σ i podział B'_1, B'_2, \dots ścieżki σ' taki, że dla każdego $j \geq 1$, B_j i B'_j są niepuste i skończone oraz każdy stan z B_j jest w relacji \cong_{sb} z każdym stanem z B'_j .

Relacja \cong_{sb} jest relacją bisymulacji z powtórzeniami, jeżeli ona i relacja do niej odwrotna są symulacjami z powtórzeniami.

Twierdzenie 2.9 [BCG88, GKPP99] Niech φ będzie formułą CTL*_X nad zbiorem zmiennych zdaniowych PV, niech \mathcal{M} i \mathcal{M}' będą dwiema skończonymi strukturami i niech relacja \cong_{sb} będzie relacją bisymulacji z powtórzeniami pomiędzy \mathcal{M} i \mathcal{M}' . Dla każdej pary stanów s i s' takich, że $s \cong_{sb} s'$ zachodzi $\mathcal{M}, s \models \varphi$ wtedy i tylko wtedy, gdy $\mathcal{M}', s' \models \varphi$.

Rozdział 3

Język bazowy

W języku bazowym system czasu rzeczywistego opisujemy jako zbiór procesów komunikujących się ze sobą poprzez bufory lub zmienne dzielone. Każdy proces jest przedstawiony jako automat skończenie stanowy. W języku nie występują jawnie zmienne reprezentujące czas, jednak z każdą tranzycją może być związane tak zwane *dozwolone opóźnienie*, które ogranicza czas jej wykonania.

W tym rozdziale przedstawiamy składnię abstrakcyjną i semantykę języka bazowego. Pokażemy także w jaki sposób możemy formułować i weryfikować własności programów w języku bazowym. Zastosowanie języka do opisu systemów czasu rzeczywistego zilustrujemy kilkoma przykładami.

3.1 Składnia abstrakcyjna

Niech V będzie skończonym zbiorem *zmiennych*, a B — skończonym zbiorem *buforów*. Bufor jest ciągiem wartości całkowitych.

Zbiór wszystkich *wyrażeń arytmetycznych* $Exp(V)$ nad zbiorem V definiujemy poprzez gramatykę:

$$exp ::= m \mid y \mid exp \oplus exp \mid -exp \mid (exp)$$

gdzie $m \in \mathbf{Z}$, $y \in V$ i $\oplus \in \{-, +, *, /, \%\}$. Przez $/$ oznaczamy dzielenie całkowite, a przez $\%$ — operację modulo (resztę z dzielenia całkowitego). Niech $BExp(V, B)$ będzie zbiorem wszystkich *wyrażeń logicznych* nad V i B zdefiniowanym następująco:

$$bexp ::= true \mid e_1 \sim e_2 \mid empty(b) \mid bexp \wedge bexp \mid bexp \vee bexp \mid \neg bexp \mid (bexp)$$

gdzie $e_1, e_2 \in Exp(V)$, $b \in B$, i $\sim \in \{=, \neq, <, >, \leq, \geq\}$. Zbiór wszystkich instrukcji $Ins(V, B)$ definiujemy jako:

$$ins ::= y := e \mid get(b, y) \mid put(b, e)$$

gdzie $y \in V$, $b \in B$ i $e \in Exp(V)$. *Instrukcją* nazywamy element zbioru $Ins(V, B)$. Zbiór wszystkich *akcji* $Act(V, B)$ nad V i B definiujemy w następujący sposób:

$$act ::= \varepsilon \mid a \mid act; act$$

gdzie ε oznacza pusty ciąg i $a \in \text{Ins}(V, B)$.

Definicja 3.1 Programem w języku bazowym nazywamy krotkę $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie V jest zbiorem zmiennych, B jest zbiorem buforów i $n \in \mathbb{N}$. Procesem P_i nazywamy krotkę $(id_i, Q_i, q_i^0, \Gamma_i, T_i)$, gdzie id_i jest nazwą procesu, Q_i jest zbiorem stanów kontrolnych (zakładamy, że $Q_i \cap Q_j = \emptyset$ dla $i \neq j$), $q_i^0 \in Q_i$ jest stanem początkowym, Γ_i jest zbiorem etykiet a T_i jest zbiorem tranzycji w postaci $(q_i, g, d, u, \gamma, \alpha, q_i')$, gdzie:

- $q_i \in Q_i$ nazywamy stanem źródłowym tranzycji,
- $q_i' \in Q_i$ nazywamy stanem docelowym tranzycji,
- $g \in \text{BExp}(V, B)$ nazywamy dozorem tranzycji,
- $d \in \{(d_1, d_2), (d_1, d_2], (d_1, \infty), [d_1, d_2], [d_1, d_2], [d_1, \infty) \mid d_1, d_2 \in \mathbb{N}\}$ nazywamy dozwo-
lonym opóźnieniem tranzycji,
- $u \in \{\text{true}, \text{false}\}$ nazywamy atrybutem pilności tranzycji,
- $\gamma \in \Gamma_i$ nazywamy etykietą tranzycji,
- $\alpha \in \text{Act}(V, B)$ nazywamy akcją tranzycji.

Dla $t = (q_i, g, d, u, \gamma, \alpha, q_i')$, wprowadzimy oznaczenia: $\text{source}(t)$, $\text{guard}(t)$, $\text{delay}(t)$, $\text{urgent}(t)$, $\text{label}(t)$, $\text{action}(t)$ i $\text{target}(t)$ odpowiednio na $q_i, g, d, u, \gamma, \alpha$ i q_i' . Wymienione elementy będziemy nazywać *atrybutami* tranzycji.

Dozór

Dozór $\text{guard}(t)$ określa warunki umożliwienia tranzycji t . Tranzycja t może być wykonana tylko wtedy, gdy jej dozór jest prawdziwy.

Dozwolone opóźnienie

Dozwolone opóźnienie $\text{delay}(t)$ reprezentuje ograniczenia nałożone na czas wykonania tranzycji t . Dla $\text{delay}(t) = (d_1, d_2)$ tranzycja może być wykonana kiedy upłynie więcej niż d_1 i mniej niż d_2 jednostek czasu od momentu wejścia procesu do stanu źródłowego tranzycji t . Jeżeli $\text{delay}(t) = (d_1, d_2]$, to tranzycja może być wykonana po upłynięciu więcej niż d_1 , ale mniej niż d_2 lub dokładnie d_2 jednostek czasu. Dla $\text{delay}(t) = (d_1, \infty)$ tranzycję można wykonać kiedy minie więcej niż d_1 jednostek czasu (bez górnego ograniczenia). Analogicznie dla pozostałych form dozwolonego opóźnienia.

Zakładamy, że dla każdej tranzycji t istnieje $r \in \mathbb{R}_+$ takie, że $r \in \text{delay}(t)$ (przedział określający dozwolone opóźnienie nie jest pusty).

Na podstawie dozwolonego opóźnienia definiujemy *górne dozwolone opóźnienie* w następujący sposób:

$$\text{upper_delay}(t) \stackrel{\text{def}}{=} \begin{cases} [0, d_2] & \text{jeżeli } \text{delay}(t) = (d_1, d_2) \text{ lub } \text{delay}(t) = [d_1, d_2], \\ [0, d_2] & \text{jeżeli } \text{delay}(t) = (d_1, d_2] \text{ lub } \text{delay}(t) = [d_1, d_2]. \end{cases}$$

Atrybut pilności

Atrybut pilności $urgent(t)$ określa czy tranzycja ma być wykonana natychmiast, gdy tylko jest możliwa do wykonania. Jeżeli dozór tranzycji jest prawdziwy, gdy proces wchodzi do stanu źródłowego tranzycji t , to ma być ona wykonana od razu po wejściu procesu do tego stanu. W przeciwnym przypadku ma być wykonana wtedy, gdy dozór stanie się prawdziwy. Tranzycję, dla której $urgent(t) = true$, nazywamy *pilną*. Zakładamy, że jeżeli tranzycja jest pilna, to jej dozwolone opóźnienie jest równe $[0, \infty)$, czyli nie ma innych ograniczeń na czas jej wykonania.

Etykieta

Etykieta $label(t)$ może być albo *lokalna*, albo *synchronizująca*. Etykieta lokalna jest unikalna w programie, a synchronizująca może występować w wielu procesach. Podobnie jak w przypadku automatów czasowych procesy wykonują jednocześnie tranzycje synchroniczne o takich samych etykietach, a tranzycje lokalne na zasadzie przepłotu.

Zauważmy, że jeżeli tranzycja jest pilna, nie jest synchroniczna i jej dozór jest równy $true$, to powinna być wykonana natychmiast, gdy proces znajdzie się w jej stanie źródłowym (bez upływu czasu). A zatem zamiast ustawiać atrybut pilności, możemy w tym przypadku zdefiniować dozwolone opóźnienie równe $[0, 0]$. Obydwa zapisy są w tej sytuacji równoważne, więc dla uproszczenia rozważań przyjmujemy, że jeżeli tranzycja ma dozór równy $true$ i nie jest synchroniczna, to nie jest pilna.

Akcja

W trakcie wykonywania tranzycji proces może zmieniać wartości zmiennych i buforów. Sposób tej zmiany jest określony przez akcję $action(t)$, która jest ciągiem przypisań na zmienne i operacji na buforach.

3.1.1 Definicje i oznaczenia

Podamy teraz definicje i oznaczenia dotyczące języka bazowego wykorzystywane w kolejnych rozdziałach rozprawy. Niech $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$ będzie programem w języku bazowym, gdzie V jest zbiorem zmiennych, B jest zbiorem buforów, P_i jest procesem i $n \in \mathbb{N}$. Niech $1 \leq i \leq n$ oznacza numer dowolnego procesu należącego do programu, a T_i i Q_i niech będą odpowiednio zbiorem tranzycji i zbiorem stanów kontrolnych tego procesu. Niech wreszcie $Q = \bigcup_{i=1}^n Q_i$ oznacza zbiór stanów kontrolnych wszystkich procesów, $T = \bigcup_{i=1}^n T_i$ — zbiór tranzycji wszystkich procesów, a $\Gamma = \bigcup_{i=1}^n \Gamma_i$ — zbiór etykiet wszystkich procesów.

Zmienne i bufory występujące w wyrażeniu, w instrukcji, w akcji i w procesie

Niech $vars(e) \subseteq V$ będzie zbiorem zmiennych *występujących* w wyrażeniu arytmetycznym $e \in Exp(V)$, który definiujemy indukcyjnie w następujący sposób:

$$vars(e) = \begin{cases} \emptyset & \text{jeżeli } e = m, \text{ gdzie } m \in \mathbb{Z}, \\ \{y\} & \text{jeżeli } e = y, \text{ gdzie } y \in V, \\ vars(e_1) \cup vars(e_2) & \text{jeżeli } e = (e_1 \oplus e_2), \\ vars(e_1) & \text{jeżeli } e = -e_1 \text{ lub } e = (e_1). \end{cases}$$

Przez $vars(g) \subseteq V \cup B$ oznaczmy zbiór zmiennych i buforów występujących w wyrażeniu logicznym $g \in Bexp(V, B)$, który definiujemy jak następuje:

$$vars(g) = \begin{cases} \emptyset & \text{dla } g = true, \\ vars(e_1) \cup vars(e_2) & \text{dla } g = (e_1 \sim e_2), \text{ gdzie } e_1, e_2 \in Exp(V), \\ \{b\} & \text{dla } g = empty(b), \text{ gdzie } b \in B, \\ vars(g_1) \cup vars(g_2) & \text{dla } g = (g_1 \wedge g_2) \text{ lub } g = (g_1 \vee g_2), \\ vars(g_1) & \text{dla } g = \neg g_1 \text{ lub } g = (g_1). \end{cases}$$

Zbiór $vars(b) \subseteq V \cup B$ zmiennych i buforów występujących w instrukcji $a \in Ins(V, B)$ definiujemy następująco:

$$vars(a) = \begin{cases} \{y\} \cup vars(e) & \text{dla } a = (y := e), \\ \{b, y\} & \text{dla } a = get(b, y), \\ \{b\} \cup vars(e) & \text{dla } a = put(b, e). \end{cases}$$

Zbiór $vars(\alpha) \subseteq V \cup B$ zmiennych występujących w akcji $\alpha \in Act(V, B)$ definiujemy jako:

$$vars(\alpha) = \begin{cases} \emptyset & \text{dla } \alpha = \varepsilon, \\ vars(a) & \text{dla } \alpha = a, \text{ gdzie } a \in Ins(V, B), \\ vars(\alpha_1) \cup vars(\alpha_2) & \text{jeżeli } \alpha = \alpha_1; \alpha_2. \end{cases}$$

Do zbioru $vars(P_i)$ zmiennych występujących w procesie P_i należą wszystkie zmienne występujące w dozorach i akcjach tranzycji procesu:

$$vars(P_i) = \bigcup_{t \in T_i} vars(guard(t)) \cup vars(action(t))$$

Zmienne lokalne i dzielone

Zbiór V_G zmiennych *globalnych* (lub *dzielonych*) zawiera zmienne, które występują w co najmniej dwóch procesach:

$$V_G = \{y \in V \mid \exists_{1 \leq i \neq j \leq n} y \in vars(P_i) \cap vars(P_j)\}$$

Zbiór V_i zmiennych lokalnych procesu i definiujemy jako różnicę zbiorów:

$$V_i = vars(P_i) \setminus V_G$$

Zakładamy, że wszystkie bufory są dzielone, a także, że żadna akcja tranzycji o etykiecie synchronizującej nie zawiera przypisania wartości żadnej zmiennej dzielonej ani operacji na żadnym buforze.

Operacje

Operacją będziemy nazywać zarówno instrukcję $a \in Ins(V, B)$, jak i wyrażenie logiczne $g \in BExp(V, B)$, które jest dozorem tranzycji. Dla tranzycji $t \in T$ niech $opers(action(t))$ oznacza zbiór operacji występujących w akcji tranzycji, a $opers(guard(t))$ — jednoelementowy zbiór zawierający operację z dozoru tranzycji. Niech również $opers(t) = opers(action(t)) \cup$

$opers(guard(t))$ oznacza zbiór wszystkich operacji tranzycji t i niech $Ops = \bigcup_{t \in T} opers(t)$ będzie zbiorem wszystkich operacji programu. Przez $guards(q)$ oznaczamy zbiór operacji z dozorów tranzycji wychodzących ze stanu q : $guards(q) = \bigcup_{t \in out(q)} opers(guard(t))$.

Niech $\#a$ oznacza kolejny numer operacji a w akcji tranzycji (akcje są ciągami instrukcji, więc możemy je w naturalny sposób ponumerować). Powiemy, że operacja $a_1 \in opers(t)$ występuje *przed* operacją (odpowiednio *po* operacji) $a_2 \in opers(t)$, jeżeli $\#a_1 < \#a_2$ (odpowiednio $\#a_1 > \#a_2$). Przyjmujemy, że operacja z dozoru tranzycji występuje *przed* wszystkimi operacjami z akcji tej tranzycji. Powiemy również, że operacja $a_3 \in opers(t)$ występuje *pomiędzy* operacjami a_1 i a_2 , jeżeli $\#a_1 < \#a_3 < \#a_2$.

Zmienne/bufory definiowane i używane

Niech $def(a) \subseteq V \cup B$ będzie zbiorem zmiennych i buforów *definiowanych* przez operację $a \in Ops$, czyli takich, którym w tej operacji nadawane są wartości.

$$def(a) = \begin{cases} \{y\} & \text{jeżeli } a = (y := e), \\ \{b, y\} & \text{jeżeli } a = get(b, y), \\ \{b\} & \text{jeżeli } a = put(b, e), \\ \emptyset & \text{jeżeli } a \in BExp(V, B). \end{cases}$$

Definiujemy również zbiór $use(a) \subseteq V \cup B$ zmiennych i buforów *używanych* w operacji a w następujący sposób:

$$use(a) = \begin{cases} vars(e) & \text{jeżeli } a = (y := e), \\ \{b\} & \text{jeżeli } a = get(b, y), \\ \{b\} \cup vars(e) & \text{jeżeli } a = put(b, e), \\ vars(a) & \text{jeżeli } a \in BExp(V, B). \end{cases}$$

Ścieżki

Niech $out(q) = \{t \in T_i \mid q = source(t)\}$ oznacza zbiór tranzycji *wychodzących* ze stanu kontrolnego $q \in Q_i$ i niech $in(q) = \{t \in T_i \mid q = target(t)\}$ oznacza zbiór tranzycji *wchodzących* do stanu kontrolnego q . Mówimy, że stan kontrolny q jest *stanem końcowym*, jeżeli $out(q) = \emptyset$. Dla pary stanów $q, q' \in Q_i$, jeżeli istnieje tranzycja t taka, że $t \in out(q)$ i $t \in in(q')$, to stan q nazwiemy *poprzednikiem* stanu q' , a stan q' — *następnikiem* stanu q .

Ścieżka w procesie P_i ze stanu kontrolnego $q_1 \in Q_i$ do stanu kontrolnego $q_m \in Q_i$ jest to ciąg stanów i tranzycji $q_1 t_1 q_2 \dots t_{m-1} q_m$ taki, że $m \geq 2$, $q_j \in Q_i$ i $t_j \in out(q_{j-1}) \cap in(q_j) \subseteq T_i$ dla każdego $1 \leq j \leq m-1$.

Mówimy, że stan¹ q_m jest *osiągalny* ze stanu q_1 , co oznaczamy $q_1 \implies q_m$, jeżeli istnieje ścieżka z q_1 do q_m . Mówimy, że stan q *należy do* ścieżki $\pi = q_1 t_1 q_2 \dots t_{m-1} q_m$ (lub że ścieżka π *przechodzi przez* stan q), jeżeli istnieje $1 \leq j \leq m$ takie, że $q = q_j$. Podobnie, tranzycja t należy do ścieżki π , jeżeli istnieje $1 \leq j \leq m-1$ takie, że $t = t_j$. Zbiór wszystkich stanów należących do ścieżki π oznaczamy przez $states(\pi)$, a zbiór wszystkich tranzycji należących do ścieżki π oznaczamy przez $trans(\pi)$.

¹Będziemy używać krótszego określenia “stan” zamiast “stan kontrolny”, o ile nie będzie prowadziło to do nieporozumień.

Ścieżkę z q_1 do q_m nazywamy *cyklem*, jeżeli $q_1 = q_m$ i wszystkie pozostałe należące do niej stany kontrolne są różne. Stan należący do cyklu jest *stanem wejściowym cyklu*, jeżeli ma poprzednika, który nie należy do tego cyklu. Ścieżka $\pi = q_1 t_1 q_2 \dots t_{m-1} q_m$ zawiera cykl, jeżeli przechodzi przez jakiś stan więcej niż raz, czyli $q_i = q_j$ dla pewnych $q \leq i < j \leq n$. Mówimy, że ścieżka jest *maksymalna*, jeżeli prowadzi do stanu końcowego lub zawiera cykl. Ścieżkę nazwiemy *prostą*, jeżeli nie zawiera cyklu. Zbiór wszystkich ścieżek prostych ze stanu q_1 do stanu q_m będziemy oznaczać przez $\Pi(q_1, q_m)$. Zbiór wszystkich stanów kontrolnych należących do ścieżek ze zbioru $\Pi(q_1, q_m)$ oznaczamy przez $states(\Pi(q_1, q_m))$, a zbiór wszystkich tranzycji należących do ścieżek ze zbioru $\Pi(q_1, q_m)$ oznaczamy przez $trans(\Pi(q_1, q_m))$. W naszych rozważaniach przyjmujemy, że każdy stan kontrolny procesu jest osiągalny z jego stanu początkowego.

Program strukturalny

Dla dwóch stanów kontrolnych q, q' procesu P_i powiemy, że q *dominuje nad* q' , jeżeli każda ścieżka ze stanu początkowego q_i^0 do stanu q' przechodzi przez q . Przyjmijmy, że relacja dominacji jest zwrotna, czyli każdy stan dominuje nad sobą samym.

Definicja 3.2 *O programie w języku bazowym powiemy, że jest strukturalny², jeżeli zbiór tranzycji T_i każdego procesu programu możemy podzielić na dwa rozłączne podzbiory F_i i B_i o następujących własnościach:*

1. ze stanu początkowego istnieje ścieżka do każdego innego stanu procesu, która składa się wyłącznie z tranzycji należących do zbioru F_i i w procesie nie istnieje cykl, do którego należą wyłącznie tranzycje ze zbioru F_i ,
2. do zbioru B_i należą wyłącznie tranzycje, których stan docelowy dominuje nad stanem źródłowym.

Kluczowa własność programu strukturalnego jest następująca.

Własność 3.3 [ASU02] *W żadnym procesie programu strukturalnego nie ma cyklu zawierającego dwa stany wejściowe.*

Niestrukuralny przepływ sterowania bardzo rzadko występuje w praktyce. Jeżeli w programie są użyte wyłącznie strukturalne instrukcje takie jak *if-then-else*, *while-do*, *continue* czy *break*, to program zawsze jest strukturalny. Nawet programy napisane przy użyciu instrukcji *goto* prawie zawsze są strukturalne, ponieważ stosowanie pętli z więcej niż jednym wejściem nie jest powszechne (według autorów [ASU02]). Metodę redukcji przedstawioną w rozdz. 5 opracowano dla strukturalnych programów w języku bazowym.

3.2 Semantyka

W tej części przedstawimy semantykę programu w języku bazowym, a więc określimy czym jest stan programu, podamy relację przejścia między stanami i zdefiniujemy wykonanie programu. Na początku wprowadzimy kilka pomocniczych definicji.

²W literaturze polskojęzycznej używa się pojęcia *redukowalny*, ale my przyjmujemy określenie *strukturalny*, ponieważ terminu *redukcja* będziemy używać w innym znaczeniu w rozdz 5.

Wartościowanie zmiennych i buforów

Niech \mathbb{Z}^* oznacza zbiór ciągów liczb całkowitych. Przez Ω będziemy oznaczać sumę $\mathbb{Z} \cup \mathbb{Z}^*$. Funkcję całkowitą $v : V \cup B \rightarrow \Omega$ taką, że $v(V) \subseteq \mathbb{Z}$ i $v(B) \subseteq \mathbb{Z}^*$ nazwiemy *wartościowaniem zmiennych i buforów* lub krótko *wartościowaniem*. Funkcja v przypisuje każdej zmiennej wartość całkowitą a każdemu buforowi ciąg wartości całkowitych (pusty ciąg będziemy oznaczać przez ε). Wartościowanie v^0 takie, że $v^0(b) = \varepsilon$ dla każdego bufora $b \in B$ (wszystkie bufory są puste) nazywamy *wartościowaniem początkowym*. Wartościowanie wyrażeń arytmetycznych $\mathcal{E}(e, v)$ dla $e \in \text{Exp}(V)$ i danego wartościowania $v \in \Omega^{V \cup B}$ definiujemy w standardowy sposób:

- $\mathcal{E}(m, v) = m$, dla $m \in \mathbb{Z}$,
- $\mathcal{E}(y, v) = v(y)$, dla $y \in V$,
- $\mathcal{E}(e_1 \oplus e_2, v) = \mathcal{E}(e_1, v) \oplus \mathcal{E}(e_2, v)$, dla $e_1, e_2 \in \text{Exp}(V)$,
- $\mathcal{E}(-e, v) = -\mathcal{E}(e, v)$,
- $\mathcal{E}((e), v) = \mathcal{E}(e, v)$.

Niech $\{tt, ff\}$ będzie zbiorem *wartości logicznych*, $e_1, e_2 \in \text{Exp}(V)$, $b \in B$ i $g_1, g_2 \in \text{BExp}(V, B)$. Wartościowanie wyrażeń logicznych $\mathcal{B}(g, v)$ dla $g \in \text{BExp}(V, B)$ i danego wartościowania $v \in \Omega^{V \cup B}$ definiujemy następująco:

- $\mathcal{B}(\text{true}, v) = tt$,
- $\mathcal{B}(e_1 \sim e_2, v) = \begin{cases} tt & \text{jeżeli } \mathcal{E}(e_1, v) \sim \mathcal{E}(e_2, v), \\ ff & \text{jeżeli } \mathcal{E}(e_1, v) \not\sim \mathcal{E}(e_2, v), \end{cases}$
- $\mathcal{B}(\text{empty}(b), v) = \begin{cases} tt & \text{jeżeli } v(b) = \varepsilon, \\ ff & \text{jeżeli } v(b) \neq \varepsilon, \end{cases}$
- $\mathcal{B}(g_1 \wedge g_2, v) = \begin{cases} tt & \text{jeżeli } \mathcal{B}(g_1, v) = tt \text{ i } \mathcal{B}(g_2, v) = tt, \\ ff & \text{jeżeli } \mathcal{B}(g_1, v) = ff \text{ lub } \mathcal{B}(g_2, v) = ff, \end{cases}$
- $\mathcal{B}(g_1 \vee g_2, v) = \begin{cases} tt & \text{jeżeli } \mathcal{B}(g_1, v) = tt \text{ lub } \mathcal{B}(g_2, v) = tt, \\ ff & \text{jeżeli } \mathcal{B}(g_1, v) = ff \text{ i } \mathcal{B}(g_2, v) = ff, \end{cases}$
- $\mathcal{B}(\neg g, v) = \begin{cases} tt & \text{jeżeli } \mathcal{B}(g, v) = ff, \\ ff & \text{jeżeli } \mathcal{B}(g, v) = tt, \end{cases}$
- $\mathcal{B}((g), v) = \begin{cases} tt & \text{jeżeli } \mathcal{B}(g, v) = tt, \\ ff & \text{jeżeli } \mathcal{B}(g, v) = ff. \end{cases}$

Niech $\text{pop} : \mathbb{Z}^* \rightarrow \mathbb{Z}^*$, $\text{top} : \mathbb{Z}^* \rightarrow \mathbb{Z}$ i $\text{push} : \mathbb{Z}^* \times \mathbb{Z} \rightarrow \mathbb{Z}^*$ będą funkcjami operującymi na ciągach wartości całkowitych. Dla ciągu $w = u_1 u_2 \dots u_k$, gdzie $k \in \mathbb{N}$ i $u \in \mathbb{Z}$:

- $\text{pop}(w) = u_2 \dots u_k$,

- $top(w) = u_1$ dla $w \neq \varepsilon$ i $top(w) = 0$ dla $w = \varepsilon$,³
- $push(w, u) = u_1 u_2 \dots u_k u$.

Dla $y \in V \cup B$ i $u \in \Omega$ niech $v[u/y]$ oznacza wartościowanie v' , takie że $v'(z) = u$ dla $z = y$ i $v'(z) = v(z)$ dla $z \neq y$. Przez $\mathcal{J}(v, \alpha)$ oznaczamy wartościowanie v' , które otrzymano z wartościowania v po wykonaniu akcji $\alpha \in Act(V, B)$. Niech $b \in B$, $y \in V$ i $e \in Exp(V)$. Wartościowanie $\mathcal{J}(v, \alpha)$ jest zdefiniowane zgodnie ze strukturą akcji α w następujący sposób:

- $\mathcal{J}(v, \varepsilon) = v$ (akcja pusta),
- $\mathcal{J}(v, x := e) = v[\mathcal{E}(e, v) / x]$ (instrukcja przypisania),
- $\mathcal{J}(v, get(b, y)) = v[top(v(b)) / y, pop(v(b)) / b]$ (operacja wyjęcia z bufora),
- $\mathcal{J}(v, put(b, e)) = v[push(v(b), \mathcal{E}(e, v)) / b]$ (operacja włożenia do bufora),
- $\mathcal{J}(v, \alpha_1; \alpha_2) = v''$, gdzie $v'' = \mathcal{J}(v', \alpha_2)$ i $v' = \mathcal{J}(v, \alpha_1)$.

Wartościowanie opóźnień

Funkcję całkowitą $\mu : Q \rightarrow \mathbb{R}_+$, która każdemu stanowi kontrolnemu przypisuje liczbę rzeczywistą, nazywamy *wartościowaniem opóźnień*. Intuicyjnie, wartość opóźnienia $\mu(q)$ informuje o tym, ile czasu minęło od ostatniego wejścia i -tego procesu do stanu q , a jeżeli proces nigdy wcześniej nie był w stanie q , to $\mu(q)$ mówi, ile czasu minęło od początku działania programu. Zauważmy, że jeżeli proces znajduje się w stanie q , to $\mu(q)$ jest równe czasowi przebywania procesu w tym stanie.

Dla $\delta \in \mathbb{R}_+$, $\mu + \delta$ oznacza wartościowanie opóźnień μ' , takie że $\mu'(q) = \mu(q) + \delta$ dla każdego stanu kontrolnego $q \in Q$. Dla $Y \subseteq Q$, $\mu[Y := 0]$ oznacza wartościowanie opóźnień μ' , takie że $\mu'(q) = 0$ dla $q \in Y$ i $\mu'(q) = \mu(q)$ dla $q \in Q \setminus Y$. Wartościowanie μ^0 , takie że $\mu^0(q) = 0$ dla każdego stanu $q \in Q$ nazywamy *początkowym wartościowaniem opóźnień*.

Stan i konfiguracja programu

Stan programu jest określony przez stany kontrolne w jakich znajdują się procesy, wartościowanie zmiennych i buforów oraz wartościowanie opóźnień. Formalnie, *stanem programu* \mathcal{P} jest krotka $(q_1, \dots, q_n, v, \mu)$, gdzie $q_i \in Q_i$, $v \in \Omega^{V \cup B}$ i $\mu \in \mathbb{R}_+^Q$. Natomiast *konfiguracją programu* \mathcal{P} będziemy nazywać krotkę (q_1, \dots, q_n, v) , gdzie $q_i \in Q_i$ i $v \in \Omega^{V \cup B}$. Konfiguracja jest wyznaczona przez stany kontrolne procesów oraz wartościowanie zmiennych i buforów.

Przez $\Gamma(\gamma)$ oznaczamy zbiór numerów procesów, które posiadają etykietę γ w swoim zbiorze etykiet.

$$\Gamma(\gamma) = \{ 1 \leq i \leq n \mid \gamma \in \Gamma_i \}$$

Zdefiniujemy jeszcze pomocniczą funkcję $synch : T \rightarrow \{true, false\}$, której wartość informuje czy tranzycja jest synchroniczna. Jeżeli $|\Gamma(label(t))| > 1$, to $synch(t) = true$, w przeciwnym przypadku $synch(t) = false$.

³Próba pobrania elementu z pustego bufora jest traktowana jako błąd i jest wykrywana w czasie weryfikacji programu.

Tranzycje możliwe i gotowe do wykonania

Niech $t_i \in T_i$, $s = (q_1, \dots, q_n, v, \mu)$ i $l = (q_1, \dots, q_n, v)$. Powiemy, że proces o numerze i może wykonać tranzycję t_i w stanie s , co oznaczamy przez $enabled_i(t_i, s)$, jeżeli stan źródłowy tej tranzycji jest równy q_i i dozór tej tranzycji jest prawdziwy przy wartościowaniu v , a zatem:

$$enabled_i(t_i, s) \stackrel{def}{=} source(t_i) = q_i \wedge \mathcal{B}(guard(t_i), v) = tt$$

Zauważmy, że wartościowanie opóźnień nie ma wpływu na to, czy proces może wykonać daną tranzycję. Dlatego możemy powiedzieć, że proces o numerze i może wykonać tranzycję t_i w konfiguracji l , co oznaczamy przez $enabled_i(t_i, l)$ i definiujemy:

$$enabled_i(t_i, l) \stackrel{def}{=} source(t_i) = q_i \wedge \mathcal{B}(guard(t_i), v) = tt$$

Tranzycja $t_i \in T_i$ jest *możliwa do wykonania* w stanie s (w konfiguracji l), jeżeli może być wykonana przez proces i w stanie s (w konfiguracji l) i każdy inny proces zawierający etykietę $label(t_i)$ może wykonać tranzycję o tej etykiecie w stanie s (w konfiguracji l).

$$\begin{aligned} enabled(t_i, s) &\stackrel{def}{=} enabled_i(t_i, s) \wedge \forall_{j \in \Gamma(label(t_i)) \setminus \{i\}} \exists_{t_j \in T_j} enabled_j(t_j, s) \wedge label(t_j) = label(t_i) \\ enabled(t_i, l) &\stackrel{def}{=} enabled_i(t_i, l) \wedge \forall_{j \in \Gamma(label(t_i)) \setminus \{i\}} \exists_{t_j \in T_j} enabled_j(t_j, l) \wedge label(t_j) = label(t_i) \end{aligned}$$

Podobnie powiemy, że tranzycja t_i jest *gotowa do wykonania przez proces i* w stanie s , co oznaczamy przez $fireable_i(t_i, s)$, jeżeli proces i może wykonać tranzycję t_i w stanie s i wartość opóźnienia w stanie źródłowym tranzycji należy do przedziału określonego przez jej dozwolone opóźnienie, a więc:

$$fireable_i(t_i, s) \stackrel{def}{=} enabled(t_i, s) \wedge \mu(source(t_i)) \in delay(t_i)$$

Tranzycja $t_i \in T_i$ jest *gotowa do wykonania* w stanie s , jeżeli jest gotowa do wykonania przez proces i w stanie s i dla każdego innego procesu zawierającego etykietę $label(t_i)$ istnieje tranzycja gotowa do wykonania w stanie s o tej etykiecie.

$$fireable(t_i, s) \stackrel{def}{=} fireable_i(t_i, s) \wedge \forall_{j \in \Gamma(label(t_i)) \setminus \{i\}} \exists_{t_j \in T_j} fireable_j(t_j, s) \wedge label(t_j) = label(t_i)$$

Powiemy wreszcie, że tranzycja $t_i \in T_i$ jest *przeterminowana* w stanie s , jeżeli jest możliwa do wykonania i zostało przekroczone górne ograniczenie na czas jej wykonania.

$$expired(t_i, s) \stackrel{def}{=} enabled(t_i, s) \wedge \mu(source(t_i)) \notin upper_delay(t_i)$$

Semantyka

Dla $j \in Z \subseteq \{1, \dots, n\}$ przez $\langle a_j \rangle_{j \in Z}$ oznaczamy ciąg akcji a_j uporządkowany rosnąco według numerów indeksów.⁴

Definicja 3.4 *Semantyką programu $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$ dla danego wartościowania początkowego $v^0 : V \cup B \rightarrow \Omega$ jest etykietowany system tranzycyjny $\mathcal{TS} = (S, s^0, \Lambda, \longrightarrow)$ zdefiniowany następująco:*

⁴Kolejność wykonania akcji nie ma znaczenia (każdy przeplot prowadzi do tego samego wartościowania zmiennych i buforów), ponieważ założyliśmy (punkt 3.1.1, str. 24), że zmienne dzielone ani bufony nie są modyfikowane przez tranzycje synchroniczne.

- $S = Q_1 \times \dots \times Q_n \times \Omega^{V \cup B} \times \mathbb{R}_+^Q$ jest zbiorem stanów,
- $s^0 = (q_1^0, \dots, q_n^0, v^0, \mu^0) \in S$ jest stanem początkowym,
- $\Lambda = \bigcup_{i=1}^n \Gamma_i \cup \mathbb{R}_+$ jest zbiorem etykiet,
- $\longrightarrow \subseteq S \times \Lambda \times S$ jest relacją przejścia, czyli najmniejszą relacją spełniającą warunki:
 - niech $s = (q_1, \dots, q_n, v, \mu)$ i $s' = (q'_1, \dots, q'_n, v', \mu')$, dla $\gamma \in \Gamma$ $s \xrightarrow{\gamma} s'$ wtedy i tylko wtedy, gdy istnieje zbiór tranzycji $T' = \{t_i \mid i \in \Gamma(\gamma)\} \subseteq T$ taki, że:
 - * $\text{fireable}_i(t_i, s)$, $\text{label}(t_i) = \gamma$ i $\text{target}(t_i) = q'_i$ dla każdej tranzycji $t_i \in T'$,
 - * $v' = \mathcal{J}(v, \langle \text{action}(t_i) \rangle_{t_i \in T'})$,
 - * $\mu' = \mu[\{q'_i \mid i \in \Gamma(\gamma)\} := 0]$,
 - * $q'_j = q_j$ dla $j \in \{1, \dots, n\} \setminus \Gamma(\gamma)$,
 - niech $s = (q_1, \dots, q_n, v, \mu)$ i $s' = (q_1, \dots, q_n, v, \mu + \delta)$, dla $\delta \in \mathbb{R}_+$ $s \xrightarrow{\delta} s'$ wtedy i tylko wtedy, gdy $\delta > 0$ i dla każdej tranzycji $t \in T$ zachodzi

$$\neg \text{enabled}(t, s) \vee (\neg \text{urgent}(t) \wedge \neg \text{expired}(t, s'))$$

lub $\delta = 0$.

W stanie początkowym wszystkie bufory są puste, a zmienne mają wartości początkowe określone przez wartościowanie v^0 . Będąc w stanie $s = (q_1, \dots, q_n, v, \mu)$ system może:

- Wykonać tranzycje o etykiecie $\gamma \in \Gamma$ (przejście akcyjne), jeżeli są one gotowe dla wszystkich procesów o numerach ze zbioru $\Gamma(\gamma)$. Stany kontrolne procesów wykonujących tranzycje zmieniają się na stany docelowe tych tranzycji, a stany kontrolne pozostałych procesów pozostają bez zmian. Wartości zmiennych i buforów zmieniają się zgodnie ze strukturą akcji tranzycji. Wartości opóźnień nie zostają zwiększone, natomiast opóźnienia dla stanów q'_i gdzie $i \in \Gamma(\gamma)$ zostają wyzerowane.
- Pozwolić, żeby minęło $\delta > 0$ jednostek czasu (przejście czasowe) pod warunkiem, że nie ma możliwości do wykonania tranzycji, która jest pilna lub która po upływie tego czasu byłaby przeterminowana. Opóźnienie dla każdego stanu jest zwiększane o δ . System zawsze może wykonać tranzycję czasową o długości 0.

Przebiegi programu

s -przebiegiem lub *wykonaniem programu* \mathcal{P} nazywamy nieskończony ciąg stanów:

$$s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} \dots$$

gdzie $s_i \in S$ i $\lambda_i \in \Gamma \cup \mathbb{R}_+$ dla każdego $i \geq 0$ oraz $s_0 = s$. Zauważmy, że skoro λ_i może być równe 0, to dwie kolejne tranzycje akcyjne mogą być wykonane bezpośrednio jedna po drugiej (bez upływu czasu pomiędzy nimi). Wprowadzimy więc pojęcie przebiegu postępującego. Przebieg programu $s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} \dots$ nazywamy *postępującym*, jeżeli suma $\sum_{\{i \in \mathbb{N} \mid \lambda_i \in \mathbb{R}_+\}} \lambda_i$ jest nieskończona. Założenie o postępowości jest naturalne, ponieważ w

rzeczywistości nie mamy wpływu na upływ czasu (nie możemy go w żaden sposób ograniczać), a poza tym żaden rzeczywisty proces nie może wykonywać swoich akcji nieskończenie szybko, czyli nie może wykonać nieskończonej liczby tranzycji akcyjnych w skończonym czasie.

W weryfikacji modelowej często bierze się pod uwagę tylko te przebiegi programu, które spełniają warunki uczciwości (ang. *fairness conditions*), definiowane zazwyczaj w następujący sposób:

- jeżeli jakaś tranzycja, jest nieskończenie często gotowa do wykonania, to będzie nieskończenie często wykonana (tak zwana *silna uczciwość*, ang. *strong fairness*),
- jeżeli jakaś tranzycja, jest nieskończenie długo gotowa do wykonania, to będzie nieskończenie często wykonana (tak zwana *słaba uczciwość*, ang. *weak fairness*).

Niech $\sigma = s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} \dots$, gdzie $\lambda_i \in \Gamma \cup \mathbb{R}_+$ dla każdego $i \geq 0$. Dla $\lambda_i \in \Gamma$ przez $trans_\sigma(\lambda_i) \subseteq T$ oznaczmy zbiór tranzycji (jednoelementowy, jeżeli λ_i jest etykietą lokalną) wykonywanych podczas przejścia λ_i . Powiemy, że przebieg σ jest *silnie uczciwy* względem tranzycji należących do zbioru $T' \subseteq T$, jeżeli dla każdej tranzycji $t \in T'$ jest spełniony następujący warunek:

$$(\forall_{k \geq 0} \exists_{j \geq k} \text{fireable}(t, s_j)) \Rightarrow (\forall_{k \geq 0} \exists_{j \geq k} t \in trans_\sigma(\lambda_j)).$$

Przebieg σ jest *słabo uczciwy* względem tranzycji należących do zbioru $T' \subseteq T$, jeżeli dla każdej tranzycji $t \in T'$ jest spełniony następujący warunek:

$$(\exists_{k \geq 0} \forall_{j \geq k} \text{fireable}(t, s_j)) \Rightarrow (\forall_{k \geq 0} \exists_{j \geq k} t \in trans_\sigma(\lambda_j)).$$

W rozprawie nie przyjmujemy żadnych założeń na temat uczciwości ani progresywności programów. Prezentowane w kolejnych rozdziałach metody generowania automatów czasowych i redukcji są przedstawione dla najbardziej ogólnej definicji przebiegu programu. Aczkolwiek w rozdziale 5 pokazujemy, że jeżeli ograniczymy nasze rozważania tylko do przebiegów silnie (słabo) uczciwych, to każdy przebieg zredukowanego programu odpowiadający silnie (słabo) uczciwemu przebiegowi oryginalnego programu, będzie również silnie (słabo) uczciwy (i odwrotnie). Podobnie dla założenia o progresywności.

Dla $s = (q_1, \dots, q_n, v, \mu) \in S$ i $\delta, \delta' \in \mathbb{R}_+$ niech $s + \delta$ oznacza stan $(q_1, \dots, q_n, v, \mu + \delta)$. *Połączeniem* dwu tranzycji czasowych $s \xrightarrow{\delta} s + \delta$ i $s + \delta \xrightarrow{\delta'} s + \delta + \delta'$ jest tranzycja czasowa $s \xrightarrow{\delta + \delta'} s + \delta + \delta'$. Podobnie, jeżeli $s \xrightarrow{\delta} s + \delta$, to dla dowolnego $k \in \mathbb{N}_+$ istnieją $\delta_1, \dots, \delta_k \in \mathbb{R}_+$ takie, że $\delta_1 + \dots + \delta_k = \delta$ i $s \xrightarrow{\delta_1} s + \delta_1 \xrightarrow{\delta_2} \dots \xrightarrow{\delta_k} s + \delta$, czyli każda tranzycja czasowa może być *podzielona* na dowolną, ale skończoną liczbę tranzycji czasowych. Nie będziemy rozróżniać przebiegów, z których jeden może powstać z drugiego przez podział lub połączenie tranzycji czasowych.

3.3 Poprawność programu

Dla języka bazowego nie przedstawiamy nowej logiki ani systemu dowodzenia. Do wyrażenia własności programu można użyć istniejących logik temporalnych takich jak CTL* opisana w rozdz. 2.

Model

Niech $\mathcal{TS} = (S, s^0, \Lambda, \longrightarrow)$ będzie etykietowanym systemem tranzycyjnym dla programu $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$. Możemy definiować następujące zmienne zdaniowe dla programu \mathcal{P} :

- $p_{e_1 \sim e_2}$, gdzie $e_1, e_2 \in Exp(V)$ i \sim jest operatorem relacyjnym,
- $p_{empty(b)}$, gdzie $b \in B$,
- $p_{i,q}$, gdzie $1 \leq i \leq n$ i $q \in Q_i$.

Dla pewnego zbioru zmiennych zdaniowych PV , wartość funkcji wartościującej $\mathcal{V} : S \rightarrow 2^{PV}$ w stanie $s = (q_1, \dots, q_n, v, \mu)$ programu \mathcal{P} definiujemy w następujący sposób:

- $p_{e_1 \sim e_2} \in \mathcal{V}(s)$ wtedy i tylko wtedy, gdy $\mathcal{B}(e_1 \sim e_2, v) = tt$,
- $p_{empty(b)} \in \mathcal{V}(s)$ wtedy i tylko wtedy, gdy $v(b) = \varepsilon$,
- $p_{i,q} \in \mathcal{V}(s)$ wtedy i tylko wtedy, gdy $q_i = q$.

Zmienna zdaniowa $p_{e_1 \sim e_2}$ etykietuje stan s , czyli zdanie $e_1 \sim e_2$ jest prawdziwe w stanie s , jeżeli wartościowanie zmiennych i buforów spełnia warunek $e_1 \sim e_2$. Podobnie, zdanie $p_{empty(b)}$ jest prawdziwe w stanie s , jeżeli w tym stanie bufor b jest pusty, a $p_{i,q}$ jest prawdziwe w stanie s , jeżeli stan kontrolny i -tego procesu w s (q_i) jest równy q .

Modelem programu \mathcal{P} nazywamy parę

$$\mathcal{M} = (\mathcal{TS}, \mathcal{V}).$$

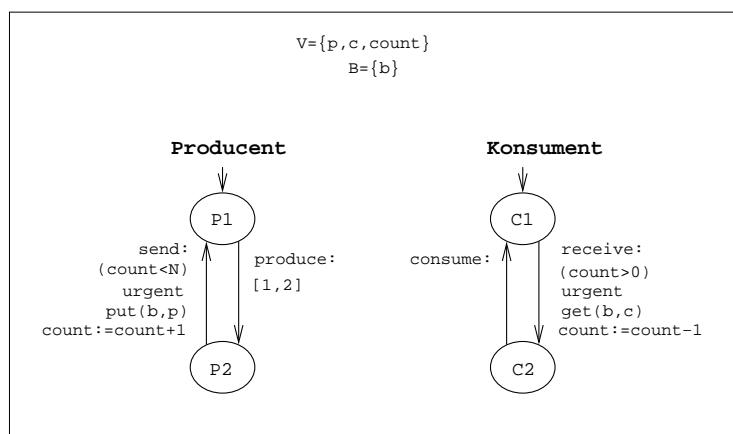
3.4 Przykłady

Dla zwiększenia czytelności wszystkie przykłady programów w języku bazowym będziemy przedstawiać w postaci rysunków, gdzie stany programu są reprezentowane przez owale, a tranzycje przez strzałki etykietowane odpowiednimi atrybutami. Małe strzałki wskazują początkowe stany kontrolne poszczególnych procesów.

3.4.1 Producent i konsument

Przedstawimy najpierw bardzo prosty program, którym zilustrujemy generowanie automatów czasowych (w rozdziale 4). Na rys. 3.1 przedstawiono system składający się z procesów *Producent* i *Konsument*, które do komunikacji wykorzystują bufor b . W systemie są oprócz tego trzy zmienne: zmienna *count* przechowująca liczbę elementów w buforze, używana i definiowana przez obydwie procesy oraz zmienne reprezentujące porcje danych procesu *Producent* (p) i *Konsument* (c).

Proces *Producent* wykonuje na przemian dwa rodzaje czynności: produkuje porcje danych (tranzycja o etykiecie *produce*) i wysyła je do procesu *Konsument* (*send*). Produkcja danych zajmuje co najmniej 1 i nie więcej niż 2 jednostki czasu. Jeżeli tylko bufor nie jest pełny ($count < N$), to proces natychmiast po wyprodukowaniu danych (informuje o tym znacznik *urgent*) umieszcza je w buforze (*put*(b, p)) i zwiększa licznik ($count := count + 1$).



Rysunek 3.1: Producent i konsument

Proces *Konsument* cyklicznie pobiera porcje danych z bufora (tranzycja o etykiecie *receive*) i konsumuje je (*consume*). Pobranie porcji odbywa się natychmiast (*urgent*), jeżeli tylko bufor nie jest pusty ($count > 0$). *Konsument* wyjmując dane z bufora ($get(b,c)$) zmniejsza licznik ($count := count - 1$). Konsumpcja danych trwa dowolnie długo.

Przykładowe wartościowanie początkowe v^0 dla programu jest następujące: $v^0(count) = v^0(p) = v^0(c) = 0$ i $v^0(b) = \varepsilon$.

Przykłady własności

Dla programu zdefiniujemy następujące zmienne zdaniowe:

- $p_1 := p_{count \leq N}$, czyli liczba elementów w buforze ($count$) nie przekracza N ,
- $p_2 := p_{2.C1}$, czyli proces o numerze 2 (*Konsument*) jest w stanie C1,
- $p_3 := p_{2.C2}$, czyli proces o numerze 2 (*Konsument*) jest w stanie C2.

Formuła

$$AG p_1$$

jest przykładem własności bezpieczeństwa⁵. Natomiast formuła

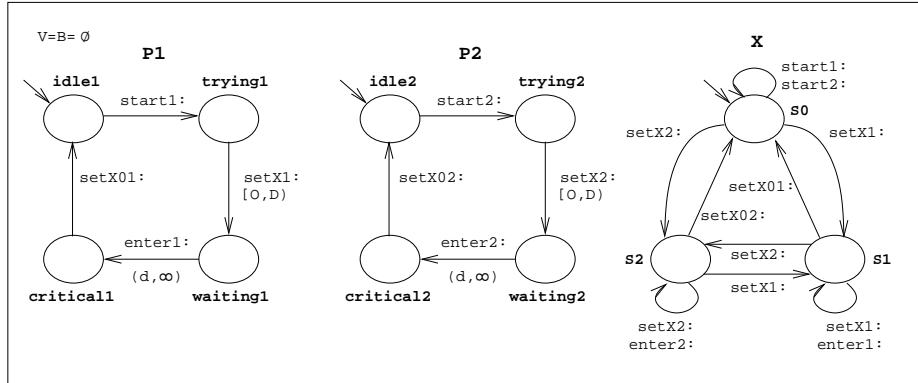
$$AG (p_2 \Rightarrow AF p_3)$$

która mówi, że zawsze, gdy *Konsument* jest gotowy do odebrania porcji danych, to w końcu je dostanie, jest przykładem własności żywotności⁶. Obie formuły są prawdziwe. Natomiast formuła

$$AG (p_3 \Rightarrow AF p_2)$$

⁵ *Bezpieczeństwo* oznacza, że "nigdy nie zdarzy się nic złego".

⁶ *Żywotność* oznacza, że "zawsze w końcu stanie się coś pożądanego".



Rysunek 3.2: Synchroniczny protokół Fischera

jest prawdziwa tylko wtedy, gdy przyjmiemy dodatkowe założenia o uczciwości, a więc w czasie weryfikacji będziemy brali pod uwagę jedynie przebiegi spełniające warunki uczciwości (słabe lub silne), czyli takie w których proces *Konsument* nie może przebywać w nieskończoność w stanie $C2$.

3.4.2 Protokół Fischera wzajemnego wykluczania

Następnym przykładem jest protokół Fischera wzajemnego wykluczania opisany w punkcie 2.1.6. Pokażemy tutaj dwa sposoby przedstawienia tego protokołu w języku bazowym.

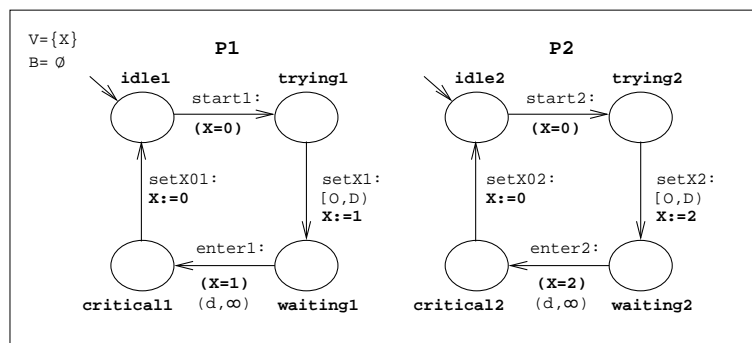
Pierwszy ze sposobów wykorzystuje mechanizm wspólnego wykonywania tranzycji przez wiele procesów (tak jak w automatach czasowych). Na rys. 3.2 przedstawiono trzy procesy: $P1$, $P2$ i X , których działanie omówiono dokładnie w rozdz. 2.1.6. Jedyna różnica polega na tym, że ograniczenia na czas wykonywania tranzycji są reprezentowane przez dozwolone opóźnienia, a nie przez warunki zegarowe.

Oczywiście protokół Fischera może być opisany w bardziej naturalny sposób, w którym zmienna sterująca protokołu jest reprezentowana prosto przez zmienną programu (dla $v^0(X) = 0$), a nie przez proces, co pokazano na rys. 3.3. Tak przedstawiony protokół daje się w bardzo łatwy sposób uogólnić na dowolną liczbę procesów (wystarczy dodać nowe procesy, których działanie jest analogiczne do działania procesów $P1$ i $P2$), co nie jest prawdą w przypadku opisu synchronicznego.

Formuła

$$AG \neg(c_1 \wedge c_2)$$

gdzie $c_1 := p1.critical1$ i $c_2 := p2.critical2$, wyraża własność wzajemnego wykluczania (nigdy dwa procesy nie przebywają jednocześnie w swoich sekcjach krytycznych). Ciekawą cechą tego protokołu jest to, że prawdziwość powyższej formuły zależy od wartości progów czasowych D i d . Formuła jest prawdziwa wtedy i tylko wtedy, gdy $D < d$.



Rysunek 3.3: Asynchroniczny protokół Fischera

3.4.3 Protokół zmieniającego się bitu

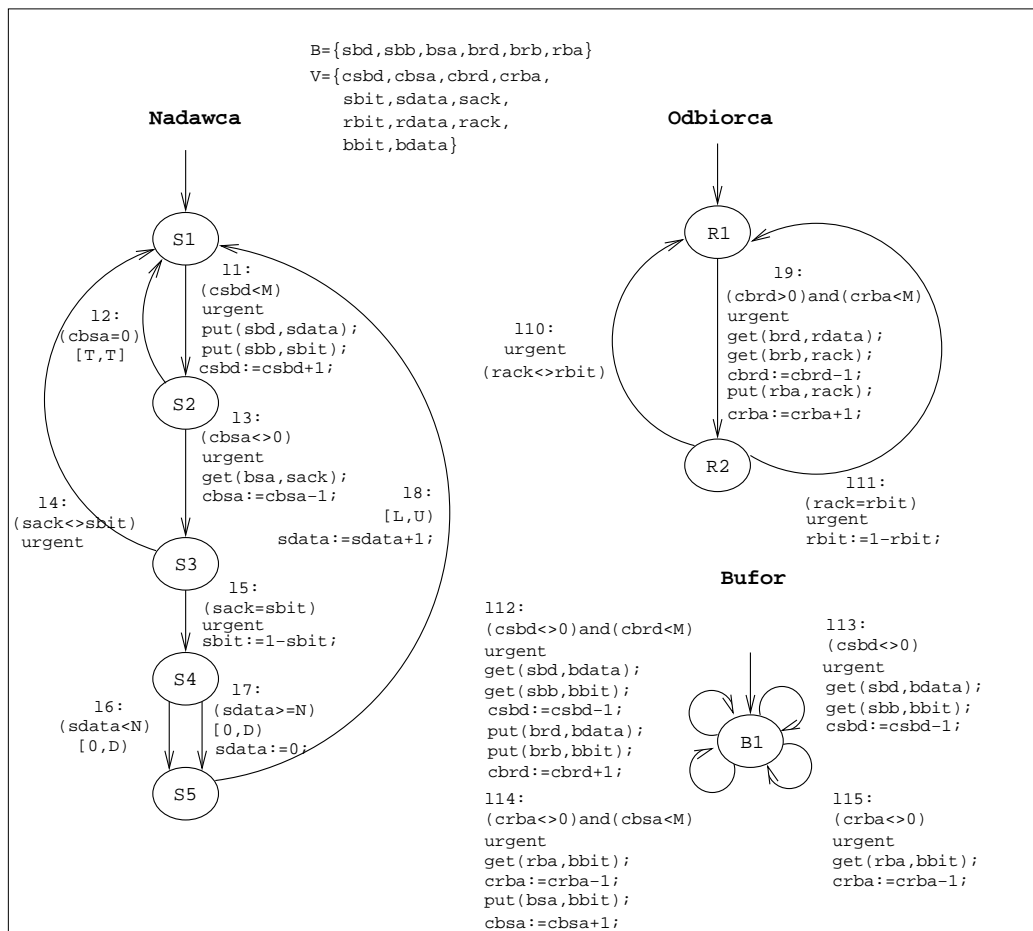
Opis protokołu

Na koniec przedstawimy trochę większy przykład, którym zilustrujemy metodę cięcia w rozdziale 5. Protokół zmieniającego się bitu (ang. *Alternating Bit Protocol*, w skrócie ABP) [BSW69] jest protokołem komunikacyjnym, który umożliwia transmisję danych w środowisku, w którym nie ma gwarancji dotarcia danych do celu. Wykorzystany jest tutaj mechanizm potwierdzania i retransmisji danych. W protokole każda porcja danych i każde potwierdzenie otrzymania danych opatrzone jest jednobitowym znacznikiem, na podstawie którego nadawca rozpoznaje, czy porcja danych została odebrana, czy też została zgubiona i wtedy wysyła ją jeszcze raz.

Rysunek 3.4 przedstawia jedną z wersji tego protokołu opisaną w języku bazowym. Program składa się z trzech procesów: *Nadawcy*, *Odbiorcy* i *Bufora*. Proces *Nadawca* (*Odbiorca*) odpowiada stronie wysyłającej (odbierającej) komunikaty. Zawodne łącze jest reprezentowane przez proces *Bufor* i 6 buforów komunikacyjnych zawierających odpowiednio:

- *sbd* — dane przesyłane między *Nadawcą* a *Buforem*,
- *sbb* — bity przesyłane między *Nadawcą* a *Buforem*,
- *bsa* — potwierdzenia przesyłane między *Buforem* a *Nadawcą*,
- *brd* — dane przesyłane między *Buforem* a *Odbiorcą*,
- *brb* — bity przesyłane między *Buforem* a *Odbiorcą*,
- *rba* — potwierdzenia przesyłane między *Odbiorcą* a *Buforem*.

Zmienne *csbd*, *csba*, *cb rd*, *crba* odpowiadają ilości komunikatów w danym buforze i są odpowiednio modyfikowane przy wysyłaniu i odbieraniu komunikatów z tych buforów. Bufor *sbb* (ani *brb*) nie ma licznika, ponieważ liczba wiadomości w buforze *sbb* jest zawsze równa liczbie wiadomości w buforze *sbd*, czyli wartości licznika *csbd* (podobnie dla buforów *brb* i *brd*).



Rysunek 3.4: Protokół zmieniającego się bitu (ABP)

Proces *Bufor* modeluje dwa rodzaje zachowania: poprawne przekazanie danych lub ich zgubienie. W tranzycji *l12 Bufor* pobiera dane ($get(sbd, bdata)$) razem ze znacznikiem ($get(sbb, bbit)$) od *Nadawcy* i przekazuje je w niezmienionej postaci do *Odbiorcy*. Natomiast w tranzycji *l13 Bufor* pobiera dane i znacznik od *Nadawcy* i nie przekazuje ich dalej. Analogicznie jest w przypadku przekazywania potwierdzenia od *Odbiorcy* do *Nadawcy* (tranzycje *l14* i *l15*).

Proces *Nadawca* umieszcza dane (*sdata*) i znacznik (*sbit*) w odpowiednich buforach (o ile nie są pełne). Następnie czeka na potwierdzenie. Jeżeli nie otrzyma potwierdzenia w ciągu T jednostek czasu, to ponownie przesyła tę samą parę: dane i znacznik. Jeżeli otrzyma potwierdzenie (w postaci jednego bitu), to sprawdza, czy wartość potwierdzenia (*sack*) jest równa znacznikowi (*sbit*). Jeżeli nie jest, to ponownie wysyła dane i znacznik. Jeżeli wartość potwierdzenia jest zgodna z oczekiwaną, to znaczy, że ta porcja danych została poprawnie odebrana przez *Odbiorcę* i *Nadawca* zmienia wartość swojego bitu (*sbit*) na przeciwną (z 0 na 1 i odwrotnie). Następnie *Nadawca* generuje nowe dane (reprezentowane przez kolejne liczby całkowite z zakresu od 1 do N). Generowanie nowych danych zajmuje co najmniej L i co najwyżej U jednostek czasu. Od tego momentu działanie *Nadawcy* powtarza się dla nowych wartości danych i znacznika.

Proces *Odbiorca* po odebraniu danych (*rdata*) i znacznika (*rack*) wysyła potwierdzenie równe wartości znacznika. Następnie sprawdza, czy wartość znacznika jest równa wartości bitu kontrolnego (*rbit*) i jeżeli tak, to zmienia wartość tego bitu na przeciwną.

Wartościowanie początkowe dla programu jest następujące: $v^0(sdata) = 1$, $v^0(y) = 0$ dla $y \in V \setminus \{sdata\}$ i $v^0(b) = \varepsilon$ dla $b \in B$.

Zauważmy, że program z rys. 3.4 jest strukturalny, ponieważ dla każdego procesu istnieje podział na zbiory F_i i B_i spełniające warunki definicji 3.2: dla procesu *Nadawcy* zbiór $F_1 = \{l1, l3, l5, l6, l7\}$, a $B_2 = \{l2, l4, l8\}$, dla procesu *Odbiorcy* $F_2 = \{l9\}$ i $B_2 = \{l9, l10\}$, a dla procesu *Bufora* $F_3 = \emptyset$ i $B_2 = \{l12, l13, l14, l15\}$.

Przykłady własności

Protokół ABP jest poprawny tylko wtedy, gdy wartość zmiennej *sbit* (używana w procesie *Nadawcy*) jest równa wartości zmiennej *rbit* (używanej w procesie *Odbiorcy*).

Sformułujmy następującą własność: jeżeli procesy *Nadawcy* i *Odbiorcy* są w swoich początkowych stanach kontrolnych (czyli w stanie początkowym i wtedy, gdy obydwa procesy zdążyły zmienić wartości swoich bitów) i wartość zmiennej *sbit* wynosi 0, to wartość zmiennej *rbit* również wynosi 0 (podobnie możemy sprawdzić dla wartości 1).⁷ Zdefiniujmy następujące zmienne zdaniowe:

- $s_1 := p_{1.S1}$ ($s_2 := p_{1.S2}$), czyli proces o numerze 1 (*Nadawca*) jest w stanie $S1$ ($S2$),
- $r_1 := p_{2.R1}$ ($r_2 := p_{2.R2}$), czyli proces o numerze 2 (*Odbiorca*) jest w stanie $R1$ ($R2$),
- $sbit0 := p_{sbit=0}$,
- $rbit0 := p_{rbit=0}$.

⁷ Powód, dla którego nie definiujemy zmiennej zdaniowej $p_{sbit=rbit}$ zostanie podany w punkcie 4.3.6.

Formuła φ wyrażająca naszą własność ma postać

$$\varphi = AG((s_1 \wedge r_1 \wedge sbit0) \Rightarrow rbit0)$$

Badaniem tej własności zajmiemy się w kolejnych rozdziałach. Następną formułą, która jest przykładem własności żywotności

$$\psi = AG(s_2 \Rightarrow AF r_2)$$

mówi, że jeżeli *Nadawca* wysłał dane, to w końcu dotrą one do *Odbiorcy*. Zauważmy, że ta własność jest prawdziwa tylko przy założeniu silnej uczciwości, w przeciwnym razie jest możliwy taki nieskończony przebieg programu, w którym *Bufor* gubi wszystkie pakiety przekazywane przez *Nadawcę* (tranzycja *l12* nie jest wykonana mimo, że nieskończenie często jest możliwa do wykonania).

3.5 Podsumowanie rozdziału

Od języków opisu systemów czasowych takich jak Estelle [ISO97] czy SDL [MT01], które możemy określić mianem języków wysokiego poziomu, język bazowy różni się tym, że ma uproszczoną składnię. Nie posiada typowych dla języków programowania instrukcji *if-then-else*, czy *while*. Jednak takie instrukcje można łatwo “wyrazić” w języku bazowym (warunek instrukcji *if-then-else* wykorzystując dozory tranzycji, a pętlę poprzez odpowiedni ciąg stanów i tranzycji tworzący cykl). Z drugiej strony takie zaprojektowanie języka ułatwia tłumaczenie do automatów czasowych. Poza tym język bazowy nie różni się siłą wyrazu od języków wysokiego poziomu, w których można nakładać ograniczenia na czas wykonywania pewnych operacji, co znaczy że każde zachowanie opisane w językach wysokiego poziomu można także opisać w języku bazowym.

Mimo, że język bazowy jest dosyć prosty, to jednak udostępnia wszystkie podstawowe mechanizmy komunikacji i synchronizacji systemów współbieżnych. Dzięki temu możliwe jest dosyć wygodne tłumaczenie z innych języków, czego dowodem może być fakt, że takie tłumaczenia powstały i nadal powstają dla języków:

- Estelle (praca magisterska Wiktora Miszurisa pt. “Tłumaczenie Estelle do języka bazowego”, obroniona w 2004 roku),
- UML (praca magisterska Jakuba Kowalskiego pt. “Translacja z języka UML do automatów czasowych”, obroniona w 2005 roku),
- Promela [NDJ06],
- Java [OWS06],
- SDL (praca magisterska Pawła Kowalskiego pt. “Translacja z języka SDL do automatów czasowych”, w przygotowaniu).

Język bazowy został także wykorzystany do opisu protokołów kryptograficznych uwzględniających zależności czasowe [JP06a, JP07].

Opis składni konkretnej języka bazowego (znanego jako IL od ang. *Intermediate Language*) i przykłady specyfikacji w tym języku można znaleźć pod adresem:

<http://www.mimuw.edu.pl/~janowska/il>

Rozdział 4

Generowanie automatów czasowych dla języka bazowego

W rozdziale opisujemy sposób budowy automatów czasowych na podstawie programu w języku bazowym. Na początku przedstawiamy jak za pomocą warunków na zegarach można wyrazić dopuszczalne opóźnienia i pilność tranzycji. Pokażemy następnie jak skonstruować jeden automat czasowy odpowiadający programowi. Będziemy go nazywać *globalnym* automatem czasowym. W kolejnym punkcie rozdziału zdefiniujemy zbiór automatów czasowych, w którym poszczególne automaty odpowiadają składowym programom. Podamy również sposoby redukcji liczby zegarów w generowanych automatach.

4.1 Zegary

Niech $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$ będzie programem, a $P_i = (id_i, Q_i, q_i^0, \Sigma_i, T_i)$ procesem programu \mathcal{P} . Dla stanu kontrolnego $q \in Q_i$ przez $nut(q)$ ¹ będziemy oznaczać liczbę tranzycji pilnych wychodzących z q , a przez $nut(P_i) = \max_{q \in Q_i} nut(q)$ — maksymalną liczbę tranzycji pilnych wychodzących z jednego stanu i -tego procesu. Każdemu procesowi P_i odpowiada zbiór zegarów $X_i = \{x_1^i, \dots, x_k^i\}$, gdzie $k = nut(P_i) + 1$ przy czym zbiory X_i i X_j są rozłączne dla $i \neq j$. Zbiór wszystkich zegarów X dla programu \mathcal{P} będzie równy sumie zbiorów X_i dla $1 \leq i \leq n$.

Warunki na zegarach modelują ograniczenia czasowe wynikające z dozwolonego opóźnienia i pilności tranzycji. Przypomnijmy, że jeżeli tranzycja t nie jest pilna i jej dozwolone opóźnienie $delay(t)$ wynosi $\langle d_1, d_2 \rangle$, gdzie $d_1 \in \mathbb{N}$ i $d_2 \in \mathbb{N} \cup \{\infty\}$ oraz \langle jest znakiem “(” lub “[” i \rangle jest znakiem “)” lub “]”, to t może zostać wykonana tylko wtedy, gdy upłynie więcej niż d_1 , ale mniej niż d_2 jednostek czasu od momentu, gdy proces wykonał tranzycję do stanu źródłowego tranzycji t (mówimy, że proces *wszedł* do stanu źródłowego tranzycji t). Tranzycja pilna jest wykonana natychmiast, gdy tylko stanie się możliwa do wykonania (lub nie jest wykonana wcale).

Każdej tranzycji odpowiada jeden zegar, przy czym przydział zegarów jest następujący.

¹Skrót od angielskiego *number of urgent transitions*.

Każdej tranzycji procesu P_i , która nie jest pilna, przydzielamy zegar x_1^i . Zegar x_1^i jest zerowany podczas wykonania każdej tranzycji automatu, której odpowiada jakaś tranzycja procesu P_i , więc jego wartość odpowiada czasowi przebywania procesu w bieżącym stanie kontrolnym (czyli w stanie, w którym proces P_i znajduje się w danym momencie). Dozwolone opóźnienie tranzycji, która nie jest pilna, jest modelowane przez odpowiednie warunki na zegarze x_1^i . Pozostałe zegary są przydzielone tranzycjom pilnym procesu P_i w ten sposób, że każde dwie tranzycje pilne wychodzące z tego samego stanu kontrolnego mają przypisane różne zegary. Taki podział istnieje, bo przyjęliśmy, że liczba zegarów procesu P_i jest równa $\text{nut}(P_i)+1$. Wartość zegara odpowiadającego tranzycji pilnej odpowiada czasowi jaki minął od momentu umożliwienia tej tranzycji, ponieważ zegar jest zerowany wtedy, gdy tranzycja staje się możliwa do wykonania.

Funkcje c , constraint i upper_constr

Niech $T_i^u = \{t \in T_i \mid \text{urgent}(t)\}$ będzie zbiorem tranzycji pilnych procesu P_i i $T^u = \bigcup_{i=1}^n T_i^u$ będzie zbiorem tranzycji pilnych programu. Niech $c : T \rightarrow X$ będzie funkcją przyporządkowującą zegary tranzycjom taką, że dla każdej tranzycji $t \in T_i \setminus T_i^u$, $c(t) = x_1^i$ i dla każdej pary tranzycji $t_1, t_2 \in T_i^u$, $c(t_1) \neq x_1^i$, $c(t_2) \neq x_1^i$ i $c(t_1) \neq c(t_2)$.²

Pomocnicza funkcja $\text{constraint} : T \rightarrow \Psi(X)$ każdej tranzycji przypisuje odpowiednie ograniczenie zegarów. Niech $d_1, d_2 \in \mathbb{N}$.

$$\text{constraint}(t) \stackrel{\text{def}}{=} \begin{cases} d_1 < c(t) \wedge c(t) < d_2 & \text{jeżeli } \text{delay}(t) = (d_1, d_2), \\ d_1 < c(t) \wedge c(t) \leq d_2 & \text{jeżeli } \text{delay}(t) = (d_1, d_2], \\ d_1 < c(t) & \text{jeżeli } \text{delay}(t) = (d_1, \infty), \\ d_1 \leq c(t) \wedge c(t) < d_2 & \text{jeżeli } \text{delay}(t) = [d_1, d_2), \\ d_1 \leq c(t) \wedge c(t) \leq d_2 & \text{jeżeli } \text{delay}(t) = [d_1, d_2], \\ d_1 \leq c(t) & \text{jeżeli } \text{delay}(t) = [d_1, \infty), \text{ gdzie } d_1 \neq 0, \\ c(t) \leq 0 & \text{jeżeli } \text{delay}(t) = [0, \infty) \text{ i } \text{urgent}(t) = \text{true}, \\ \text{true} & \text{jeżeli } \text{delay}(t) = [0, \infty) \text{ i } \text{urgent}(t) = \text{false}. \end{cases}$$

Aby wymusić wykonanie tranzycji przed upłynięciem jakiegoś czasu, musimy skonstruować odpowiedni niezmiennik lokacji źródłowej tej tranzycji. Funkcja $\text{upper_constr} : T \rightarrow \Psi(X)$, która będzie potrzebna do skonstruowania niezmiennika, jest zdefiniowana w następujący sposób.³ Niech $d \in \mathbb{N}$.

$$\text{upper_constr}(t) \stackrel{\text{def}}{=} \begin{cases} c(t) < d & \text{jeżeli } \text{upper_delay}(t) = [0, d), \\ c(t) \leq d & \text{jeżeli } \text{upper_delay}(t) = [0, d], \\ c(t) \leq 0 & \text{jeżeli } \text{upper_delay}(t) = [0, \infty) \text{ i } \text{urgent}(t) = \text{true}, \\ \text{true} & \text{jeżeli } \text{upper_delay}(t) = [0, \infty) \text{ i } \text{urgent}(t) = \text{false}. \end{cases}$$

² Aby ograniczyć liczbę zegarów możemy ten sam zegar przyporządkować wielu tranzycjom wychodzącym z tego samego stanu, jeżeli te tranzycje nie będą jednocześnie możliwe do wykonania i nie przydzielać zegara tranzycjom, które nie mają nałożonych żadnych ograniczeń czasowych. Konstrukcję wykorzystującą mniejszą liczbę zegarów omawiamy w rozdziale 4.4.

³ Definicja górnego dozwolonego opóźnienia (upper_delay) znajduje się na stronie 22.

4.2 Globalny automat czasowy

Globalny automat czasowy konstruujemy dla danego programu i początkowego wartościowania zmiennych i buforów. Lokacja globalnego automatu czasowego odpowiada konfiguracji programu określonej przez stany kontrolne w jakich znajdują się procesy oraz wartościowanie zmiennych i buforów. Każdej lokacji globalnego automatu czasowego odpowiada krotka (q_1, \dots, q_n, v) , gdzie $q_i \in Q_i$ dla $1 \leq i \leq n$ i $v \in \Omega^{V \cup B}$. W szczególności, lokacji początkowej odpowiada konfiguracja programu, w której wszystkie procesy znajdują się w swoich stanach początkowych a wartości zmiennych i buforów są określone przez wartościowanie początkowe v^0 .

Intuicyjnie, w globalnym automacie czasowym istnieje tranzycja o etykiecie γ wychodząca z pewnej lokacji, jeżeli w programie dla każdego $i \in \Gamma(\gamma)$ ⁴ istnieje tranzycja o etykiecie γ możliwa do wykonania w konfiguracji określonej przez tę lokację. Lokacja docelowa tranzycji w automacie odpowiada konfiguracji programu po wykonaniu tych tranzycji. Warunek umożliwienia tranzycji w automacie (na zegarach) jest koniunkcją warunków czasowych odpowiadających opóźnieniom poszczególnych tranzycji w programie. Do zbioru zegarów do wyzerowania należą zegary mierzące czas przebywania procesu w bieżącym stanie kontrolnym dla wszystkich procesów wykonujących tranzycje oraz zegary odpowiadające tranzyjom pilnym, które nie są możliwe do wykonania w konfiguracji źródłowej, a stają się możliwe do wykonania w konfiguracji docelowej tej tranzycji. Niezmiennik lokacji jest wyznaczony przez górne ograniczenia czasowe tranzycji możliwych do wykonania w konfiguracji odpowiadającej tej lokacji. Jeżeli tranzycja t musi być wykonana przed upłynięciem jakiegoś czasu (lub natychmiast) i jest możliwa do wykonania w danej konfiguracji (lokacji), to w niezmienniku lokacji znajduje się warunek *upper_constr*(t).

Automaty czasowe generujemy przy założeniu, że liczba osiągalnych konfiguracji programu jest skończona. W przeciwnym przypadku liczba lokacji automatu czasowego byłaby nieskończona, a więc taki automat nie byłby poprawnym automatem czasowym.

4.2.1 Definicja automatu globalnego

Definicja 4.1 *Automat czasowy* $TA = (\Sigma, L, l^0, X, E, \mathcal{I})$ dla programu $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$, i wartościowania początkowego $v^0 : V \cup B \rightarrow \Omega$ definiujemy następująco:

- $\Sigma = \bigcup_{i=1}^n \Gamma_i$ jest zbiorem etykiet,
- $L = Q_1 \times \dots \times Q_n \times \Omega^{V \cup B}$ jest zbiorem lokacji,
- $l^0 = (q_1^0, \dots, q_n^0, v^0) \in L$ jest lokacją początkową,
- $X = \bigcup_{i=1}^n \bigcup_{j=1}^{mut(P_i)+1} \{x_j^i\}$ jest zbiorem zegarów,
- $E \subseteq L \times \Sigma \times \Psi(X) \times 2^X \times L$ jest relacją przejścia zdefiniowaną następująco dla $l, l' \in L$, gdzie $l = (q_1, \dots, q_n, v)$ i $l' = (q'_1, \dots, q'_n, v')$:

⁴Przypomnijmy, że $\Gamma(\gamma)$ oznacza zbiór numerów procesów, które w swoim zbiorze etykiet mają etykietę γ (rozd. 3.2).

$l \xrightarrow{\gamma, \psi, Y} l' \in E$ wtedy i tylko wtedy, gdy istnieje etykieta $\gamma \in \Gamma$, zbiór tranzycji $T' = \{t_i \mid i \in \Gamma(\gamma)\} \subseteq T$, taki że

- dla każdej tranzycji $t_i \in T'$ zachodzi $enabled_i(t_i, l)$, $label(t_i) = \gamma$ i $target(t_i) = q'_i$,
- $q'_j = q_j$ dla $j \in \{1, \dots, n\} \setminus \Gamma(\gamma)$,
- $v' = \mathcal{J}(v, \langle action(t_i) \rangle_{t_i \in T'})$,
- $\psi = \bigwedge_{t_i \in T'} constraint(t_i)$,
- $Y = \bigcup_{i \in \Gamma(\gamma)} \{x_1^i\} \cup \{c(t) \mid urgent(t) \wedge \neg enabled(t, l) \wedge enabled(t, l')\}$,

- $\mathcal{I} : L \rightarrow \Psi(X)$ jest zdefiniowany w następujący sposób dla $l \in L$:

$$\mathcal{I}(l) = \bigwedge_{\{t \in T \mid enabled(t, l)\}} upper_constr(t).$$

4.2.2 Funkcja wartościująca dla automatu globalnego

Funkcję wartościującą $\mathcal{V}_{TA} : L \rightarrow 2^{PV}$ dla automatu czasowego TA , zbudowanego dla programu \mathcal{P} i zbioru zmiennych zadaniowych PV , definiujemy w następujący sposób. Dla lokacji $l = (q_1, \dots, q_n, v) \in L$:

- $p_{e_1 \sim e_2} \in \mathcal{V}_{TA}(l)$ wtedy i tylko wtedy, gdy $\mathcal{B}(e_1 \sim e_2, v) = tt$,
- $p_{empty(b)} \in \mathcal{V}_{TA}(l)$ wtedy i tylko wtedy, gdy $v(b) = \varepsilon$,
- $p_{i.q} \in \mathcal{V}_{TA}(l)$ wtedy i tylko wtedy, gdy $q_i = q$.

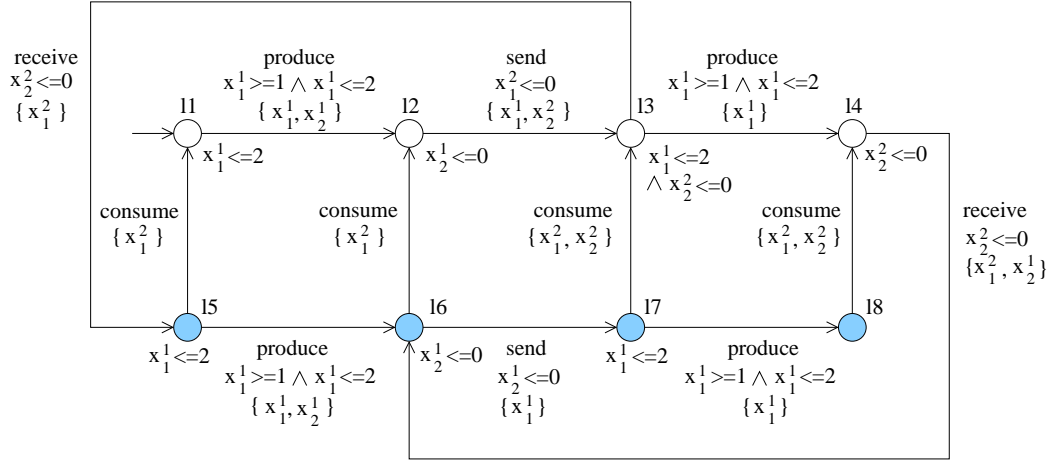
4.2.3 Przykład

Rysunek 4.1 przedstawia automat czasowy dla przykładu producenta i konsumenta (rozdz. 3.4.1), gdzie N (rozmiar bufora) wynosi 1.

Poszczególnym lokacjom odpowiadają następujące konfiguracje programu i zbiory zmiennych zdaniowych (przedstawiono tylko lokacje osiągalne z lokacji początkowej):

$$\begin{aligned} l1 &= (P1, C1, p = 0, c = 0, count = 0, b = \langle \rangle), & \mathcal{V}_{TA}(l1) &= \{p_1, p_2\}, \\ l2 &= (P2, C1, p = 0, c = 0, count = 0, b = \langle \rangle), & \mathcal{V}_{TA}(l2) &= \{p_1, p_2\}, \\ l3 &= (P1, C1, p = 0, c = 0, count = 1, b = \langle 0 \rangle), & \mathcal{V}_{TA}(l3) &= \{p_1, p_2\}, \\ l4 &= (P2, C1, p = 0, c = 0, count = 1, b = \langle 0 \rangle), & \mathcal{V}_{TA}(l4) &= \{p_1, p_2\}, \\ l5 &= (P1, C2, p = 0, c = 0, count = 0, b = \langle \rangle), & \mathcal{V}_{TA}(l5) &= \{p_1, p_3\}, \\ l6 &= (P2, C2, p = 0, c = 0, count = 0, b = \langle \rangle), & \mathcal{V}_{TA}(l6) &= \{p_1, p_3\}, \\ l7 &= (P1, C2, p = 0, c = 0, count = 1, b = \langle 0 \rangle), & \mathcal{V}_{TA}(l7) &= \{p_1, p_3\}, \\ l8 &= (P2, C2, p = 0, c = 0, count = 1, b = \langle 0 \rangle), & \mathcal{V}_{TA}(l8) &= \{p_1, p_3\}. \end{aligned}$$

Przypomnijmy, że p_1 oznacza, że wartość zmiennej $count$ jest nie większa niż $N = 1$, a p_2 (p_3) oznacza, że proces *Konsument* znajduje się w stanie kontrolnym $C1$ ($C2$). Każda lokacja automatu globalnego jest etykietowana zmienną zdaniową p_1 , innymi słowy zdanie reprezentowane przez p_1 jest prawdziwe w każdej lokacji tego automatu. Natomiast zdanie reprezentowane przez p_2 jest prawdziwe w lokacjach oznaczonych kolorem białym, a przez p_3 — szarym.



Rysunek 4.1: Automat czasowy dla przykładu producenta i konsumenta

Zegar x_1^1 odpowiada tranzycji o etykiecie *produce*. Nierówności $x_1^1 \geq 1 \wedge x_1^1 \leq 2$ w warunkach umożliwienia tranzycji o etykiecie *produce* oraz $x_1^1 \leq 2$ w niezmiennikach lokalacji źródłowych tych tranzycji modelują dopuszczalne opóźnienie tranzycji o tej etykiecie w procesie *Producent* ([1, 2]). Podobnie jest dla tranzycji pilnych o etykietach *send* (zegar x_2^2 i warunek $x_2^2 \leq 0$) oraz dla tranzycji o etykiecie *consume* (zegar x_1^1 , brak warunków, bo czas wykonania tranzycji jest dowolny). Zegar x_1^1 jest zerowany podczas każdej tranzycji odpowiadającej tranzycjom procesu *Producent* (o etykietach *produce* i *send*). Podobnie, zegar x_2^2 jest zerowany podczas każdej tranzycji odpowiadającej tranzycjom procesu *Konsument* (o etykietach *consume* i *receive*). Natomiast zegar x_2^2 jest zerowany wtedy, gdy tranzycja o etykiecie *send* staje się możliwa do wykonania, czyli podczas wykonywania tranzycji z lokalacji l1 do l2 (*produce*), z l4 do l6 (*receive*) i z l5 do l6 (*produce*). Zauważmy, że zegar x_2^2 nie jest zerowany przy przejściu z lokalacji z l6 do lokalacji l2 (*consume*), ponieważ tranzycja o etykiecie *send* jest możliwa do wykonania w obydwu konfiguracjach określonych przez lokalacje l6 i l2. Podobnie jest dla zegara x_2^2 i tranzycji o etykiecie *receive*.

4.2.4 Poprawność

Niech $TS_p = \{S_p, s_p^0, \Lambda_p, \longrightarrow_p\}$ będzie systemem tranzycyjnym dla programu $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$, a $TS_a = \{S_a, s_a^0, \Lambda_a, \longrightarrow_a\}$ niech będzie systemem tranzycyjnym dla automatu czasowego $\mathcal{TA} = (\Sigma, L, l^0, X, E, \mathcal{I})$ skonstruowanego dla programu \mathcal{P} zgodnie z def. 4.1. Rozważmy następującą relację między stanami tych systemów.

Definicja 4.2 Niech $\cong_{pa} \subseteq S_p \times S_a$ będzie relacją taką, że dla stanów $s^p = (q_1, \dots, q_n, v, \mu)$ i $s^a = (l, \tau)$, gdzie $l = (q'_1, \dots, q'_n, v')$, zachodzi $s^p \cong_{pa} s^a$ wtedy i tylko wtedy, gdy spełnione są trzy następujące warunki:

1. $q_i = q'_i$ dla każdego $1 \leq i \leq n$ i $v = v'$,
2. $\tau(x_1^i) = \mu(q_i)$ dla każdego $1 \leq i \leq n$,
3. $enabled(t, s^p) \wedge urgent(t) \Rightarrow \tau(c(t)) = 0$ dla każdej tranzycji $t \in T$.

Powiemy, że dwie ścieżki $s_0^p \xrightarrow{\lambda_0} s_1^p \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{j-1}} s_j^p \xrightarrow{\lambda_j} \dots$ w TS_p i $s_0^a \xrightarrow{\lambda_0} s_1^a \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{j-1}} s_j^a \xrightarrow{\lambda_j} \dots$ w TS_a odpowiadają sobie, jeżeli dla każdego $j \geq 0$, $s_j^p \cong_{pa} s_j^a$. Zauważmy, że jeżeli dwie ścieżki odpowiadają sobie, to ich odpowiednie pary sufiksów również sobie odpowiadają.

Lemat 4.3 *Jeżeli $s^p \cong_{pa} s^a$, to dla każdej ścieżki ze stanu s^p istnieje odpowiadająca jej ścieżka ze stanu s^a i dla każdej ścieżki ze stanu s^a istnieje odpowiadająca jej ścieżka ze stanu s^p .*

Dowód:

W dowodzie wykorzystamy konstrukcję indukcyjną. Załóżmy, że dla prefiksu $s_0^p \xrightarrow{\lambda_0} s_1^p \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{k-1}} s_k^p$ mamy zbudowany prefiks $s_0^a \xrightarrow{\lambda_0} s_1^a \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{k-1}} s_k^a$ taki, że $s_0^p = s^p$, $s_0^a = s^a$ i $s_j^p \cong_{pa} s_j^a$ dla każdego $0 \leq j \leq k$. Pokażemy, że :

- (\Rightarrow) dla przejścia $s_k^p \xrightarrow{\lambda_k} s_{k+1}^p$ w TS_p istnieje stan s_{k+1}^a i przejście $s_k^a \xrightarrow{\lambda_k} s_{k+1}^a$ w TS_a takie, że $s_{k+1}^p \cong_{pa} s_{k+1}^a$ oraz
- (\Leftarrow) dla przejścia $s_k^a \xrightarrow{\lambda_k} s_{k+1}^a$ w TS_a istnieje stan s_{k+1}^p i przejście $s_k^p \xrightarrow{\lambda_k} s_{k+1}^p$ w TS_p takie, że $s_{k+1}^p \cong_{pa} s_{k+1}^a$.

Niech $s_k^p = (q_1^k, \dots, q_n^k, v^k, \mu^k)$ oraz $s_k^a = (l^k, \tau^k)$ dla $k \in \mathbb{N}$.

(\Rightarrow) Rozważmy dwa przypadki.

1. $\lambda_k \in \Gamma$, czyli przejście z s_k^p do s_{k+1}^p odpowiada wykonaniu tranzycji akcyjnej przez jeden (lokalnie) bądź wiele (synchronicznie) procesów programu \mathcal{P} .

Pokażemy, że w automacie czasowym \mathcal{TA} istnieje tranzycja t' o etykiecie λ_k , która może być wykonana ze stanu s_k^a i stan s_{k+1}^a taki, że $s_k^a \xrightarrow{\lambda_k} s_{k+1}^a$ oraz $s_{k+1}^p \cong_{pa} s_{k+1}^a$.

Jeżeli procesy (lub jeden proces) programu \mathcal{P} wykonują tranzycje akcyjne (lub jedną, lokalną tranzycję akcyjną) o etykiecie λ_k ze stanu s_k^p do stanu s_{k+1}^p , to z def. 3.4 istnieje zbiór tranzycji $T' = \{t_i \mid i \in \Gamma(\lambda_k)\} \subseteq T$, taki że dla każdej tranzycji $t_i \in T'$ zachodzi $fireable_i(t_i, s_k^p)$, $label(t_i) = \lambda_k$ i $q_i^{k+1} = target(t_i)$ oraz $v^{k+1} = \mathcal{J}(v^k, \langle action(t_i) \rangle_{t_i \in T'})$, $\mu^{k+1} = \mu^k[\{q_i^{k+1} \mid i \in \Gamma(\lambda_k)\} := 0]$ i $q_j^{k+1} = q_j^k$ dla $j \in \{1, \dots, n\} \setminus \Gamma(\lambda_k)$.

Przypomnijmy, że $fireable_i(t_i, s_k^p) \stackrel{def}{=} enabled_i(t_i, s_k^p) \wedge \mu^k(source(t_i)) \in delay(t_i)$. Ponieważ $fireable_i(t_i, s_k^p) \Rightarrow enabled_i(t_i, s_k^p)$ i z założenia indukcyjnego wiemy, że $l^k = (q_1^k, \dots, q_n^k, v^k)$, więc zgodnie z def. 4.1 w automacie czasowym \mathcal{TA} istnieje lokacja l^{k+1} i tranzycja $e = l^k \xrightarrow{\lambda_k, \psi, Y} l^{k+1}$ taka, że $\psi = \bigwedge_{t_i \in T'} constraint(t_i)$ i $Y = \bigcup_{i \in \Gamma(\lambda_k)} \{x_1^i\} \cup \{c(t) \mid urgent(t) \wedge \neg enabled(t, l^k) \wedge enabled(t, l^{k+1})\}$.

Tranzycja e może być wykonana ze stanu s_k^a , jeżeli $\tau^k \models \psi$ i $\tau^{k+1} \models \mathcal{I}(l^{k+1})$, gdzie $\tau^{k+1} = \tau[Y := 0]$. Pokażemy, że warunki te są spełnione.

Dla każdej tranzycji $t_i \in T'$ zachodzi $source(t_i) = q_i^k$. Z tego, że $fireable_i(t_i, s_k^p)$ otrzymujemy $\mu^k(q_i^k) \in delay(t_i)$. Każdej tranzycji $t_i \in T'$, która nie jest pilna odpowiada zegar x_1^i . Z założenia indukcyjnego $\tau^k(x_1^i) = \mu^k(q_i^k)$ dla każdego $1 \leq i \leq n$, a więc dla każdej tranzycji $t_i \in T'$, która nie jest pilna $\tau^k \models constraint(t_i)$. Natomiast dla każdej tranzycji $t_i \in T'$, która jest pilna z założenia indukcyjnego $\tau^k(c(t_i)) = 0$. Dla tranzycji pilnej $constraint(t_i) \stackrel{def}{=} (c(t_i) \leq 0)$, a więc $\tau^k(c(t_i)) \models constraint(t_i)$. Stąd otrzymujemy $\tau^k \models \psi$ dla $\psi = \bigwedge_{t_i \in T'} constraint(t_i)$.

Pokażemy, że $\tau^{k+1} \models \mathcal{I}(l^{k+1})$, gdzie $\mathcal{I}(l^{k+1}) = \bigwedge_{\{t \in T \mid enabled(t, l^{k+1})\}} upper_constr(t)$. Dla każdej tranzycji możliwej do wykonania w konfiguracji l^{k+1} pokażemy, że $\tau^{k+1} \models upper_constr(t)$.

Niezmiennik $\mathcal{I}(l^{k+1})$ zawiera tylko warunki w postaci $x < d$ lub $x \leq d$ dla $x \in X$ i $d \in \mathbb{N}$. Z definicji 4.1 wynika, że zegary x_1^i dla $i \in \Gamma(\lambda_k)$ należą do zbioru Y zawierającego zegary do wyzerowania, a więc $\tau^{k+1}(x_1^i) = 0$, czyli $\tau^{k+1} \models upper_constr(t)$, dla każdej niepilnej tranzycji $t \in T_i$ dla $i \in \Gamma(\lambda_k)$. Z kolei z tego, że $\tau^k \models \mathcal{I}(l^k)$ wynika, że dla każdej niepilnej tranzycji $t \in T_i$ możliwej do wykonania w stanie s_{k+1}^p , gdzie $i \in \{1, \dots, n\} \setminus \Gamma(\lambda_k)$, zachodzi $\tau^k \models upper_constr(t)$, z czego wynika, że $\tau^{k+1} \models upper_constr(t)$.

Jeżeli t jest tranzycją pilną, która nie jest możliwa do wykonania w stanie s_k^p , a jest możliwa do wykonania w stanie s_{k+1}^p , to odpowiadający jej zegar $c(t)$ również zostanie wyzerowany, a więc $\tau^{k+1} \models upper_constr(t)$. Natomiast jeżeli t jest tranzycją pilną, która jest możliwa do wykonania zarówno w stanie s_k^p , jak i w s_{k+1}^p , to z założenia indukcyjnego dostajemy $\tau^k(c(t)) = 0$, a ponieważ $\tau^{k+1}(c(t)) = \tau^k(c(t))$, to $\tau^{k+1}(c(t)) = 0$. Widać więc, że dla każdej tranzycji t możliwej do wykonania w stanie s_{k+1}^p zachodzi $\tau^{k+1} \models upper_constr(t)$. A zatem $\tau^{k+1} \models \mathcal{I}(l^{k+1})$, co w połączeniu z $\tau^k \models \psi$, kończy dowód tego, że automat \mathcal{TA} będąc w stanie s_k^a może wykonać tranzycję e do stanu s_{k+1}^a .

Należy jeszcze pokazać, że $s_{k+1}^p \cong_{pa} s_{k+1}^a$. Z definicji tranzycji e (def. 4.1) widać natomiast, że $l^{k+1} = (q_1^{k+1}, \dots, q_n^{k+1}, v^{k+1})$, więc spełniony jest punkt 1 def. 4.2.

Z def. 3.4 $\mu^{k+1} = \mu^k[\{q_i^{k+1} \mid i \in \Gamma(\lambda_k)\} := 0]$, a zgodnie z definicją 4.1 do zbioru Y należą zegary x_1^i dla $i \in \Gamma(\lambda_k)$, więc $\tau^{k+1}(x_1^i) = \mu^{k+1}(q_i^{k+1}) = 0$ dla $i \in \Gamma(\lambda_k)$. Dla $i \in \{1, \dots, n\} \setminus \Gamma(\lambda_k)$ otrzymujemy $\tau^{k+1}(x_1^i) = \tau^k(x_1^i) = \mu^k(q_i^k) = \mu^{k+1}(q_i^{k+1})$. Warunek 2 def. 4.2 jest zatem spełniony.

Jak już pokazaliśmy wcześniej dla każdej pilnej tranzycji t możliwej do wykonania w stanie s_{k+1}^p zachodzi $\tau^{k+1}(c(t)) = 0$, więc spełniony jest również warunek 3 def. 4.2.

2. $\lambda_k \in \mathbb{R}_+$, co oznacza, że przejście λ_k reprezentuje upływ czasu. Podobnie jak poprzednio pokażemy, że automat czasowy \mathcal{TA} może wykonać przejście czasowe o długości λ_k i że $s_k^p + \lambda_k \cong_{pa} s_k^a + \lambda_k$. Przypadek $\lambda_k = 0$ jest oczywisty. Przyjmijmy, że $\lambda_k > 0$. Będąc w stanie s_k^a automat może wykonać przejście czasowe $\lambda_k > 0$, jeżeli $\tau^k + \lambda_k \models \mathcal{I}(l^k)$, gdzie $\mathcal{I}(l^k) = \bigwedge_{\{t \in T \mid enabled(t, l^k)\}} upper_constr(t)$. Skoro w programie \mathcal{P} może upłynąć λ_k jednostek czasu, to zgodnie z def. 3.4 dla każdej tranzycji

$t \in T$ zachodzi $\neg enabled(t, s_k^p) \vee (\neg urgent(t) \wedge \neg expired(t, s_k^p + \lambda_k))$. Wynika z tego po pierwsze, że żadna tranzycja pilna nie jest możliwa do wykonania w stanie s_k^p i po drugie, że dla każdej tranzycji $t \in T$ możliwej do wykonania w stanie s_k^p zachodzi $\mu^k(source(t)) + \lambda_k \in upper_delay(t)$ (z definicji *expired*, str. 29). Jeżeli tranzycja t jest możliwa do wykonania, to $source(t) = q_i^k$, gdzie i jest numerem procesu, do którego należy tranzycja t . Zatem dla każdej tranzycji $t \in T$ możliwej do wykonania $\mu^k(q_i^k) + \lambda_k \in upper_delay(t)$. Z założenia indukcyjnego $\tau^k(x_1^i) = \mu^k(q_i^k)$. Z definicji funkcji *upper_constr* otrzymujemy, że dla każdej tranzycji $t \in T$ możliwej do wykonania $\tau^k(x_1^i) + \lambda_k \models upper_constr(t)$. A więc $\tau^k + \lambda_k \models \mathcal{I}(l^k)$, co należało wykazać.

Pokażemy jeszcze, że stan $s_k^p + \lambda_k = (q_1^k, \dots, q_n^k, v^k, \mu^k + \lambda_k)$ jest w relacji \cong_{pa} ze stanem $s^a + \lambda_k = (l^k, \tau^k + \lambda_k)$. Prawdziwość warunku 1 def. 4.2 wynika wprost z prawdziwości tego warunku dla s_k^p i s_k^a . Podobnie dla warunku 2 dla $1 \leq i \leq n$, korzystając z tego, że $\tau^k(x_1^i) = \mu^k(q_i^k)$, otrzymujemy $\tau^k(x_1^i) + \lambda_k = \mu^k(q_i^k) + \lambda_k$. Warunek 3 jest spełniony w sposób oczywisty, ponieważ jak pokazaliśmy wcześniej żadna tranzycja pilna nie jest możliwa do wykonania w stanie s^p , a więc także w stanie $s_k^p + \lambda_k$. Zatem $s_k^p + \lambda_k \cong_{pa} s_k^a + \lambda_k$, co kończy dowód tego przypadku.

(\Leftarrow) Rozważmy dwa przypadki.

1. $\lambda_k \in \Sigma$, czyli przejście odpowiada wykonaniu tranzycji akcyjnej przez automat czasowy \mathcal{TA} . Pokażemy, że w programie \mathcal{P} istnieje stan s_{k+1}^p i zbiór (być może jednoelementowy) tranzycji $T' = \{t_i \mid i \in \Gamma(\lambda_k)\} \subseteq T$ gotowych do wykonania w stanie s_k^p takich, że po ich wykonaniu $s_k^a \xrightarrow{\lambda_k} s_{k+1}^a$ oraz $s_{k+1}^p \cong_{pa} s_{k+1}^a$.

Niech $e = l^k \xrightarrow{\lambda_k, \psi, Y} l^{k+1}$ będzie tranzycją wykonywaną przez automat \mathcal{TA} . Z def. 4.1 istnieje zbiór tranzycji $T' = \{t_i \mid i \in \Gamma(\lambda_k)\} \subseteq T$ taki, że dla każdej tranzycji $t_i \in T'$ zachodzi: $enabled_i(t_i, l^k)$, $label(t_i) = \lambda_k$ i $target(t_i) = q_i^{k+1}$ oraz $q_j^{k+1} = q_j^k$ dla $j \in \{1, \dots, n\} \setminus \Gamma(\lambda_k)$, $v^{k+1} = \mathcal{J}(v^k, \langle action(t_i) \rangle_{t_i \in T'})$, $\psi = \bigwedge_{t_i \in T'} constraint(t_i)$ i $Y = \bigcup_{i \in \Gamma(\lambda_k)} \{x_1^i\} \cup \{c(t) \mid urgent(t) \wedge \neg enabled(t, l^k) \wedge enabled(t, l^{k+1})\}$.

Pokażemy, że każda tranzycja $t_i \in T'$ jest gotowa do wykonania w stanie s_k^p . Ponieważ z założenia indukcyjnego $l^k = (q_1^k, \dots, q_n^k, v^k)$, więc $enabled_i(t_i, l^k)$ jest równoważne $enabled_i(t_i, s_k^p)$. Wystarczy zatem pokazać, że dla $i \in \Gamma(\lambda_k)$ $\mu^k(source(t_i)) \in delay(t_i)$. Skoro tranzycja e może być wykonana ze stanu s_k^a , więc $\tau^k \models \psi$, a więc dla $t_i \in T'$ zachodzi $\tau^k \models constraint(t_i)$. Zauważmy także, że skoro t_i jest możliwa do wykonania w stanie s_k^p , to $source(t_i) = q_i^k$. Z założenia indukcyjnego wiemy, że $\tau^k(x_1^i) = \mu^k(q_i^k)$ dla $1 \leq i \leq n$. Korzystając z definicji funkcji *constraint* i c otrzymujemy, że dla każdej tranzycji $t_i \in T'$, która nie jest pilna $\mu^k(source(t_i)) = \mu^k(q_i^k) \in delay(t_i)$. Natomiast dla każdej tranzycji $t_i \in T'$, która jest pilna $delay(t_i) = [0, \infty)$. Zatem dla każdej tranzycji $t_i \in T'$ zachodzi $\mu^k(source(t_i)) \in delay(t_i)$.

Dowód tego, że $s_{k+1}^p \cong_{pa} s_{k+1}^a$ jest analogiczny do dowodu tego faktu w części (\Rightarrow).

2. $\lambda_k \in \mathbb{R}_+$, co oznacza, że przejście reprezentuje upływ czasu. Podobnie jak poprzednio pokażemy, że w programie \mathcal{P} jest możliwe przejście czasowe o długości λ_k i że $s_k^p + \lambda_k \cong_{pa} s_k^a + \lambda_k$.

Przypadek $\lambda_k = 0$ jest oczywisty (system zawsze może wykonać taką tranzycję i po jej wykonaniu stan systemu nie zmienia się). Będąc w stanie s_k^p automat może wykonać przejście czasowe $\lambda_k > 0$, jeżeli dla każdej tranzycji $t \in T$ zachodzi $\neg enabled(t, s_k^p) \vee (\neg urgent(t) \wedge \neg expired(t, s_k^p + \lambda_k))$.

Skoro w automacie \mathcal{TA} może upłynąć λ_k jednostek czasu, to $\tau^k + \lambda_k \models \mathcal{I}(l^k) = \bigwedge_{\{t \in T \mid enabled(t, l^k)\}} upper_constr(t)$, czyli dla każdej tranzycji t możliwej do wykonania w stanie s_k^p zachodzi $\tau^k + \lambda_k \models upper_constr(t)$. Przypuśćmy, że istnieje tranzycja $t \in T$, która jest pilna i jest możliwa do wykonania. Jeżeli t jest tranzycją pilną, to $upper_constr(t) = (c(t) \leq 0)$. Dla $\lambda_k > 0$ otrzymujemy, że nieprawda, że $\tau^k + \lambda_k \models (c(t) \leq 0)$, co jest sprzeczne z założeniem. Zatem żadna pilna nie jest możliwa do wykonania w stanie s_k^p .

Z założenia indukcyjnego $\tau^k(x_1^i) = \mu^k(q_i^k)$ dla $1 \leq i \leq n$. Dla każdej tranzycji $t \in T$ możliwej do wykonania, skoro $\tau^k + \lambda_k \models upper_constr(t)$, to z definicji funkcji $upper_constr$ i c otrzymujemy $\mu^k(q_i^k) + \lambda_k \in upper_delay(t)$, czyli nieprawda, że $expired(t, s_k^p + \lambda_k)$.

A więc dla każdej tranzycji $t \in T$ możliwej do wykonania w s_k^p zachodzi: nieprawda, że $urgent(t)$ i nieprawda, że $expired(t, s_k^p + \lambda_k)$, co należało pokazać.

Dowodzimy, że $s_k^p + \lambda_k \cong_{pa} s_a^p + \lambda_k$ w taki sam sposób jak w części (\Rightarrow).

□

Fakt 4.4 $s_p^0 \cong_{pa} s_a^0$.

Dowód:

Należy pokazać, że dla $s_a^0 = (q_1^0, \dots, q_n^0, v^0, \tau^0)$, gdzie $\tau^0(x) = 0$ dla każdego $x \in X$ i dla $s_p^0 = (q_1^0, \dots, q_n^0, v^0, \mu^0)$, gdzie $\mu^0(q) = 0$ dla każdego stanu $q \in Q$, zachodzi $s_p^0 \cong_{pa} s_a^0$. Łatwo widać, że warunek 1 def. \cong_{pa} jest spełniony. Warunek 2 również jest spełniony, ponieważ $\tau^0(x_1^i) = 0 = \mu^0(q_i)$ dla $1 \leq i \leq n$. Prawdziwość warunku 3 wynika z tego, że $\tau^0(c(t)) = 0$ dla każdego $t \in T$. □

Niech funkcja $\mathcal{V}_p : S_p \rightarrow 2^{PV}$ będzie funkcją wartościującą dla systemu tranzycyjnego programu \mathcal{P} i niech $\mathcal{V}_a : S_a \rightarrow 2^{PV}$ będzie funkcją wartościującą dla systemu tranzycyjnego automatu \mathcal{TA} .

Fakt 4.5 Jeżeli $s^p \cong_{pa} s^a$, to $\mathcal{V}_p(s^p) = \mathcal{V}_a(s^a)$.

Dowód:

Wynika wprost z punktu 1 def. 4.2. □

Konsekwencją lematu 4.3 oraz faktów 4.4 i 4.5 jest następujący fakt.

Fakt 4.6 Modele $\mathcal{M}_a = (\mathcal{TS}_a, \mathcal{V}_a)$ i $\mathcal{M}_p = (\mathcal{TS}_p, \mathcal{V}_p)$ są w relacji silnej bisymulacji.

Przedstawimy teraz główne twierdzenie tej części pracy mówiące o tym, że dana własność wyrażona jako formuła logiki CTL* jest prawdziwa dla programu w języku bazowym wtedy i tylko wtedy, gdy jest prawdziwa dla automatu globalnego zbudowanego dla tego programu.

Twierdzenie 4.7 Dla każdej formuły φ logiki CTL*

$$\mathcal{M}_p, s_p^0 \models \varphi \text{ wtedy i tylko wtedy, gdy } \mathcal{M}_a, s_a^0 \models \varphi.$$

Zauważmy, że nie możemy skorzystać z faktu 4.6 i twierdzenia 2.7, które mówi, że jeżeli dwa modele są w relacji silnej bisymulacji, to spełniają te same formuły CTL*, ponieważ twierdzenie 2.7 dotyczy skończonych modeli, a w naszym przypadku są one nieskończone. Dlatego dowiedzimy prawdziwości twierdzenia 4.7 stosując standardowy dowód indukcyjny po długości formuły.

Lemat 4.8 Niech φ będzie dowolną formułą logiki CTL*, stanową lub ścieżkową. Niech σ^p będzie ścieżką ze stanu s^p w \mathcal{M}_p i niech σ^a będzie odpowiadającą jej ścieżką ze stanu s^a w \mathcal{M}_a . Zachodzą wtedy następujące warunki:

1. $s^p \models \varphi$ wtedy i tylko wtedy, gdy $s^a \models \varphi$, gdzie φ jest formułą stanową i
2. $\sigma^p \models \varphi$ wtedy i tylko wtedy, gdy $\sigma^a \models \varphi$, gdzie φ jest formułą ścieżkową.

Dowód:

Dowód przeprowadzimy przez indukcję po długości formuły φ .

Podstawa. $\varphi = p$, gdzie $p \in PV$. Z def. 2.5 $s^p \models p$ wtedy i tylko wtedy, gdy $p \in \mathcal{V}_p(s^p)$. Z faktu 4.5 $\mathcal{V}_p(s^p) = \mathcal{V}_a(s^a)$, a więc $p \in \mathcal{V}_p(s^p) \Leftrightarrow p \in \mathcal{V}_a(s^a)$. Korzystając ponownie z def. 2.5 otrzymujemy $s^p \models p \Leftrightarrow s^a \models p$.

Krok indukcyjny. Rozważmy następujące przypadki:

1. $\varphi = \neg\psi$, gdzie φ i ψ są formułami stanowymi. Z def. 2.5 dla operatora negacji, $s^p \models \varphi$ wtedy i tylko wtedy, gdy nieprawda, że $s^p \models \psi$. Z założenia indukcyjnego wiemy, że $s^p \models \psi \Leftrightarrow s^a \models \psi$. Zatem nieprawda, że $s^p \models \psi$ jest równoważne nieprawdą, że $s^a \models \psi$, a więc $s^p \models \neg\psi \Leftrightarrow s^a \models \neg\psi$.

Takie samo rozumowanie możemy przeprowadzić, gdy φ jest formułą ścieżkową.

2. $\varphi = \psi_1 \wedge \psi_2$, gdzie φ , ψ_1 i ψ_2 są formułami stanowymi. Z def. 2.5 dla operatora koniunkcji, $s^p \models \varphi$ wtedy i tylko wtedy, gdy $s^p \models \psi_1$ i $s^p \models \psi_2$. Z założenia indukcyjnego $s^p \models \psi_1 \Leftrightarrow s^a \models \psi_1$ i $s^p \models \psi_2 \Leftrightarrow s^a \models \psi_2$. Zatem $s^p \models \psi_1 \wedge \psi_2 \Leftrightarrow s^a \models \psi_1 \wedge \psi_2$, a więc $s^p \models \psi_1 \wedge \psi_2 \Leftrightarrow s^a \models \psi_1 \wedge \psi_2$.

Takie samo rozumowanie możemy przeprowadzić, gdy φ jest formułą ścieżkową.

3. Dla $\varphi = A\psi$, gdzie φ jest formułą stanową, a ψ jest formułą ścieżkową, dowód przeprowadzimy nie wprost.

Przypuśćmy, że $s^p \models \varphi$ i nieprawda, że $s^a \models \varphi$. Zatem z def. 2.5 dla operatora A otrzymujemy, że istnieje ścieżka σ^a ze stanu s^a taka, że nieprawda, że $\sigma^a \models \psi$. Z lematu 4.3 istnieje ścieżka σ^p ze stanu s^p odpowiadająca ścieżce σ^a . Z założenia indukcyjnego otrzymujemy nieprawdą, że $\sigma^p \models \psi$, czyli ze stanu s^p istnieje ścieżka σ^p taka, że nieprawda, że $\sigma^p \models \psi$, a więc nieprawda, że $s^p \models A\psi$. Sprzeczność.

Przypuśćmy z kolei, że $s^a \models \varphi$ i nieprawda, że $s^p \models \varphi$. Zatem istnieje ścieżka σ^p ze stanu s^p taka, że nieprawda, że $\sigma^p \models \psi$. Z lematu 4.3 istnieje ścieżka σ^a ze stanu s^a odpowiadająca ścieżce σ^p . Z założenia indukcyjnego otrzymujemy nieprawdą, że $\sigma^a \models \psi$, czyli ze stanu s^a istnieje ścieżka σ^a taka, że nieprawda, że $\sigma^a \models \psi$, a więc nieprawdą, że $s^a \models A\psi$. Sprzeczność.

Pokazaliśmy, że $s^p \models A\psi \Leftrightarrow s^a \models A\psi$.

4. Niech $\varphi = \psi$, gdzie φ jest formułą ścieżkową, a ψ jest formułą stanową. W takim przypadku, z def. 2.5 $\sigma^p \models \varphi \Leftrightarrow s^p \models \psi$. Z założenia indukcyjnego $s^p \models \psi \Leftrightarrow s^a \models \psi$, a więc $\sigma^p \models \varphi \Leftrightarrow \sigma^a \models \varphi$.
5. $\varphi = X\psi$, gdzie φ i ψ są formułami ścieżkowymi. Z def. 2.5 dla operatora X mamy $\sigma^p \models X\psi \Leftrightarrow \sigma_1^p \models \psi$. Ścieżki σ^p i σ^a odpowiadają sobie, a więc odpowiadają sobie również ścieżki σ_1^p i σ_1^a . Z założenia indukcyjnego $\sigma_1^p \models \psi \Leftrightarrow \sigma_1^a \models \psi$, a więc $\sigma^p \models X\psi \Leftrightarrow \sigma^a \models X\psi$.
6. $\varphi = U(\psi_1, \psi_2)$, gdzie φ, ψ_1 i ψ_2 są formułami ścieżkowymi. Z def. 2.5 dla operatora U mamy $\sigma^p \models U(\psi_1, \psi_2)$ wtedy i tylko wtedy, gdy istnieje $k \geq 0$ takie, że $\sigma_k^p \models \psi_2$ i dla $0 \leq j < k$ zachodzi $\sigma_j^p \models \psi_1$. Skoro ścieżki σ^p i σ^a odpowiadają sobie, to odpowiadają sobie również ścieżki σ_j^p i σ_j^a dla $j \in \mathbb{N}$. Zatem z założenia indukcyjnego $\sigma_k^p \models \psi_2 \Leftrightarrow \sigma_k^a \models \psi_2$ i $\sigma_j^p \models \psi_1 \Leftrightarrow \sigma_j^a \models \psi_1$ dla $0 \leq j < k$, a więc $\sigma^p \models U(\psi_1, \psi_2) \Leftrightarrow \sigma^a \models U(\psi_1, \psi_2)$. □

Dowód twierdzenia 4.7

Wynika z faktu 4.4 oraz lematów 4.3 i 4.8. □

4.3 Zbiór automatów czasowych

Niektóre metody i narzędzia weryfikacji modelowej działają dużo efektywniej, jeżeli zamiast jednego (najczęściej dużego) automatu dostaną na wejściu zbiór mniejszych automatów opisujących poszczególne składowe systemy, mimo że w czasie weryfikacji analizowane są przebiegi produktu automatów składowych. Dlatego przedstawimy teraz sposób generowania zbioru automatów czasowych dla programu w języku bazowym. Ten zbiór będzie się składał z automatów odpowiadających poszczególnym procesom oraz niektórym zmiennym globalnym i buforom.

Sposób konstrukcji zbioru automatów czasowych przedstawiony w tym rozdziale nie jest oczywiście jedyny. Przyjeliśmy zasadę, aby jak najwięcej niezależnych składowych systemu⁵ (procesów, zmiennych i buforów) było reprezentowanych przez osobne automaty, dzięki czemu uzyskujemy automaty o mniejszych rozmiarach. Jedynym wyjątkiem są zmienne lokalne procesu (czyli takie, które nie są definiowane ani używane przez inne procesy), które razem ze stanem kontrolnym wyznaczają lokalizację automatu dla tego procesu. Inną możliwość to konstrukcja osobnych automatów również dla niezależnych zmiennych lokalnych, w wyniku której otrzymamy zbiór o większej liczbie (mniejszych) automatów. Można budować automaty dla różnych podzbiorów składowych programu (w wyniku otrzymamy mniej

⁵Pojęcie niezależnych składowych wyjaśniamy w punkcie 4.3.2.

automatów, bardziej złożonych). Przyjęty w tej rozprawie sposób konstrukcji wydaje się najbardziej intuicyjny, ponieważ automaty dla procesów grupują wszystkie elementy związane z tymi procesami, a pozostałe, niezależne składowe są reprezentowane przez osobne automaty.

Przypomnijmy, że jeżeli tranzycja, która ma górne ograniczenie czasowe, jest możliwa do wykonania, to może zostać wykonana jedynie zanim upłynie czas określony przez to ograniczenie. W automatach czasowych jedynym sposobem na wymuszenie wykonania tranzycji w określonym czasie jest zdefiniowanie odpowiedniego niezmiennika w lokacji źródłowej tej tranzycji. Zauważmy jednak, że aby wiedzieć czy tranzycja jest możliwa do wykonania musimy znać wartości wszystkich zmiennych i buforów występujących w dozorze tranzycji. Stąd nasuwa się wniosek, że zmienne globalne i bufor, od których zależy umożliwienie tranzycji z górnym ograniczeniem czasowym, muszą określać lokację automatu czasowego dla procesu, do którego należy ta tranzycja. Lokacja automatu czasowego odpowiadającego procesowi P_i powinna być zatem określona przez stan kontrolny procesu, wartościowanie zmiennych lokalnych dla tego procesu oraz wartościowanie tych zmiennych globalnych i buforów, od których zależy postać niezmiennika tej lokacji, czyli takich, które występują w dozorach tranzycji z górnym ograniczeniem czasowym wychodzących z tej lokacji.

4.3.1 Ograniczenia

Podamy definicję zbioru automatów czasowych dla programu w języku bazowym, w którym:

1. tranzycje synchroniczne nie są pilne oraz
2. jeżeli dwie tranzycje o takich samych etykietach synchronizujących (czyli takich, które występują w więcej niż jednym procesie) mają wspólny stan źródłowy, to ich stany docelowe są różne.

Powód pierwszego ograniczenia jest następujący. Jak już wspomniano, aby wyrazić, że tranzycja ma być wykonana natychmiast, gdy tylko jest możliwa do wykonania, musimy określić odpowiedni niezmiennik dla lokacji źródłowej tej tranzycji w automacie składowym reprezentującym proces. Pilna tranzycja synchroniczna o etykiecie γ jest możliwa do wykonania, gdy dla każdego procesu o numerze ze zbioru $\Gamma(\gamma)$ istnieje tranzycja możliwa do wykonania o etykiecie γ . Dla pojedynczego procesu nie możemy więc jednoznacznie stwierdzić, czy dla danej lokacji automatu tego procesu powinien być niezmiennik zabraniający upływu czasu (powinien być w przypadku, gdy można wykonać tranzycję pilną, a nie powinno go być, gdy nie można jej wykonać). Rozwiązaniem tego problemu jest konstrukcja zbioru automatów reprezentujących zbioru procesów (zamiast pojedynczych procesów), gdzie procesy należące do różnych zbiorów nie mają wspólnych (synchronicznych) etykiet tranzycji pilnych, ale nie będziemy tutaj omawiać tej konstrukcji.

Drugie ograniczenie ma na celu ułatwienie konstruowania zbioru automatów. Dzięki niemu zbiór tranzycji możliwych do wykonania z danego stanu dla zbioru automatów będzie jednoznacznie określony dla każdej etykiety. Zauważmy, że jeżeli jakiś program nie spełnia tego warunku, czyli między parą stanów pewnego procesu istnieją dwie tranzycje synchroniczne o tej samej etykiecie γ , to możemy przekształcić program nie zmieniając jego działania, w ten sposób, że zastąpimy jedną z etykiet γ inną, nieużywaną etykietą γ' , w procesie dla którego warunek nie jest spełniony oraz w każdym z pozostałych procesów o

numerach ze zbioru $\Gamma(\gamma)$ dodamy nową tranzycję o etykietcie γ' , której pozostałe atrybuty są takie same jak tranzycji o etykietcie γ w tym procesie. Postępowanie to powtarzamy do momentu, gdy dla każdej pary tranzycji warunek będzie spełniony.

4.3.2 Niezależne zmienne globalne i bufory

Dla procesu P_i niech V'_i będzie zbiorem zmiennych lokalnych i takich zmiennych globalnych i buforów, od których zależy postać niezmienników, czyli takich, które występują w dozorach tranzycji z górnym ograniczeniem czasowym. Przyjmujemy, że tranzycja z górnym ograniczeniem czasowym to tranzycja pilna lub tranzycja, dla której górne ograniczenie na czas wykonania jest różne od ∞ . Dla tranzycji $t \in T$ z górnym ograniczeniem czasowym $upper_constr(t) \neq true$.

$$V'_i = V_i \cup \{y \in V_G \cup B \mid \exists t \in T_i, y \in vars(guard(t)) \wedge upper_constr(t) \neq true\}$$

Zbiór *niezależnych zmiennych globalnych* V'_G i zbiór *niezależnych buforów* B' definiujemy jak następuje:

$$V'_G = V \setminus \left(\bigcup_{i=1}^n V'_i \right), \quad B' = B \setminus \left(\bigcup_{i=1}^n V'_i \right).$$

4.3.3 Etykiety w automatach składowych

Dla programu $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$, konstruujemy zbiór m automatów czasowych \mathcal{TA}_i , gdzie $m = n + |V'_G| + |B'|$. Przyjmijmy, że niezależne zmienne globalne i bufory są w jednoznaczny sposób ponumerowane od $n+1$ do m . Przez y_i będziemy oznaczać zmienną lub bufor y wraz z jego numerem porządkowym. Natomiast przez $v|_Y$, gdzie $Y \subseteq V \cup B$, oznaczamy wartościowanie zmiennych i buforów $v : V \cup B \rightarrow \Omega$ obcięte do zbioru Y .

Niech $l_1 = (q_1, \dots, q_n, v_1)$, $l'_1 = (q'_1, \dots, q'_n, v'_1)$, $l_2 = (r_1, \dots, r_n, v_2)$ i $l'_2 = (r'_1, \dots, r'_n, v'_2)$, będą konfiguracjami programu, gdzie $q_i, r_i, q'_i, r'_i \in Q_i$ dla $1 \leq i \leq n$ oraz $v_1, v'_1, v_2, v'_2 \in \Omega^{V \cup B}$. Niech również γ_k oznacza napis złożony z etykiety $\gamma \in \Gamma$, podkreślenia i liczby $k \in \mathbb{N}$. Przez $\bar{\Gamma}$ oznaczmy zbiór $\{\gamma_k \mid \gamma \in \Gamma \wedge k \in \mathbb{N}\}$.

Niech $label : \Gamma \times L \times L \rightarrow \Gamma \cup \bar{\Gamma}$ będzie funkcją, która konstruuje etykietę tranzycji w automacie czasowym dla danej etykiety i konfiguracji programu, w taki sposób, że $label(\gamma_1, l_1, l'_1) = label(\gamma_2, l_2, l'_2)$ wtedy i tylko wtedy gdy:

- $\gamma_1 = \gamma_2$,
- $q'_j = r'_j$ i $v'_1|_{V'_j} = v'_2|_{V'_j}$ dla $1 \leq j \leq n$,
- $v_1(y_j) = v_2(y_j)$ i $v'_1(y_j) = v'_2(y_j)$ dla $n < j \leq m$.

Zauważmy, że funkcja $label$ może dla każdej pary konfiguracji l i l' przekazywać inną etykietę. Jest to jednak rozwiązanie nieefektywne, ponieważ w automatach składowych wygenerowanych w ten sposób powstałoby wiele tranzycji różniących się tylko etykietą, a łączna liczba tranzycji w automatach składowych byłaby co najmniej równa liczbie tranzycji

w automacie globalnym. W naszej implementacji pamiętamy jakie etykiety przydzieliliśmy poszczególnym trójkom (złożonym z etykiety programu i konfiguracji przed i po wykonaniu tranzycji). Tranzycji automatu składowego przydzielamy etykietę “najmniejszą z możliwych” (bez przyrostka lub o najmniejszym przyrostku k), która spełnia powyższe warunki.

4.3.4 Automaty składowe dla procesów

Lokacja automatu dla procesu jest wyznaczona przez stan kontrolny w jakim znajduje się proces oraz wartościowanie zmiennych ze zbioru V_i' , czyli zbioru zmiennych lokalnych tego procesu oraz zmiennych globalnych i buforów, od których zależy postać niezmienników. Intuicyjnie, automat czasowy dla procesu P_i wykonuje tranzycję wtedy, gdy odpowiadający mu proces P_i wykonuje tranzycję, a także wtedy, gdy inny proces $j \neq i$ zmienia wartość zmiennej należącej do zbioru V_i' (jest to tranzycja wykonywana synchronicznie z automatem dla procesu P_j).

Definicja 4.9 *Automat czasowy $\mathcal{TA}_i = (\Sigma_i, L_i, l_i^0, X_i, E_i, \mathcal{I}_i)$ odpowiadający procesowi $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$ dla każdego $1 \leq i \leq n$ definiujemy następująco:*

- $L_i = Q_i \times \Omega^{V_i'}$,
- $l_i^0 = (q_i^0, v_i^0) \in L_i$, gdzie $v_i^0 = v^0|_{V_i'}$,
- $X_i = \{x_1^i, \dots, x_k^i\}$, gdzie $k = \text{nut}(P_i) + 1$,
- $E \subseteq L_i \times \Sigma_i \times \Psi(X_i) \times 2^{X_i} \times L_i$ jest relacją przejścia zdefiniowaną dla $l_i = (q_i, v_i)$ i $l'_i = (q'_i, v'_i)$ w następujący sposób:

$l_i \xrightarrow{\sigma, \psi, Y} l'_i \in E_i$ wtedy i tylko wtedy, gdy istnieją: etykieta $\gamma \in \Gamma$, konfiguracje programu $l = (q_1, \dots, q_n, v)$ i $l' = (q'_1, \dots, q'_n, v')$ oraz zbiór tranzycji $T' = \{t_j \mid j \in \Gamma(\gamma)\} \subseteq T$ takie, że:

- $i \in \Gamma(\gamma)$ lub $v|_{V_i'} \neq v'|_{V_i'}$,
- $\text{enabled}_j(t_j, l)$, $\text{label}(t_j) = \gamma$ i $\text{target}(t_j) = q'_j$ dla każdej tranzycji $t_j \in T'$,
- $q'_k = q_k$ dla $k \in \{1, \dots, n\} \setminus \Gamma(\gamma)$,
- $v' = \mathcal{J}(v, \langle \text{action}(t_j) \rangle_{t_j \in T'})$ oraz $v_i = v|_{V_i'}$ i $v'_i = v'|_{V_i'}$,
- $\sigma = \text{label}(\gamma, l, l')$,
- $\psi = \text{constraint}(t_i)$, jeżeli $i \in \Gamma(\gamma)$, w przeciwnym przypadku $\psi = \text{true}$.
- $Y = Y' \cup \{x_1^i\}$, jeżeli $i \in \Gamma(\gamma)$, w przeciwnym przypadku $Y = Y'$, gdzie $Y' = \{c(t) \in X_i \mid \text{urgent}(t) \wedge \neg \text{enabled}(t, l) \wedge \text{enabled}(t, l')\}$,

- $\Sigma_i = \{\sigma \in \Gamma \cup \bar{\Gamma} \mid \exists l_i, l'_i \in L_i, \psi \in \Psi(X_i), Y \subseteq X_i \ l_i \xrightarrow{\sigma, \psi, Y} l'_i \in E_i\}$ jest zbiorem etykiet,
- $\mathcal{I}_i : L_i \longrightarrow \Psi(X_i)$ jest zdefiniowane dla $l_i = (q_i, v_i)$ jak następuje:

$$\mathcal{I}_i(l_i) = \bigwedge_{\{t \in \text{out}(q_i) \mid \text{upper_constr}(t) \neq \text{true}\}} \text{inv}(t, v_i)$$

gdzie $inv(t, v_i)$ jest zdefiniowane dla $t \in T_i$ i $v_i \in \Omega^{V'_i}$ w ten sposób, że jeżeli $\mathcal{B}(guard(t), v_i) = tt$, to $inv(t, v_i) = upper_constr(t)$, a w przeciwnym przypadku $inv(t, v_i) = true$.

Jeżeli tranzycja e_i w automacie \mathcal{TA}_i dla procesu P_i odpowiada pewnej tranzycji t_i o etykiecie γ wykonywanej przez proces P_i dla $i \in \Gamma(\gamma)$, to warunek umożliwienia tej tranzycji jest budowany na podstawie dozwolonego opóźnienia i atrybutu pilności tranzycji t_i z pomocą funkcji $constraint(t_i)$. W przeciwnym przypadku, czyli wtedy, gdy tranzycja e_i nie odpowiada żadnej tranzycji procesu P_i , ale inny proces zmienia wartości zmiennych lub buforów ze zbioru V'_i , warunek umożliwienia tranzycji e_i jest równy $true$.

Każda tranzycja e_i w automacie \mathcal{TA}_i zeruje zegary odpowiadające tranzycjom pilnym, które nie są możliwe do wykonania w konfiguracji określonej przez lokację źródłową tranzycji e_i i są możliwe do wykonania w konfiguracji określonej przez lokację docelową tranzycji e_i . Jeżeli tranzycja e_i odpowiada pewnej tranzycji t_i wykonywanej przez proces P_i , to dodatkowo jest zerowany również zegar x_1^i , którego wartość określa czas przebywania procesu w bieżącym stanie kontrolnym.

Powyższa konstrukcja niezmiennika dla lokacji $l_i \in L_i$ wymusza wykonanie tranzycji odpowiadających tranzycjom procesu, które są możliwe do wykonania, zgodnie z dopuszczalnym opóźnieniem i atrybutem pilności danej tranzycji. Zauważmy, że V'_i zawiera wszystkie zmienne globalne i bufora z dozoru $guard(t_i)$ dla tranzycji $t_i \in T_i$ z górnym ograniczeniem czasowym, czyli takich, dla których $upper_constr(t_i) \neq true$. Zatem $\mathcal{B}(guard(t), v_i)$ jest określone dla v_i , a co za tym idzie niezmiennik $\mathcal{I}(l_i)$ jest dobrze zdefiniowany.

4.3.5 Automaty składowe dla zmiennych globalnych i buforów

Lokacja automatu odpowiadającego niezależnej zmiennej globalnej lub niezależnemu buforowi jest wyznaczona przez wartość tej zmiennej lub zawartość bufora. Zbiór tranzycji w automacie składa się wyłącznie z tranzycji synchronicznych wykonywanych wspólnie z co najmniej jednym automatem odpowiadającym procesowi. Intuicyjnie, automat dla zmiennej lub bufora wykonuje przejście wtedy, gdy jakiś proces zmienia wartość odpowiadającej mu zmiennej lub bufora, a także wtedy, gdy dana zmienna lub bufor wchodzi w skład dozoru wykonywanej tranzycji (tranzycja jest możliwa do wykonania tylko dla odpowiedniego wartościowania zmiennych i buforów).

Definicja 4.10 *Automat czasowy $\mathcal{TA}_i = (\Sigma_i, L_i, l_i^0, X_i, E_i, \mathcal{I}_i)$ odpowiadający zmiennej lub buforowi $y_i \in V'_G \cup B'$ definiujemy następująco:*

- $L_i = \mathbb{Z}$ dla $y_i \in V'_G$ i $L_i = \mathbb{Z}^*$ dla $y_i \in B'$,
- $l_i^0 = v_i^0 \in L_i$, gdzie $v_i^0 = v^0(y_i)$,
- $X_i = \emptyset$,
- $E_i \subseteq L_i \times \Sigma_i \times \{true\} \times \{\emptyset\} \times L_i$ jest relacją przejścia zdefiniowaną następująco dla $l_i = v_i$ i $l'_i = v'_i$:
 $l_i \xrightarrow{\sigma, \psi, Y} l'_i \in E_i$, wtedy i tylko wtedy, gdy istnieją: etykieta $\gamma \in \Gamma$, konfiguracje $l = (q_1, \dots, q_n, v)$ i $l' = (q'_1, \dots, q'_n, v')$ oraz zbiór tranzycji $T' = \{t_j \mid j \in \Gamma(\gamma)\} \subseteq T$ takie, że:

- $v_i \neq v'_i$ lub $y_i \in \text{vars}(\text{guard}(t_j))$ dla pewnej tranzycji $t_j \in T'$,
 - $\text{enabled}_j(t_j, l)$, $\text{label}(t_j) = \gamma$ i $\text{target}(t_j) = q'_j$ dla każdej tranzycji $t_j \in T'$,
 - $q'_k = q_k$ dla $k \in \{1, \dots, n\} \setminus \Gamma(\gamma)$,
 - $v' = \mathcal{J}(v, \langle \text{action}(t_j) \rangle_{t_j \in T'})$ oraz $v_i = v(y_i)$ i $v'_i = v'(y_i)$,
 - $\sigma = \text{label}(\gamma, l, l')$,
 - $\psi = \text{true}$,
 - $Y = \emptyset$,
- $\Sigma_i = \{\sigma \in \Gamma \cup \bar{\Gamma} \mid \exists l_i, l'_i \in L_i, \psi \in \Psi(X_i), Y \subseteq X_i, l_i \xrightarrow{\sigma, \psi, Y} l'_i \in E_i\}$ jest zbiorem etykiet,
 - $\mathcal{I}_i : L_i \longrightarrow \{\text{true}\}$.

Zbiór automatów czasowych dla programu

Etykiety tranzycji skonstruowanych dla danego zbioru T' tranzycji możliwych do wykonania w danej konfiguracji są takie same dla wszystkich automatów \mathcal{TA}_i wykonujących te tranzycje. Co więcej, etykiety odpowiadające różnym zbiorom tranzycji możliwych do wykonania w danej konfiguracji są różne, chyba że konfiguracje otrzymane po ich wykonaniu są takie same, czyli wtedy, gdy ich wykonania są nierozróżnialne (pomijamy ten przypadek w dalszych rozważaniach). Istotnie, $\text{label}(\gamma, l, l'_1) = \text{label}(\gamma, l, l'_2)$ tylko wtedy, gdy konfiguracje l'_1 i l'_2 są takie same (odpowiednie stany kontrolne i wartościowania są równe).

Taka konstrukcja zapewnia, że po złożeniu automatów składowych, działanie automatu produktowego będzie odpowiadać działaniu programu, co pokażemy formalnie w punkcie 4.3.8.

Definicja 4.11 Zbiór m automatów czasowych \mathcal{TA}_i , gdzie $m = n + |V'_G| + |B'|$, odpowiadający programowi $\mathcal{P}(V, B, \{P_i \mid 1 \leq i \leq n\})$ składa się z n automatów odpowiadających procesom programu i z $|V'_G| + |B'|$ automatów odpowiadających poszczególnym niezależnym zmiennym globalnym i buforom.

4.3.6 Funkcje wartościujące dla automatów składowych

Zbiór zmiennych zdaniowych jakie możemy zdefiniować dla programu, dla którego chcemy zbudować zbiór automatów, jest ograniczony w stosunku do zbioru zmiennych zdaniowych jakie można zdefiniować w przypadku konstrukcji automatu globalnego. Nie możemy na przykład zdefiniować zmiennej zdaniowej $p_{a=b}$, gdzie a i b są zmiennymi lokalnymi różnych procesów lub są niezależnymi zmiennymi globalnymi, ponieważ prawdziwość zdania $a = b$ nie jest określona dla żadnej lokacji automatów czasowych ze zbioru, innymi słowy nie istnieje automat, do którego lokacji moglibyśmy przypisać takie zdanie. Zmienne zdaniowe, które możemy definiować są następujące:

- $p_{e_1 \sim e_2}$, gdzie $e_1, e_2 \in \text{Exp}(V'_i)$ dla $1 \leq i \leq n$ i \sim jest operatorem relacyjnym,
- $p_{i.q}$, gdzie $1 \leq i \leq n$ i $q \in Q_i$,
- $p_y \sim z$, gdzie $y \in V'_G$, $z \in \mathbb{Z}$ i \sim jest operatorem relacyjnym,

- $p_{empty(b)}$, gdzie $b \in B$.

Niech PV' oznacza zbiór zmiennych zdaniowych, które możemy definiować dla programu, dla którego budujemy zbiór automatów czasowych. Rodzinę funkcji wartościujących $\mathcal{V}'_{TA_i} : L_i \rightarrow 2^{PV'}$ dla zbioru automatów TA_1, \dots, TA_m definiujemy w następujący sposób. Niech $l_i = (q_i, v_i)$ dla $1 \leq i \leq n$ i $l_i = v_i$ dla $n < i \leq m$.

- $p_{e_1 \sim e_2} \in \mathcal{V}'_{TA_i}(l_i)$, gdzie $1 \leq i \leq n$ i $e_1, e_2 \in Exp(V'_i)$, wtedy i tylko wtedy, gdy $\mathcal{B}(e_1 \sim e_2, v_i) = tt$,
- $p_{empty(b)} \in \mathcal{V}'_{TA_i}(l_i)$, gdzie $1 \leq i \leq n$ i $b \in V'_i$, wtedy i tylko wtedy, gdy $\mathcal{B}(empty(b), v_i) = tt$,
- $p_{i.q} \in \mathcal{V}'_{TA_i}(l_i)$, gdzie $1 \leq i \leq n$, wtedy i tylko wtedy, gdy $q_i = q$,
- $p_{y_i \sim z} \in \mathcal{V}'_{TA_i}(l_i)$, gdzie TA_i jest automatem dla $y_i \in V'_G$, wtedy i tylko wtedy, gdy $\mathcal{B}(y_i \sim z, v_i) = tt$,
- $p_{empty(b_i)} \in \mathcal{V}'_{TA_i}(l_i)$, gdzie TA_i jest automatem dla $b_i \in B'$, wtedy i tylko wtedy, gdy $\mathcal{B}(empty(b_i), v_i) = tt$.

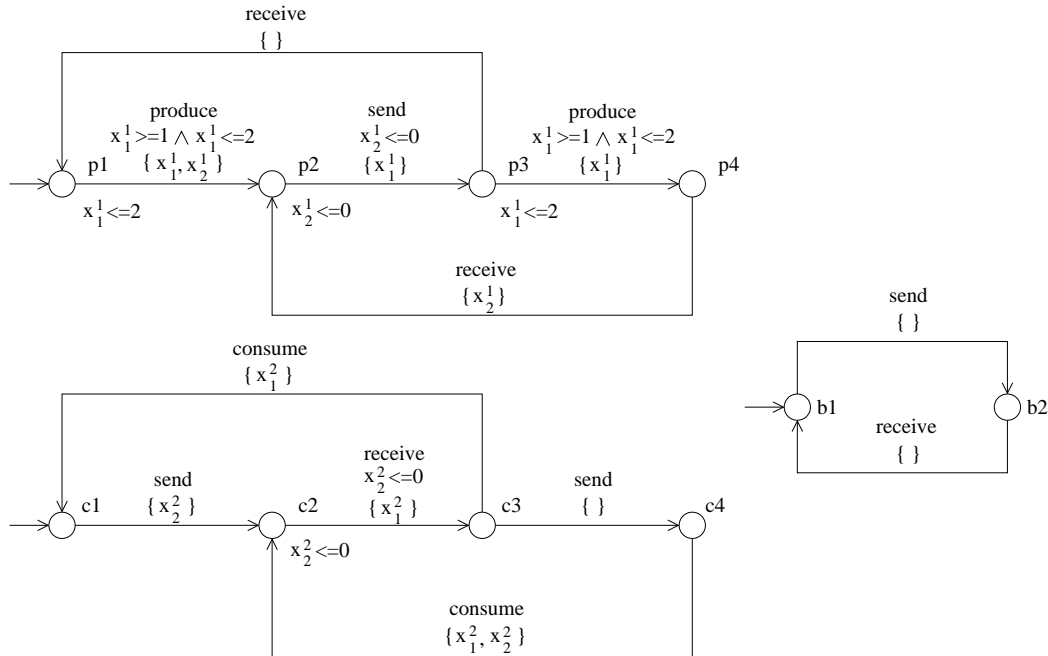
4.3.7 Przykład

Zbiór automatów czasowych dla przykładu producenta i konsumenta (dla $N = 1$) pokazuje rysunek 4.2. Podobnie jak poprzednio we wszystkich automatach przedstawiono tylko lokacje osiągalne z lokacji początkowych tych automatów. W skład zbioru automatów wchodzi trzy automaty: automat odpowiadający procesowi *Producent*, automat dla procesu *Konsument* i automat dla bufora b . Zmienna p jest zmienną lokalną dla procesu *Producent*, a zmienna c jest zmienną lokalną dla procesu *Konsument*. Natomiast zmienna *count* jest zmienną, od której zależy postać niezmienników lokacji automatów zarówno dla procesu *Producent*, jak i dla procesu *Konsument*, dlatego wchodzi w skład lokacji automatów dla obydwu procesów.

$$\begin{aligned}
p1 &= (P1, p = 0, count = 0), & c1 &= (C1, c = 0, count = 0), \\
p2 &= (P2, p = 0, count = 0), & c2 &= (C1, c = 0, count = 1), \\
p3 &= (P1, p = 0, count = 1), & c3 &= (C2, c = 0, count = 0), \\
p4 &= (P2, p = 0, count = 1), & c4 &= (C2, c = 0, count = 1), \\
b1 &= (b = \langle \rangle), \\
b2 &= (b = \langle 0 \rangle).
\end{aligned}$$

Poszczególnym lokacjom automatów przypisane są następujące zmienne zdaniowe:

$$\begin{aligned}
\mathcal{V}'_{TA_1}(p1) &= \{p1\}, & \mathcal{V}'_{TA_2}(c1) &= \{p1, p2\}, \\
\mathcal{V}'_{TA_1}(p2) &= \{p1\}, & \mathcal{V}'_{TA_2}(c2) &= \{p1, p2\}, \\
\mathcal{V}'_{TA_1}(p3) &= \{p1\}, & \mathcal{V}'_{TA_2}(c3) &= \{p1, p3\}, \\
\mathcal{V}'_{TA_1}(p4) &= \{p1\}, & \mathcal{V}'_{TA_2}(c4) &= \{p1, p3\}, \\
\mathcal{V}'_{TA_3}(b1) &= \emptyset, \\
\mathcal{V}'_{TA_3}(b2) &= \emptyset.
\end{aligned}$$



Rysunek 4.2: Zbiór automatów czasowych dla przykładu producenta i konsumenta

Automat dla procesu *Producent* ma tranzycje odpowiadające tranzycjom procesu (tranzycje o etykietach *produce* i *send*), ale także tranzycje, które zmieniają wartość zmiennej *count* (*receive*). Podobnie dla procesu *Konsument*. W automacie dla bufora występują jedynie tranzycje synchroniczne o etykietach *send* i *receive* (wykonywane wspólnie z automatami dla procesów *Producent* i *Konsument*). Przydział zegarów do poszczególnych procesów jest następujący: $X_1 = \{x_1^1, x_2^1\}$ (dla automatu odpowiadającego procesowi *Producent*), $X_2 = \{x_1^2, x_2^2\}$ (dla automatu dla procesu *Konsument*) i $X_3 = \emptyset$ (dla automatu dla bufora). Podobnie jak poprzednio warunki umożliwienia tranzycji i niezmienniki lokacji modelują dozwolone opóźnienia odpowiednich tranzycji poszczególnych procesów.

Zauważmy, że po złożeniu automatów z rys. 4.2 zgodnie z definicją 2.4 otrzymamy automat dokładnie taki jak przedstawiono na rys. 4.1, gdzie lokacjom automatu produktowego odpowiadają następujące trójki lokacji automatów składowych:

$$\begin{aligned}
 l1 &= (p1, c1, b1), & l5 &= (p1, c3, b1), \\
 l2 &= (p2, c1, b1), & l6 &= (p2, c3, b1), \\
 l3 &= (p3, c2, b2), & l7 &= (p3, c4, b2), \\
 l4 &= (p4, c2, b2), & l8 &= (p4, c4, b2).
 \end{aligned}$$

4.3.8 Poprawność

W tej części pokażemy, że zachowanie automatu globalnego i produktu automatów składowych, otrzymanych dla tego samego programu, jest takie samo i przedstawimy tego konsekwencje.

Niech $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$ będzie programem, w którym tranzycje synchroniczne nie są pilne. Niech również $\mathcal{TA}_1, \dots, \mathcal{TA}_m$ będzie zbiorem $m = n + |V'_G| + |B'|$ automatów czasowych zbudowanych dla programu \mathcal{P} i wartościowania początkowego $v^0 : V \cup B \rightarrow \Omega$ zgodnie z def. 4.11, a \mathcal{TA}' niech będzie produktem automatów $\mathcal{TA}_1, \dots, \mathcal{TA}_m$ i niech wreszcie \mathcal{TA} będzie globalnym automatem czasowym skonstruowanym dla programu \mathcal{P} i wartościowania v^0 zgodnie z def. 4.1. Przyjmijmy, że $\mathcal{TA} = (\Sigma, L, l^0, X, E, \mathcal{I})$, $\mathcal{TA}_i = (\Sigma_i, L_i, l_i^0, X_i, E_i, \mathcal{I}_i)$ dla $1 \leq i \leq m$ oraz $\mathcal{TA}' = (\Sigma', L', l^{0'}, X', E', \mathcal{I}')$. Zgodnie z definicją produktu automatów (def. 2.4) automat czasowy \mathcal{TA}' definiujemy w następujący sposób:

- $\Sigma' = \bigcup_{i=1}^m \Sigma_i$. Z def. 4.9 $\Sigma_i \subseteq \Gamma \cup \bar{\Gamma}$ dla $1 \leq i \leq n$, a zatem $\Sigma' \subseteq \Gamma \cup \bar{\Gamma}$.
- $L' = \prod_{i=1}^m L_i$. Z def. 4.9 i 4.10 $L' = \prod_{i=1}^n L_i \times \prod_{i=n+1}^m L_i = \prod_{i=1}^n (Q_i \times \Omega^{V'_i}) \times \prod_{i=n+1}^m \Omega$. Zauważmy, że $\bigcup_{i=1}^m V'_i \cup \bigcup_{i=n+1}^m \{y_i\} = V \cup B$.
- $l^{0'} = (l_1^0, \dots, l_m^0) = (q_1^0, v_1^0, \dots, q_n^0, v_n^0, v_{n+1}^0, \dots, v_m^0)$,
- $X' = \bigcup_{i=1}^n X_i \cup \bigcup_{i=n+1}^m \emptyset = \bigcup_{i=1}^n \bigcup_{j=1}^{nut(P_i)+1} \{x_j^i\} = X$,
- Niech $l'_1 = (q_1^1, v_1^1, \dots, q_n^1, v_n^1, v_{n+1}^1, \dots, v_m^1)$ i $l'_2 = (q_1^2, v_1^2, \dots, q_n^2, v_n^2, v_{n+1}^2, \dots, v_m^2)$. Tranzycja $(l'_1, \sigma', \bigwedge_{i \in \Sigma(\sigma)} \psi'_i, \bigcup_{i \in \Sigma(\sigma)} Y'_i, l'_2) \in E'$ wtedy i tylko wtedy, gdy $(l_i^1, \sigma_i, \psi_i, Y_i, l_i^2) \in E_i$ dla każdego $i \in \Sigma(\sigma)$ oraz $l_i^1 = l_i^2$ dla $i \in \{1, \dots, m\} \setminus \Sigma(\sigma)$.
- $\mathcal{I}'(l_1, \dots, l_m) = \bigwedge_{i=1}^m \mathcal{I}_i(l_i) = \bigwedge_{i=1}^n \bigwedge_{\{t \in out(q_i) \mid upper_constr(t) \neq true\}} inv(t, v_i) \wedge \bigwedge_{i=n+1}^m true$, gdzie $inv(t, v_i) = upper_constr(t)$, jeżeli $\mathcal{B}(guard(t), v_i) = tt$, a w przeciwnym przypadku $inv(t, v_i) = true$.

Niech $\mathcal{TS}_a = \{S_a, s_a^0, \Lambda_a, \longrightarrow_a\}$ będzie systemem tranzycyjnym dla automatu \mathcal{TA} , a $\mathcal{TS}'_a = \{S'_a, s'_a, \Lambda'_a, \longrightarrow'_a\}$ będzie systemem tranzycyjnym dla automatu produktowego $\mathcal{TA}' = \mathcal{TA}_1 \parallel \dots \parallel \mathcal{TA}_m$.

Niech $s = (q_1, \dots, q_n, v, \tau)$ i $s' = (q'_1, v'_1, \dots, q'_n, v'_n, v'_{n+1}, \dots, v'_m, \tau')$. Zauważmy, że $X' = X$, gdzie X' jest zbiorem zegarów w automacie produktowym \mathcal{TA}' , a X jest zbiorem zegarów w automacie globalnym \mathcal{TA} . Z tego wynika, że funkcje wartościowania zegarów τ i τ' mają takie same dziedziny. Rozważmy zatem następującą definicję równoważności na stanach systemów \mathcal{TS}_a i \mathcal{TS}'_a .

Definicja 4.12 Niech $\cong_{gp} \subseteq S_a \times S'_a$ będzie relacją taką, że dla stanów $s = (q_1, \dots, q_n, v, \tau)$ i $s' = (q'_1, v'_1, \dots, q'_n, v'_n, v'_{n+1}, \dots, v'_m, \tau')$, zachodzi $s \cong_{gp} s'$ wtedy i tylko wtedy, gdy spełnione są trzy następujące warunki:

1. $q_i = q'_i$ dla $1 \leq i \leq n$,
2. $v'_i = v|_{V'_i}$ dla $1 \leq i \leq n$ oraz $v'_i = v(y_i)$ dla $n < i \leq m$,

3. $\tau = \tau'$.

Powiemy, że dwie ścieżki $s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{j-1}} s_j$ i $s'_0 \xrightarrow{\lambda'_0} s'_1 \xrightarrow{\lambda'_1} \dots \xrightarrow{\lambda'_{j-1}} s'_j$ odpowiadają sobie, jeżeli dla każdego $j \geq 0$, $s_j \cong_{gp} s'_j$. Zauważmy, że jeżeli dwie ścieżki odpowiadają sobie, to ich odpowiednie pary sufixów również sobie odpowiadają.

Lemat 4.13 *Jeżeli $s \cong_{gp} s'$, to dla każdej ścieżki ze stanu s istnieje odpowiadająca jej ścieżka ze stanu s' i dla każdej ścieżki ze stanu s' istnieje odpowiadająca jej ścieżka ze stanu s .*

Dowód:

W tym dowodzie również wykorzystamy konstrukcję indukcyjną. Załóżmy, że dla prefiksu $s_0 \xrightarrow{\lambda_0} s_1 \xrightarrow{\lambda_1} \dots \xrightarrow{\lambda_{k-1}} s_k$ mamy zbudowany prefiks $s'_0 \xrightarrow{\lambda'_0} s'_1 \xrightarrow{\lambda'_1} \dots \xrightarrow{\lambda'_{k-1}} s'_k$ taki, że $s_0 = s$, $s'_0 = s'$ i $s_j \cong_{gp} s'_j$ dla każdego $0 \leq j \leq k$. Pokażemy, że:

- (\Rightarrow) dla przejścia $s_k \xrightarrow{\lambda_k} s_{k+1}$ w TS_a istnieje stan s'_{k+1} i przejście $s'_k \xrightarrow{\lambda'_k} s'_{k+1}$ w TS'_a takie, że $s_{k+1} \cong_{gp} s'_{k+1}$,
- (\Leftarrow) dla przejścia $s'_k \xrightarrow{\lambda'_k} s'_{k+1}$ w TS'_a istnieje stan s_{k+1} i przejście $s_k \xrightarrow{\lambda_k} s_{k+1}$ w TS_a takie, że $s_{k+1} \cong_{gp} s'_{k+1}$.

Dla $k \in \mathbb{N}$ niech $s_k = (l^k, \tau^k)$, gdzie $l^k = (q_1^k, \dots, q_n^k, v^k)$ będzie stanem w S_a oraz $s' = (l^{k'}, \tau^{k'})$, gdzie $l^{k'} = (l_1^{k'}, \dots, l_m^{k'})$ i $l_i^{k'} = (q_i^{k'}, v_i^{k'})$ dla $1 \leq i \leq n$ oraz $l_i^{k'} = v_i^{k'}$ dla $n < i \leq m$, będzie stanem w S'_a .

(\Rightarrow) Rozważmy dwa przypadki.

1. $\lambda_k \in \Sigma$, czyli przejście odpowiada wykonaniu tranzycji akcyjnej w automacie globalnym \mathcal{TA} . Pokażemy, że w automacie produktowym \mathcal{TA}' istnieje tranzycja e' o etykiecie λ'_k , która może być wykonana ze stanu s'_k i stan s'_{k+1} taki, że $s'_k \xrightarrow{\lambda'_k} s'_{k+1}$ oraz $s_{k+1} \cong_{gp} s'_{k+1}$.

W automacie produktowym istnieje tranzycja o etykiecie λ'_k wtedy i tylko wtedy, gdy istnieją tranzycje o tej etykiecie w automatach składowych. Należy zatem pokazać, że dla każdego $i \in \Sigma(\lambda'_k)$ istnieje tranzycja $(l_i^{k'}, \lambda'_k, \psi'_i, Y'_i, l_i^{k+1'}) \in E_i$.

Niech $e = l^k \xrightarrow{\lambda_k, \psi, Y} l^{k+1}$ będzie tranzycją wykonywaną przez automat \mathcal{TA} . Z def. 4.1 istnieje zbiór tranzycji $T' = \{t_i \mid i \in \Gamma(\lambda_k)\} \subseteq T$ taki, że dla każdej tranzycji $t_i \in T'$ zachodzi $enabled_i(t_i, l^k)$, $label(t_i) = \lambda_k$ i $target(t_i) = q_i^{k+1}$ oraz $q_j^{k+1} = q_j^k$ dla $j \in \{1, \dots, n\} \setminus \Gamma(\lambda_k)$, $v^{k+1} = \mathcal{J}(v^k, \langle action(t_i) \rangle_{t_i \in T'})$, $\psi = \bigwedge_{t_i \in T'} constraint(t_i)$ i $Y = \bigcup_{i \in \Gamma(\lambda_k)} \{x_1^i\} \cup \{c(t) \mid urgent(t) \wedge \neg enabled(t, l^k) \wedge enabled(t, l^{k+1})\}$. Z założenia indukcyjnego $s_k \cong_{gp} s'_k$, a więc $q_i^k = q_i^{k'}$ dla $1 \leq i \leq n$ oraz $v_i^{k'} = v^k|_{V'_i}$ dla $1 \leq i \leq n$ i $v_i^{k'} = v^k(y_i)$ dla $n < i \leq m$.

Niech $\lambda'_k = label(\lambda_k, l^k, l^{k+1})$. Dla każdego $1 \leq i \leq m$ z definicji Σ_i mamy, że $\lambda'_k \in \Sigma_i$, jeżeli w zbiorze E_i istnieje tranzycja o etykiecie λ'_k , czyli wtedy i tylko wtedy, gdy (a) $i \in \Gamma(\lambda_k)$ lub (b) $i \in \{1, \dots, n\} \setminus \Gamma(\lambda_k)$ i $v^k|_{V'_i} \neq v^{k+1}|_{V'_i}$ lub (c) $n < i \leq m$ i $(v^k(y_i) \neq v^{k+1}(y_i))$ lub $y_i \in vars(guard(t))$ dla pewnej tranzycji $t \in T'$. Rozważmy wymienione przypadki.

- (a) jeżeli $i \in \Gamma(\lambda_k)$, to z def. 4.9 istnieje lokacja $l_i^{k+1'}$ i tranzycja $l_i^{k'} \xrightarrow{\lambda'_k, \psi_i, Y_i} l_i^{k+1'}$ $l_i^{k+1'} \in E_i$, gdzie $l_i^{k+1'} = (q_i^{k+1'}, v_i^{k+1'})$, $q_i^{k+1'} = \text{target}(t_i)$, $v_i^{k+1'} = v^{k+1}|_{V'_i}$, $\psi_i = \text{constraint}(t_i)$ i $Y_i = \{x_1^i\} \cup \{c(t) \in X_i \mid \text{urgent}(t) \wedge \neg \text{enabled}(t, l^k) \wedge \text{enabled}(t, l^{k+1})\}$,
- (b) jeżeli $i \in \{1, \dots, n\} \setminus \Gamma(\lambda_k)$ i $v^k|_{V'_i} \neq v^{k+1}|_{V'_i}$, to z def. 4.9 istnieje lokacja $l_i^{k+1'}$ i tranzycja $l_i^{k'} \xrightarrow{\lambda'_k, \psi_i, Y_i} l_i^{k+1'} \in E_i$, gdzie $l_i^{k+1'} = (q_i^{k+1'}, v_i^{k+1'})$, $q_i^{k+1'} = q_i^{k'}$, $v_i^{k+1'} = v^{k+1}|_{V'_i}$, $\psi_i = \text{true}$ i $Y_i = \{c(t) \in X_i \mid \text{urgent}(t) \wedge \neg \text{enabled}(t, l^k) \wedge \text{enabled}(t, l^{k+1})\}$,
- (c) jeżeli $n < i \leq m$ i $(v^k(y_i) \neq v^{k+1}(y_i))$ lub $y_i \in \text{vars}(\text{guard}(t))$ dla pewnej tranzycji $t \in T'$, to z def. 4.10 istnieje lokacja $l_i^{k+1'}$ i tranzycja $l_i^{k'} \xrightarrow{\lambda'_k, \psi_i, Y_i} l_i^{k+1'} \in E_i$, gdzie $l_i^{k+1'} = v_i^{k+1'}$, $v_i^{k+1'} = v^{k+1}(y_i)$, $\psi_i = \text{true}$ i $Y_i = \emptyset$.

Zatem dla każdego $i \in \Sigma(\lambda'_k)$ w automacie \mathcal{TA}_i istnieje tranzycja o etykiecie λ'_k z lokacji $l_i^{k'}$, a więc w automacie produktowym istnieje tranzycja e' o etykiecie λ'_k z lokacji $l^{k'}$.

Pokażemy teraz, że automat produktowy będąc w stanie s'_k może wykonać tranzycję $e' = (l^{k'}, \lambda'_k, \psi', Y', l^{k+1'})$, gdzie $\psi' = \bigwedge_{i \in \Sigma(\lambda'_k)} \psi'_i = \bigwedge_{i \in \Gamma(\lambda_k)} \text{constraint}(t_i) \wedge \bigwedge_{i \in \Sigma(\lambda'_k) \setminus \Gamma(\lambda_k)} \text{true} = \bigwedge_{t_i \in T'} \text{constraint}(t_i) = \psi$ i $Y' = \bigcup_{i \in \Sigma(\lambda'_k)} Y'_i = \bigcup_{i \in \Gamma(\lambda_k)} \{x_1^i\} \cup \bigcup_{i \in \Sigma(\lambda'_k)} \{c(t) \in X_i \mid \text{urgent}(t) \wedge \neg \text{enabled}(t, l^k) \wedge \text{enabled}(t, l^{k+1})\} = \bigcup_{i \in \Gamma(\lambda_k)} \{x_1^i\} \cup \{c(t) \in X \mid \text{urgent}(t) \wedge \neg \text{enabled}(t, l^k) \wedge \text{enabled}(t, l^{k+1})\} = Y$. Należy zatem pokazać, że $\tau^{k'} \models \psi'$ i $\tau^{k+1'} \models \mathcal{I}(l^{k+1'})$, gdzie $\tau^{k+1'} = \tau^{k'}[Y' := 0]$.

Automat globalny \mathcal{TA} może wykonać tranzycję e , więc $\tau^k \models \psi$, $\tau^{k+1} \models \mathcal{I}(l^{k+1})$, gdzie $\tau^{k+1} = \tau[Y := 0]$ i $\mathcal{I}(l^{k+1}) = \bigwedge_{\{t \in T \mid \text{enabled}(t, l^{k+1})\}} \text{upper_constr}(t)$.

$\mathcal{I}'(l^{k+1'}) = \bigwedge_{i=1}^m \mathcal{I}'_i(l^{k+1'}) = \bigwedge_{\{t \in \text{out}(q_i^{k+1'}) \mid \text{upper_constr}(t) \neq \text{true}\}} \text{inv}(t, v_i^{k+1'})$, gdzie $\text{inv}(t, v_i^{k+1'}) = \text{upper_constr}(t)$, jeżeli $\mathcal{B}(\text{guard}(t), v_i^{k+1'}) = tt$, natomiast w przeciwnym przypadku $\text{inv}(t, v_i^{k+1'}) = \text{true}$. Zauważmy, że dla tranzycji $t \in \text{out}(q_i^{k+1'})$ takiej, że $\text{upper_constr}(t) \neq \text{true}$ warunek $\text{enabled}(t, l^{k+1})$ jest równoważny warunkowi $\mathcal{B}(\text{guard}(t), v_i^{k+1'}) = tt$, ponieważ $q_i^{k+1'} = q_i^{k+1}$ a do zbioru V'_i należą wszystkie zmienne z dozoru tranzycji t . Zatem $\text{inv}(t, v_i^{k+1'}) = \text{upper_constr}(t)$, jeżeli $\text{enabled}(t, l^{k+1})$, a w przeciwnym przypadku $\text{inv}(t, v_i^{k+1'}) = \text{true}$. Zatem

$$\mathcal{I}'(l^{k+1'}) = \bigwedge_{\{t \in T' \mid \text{enabled}(t, l^{k+1}) \wedge \text{upper_constr}(t) \neq \text{true}\}} \text{upper_constr}(t) = \mathcal{I}(l^{k+1}).$$

Z założenia indukcyjnego $\tau^k = \tau^{k'}$. Korzystając z tego, że $\psi' = \psi$, $Y' = Y$ oraz $\mathcal{I}'(l^{k+1'}) = \mathcal{I}(l^{k+1})$ otrzymujemy $\tau^{k'} \models \psi'$ oraz $\tau^{k+1'} \models \mathcal{I}'(l^{k+1'})$ co oznacza, że automat produktowy może wykonać tranzycję e' .

Należy jeszcze pokazać, że $s_{k+1} \cong_{gp} s'_{k+1}$. Z punktów (a)-(c) otrzymujemy, że dla każdego $i \in \Sigma(\lambda'_k)$: jeżeli $1 \leq i \leq n$, to $q_i^{k+1'} = q_i^{k+1}$ i $v_i^{k+1'} = v^{k+1}|_{V'_i}$ oraz jeżeli $n < i \leq m$, to $v_i^{k+1'} = v_i^{k+1}(y_i)$. Natomiast dla $i \in \{1, \dots, m\} \setminus \Sigma(\lambda'_k)$ zachodzi: jeżeli

$i \leq n$, to $q_i^{k+1'} = q_i^{k'}$ oraz $v_i^{k+1'} = v_i^{k'}$, a zatem korzystając z założenia indukcyjnego i z tego, że $q_j^{k+1} = q_j^k$ i $v^k|_{V_i'} = v^{k+1}|_{V_i'}$ dla $j \in \{1, \dots, n\} \setminus \Gamma(\lambda_k)$ oraz $v^k(y_i) = v^{k+1}(y_i)$ dla $j \in \{n+1, \dots, m\} \setminus \Gamma(\lambda_k)$ otrzymujemy: jeżeli $i \leq n$, to $q_i^{k+1'} = q_i^{k'}$ oraz $v_i^{k+1'} = v_i^{k'}$ i $v_i^{k+1'} = v_i^{k'} = v^k|_{V_i'} = v^{k+1}|_{V_i'}$ oraz jeżeli $n < i \leq m$, to $v_i^{k+1'} = v_i^{k'} = v^k(y_i) = v^{k+1}(y_i)$. Z czego wynika, że warunki 1 i 2 def. 4.12 są spełnione. Jak zauważyliśmy wcześniej $\tau^{k'} = \tau^k$ i $Y' = Y$, zatem $\tau^{k+1'} = \tau^{k'}[Y' := 0] = \tau^k[Y := 0] = \tau^{k+1}$, czyli spełniony jest również warunek 3 def. 4.12.

2. $\lambda_k \in \mathbb{R}_+$, czyli przejście reprezentuje upływ czasu. Pokażemy, że w automacie produktowym \mathcal{TA}' jest możliwe przejście czasowe o długości λ_k i że $s_k + \lambda_k \cong_{gp} s_k' + \lambda_k$. Będąc w stanie s_k' automat może wykonać przejście czasowe λ_k , jeżeli $\tau^k + \lambda_k \models \mathcal{I}(l^k)$, gdzie $\mathcal{I}(l^k) = \bigwedge_{\{t \in T \mid \text{enabled}(t, l^k)\}} \text{upper_constr}(t)$. Z założenia indukcyjnego wiemy, że $\tau^k = \tau^{k'}$. Jak już pokazaliśmy dla tranzycji $t \in \text{out}(q_i^k)$ takiej, że $\text{upper_constr}(t) \neq \text{true}$ warunek $\text{enabled}(t, l^k)$ jest równoważny $\mathcal{B}(\text{guard}(t), v_i^{k'}) = tt$ i stąd $\mathcal{I}'(l^{k'}) = \mathcal{I}(l^k)$. Zatem $\tau^{k'} + \lambda_k \models \mathcal{I}'(l^{k'})$, czyli automat produktowy \mathcal{TA}' może wykonać przejście czasowe o długości λ_k .

Pokażemy jeszcze, że stan $s_k + \lambda_k$ jest w relacji \cong_{gp} ze stanem $s_k' + \lambda_k$. Prawdziwość warunków 1 i 2 def. 4.12 wynika wprost z prawdziwości tego warunku dla s_k i s_k' . Podobnie dla warunku 3, korzystając z tego, że $\tau^k = \tau^{k'}$ otrzymujemy $\tau^k + \lambda_k = \tau^{k'} + \lambda_k$.

(\Leftarrow) Rozważmy dwa przypadki.

1. $\lambda_k' \in \Sigma'$, czyli przejście odpowiada wykonaniu tranzycji akcyjnej w automacie produktowym \mathcal{TA}' . Pokażemy, że w automacie globalnym \mathcal{TA} istnieje tranzycja e o etykietcie λ_k , która może być wykonana ze stanu s_k i stan s_{k+1} taki, że $s_k \xrightarrow{\lambda_k} s_{k+1}$ oraz $s_{k+1} \cong_{gp} s_{k+1}'$.

Niech $e' = l^{k'} \xrightarrow{\lambda_k', \psi', Y'} l^{k+1'}$ będzie tranzycją wykonywaną przez automat \mathcal{TA}' . Z definicji złożenia dla każdego $i \in \Sigma(\lambda_k')$ istnieje tranzycja $(l_i^{k'}, \lambda_k', \psi_i', Y_i', l_i^{k+1'}) \in E_i$ oraz $\psi' = \bigwedge_{i \in \Sigma(\lambda_k')} \psi_i'$ i $Y' = \bigcup_{i \in \Sigma(\lambda_k')} Y_i'$. Niech $l_i^{k'} = (q_i^{k'}, v_i^{k'})$ i $l_i^{k+1'} = (q_i^{k+1'}, v_i^{k+1'})$ dla $1 \leq i \leq n$ oraz $l_i^{k'} = v_i^{k'}$ i $l_i^{k+1'} = v_i^{k+1'}$ dla $n < i \leq m$.

A zatem istnieją: etykieta γ i konfiguracje r^k, r^{k+1} takie, że $\lambda_k' = \text{label}(\gamma, r^k, r^{k+1})$, gdzie $r^k = (q_1^{k'}, \dots, q_n^{k'}, v^{k'})$, $r^{k+1} = (q_1^{k+1'}, \dots, q_n^{k+1'}, v^{k+1'})$, $v_i^{k'} = v^{k'}|_{V_i'}$ dla $1 \leq i \leq n$ oraz $v_i^{k'} = v^{k'}(y_i)$ dla $n < i \leq m$ a także $v_i^{k+1'} = v^{k+1'}|_{V_i'}$ dla $1 \leq i \leq n$ oraz $v_i^{k+1'} = v^{k+1'}(y_i)$ dla $n < i \leq m$,

Zauważmy, że zgodnie z konstrukcją zbioru automatów istnieje co najmniej jedno i takie, że $i \in \Gamma(\gamma) \subseteq \Sigma(\lambda_k')$ (dla $j \in \Sigma(\lambda_k') \setminus \Gamma(\gamma)$ automat \mathcal{TA}_j wykonuje wyłącznie tranzycje synchronicznie z automatami odpowiadającymi procesom). Zatem dla każdego $i \in \Sigma(\lambda_k')$ z def. 4.9 i 4.10 istnieje zbiór tranzycji $T' = \{t_j \mid j \in \Gamma(\gamma)\} \subseteq T$, taki że dla każdej tranzycji $t_j \in T'$ zachodzi

- (a) $\text{enabled}_j(t_j, r^k)$ i $\text{label}(t_j) = \gamma$,

- (b) $target(t_j) = q_j^{k+1'}$ oraz $q_h^{k+1'} = q_h^{k'}$ dla $h \in \{1, \dots, n\} \setminus \Gamma(\gamma)$,
(c) $v^{k+1'} = \mathcal{J}(v^{k'}, \langle action(t_j) \rangle_{t_j \in T'})$.

Jak zauważyliśmy wcześniej (str. 54) dla danej konfiguracji r^k i etykiety λ'_k jest tylko jeden zbiór tranzycji T' spełniający powyższe warunki. Zatem dla każdego $i \in \Sigma(\lambda'_k)$ zbiór T' jest taki sam.

Z def. 4.9 dla każdego $i \in \Gamma(\gamma)$ $\psi'_i = constraint(t_i)$ i $Y_i = Y'_i \cup \{x_1^i\}$, gdzie $Y'_i = \{c(t) \in X_i \mid urgent(t) \wedge \neg enabled(t, r^k) \wedge enabled(t, r^{k+1})\}$. Co więcej, z def. 4.9 dla $i \in \Sigma(\lambda'_k) \setminus \Gamma(\gamma)$ takiego, że $i \leq n$ (automaty dla procesów nie wykonujących tranzycji): $v_i^{k+1'} \neq v_i^{k'}$, $\psi'_i = true$ i $Y_i = \{c(t) \in X_i \mid urgent(t) \wedge \neg enabled(t, r^k) \wedge enabled(t, r^{k+1})\}$ oraz z def. 4.10 dla $i \in \Sigma(\lambda'_k)$ takiego, że $i > n$ (automaty dla zmiennych lub buforów): $v_i^{k+1'} \neq v_i^{k'}$ lub $y_i \in vars(guard(t_j))$, gdzie $t_j \in T'$, $\psi'_i = true$ i $Y_i = \emptyset$.

Z założenia indukcyjnego otrzymujemy $q_i^k = q_i^{k'}$ i $v_i^{k'} = v^k|_{V_i}$ dla $1 \leq i \leq n$ oraz $v_i^{k'} = v^k(y_i)$ dla $n < i \leq m$, z czego wynika, że $v^{k'} = v^k$.

Zauważmy, że dla każdego $n < i \leq m$ takiego, że $y_i \in vars(guard(t_j))$ dla pewnej tranzycji $t_j \in T'$, z def. 4.10 istnieje tranzycja w automacie \mathcal{TA}_i o etykiecie λ'_k . Wobec powyższych, zgodnie z def. 4.1 w automacie globalnym istnieje tranzycja $e = l^k \xrightarrow{\lambda_k, \psi, Y} l^{k+1}$, gdzie $l^k = (q_1^k, \dots, q_n^k, v^k)$, $l^{k+1} = (q_1^{k+1}, \dots, q_n^{k+1}, v^{k+1})$, $\psi = \bigwedge_{t_i \in T'} constraint(t_i)$ i $Y = \bigcup_{i \in \Gamma(\lambda_k)} \{x_1^i\} \cup \{c(t) \mid urgent(t) \wedge \neg enabled(t, l^k) \wedge enabled(t, l^{k+1})\}$.

Podobnie jak w poprzedniej części dowodu (\Rightarrow) możemy pokazać, że $\psi = \psi'$, $Y = Y'$ i $\mathcal{I}(l^{k+1}) = \mathcal{I}'(l^{k+1'})$ a także, że automat globalny \mathcal{TA} może wykonać tranzycję e (w analogiczny sposób jak pokazaliśmy, że automat produktowy \mathcal{TA}' może wykonać tranzycję e').

Należy jeszcze pokazać, że $s_{k+1} \cong_{gp} s'_{k+1}$.

Z def. 4.1 $q_j^{k+1} = target(t_j)$ dla $j \in \Gamma(\gamma)$ oraz $q_j^{k+1} = q_j^k$ dla $j \in \{1, \dots, n\} \setminus \Gamma(\gamma)$. Z punktu (b) i z założenia indukcyjnego otrzymujemy, że $q_j^{k+1'} = target(t_j) = q_j^{k+1}$ dla $j \in \Gamma(\gamma)$ oraz $q_j^{k+1'} = q_j^{k'} = q_j^k = q_j^{k+1}$ dla $j \in \{1, \dots, n\} \setminus \Gamma(\gamma)$, czyli warunek 1 def. 4.12 jest spełniony.

Korzystając ponownie z def. 4.1 otrzymujemy: $v^{k+1} = \mathcal{J}(v^k, \langle action(t_i) \rangle_{t_i \in T'})$. Z punktu (c) i z założenia indukcyjnego dostajemy $v^{k+1'} = \mathcal{J}(v^{k'}, \langle action(t_i) \rangle_{t_i \in T'}) = \mathcal{J}(v^k, \langle action(t_i) \rangle_{t_i \in T'}) = v^{k+1}$, czyli $v_i^{k+1'} = v^{k+1}|_{V_i}$ dla $1 \leq i \leq n$ oraz $v_i^{k+1'} = v^{k+1}(y_i)$ dla $n < i \leq m$, a więc spełniony jest warunek 2 def. 4.12.

Z założenia indukcyjnego $\tau^{k'} = \tau^k$, a jak zauważyliśmy wcześniej $Y' = Y$, zatem $\tau^{k+1'} = \tau^{k'}[Y' := 0] = \tau^k[Y := 0] = \tau^{k+1}$, czyli spełniony jest również warunek 3 def. 4.12.

2. $\lambda'_k \in \mathbb{R}_+$, czyli przejście reprezentuje wpływ czasu. Dowód jest analogiczny do przeprowadzonego w części (\Rightarrow).

□

Fakt 4.14 $s^0 \cong_{gp} s^{0'}$.

Dowód:

Należy pokazać, że dla $s^0 = (q_1^0, \dots, q_n^0, v^0, \tau^0)$, gdzie $\tau^0(x) = 0$ dla każdego $x \in X$ i dla $s^{0'} = (l_1^{0'}, \dots, l_m^{0'}, \tau^{0'})$, gdzie $l_i^{0'} = (q_i^{0'}, v_i^{0'})$ dla $1 \leq i \leq n$, $l_i^{0'} = v_i^{0'}$ dla $n < i \leq m$ oraz $\tau^{0'}(x) = 0$ dla każdego $x \in \bigcup_{i=1}^n X_i = X$ zachodzi $s^0 \cong_{gp} s^{0'}$.

Prawdziwość warunków 1 i 2 definicji relacji \cong_{pa} wynika wprost z definicji lokacji początkowych automatów składowych (def. 4.9 i 4.10). Warunek 3 również jest spełniony, ponieważ $\tau^0(x) = 0 = \tau^{0'}(x)$ dla każdego $x \in X$. \square

Niech funkcja $\mathcal{V}_p : S_p \rightarrow 2^{PV'}$ będzie funkcją wartościującą dla systemu tranzycyjnego programu \mathcal{P} i niech $\mathcal{V}_a : S_a \rightarrow 2^{PV'}$ oraz $\mathcal{V}'_a : S'_a \rightarrow 2^{PV'}$ będą funkcjami wartościującymi dla systemów tranzycyjnych automatów \mathcal{TA} i \mathcal{TA}' . Niech również $\mathcal{M}_p = (\mathcal{TS}_p, \mathcal{V}_p)$, $\mathcal{M}_a = (\mathcal{TS}_a, \mathcal{V}_a)$ i $\mathcal{M}'_a = (\mathcal{TS}'_a, \mathcal{V}'_a)$

Fakt 4.15 Jeżeli $s \cong_{gp} s'$, to $\mathcal{V}_a(s) = \mathcal{V}'_a(s')$.

Dowód:

Wynika wprost z punktu 1 i 2 def. 4.12. \square

Z lematu 4.13 oraz z faktów 4.14 i 4.15 wynika następujący fakt.

Fakt 4.16 Modele $\mathcal{M}_a = (\mathcal{TS}_a, \mathcal{V}_a)$ i $\mathcal{M}'_a = (\mathcal{TS}'_a, \mathcal{V}'_a)$ są w relacji silnej bisymulacji.

Następne twierdzenie mówi, że każda formuła φ logiki CTL*, która jest prawdziwa dla automatu globalnego jest również prawdziwa dla automatu produktowego, i odwrotnie.

Twierdzenie 4.17 Dla każdej formuły φ logiki CTL*

$$\mathcal{M}_a, s_a^0 \models \varphi \text{ wtedy i tylko wtedy, gdy } \mathcal{M}'_a, s_a^{0'} \models \varphi.$$

Dowód:

Dowód można przeprowadzić w ten sam sposób, co dowód twierdzenia 4.7 poprzez indukcję po długości formuły φ . \square

Z ostatniego twierdzenia tego rozdziału wynika, że konstrukcja zbioru automatów jest poprawna w tym sensie, że wszelkie własności wyrażone w postaci formuł logiki CTL*, które są prawdziwe dla zbioru automatów wygenerowanych dla pewnego programu, są również prawdziwe dla tego programu, i odwrotnie.

Twierdzenie 4.18 Dla każdej formuły φ logiki CTL*

$$\mathcal{M}_p, s_p^0 \models \varphi \text{ wtedy i tylko wtedy, gdy } \mathcal{M}'_a, s_a^{0'} \models \varphi.$$

Dowód:

Na podstawie twierdzeń 4.7 i 4.17:

$$\mathcal{M}_p, s_p^0 \models \varphi \Leftrightarrow \mathcal{M}_a, s_a^0 \models \varphi \Leftrightarrow \mathcal{M}'_a, s_a^{0'} \models \varphi.$$

\square

4.4 Redukcja liczby zegarów w automatach czasowych

Złożoność problemu weryfikacji modelowej automatów czasowych jest wykładnicza względem liczby użytych zegarów, co pokazano w pracy [AD94]. Podczas generowania automatów czasowych wykorzystamy spostrzeżenia z pracy [DY96] pozwalające na zmniejszenie liczby zegarów każdego z automatów bez zmiany zachowania jego działania.

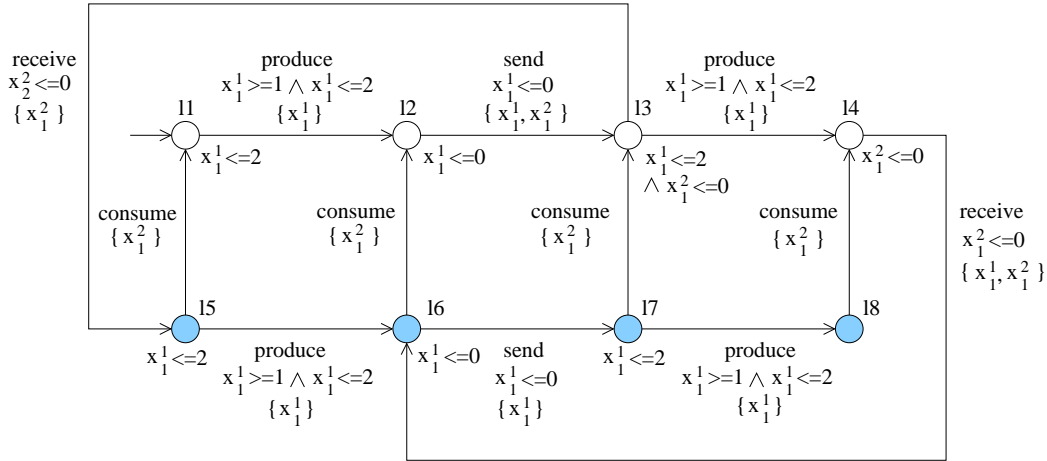
Autorzy wymienionej pracy zauważają po pierwsze, że jeżeli dwa zegary są zerowane w tym samym momencie, to ich wartość jest przez jakiś czas równa, więc jeden z tych zegarów nie jest w tym czasie potrzebny. Przenosząc to spostrzeżenie na nasz grunt, możemy dwóm tranzycjom pilnym, przypisać te same zegary, jeżeli te tranzycje są zawsze jednocześnie możliwe do wykonania. Łatwo widać, że warunek ten spełniają tranzycje tego samego procesu, które mają wspólny stan źródłowy i równe dozory. Zegary odpowiadające tym tranzycjom są zerowane i wykorzystywane w tym samym czasie, gdzie przez czas wykorzystania zegarów rozumiemy okres kiedy tranzycje, które im odpowiadają są możliwe do wykonania. Redukcja dotyczy tylko tranzycji pilnych, bo zgodnie z konstrukcją automatów opisaną w poprzednich rozdziałach wszystkim tranzycjom, które nie są pilne, i tak przydzielamy jeden zegar.

Po drugie, dwa zegary, które nie są wykorzystywane jednocześnie, mogą zostać zastąpione jednym zegarem. W naszym przypadku, jeżeli dwie tranzycje nie są nigdy jednocześnie możliwe do wykonania, to możemy im przypisać ten sam zegar. Warunek ten spełniają tranzycje tego samego procesu wychodzące z różnych stanów. Również, jeżeli dozór jednej tranzycji jest zaprzeczeniem dozoru innej tranzycji należącej do tego samego procesu, to tym tranzycjom również możemy przypisać ten sam zegar. W ogólnym przypadku dotyczy to wielu tranzycji, których dozory są rozłączne.

Zauważmy wreszcie, że jeżeli dozwolone opóźnienie tranzycji jest w postaci $[0, \infty)$ i tranzycja ta nie jest pilna, czyli może być wykonana w dowolnym czasie (mówimy o niej, że nie ma ograniczeń czasowych), to nie musimy takiej tranzycji w ogóle przypisywać zegara.

Aby zastosować powyższe redukcje możemy zmienić funkcję c przypisującą zegary tranzycjom na odpowiednią funkcję częściową. Rozważmy przydział zegarów tranzycjom procesu P_i . Dla każdego stanu kontrolnego q procesu P_i postępujemy w następujący sposób. Na początku wszystkie zegary ze zbioru X_i zdefiniowanego jak w punkcie 4.1 zaznaczamy jako wolne. Potem, wszystkim tranzycjom wychodzącym ze stanu q , które nie są pilne i mają ograniczenia czasowe przypisujemy zegar x_i^1 . Jeżeli jest przynajmniej jedna tranzycja spełniająca powyższe warunki, to zegar zaznaczamy jako zajęty. Następnie analizujemy w dowolnej kolejności wszystkie tranzycje pilne wychodzące ze stanu q . Dla kolejnej tranzycji t sprawdzamy, czy wśród tranzycji wychodzących ze stanu q , które mają już przydzielone zegary istnieje tranzycja t' , której dozór jest równy lub jest zaprzeczeniem dozoru tranzycji t . Jeżeli tak, to tranzycji t przypisujemy ten sam zegar co tranzycji t' , a jeżeli nie to przydzielamy jej pierwszy wolny zegar ze zbioru X_i i zaznaczamy go jako zajęty. Oczywiście po przydzieleniu zegarów wszystkim tranzycjom procesu, jeżeli jakiś zegar ze zbioru X_i nie został przypisany żadnej z nich, to usuwamy go ze zbioru. Jeżeli tranzycji $t \in T$ nie przydzielono zegara, to $constraint(t) = true$.

Przykład 4.19 W przykładzie producenta i konsumenta korzystając z zasady rozłączności możemy tranzycji o etykiecie *send* przypisać ten sam zegar, co tranzycji o etykiecie *produce*. Podobnie dla tranzycji *consume* i *receive*. W ten sposób każdemu procesowi odpowiada



Rysunek 4.3: Zredukowany automat czasowy dla przykładu producenta i konsumenta

tylko jeden zegar (x_1^1 dla *Producenta* i x_1^2 dla *Konsumenta*). Zredukowany automat globalny i zbiór zredukowanych automatów czasowych dla tego przykładu pokazano na rysunkach 4.3 i 4.4.

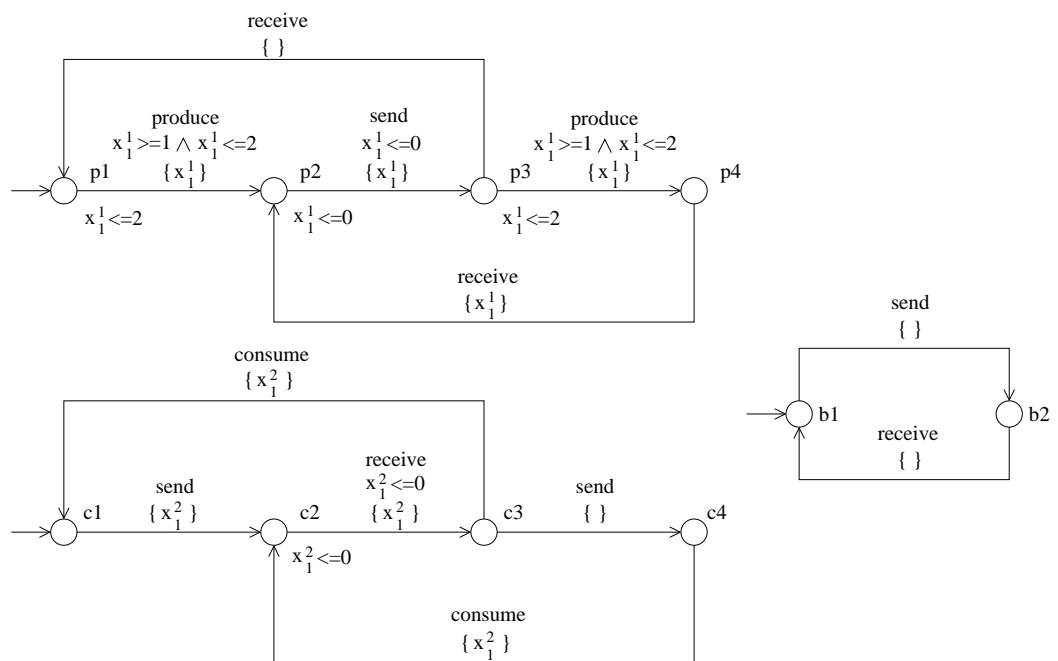
4.5 Podsumowanie rozdziału

Sprowadzanie jednego formalizmu do innego jest często stosowanym rozwiązaniem. Tłumaczenie do automatów czasowych zostało wykonane dla algebry procesów ATP [NSY92], języka ET-LOTOS [DOY94] i języka Esterel wzbogaconego o zależności czasowe [BCP⁺01]. Powyższe tłumaczenia dotyczą języków synchronicznych stosowanych głównie przy projektowaniu sprzętu komputerowego. Wspólną cechą tych modeli jest zastosowanie mechanizmów komunikacji synchronicznej, w której składowe systemy (procesy) wykonują wspólnie pewne operacje. W tym przypadku tłumaczenie polega na zbudowaniu osobnego automatu czasowego dla każdego procesu i naturalnym przeniesieniu metody komunikacji. Takie naturalne tłumaczenie nie jest możliwe, kiedy język opisu systemu czasowego wykorzystuje komunikację asynchroniczną (na przykład przez wymianę komunikatów).

Na koniec wspomnimy o dwóch sposobach bardziej efektywnej konstrukcji zbioru automatów czasowych dla pewnych przypadków oraz o implementacji zaprezentowanych metod.

Procesy niezależne od buforów

Chcielibyśmy, aby w konstruowanym zbiorze automatów czasowych buforom odpowiadały osobne automaty. Jednak umożliwienie tranzycji wykonywanej przez proces zależy od pustości czy niepustości buforów, jeżeli w dozorze takiej tranzycji znajduje się warunek *empty(b)*, gdzie *b* jest jednym z buforów programu. Jeżeli tranzycja, której wykonanie zależy od niepustości bufora, jest pilna lub ma górne ograniczenie czasowe, to stan automatu dla procesu,



Rysunek 4.4: Zbiór zredukowanych automatów czasowych dla producenta i konsumenta

do którego należy tranzycja będzie wyznaczony między innymi przez zawartość tego bufora. Aby tego uniknąć można zmodyfikować program w sposób, który nie zmieni jego działania, a pozwoli na efektywniejsze generowanie zbioru automatów czasowych. Można to zrobić wprowadzając dodatkowe zmienne (liczniki), których wartość będzie odpowiadać liczbie elementów znajdujących się w buforze.

Niech $B' \subseteq B$ będzie zbiorem buforów takim, że wyrażenie $empty(b)$ wchodzi w skład dozoru dowolnego procesu należącego do programu. Niech $V_B = \{count_b \mid b \in B'\}$ będzie zbiorem dodatkowych zmiennych takim, że $V_B \cap V = \emptyset$. Wartość zmiennej $count_b$ będzie odpowiadać liczbie elementów w buforze b . Przyjmujemy, że $v^0(count_b) = 0$ dla każdego $b \in B$.

Dla dowolnego programu $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$ możemy zbudować program $\mathcal{P}' = (V', B, \{P'_i \mid 1 \leq i \leq n\})$, w którym możliwość wykonania tranzycji nie zależy od zawartości buforów. Niech $V' = V \cup V_B$. Program \mathcal{P}' jest programem \mathcal{P} , w którym dokonano następujących zmian — dla każdego bufora $b \in B'$:

- każdą operację $put(b, e)$, gdzie $e \in Exp(V)$, zastąpiono operacjami $put(b, e); count_b = count_b + 1$,
- każdą operację $get(b, y)$, gdzie $y \in V$, zastąpiono operacjami $get(b, y); count_b = count_b - 1$,
- każde wyrażenie logiczne $empty(b)$ występujące w dozorach zastąpiono wyrażeniem $count_b = 0$.

Zauważmy, że powyższe zmiany nie wpływają na działanie programu, ponieważ zachodzi $\mathcal{B}(empty(b), v) = tt$ wtedy i tylko wtedy, gdy $\mathcal{B}(count_b = 0, v') = tt$, gdzie $v \in \Omega^{V \cup B}$ i $v' \in \Omega^{V' \cup B}$.

Automaty czasowe dla programów bez buforów i zmiennych globalnych

W trakcie prac rozważano również inne podejścia takie jak generowanie zbioru automatów na podstawie opisu poszczególnych procesów, czyli bez badania przestrzeni konfiguracji programu. Jak już wspomnieliśmy takie podejście jest z powodzeniem stosowane, gdy w języku opisu systemów czasowych, dla którego przeprowadzamy tłumaczenie, jest zastosowana wyłącznie komunikacja synchroniczna (przez wspólne wykonywanie tranzycji).

Jeżeli ograniczymy się do klasy programów języka bazowego, w którym komunikacja jest wyłącznie synchroniczna, oraz tranzycje synchroniczne nie są pilne, to możemy zbudować zbiór automatów czasowych w bardzo prosty sposób. Przykładem takiego programu jest synchroniczny protokół Fischera (rozd. 3.4.2).

Niech $\mathcal{P} = (V, \emptyset, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$, będzie programem, w którym nie występują zmienne globalne, ani bufory, oraz żadna tranzycja synchroniczna nie jest pilna. Automat czasowy $TA_i = (\Sigma_i, L_i, l_i^0, X_i, E_i, \mathcal{I}_i)$ dla procesu P_i definiujemy następująco:

- $\Sigma_i = \Gamma_i$ jest zbiorem etykiet,
- $L_i = Q_i \times \Omega^{V_i}$ jest zbiorem lokacji,
- $l_i^0 = (q_i^0, v_i^0) \in L_i$, gdzie $v_i^0 = v^0|_{V_i}$, jest lokacją początkową,

- $X_i = \{x_i\}$, jest zbiorem zegarów,
- $E_i \subseteq L_i \times \Sigma_i \times \Psi(X_i) \times 2^{X_i} \times L_i$ jest relacją przejścia zdefiniowaną następująco dla $l = (q_i, v_i)$ i $l' = (q'_i, v'_i)$:

$l_i \xrightarrow{\gamma, \psi, Y} l'_i \in E$, wtedy i tylko wtedy, gdy

- istnieje tranzycja $t_i \in T_i$ taka, że $source(t) = q_i$, $target(t_i) = q'_i$, $\mathcal{B}(guard(t_i), v_i) = tt$ oraz $\gamma = label(t_i)$
- $v'_i = \mathcal{J}(v_i, action(t_i))$,
- $\psi = constraint(t_i)$, gdzie $c(t_i) = x_i$,
- $Y = \{x_i\}$,

- $\mathcal{I}_i : L_i \longrightarrow \Psi(X_i)$, gdzie dla $l_i = (q_i, v_i)$

$$\mathcal{I}_i(l_i) = \bigwedge_{\{t \in out(q_i) \mid \mathcal{B}(guard(t_i), v_i) = tt\}} upper_constr(t).$$

Dla synchronicznego protokołu Fischera stosując zarówno właśnie przedstawiony sposób konstrukcji, jak i ogólną metodę z punktu 4.3, otrzymamy zbiór automatów przedstawiony w punkcie 2.1.6.

Natomiast, jeżeli weźmiemy pod uwagę dualny model (asynchroniczny), czyli klasę programów, w których wszystkie tranzycje są lokalne, to nie możemy podać istotnie prostszej metody generowania automatów czasowych. Jedyna różnica polega na tym, że zamiast zbiory tranzycji możliwych do wykonania będą się składać zawsze z jednej tranzycji.

Implementacja

Implementacja opisanej metody generowania automatu globalnego dla programu w języku bazowym została wykonana przez Annę Doroś i była przedmiotem jej pracy magisterskiej pt. *Translacja z pewnego języka specyfikacji do automatów z czasem*. Metodę generowania zbioru automatów zaimplementowała autorka rozprawy wspólnie z Pawłem Janowskim.

W obydwu przypadkach algorytm generowania automatów czasowych rozpoczyna działanie od obliczenia lokacji początkowych na podstawie początkowej konfiguracji programu. Dla lokacji początkowej obliczany jest zbiór tranzycji możliwych do wykonania z konfiguracji odpowiadającej tej lokacji i zbiór następników, czyli lokacji, które odpowiadają konfiguracjom programu po wykonaniu tych tranzycji. Działanie to jest powtarzane rekurencyjnie dla każdego z następników analizowanej lokacji, do momentu gdy wszystkie wygenerowane lokacje zostaną przeanalizowane. Jak widać przedstawiony algorytm generuje tylko lokacje osiągalne z lokacji początkowych automatów. Algorytm opiera się na algorytmie przeszukiwania wszcz lub wgląd (metoda przeszukiwania jest parametrem algorytmu), więc jego koszt jest liniowy ze względu na liczbę konfiguracji programu i przejść między nimi.

Implementacje obydwu metod stanowią część systemu weryfikacyjnego Verics[DJJ⁺03a].

Rozdział 5

Redukcja przestrzeni stanów programu metodą cięcia

W tym rozdziale przedstawiamy zastosowanie metody cięcia do języka bazowego. Metoda ta polega na wykorzystaniu statycznej analizy kodu programu do eliminacji jego nieistotnych fragmentów, czyli w naszym przypadku takich, które nie mają wpływu na weryfikowane własności. Pozwala to na uproszczenie kodu programu, co z kolei powoduje zmniejszenie przestrzeni stanów potrzebnej do jego weryfikacji. Zaletą metody jest to, że pełna przestrzeń stanów nie musi być generowana, ponieważ można ją skonstruować już dla zredukowanego kodu programu.

Metoda cięcia zależy od danej formuły logicznej wyrażającej własność, której prawdziwość chcemy sprawdzić, a dokładniej — od zbioru zmiennych zdaniowych, które wchodzi w skład formuły. Punktem wyjściowym jest ustalenie tak zwanego kryterium cięcia, czyli zbiorów akcji i stanów, od których bezpośrednio zależą wartości zmiennych zdaniowych z formuły. Następnie, algorytm redukcji bada rekurencyjnie zależności między akcjami i stanami poszczególnych procesów systemu zaczynając od kryterium cięcia.

We współbieżnym systemie czasowym występują cztery podstawowe rodzaje zależności. Zależność danych i zależność przepływu sterowania są klasycznymi pojęciami metody cięcia. Zależność od synchronizacji jest charakterystyczna dla systemów współbieżnych, które mogą wykonywać pewne akcje synchronicznie. Natomiast pojęcie zależności czasowej jest nowym rodzajem zależności występującym tylko w systemach czasowych. Pojęcie to nie zostało wcześniej zdefiniowane.

Struktura rozdziału jest następująca. Rozpoczynamy od zdefiniowania relacji zależności dla programów w języku bazowym. Następnie podajemy algorytmy obliczania istotnych operacji i istotnych stanów kontrolnych oraz pokazujemy jak z istotnych elementów skonstruować zredukowany program. Ważną częścią rozdziału jest wykazanie poprawności przedstawionej metody, co czynimy w kolejnym punkcie. Przedostatni punkt rozdziału zawiera opis przeprowadzonych eksperymentów. Podsumowujemy rozdział podając między innymi przegląd literatury dotyczącej zastosowania metody cięcia do redukcji przestrzeni stanów.

5.1 Relacje zależności

Jak już wspomniano zależność danych (ang. *data dependence*) i zależność przepływu sterowania (ang. *control dependence*) to klasyczne pojęcia metody cięcia [Tip95]. Różne postacie zależności od synchronizacji także pojawiają się w literaturze dotyczącej metody cięcia dla systemów współbieżnych [HCD⁺99]. Natomiast pojęcie zależności czasowej nie było do tej pory wprowadzone, ponieważ nie stosowano tej techniki dla systemów czasowych.

Podamy teraz definicje wszystkich zależności dla programów w języku bazowym. Relacje zależności dotyczą operacji i stanów kontrolnych programu. Przypomnijmy, że $opers(t)$ oznacza zbiór operacji tranzycji t . Natomiast $vars(a)$, $def(a)$ i $use(a)$ oznaczają odpowiednio zbiory zmiennych i buforów występujących, definiowanych i używanych w operacji a (definicje pojęć używanych w tym rozdziale znajdują się w punkcie 3.1.1). Metodę cięcia dla programu w języku bazowym przedstawimy przy założeniu, że program jest strukturalny (punkt 3.2).

5.1.1 Zależność danych

Nieformalnie, operacja $a_1 \in Ops$ zależy od operacji $a_2 \in Ops$ i jest to zależność danych, jeżeli: (1) operacja a_2 ma wpływ na wartość zmiennej lub bufora używanego w operacji a_1 i (2) jest możliwe takie wykonanie programu, że pomiędzy operacją a_2 a a_1 nie wystąpi inna operacja, która zmieni wartość tej zmiennej lub bufora. Innymi słowy, operacja a_1 zależy od operacji a_2 , jeżeli definicja zmiennej lub bufora w operacji a_2 jest definicją osiągniętą (ang. *reaching definition*) [ASU02] tej zmiennej lub bufora dla operacji a_1 .

Jeżeli obydwie operacje należą do jednego procesu możemy sprawdzić warunek (2) analizując strukturę przepływu sterowania tego procesu, a dokładniej — tranzycje, do których należą operacje a_1 i a_2 oraz tranzycje, które mogą być wykonane pomiędzy tymi tranzycjami. Należy zatem sprawdzić ścieżki prowadzące ze stanu docelowego tranzycji, która zawiera operację a_2 do stanu źródłowego tranzycji, która zawiera operację a_1 . Jeżeli natomiast, operacje a_1 i a_2 należą do różnych procesów sprawdzenie tego warunku przez analizę kodu nie jest możliwe i musimy przyjąć, że operacja używająca zmiennej lub bufora jest zależna od każdej operacji definiującej tę zmienną lub bufor wchodzącej w skład akcji tranzycji innego procesu.

Definicja 5.1 (Zależność danych) Dla $t_1 \in T_i$, $t_2 \in T_j$, gdzie $1 \leq i, j \leq n$, operacja $a_1 \in opers(t_1)$ jest zależna od operacji $a_2 \in opers(t_2)$ i jest to zależność danych (co oznaczamy $a_1 \xrightarrow{dd} a_2$), jeżeli istnieje zmienna lub bufor $y \in V \cup B$ taki, że $y \in def(a_2) \cap use(a_1)$ oraz zachodzi jeden z następujących warunków:

DD1 $t_2 = t_1$ i operacja a_2 występuje przed operacją a_1 oraz żadna operacja pomiędzy operacjami a_2 i a_1 nie definiuje y ,

DD2 $target(t_2) = source(t_1)$ lub istnieje ścieżka ze stanu $target(t_2)$ do stanu $source(t_1)$ taka, że żadna tranzycja należąca do tej ścieżki nie zawiera operacji definiującej y oraz żadna z operacji występujących po operacji a_2 w tranzycji t_2 ani żadna z operacji występujących przed operacją a_1 w tranzycji t_1 nie definiuje y ,

DD3 $i \neq j$.

Przykład 5.2 Na rysunku 5.1 pokazano przykładowe zależności danych w protokole zmieniającego się bitu. Operacja $put(bsa, bbit)$ z tranzycji o etykiecie $l14$ zależy od operacji $get(rba, bbit)$ z tej samej tranzycji, ponieważ operacja $put(bsa, bbit)$ używa zmiennej $bbit$, która jest definiowana przez operację $get(rba, bbit)$ i jest spełniony warunek **DD1** (pomiędzy tymi operacjami nie ma innych definicji zmiennej $bbit$). Również dozór $sack = sbit$ tranzycji¹ $l5$ zależy od operacji $get(bsa, sack)$ z akcji tranzycji $l3$, ponieważ zmienna $sack$ używana w operacji $sack = sbit$ jest definiowana w operacji $get(bsa, sack)$ i jest spełniony warunek **DD2** (stan źródłowy tranzycji $l5$ jest stanem docelowym tranzycji $l3$ i zmienna $sack$ nie jest definiowana po operacji $get(bsa, sack)$ w tranzycji $l3$ ani przed operacją $sack = sbit$ w tranzycji $l5$). Z kolei operacja $get(bsa, sack)$ z tranzycji $l3$, w której jest użyty bufor bsa zależy od operacji $put(bsa, bbit)$ z tranzycji $l14$ definiującej bufor bsa , na mocy warunku **DD3**.

W dalszej części rozdziału piszemy $a_1 \xrightarrow{dd1} a_2$, żeby zaznaczyć, że jest to wariant **DD1** relacji zależności danych między operacjami a_1 i a_2 (czyli $a_1 \xrightarrow{dd} a_2$ i jest spełniony warunek **DD1**). Podobnie dla warunków **DD2** i **DD3**. Przez $(\xrightarrow{dd})^*$ będziemy oznaczać domknięcie przechodnie relacji (\xrightarrow{dd}) .

5.1.2 Zależność przepływu sterowania

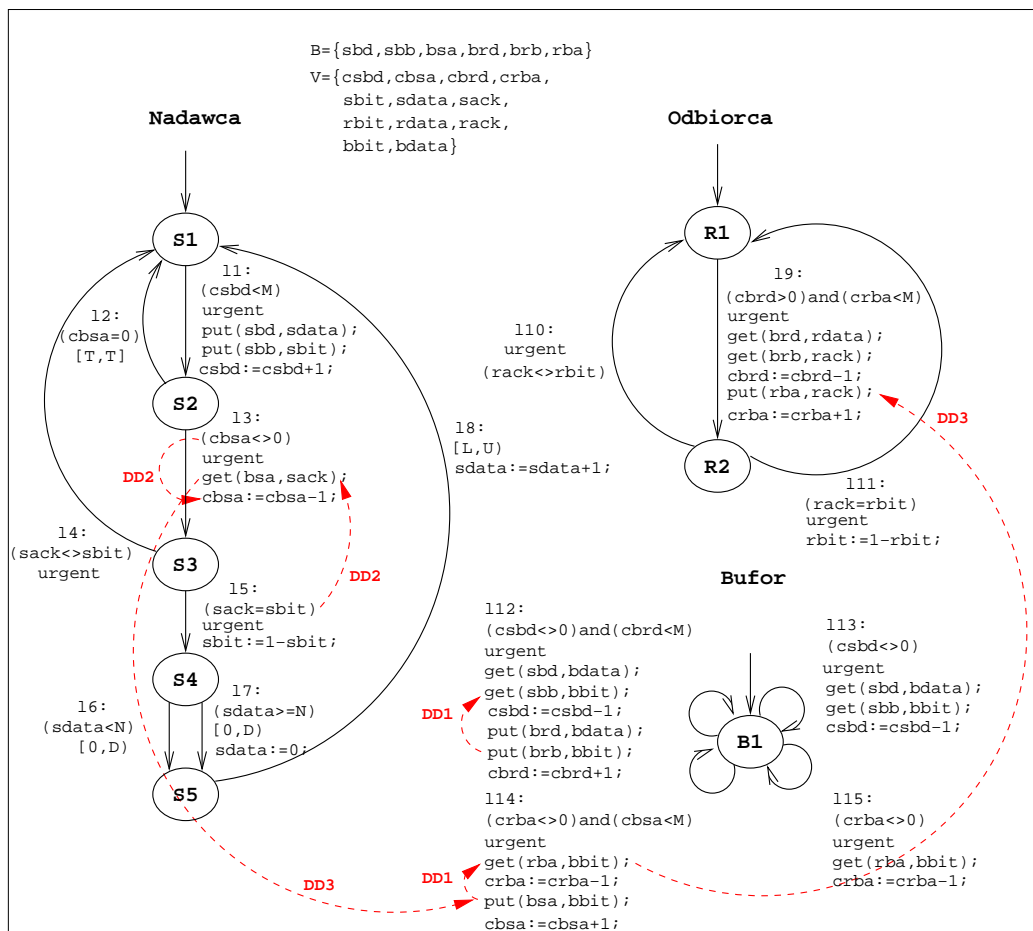
Zależność przepływu sterowania dotyczy wyłącznie stanów kontrolnych należących do jednego procesu. Klasyczna definicja zależności przepływu sterowania ([Tip95]) mówi, że stan q_1 zależy od stanu q_2 , jeżeli ze stanu q_2 istnieje ścieżka do stanu końcowego, do której należy stan q_1 i wszystkie ścieżki prowadzące do stanu końcowego ze stanów leżących pomiędzy stanem q_2 a stanem q_1 przechodzą przez stan q_1 . Definicja ta dotyczy języków z wyróżnionym jednym stanem końcowym i nie jest wystarczająca dla języka bazowego.

Intuicyjnie, stan kontrolny q_1 zależy od innego stanu kontrolnego q_2 i jest to zależność przepływu sterowania, jeżeli istnieje takie wykonanie programu, w którym proces znajduje się w stanie kontrolnym q_2 , a nie znajduje się w stanie kontrolnym q_1 . Taka sytuacja w języku bazowym jest możliwa nie tylko wtedy, gdy ze stanu q_2 istnieje ścieżka do stanu końcowego, do której nie należy stan q_1 , ale również wtedy, gdy istnieje ścieżka zawierająca cykl, do której nie należy stan q_1 . Należy zatem sprawdzić, czy istnieje ścieżka *maksymalna* ze stanu q_2 , do której nie należy stan q_1 . Ponadto proces może pozostawać w stanie q_2 wtedy, gdy żadna tranzycja wychodząca z tego stanu nie jest możliwa do wykonania.

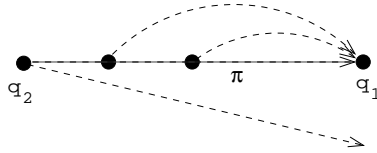
Definicja 5.3 (Zależność przepływu sterowania) *Mówimy, że stan kontrolny $q_1 \in Q_i$ zależy od stanu kontrolnego $q_2 \in Q_i$ ($q_2 \neq q_1$), gdzie $1 \leq i \leq n$, i jest to zależność przepływu sterowania (co oznaczamy $q_1 \xrightarrow{cd} q_2$), jeżeli stan q_1 jest osiągalny ze stanu q_2 i zachodzi co najmniej jeden z następujących warunków:*

$$\mathbf{CD1} \quad \left(\bigvee_{t \in out(q_2)} guard(t) \right) \neq true,$$

¹Będziemy używać krótszego określenia “tranzycja l ” zamiast “tranzycja o etykiecie l ”, o ile nie będzie prowadziło to do nieporozumień.



Rysunek 5.1: Przykłady zależności danych w protokole ABP



Rysunek 5.2: Zależność przepływu sterowania typu **CD2**

CD2 istnieje maksymalna ścieżka ze stanu q_2 , która nie przechodzi przez stan q_1 i istnieje ścieżka π z q_2 do q_1 taka, że każda maksymalna ścieżka z każdego stanu należącego do ścieżki π przechodzi przez stan q_1 (rys. 5.2).

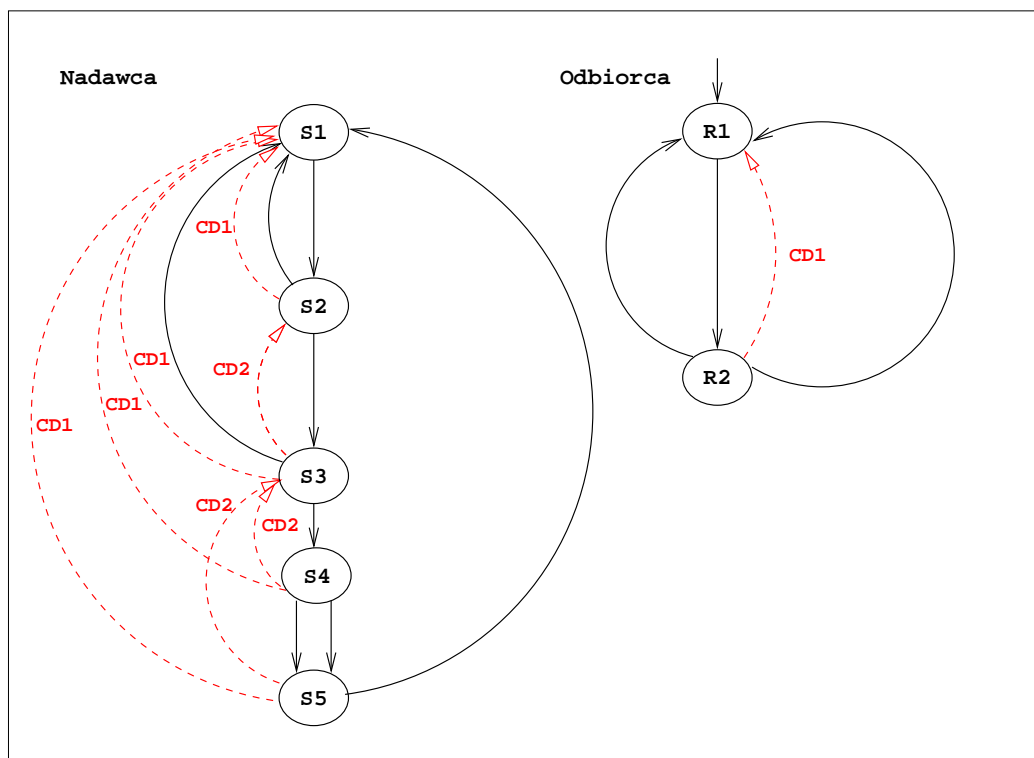
Przykład 5.4 Zależności przepływu sterowania dla stanów kontrolnych procesów *Nadawcy* i *Odbiorcy* w protokole zmieniającego się bitu pokazuje rysunek 5.3. W obydwu procesach każdy stan kontrolny jest osiągalny z każdego innego stanu kontrolnego. Wystarczy zatem sprawdzić pozostałe warunki. Ze stanu $S1$ procesu *Nadawcy* wychodzi jedna dozorowana tranzycja, a więc na mocy warunku **CD1** każdy z pozostałych stanów zależy od stanu $S1$. Analogiczna sytuacja występuje w procesie *Odbiorcy*.

Ze stanu $S3$ istnieje ścieżka zawierająca cykl (obejmujący stany $S1$, $S2$ i $S3$), do której nie należy stan $S4$ i żaden stan nie należy do ścieżki prostej ze stanu $S3$ do stanu $S4$, zatem stan $S4$ zależy od stanu $S3$ na mocy warunku **CD2**. Również stan $S5$ nie należy do ścieżki zawierającej ten cykl, a każda maksymalna ścieżka z każdego ze stanów należącego do ścieżki prostej ze stanu $S3$ do stanu $S5$ (czyli każda ścieżki wychodząca ze stanu $S3$) przechodzi przez stan $S4$, więc ponownie na mocy warunku **CD2** stan $S5$ zależy od stanu $S3$. Analogiczną sytuację mamy dla stanu $S3$ i cyklu, do którego należą stany $S1$ i $S2$ — na mocy warunku **CD2** stan $S3$ zależy od stanu $S2$. Zauważmy, że stany $S5$ i $S4$ nie są zależne od stanu $S2$, ponieważ nie jest spełniona druga część warunku **CD2**, a mianowicie ze stanu $S3$ istnieje ścieżka maksymalna, która nie przechodzi przez stan $S5$ ani przez stan $S4$.

W dalszej części rozdziału piszemy $q_1 \xrightarrow{cd1} q_2$, żeby zaznaczyć, że jest to wariant **CD1** relacji zależności przepływu sterowania między stanami q_1 i q_2 (czyli $q_1 \xrightarrow{cd} q_2$ i jest spełniony warunek **CD1**). Przez $(\xrightarrow{cd1})^*$ będziemy oznaczać domknięcie przechodnie relacji $(\xrightarrow{cd1})$. Podobnie dla warunku **CD2**.

5.1.3 Zależność czasowa

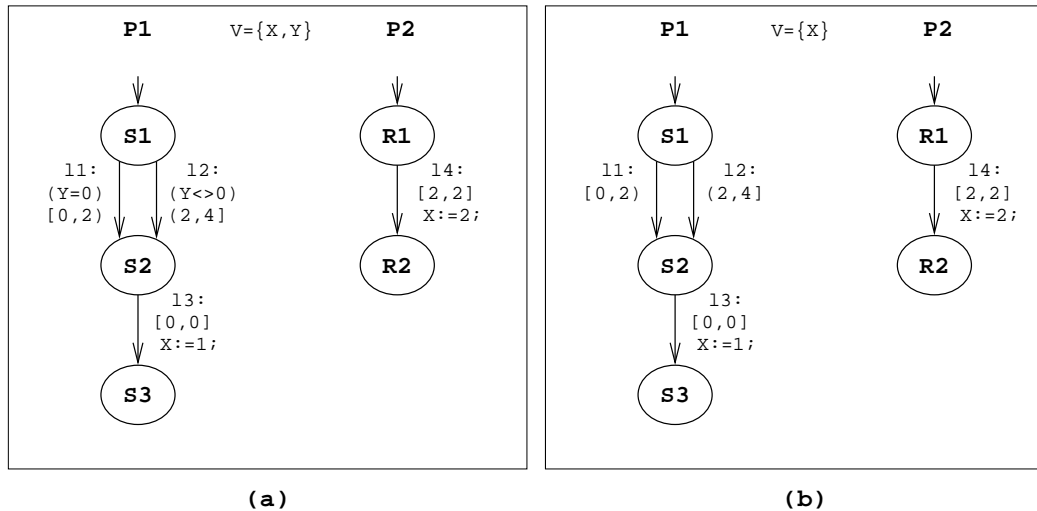
O programach, które nie zależą od czasu zakłada się przeważnie, że poszczególne ich instrukcje są wykonywane natychmiast, czyli bez wpływu czasu lub że są wykonywane w nieznanym, ale skończonym czasie. W przyjętym przez nas modelu każda tranzycja programu ma określone dozwolone opóźnienie, które może być nieskończone. Ograniczenia na czas wykonania tranzycji powodują, że w programie pojawiają się zależności nowego rodzaju.



Rysunek 5.3: Zależności przepływu sterowania w protokole ABP

Relacja zależności między stanami jest nam potrzebna nie tylko po to, aby wiedzieć jakie stany powinny się znaleźć w zredukowanym programie, ale także po to, aby wskazać dozory mające wpływ na weryfikowaną własność, ponieważ w zredukowanym programie znajdują się dozory tranzycji wychodzących ze stanów, od których zależą (między innymi) stany występujące w formule, której prawdziwość chcemy sprawdzić. Zależności czasowe zaprezentujemy najpierw na przykładach.

Przykład 5.5 Rozważmy program z rysunku 5.4 (a).



Rysunek 5.4: Przykład 1 zależności czasowej

Przyjmijmy, że $v^0(X) = 0$ i $v^0(Y) = 1$. Chcemy sprawdzić następującą własność: czy zmienna X może mieć wartość 2 po wykonaniu programu, czyli wtedy gdy obydwa procesy są w swoich stanach końcowych. Przeanalizujemy działanie programu. Przez pierwsze 2 jednostki czasu żadna z tranzycji nie może być wykonana, ponieważ tranzycja $l1$ nie jest możliwa, a tranzycje $l2$ i $l4$ nie są gotowe do wykonania. Dokładnie po 2 jednostkach czasu zostanie wykonana tranzycja $l4$. Następnie, zanim upłyną kolejne 2 jednostki — tranzycja $l2$ i natychmiast po niej $l3$. Zatem po wykonaniu programu zmienna X będzie miała wartość 1, czyli nasza własność nie jest prawdziwa.

Zauważmy, że w programie nie ma zależności danych pomiędzy operacjami definiującymi zmienną X , a operacjami, w których występuje zmienna Y . Nie ma również zależności przepływu sterowania. Jednak dozory tranzycji wychodzących ze stanu $S1$ mają wpływ na operacje definiujące zmienną X . Aby się o tym przekonać usuńmy z programu wszystkie operacje na zmiennej Y jak pokazano na rysunku 5.4 (b) i przeanalizujmy działanie tak zmodyfikowanego programu. Jest możliwy następujący przebieg:

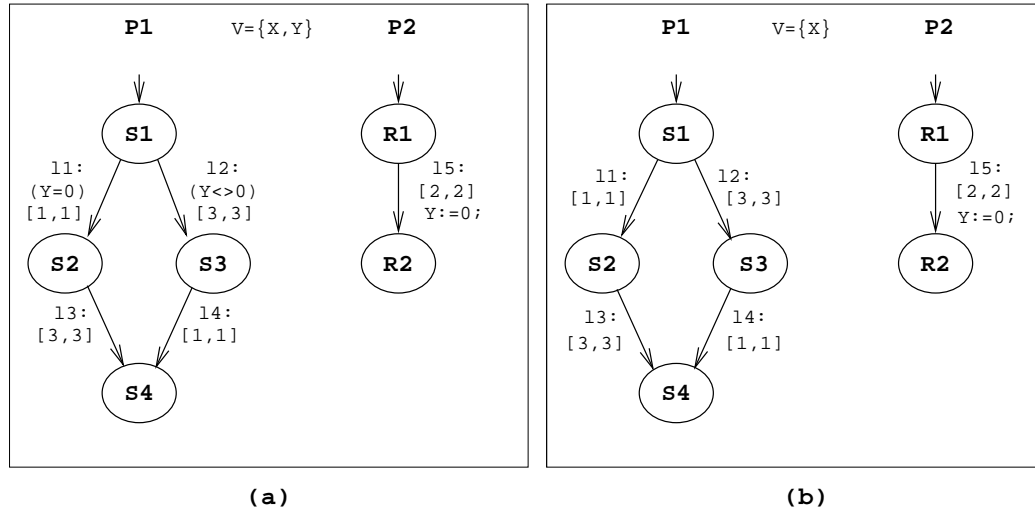
$$s_0 \xrightarrow{1} s_1 \xrightarrow{l1} s_2 \xrightarrow{l3} s_3 \xrightarrow{1} s_4 \xrightarrow{l4} s_5 \dots$$

Po takim wykonaniu programu wartość zmiennej X wynosi 2, więc nasza własność w tym przypadku jest prawdziwa.

W zaprezentowanym przykładzie pomiędzy stanem $S3$ a stanem $S1$ występuje zależność czasowa, ponieważ istotne jest nie tylko to, czy ze stanu $S1$ dotrzemy do stanu $S3$, ale także *kiedy* do niego dotrzemy.

Wprowadźmy jeszcze jedną zmianę do programu z rysunku 5.4(a) — zmienimy ograniczenia czasowe tranzycji $l1$ i $l2$ tak, aby były równe, na przykład $[0, 4]$ dla obydwu tranzycji. Dla tak zmienionego programu nasza własność jest prawdziwa niezależnie od tego czy operacje na zmiennej Y występują w programie, czy nie.

Przykład 5.6 Dla programu z rysunku 5.5(a) chcemy sprawdzić czy stan $S4$ jest osiągalny. Niech $v^0(X) = 0$ i $v^0(Y) = 1$. Przeanalizujemy działanie programu. Przez pierwsze 2



Rysunek 5.5: Przykład 2 zależności czasowej

jednostki czasu żadna z tranzycji nie może być wykonana, ponieważ tranzycja $l1$ nie jest możliwa, a tranzycje $l2$ i $l5$ nie są gotowe do wykonania. Dokładnie po 2 jednostkach czasu zostanie wykonana tranzycja $l5$. Zauważmy, że teraz nie można wykonać żadnej tranzycji, ponieważ tranzycja $l1$ nie jest gotowa do wykonania (minęły już 2 jednostki czasu od kiedy proces wszedł do stanu $S1$) ani tranzycja $l2$ nie jest gotowa do wykonania (bo nie jest możliwa). Zatem proces pozostaje w stanie $S1$ i stan $S4$ nie jest osiągalny.

Zauważmy, że nie ma zależności przepływu sterowania stanu $S4$ od stanu $S1$, ani zależności czasowej opisaną w poprzednim przykładzie. Jednak dozory tranzycji wychodzących ze stanu $S1$ mają wpływ na osiągalność stanu $S4$. Aby się o tym przekonać, tak jak poprzednio usuńmy te dozory z programu jak pokazano na rysunku 5.5 (b) i przeanalizujemy działanie tak zmodyfikowanego programu. Jest możliwy następujący przebieg:

$$s_0 \xrightarrow{1} s_1 \xrightarrow{l1} s_2 \xrightarrow{1} s_3 \xrightarrow{l5} s_4 \xrightarrow{2} s_5 \xrightarrow{l3} s_6 \dots$$

Jak widzimy istnieje przebieg, w którym proces $P1$ znajduje się w stanie $S4$.

Jeżeli zmienimy dozwolone opóźnienia tranzycji $l1$ i $l2$ w ten sposób, żeby były równe, to dla tak zmienionego programu nasza własność jest prawdziwa niezależnie od tego czy dozory tranzycji $l1$ i $l2$ występują w programie, czy nie. Dzieje się tak dlatego, że przedział czasu, w którym można wykonać każdą możliwą tranzycję jest wspólny i co najmniej jedna tranzycja jest możliwa do wykonania niezależnie od wartościowania.

Podsumowując, stan q_1 zależy od stanu q_2 i jest to zależność czasowa, jeżeli stan q_1 jest osiągalny ze stanu q_2 i jeżeli ze stanu q_2 do stanu q_1 prowadzą ścieżki różniące się dopuszczalnymi opóźnieniami, a także wtedy, gdy tranzycje wychodzące ze stanu q_2 mają różne opóźnienia i ich dozory są różne od $true$.

Opóźnienie na ścieżce

Dla każdej tranzycji programu zdefiniujemy teraz górne i dolne ograniczenia na czas jej wykonania, a dokładnie na czas, jaki proces może spędzić w stanie źródłowym tej tranzycji przed jej wykonaniem (w przyjętym przez nas modelu samo wykonanie tranzycji nie zajmuje czasu). Dla tranzycji, które nie są pilne, ograniczenia te wynikają wprost z dopuszczalnego opóźnienia. W przypadku tranzycji pilnej zauważmy, że jeżeli ze stanu źródłowego tej tranzycji wychodzą same tranzycje pilne lub z dozwolonym opóźnieniem równym $[0, 0]$ i zawsze jedna z nich jest możliwa do wykonania, to proces musi opuścić stan źródłowy tranzycji natychmiast po wejściu do niego (bez upływu czasu). W innych przypadkach powiemy, że górne i dolne ograniczenia tranzycji są nieznane.

Niech symbol \perp reprezentuje nieznaną wartość. Zdefiniujemy funkcje: $lower_bound : T \rightarrow \mathbb{N} \cup \{\perp\}$, $upper_bound : T \rightarrow \mathbb{N} \cup \{\infty\} \cup \{\perp\}$, $lower_bound_sign : T \rightarrow \{", "[", \perp\}$ i $upper_bound_sign : T \rightarrow \{")", "]" , \perp\}$. Dla tranzycji $t \in T$, która nie jest pilna i dla której dozwolone opóźnienie jest w postaci $\langle d_1, d_2 \rangle$, gdzie $\langle \in \{", "[", \perp\}$, $\rangle \in \{")", "]" , \perp\}$, $d_1 \in \mathbb{N}$ i $d_2 \in \mathbb{N} \cup \{\infty\}$:

- $lower_bound(t) = d_1$,
- $upper_bound(t) = d_2$,
- $lower_bound_sign(t) = \begin{cases} "(" & \text{jeżeli } delay(t) \text{ jest w postaci } \langle d_1, d_2 \rangle, \\ "[" & \text{w przeciwnym przypadku,} \end{cases}$
- $upper_bound_sign(t) = \begin{cases} ")" & \text{jeżeli } delay(t) \text{ jest w postaci } \langle d_1, d_2 \rangle, \\ "]" & \text{w przeciwnym przypadku.} \end{cases}$

Niech $t \in T$ będzie tranzycją pilną. Jeżeli są spełnione dwa następujące warunki:

1. $(\forall t' \in out(source(t)) \ guard(t')) = true$ i
2. $\forall t' \in out(source(t)) \ (synch(t') = false \wedge (urgent(t') = true \vee delay(t') = [0, 0]))$,

to przyjmujemy że:

$$\begin{aligned} lower_bound(t) &= upper_bound(t) = 0, \\ lower_bound_sign(t) &= "[", \quad upper_bound_sign(t) = "]" , \end{aligned}$$

w przeciwnym przypadku powiemy, że opóźnienie tranzycji jest *nieznane* i przyjmujemy, że $lower_bound(t) = upper_bound(t) = lower_bound_sign(t) = upper_bound_sign(t) = \perp$.

Dla ścieżki $\pi = q_1 t_1 q_2 \dots t_{m-1} q_m$, możemy określić *opóźnienie na ścieżce* ($delay(\pi)$) w następujący sposób: $delay(\pi) = \langle d_1, d_2 \rangle$, gdzie (przyjmujemy, że $\infty + \infty = \infty$ oraz dla dowolnego $d \in N$, $d + \infty = \infty$ i $\infty + d = \infty$, a także $d + \perp = \perp$ i $\perp + d = \perp$):

- $d_1 = \sum_{j=1}^{m-1} lower_bound(t_j)$,
- $d_2 = \sum_{j=1}^{m-1} upper_bound(t_j)$,
- $\langle = \begin{cases} \text{"["} & \text{jeżeli } lower_bound_sign(t_i) = \text{"["} \text{ dla każdego} \\ & 1 \leq i < m \\ \text{"("} & \text{w przeciwnym przypadku,} \end{cases}$
- $\rangle = \begin{cases} \text{"]"} & \text{jeżeli } upper_bound_sign(t_i) = \text{"]"} \text{ dla każdego} \\ & 1 \leq i < m \\ \text{")"} & \text{w przeciwnym przypadku.} \end{cases}$

Podobnie jak dla dozwolonego opóźnienia tranzycji, d_1 , d_2 , \langle i \rangle będziemy oznaczać przez $lower_bound(\pi)$, $upper_bound(\pi)$, $lower_bound_sign(\pi)$ i $upper_bound_sign(\pi)$, odpowiednio.

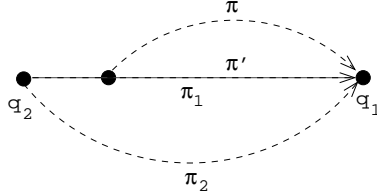
Powiemy, że dwie ścieżki π_1 i π_2 mają równe opóźnienia (piszemy $delay(\pi_1) = delay(\pi_2)$), jeżeli:

- $lower_bound(\pi_1) = lower_bound(\pi_2)$,
- $upper_bound(\pi_1) = upper_bound(\pi_2)$,
- $lower_bound_sign(\pi_1) = lower_bound_sign(\pi_2)$,
- $upper_bound_sign(\pi_1) = upper_bound_sign(\pi_2)$.

W przeciwnym przypadku mówimy, że ścieżki π_1 i π_2 mają różne opóźnienia, co oznaczamy $delay(\pi_1) \neq delay(\pi_2)$.

Przykład 5.7 W tabelce poniżej pokazane są opóźnienia na ścieżkach prostych łączących każde dwa stany w procesie *Nadawcy*. Na przykład ze stanu $S3$ do stanu $S1$ można dojść dokładnie w czasie 0 (gdy zostanie wykonana tranzycja $l4$) lub czasie nie krótszym niż L i krótszym niż $U + D$ (gdy zostaną wykonane tranzycje $l6$ lub $l7$, a następnie $l8$).

do/z	S1	S2	S3	S4	S5
S1	–	$[T, T]$ lub \perp	$[0, 0]$ lub $[L, U + D)$	$[L, U + D)$	$[L, U)$
S2	\perp	–	\perp	\perp	\perp
S3	\perp	\perp	–	\perp	\perp
S4	\perp	\perp	$[0, 0]$	–	\perp
S5	\perp	\perp	$[0, D)$	$[0, D)$	–



Rysunek 5.6: Zależność czasowa typu **TD2**

Zależność czasowa

Definicja 5.8 (Zależność czasowa) Mówimy, że stan kontrolny $q_1 \in Q_i$ zależy od stanu kontrolnego $q_2 \in Q_i$ ($q_2 \neq q_1$), gdzie $1 \leq i \leq n$, i jest to zależność czasowa (co oznaczamy $q_1 \xrightarrow{td} q_2$), jeżeli stan q_1 jest osiągalny ze stanu q_2 i zachodzi co najmniej jeden z następujących warunków:

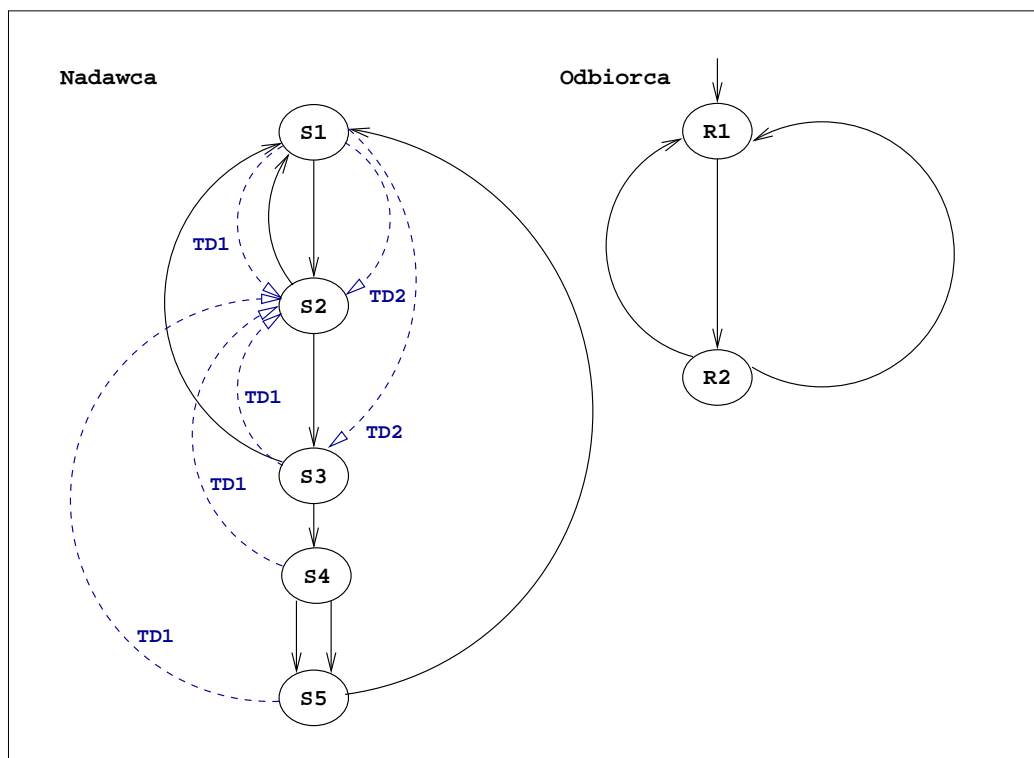
TD1 istnieją dwie tranzycje $t_1, t_2 \in out(q_2)$ takie, że $delay(t_1) \neq delay(t_2)$ i dla każdej tranzycji $t \in out(q_2)$, $guard(t) \neq true$,

TD2 istnieją co najmniej dwie ścieżki proste π_1 i π_2 z q_2 do q_1 takie, że $delay(\pi_1) \neq delay(\pi_2)$ i istnieje ścieżka ze stanu q_2 do stanu q_1 taka, że dla każdego stanu $q \neq q_2$ należącego do tej ścieżki zachodzi $\forall_{\pi, \pi' \in \Pi(q, q_1)} delay(\pi) = delay(\pi')$ (rys. 5.6).

Zauważmy, że jeżeli opóźnienia na wszystkich ścieżkach prostych z q_2 do q_1 są nieznanne (równe \perp), to nie ma zależności czasowej q_1 od q_2 typu **TD2**. W takiej sytuacji o czasie w jakim proces znajdzie się w stanie q_1 decydują inne czynniki takie jak synchronizacja z innymi procesami, czy wartości zmiennych, bo od nich zależy moment wykonania tranzycji pilnych.

Podobnie jak poprzednio piszemy $q_1 \xrightarrow{td1} q_2$, żeby zaznaczyć, że jest to wariant **TD1** relacji zależności czasowej między stanami q_1 i q_2 (czyli $q_1 \xrightarrow{td} q_2$ i jest spełniony warunek **TD1**). Podobnie dla warunku **TD2**. Przez $(\xrightarrow{td})^*$ będziemy oznaczać domknięcie przechodnie relacji (\xrightarrow{td}) .

Przykład 5.9 Aby zobaczyć jakie zależności czasowe występują w procesie *Nadawcy* przeanalizujmy tabelę ze strony 78. Ścieżki, na których opóźnienie jest różne występują tylko między stanem $S2$ a $S1$ oraz między stanem $S3$ a $S1$. Dla obydwu par stanów warunek **TD2** jest spełniony, ponieważ istnieją ścieżki ($S2-l2-S1$ i $S3-l4-S1$), do których nie należą inne stany (więc ostatnia część warunku jest spełniona). Zatem: $S1 \xrightarrow{td2} S2$ i $S1 \xrightarrow{td2} S3$. Natomiast warunek **TD1** jest spełniony dla stanu $S2$, więc dla każdy z pozostałych stanów procesu *Nadawcy* zależy od stanu $S2$. Wszystkie zależności czasowe przedstawiono na rysunku 5.7.



Rysunek 5.7: Zależności czasowe w protokole ABP

5.1.4 Zależność od synchronizacji

Zauważmy, że proces może pozostawać w stanie q_2 , jeżeli nie jest spełniony żaden z dozorów tranzycji wychodzących z tego stanu (zależność typu **CD1**), ale także wtedy, gdy jedna z tranzycji wychodzących ze stanu q_2 jest synchroniczna (i inny proces nie może wykonać tranzycji o tej samej etykiecie). Wystarczy, aby jedna z tranzycji wychodzących ze stanu q_2 była synchroniczna, aby proces pozostawał w stanie q_2 , ponieważ pozostałe tranzycje mogą w danej chwili nie być możliwe do wykonania.

Definicja 5.10 (Zależność od synchronizacji) Powiemy, że stan $q_1 \in Q_i$ zależy od stanu $q_2 \in Q_i$ ($q_2 \neq q_1$), gdzie $1 \leq i \leq n$, i jest to zależność od synchronizacji (co oznaczamy $q_1 \xrightarrow{sd} q_2$), jeżeli stan q_1 jest osiągalny ze stanu q_2 i zachodzi następujący warunek:

SD $\exists_{t \in out(q_2)} synch(t) = true$.

Przykład 5.11 W przykładzie zmieniającego się bitu nie ma zależności od synchronizacji, ponieważ jest on w pełni asynchroniczny, czyli żadna występująca w nim tranzycja nie jest synchroniczna. Dlatego zależność od synchronizacji pokażemy w synchronicznym protokole Fischera wzajemnego wykluczania (rozdz. 3.4.2). W tym przykładzie z kolei w każdym z procesów wszystkie tranzycje są synchroniczne i każdy stan jest osiągalny z każdego innego stanu, dlatego wszystkie stany danego procesu zależą od siebie nawzajem.

Zależność stanów

Niech \xrightarrow{d} będzie sumą relacji zależności przepływu sterowania, synchronizacji i czasowej. Powiemy, że stan kontrolny $q_1 \in Q_i$ zależy od stanu kontrolnego $q_2 \in Q_i$, gdzie $1 \leq i \leq n$ (co oznaczamy $q_1 \xrightarrow{d} q_2$), jeżeli $q_1 \xrightarrow{cd} q_2$ lub $q_1 \xrightarrow{td} q_2$ lub $q_1 \xrightarrow{sd} q_2$. Przez $(\xrightarrow{d})^*$ będziemy oznaczać domknięcie przechodnie relacji (\xrightarrow{d}) .

5.2 Zredukowany program

Do tej pory pokazaliśmy jakiego rodzaju zależności występują w programie. W następnej części będziemy korzystać z wprowadzonych zależności do ustalenia, które fragmenty programu mają wpływ na prawdziwość weryfikowanych własności i przedstawimy metodę redukcji programów w języku bazowym. Metoda redukcji zależy od formuły logicznej wyrażającej własność, której prawdziwość chcemy sprawdzić, a dokładniej — od zbioru zmiennych zdaniowych, które wchodzi w skład formuły. Punktem wyjściowym jest ustalenie początkowych zbiorów operacji i stanów kontrolnych, od których bezpośrednio zależą wartości zmiennych zdaniowych z formuły, czyli tak zwanego *kryterium cięcia* (ang. *slicing criterion*) [Tip95]. Operacje, etykiety i stany kontrolne, od których zależą (bezpośrednio lub pośrednio) wartości zmiennych zdaniowych z formuły będziemy nazywać *istotnymi*. Wszystkie istotne operacje i stany kontrolne będą wchodzić w skład procesów zredukowanego programu.

Zmienne, bufory i stany obserwowane

Przypuśćmy, że chcemy sprawdzić prawdziwość formuły φ dla programu \mathcal{P} . Niech $PV_\varphi \subseteq PV$ oznacza zbiór zmiennych zdaniowych występujących w formule φ . Niech $states_i(PV_\varphi) = \{q \in Q_i \mid p_{i,q} \in PV_\varphi\}$ będzie zbiorem *obserwowanych stanów kontrolnych* i -tego procesu, czyli stanów kontrolnych tego procesu, które występują w definicjach zmiennych zdaniowych formuły φ . Niech również $vars(PV_\varphi) = \{y \in vars(e_1) \cup vars(e_2) \mid p_{e_1} \sim e_2 \in PV_\varphi\} \cup \{b \in B \mid p_{empty(b)} \in PV_\varphi\}$ będzie zbiorem *obserwowanych zmiennych i buforów*, czyli zmiennych i buforów występujących w wyrażeniach definiujących zmienne zdaniowe z formuły φ .

5.2.1 Kryterium cięcia

Definicja 5.12 (Kryterium cięcia) Kryterium cięcia dla programu $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$ i zbioru zmiennych zdaniowych PV_φ stanowią: zbiór operacji $A^0 \subseteq Ops$ i zbiory stanów kontrolnych $R_i^0 \subseteq Q_i$ dla $1 \leq i \leq n$, zdefiniowane następująco:

- $A^0 = \{a \in Ops \mid def(a) \cap vars(PV_\varphi) \neq \emptyset\}$
- $R_i^0 = states_i(PV_\varphi)$

W skład kryterium cięcia dla operacji wchodzi operacje definiujące zmienne i bufory obserwowane. Natomiast kryterium cięcia dla stanów kontrolnych określa stany obserwowane.

Zauważmy, że kryterium cięcia zależy od zbioru zmiennych zdaniowych, a nie od konkretnej formuły, co oznacza, że może być definiowane tak samo dla wielu formuł zbudowanych ze zmiennych zdaniowych należących do tego zbioru.

Przykład 5.13 Przypuśćmy, że dla protokołu zmieniającego się bitu chcemy sprawdzić prawdziwość formuły $\varphi = AG((p_{1.S1} \wedge p_{2.R1} \wedge p_{sbit=0}) \Rightarrow rp_{rbit=0})$ (opisanej w rozdz. 3.4.3). Zgodnie z definicją kryterium cięcia, zbiór A^0 zawiera operacje definiujące zmienne $sbit$ i $rbit$: $sbit := 1 - sbit$ z tranzycji $l5$ i $rbit := 1 - rbit$ z tranzycji $l11$. Natomiast zbiory R_i^0 składają się ze stanów początkowych *Nadawcy* i *Odbiorcy* (które występują w definicjach zmiennych zdaniowych formuły φ): $R_1^0 = \{S1\}$ i $R_2^0 = \{R1\}$. Natomiast zbiór R_3^0 (dla procesu *Bufora*) jest pusty.

5.2.2 Istotne operacje

Algorytm 1 przedstawia sposób obliczania istotnych operacji. Zbiory R_i , R'_i i R''_i zawierają stany kontrolne, zbiory L i L' — etykiety, a zbiory A , A' i A'' — operacje. Początkowo, w zbiorze R_i znajdują się obserwowane stany procesu i -tego, a zbiory R'_i są puste (zbiory R''_i są pomocnicze i służą do przechowywania wyników częściowych). Zbiór A zawiera na początku operacje definiujące zmienne i bufory obserwowane, a zbiór A' jest pusty. Zbiory L i L' są początkowo puste.

W czasie działania algorytmu zbiory R'_i zawierają zbiory obliczone w poprzednim kroku, czyli w poprzednim obrocie głównej pętli **while** i służą do porównania z nowo obliczonymi zbiorami R'_i . Analogicznie dla zbiorów A i A' , które zawierają operacje istotne obliczone do tej pory oraz L i L' , które zawierają istotne etykiety (synchronizujące) obliczone do tej pory. W każdym obrocie pętli do zbiorów R_i dla każdego $1 \leq i \leq n$ dodane są stany, z których

Algorytm 1 Algorytm obliczania istotnych operacji

```
1:  $A = A^0; L = \emptyset; A' = \emptyset; L' = \emptyset;$ 
2: for  $i = 1$  to  $n$  do
3:    $R_i = R_i^0; R'_i = \emptyset;$ 
4: end for
5: while  $\exists_{i \in \{1, \dots, n\}} (R_i \neq R'_i) \vee (A \neq A') \vee (L \neq L')$  do
6:   for  $i = 1$  to  $n$  do
7:      $R'_i = R_i;$ 
8:      $R''_i = R_i \cup \{q \in Q \mid \exists_{t \in \text{out}(q)} \text{label}(t) \in L \vee (\text{opers}(t) \cap A \neq \emptyset)\};$ 
9:      $R_i = R''_i \cup \{q \in Q \mid \exists_{q' \in R''_i} q' \xrightarrow{d} q\};$ 
10:  end for
11:   $A' = A; L' = L;$ 
12:   $A'' = A \cup \bigcup_{i=1}^n \{\text{guards}(q) \mid \exists_{q' \in R_i} q' \xrightarrow{d} q \vee \exists_{t \in \text{out}(q)} \text{opers}(t) \cap A \neq \emptyset\};$ 
13:   $A = A'' \cup \{a \in \text{Ops} \mid \exists_{a' \in A''} a' \xrightarrow{dd} a\};$ 
14:   $L = L' \cup \bigcup_{i=1}^n \{l \in \Gamma \mid |\Gamma(l)| > 1 \wedge \exists_{q \in R_i} \exists_{t \in \text{out}(q)} l = \text{label}(t)\};$ 
15: end while
16:  $A^\varphi = A;$ 
```

wychodzą tranzycje z istotną etykietą lub takie, w których występują istotne operacje oraz te wszystkie stany, od których zależą stany obliczone do tej pory (domknięcie przechodnie relacji zależności stanów). Podobnie, zbiór operacji zostaje (ewentualnie) powiększony o wszystkie dozory tranzycji wychodzących ze stanów: od których zależą stany należące do R_i lub z których wychodzą tranzycje posiadające istotne akcje obliczone do tej pory. Następnie do zbioru A dodajemy te wszystkie operacje, od których zależą operacje istotne obliczone do tej pory (domknięcie przechodnie relacji zależności danych). Wreszcie do zbioru L dodajemy etykiety synchronizujące wychodzące ze stanów należących do zbiorów R_i .

Wykonanie algorytmu kończy się, gdy w ostatnim kroku do żadnego ze zbiorów R_i , A ani L nie dodano nowych elementów. Po zakończeniu działania zbiór A zawiera wszystkie istotne operacje programu, a zbiór L — wszystkie etykiety, które są istotne dla działania programu.

Przykład 5.14 Prześledźmy działanie algorytmu 1 na przykładzie protokołu zmieniającego się bitu (punkt 3.4.3). Zauważmy, że zbiór L będzie zawsze pusty, ponieważ w protokole nie występują tranzycje synchroniczne. Początkowo zbiór A zawiera operacje $sbit := 1 - sbit$ z tranzycji $l5$ i $rbit := 1 - rbit$ z tranzycji $l11$, $R_1^0 = \{S1\}$, $R_2^0 = \{R1\}$ i $R_3^0 = \emptyset$ (kryterium cięcia).

W pierwszym kroku (w pierwszym obrocie głównej pętli) do zbioru R_1 dodajemy stan $S3$ (wiersz 8: jedna z tranzycji wychodzących z tego stanu zawiera operacje z A) i stan $S2$ (wiersz 9: $S1 \xrightarrow{td2} S2$), a do zbioru R_2 dodajemy stan $R2$ (wiersz 8: tranzycja $l11$ wychodząca z tego stanu zawiera operacje z A). Zbiór R_3 nie zmienia się. Następnie (wiersz 12) do zbioru A dodajemy dozory tranzycji: $l1$ ($S2 \xrightarrow{cd1} S1$), $l2$ i $l3$ ($S1 \xrightarrow{td2} S2$), $l4$ i $l5$ ($S1 \xrightarrow{td2} S3$), $l9$ ($R2 \xrightarrow{cd1} R1$), $l10$ i $l11$ (tranzycja $l11$ wychodząca ze stanu $R2$ zawiera operacje z A). Teraz kolej na zależność danych (wiersz 13). Do zbioru A dołączamy

operacje: $csbd := csbd + 1$ z tranzycji $l1$ (zależy od niej dozór tranzycji $l1$), $get(bsa, sack)$ z tranzycji $l3$ (zależy od niej dozór tranzycji $l4$), $cbsa := cbsa - 1$ z tranzycji $l3$ (zależy od niej dozór tranzycji $l3$), $get(brb, rack)$ z tranzycji $l9$ (zależy od niej dozór tranzycji $l10$), $cbrd := cbrd - 1$ i $crba := crba + 1$ z tranzycji $l9$ (zależy od nich dozór tranzycji $l3$), itd. W tym kroku do zbioru A (ze względu na zależność danych **DD3** i **DD1** w procesie *Bufora*) zostaną dodane jeszcze wszystkie pozostałe operacje definiujące bufor: sbb , bsa , brb i rba oraz zmienne: $csbd$, $csba$, $cbrd$ i $crba$.

W drugim obrocie głównej pętli do zbioru R_3 dodajemy stan $B1$ (wychodzące z niego tranzycje mają istotne operacje). Pozostałe zbiory stanów nie zmieniają się. Do zbioru A dodajemy dozory wszystkich tranzycji w procesie *Bufora* (tranzycje te zawierają istotne operacje). W trzecim obrocie pętli zawartość żadnego ze zbiorów R_i , A ani L nie ulega zmianie i algorytm się kończy. Jak widzimy operacjami istotnymi są wszystkie operacje definiujące bufor: sbb , bsa , brb i rba oraz zmienne: $csbd$, $csba$, $cbrd$ i $crba$ oraz $sbit$, $rbit$, $bbit$, $sack$ i $rack$. Operacjami nieistotnymi są wszystkie pozostałe operacje, czyli operacje na zmiennych $sdata$, $rdata$ i $bdata$ oraz na buforach sbd i brd .

5.2.3 Istotne stany kontrolne

Teraz szukamy stanów, które powinny znaleźć się w zredukowanym procesie, czyli stanów istotnych. Niech $T_i^\varphi = \{t \in T_i \mid label(t) \in L \vee (opers(t) \cap A^\varphi) \neq \emptyset\}$ będzie zbiorem *istotnych tranzycji* (tranzycja jest *istotna*, jeżeli ma istotne operacje w swoim dozorze lub w akcji, a także wtedy, gdy jej etykieta jest istotna).

Przez $q_1 \xrightarrow{d'} q_2$ oznaczmy, że $q_1 \xrightarrow{cd2} q_2$ lub $q_1 \xrightarrow{td} q_2$, a przez $(\xrightarrow{d'})^*$ domknięcie przechodnie relacji $(\xrightarrow{d'})$.

Algorytm 2 Algorytm obliczania istotnych stanów

- 1: **for** $i = 1$ to n **do**
 - 2: $Q_i^O = R_i^0 \cup \{q \in Q_i \mid \exists t \in in(q) \ source(t) \in R_i^0\}$;
 - 3: $Q_i^A = \{q \in Q_i \mid \exists t \in in(q) \ t \in T_i^\varphi\}$;
 - 4: $Q_i^U = Q_i^O \cup Q_i^A \cup \{q \in Q_i \mid \exists t \in out(q) \ target(t) \in Q_i^A \wedge urgent(t)\}$;
 - 5: $Q_i^D = \{q \in Q_i \mid \exists q' \in Q_i^U \ q' (\xrightarrow{d'})^* q\}$;
 - 6: $Q_i^N = Q_i^U \cup Q_i^D \cup \{q \in Q_i \mid \exists t \in in(q) \ source(t) \in Q_i^D\}$;
 - 7: $Q_i^R = \{q \in Q_i \mid \exists q' \in Q_i^N \ q \implies q'\}$;
 - 8: $Q_i^\varphi = Q_i^N \cup \{q_i^0\} \cup \{q \in Q_i \setminus Q_i^R \mid \exists t \in in(q) \ source(t) \in Q_i^R\}$;
 - 9: **end for**
-

Główna zasada redukcji programu jest następująca: stany, których nie uznajemy za istotne będziemy “sklejać” z ich poprzednikami (rekurencyjnie). W stanach, do których “przykleimy” inne stany, proces będzie mógł przebywać odpowiednio dłużej. Osiągniemy to przez odpowiednią modyfikację dozwolonego opóźnienia tranzycji wychodzących z takich stanów. Opóźnienie takich tranzycji będzie równe sumie opóźnień na ścieżkach składających się z nieistotnych stanów.

Budowanie zbiorów stanów istotnych odbywa się etapami (dla każdego procesu niezależnie). W algorytmie 2 budujemy kolejno następujące zbiory:

- Q_i^O — stanami istotnymi są oczywiście stany obserwowane. Dołączenie następników stanów obserwowanych jest konieczne, ponieważ nie chcemy utożsamić (skleić) dwóch stanów, z których jeden jest obserwowany, a drugi nie.
- Q_i^A — istotne są stany, do których wchodzi istotne tranzycje. Ponownie, nie możemy skleić dwóch stanów, jeżeli pomiędzy tymi stanami istnieje istotna tranzycja.
- Q_i^U — zawiera sumę obliczonych do tej pory zbiorów stanów oraz stany, z których wychodzą tranzycje pilne wchodzące do stanów należących do Q_i^A . Takich stanów nie możemy skleić z ich poprzednikami, ponieważ nie jest możliwe podanie opóźnienia dla istotnej tranzycji pilnej.
- Q_i^D — stany, od których zależą na mocy zależności czasowej lub przepływu sterowania typu **CD2** stany obliczone do tej pory (należące do Q_i^U). W zredukowanym programie muszą się znaleźć wszystkie istotne “rozgałęzienia” (także ze względu na czas). Zauważmy, że stany, dla których jest prawdziwy warunek **CD1**, nie są istotne (istotne są tylko dozory tranzycji wychodzących z tych stanów).
- Q_i^N — zawiera sumę obliczonych do tej pory zbiorów stanów oraz następniki stanów z Q_i^D , ponieważ chcemy zachować strukturę programu (**CD2**) oraz dokładny czas przebywania w stanach, od których zależą inne stany zgodnie z zależnością czasową.
- Q_i^R — stany, z których są osiągalne stany z Q_i^N (zbiór pomocniczy, w przeciwieństwie do pozostałych zbiorów nie będzie zawierał się w całości w ostatecznym zbiorze stanów istotnych).
- Q_i^φ — suma obliczonych do tej pory zbiorów stanów, stan początkowy oraz stany, które nie należą do Q_i^R i mają poprzedniki w Q_i^R (reprezentanci stanów, z których nie są osiągalne inne stany istotne).

Przykład 5.15 Pokażemy jak budujemy zbiór stanów istotnych dla procesu *Nadawcy* w protokole zmieniającego się bitu. Zbiór Q_1^O składa się ze stanu $S1$ (obserwowanego) i $S2$ (następnika $S1$). Do zbioru Q_1^A należą stany $S1$, $S2$, $S3$ i $S4$ (tranzycje wchodzące do tych stanów są istotne), a zbiór Q_1^U jest równy zbiorowi Q_1^A . Zbiór Q_1^D składa się ze stanów $S2$ i $S3$ (zależy od nich na przykład stan $S1$), a Q_1^N — ze stanów $S1$, $S3$ i $S4$ (następniki stanów z Q_1^D). Wreszcie $Q_1^R = Q_1$ i $Q_1^\varphi = \{S1, S2, S3, S4\}$.

Złożoność algorytmu

Zbadajmy teraz złożoność przedstawionego algorytmu redukcji. Dla dwóch operacji $a_1 \in T_i$ i $a_2 \in T_j$ czas sprawdzenia, czy między a_1 a a_2 istnieje zależność danych, jest liniowy względem liczby operacji $|A|$, stanów kontrolnych $|Q|$ i tranzycji $|T|$. Istotnie, warunek $def(a_2) \cap use(a_1) \neq \emptyset$ i **DD3** sprawdzamy w czasie stałym. Sprawdzenie warunku **DD1** jest liniowe względem liczby operacji. Natomiast warunek **DD2** wymaga znalezienia wszystkich ścieżek pomiędzy danymi dwoma stanami. W tym celu wykorzystujemy algorytm przeszukiwania grafu włąb (DFS) o złożoności $O(|Q_i| + |T_i|)$ [CLRS01], gdzie i jest numerem procesu, do którego należą operacje a_1 i a_2 .

W obecnej implementacji zależność przepływu sterowania typu **CD1** między dwoma stanami q_1 i q_2 procesu P_i obliczamy w następujący sposób. Przyjmujemy, że stan q_1 nie zależy od stanu q_2 , jeżeli (1) istnieje tranzycja $t \in out(q_2)$ taka, że $guard(t) = true$ lub (2) ze stanu q_2 wychodzą dokładnie dwie tranzycje t_1 i t_2 takie, że $guard(t_1) = \neg guard(t_2)$ (czyli jest to odpowiednik instrukcji *if-then-else*). W pozostałych przypadkach uznajemy, że stan q_1 zależy od stanu q_2 . Zbadanie takiego warunku jest realizowane w czasie stałym. Natomiast zależność typu **CD2** sprawdzamy w czasie $O((|Q_i| + |T_i|)^2)$ (ponownie wykorzystujemy algorytm DFS do sprawdzenia, czy istnieje maksymalna ścieżka ze stanu q_2 , która nie przechodzi przez stan q_1 oraz do sprawdzenia warunku przeciwnego dla wszystkich stanów należących do ścieżek z q_2 do q_1). Koszt zbadania złożoności czasowej dla dwóch stanów jest taki sam jak koszt zbadania zależności przepływu sterowania, natomiast sprawdzenie zależności od synchronizacji wykonujemy w czasie stałym.

Początkowo, dla każdej operacji $a \in A$ obliczamy zbiory $vars(a)$, $def(a)$ i $use(a)$, a także macierze osiągalności stanów kontrolnych dla każdego z procesów. Koszt konstrukcji macierzy dla i -tego procesu jest ograniczony z przez $(|Q_i| + |T_i|) \times |Q_i|^2$. W algorytmie 1, co najwyżej raz wstawiamy każdą operację do zbioru A , a więc dla każdej operacji co najwyżej raz sprawdzamy, od których z pozostałych operacji ona zależy. Podobnie jest dla stanów kontrolnych. Zatem łączny koszt badania zależności między operacjami wynosi $O((|Q| + |T| + |A|) \times |A|^2)$, a między stanami $O((|Q| + |T|)^2 \times |Q|^2)$. Koszt wykonania algorytmu 2 możemy zaniedbać, ponieważ wykorzystujemy w nim informacje o zależnościach obliczone w algorytmie 1. Stąd otrzymujemy, że czas obliczania istotnych operacji i stanów kontrolnych jest wielomianowy względem łącznej liczby operacji, tranzycji i stanów kontrolnych.

5.2.4 Zredukowany program

Akcje i dozory zredukowane do operacji istotnych

Dla tranzycji $t \in T$ akcję tranzycji zredukowaną do istotnych operacji definiujemy jako:

$$action(t)|_{A^\varphi} = a_1 a_2 \dots a_k,$$

gdzie $a_j \in ops(action(t)) \cap A^\varphi$ dla każdego $j = 1, \dots, k$ przy czym kolejność występowania operacji a_j jest taka sama jak w akcji $action(t)$. Dozór tranzycji zredukowany do istotnych operacji definiujemy następująco:

$$guard(t)|_{A^\varphi} = \begin{cases} guard(t) & \text{jeżeli } ops(guard(t)) \in A^\varphi \\ true & \text{w przeciwnym przypadku.} \end{cases}$$

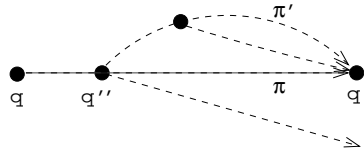
Ścieżki niewidoczne

Powiemy, że ścieżka $q_1 t_1 q_2 t_2 \dots q_m$ ze stanu $q_1 \in Q_i$ do stanu $q_m \in Q_i$ jest *niewidoczna*, jeżeli $m > 2$ i $q_j \in Q_i \setminus Q_i^\varphi$ dla każdego $1 \leq j \leq m - 1$ lub $m = 2$ i $q_2 \in Q_i \setminus Q_i^\varphi$, czyli wszystkie stany należące do niewidocznej ścieżki są nieistotne i co najmniej jeden stan różny od q_1 i q_m należy do ścieżki lub ścieżka składa się z jednej tranzycji, której stan docelowy jest nieistotny. Przez $q_1 \xrightarrow{inv} q_m$ oznaczamy, że istnieje niewidoczna ścieżka ze stanu q_1 do stanu q_m .

Własność 5.16 Niech $q, q' \in Q_i^\varphi$. Jeżeli $q \xrightarrow{inv} q'$, to dla każdego stanu q'' należącego do $(\Pi(q, q')) \setminus \{q'\}$ każda maksymalna ścieżka z q'' przechodzi przez q' . Ponadto, jeżeli dla stanu $q \in Q_i^\varphi$ istnieje stan $q' \in Q_i^\varphi$ taki, że $q \xrightarrow{inv} q'$, to jest on dokładnie jeden.

Dowód:

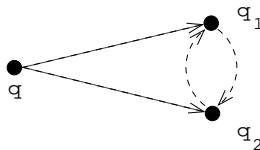
W algorytmie 2 (wiersz 6) do zbioru stanów istotnych dodajemy wszystkie następniki stanów, od których zależą zgodnie z warunkiem **CD2** inne stany istotne. Zauważmy, że skoro ze stanu q istnieje niewidoczna ścieżka do stanu q' dla $q, q' \in Q_i^\varphi$, to do tej ścieżki należy co najmniej jeden stan nieistotny. Zatem co najmniej jeden następnik q jest nieistotny, z czego wynika, że nie ma zależności typu **CD2** między żadnym stanem istotnym, a stanem q . W szczególności nieprawda, że $q' \xrightarrow{cd2} q$. Wobec tego co najmniej jeden z warunków definicji zależności **CD2** nie jest spełniony dla stanów q' i q , a więc zachodzi: (1) każda maksymalna ścieżka z q przechodzi przez q' lub (2) nie istnieje ścieżka π z q do q' taka, że każda maksymalna ścieżka z każdego stanu należącego do ścieżki π przechodzi przez stan q' .



Przypuśćmy, że zachodzi warunek (2). Zatem na każdej ścieżce z q do q' istnieje stan, z którego istnieje maksymalna ścieżka nie przechodząca przez q' . Na mocy założenia istnieje niewidoczna ścieżka π z q do q' . Wobec tego istnieje stan q'' należący do ścieżki π (skoro q'' należy do niewidocznej ścieżki, to nie jest istotny), z którego istnieje maksymalna ścieżka nie przechodząca przez q' . Jeżeli na ścieżce π jest więcej stanów spełniających powyższy warunek, to niech q'' będzie stanem “najbliższym” q' , czyli takim, z którego istnieje ścieżka π' do stanu q' taka, że każda maksymalna ścieżka z każdego stanu należącego do ścieżki π' przechodzi przez stan q' . Zauważmy jednak, że na mocy warunku **CD2** stan q' zależy od stanu q'' , co jest sprzeczne z tym, że q'' jest nieistotny.

Skoro warunek (2) nie jest spełniony, to zachodzi warunek (1), czyli każda maksymalna ścieżka z q przechodzi przez q' . Niech q'' będzie dowolnym stanem należącym do dowolnej ścieżki z q do q' . Z tego, że q'' jest nieistotny otrzymujemy, że nieprawda, że $q' \xrightarrow{cd2} q''$, z czego jak właśnie pokazaliśmy wynika, że każda maksymalna ścieżka z q'' przechodzi przez q' .

Przypuśćmy teraz, że dla stanu $q \in Q_i^\varphi$ istnieją dwa stany $q_1, q_2 \in Q_i^\varphi$ takie, że $q_1 \neq q_2$, $q \xrightarrow{inv} q_1$ i $q \xrightarrow{inv} q_2$.



Jeżeli tak, to zgodnie z powyższym każda maksymalna ścieżka z q przechodzi przez q_1 i jednocześnie każda maksymalna ścieżka z q przechodzi przez q_2 . Po za tym, skoro $q_1, q_2 \in Q_i^{\varphi}$, to q_1 nie należy do niewidocznej ścieżki z q do q_2 i symetrycznie, q_2 nie należy do niewidocznej ścieżki z q do q_1 . Wobec tego istnieje cykl zawierający stany q_1 i q_2 , do którego nie należy stan q . Obydwa stany q_1 i q_2 są stanami wejściowymi tego cyklu. Na mocy własności 3.3 otrzymujemy sprzeczność z tym, że program jest strukturalny (def. 3.2). \square

Opóźnienie dla zbioru ścieżek

Opóźnienie dla zbioru ścieżek ze stanu q do stanu q' definiujemy następująco:

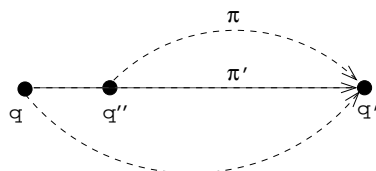
$delay(\Pi(q, q')) = \langle d_1, d_2 \rangle$, gdzie:

- $d_1 = \min_{\pi \in \Pi(q, q')} (lower_bound(\pi))$,
- $d_2 = \max_{\pi \in \Pi(q, q')} (upper_bound(\pi))$,
- $\langle = \left\{ \begin{array}{l} \text{"(" jeżeli dla wszystkich } \pi \in \Pi(q, q') \text{ takich, że } lower_bound(\pi) = d_1, \\ \text{lower_bound_sign}(\pi) = \text{"("}, \\ \text{"[" w przeciwnym przypadku.} \end{array} \right.$
- $\rangle = \left\{ \begin{array}{l} \text{")" jeżeli dla wszystkich } \pi \in \Pi(q, q') \text{ takich, że } upper_bound(\pi) = d_2, \\ \text{upper_bound_sign}(\pi) = \text{")"}, \\ \text{"]" w przeciwnym przypadku.} \end{array} \right.$

Własność 5.17 Niech $q, q' \in Q_i^{\varphi}$. Jeżeli $q \xrightarrow{inv} q'$, to $delay(\Pi(q, q')) = delay(\pi)$ dla dowolnej ścieżki $\pi \in \Pi(q, q')$.

Dowód:

W algorytmie 2 (wiersz 6) do zbioru stanów istotnych dodajemy również wszystkie następniki stanów, od których zależą zgodnie z zależnością czasową inne stany istotne. Skoro ze stanu q istnieje niewidoczna ścieżka do stanu q' dla $q, q' \in Q_i^{\varphi}$, to do tej ścieżki należy co najmniej jeden stan nieistotny. Zatem co najmniej jeden następnik q jest nieistotny, z czego wynika, że nie ma zależności czasowej między żadnym stanem istotnym, a stanem q . W szczególności nieprawda, że $q' \xrightarrow{td2} q$. Wobec tego co najmniej jeden z warunków definicji zależności **TD2** nie jest spełniony dla stanów q' i q , a więc: (1) opóźnienia na wszystkich ścieżkach ze stanu q do q' są równe lub (2) nie istnieje ścieżka z q do q' taka, że dla każdego stanu $q'' \neq q$ należącego do tej ścieżki zachodzi $\forall_{\pi, \pi' \in \Pi(q'', q')} delay(\pi) = delay(\pi')$.



Przypuśćmy, że zachodzi warunek (2). Zatem na każdej ścieżce z q do q' istnieje stan q'' taki, że istnieją dwie ścieżki π i π' z q'' do q' takie, że $delay(\pi) \neq delay(\pi')$. Na mocy

założenia istnieje niewidoczna ścieżka z q do q' . Wobec tego istnieje stan q'' należący do tej ścieżki spełniający powyższy warunek. Jeżeli na niewidocznej ścieżce z q do q' jest więcej stanów spełniających ten warunek, to niech q'' będzie stanem “najbliższym” q' . Zauważmy jednak, że na mocy warunku **TD2** stan q' zależy od stanu q'' , co jest sprzeczne z tym, że q'' jest nieistotny.

Skoro warunek (2) nie jest spełniony, to zachodzi warunek (1), co należało pokazać. \square

Własność 5.18 Jeżeli $q, q' \in Q_i^\varphi$ i $q \xrightarrow{inv} q'$, to opóźnienie $delay(\Pi(q, q'))$ jest znane.

Dowód:

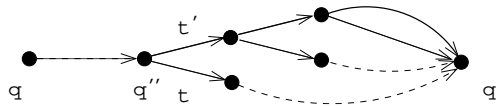
Przypomnijmy, że wartości opóźnienia na ścieżce nie są znane tylko wtedy, gdy do ścieżki należy pilna tranzycja t , dla której nie jest spełniony co najmniej jeden z warunków:

- (1) $(\bigvee_{t' \in out(source(t))} guard(t')) = true$
- (2) $\forall t' \in out(source(t)) (synch(t') = false \wedge (urgent(t') = true \vee delay(t') = [0, 0]))$

Zgodnie z algorytmem 2 (wiersz 4) wszystkie stany, z których wychodzą tranzycje pilne wchodzące do stanów istotnych są dodane do zbioru stanów istotnych. Zatem nie istnieje niewidoczna ścieżka, której ostatnia tranzycja jest pilna.

Pokażemy nie wprost, że dla każdej tranzycji pilnej $t \in trans(\Pi(q, q')) \setminus in(q')$ warunki (1) i (2) są spełnione. Przypuśćmy przeciwnie, że istnieje pilna tranzycja t , której opóźnienie nie jest znane. Powiedzmy, że nie jest spełniony warunek (1). Niech q'' będzie stanem źródłowym tranzycji t . Skoro t należy do niewidocznej ścieżki i nie jest jej ostatnią tranzycją, więc dozór tranzycji t nie jest istotny, a zatem nie ma zależności typu **CD1** żadnego stanu istotnego od stanu q'' , co jest sprzeczne z tym, że nie jest spełniony warunek (1).

Przypuśćmy z kolei, że nie jest spełniony warunek (2). Z konstrukcji zbioru stanów istotnych (wiersz 3) wynika, że żadna z tranzycji należących do $trans(\Pi(q, q')) \setminus in(q')$ nie jest synchroniczna. Skoro warunek (2) nie jest spełniony, to istnieje tranzycja $t' \in out(q'')$, która nie jest pilna i której opóźnienie jest różne od $[0, 0]$.



Następnie, z własności 5.16 otrzymujemy, że każda maksymalna ścieżka z q'' przechodzi przez q' oraz (jeżeli $q'' = q$, to z własności 5.17, a jeżeli $q'' \neq q$, to z warunku **TD2**) że wszystkie ścieżki z q'' do q' mają równe opóźnienia, a ponieważ przyjęliśmy, że tranzycja t ma nieznanne opóźnienie, to w takim razie na wszystkich ścieżkach z $target(t')$ do q' opóźnienie jest nieznanne. Powtarzając całe rozumowanie dla kolejnych tranzycji na ścieżce z q'' do q' otrzymujemy w końcu, że istnieje tranzycja wchodząca do stanu q' , która nie jest pilna i jej opóźnienie jest różne od $[0, 0]$. W takim razie istnieje ścieżka z q'' do q' na której żadna tranzycja nie jest pilna, a więc opóźnienie tej ścieżki jest znane, a więc dochodzimy do sprzeczności. \square

Definicja 5.19 (Zredukowany proces) Dla procesu $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$, gdzie $1 \leq i \leq n$, i zbioru zmiennych zdaniowych PV_φ zredukowany proces $P'_i = (id_i, Q'_i, q_i^{0'}, \Gamma'_i, T'_i)$ budujemy w następujący sposób:

1. $id'_i = id_i$,
2. $Q'_i = Q_i^\varphi$,
3. $q_i^{0'} = q_i^0$,
4. Dla każdej pary stanów $q \in Q_i^\varphi \cap Q_i^R$ i $q' \in Q_i^\varphi$ mamy dwa przypadki:
 - (a) jeżeli $q \xrightarrow{inv} q'$, to dla każdej tranzycji $t \in in(q')$ takiej, że $t \in T_i^\varphi$, a jeżeli nie ma takiej tranzycji, to dla jednej, dowolnie wybranej tranzycji $t \in in(q')$ konstruujemy tranzycję $t' \in T'_i$ w następujący sposób:
 - $source(t') = q$,
 - $target(t') = q'$,
 - $guard(t') = guard(t)|_{A^\varphi}$,
 - $action(t') = action(t)|_{A^\varphi}$,
 - $delay(t') = delay(\Pi(q, q'))$,
 - $urgent(t') = false$,
 - $label(t') = label(t)$.
 - (b) jeżeli q' jest następnikiem stanu q , to dla każdej tranzycji $t \in out(q) \cap in(q')$ konstruujemy tranzycję $t' \in T'_i$ w następujący sposób:
 - $source(t') = q$,
 - $target(t') = q'$,
 - $guard(t') = guard(t)|_{A^\varphi}$,
 - $action(t') = action(t)|_{A^\varphi}$,
 - $delay(t') = delay(t)$,
 - $urgent(t') = urgent(t)$,
 - $label(t') = label(t)$.
5. $\Gamma'_i = \bigcup_{t' \in T'_i} label(t')$.

Powyższa definicja jest poprawna (opóźnienie każdej tranzycji jest dobrze zdefiniowane) na mocy własności 5.18. W skład zredukowanego programu wchodzi te i tylko te zmienne i bufory z oryginalnego programu, które występują w istotnych operacjach.

Definicja 5.20 (Zredukowany program) Dla strukturalnego programu $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, wartościowania początkowego $v^0 : V \cup B \rightarrow \Omega$ i zbioru zmiennych zdaniowych PV_φ konstruujemy zredukowany program $\mathcal{P}' = (V', B', \{P'_i \mid 1 \leq i \leq n\})$ i wartościowanie początkowe $v^{0'} : V' \cup B' \rightarrow \Omega$ w następujący sposób:

1. $V' = V \cap \bigcup_{a \in A^\varphi} vars(a)$,

2. $B' = B \cap \bigcup_{a \in A^\varphi} \text{vars}(a)$,
3. P'_i jest zredukowanym procesem P_i ,
4. $v^{0'} = v^0|_{(V' \cup B')}$.

Przykład 5.21 W zredukowanym programie zmieniającego się bitu (rys. 5.8) zbiory stanów poszczególnych procesów zawierają wyłącznie stany istotne. W procesie *Nadawcy* tranzycję ze stanu $S1$ do stanu $S2$ budujemy zgodnie z konstrukcją opisaną w punkcie (b) def. 5.19 dla tranzycji $l1$. Podobnie dla stanów: $S2$ i $S1$ (tranzycja $l2$), $S2$ i $S3$ (tranzycja $l3$), $S3$ i $S1$ (tranzycja $l4$), $S3$ i $S1$ (tranzycja $l5$). Natomiast ze stanu $S4$ do stanu $S1$ istnieje niewidoczna ścieżka. Dlatego tranzycję t' między tymi stanami budujemy jak opisano w punkcie (a) def. 5.19 dla tranzycji $l8$ (dozór tranzycji t' jest równy *true*, akcja jest pustym ciągiem operacji, atrybut pilności nie jest ustawiony, etykietą jest $l8$, a opóźnienie jest równe opóźnieniu na ścieżkach ze stanu $S4$ do stanu $S1$, czyli $[L, U + D]$). Struktura procesów (czyli stany i tranzycje między nimi) *Odbiorcy* i *Bufora* nie zmienia się. Wszystkie tranzycje w zredukowanym programie zawierają tylko istotne operacje, czyli nie zawierają operacji na zmiennych $sdata$, $rdata$ i $bdata$, ani na buforach sbd i brd . Zbiór zmiennych zredukowanego programu V' jest równy $V \setminus \{sdata, rdata, bdata\}$, a zbiór buforów $B' = B \setminus \{sbd, brd\}$.

5.3 Poprawność

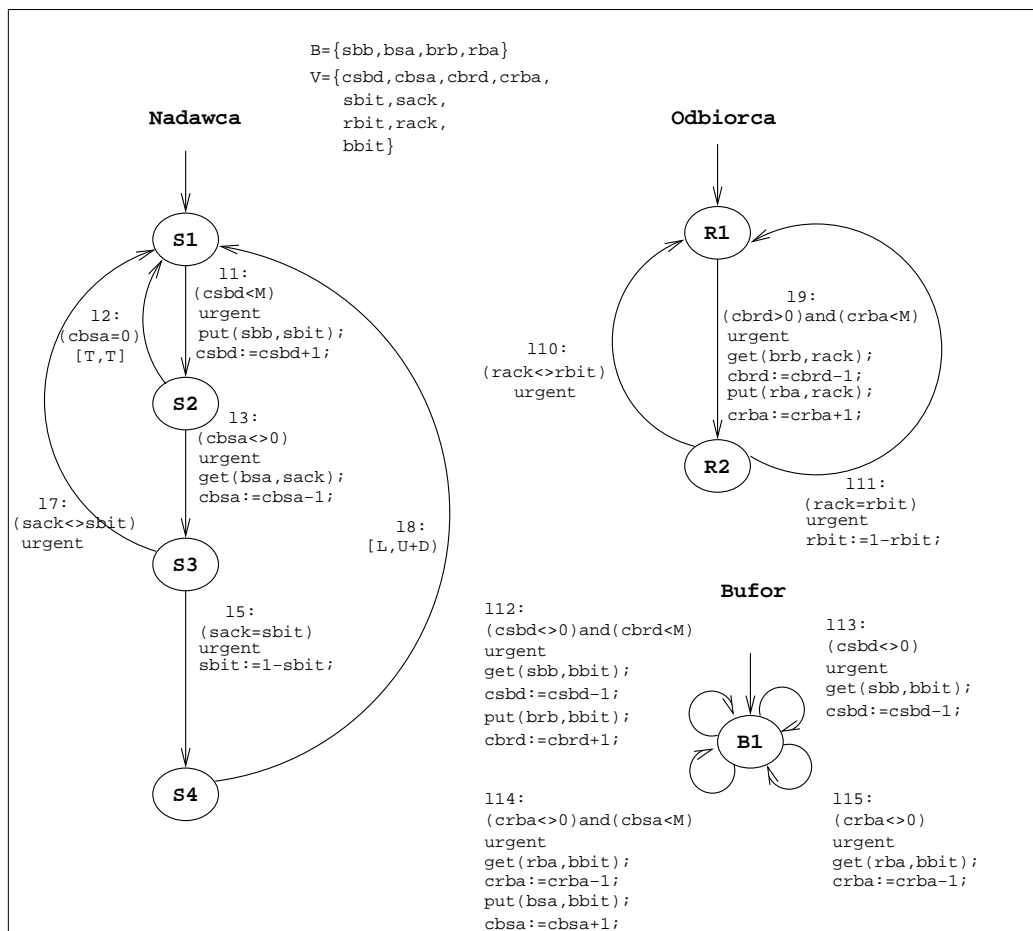
Niech PV_φ będzie zbiorem zmiennych zdaniowych pewnej formuły logicznej φ . Niech $\mathcal{P} = (V, B, \{P_i \mid 1 \leq i \leq n\})$, gdzie $P_i = (id_i, Q_i, q_i^0, \Gamma_i, T_i)$, będzie dowolnym strukturalnym programem, a $v^0 : V \cup B \rightarrow \Omega$ — początkowym wartościowaniem zmiennych i buforów. Niech $\mathcal{P}' = (V', B', \{P'_i \mid 1 \leq i \leq n\})$, gdzie $P'_i = (id'_i, Q'_i, q_i^{0'}, \Gamma'_i, T'_i)$, będzie zredukowanym programem a $v^{0'} : V' \cup B' \rightarrow \Omega$ — wartościowaniem początkowym, skonstruowanymi zgodnie z definicją 5.20 dla programu \mathcal{P} i zbioru zmiennych zdaniowych PV_φ . Ponadto, niech $\mathcal{TS} = (S, s^0, \Gamma, \longrightarrow)$ i $\mathcal{TS}' = (S', s^{0'}, \Gamma', \longrightarrow')$ będą etykietowanymi systemami tranzycyjnymi odpowiednio dla programu \mathcal{P} i wartościowania początkowego v^0 oraz dla programu \mathcal{P}' i wartościowania początkowego $v^{0'}$ oraz niech \mathcal{V} i \mathcal{V}' będą funkcjami wartościującymi takimi, że $\mathcal{V} : \mathcal{TS} \rightarrow 2^{PV_\varphi}$ i $\mathcal{V}' : \mathcal{TS}' \rightarrow 2^{PV_\varphi}$.

Przedstawimy teraz główne twierdzenie rozdziału mówiące o tym, że dana własność wyrażona jako formuła logiki CTL^*_X jest prawdziwa dla programu \mathcal{P} wtedy i tylko wtedy, gdy jest prawdziwa dla zredukowanego programu \mathcal{P}' .

Twierdzenie 5.22 *Niech φ będzie formułą CTL^*_X nad zbiorem zmiennych zdaniowych PV_φ . Niech $\mathcal{M} = (\mathcal{TS}, \mathcal{V})$ i $\mathcal{M}' = (\mathcal{TS}', \mathcal{V}')$ będą dwoma modelami. Wtedy zachodzi:*

$$\mathcal{M}, s^0 \models \varphi \Leftrightarrow \mathcal{M}', s^{0'} \models \varphi.$$

Aby udowodnić powyższe twierdzenie zdefiniujemy relację \cong_{pp} między stanami systemów \mathcal{TS} i \mathcal{TS}' oraz wprowadzimy pomocnicze własności i lematy.



Rysunek 5.8: Zredukowany protokół ABP

Definicja 5.23 (Relacja \cong_{pp}) Niech $\cong_{pp} \subseteq S \times S'$ będzie relacją taką, że dla pary stanów $s = (q_1, \dots, q_n, v, \mu) \in S$ i $s' = (q'_1, \dots, q'_n, v', \mu') \in S'$, zachodzi $s \cong_{pp} s'$ wtedy i tylko wtedy, gdy są spełnione następujące warunki:

1. dla każdego $1 \leq i \leq n$, jeżeli $q_i \in Q'_i$, to $q'_i = q_i$, a w przeciwnym przypadku $q'_i \xrightarrow{inv} q_i$,
2. $v' = v|_{V' \cup B'}$,
3. $\mu'(q'_i) = \mu(q'_i)$ dla każdego $1 \leq i \leq n$.

Stany $s \in TS$ i $s' \in TS'$ są ze sobą w relacji, jeżeli są spełnione odpowiednie warunki dotyczące stanów kontrolnych wszystkich procesów, wartości zmiennych i buforów, a także wartości opóźnień w stanach kontrolnych. Po pierwsze, dla każdego $1 \leq i \leq n$, jeżeli stan kontrolny q_i w stanie s jest istotny, to musi zachodzić $q_i = q'_i$. Jeżeli stan q_i nie jest istotny, to ze stanu q'_i musi istnieć niewidoczna ścieżka do stanu q_i . Następnie, wszystkie istotne zmienne i bufor programu \mathcal{P} w stanie s mają takie same wartości jak w stanie s' . Trzeci warunek dotyczy opóźnienia w stanach kontrolnych: opóźnienie $\mu(q'_i)$ w stanie s jest równe opóźnieniu $\mu'(q'_i)$ w stanie s' .

Własność 5.24 Niech $t \in T$ będzie dowolną tranzycją programu \mathcal{P} , $v_1, v'_1 \in \Omega^{V \cup B}$, $v_2, v'_2 \in \Omega^{V' \cup B'}$, $v'_1 = \mathcal{J}(v_1, \text{action}(t))$ oraz $v'_2 = \mathcal{J}(v_2, \text{action}(t)|_{A^\varphi})$. Jeżeli $v_2 = v_1|_{(V' \cup B')}$, to $v'_2 = v'_1|_{(V' \cup B')}$.

Dowód:

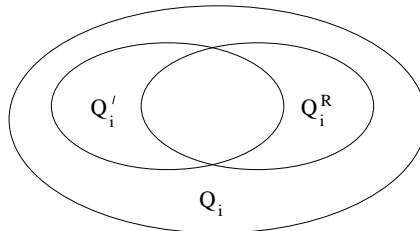
Zgodnie z definicją akcji zredukowanej do istotnych operacji wszystkie operacje z akcji tranzycji t na zmiennych ze zbioru V' i buforach ze zbioru B' wchodzi w skład zredukowanej akcji tranzycji t i ich kolejność jest taka sama w obydwu akcjach. Skoro $v_2 = v_1|_{(V' \cup B')}$, to po wykonaniu takiej samej sekwencji operacji $v'_2 = v'_1|_{(V' \cup B')}$. \square

Własność 5.25 Dla każdej pary stanów $s = (q_1, \dots, q_n, v, \mu) \in S$ i $s' = (q'_1, \dots, q'_n, v', \mu') \in S'$ takich, że $s \cong_{pp} s'$ i dla każdego $1 \leq i \leq n$ zachodzi:

$$q_i \in Q_i \setminus Q_i^R \text{ wtedy i tylko wtedy, gdy } q'_i \in Q'_i \setminus Q_i^R.$$

Dowód:

(\Rightarrow) Przypuśćmy, że $q_i \in Q_i \setminus Q_i^R$ i $q'_i \in Q'_i \cap Q_i^R$ ($q'_i \in Q'_i$ ponieważ w zredukowanym programie wszystkie stany są istotne). Wtedy $q_i \neq q'_i$, a zatem z punktu 1 definicji relacji \cong_{pp} zachodzi $q'_i \xrightarrow{inv} q_i$. Zgodnie z algorytmem obliczania stanów istotnych (algorytm 2, wiersz 8) do zbioru Q'_i należą wszystkie stany z $Q_i \setminus Q_i^R$, które mają poprzedniki w Q_i^R , czyli nie istnieje niewidoczna ścieżka ze stanu $q'_i \in Q'_i \cap Q_i^R$ do $q_i \in Q_i \setminus Q_i^R$, co jest sprzeczne z założeniem.



(\Leftarrow) Przypuśćmy, że $q_i \in Q_i^R$ i $q'_i \in Q'_i \setminus Q_i^R$. Wtedy również $q_i \neq q'_i$, a zatem z punktu 1 definicji relacji \cong_{pp} zachodzi $q'_i \xrightarrow{inv} q_i$. Z def. 5.19 wynika jednak, że w zredukowanym procesie nie ma tranzycji wychodzących ze stanów należących do $Q'_i \setminus Q_i^R$, czyli nie ma ścieżek do żadnych stanów, a w szczególności do q_i , czyli ponownie dochodzimy do sprzeczności. \square

Fakt 5.26 *Jeżeli $s \cong_{pp} s'$, to $\mathcal{V}(s) = \mathcal{V}(s')$.*

Dowód:

Niech $s = (q_1, \dots, q_n, v, \mu) \in S$, $s' = (q'_1, \dots, q'_n, v', \mu') \in S'$ i $s \cong_{pp} s'$. Przypomnijmy, że stany mogą być etykietowane trzema rodzajami zdań: $p_{e_1 \sim e_2}$, $p_{empty(b)}$ i $p_{i,q}$, gdzie $e_1, e_2 \in Expr(V)$, $b \in B$, $q \in Q_i$ dla pewnego $1 \leq i \leq n$. Z definicji kryterium cięcia (def. 5.12) i z punktów 1 i 2 definicji zredukowanego programu (def. 5.20) otrzymujemy $vars(PV_\varphi) \subseteq V' \cup B'$. Następnie, skoro $s \cong_{pp} s'$, więc zachodzi $v' = v|_{(V' \cup B')}$. Wobec powyższych otrzymujemy: $v(e_1 \sim e_2) = v'(e_1 \sim e_2)$ i $v(b) = v'(b)$, a zatem $p_{e_1 \sim e_2} \in \mathcal{V}(s)$ wtedy i tylko wtedy, gdy $p_{e_1 \sim e_2} \in \mathcal{V}(s')$ oraz $p_{empty(b)} \in \mathcal{V}(s)$ wtedy i tylko wtedy, gdy $p_{empty(b)} \in \mathcal{V}(s')$. Musimy jeszcze pokazać, że $p_{i,q} \in \mathcal{V}(s)$ wtedy i tylko wtedy, gdy $p_{i,q} \in \mathcal{V}(s')$, czyli że $q_i = q$ wtedy i tylko wtedy, gdy $q'_i = q$. Zgodnie z definicją kryterium cięcia (def. 5.12) $q \in states(PV_\varphi) \subseteq Q'_i$. Dla każdego stanu q_i , gdzie $1 \leq i \leq n$ rozważmy dwa przypadki: $q_i \in Q'_i$ i $q_i \in Q_i \setminus Q'_i$. Z def. 5.23 dla $q_i \in Q'_i$ mamy $q_i = q'_i$, a zatem $q_i = q$ wtedy i tylko wtedy, gdy $q'_i = q$. Natomiast dla $q_i \in Q_i \setminus Q'_i$ mamy $q'_i \xrightarrow{inv} q_i$. Zauważmy, że gdyby q_i był stanem obserwowanym, to zgodnie z algorytmem 2 (wiersz 2) należałby do Q'_i , z zatem q_i nie jest obserwowany, czyli na pewno $q_i \neq q$. Przypuśćmy, że $q'_i = q$. Zgodnie z algorytmem 2 (wiersz 2) do zbioru Q'_i należą wszystkie następniki stanów obserwowanych, więc nie istnieje niewidoczna ścieżka ze stanu q'_i do stanu q_i , co jest sprzeczne z założeniem. A zatem $q'_i \neq q$. Widzimy więc, że w obydwu przypadkach $q_i = q$ wtedy i tylko wtedy, gdy $q'_i = q$, co kończy dowód. \square

Odpowiadające sobie tranzycje

Powiemy, że tranzycje $t \in T_i$ i $t' \in T'_i$, gdzie $1 \leq i \leq n$, *odpowiadają* sobie, jeżeli $target(t') = target(t)$, $guard(t') = guard(t)|_{A^\varphi}$, $action(t') = action(t)|_{A^\varphi}$ i zachodzi jeden z dwóch przypadków:

1. $source(t') = source(t)$, $delay(t') = delay(t)$ i $urgent(t') = urgent(t)$ lub
2. $source(t') \xrightarrow{inv} source(t)$, $delay(t') = delay(\Pi(source(t'), target(t')))$ i $urgent(t') = false$.

Własność 5.27 *Dla każdego stanu $q \in Q'_i$, gdzie $1 \leq i \leq n$, i dla każdej tranzycji $t \in in(q) \subseteq T_i$ istnieje odpowiadająca jej tranzycja $t' \in in(q) \subseteq T'_i$. I odwrotnie, dla każdego stanu $q' \in Q'_i$ i dla każdej tranzycji $t' \in in(q') \subseteq T'_i$ istnieje odpowiadająca jej tranzycja $t \in in(q') \subseteq T_i$.*

Dowód:

Wynika natychmiast z konstrukcji zredukowanego procesu (def. 5.19). \square

Odpowiadające sobie wykonania

Powiemy, że ścieżka σ w TS odpowiada ścieżce σ' w TS' , jeżeli istnieje podział B_1, B_2, \dots ścieżki σ i podział B'_1, B'_2, \dots ścieżki σ' taki, że dla każdego $j \geq 1$, B_j i B'_j są skończone i niepuste oraz każdy stan z B_j jest w relacji \cong_{pp} z każdym stanem z B'_j .

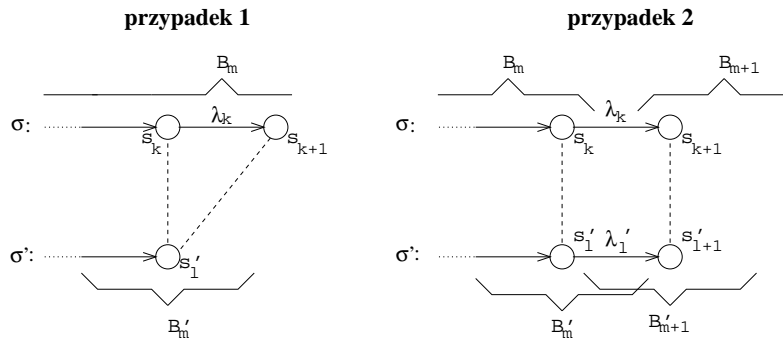
Lemat 5.28 *Jeżeli $s \cong_{pp} s'$, to dla każdej ścieżki σ w TS ze stanu s , istnieje odpowiadająca jej ścieżka σ' w TS' ze stanu s' .*

Dowód:

Niech σ będzie ścieżką w TS ze stanu s i niech $s \cong_{pp} s'^2$. Pokażemy przez konstrukcję, że ze stanu s' istnieje ścieżka σ' w TS' odpowiadająca σ . Konstrukcja jest indukcyjna. Załóżmy, że dla prefiksu $s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_{k-1}} s_k$ ścieżki σ i mamy zbudowany prefiks $s'_1 \xrightarrow{\lambda'_1} s'_2 \xrightarrow{\lambda'_2} \dots \xrightarrow{\lambda'_{l-1}} s'_l$, ścieżki σ' taki, że $s_1 = s$, $s'_1 = s'$ i $s_k \cong_{pp} s'_l$ oraz podziały σ i σ' na odpowiadające sobie bloki. Załóżmy także, że stan s_k należy do bloku B_m , a stan s'_l należy do odpowiadającego mu bloku B'_m . Niech $s_k \xrightarrow{\lambda_k} s_{k+1}$. Mamy dwa przypadki pokazane na rys. 5.9. Linia przerywana łącząca dwa stany oznacza, że są one w relacji \cong_{pp} . Linia ciągła ze strzałką oznacza tranzycję, a linia przerywana ze strzałką oznacza ścieżkę.

Przypadek 1. s_{k+1} należy do tego samego bloku co s_k , czyli do bloku B_m . Pokażemy, że $s_{k+1} \cong_{pp} s'_l$.

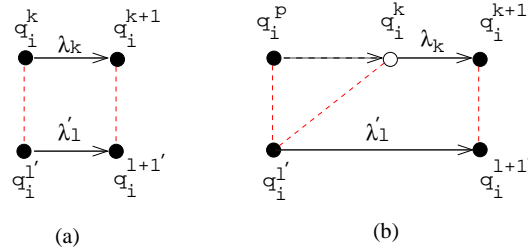
Przypadek 2. s_{k+1} należy do nowego bloku B_{m+1} . W tym przypadku pokażemy, że istnieją: stan s'_{l+1} i tranzycja o etykiecie λ'_l takie, że $s'_l \xrightarrow{\lambda'_l} s'_{l+1}$ i $s_{k+1} \cong_{pp} s'_{l+1}$. Stan s'_{l+1} należy do nowego bloku B'_{m+1} .



Rysunek 5.9: Główne przypadki w dowodzie lematu 5.28

Zasada konstrukcji jest następująca. Stan s_{k+1} należy do tego samego bloku co stan s_k i stan systemu TS' nie zmienia się (**przypadek 1**) wtedy i tylko wtedy, gdy docelowymi

²Przyjmujemy zasadę, że oznaczenia z primem (') odnoszą się do systemu zredukowanego, a bez primu — do oryginalnego.



Rysunek 5.10: Stany kontrolne i -tego procesu w przypadkach 2(a) i 2(b) lematu 5.28

stanami kontrolnymi wykonywanych tranzycji akcyjnych o etykiecie λ_k są stany nieistotne³. Dla wszystkich pozostałych tranzycji akcyjnych i dla wszystkich tranzycji czasowych stan s_{k+1} należy do nowego bloku (**przypadek 2**).

Niech $s_k = (q_1^k, \dots, q_n^k, v^k, \mu^k) \in S$, $s'_l = (q_1^{l'}, \dots, q_n^{l'}, v^{l'}, \mu^{l'}) \in S'$ dla dowolnego $k, l \in \mathbb{N}$. Rozważmy obydwie przypadki.

Przypadek 1. Niech $t_i^k \in T_i$ dla każdego $i \in \Gamma(\lambda_k)$ będzie tranzycją wykonywaną przez i -ty proces i niech $target(t_i^k) \in Q_i \setminus Q'_i$. Pokażemy, że $s_{k+1} \cong_{pp} s'_l$. Z punktu 1 definicji relacji \cong_{pp} (def. 5.23), wiemy że dla $1 \leq i \leq n$ zachodzi $q_i^{l'} = q_i^k$ lub $q_i^{l'} \xrightarrow{inv} q_i^k$, a zatem skoro dla każdego $i \in \Gamma(\lambda_k)$ istnieje tranzycja z q_i^k do q_i^{k+1} i $q_i^{k+1} \in Q_i \setminus Q'_i$, to $q_i^{l'} \xrightarrow{inv} q_i^{k+1}$ (możemy przedłużyć niewidoczną ścieżkę o jedną tranzycję). Punkt 1 definicji relacji \cong_{pp} jest zatem spełniony dla s_{k+1} i s'_l .

Zauważmy, że jeżeli $target(t_i^k) \in Q_i \setminus Q'_i$, to z algorytmu 2 (wiersz 3) wynika, że $opers(t_i^k) \cap A^\varphi = \emptyset$. Zatem, $vars(t_i^k) \cap (V' \cup B') = \emptyset$, czyli akcja tranzycji t_i^k nie zmienia zmiennych z V' ani buforów z B' . Dlatego $v^{k+1}|_{(V' \cup B')} = v^k|_{(V' \cup B')} = v^{l'}$, czyli warunek 2 definicji relacji \cong_{pp} jest spełniony dla stanów s_{k+1} i s'_l .

Aby wykazać, że punkt 3 definicji relacji \cong_{pp} jest spełniony, czyli że $\mu'(q_i^{l'}) = \mu(q_i^{l'})$ dla $1 \leq i \leq n$ wystarczy zauważyć, że $\mu(q)$ dla dowolnego q zmienia się tylko w wyniku wykonania tranzycji czasowej (zwiększa swoją wartość) lub wtedy, gdy proces wchodzi do stanu kontrolnego q (ustawienie na 0). W analizowanej sytuacji przyjeśliśmy, że stanem docelowym tranzycji akcyjnej o etykiecie λ_k jest stan nieistotny. Skoro stan $q_i^{l'}$ jest istotny, to opóźnienie w tym stanie nie ulega zmianie, czyli $\mu^{k+1}(q_i^{l'}) = \mu^k(q_i^{l'})$. Zatem prawdziwość warunku 3 definicji relacji \cong_{pp} dla s_{k+1} i s'_l wynika z prawdziwości tego warunku dla s_k i s'_l .

Przypadek 2. Rozważmy dwa rodzaje tranzycji — akcyjną i czasową.

1. $\lambda_k \in \Gamma$, czyli jeden lub wiele procesów programu \mathcal{P} wykonuje tranzycję akcyjną. Niech $t_i^k \in T_i$ dla każdego $i \in \Gamma(\lambda_k)$ będą wykonywanymi tranzycjami i niech $target(t_i^k) \in Q'_i$. Dla każdego $i \in \Gamma(\lambda_k)$ zachodzi jeden z dwóch przypadków (rys. 5.10, stany zaznaczone kolorem białym są nieistotne):

³Tranzycję, której stan docelowy jest nieistotny, nazwiemy *niewidoczną*.

- (a) Jeżeli $q_i^k \in Q'_i \cap Q_i^R$, to z konstrukcji zredukowanego procesu (def. 5.19) wynika, że istnieje tranzycja $t'_i \in T'_i$ odpowiadająca tranzycji t_i^k taka, że $source(t'_i) = source(t_i^k) = q_i^k$, $delay(t'_i) = delay(t_i^k)$ i $urgent(t'_i) = urgent(t_i^k)$. Pokażemy, że tranzycja t'_i jest gotowa do wykonania przez proces i w stanie s'_i .

Skoro tranzycja t_i^k jest możliwa do wykonania w stanie s_k , więc zachodzi $q_i^k = source(t_i^k)$ i $\mathcal{B}(guard(t_i^k), v^k) = tt$. Ponieważ z założenia indukcyjnego $s_k \cong_{pp} s'_i$ i $q_i^k \in Q'_i$, więc z punktu 1 definicji relacji \cong_{pp} otrzymujemy $q_i^k = q_i^{l'}$, a więc $source(t'_i) = q_i^k = q_i^{l'}$. Poza tym skoro $guard(t'_i) = guard(t_i^k)|_{A^\varphi}$, więc mamy dwa przypadki: albo $guard(t'_i) = guard(t_i^k)$, albo $guard(t'_i) = true$. Z punktu 2 definicji relacji \cong_{pp} wiemy, że $v^{l'} = v^k|_{(V' \cup B')}$. A zatem ponieważ $vars(guard(t'_i)) \subseteq (V' \cup B')$, to $\mathcal{B}(guard(t'_i), v^{l'}) = tt$, a więc tranzycja t'_i jest możliwa do wykonania w stanie s'_i . Z punktu 3 definicji relacji \cong_{pp} wynika, że $\mu^k(q_i^{l'}) = \mu^{l'}(q_i^{l'})$. Zachodzi więc $\mu^{l'}(q_i^{l'}) = \mu^k(q_i^{l'}) = \mu^k(q_i^k) \in delay(t_i^k) = delay(t'_i)$, czyli tranzycja t'_i jest gotowa do wykonania w stanie s'_i .

- (b) Jeżeli $q_i^k \in Q_i^R \setminus Q'_i$, to z założenia indukcyjnego otrzymujemy $q_i^{l'} \xrightarrow{inv} q_i^k$. Z konstrukcji zredukowanego procesu (def. 5.19) wynika, że istnieje tranzycja $t'_i \in T'_i$ odpowiadająca t_i^k taka, że $source(t'_i) = q_i^{l'}$ i $delay(t'_i) = delay(\Pi(q_i^{l'}, q_i^{k+1}))$ i $urgent(t'_i) = false$. Dowód tego, że tranzycja t'_i jest możliwa do wykonania przez proces i w stanie s'_i jest taki sam jak w poprzednim punkcie.

Musimy jeszcze pokazać, że tranzycja t'_i jest gotowa do wykonania w stanie s'_i , czyli $\mu^{l'} \in delay(t'_i) = delay(\Pi(q_i^{l'}, q_i^{k+1}))$.

Zauważmy, że opóźnienie $\mu^k(q_i^{l'})$ jest zerowane kiedy i -ty proces wchodzi do stanu kontrolnego $q_i^{l'}$, dlatego odzwierciedla czas jaki minął od momentu kiedy proces wszedł do stanu $q_i^{l'}$ (czyli łączny czas przebywania w stanach kontrolnych należących do ścieżki ze stanu $q_i^{l'}$ do stanu q_i^{k+1} z wyłączeniem tego ostatniego).

Jeżeli $lower_bound_sign(\Pi(q_i^{l'}, q_i^{k+1})) = \text{"}"$, to

$$lower_bound(t'_i) = lower_bound(\Pi(q_i^{l'}, q_i^{k+1})) \leq \mu^k(q_i^{l'}),$$

a jeżeli $lower_bound_sign(\Pi(q_i^{l'}, q_i^{k+1})) = \text{"}"$, to

$$lower_bound(t'_i) = lower_bound(\Pi(q_i^{l'}, q_i^{k+1})) < \mu^k(q_i^{l'}).$$

Opóźnienie $\mu^k(q_i^{l'})$ jest nie dłuższe⁴ niż $upper_bound(\Pi(q_i^{l'}, q_i^{k+1})) = upper_bound(t'_i)$. Z założenia indukcyjnego $\mu^{l'}(q_i^{l'}) = \mu^k(q_i^{l'})$. A zatem $lower_bound(t'_i) \leq \mu^k(q_i^{l'}) = \mu^{l'}(q_i^{l'})$ i $\mu^{l'}(q_i^{l'}) = \mu^k(q_i^{l'}) \leq upper_bound(t'_i)$, czyli $\mu^{l'} \in delay(t'_i)$.

Zauważmy, że w rozważanym przypadku stan q_i^k nie może należeć do $Q'_i \setminus Q_i^R$, ponieważ zgodnie z algorytmem obliczania stanów istotnych stany docelowe tranzycji wychodzących ze stanów należących do $Q'_i \setminus Q_i^R$ nie są istotne.

Pokażemy teraz, że $s_{k+1} \cong_{pp} s'_{i+1}$.

W obydwu przypadkach wymienionych wyżej po wykonaniu tranzycji t'_i zachodzi $q_i^{l'+1'} = q_i^{k+1}$, a zatem dla każdego $i \in \Gamma(\lambda_k)$ jest spełniony warunek 1 definicji relacji

⁴W dalszej części dowodu dla uproszczenia będziemy rozważać wyłącznie nierówność nieostrą. W przypadku nierówności ostrej dowód przebiega w analogiczny sposób.

\cong_{pp} . Zauważmy także, że $\mu^{l+1'}(q_i^{l+1'}) = 0$ i $\mu^{k+1}(q_i^{l+1'}) = \mu^{k+1}(q_i^{k+1}) = 0$, a więc warunek 3 definicji relacji \cong_{pp} jest spełniony dla każdego $i \in \Gamma(\lambda_k)$.

Stany kontrolne i opóźnienia w stanach kontrolnych procesów, które nie wykonują tranzycji (o numerach nie należących do $\Gamma(\lambda_k)$) nie zmieniają się, a zatem prawdziwość warunków 1 i 3 definicji relacji \cong_{pp} dla tych procesów dla s_{k+1} i s'_{l+1} wynika wprost z tego, że te warunki są spełnione dla s_k i s'_l .

Musimy jeszcze pokazać, że spełniony jest warunek 2 definicji relacji \cong_{pp} . Zgodnie z definicją zredukowanego procesu (def. 5.19) akcja tranzycji t'_i składa się z akcji tranzycji t_i^k zredukowanej do istotnych operacji. Poza tym z założenia indukcyjnego $v^{l'} = v^k|_{(V' \cup B')}$. W trakcie wykonywania przejścia λ_k wykonywane są akcje wszystkich tranzycji t_i^k i t'_i dla każdego $i \in \Gamma(\lambda_k)$, zgodnie z kolejnością indeksów. Zatem korzystając z własności 5.24 dla kolejnych par t_i^k i t'_i otrzymujemy $v^{l+1'} = v^{k+1}|_{(V' \cup B')}$, czyli punkt 2 definicji relacji \cong_{pp} jest również spełniony dla s_{k+1} i s'_{l+1} .

Do tej pory pokazaliśmy, że dla każdej tranzycji akcyjnej t_i^k w systemie \mathcal{TS} istnieje gotowa do wykonania tranzycja t'_i i -tego procesu w systemie \mathcal{TS}' oraz $s'_l \xrightarrow{\lambda'_i} s'_{l+1}$, gdzie λ'_i jest etykietą tranzycji t'_i i $s_{k+1} \cong_{pp} s'_{l+1}$.

2. $\lambda_k \in \mathbb{R}_+$, co oznacza, że tranzycja reprezentuje upływ czasu. W tym przypadku $q_i^{k+1} = q_i^k$ dla $1 \leq i \leq n$, $v^{k+1} = v^k$ i $\mu^{k+1} = \mu^k + \lambda_k$. Pokażemy, że ze stanu s'_l również można wykonać tranzycję czasową o długości λ_k . Oczywiście po wykonaniu takiej tranzycji $q_i^{l+1'} = q_i^{l'}$ dla każdego $1 \leq i \leq n$ oraz $v^{l+1'} = v^{l'}$, czyli prawdziwość warunków 1 i 2 definicji relacji \cong_{pp} dla s_{k+1} i s'_{l+1} wynika z prawdziwości tych warunków dla s_k i s'_l . Ponadto, $\mu^{l+1'} = \mu^{l'} + \lambda_k = \mu^k + \lambda_k = \mu^{k+1}$, więc spełniony jest również warunek 3 definicji relacji \cong_{pp} . A zatem $s_{k+1} \cong_{pp} s'_{l+1}$, gdzie $s'_l \xrightarrow{\lambda_k} s'_{l+1}$. Musimy tylko pokazać, że upływ czasu o długości λ_k jest możliwy w stanie s'_l . Przypadek $\lambda_k = 0$ jest oczywisty. Rozważmy $\lambda_k > 0$. Przypominamy, że tranzycja czasowa $\lambda_k > 0$ jest możliwa do wykonania w stanie s'_l , jeżeli dla każdej tranzycji $t'_i \in T'_i$ zachodzi:

$$\neg enabled(t'_i, s'_l) \vee (\neg urgent(t'_i) \wedge \neg expired(t'_i, s'_{l+1})) \quad (5.1)$$

Niech $t'_i \in T'_i$ będzie dowolną tranzycją zredukowanego programu. Jeżeli tranzycja t'_i nie jest możliwa do wykonania w stanie s'_l , to warunek (5.1) jest spełniony.

Niech t'_i będzie tranzycją możliwą do wykonania w stanie s'_l , z czego wynika, że $source(t'_i) = q_i^{l'}$ i $\mathcal{B}(guard(t'_i), v'_i) = tt$. Pokażemy, że tranzycja t'_i nie jest pilna i nie jest przeterminowana w stanie s'_{l+1} . Będziemy korzystać z tego, że ze stanu s_k można wykonać tranzycję czasową o długości λ_k . Dla każdego $1 \leq i \leq n$ możliwe są dwa przypadki:

- (a) $q_i^k \in Q_i^R \setminus Q'_i$, czyli z założenia indukcyjnego $q_i^{l'} \xrightarrow{inv} q_i^k$. Z konstrukcji zredukowanego procesu otrzymujemy, że istnieje niewidoczna ścieżka ze stanu $q_i^{l'}$ = $source(t'_i)$ do stanu $target(t'_i)$ w procesie P_i . Z własności 5.16 wynika, że q_i^k należy do niewidocznej ścieżki z $q_i^{l'}$ do $target(t'_i)$.

Zauważmy, że $\mu^k(q_i^{l'})$ odzwierciedla czas, w jakim proces przebył ścieżkę ze stanu $q_i^{l'}$ do q_i^k (oznaczmy ją π), który jest nie dłuższy niż $upper_bound(\pi)$. Ponadto skoro można wykonać tranzycję czasową λ_k ze stanu s_k , to dla każdej tranzycji t wychodzącej z q_i^k , $\lambda_k \leq upper_bound(t)$. Otrzymujemy $\mu^k(q_i^{l'}) + \lambda_k \leq upper_bound(\pi) + upper_bound(t)$, co jest z kolei nie większe niż górne ograniczenie na czas potrzebny na przebycie całej ścieżki z $q_i^{l'}$ do $target(t'_i)$, czyli $upper_bound(\Pi(q_i^{l'}, target(t'_i)))$, z własności 5.17 równe $upper_bound(t'_i)$. Z założenia indukcyjnego $\mu^k(q_i^{l'}) = \mu^{l'}(q_i^{l'})$. A zatem $\mu^{l'}(q_i^{l'}) + \lambda_k \leq upper_bound(t'_i)$, czyli $\neg expired(t'_i, s'_{l+1})$.

Pokażemy jeszcze, że tranzycja t'_i nie jest pilna. Z konstrukcji tranzycji t'_i (def. 5.19) widzimy, że istnieje tranzycja $t_i \in T_i$ taka, że $target(t'_i) = target(t_i) \in Q'_i$. W przypadku (a) def. 5.19 $urgent(t'_i) = false$. Natomiast w przypadku (b) z algorytmu 2 (wiersz 4) wynika, że stany z których wychodzi tranzycja pilna wchodząca do $target(t'_i) \in Q'_i$ są istotne, czyli nie istnieje niewidoczna ścieżka której ostatnia tranzycja jest pilna (a taką mamy sytuację, bo założyliśmy, że $q_i^k \in Q_i^R \setminus Q'_i$).

Zatem warunek (5.1) jest spełniony dla tego przypadku.

- (b) $q_i^k \in Q'_i \cap Q_i^R$. Z własności 5.27 dla każdego $1 \leq i \leq n$ istnieje tranzycja $t_i \in T_i$ odpowiadająca t'_i .

Jeżeli zachodzi przypadek (2) z definicji odpowiadających sobie tranzycji, to możemy powtórzyć argumentację z poprzedniego punktu (proces P_i znajduje się w pierwszym stanie niewidocznej ścieżki). Dla przypadku (1) z definicji odpowiadających sobie tranzycji możliwe są dwie sytuacje: $guard(t'_i) = true$ lub $guard(t'_i) = guard(t_i)$.

W pierwszym przypadku zauważmy, że z algorytmu 2 (wiersz 4) wynika, że od stanu $q_i^{l'}$ nie zależą stany istotne (bo wtedy dozory tranzycji byłyby istotne), a w szczególności stan $target(t'_i)$. A zatem z warunku **TD2** wynika, że opóźnienie na wszystkich ścieżkach z $source(t'_i)$ do $target(t'_i)$ jest równe i wynosi $delay(\Pi(source(t'_i), target(t'_i)))$. Analogicznie jak w poprzednim punkcie możemy pokazać, że tranzycja t'_i nie jest przeterminowana w stanie s'_{l+1} . Tranzycja t'_i nie jest pilna, bo wtedy byłaby też pilna odpowiadająca jej tranzycja t_i , a gdyby tak było, to w stanie s_k nie byłby możliwy upływ czasu.

Natomiast w drugim przypadku zauważmy, że jeżeli $guard(t'_i) = guard(t_i)$, to w analogiczny sposób jak w punkcie 1(a) możemy pokazać, że jeżeli t'_i jest możliwa do wykonania w s'_l , to t_i jest możliwa do wykonania w s_k oraz jeżeli t'_i nie jest przeterminowana w stanie s'_{l+1} , to t_i nie jest przeterminowana w stanie s_{k+1} . A zatem z tego, że w stanie s_k jest możliwa tranzycja czasowa otrzymujemy, że tranzycja t_i nie jest pilna, a więc t'_i również nie jest pilna oraz, że t_i nie jest przeterminowana w stanie s_{k+1} , a więc tranzycja t'_i również nie jest przeterminowana w stanie s'_{l+1} , czyli warunek (5.1) jest spełniony, co należało pokazać.

Przypadek $q_i^k \in Q_i \setminus Q_i^R$ nie jest możliwy. Przypuśćmy przeciwnie. Wtedy z własności 5.25 otrzymujemy, że $q_i^{l'} \in Q'_i \setminus Q_i^R$. Z konstrukcji programu wynika, że $q_i^{l'}$ jest stanem końcowym w procesie P'_i (z którego nie ma wychodzących tranzycji), co jest sprzeczne z tym, że tranzycja t'_i jest możliwa do wykonania w stanie s'_l .

Pokazaliśmy, że dla każdej tranzycji $t'_i \in T'_i$ jest możliwy upływ czasu o długości λ_k .

Każdy z bloków B_1, B_2, \dots i B'_1, B'_2, \dots jest skończony, ponieważ liczba niewidocznych tranzycji akcyjnych jest skończona i nie tworzą one cyklu, a każda inna tranzycja akcyjna i każda tranzycja czasowa powoduje przejście do nowego bloku. A zatem warunki lematu 5.28 są spełnione, co należało wykazać. \square

Lemat 5.29 *Jeżeli $s \cong_{pp} s'$, to dla każdej ścieżki σ' w TS' ze stanu s' , istnieje odpowiadająca jej ścieżka σ w TS ze stanu s .*

Dowód:

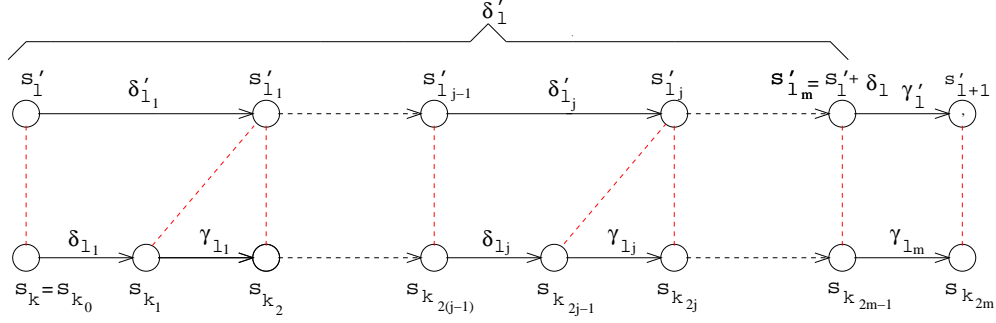
Niech σ' będzie ścieżką w TS' ze stanu s' . Pokażemy przez konstrukcję, że istnieje ścieżka σ w TS ze stanu s , podział B_1, B_2, \dots ścieżki σ i podział B'_1, B'_2, \dots ścieżki σ' spełniające warunki lematu. Podobnie jak poprzednio konstrukcja jest indukcyjna. Załóżmy, że dla prefiksu $s'_1 \xrightarrow{\lambda'_1} s'_2 \xrightarrow{\lambda'_2} \dots \xrightarrow{\lambda'_{i-1}} s'_i$, ścieżki σ' i mamy zbudowany prefiks $s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_{k-1}} s_k$ ścieżki σ taki, że $s_1 = s$, $s'_1 = s'$ i $s_k \cong_{pp} s'_i$ oraz podziały ścieżek σ' i σ na odpowiadające sobie bloki. Załóżmy także, że stan s'_i należy do bloku B'_m , a stan s_k należy do odpowiadającego mu bloku B_m . Rozważmy dwa przypadki.

Przypadek 1. Jeżeli do ścieżki σ'_i należy tranzycja akcyjna, czyli istnieje $j \geq 0$ takie, że $\lambda'_{i+j} \in \Gamma$, to niech $s'_i \xrightarrow{\delta'_i} s'_i + \delta'_i \xrightarrow{\gamma'_i} s'_{i+1}$, gdzie $\delta'_i \in \mathbb{R}_+$ i $\gamma'_i = \lambda'_{i+j}$. Możemy tak przedstawić następane dwie tranzycje na ścieżce σ' ze stanu s'_i , bo jeżeli ze stanu s'_i wychodzi tranzycja akcyjna, to przyjmujemy, że $\delta'_i = 0$. Natomiast, jeżeli na tej ścieżce jest więcej tranzycji czasowych przed pierwszą tranzycją akcyjną, to możemy je połączyć w jedną tranzycję czasową.

Przyjmijmy, że $s'_i = (q_1^{l'}, \dots, q_n^{l'}, v^{l'}, \mu^{l'})$. Niech $i \in \Gamma(\gamma'_i)$ będzie numerem procesu wykonującego tranzycję akcyjną o etykiecie γ'_i , którą oznaczymy przez $t_i^{l'}$. Zauważmy, że $q_i^{l'} \in Q_i^R$, ponieważ zgodnie z konstrukcją zredukowanego procesu (def. 5.19) stany należące do zbioru $Q'_i \setminus Q_i^R$ są stanami końcowymi. Zatem z własności 5.25 otrzymujemy $q_i^k \in Q_i^R$. Jeżeli $q_i^k \in Q'_i \cap Q_i^R$, to z założenia indukcyjnego zachodzi $q_i^k = q_i^{l'}$, w przeciwnym przypadku $q_i^{l'} \xrightarrow{inv} q_i^k$.

Wykonaniu ciągu złożonego z dwóch tranzycji o etykietach δ'_i i γ'_i w systemie TS' odpowiada wykonanie ciągu tranzycji $\delta_{l_1}, \gamma_{l_1}, \dots, \delta_{l_m}, \gamma_{l_m}$ w systemie TS dla pewnego $m \geq 1$ takiego, że ostatnia tranzycja z tego ciągu odpowiada tranzycji $t_i^{l'}$ (rys. 5.11). Intuicyjnie, jeżeli $q_i^k = q_i^{l'}$ dla każdego $i \in \Gamma(\gamma'_i)$, czyli każdy proces wykonujący tranzycję znajduje się w stanie istotnym, to po upływie δ'_i jednostek czasu proces i -ty wykonuje tranzycję odpowiadającą tranzycji $t_i^{l'}$. A jeżeli istnieje $i \in \Gamma(\gamma'_i)$ takie, że $q_i^{l'} \xrightarrow{inv} q_i^k$, to proces i -ty wykonuje ciąg złożony z tranzycji należących do niewidocznej ścieżki ze stanu q_i^k do stanu $q_i^{l'}$ oraz z tranzycji odpowiadającej tranzycji $t_i^{l'}$.

Konstrukcję pokażemy najpierw dla jednego procesu o numerze $i \in \Gamma(\gamma'_i)$. Z def. 5.19 widzimy, że ponieważ $target(t_i^{l'}) \in Q'_i$, to dla $t_i^{l'}$ istnieje ścieżka (składająca się być może z pojedynczej tranzycji) $\pi = q_0 t_1 q_1 \dots t_m q_m$, gdzie $q_0 = q_i^k$ i $q_m = q_i^{l'+1}$ i $q_j \in Q_i$, taka, że każda tranzycja t_j dla $1 \leq j \leq m$ jest możliwa do wykonania ze stanu q_{j-1} . Wybór



Rysunek 5.11: Konstrukcja ciągu tranzycji $\delta_{l_1}, \gamma_{l_1}, \dots, \delta_{l_m}, \gamma_{l_m}$

takiej ścieżki jest możliwy, ponieważ stan q_{j-1} dla $1 \leq j \leq m$ jest nieistotny, więc warunek **CD1** nie jest dla niego spełniony i dlatego istnieje co najmniej jedna tranzycja możliwa do wykonania w tym stanie. Ponadto z tego, że warunek **TD1** nie jest spełniony dla nieistotnych stanów wynika, że albo wszystkie tranzycje wychodzące z ze stanu q_{j-1} mają równe opóźnienia, albo istnieje tranzycja, która jest zawsze możliwa do wykonania. Z własności 5.16 wiemy także, że każda ścieżka z q_j przechodzi przez $q_i^{l+1'}$.

Skoro ze stanu s'_l jest możliwa tranzycja czasowa, to $\mu^{l'}(q_i^{l'}) + \delta'_l \in \text{delay}(t_i^{l'})$, a więc

$$\text{lower_bound}(\text{delay}(t_i^{l'})) \leq \mu^{l'}(q_i^{l'}) + \delta'_l \leq \text{upper_bound}(\text{delay}(t_i^{l'}))$$

Z konstrukcji dozwolonego opóźnienia $\text{delay}(t_i^{l'})$ jest równe dozwolonemu opóźnieniu dla zbioru ścieżek prowadzących ze stanu $q_i^{l'}$ do stanu $q_i^{l+1'}$ w P_i . Z własności 5.17 wiemy, że opóźnienia na wszystkich ścieżkach niewidocznych są równe, więc dozwolone opóźnienie na ścieżce ze stanu $q_i^{l'}$ do stanu $q_i^{l+1'}$ (oznaczmy ją przez $\pi' = q_{-p-1}t_{-p} \dots t_0q_0t_1q_1 \dots t_mq_m$, gdzie $q_{-p-1} = q_i^{l'}$, $q_m = q_i^{l+1'}$ i $q_0 = q_i^k$), której sufiksem jest π , jest również równe $\text{delay}(t_i^{l'})$, wobec czego

$$\text{lower_bound}(\text{delay}(\pi')) \leq \mu^{l'}(q_i^{l'}) + \delta'_l \leq \text{upper_bound}(\text{delay}(\pi'))$$

Korzystając z tego, że $\text{lower_bound}(\text{delay}(\pi')) = \sum_{j=-p}^m \text{lower_bound}(\text{delay}(t_j))$ oraz z tego, że $\text{upper_bound}(\text{delay}(\pi')) = \sum_{j=-p}^m \text{upper_bound}(\text{delay}(t_j))$ otrzymujemy

$$\sum_{j=-p}^m \text{lower_bound}(\text{delay}(t_j)) \leq \mu^{l'}(q_i^{l'}) + \delta'_l \leq \sum_{j=-p}^m \text{upper_bound}(\text{delay}(t_j))$$

Zauważmy, że $\mu^{l'}(q_i^{l'})$ odzwierciedla czas jaki minął od momentu, kiedy proces P_i' wszedł do stanu $q_i^{l'}$. Z założenia indukcyjnego otrzymujemy $\mu^{l'}(q_i^{l'}) = \mu^k(q_i^{l'})$, gdzie $\mu^k(q_i^{l'})$ reprezentuje czas jaki minął od momentu, kiedy proces P_i wszedł do stanu $q_i^{l'}$, który jest równy czasowi potrzebnemu na przejście ścieżki $q_{-p-1}t_{-p} \dots t_0q_0$, czyli sumie czasów przebywania procesu P_i w stanach q_{-p-1}, \dots, q_0 . Wobec tego istnieje ciąg wartości

$\mu_{-p-1}, \dots, \mu_0, \delta'_{l_1}, \dots, \delta'_{l_m}$ taki, że $\mu_{-p-1} + \dots + \mu_0 + \delta'_{l_1} + \dots + \delta'_{l_m} = \mu'(q'_i) + \delta'_l$ oraz

$$\begin{aligned} \text{lower_bound}(\text{delay}(t_j)) &\leq \mu_{j-1} \leq \text{upper_bound}(\text{delay}(t_j)) \text{ dla } j = -p, \dots, 0 \\ \text{lower_bound}(\text{delay}(t_1)) &\leq \mu_0 + \delta'_{l_1} \leq \text{upper_bound}(\text{delay}(t_1)) \\ \text{lower_bound}(\text{delay}(t_j)) &\leq \delta'_{l_j} \leq \text{upper_bound}(\text{delay}(t_j)) \text{ dla } j = 2, \dots, m \end{aligned}$$

μ_j jest czasem przebywania w stanie q_j dla $j = -p - 1, \dots, -1$, $\mu_0 + \delta'_{l_1}$ jest czasem przebywania w stanie q_0 , a δ'_{l_j} — w stanie q_j dla $j = 1, \dots, m - 1$.

Tranzycję czasową $s'_l \xrightarrow{\delta'_l} s'_{l+1}$ możemy podzielić na ciąg tranzycji $s'_l \xrightarrow{\delta'_{l_1}} s'_{l_1} \xrightarrow{\delta'_{l_2}} \dots \xrightarrow{\delta'_{l_m}} s'_{l_m}$, gdzie $s'_{l_m} = s'_{l+1}$, $\delta'_{l_j} \in \mathbb{R}_+$ dla $j = 1, \dots, m$ i $\sum_{j=1}^m \delta'_{l_j} = \delta'_l$. Sufiks ścieżki σ ze stanu s_k konstruujemy w następujący sposób: $s_k \xrightarrow{\delta_{k_1}} s_{k_1} \xrightarrow{\gamma_{k_1}} \dots \xrightarrow{\delta_{k_m}} s_{k_{2m-1}} \xrightarrow{\gamma_{k_m}} s_{k_{2m}}$, gdzie $\delta_{k_j} = \delta'_{l_j}$ i $\gamma_{k_j} = \text{label}(t_j)$ dla $j = 1, \dots, m$. Dla każdej tranzycji t_j , gdzie $j = 1, \dots, m$, musimy pokazać, że można ją wykonać ze stanu $s_{k_{2j-1}}$. Dla $j = 1, \dots, m - 1$ każda tranzycja t_j jest możliwa do wykonania, bo w ten sposób wybraliśmy ścieżkę π . Pokażemy teraz, że istnieje tranzycja t_m odpowiadająca t'_i taka, że t_m jest możliwa do wykonania w stanie $s_{k_{2m-1}}$. Rozważmy tranzycje wchodzące do stanu $\text{target}(t'_i)$. Skoro tranzycja t'_i jest możliwa do wykonania w stanie s'_{l+1} , więc zachodzi $\mathcal{B}(\text{guard}(t'_i), v') = tt$. Poza tym skoro $\text{guard}(t'_i) = \text{guard}(t_m)|_{A^\varnothing}$, więc mamy dwa przypadki: albo $\text{guard}(t'_i) = \text{guard}(t_m)$, albo $\text{guard}(t'_i) = \text{true}$ (czyli dozór tranzycji t_m nie jest istotny). W pierwszym przypadku korzystając z tego, że $s_{k_{2m-1}} \cong_{pp} s'_{l+1}$ (co pokazujemy niżej) otrzymujemy $v' = v^{k_{2m-1}}|_{(V' \cup B')}$. A zatem ponieważ $\text{vars}(\text{guard}(t'_i)) \subseteq (V' \cup B')$, to $\mathcal{B}(\text{guard}(t_m), v^{k_{2m-1}}) = tt$. Natomiast w drugim przypadku zauważmy, że żaden ze stanów należących do Q'_i nie jest zależny od stanu q_{m-1} , żadna z operacji należących do akcji tranzycji wychodzących z q_{m-1} nie jest istotna, ani żadna tranzycja wychodząca z q_{m-1} nie jest synchroniczna (w przeciwnym przypadku zgodnie z algorytmem obliczania operacji istotnych dozór tranzycji t_m byłby istotny). Korzystając ponownie z warunku **CD1** otrzymujemy, że zawsze co najmniej jedna tranzycja jest możliwa do wykonania ze stanu q_{m-1} . Z powyższego wynika również, że odpowiada ona tranzycji t'_i .

Ponieważ proces P_i przebywa w stanie kontrolnym $q_{j-1} = \text{source}(t_j)$ dokładnie $\delta'_{l_j} = \delta_{k_j}$ jednostek czasu ($\mu^{k_{2j-1}}(\text{source}(t_j)) = \delta_{k_j}$) i $\text{lower_bound}(\text{delay}(t_j)) \leq \delta_{k_j} \leq \text{upper_bound}(\text{delay}(t_j))$, więc $\mu^{k_{2j-1}}(\text{source}(t_j)) \in \text{delay}(t_j)$, czyli t_j jest gotowa do wykonania w $s_{k_{2j-1}}$ dla $j = 1, \dots, m$.

Należy jeszcze pokazać, że $s_{k_{2j-1}} \cong_{pp} s'_{l_j}$ i $s_{k_{2j}} \cong_{pp} s'_{l_j}$ dla każdego $1 \leq j \leq m$. Dowód jest indukcyjny. Zakładając, że $s_{k_{2(j-1)}} \cong_{pp} s'_{l_{j-1}}$ pokażemy, że $s_{k_{2j-1}} \cong_{pp} s'_{l_j}$ i zakładając, że $s_{k_{2j-1}} \cong_{pp} s'_{l_j}$ pokażemy, że $s_{k_{2j}} \cong_{pp} s'_{l_j}$.

Dowód $s_{k_{2j-1}} \cong_{pp} s'_{l_j}$ dla $1 \leq j < m$ przebiega w ten sam sposób co dowód $s_{k+1} \cong_{pp} s'_{l+1}$ w **przypadku 2.2** dowodu lematu 5.28 (wykonanie tranzycji czasowej nie zmienia stanów kontrolnych, ani wartości zmiennych ani zawartości buforów, a opóźnienie zmienia się w ten sam sposób dla wszystkich stanów). Dowód $s_{k_{2j}} \cong_{pp} s'_{l_j}$ dla $1 \leq j < m$ przebiega w ten sam sposób co dowód $s_{k+1} \cong_{pp} s'_l$ w **przypadku 1** dowodu lematu 5.28 (wykonanie nieistotnej tranzycji, nie zmienia istotnych stanów kontrolnych ani wartości istotnych zmiennych ani buforów). Natomiast dowód $s_{k_{2m}} \cong_{pp} s'_{l+1}$ jest analogiczny do **przypadku 2.1** dowodu

lematu 5.28 (ostatnia tranzycja na ścieżce albo jest nieistotna, albo zmienia stan systemu w ten sam sposób, co tranzycja $t_i^{l'}$).

Jeżeli po przedstawionym podziale istnieje proces l -ty, taki że $q_i^k \in Q_l \setminus Q_l'$ dla którego upływ czasu o długości δ_j dla pewnego j nie jest możliwy, to dokonujemy podziału tranzycji czasowej δ_j na krótsze odcinki i wybieramy tranzycje akcyjne procesu l w opisany powyżej sposób. Postępujemy tak, dopóki istnieją procesy dla których upływ czasu o danej długości nie jest możliwy.

Przypadek 2. Jeżeli do ścieżki σ_l' nie należy tranzycja akcyjna, czyli nie istnieje $j \geq 0$ takie, że $\lambda_{l+j}' \in \Gamma$, to niech $s_l' \xrightarrow{\delta_l'} s_l' + \delta_l'$, gdzie $\delta_l' \in \mathbb{R}_+$, będzie następną tranzycją na ścieżce σ' . Przypominamy, że tranzycja czasowa $\delta_k = \delta_l'$ jest możliwa do wykonania w stanie s_k , jeżeli każda tranzycja $t_i \in T_i$ albo nie jest możliwa do wykonania, albo nie jest pilna i nie zostanie przeterminowana po upływie δ_l' jednostek czasu. Jeżeli istnieje tranzycja t_i , dla której powyższy warunek nie jest spełniony, to tranzycję czasową o etykiecie δ_l' możemy podzielić na krótsze tranzycje czasowe i skonstruować ścieżkę σ_k w sposób analogiczny do przypadku 1 niniejszego dowodu (bez ostatniej tranzycji akcyjnej).

Wszystkie skonstruowane bloki B_1, B_2, \dots i B_1', B_2', \dots są niepuste i skończone, ponieważ w każdym bloku istnieje skończona liczba tranzycji akcyjnych, a tranzycje czasowe przeprowadzają do nowego bloku. Zatem warunki lematu 5.29 są spełnione, co należało wykazać. \square

Zauważmy, że jeżeli ścieżka $\sigma = s_1 \xrightarrow{\lambda_1} s_2 \xrightarrow{\lambda_2} \dots$, gdzie $\lambda_j \in \mathbb{R}_+ \cup \Gamma$ dla $j \geq 1$, jest progresywnym przebiegiem, to istnieje odpowiadająca jej ścieżka $\sigma' = s_1' \xrightarrow{\lambda_1'} s_2' \xrightarrow{\lambda_2'} \dots$, gdzie $\lambda_j' \in \mathbb{R}_+ \cup \Gamma'$ dla $j \geq 1$, która jest również progresywnym przebiegiem i odwrotnie. Wynika to z tego, że $\sum_{\{j \in \mathbb{N} \mid \lambda_j \in \mathbb{R}_+\}} \lambda_j = \sum_{\{j \in \mathbb{N} \mid \lambda_j' \in \mathbb{R}_+\}} \lambda_j'$. Ponadto, jeżeli ścieżka σ jest silnie (słabo) uczciwym przebiegiem w stosunku do zbioru tranzycji widocznych (czyli takich, których stany docelowe są istotne), to istnieje odpowiadająca jej ścieżka σ' , która jest również silnie (słabo) uczciwym przebiegiem w stosunku do zbioru tranzycji widocznych, ponieważ zgodnie z konstrukcją opisaną w dowodzie lematu 5.28 ilekroć tranzycja widoczna występuje w σ , tyle razy występuje również w σ' . Stwierdzenie odwrotne jest również prawdziwe, co można wykazać w analogiczny sposób korzystając z lematu 5.29.

Fakt 5.30 $s^0 \cong_{pp} s^{0'}$.

Dowód:

Należy pokazać, że dla $s^0 = (q_1^0, \dots, q_n^0, v^0, \mu^0) \in S$ i $s^{0'} = (q_1^{0'}, \dots, q_n^{0'}, v^{0'}, \mu^{0'}) \in S'$ zachodzi $s^0 \cong_{pp} s^{0'}$. Zauważmy, że zgodnie z definicją zredukowanego procesu (def. 5.19) zachodzi $q_i^{0'} = q_i^0$ dla $1 \leq i \leq n$, czyli warunek 1 definicji relacji \cong_{pp} jest spełniony. Następnie, z punktu 4 definicji 5.20 otrzymujemy, że $v^{0'} = v^0|_{(V' \cup B')}$, czyli warunek 2 definicji relacji \cong_{pp} jest również spełniony. Wreszcie z założenia, że początkowe wartości opóźnień wszystkich tranzycji są równe 0 wynika, że $\mu^0(q) = 0$ dla każdego stanu $q \in Q_i$ i $\mu^{0'}(q_i') = 0$ dla każdego stanu $q_i' \in Q_i'$, gdzie $1 \leq i \leq n$. Zatem warunek 3 definicji relacji \cong_{pp} jest także spełniony. \square

Fakt 5.31 *Relacja $\cong_{pp} \subseteq S \times S'$ jest bisymulacją z powtórzeniami pomiędzy strukturą $\mathcal{M} = (\mathcal{TS}, \mathcal{V})$ a $\mathcal{M}' = (\mathcal{TS}', \mathcal{V}')$.*

Dowód:

Warunki definicji bisymulacji z powtórzeniami 2.8 wynikają wprost z faktu 5.30 i lematów 5.28 i 5.29. \square

W dowodzie twierdzenia 5.22 nie możemy skorzystać z powyższego faktu i twierdzenia 2.9, ponieważ jest ono prawdziwe dla skończonych struktur \mathcal{M} i \mathcal{M}' , a w naszym przypadku są one nieskończone. Dlatego przeprowadzimy klasyczny dowód indukcyjny po długości formuły.

Lemat 5.32 *Niech φ będzie dowolną formułą logiki CTL_{-X}^* , stanową lub ścieżkową. Niech σ będzie ścieżką ze stanu s w \mathcal{M} i niech σ' będzie odpowiadającą jej ścieżką ze stanu s' w \mathcal{M}' . Zachodzą wtedy następujące warunki:*

1. $s \models \varphi$ wtedy i tylko wtedy, gdy $s' \models \varphi$, gdzie φ jest formułą stanową i
2. $\sigma \models \varphi$ wtedy i tylko wtedy, gdy $\sigma' \models \varphi$, gdzie φ jest formułą ścieżkową.

Dowód:

Dowód przeprowadzimy przez indukcję po długości formuły φ .

Podstawa. $\varphi = p$, gdzie $p \in PV$. Z def. 2.5 $s \models p$ wtedy i tylko wtedy, gdy $p \in \mathcal{V}(s)$. Z faktu 5.26 $\mathcal{V}(s) = \mathcal{V}'(s')$, a więc $p \in \mathcal{V}(s) \Leftrightarrow p \in \mathcal{V}'(s')$. Korzystając ponownie z def. 2.5 otrzymujemy $s \models p \Leftrightarrow s' \models p$.

Krok indukcyjny. Rozważmy następujące przypadki:

1. $\varphi = \neg\psi$, gdzie φ i ψ są formułami stanowymi. Z def. 2.5 dla operatora negacji, $s \models \varphi$ wtedy i tylko wtedy, gdy nieprawda, że $s \models \psi$. Z założenia indukcyjnego wiemy, że $s \models \psi \Leftrightarrow s' \models \psi$. Zatem nieprawda, że $s \models \psi$ jest równoważne nieprawda, że $s' \models \psi$, a więc $s \models \neg\psi \Leftrightarrow s' \models \neg\psi$.

Takie samo rozumowanie możemy przeprowadzić, gdy φ jest formułą ścieżkową.

2. $\varphi = \psi_1 \wedge \psi_2$, gdzie φ , ψ_1 i ψ_2 są formułami stanowymi. Z def. 2.5 dla operatora koniunkcji, $s \models \varphi$ wtedy i tylko wtedy, gdy $s \models \psi_1$ i $s \models \psi_2$. Z założenia indukcyjnego $s \models \psi_1 \Leftrightarrow s' \models \psi_1$ i $s \models \psi_2 \Leftrightarrow s' \models \psi_2$. Zatem $s \models \psi_1 \wedge \psi_2 \Leftrightarrow s' \models \psi_1 \wedge s' \models \psi_2$, a więc $s \models \psi_1 \wedge \psi_2 \Leftrightarrow s' \models \psi_1 \wedge \psi_2$.

Takie samo rozumowanie możemy przeprowadzić, gdy φ jest formułą ścieżkową.

3. Dla $\varphi = A\psi$, gdzie φ jest formułą stanową, a ψ jest formułą ścieżkową, dowód przeprowadzimy nie wprost.

Przypuśćmy, że $s \models \varphi$ i nieprawda, że $s' \models \varphi$. Zatem z def. 2.5 dla operatora A otrzymujemy, że istnieje ścieżka σ' ze stanu s' taka, że nieprawda, że $\sigma' \models \psi$. Z lematu 5.29 istnieje ścieżka σ ze stanu s odpowiadająca ścieżce σ' . Z założenia

indukcyjnego otrzymujemy nieprawdą, że $\sigma \models \psi$, czyli ze stanu s istnieje ścieżka σ taka, że nieprawda, że $\sigma \models \psi$, a więc nieprawda, że $s \models A\psi$. Sprzeczność.

Przypuśćmy z kolei, że $s' \models \varphi$ i nieprawda, że $s \models \varphi$. Zatem istnieje ścieżka σ ze stanu s taka, że nieprawda, że $\sigma \models \psi$. Z lematu 5.28 istnieje ścieżka σ' ze stanu s' odpowiadająca ścieżce σ . Z założenia indukcyjnego otrzymujemy nieprawdą, że $\sigma' \models \psi$, czyli ze stanu s' istnieje ścieżka σ' taka, że nieprawda, że $\sigma' \models \psi$, a więc nieprawda, że $s' \models A\psi$. Sprzeczność.

Pokazaliśmy, że $s \models A\psi \Leftrightarrow s' \models A\psi$.

4. Niech $\varphi = \psi$, gdzie φ jest formułą ścieżkową, a ψ jest formułą stanową. W takim przypadku, z def. 2.5 $\sigma \models \varphi \Leftrightarrow s \models \psi$. Z założenia indukcyjnego $s \models \psi \Leftrightarrow s' \models \psi$, a więc $\sigma \models \varphi \Leftrightarrow \sigma' \models \varphi$.
5. $\varphi = U(\psi_1, \psi_2)$ jest formułą ścieżkową. Przypuśćmy, że $\sigma \models U(\psi_1, \psi_2)$. Z definicji operatora U istnieje $k \geq 0$ takie, że $\sigma_k \models \psi_2$ i dla każdego $0 \leq i < k$ zachodzi $\sigma_i \models \psi_1$. Skoro ścieżki σ i σ' odpowiadają sobie, to zgodnie z lematem 5.28 istnieje podział ścieżki σ na bloki B_1, B_2, \dots taki, że dla każdego $j \geq 1$, B_j i B'_j są niepuste i skończone oraz każdy stan z B_j jest w relacji \cong_{pp} z każdym stanem z B'_j . Niech σ_k będzie ścieżką ze stanu należącego do bloku B_m . Wobec tego istnieje ścieżka σ'_l wychodząca z pierwszego stanu należącego do bloku B'_m taka, że σ_k i σ'_l odpowiadają sobie. Korzystając z założenia indukcyjnego $\sigma'_l \models \psi_2$.

Przypuśćmy, że istnieje $0 \leq j < l$ takie, że nieprawda, że $\sigma'_j \models \psi_1$. Niech s'_j będzie pierwszym stanem ścieżki σ'_j i niech B'_n będzie blokiem do którego należy stan s'_j ($n < m$). Z lematu 5.28 do bloku B_n należy co najmniej jeden stan, który jest w relacji \cong_{pp} ze stanem s'_j . A zatem istnieje $0 \leq i < k$ takie, że nieprawda, że $\sigma_i \models \psi_1$, co jest sprzeczne z założeniem. A zatem dla każdego $0 \leq j < l$ zachodzi $\sigma'_j \models \psi_1$. Co kończy dowód tego, że $\sigma' \models \varphi$.

Dowód w drugą stronę możemy przeprowadzić w analogiczny sposób korzystając z lematu 5.29. □

Dowód twierdzenia 5.22:

Wynika z faktu 5.30 i z lematu 5.32. □

5.4 Wyniki eksperymentalne

W tej części rozdziału chcieliśmy pokazać jakie znaczenie praktyczne ma metoda cięcia. Prototyp implementacji metody dla podklasy języka bazowego (bez komunikacji synchronicznej) został wykonany przez Pawła Janowskiego. Implementacja dla obecnej, pełnej wersji języka jest przedmiotem pracy magisterskiej Jerzego Łosia. Przedstawiamy wybrane wyniki eksperymentów sprawdzających efektywność metody. Testy przeprowadzono na systemie Linux Fedora, na maszynie wyposażonej w procesor Intel Pentium 4–3 GHz i 2 GB pamięci RAM.

Oczywiście, nie dla wszystkich programów po zastosowaniu metody redukcji ich rozmiar się zmniejsza. Z przykładów prezentowanych w rozprawie (rodz. 3.4) widoczną redukcję

udało się uzyskać dla protokołu zmieniającego się bitu. W obydwu wersjach protokołu Fischera (punkt 3.4.2) wszystkie elementy systemu są istotne dla prawdziwości przedstawionych własności. W przykładzie producenta i konsumenta (punkt 3.4.1) dla przedstawionego zbioru zmiennych zdaniowych nie jest istotny bufor b , ani zmienne p i c . Redukcja ta nie ma jednak wpływu na rozmiar automatu globalnego. Natomiast zbiór automatów dla zredukowanego systemu składa się tylko z dwóch automatów (dla producenta i konsumenta, nie ma automatu dla bufora). Uważamy, że metoda cięcia będzie dawać najbardziej widoczne rezultaty w przypadku złożonych systemów, w których występują duże ilości danych niekoniecznie istotnych dla każdej z badanych własności.

Protokół zmieniającego się bitu

Rozważmy po raz ostatni protokół zmieniającego się bitu. Jak poprzednio, dla protokołu badamy własność zdefiniowaną w punkcie 3.4.3 wyrażoną przez:

$$\varphi = AG((s_1 \wedge r_1 \wedge sbit0) \Rightarrow rbit0)$$

a także symetryczną formułę

$$\theta = AG((s_1 \wedge r_1 \wedge rbit0) \Rightarrow sbit0)$$

dla kilku wartości parametrów M (rozmiar bufora) i N (liczba różnych danych). Zauważmy, że problem sprawdzenia prawdziwości formuły φ sprowadza się do zbadania osiągalności stanu, w którym zachodzi $s_1 \wedge r_1 \wedge sbit0 \wedge \neg rbit0$. Jeżeli taki stan nie jest osiągalny, to formuła φ jest prawdziwa. Podobnie, żeby sprawdzić prawdziwość formuły θ , badamy osiągalność stanu, w którym zachodzi $s_1 \wedge r_1 \wedge \neg sbit0 \wedge rbit0$.

Na początku, dla każdej pary parametrów z programu w języku bazowym opisującego protokół został wygenerowany globalny automat czasowy i zbiór automatów lokalnych. Tę samą czynność powtórzono dla zredukowanego protokołu. Redukcja dla obydwu formuł jest taka sama, ponieważ obydwie są zbudowane nad tym samym zbiorem zmiennych zdaniowych.

Chcieliśmy zbadać wpływ redukcji dla różnych podejść do weryfikacji modelowej automatów czasowych. Wybraliśmy dwa zupełnie różne weryfikatory: Kronos [DOTY95] oparty o tradycyjną metodę przeszukiwania przestrzeni stanów oraz VerICS [DJJ⁺03a] wykorzystujący metodę kodowania rozwinięcia modelu do pewnej głębokości w postaci formuły logiki zdaniowej (realizowane przez moduł o nazwie BMC [WZP03]), której spełnialność jest następnie badana przez odpowiednie narzędzia (wybraliśmy MiniSat [Min06]).

W tabeli 5.1 przedstawiono wyniki weryfikacji protokołu ABP za pomocą Kronosa. Weryfikator przyjmuje na wejściu globalny automat czasowy. Rozmiary automatów globalnych (liczba lokacji i tranzycji) znajdują w czwartej i piątej kolumnie tabeli. Testy oznaczone przez **O** przeprowadzono dla automatu wygenerowanego dla pierwotnej wersji protokołu (rys. 3.4). Testy oznaczone jako **A** wykonano dla tego samego automatu przy zastosowaniu techniki redukcji zwanej metodą aktywnych zegarów [DY96], która jest zaimplementowana w Kronosie. Wreszcie symbolem **R** oznaczone są testy przeprowadzone dla automatu wygenerowanego dla protokołu zredukowanego metodą cięcia (rys. 5.8).

formuła	parametry	test	lokacje	tranzycje	głębokość	czas(s)
φ	$M = 1, N = 2$	O	1287	3791	6	0.193
		A	1287	3791	6	0.195
		R	1128	3303	6	0.159
φ	$M = 2, N = 2$	O	9081	35257	6	1.609
		A	9081	35257	6	1.633
		R	8142	31457	6	1.414
φ	$M = 3, N = 2$	O	47926	210228	6	9.054
		A	47926	210228	6	9.198
		R	33804	147920	6	6.641
θ	$M = 1, N = 2$	O	1287	3791	15	0.199
		A	1287	3791	15	0.198
		R	1128	3303	14	0.137
θ	$M = 2, N = 2$	O	9081	35257	15	1.619
		A	9081	35257	15	1.624
		R	8142	31457	14	1.461
θ	$M = 3, N = 2$	O	47926	210228	15	9.186
		A	47926	210228	15	9.200
		R	33804	147920	14	6.496

Tablica 5.1: Wyniki eksperymentalne weryfikacji ABP (Kronos) dla $D = L = U = 1$

Obydwie weryfikowane formuły okazały się nieprawdziwe. Kontrprzykład zbudowany przez weryfikator dla formuły φ (taki sam dla wszystkich przypadków) jest następujący:

$$s_0 \xrightarrow{l1} s_1 \xrightarrow{l12} s_2 \xrightarrow{l9} s_3 \xrightarrow{l11} s_4 \xrightarrow{l15} s_5 \xrightarrow{10} s_6 \xrightarrow{l2} s_7 \dots$$

Jeżeli dane i komunikat dotrą do *Odbiorcy*, a potwierdzenie odbioru zostanie zgubione, to wartość bitu *Nadawcy* nadal będzie równa 0, podczas gdy bit *Odbiorcy* zostanie zmieniony na 1. W wyniku weryfikacji formuły θ otrzymano następujące kontrprzykłady — dla programu oryginalnego:

$$s_0 \xrightarrow{l1} s_1 \xrightarrow{l12} s_2 \xrightarrow{l9} s_3 \xrightarrow{l11} s_4 \xrightarrow{l14} s_5 \xrightarrow{10} s_6 \xrightarrow{l3} s_7 \xrightarrow{l5} s_8 \xrightarrow{l7} s_9 \xrightarrow{1} s_{10} \xrightarrow{l8} s_{11} \xrightarrow{l1} s_{12} \xrightarrow{l12} s_{13} \xrightarrow{l9} s_{14} \xrightarrow{l1} s_{15} \xrightarrow{10} s_{16} \xrightarrow{l2} s_{17} \dots$$

a dla programu zredukowanego:

$$s_0 \xrightarrow{l1} s_1 \xrightarrow{l12} s_2 \xrightarrow{l9} s_3 \xrightarrow{l11} s_4 \xrightarrow{l14} s_5 \xrightarrow{10} s_6 \xrightarrow{l3} s_7 \xrightarrow{l5} s_8 \xrightarrow{1} s_9 \xrightarrow{l8} s_{10} \xrightarrow{l1} s_{11} \xrightarrow{l12} s_{12} \xrightarrow{l9} s_{13} \xrightarrow{l1} s_{14} \xrightarrow{10} s_{15} \xrightarrow{l2} s_{16} \dots$$

Szósta kolumna zawiera liczbę tranzycji akcyjnych w kontrprzykładzie. W ostatniej kolumnie umieszczono czas weryfikacji formuły (w sekundach).

Na podstawie wyników z tabeli 5.1 można zauważyć, że metoda aktywnych zegarów nie daje rezultatów (niewielki wzrost czasu weryfikacji jest spowodowany dodatkową analizą).

formuła	parametry	test	głębokość	bmc			minisat	
				klauzule	czas(s)	MB	czas(s)	MB
φ	$M = 1, N = 1$	O	6	278188	44.1	23.4	0.39	21.0
		R	6	235545	39.8	20.1	0.29	18.9
φ	$M = 1, N = 2$	O	6	285890	45.7	24.0	0.37	21.2
		R	6	235545	39.8	20.1	0.29	18.9
φ	$M = 2, N = 2$	O	6	1371623	662.9	105.6	2.78	84.2
		R	6	1328838	522.3	101.0	3.17	82.2
θ	$M = 1, N = 1$	O	15	1151981	232.5	76.0	4.26	65.3
		R	14	828979	190.7	56.8	2.44	49.9
θ	$M = 1, N = 2$	O	15	1174353	245.2	77.5	2.28	72.5
		R	14	828979	190.7	56.8	2.44	49.9
θ	$M = 2, N = 2$	O	15	4346778	3649.1	278.8	60.8	232.2
		R	14	3783769	2792.4	254.0	237.2	194.0

Tablica 5.2: Wyniki eksperymentalne weryfikacji ABP (VerICS) dla $D = L = U = 1$

Stanowi to potwierdzenie faktu, że liczba zegarów w automatach czasowych generowanych zgodnie z metodą opisaną w rozdz. 4 jest minimalna (nie daje się zredukować). Rozmiary automatów uzyskanych dla zredukowanych programów są mniejsze od rozmiarów automatów dla oryginalnych programów. W przypadku weryfikacji formuły θ ma to wpływ na długość kontrprzykładu (jedna z tranzycji kontrprzykładu dla oryginalnego programu jest nieistotna). Widać również różnicę w czasach weryfikacji (proporcjonalną do różnicy w rozmiarach automatów).

Tabela 5.2 przedstawia zestawienie testów przeprowadzonych za pomocą weryfikatora VerICS. Tym razem dla oryginalnego i zredukowanego protokołu wygenerowano zbiory automatów składowych. W kolejnych kolumnach tabeli przedstawiono liczbę klauzul formuły zdaniowej zbudowanej przez moduł BMC, a także ilość czasu (w sekundach) i pamięci (w megabajtach) potrzebnej do jej zbudowania. Podane wielkości dotyczą tylko rozwinięcia o głębokości⁵ podanej w trzeciej kolumnie (czyli najmniejszej głębokości, na której można znaleźć kontrprzykład). Dwie ostatnie kolumny pokazują ilość czasu i pamięci wykorzystanej do sprawdzenia spełnialności formuły przez MiniSat.

W opracowaniu brak wyników weryfikacji formuł dla $N = 2$ i $M = 3$ ze względu na bardzo długi czas generowania formuł przez moduł (testy przerwano po 2 godzinach). Można zauważyć, że wielkość formuł zdaniowych (wyrażona jako liczba klauzul) jest mniejsza dla automatów powstałych dla zredukowanych programów, a także, że krótszy jest czas i ilość pamięci potrzebnej do ich zbudowania. Różnica jest bardziej wyraźna w testach dla formuły θ ze względu na większą głębokość rozwinięcia modelu. Redukcja nie ma natomiast jednoznacznego wpływu na czas działania programu MiniSat, który dla pewnych przypadków jest krótszy, a dla innych dłuższy w porównaniu z czasem uzyskanym dla oryginalnego programu.

⁵Rozumianej jako liczba tranzycji akcyjnych.

parametry	test	lokacje	tranzycje	głębokość	czas(s)
$N = 2$	O	444	1141	10	0.043
	A	444	1141	10	0.047
	R	326	848	8	0.035
$N = 3$	O	2604	8358	11	0.212
	A	2604	8358	11	0.295
	R	1614	5238	9	0.120

Tablica 5.3: Wyniki eksperymentalne weryfikacji Fabryki (Kronos)

Fabryka

Protokół zmieniającego się bitu jest przykładem systemu asynchronicznego. Chcieliśmy również pokazać przykład weryfikacji programu synchronicznego. Wybraliśmy i opisaliśmy w języku bazowym działanie pewnej Fabryki omówione w pracy [DY95] (rys. 5.12). System składa się z procesów modelujących robota umieszczającego towary na przesuwałce (robot *D*), robota zdejmującego towary z taśmy (robot *G*), modułu przetwarzającego towary (stacja) oraz z N procesów reprezentujących towary. Procesy systemu wykonują wspólnie pewne tranzycje (na przykład podniesienie towaru z taśmy jest wykonywane przez proces modelujący odpowiedniego robota, stację, a także przez proces reprezentujący podnoszony towar). W programie nie ma zmiennych ani buforów.

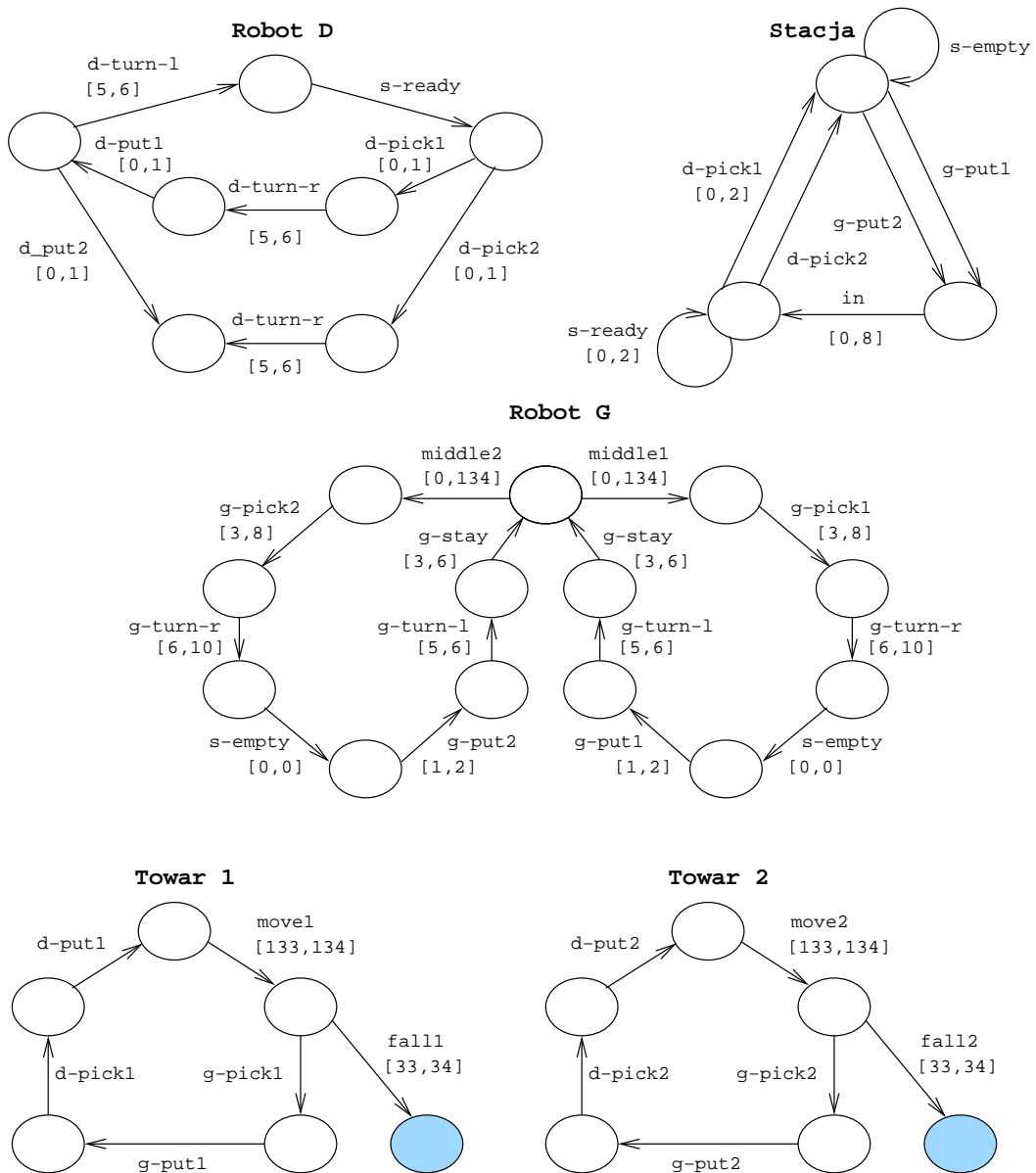
Dla tego przykładu sprawdzamy jedną z własności również przedstawionych w pracy [DY95], a mianowicie, czy możliwa jest sytuacja, w której jeden z towarów spadnie z taśmy (jeden z procesów opisujących towary znajdzie się w stanie oznaczonym szarym kolorem). Własność okazała się prawdziwa. Metoda testowania tego przykładu była taka sama jak poprzednio. Po zastosowaniu metody cięcia okazało się, że w obydwu procesach opisujących roboty po dwa stany są nieistotne dla badanej własności.

Wyniki weryfikacji Fabryki za pomocą Kronosa znajdują się w tabeli 5.3. Testy udało się przeprowadzić tylko dla dwóch wartości parametru N . Podobnie jak w poprzednim przypadku, metoda aktywnych zegarów nie daje żadnej redukcji. Natomiast dzięki metodzie cięcia rozmiar automatu globalnego dla zredukowanego programu jest mniejszy, a co za tym idzie także długość świadka (przykładu świadczącego o tym, że formuła jest prawdziwa) oraz czas potrzebny na jego znalezienie.

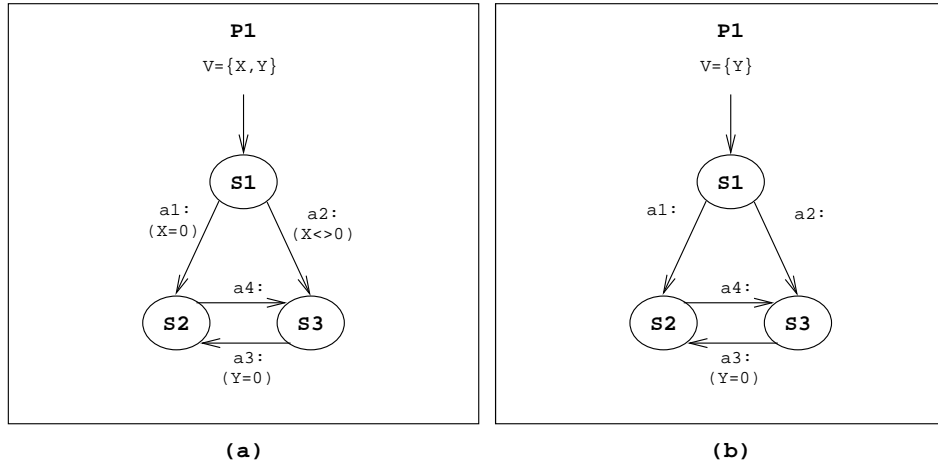
parametry	test	głębokość	klauzule	czas(s)	MB	czas(s)	MB
$N = 2$	O	10	305696	3.91	20.6	3.27	19.13
	R	8	224986	2.76	15.9	1.31	14.29
$N = 3$	O	11	426534	5.83	27.4	13.86	27.77
	R	9	317234	4.10	21.2	3.97	19.66
$N = 4$	O	12	571001	7.88	35.8	221.23	130.51
	R	10	428707	5.33	27.7	64.54	42.95

Tablica 5.4: Wyniki eksperymentalne weryfikacji Fabryki (VerICS)

Następna tabela (5.4) prezentuje wyniki weryfikacji tego przykładu przy użyciu systemu



Rysunek 5.12: Fabryka dla $N = 2$



Rysunek 5.13: Przykład programu niestukturalnego

VerICS. W tym przypadku udało się przeprowadzić testy także dla $N = 4$ (dla większej wartości N nie przeprowadzono testów). Widać, że efektywność metody cięcia (ze względu na wszystkie mierzone wielkości) rośnie wraz ze wzrostem liczby składowych systemu. Tym razem czas sprawdzania spełnialności formuły przez Minisat okazał się wyraźnie krótszy dla automatów uzyskanych dla zredukowanego programu (co jest spowodowane dosyć dużą różnicą długości najkrótszego świadka, a także rozmiarów automatów).

Podsumowując możemy stwierdzić, że metoda cięcia okazuje się dosyć skuteczna. Jej efektywność rośnie wraz ze wzrostem rozmiar weryfikowanych systemów.

5.5 Podsumowanie rozdziału

Konstrukcje niestukturalne

Metoda cięcia nie redukuje poprawnie programów niestukturalnych, ponieważ warunek **CD2** nie jest wystarczający dla programów zawierających cykle z wieloma stanami wejściowymi. Jest możliwa sytuacja, w której mimo, że stan q_1 należy do wszystkich maksymalnych ścieżek ze stanu q_2 i zawsze istnieje tranzycja możliwa do wykonania ze stanu q_2 , to jednak stan q_1 nie zostanie osiągnięty. Przykład takiej sytuacji pokazuje rysunek 5.13. Dla wartościowania początkowego $v^0(X) = v^0(Y) = 1$ w programie (a) stan $S2$ nie jest osiągalny, a w programie (b) jest. Usunięcie dozorów tranzycji wychodzących ze stanu $S1$ ma wpływ na osiągalność stanu $S2$.

Aby skorzystać z metody cięcia dla programu niestukturalnego należy przekształcić go najpierw na równoważny program strukturalny. Metoda takiego przekształcenia podana w [ASU02] stosuje się również do języka bazowego.

Metoda cięcia w literaturze

Metoda cięcia zaproponowana po raz pierwszy przez Weisera [Wei84] pierwotnie była stosowana do uproszczania procesów śledzenia (ang. *debugging*) i symulacji programów. Dobry przegląd wielu technik i odmian metody cięcia stanowi praca [Tip95].

Pierwsza próba wykorzystania tej metody do redukcji przestrzeni stanów w weryfikacji modelowej systemów bez czasu została podjęta w [MT98] dla Promeli, języka wejściowego weryfikatora modelowego SPIN [Hol97]. Autorzy tej pracy wprowadzili intuicję dla tak zwanego *niedokładnego cięcia* (ang. *imprecise slice*), jednak nie przedstawili swojej metody formalnie.

Praca [HDZ99] prezentuje metodę cięcia sekwencyjnych programów w języku Java, która zachowuje prawdziwość formuł logiki temporalnej LTL. Kolejna praca tych autorów [HCD⁺99] uogólnia tę technikę dla wielowątkowych programów w Javie. Moduł implementujący tę metodę jest częścią systemu weryfikacyjnego Bandera [DHS99].

Metoda cięcia jest również wykorzystywana do redukcji języka opisu systemów czasowych IF [BFG⁺99], jednak podobnie jak poprzednio jest zdefiniowana jedynie dla bezczasowego podzbioru tego języka [BFG00].

Rozdział 6

Podsumowanie

Podsumowując, do głównych wyników rozprawy należy opracowanie:

- Języka bazowego do opisu systemów czasowych,
- Metody generowania automatów czasowych dla specyfikacji w języku bazowym,
- Metody redukcji systemów czasowych wykorzystującej metodę cięcia.

Przedstawiona metoda generowania automatów czasowych została wykorzystana między innymi w procesie weryfikacji systemów (także bezczasowych) opisanych w językach Estelle [DJJ02], UML [NPLK06], Promela [NDJ06] i Java [OWS06], a także do weryfikacji protokołów kryptograficznych uwzględniających zależności czasowe [JP06a, JP07], gdzie jako język opisu został wykorzystany język bazowy.

Dalsze plany badawcze autorki rozprawy obejmują opracowanie innych metod statycznej redukcji dla systemów z czasem, takich jak statyczne redukcje częściowo porządkowe.

Spis rysunków

2.1	Produkt automatów czasowych	14
2.2	Protokół Fischera dla dwóch procesów	15
3.1	Producent i konsument	33
3.2	Synchroniczny protokół Fischera	34
3.3	Asynchroniczny protokół Fischera	35
3.4	Protokół zmieniającego się bitu (ABP)	36
4.1	Automat czasowy dla przykładu producenta i konsumenta	43
4.2	Zbiór automatów czasowych dla przykładu producenta i konsumenta	56
4.3	Zredukowany automat czasowy dla przykładu producenta i konsumenta	64
4.4	Zbiór zredukowanych automatów czasowych dla producenta i konsumenta	65
5.1	Przykłady zależności danych w protokole ABP	72
5.2	Zależność przepływu sterowania typu CD2	73
5.3	Zależności przepływu sterowania w protokole ABP	74
5.4	Przykład 1 zależności czasowej	75
5.5	Przykład 2 zależności czasowej	76
5.6	Zależność czasowa typu TD2	79
5.7	Zależności czasowe w protokole ABP	80
5.8	Zredukowany protokół ABP	92
5.9	Główne przypadki w dowodzie lematu 5.28	95
5.10	Stany kontrolne i -tego procesu w przypadkach 2(a) i 2(b) lematu 5.28	96
5.11	Konstrukcja ciągu tranzycji $\delta_{l_1}, \gamma_{l_1}, \dots, \delta_{l_m}, \gamma_{l_m}$	101
5.12	Fabryka dla $N = 2$	110
5.13	Przykład programu niestukturalnego	111

Indeks

- akcja, 21
 - tranzycji, 23
- algorytm obliczania istotnych operacji, 82
- algorytm obliczania istotnych stanów, 84
- atrybut pilności, 22, 23
- atrybut tranzycji, 22
- automat czasowy, 10
 - globalny, 41
 - dla procesu, 52
 - dla zmiennej lub bufora, 53
 - produktowy, 12
- bisymulacja z powtórzeniami, 20
- bufor, 21
 - definiowany przez operację, 25
 - niezależny, 51
 - obserwowany, 82
 - używany w operacji, 25
- CTL*, 16
- CTL*_X, 18
- cykl, 26
- dozór, 22
- etykieta, 22, 23
 - lokalna, 23
 - synchronizująca, 23
 - tranzycji automatu, 11
 - – składowego, 51
- formuła ścieżkowa, 16
- formuła stanowa, 16
- funkcja wartościująca, 9
 - dla automatu czasowego, 13
 - dla programu, 32
 - lokalnie automatu globalnego, 42
 - lokalnie automatu składowego, 55
- instrukcja, 21
- konfiguracja programu, 28
- kryterium cięcia, 82
- kwantyfikatory ścieżkowy, 16
- lokacja, 10
 - źródłowa, 11
 - docelowa, 11
 - początkowa, 11
- model, 9
 - automatu czasowego, 13
 - programu, 32
- multisynchronizacja, 12
- następnik, 9
 - stanu kontrolnego, 25
- niewidoczna ścieżka, 86
- niezmiennik, 11
- odpowiadające sobie tranzycje, 94
- odpowiadające sobie wykonania
 - automatu globalnego i produktowego, 58
 - programów, 95
 - programu i automatu, 44
- opóźnienie, dozwolone opóźnienie,
 - tranzycji, 22
 - – górne, 22
 - – nieznanne, 78
 - dla zbioru ścieżek, 88
 - na ścieżce, 78
- operacja, 24
 - istotna, 81
 - programu, 25
 - występująca w akcji, 24
 - występująca w dozorze, 24, 25

- operator
 - A, 16
 - E, 16
 - F, 17
 - G, 17
 - R, 17
 - U, 16
- połączenie tranzycji czasowych, 31
- podzielenie tranzycji czasowych, 31
- poprzednik stanu kontrolnego, 25
- prawdziwość formuły, 17
- proces, 22
 - niezależny od buforów, 66
 - zredukowany, 90
- produkt automatów czasowych, 12
- program, 22
 - strukturalny, 26
 - zredukowany, 90
- protokół Fischera, 34
- przebieg, 9
 - automatu czasowego, 12
 - – postępujący, 12
 - programu, 30
 - – postępujący, 30
 - – silnie uczciwy, 31
 - – słabo uczciwy, 31
- przejście akcyjne
 - automatu, 11
 - programu, 30
- przejście czasowe
 - automatu, 11
 - programu, 30
- relacja
 - \cong_{gp} , 57
 - \cong_{pa} , 43
 - \cong_{pp} , 93
- silna bisymulacja, 19
- stan, 9
 - automatu, 11
 - kontrolny, 22
 - – źródłowy, 22
 - – docelowy, 22
 - – istotny, 81
 - – końcowy, 25
 - – należący do ścieżki, 25
 - – obserwowany, 82
 - – osiągalny, 25
 - – początkowy, 22
 - – wejściowy cyklu, 26
- osiągalny, 9
- programu, 28
- system tranzycyjny, 9
 - automatu, 11
 - programu, 29
- ścieżka, 9
 - w procesie, 25
 - – maksymalna, 26
 - – prosta, 26
- tranzycja
 - gotowa do wykonania, 29
 - istotna, 84
 - możliwa do wykonania, 29
 - należąca do ścieżki, 25
 - pilna, 23
 - przeterminowana, 29
 - wchodząca do stanu, 25
 - wychodząca ze stanu, 25
- wartościowanie
 - opóźnień, 28
 - – początkowe, 28
 - wyrażeń arytmetycznych, 27
 - wyrażeń logicznych, 27
 - zegarów, 10
 - – początkowe, 10
 - zmiennych i buforów, 27
 - – po wykonaniu akcji, 28
 - – początkowe, 27
- warunek
 - na zegarach, 10
 - – dla tranzycji, 40
 - umożliwienia tranzycji automatu, 11
 - **CD1**, 71
 - **CD2**, 73
 - **DD1**, 70
 - **DD2**, 70
 - **DD3**, 70

- **SD**, 81
- **TD1**, 79
- **TD2**, 79

weryfikacja modelowa, 18

wykonanie, 9

- automatu czasowego, 12
- programu, 30

wyrażenie arytmetyczne, 21

wyrażenie logiczne, 21

złożenie automatów czasowych, 12

zależność

- czasowa, 79
- danych, 70
- od synchronizacji, 81
- przepływu sterowania, 71
- stanów, 81

zegar, 10

- do wyzerowania, 11
- odpowiadający tranzycji, 40

zerowanie zegarów, 10

zmienna programu, 21

- definiowana przez operację, 25
- dzielona, globalna, 24
 - – niezależna, 51
- używana w operacji, 25
- lokalna procesu, 24
- obserwowana, 82
- występująca w akcji, 24
- występująca w instrukcji, 24
- występująca w procesie, 24
- występująca w wyrażeniu arytmetycznym, 23
- występująca w wyrażeniu logicznym, 24

zmienna zdaniowa dla programu, 32

Symbole i oznaczenia

\xrightarrow{dd}	zależność danych, str. 70
\xrightarrow{cd}	zależność przepływu sterowania, str. 71
\xrightarrow{td}	zależność czasowa, str. 79
\xrightarrow{sd}	zależność od synchronizacji, str. 81
\cong_{gp}	relacja na stanach automatu globalnego i produktowego, str. 93
\cong_{pa}	relacja na stanach programu i automatu, str. 43
\cong_{pp}	relacja na stanach programu i zredukowanego programu, str. 57
$action(t)$	akcja tranzycji t , str. 22
$\mathcal{B}(g, v)$	wartość wyrażenia logicznego g dla wartościowania v , str. 27
B	zbiór buforów programu, str. 21
$\mathcal{E}(e, v)$	wartość wyrażenia arytmetycznego e dla wartościowania v , str. 27
$constraint(t)$	warunek na zegarach dla tranzycji t , str. 40
$c(t)$	zegar odpowiadający tranzycji t , str. 40
$def(a)$	zbiór zmiennych definiowanych przez operację a , str. 25
$delay(t)$	dozwolone opóźnienie tranzycji t , str. 22
$delay(\pi)$	opóźnienie na ścieżce π , str. 78
$\delta \in \mathbb{R}_+$	etykieta przejścia czasowego
$enabled(t, l)$	tranzycja t jest możliwa do wykonania w konfiguracji l , str. 29
$enabled(t, s)$	tranzycja t jest możliwa do wykonania w stanie s , str. 29
$expired(t, s)$	tranzycja t jest przeterminowana w stanie s , str. 29
$fireable(t, s)$	tranzycja t jest gotowa do wykonania w stanie s , str. 29
Γ_i	zbiór etykiet procesu P_i , str. 21
$\Gamma = \bigcup_{i=1}^n \Gamma_i$	zbiór etykiet programu, str. 21
$\gamma \in \Gamma$	etykieta przejścia akcyjnego
$\Gamma(\gamma)$	zbiór numerów procesów mających etykietę γ , str. 28
$guard(t)$	dozór tranzycji t , str. 22
$guards(q)$	zbiór operacji z dozorów tranzycji wychodzących z $q \in Q$, str. 25
$in(q)$	zbiór tranzycji wchodzących do stanu q , str. 25
$\mathcal{J}(v, \alpha)$	wartościowanie otrzymane z v po wykonaniu akcji α , str. 28
$\lambda \in \Gamma \cup \mathbb{R}_+$	etykieta przejścia w systemie tranzycyjnym dla programu, str. 29
$l = (q_1, \dots, q_n, v)$	konfiguracja programu, str. 28
$label(t)$	etykieta tranzycji t , str. 22
$label(\gamma, l, l')$	etykieta dla tranzycji automatu składowego, str. 51
$lower_bound(t)$	dolne ograniczenie dozwolonego opóźnienia tranzycji t , str. 77

$lower_bound_sign : (t)$	ostrość dolnego ograniczenia dozwolonego opóźnienia tranzycji t , str. 77
$lower_bound(\pi)$	dolne ograniczenie opóźnienia na ścieżce π , str. 78
$lower_bound_sign : (\pi)$	ostrość dolnego ograniczenia opóźnienia na ścieżce π , str. 78
\mathcal{M}_a	model automatu globalnego \mathcal{TA} , str. 13
\mathcal{M}'_a	model automatu produktowego \mathcal{TA}' , str. 13
\mathcal{M}'	model zredukowanego programu \mathcal{P}' , str. 32
$\mathcal{M}, \mathcal{M}_p$	model programu \mathcal{P} , str. 32
μ	wartościowanie opóźnień, str. 28
μ^0	początkowe wartościowanie opóźnień, str. 28
$\mu(q)$	opóźnienie w stanie kontrolnym q , str. 28
$nut(P_i)$	maksymalna liczba tranzycji pilnych wychodzących z jednego stanu, str. 39
$nut(q)$	liczba tranzycji pilnych wychodzących ze stanu q , str. 39
$opers(action(t))$	zbiór wszystkich operacji z akcji tranzycji t , str. 24
$opers(guard(t))$	zbiór wszystkich operacji z dozoru tranzycji t , str. 24
$opers(t)$	zbiór wszystkich operacji tranzycji t , str. 24
Ops	zbiór wszystkich operacji programu, str. 24
$out(q)$	zbiór tranzycji wychodzących ze stanu q , str. 25
π	ścieżka w procesie, str. 25
P_i	i -ty proces, str. 21
P'_i	zredukowany i -ty proces, str. 90
\mathcal{P}	program, str. 21
\mathcal{P}'	zredukowany program, str. 90
$\Psi(X)$	zbiór warunków na zegarach ze zbior X , str. 10
ψ	warunek na zegarach, str. 10
PV	zbiór zmiennych zdaniowych, str. 32
PV'	zbiór zmiennych zdaniowych dla automatów składowych, str. 54
$q \in Q$	stan kontrolny, str. 21
$q^0 \in Q$	początkowy stan kontrolny, str. 21
$q_1 \implies q_2$	istnieje ścieżka z q_1 do q_2 w procesie, str. 86
$q_1 \xrightarrow{inv} q_2$	istnieje niewidoczna ścieżka z q_1 do q_2 w procesie, str. 86
$Q = \bigcup_{i=1}^n Q_i$	zbiór stanów kontrolnych programu, str. 21
Q_i	zbiór stanów kontrolnych procesu P_i , str. 21
Q'_i	zbiór stanów w zredukowanym procesie P'_i , str. 90
Q_i^R	zbiór stanów, z których są osiągalne stany istotne w P_i , str. 84
Q_i^φ	zbiór stanów istotnych w procesie P_i , str. 84
$s = (q_1, \dots, q_n, v, \mu)$	stan programu, str. 28
$s^0 = (q_1^0, \dots, q_n^0, v^0, \mu^0)$	początkowy stan programu, str. 29
$source(t)$	stan źródłowy tranzycji t , str. 22
$synch(t)$	czy tranzycja t jest synchroniczna, str. 28
Σ	zbiór etykiet automatu \mathcal{TA} , str. 10
$\Sigma(\lambda)$	zbiór numerów automatów mających etykietę λ , str. 12
σ	ścieżka w systemie tranzycyjnym, przebieg programu, str. 30
TS_a	system tranzycyjny automatu globalnego \mathcal{TA} , str. 11
TS'_a	system tranzycyjny automatu produktowego \mathcal{TA}' , str. 11
TS, TS_p	system tranzycyjny programu \mathcal{P} , str. 29
TS'	system tranzycyjny zredukowanego programu \mathcal{P}' , str. 29

\mathcal{TA}	automat globalny dla programu \mathcal{P} , str. 41
\mathcal{TA}_i	i -ty automat składowy, str. 52-53
\mathcal{TA}'	automat produktowy, str. 57
$target(t)$	stan źródłowy tranzycji t , str. 22
τ^0	początkowe wartościowanie zegarów, str. 10
τ	wartościowanie zegarów, str. 10
$\tau(x)$	wartość zegara x , str. 10
$T = \bigcup_{i=1}^n T_i$	zbiór tranzycji programu, str. 21
T_i	zbiór tranzycji procesu P_i , str. 21
$t \in T$	tranzycja, str. 21
$upper_bound(t)$	górne ograniczenie dozwolonego opóźnienia tranzycji t , str. 77
$upper_bound_sign : (t)$	ostrość górnego ograniczenia dozwolonego opóźnienia tranzycji t , str. 77
$upper_bound(\pi)$	górne ograniczenie opóźnienia na ścieżce π , str. 78
$upper_bound_sign : (\pi)$	ostrość górnego ograniczenia opóźnienia na ścieżce π , str. 78
$upper_contr(t)$	górne ograniczenie na zegary dla tranzycji t , str. 40
$upper_delay(t)$	górne dozwolone opóźnienie tranzycji t , str. 22
$urgent(t)$	atrybut pilności tranzycji t , str. 22
$use(a)$	zbiór zmiennych używanych w operacji a , str. 25
v	wartościowanie zmiennych i buforów, str. 27
v^0	początkowe wartościowanie zmiennych, str. 27
$vars(a)$	zbiór zmiennych występujących w operacji a , str. 23
$vars(e)$	zbiór zmiennych występujących w wyrażeniu e , str. 23
V	zbiór zmiennych programu, str. 21
V_G	zbiór zmiennych globalnych, str. 24
V'_G	zbiór niezależnych zmiennych globalnych, str. 51
V_i	zbiór zmiennych lokalnych procesu P_i , str. 24
V'_i	zbiór zmiennych od których zależy postać niezmiennika procesu P_i , str. 51
\mathcal{V}	funkcja wartościująca stany z \mathcal{TS} , str. 32
\mathcal{V}'	funkcja wartościująca stany z \mathcal{TS}' , str. 32
$\mathcal{V}_{\mathcal{TA}}$	funkcja wartościująca lokacje automatu globalnego, str. 42
$\mathcal{V}'_{\mathcal{TA}_i}$	funkcja wartościująca lokacje i -tego automatu składowego, str. 54
\mathcal{V}_a	funkcja wartościująca stany z \mathcal{TS}' , str. 13
\mathcal{V}'_a	funkcja wartościująca stany \mathcal{TS}'_a , str. 13
$v(y)$	wartość zmiennej y , str. 27
X	zbiór zegarów automatu, str. 10
Y	zbiór zegarów do wyzerowania, str. 10

Bibliografia

- [ABK⁺97] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. In *Proc. of Int. Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *LNCS*, pages 346–360. Springer-Verlag, 1997.
- [ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model checking for real-time systems. In *Proc. of the 5th Symp. on Logic in Computer Science (LICS'90)*, pages 414–425. IEEE Computer Society, 1990.
- [ACD⁺92] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. of the 13th IEEE Real-Time Systems Symposium (RTSS'92)*, pages 157–166. IEEE Computer Society, 1992.
- [ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
- [AD90] R. Alur and D. Dill. Automata for modelling real-time systems. In *Proc. of the Int. Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer-Verlag, 1990.
- [AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [AH92] R. Alur and T. Henzinger. Logics and models of real time: A survey. In *Proc. of REX Workshop 'Real Time: Theory and Practice'*, volume 600 of *LNCS*, pages 74–106. Springer-Verlag, 1992.
- [AH97] R. Alur and T. Henzinger. Modularity for timed and hybrid systems. In *Proc. of the 8th Int. Conf. on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*, pages 74–88. Springer-Verlag, 1997.
- [AK96] R. Alur and R. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 220–231. Springer-Verlag, 1996.
- [AL91] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In *REX workshop on Real-Time: Theory in Practice*, volume 600 of *LNCS*, pages 1–27. Springer-Verlag, 1991.

- [ASU02] A Aho, R Sethi, and J Ullman. *Kompilatory*. WNT, 2002.
- [BBFL03] G. Behrmann, P. Bouyer, E. Fleury, and K. Larsen. Static guard analysis in timed automata verification. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 254–277. Springer-Verlag, 2003.
- [BCC+99] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. of the ACM/IEEE Design Automation Conference (DAC'99)*, pages 317–320, 1999.
- [BCG88] M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1/2):115–131, 1988.
- [BCP+01] V. Bertin, E. Closse, M. Poize, J. Pulous, J. Sifakis, P. Venier, D. Weil, and S. Yovine. Taxys = esterel + kronos: A tool for verifying real-time properties of embedded systems. *Proc. of the 40th IEEE Conf. Decision and Control*, 3, 2001.
- [BFG+99] M. Bozga, J-C. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier, and J. Sifakis. IF: An intermediate representation for SDL and its applications. In *Proc. of SDL Forum'99*, 1999.
- [BFG00] M. Bozga, J-C. Fernandez, and L. Ghirvu. Using static analysis to improve automatic test generation. In *Proc. of the 6th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 235–250. Springer-Verlag, 2000.
- [BG00] G. Bollella and J. Gosling. The real-time specification for Java. *Computer*, 33(6):47–54, 2000.
- [BGO02] V. Braberman, D. Garbervetsky, and A. Olivero. Improving the verification of timed systems using influence information. In *Proc. of Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 21–36. Springer-Verlag, 2002.
- [Bry86] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.
- [BSW69] K. Bartlett, R. Scantlebury, and P. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.
- [CES86] E. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
- [CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

- [CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
- [DGG94] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving ACTL*, ECTL* and CTL*. In *Proc. of the IFIP Working Conference on Programming Concepts, Methods and Calculi (PRO-COMET'94)*. Elsevier, 1994.
- [DGKK98] D. Dams, R. Gerth, B. Knaack, and R. Kuiper. Partial-order reduction techniques for real-time model checking. In *Proc. of the 3rd Int. Workshop on Formal Methods for Industrial Critical Systems*, pages 157–169, 1998.
- [DHS99] M.B. Dwyer, J. Hatcliff, and D. Schmidt. Bandera: Tools for automated reasoning about software systems behaviour. *ECRIM News*, 36, 1999.
- [DJJ02] A. Doroś, A. Janowska, and P. Janowski. From specification languages to Timed Automata. In *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'02)*, volume 161(1) of *Informatik-Berichte*, pages 117–128. Humboldt University, 2002.
- [DJJ⁺03a] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Pólróla, M. Szreter, B. Woźna, and A. Zbrzezny. Verics: A tool for verifying timed automata and Estelle specifications. In *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 278–283. Springer-Verlag, 2003.
- [DJJ⁺03b] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Pólróla, M. Szreter, B. Woźna, and A. Zbrzezny. Verics: weryfikator dla automatów czasowych i specyfikacji zapisanych w języku Estelle. In *Mat. X Konf. Systemy Czasu Rzeczywistego (SCR'03)*, pages 17–26. Instytut Informatyki Politechniki Śląskiej, 2003. In Polish.
- [DJPV02] W. Damm, B. Josko, A. Pnueli, and A. Votintseva. Understanding UML: A formal semantics of concurrency and communication in real-time uml, 2002.
- [DOTY95] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 208–219. Springer-Verlag, 1995.
- [DOY94] C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In *Proc. of the 7th IFIP WG. G.1 Int. Conf. on Formal Description Techniques (FORTE'94)*, pages 227–242. Chapman & Hall, October 1994.
- [DT98] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Proc. of the 4th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *LNCS*, pages 313–329. Springer-Verlag, 1998.
- [DW99] M. Dickhofer and T. Wilke. Timed alternating tree automata: The automata-theoretic solution to the TCTL model checking problem. In *Proc. of the 26th Int. Colloquium on Automata, Languages and Programming (ICALP'99)*, volume 1664 of *LNCS*, pages 281–290. Springer-Verlag, 1999.

- [DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with KRONOS. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, pages 66–75, 1995.
- [DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS'96)*, pages 73–81. IEEE Computer Society, 1996.
- [EC82] E. A. Emerson and E. M. Clarke. Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.
- [EJ93] E. A. Emerson and C. S. Jutla. Symmetry and model checking. In *Proc. of the 5th Int. Conf. on Computer Aided Verification (CAV'93)*, volume 697 of *LNCS*, pages 463–478. Springer-Verlag, 1993.
- [Eme90] E. A. Emerson. *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, chapter Temporal and Modal Logic, pages 995–1067. Elsevier, 1990.
- [GKPP99] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *Information and Computation*, 150:132–152, 1999.
- [Gol92] R. Goldblatt. *Logics of Time and Computation*. CSLI Lecture Notes. CSLI Publications, Stanford University, 1992.
- [GSSAL94] R. Gawlick, R. Segala, J. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. In *Proc. of the 21st Int. Colloquium on Automata, Languages and Programming (ICALP'94)*, volume 820 of *LNCS*, pages 166–177. Springer-Verlag, 1994.
- [HC84] G. E. Hughes and M. J. Cresswell. *A Companion to Modal Logic*. Methuen, 1984.
- [HCD⁺99] J. Hatcliff, J. C. Corbett, M. B. Dwyer, S. Sokolowski, and H. Zheng. A formal study of slicing for multi-threaded programs with JVM concurrency primitives. In *Proc. of the Int. Symposium on Static Analysis (SAS'99)*, pages 1–18, 1999.
- [HDZ99] J. Hatcliff, M. B. Dwyer, and H. Zheng. Slicing software for model construction. In *Proc. of the ACM Workshop on Partial Evaluation and Program Manipulation (PEPM'99)*, volume BRICS NS-99-1 of *BRICS Note Series*, 1999.
- [HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.
- [HNSY94a] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–224, 1994.

- [HNSY94b] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.
- [Hol97] G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Eng.*, 23(5):279–295, 1997.
- [ISO97] ISO/IEC 9074(E), Estelle - a formal description technique based on an extended state-transition model. International Standards Organization, 1997.
- [JJ03] A. Janowska and P. Janowski. Slicing timed systems. In L. Czaja, editor, *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'03)*, volume 1, pages 235–250. Warsaw University, 2003.
- [JJ04] A. Janowska and P. Janowski. Slicing timed systems. *Fundamenta Informaticae*, 60(1-4):187–210, 2004.
- [JP06a] G. Jakubowska and W. Penczek. Modeling and checking timed authentication of security protocols. In *Proc. of the Workshop on Concurrency, Specification and Programming (CSP'06)*, volume 206 of *Informatik Berichte*, pages 280–291. Humboldt University, 2006.
- [JP06b] A. Janowska and W. Penczek. Path compression in timed automata. In *Proc. of the Workshop on Concurrency, Specification and Programming (CSP'06)*, volume 206 of *Informatik Berichte*, pages 340–351. Humboldt University, 2006.
- [JP07] G. Jakubowska and W. Penczek. Is your security protocol on time? In *Proc. of Int. Symposium on Fundamentals of Software Engineering (FSEN'07)*, LNCS (To appear). Springer-Verlag, 2007.
- [KLM⁺98] R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigün. Static partial order reduction. In *Proc. of the 4th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *LNCS*, pages 345–357. Springer-Verlag, 1998.
- [Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
- [McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
- [Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.
- [Min06] MiniSat. <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat>, 2006.
- [MLAH99] J. Møller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Difference Decision Diagrams. In *Proc. of the 13th Int. Workshop Computer Science Logic (CSL'99)*, volume 1683 of *LNCS*, pages 111–125. Springer-Verlag, 1999.

- [MT98] L. Millett and T. Teitelbaum. Slicing promela and its applications to model checking, simulation and protocol understanding. In *Proc. of the 4th Int. SPIN Workshop*, 1998.
- [MT01] A. Mitschele-Thiel. *Systems Engineering with SDL - Developing Performance-Critical Communication Systems*. John Wiley & Sons, 2001.
- [NDJ06] W. Nabiałek, P. Dembiński, and P. Janowski. Towards Promela verification using VerICS. In *Proc. of the Workshop on Concurrency, Specification and Programming (CSP'06)*, volume 206 of *Informatik Berichte*, pages 219–230. Humboldt University, 2006.
- [NPLK06] A. Niewiadomski, W. Penczek, S. Lasota, and J. Kowalski. Weryfikacja UML z wykorzystaniem systemu VerICS. In *Systemy Czasu Rzeczywistego 2006*, Systemy informatyczne z ograniczeniami czasowymi, 2006.
- [NSY92] X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *Software Engineering*, 18(9):794–804, 1992.
- [OWS06] M. Orzechowski, B. Woźna, and T. Siwiak. Towards verification of Java programs in VerICS. Technical Report 997, ICS PAS, 2006.
- [Pag96] F. Pagani. Partial orders and verification of real-time systems. In *Proc. of the 4th Int. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'96)*, volume 1135 of *LNCS*, pages 327–346. Springer-Verlag, 1996.
- [PBG05] M. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *International Journal of Software Tools for Technology Transfer*, 7(2):156–173, 2005.
- [Pel98] D. Peled. Ten years of partial-order reductions. In *Proc. of the 10th Int. Conf. on Computer Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 17–28. Springer-Verlag, 1998.
- [Pen95] W. Penczek. Branching time and partial order in temporal logics. In L. Bolc and A. Szalas, editors, *Time and Logic: A Computational Approach*, pages 179–228. UCL Press Ltd., 1995.
- [PL00] P. Pettersson and K. G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.
- [Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th Int. Symp. Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977.
- [PP06] W. Penczek and A. Pórlola. *Advances in Verification of Time Petri Nets And Timed Automata (A Temporal Logic Approach)*. Springer, 2006.
- [Rei85] W. Reisig. *Petri Nets. An Introduction*, volume 4 of *EACTS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

- [Tip95] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3, 1995.
- [Val89] A. Valmari. Stubborn sets for reduced state space generation. In *Proc. of the 10th Int. Conf. on Applications and Theory of Petri Nets (ICATPN'89)*, volume 483 of *LNCS*, pages 491–515. Springer-Verlag, 1989.
- [Val98] A. Valmari. The state explosion problem. In *Lecture Notes on Petri Nets I: Basic Models. Advances in Petri Nets*, volume 1491 of *LNCS*. Springer-Verlag, 1998.
- [Wal83] B. Walter. Timed Petri nets for modelling and analysing protocols with real-time characteristics. In *Proc. of the 3rd IFIP Workshop on Protocol Specification, Testing, and Verification*, pages 149–159. North Holland, 1983.
- [Wan00] F. Wang. Region Encoding Diagram for fully symbolic verification of real-time systems. In *Proc. of the 24th Int. Computer Software and Applications Conf. (COMPSAC'00)*, pages 509–515. IEEE Computer Society, October 2000.
- [Wei84] M. Weiser. Program slicing. *IEEE Trans. on Software Eng.*, 10(4), 1984.
- [WZP03] B. Woźna, A. Zbrzezny, and W. Penczek. Checking reachability properties for timed automata via SAT. *Fundamenta Informaticae*, 55(2):223–241, 2003.
- [YSSC93] T. Yoneda, A. Shibayama, B. H. Schlingloff, and E. M. Clarke. Efficient verification of parallel real-time systems. In *Proc. of the 5th Int. Conf. on Computer Aided Verification (CAV'93)*, volume 697 of *LNCS*. Springer-Verlag, 1993.
- [Zie87] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O.-Informatique Theoretique et Applications*, 21:99–135, 1987.