



Politechnika
Wrocławska

Politechnika Wrocławska
Wydział Informatyki i Zarządzania
Instytut Informatyki Stosowanej

Dziedziczenie zachowań czasowych diagramów stanów języka UML

(praca doktorska)

Dariusz Konieczny

Promotor:
prof. dr hab. Zbigniew Huzar

Wrocław
2005

*Podziękowania dla Promotora
za pozytywne rady
i dla żony - za cierpliwość.*

Spis treści

1	Problematyka i cele	3
1.1	Przedmiot pracy	3
1.2	Tło problemu	5
1.3	Cel pracy	9
1.4	Zawartość pracy	10
2	Język formalny etykietowany czasem	10
2.1	Kwestia czasu	10
2.2	Definicja języka formalnego etykietowanego czasem (JEC)	12
2.3	Zachowanie środowiska wyrażone za pomocą JEC	14
2.4	Zachowanie obiektu wyrażone za pomocą JEC	15
3	Dziedziczenie zachowań czasowych	16
3.1	Składniki definicji	16
3.2	Definicja dziedziczenia zachowań wyrażona poprzez JEC	17
3.3	Rodzaje dziedziczenia zachowań	18
4	Maszyna stanowa w UML	19
4.1	Składnia diagramów stanów w UML	20
4.2	Semantyka maszyny stanowej w UML	22
5	Model diagramu stanów	34
5.1	Graf diagramu stanów	34
5.2	Diagram stanów	35
6	Konkretyzacja metamodelu diagramu stanów	37
6.1	Wzbudzenie tranzycji a uruchomienie tranzycji	37
6.2	Kolejność i czas realizacji tranzycji zdarzeniowych	37
6.3	Preferencja tranzycji wewnętrznych	41
6.4	Preferencja tranzycji nieczasowych	41
6.5	Kolejność realizacji tranzycji czasowych	42
6.6	Moment reakcji maszyny podczas wykonywania tranzycji	42
6.7	Zbiór reakcji a nie kolekcja	42
7	Model zachowania środowiska	42
7.1	Właściwości	42
7.2	Operacje na diagramie stanów środowiska	43
7.3	Generowanie JEC przez diagram stanów reprezentujący środowisko	48
8	Model zachowania obiektu	50
8.1	Właściwości	50
8.2	Operacje na diagramie stanów obiektu	50
8.3	Opis działania maszyny stanowej obiektu	61
8.4	Diagram stanów maszyny jako transformator języka środowiska	61

9	Funkcje porównujące zachowania czasowe	63
10	Algorytm badania zgodności zachowań czasowych	64
10.1	Wstęp	64
10.2	Operacje na ciągach reakcji	65
10.3	Pojęcie konfrontacji	66
10.4	Opis algorytmu	68
11	Właściwości algorytmu	70
11.1	Poprawność algorytmu	70
11.2	Implementacja algorytmu	78
11.3	Przykładowe zastosowania i wyniki działania algorytmu	78
11.4	Złożoność algorytmu	83
11.5	Zastosowanie algorytmu do badania zgodności zachowań bezczasowych	84
12	Podsumowanie	85
12.1	Wyniki prac	85
12.2	Problemy na przyszłość	86
13	Literatura	91

1 Problematyka i cele

1.1 Przedmiot pracy

Wytwarzanie oprogramowania jest procesem wieloetapowym i wielokierunkowym. Znaczną część tego procesu polega na modelowaniu, czyli "badaniu złożonych układów, zjawisk i procesów na podstawie skonstruowanych modeli"¹. Metodologie obecnie wykorzystywane w tym procesie są zorientowane na obiektowość, czyli na wyodrębnianiu dobrze zdefiniowanych bytów (klas, obiektów), ich cech oraz związków między nimi. Obecnie wytwarza się oprogramowanie służące w każdej dziedzinie życia i działalności człowieka, zatem pożądanym byłby jeden język do modelowania w możliwie największej liczbie dziedzin. Językiem, który staje się standardem modelowania w procesie wytwarzania oprogramowania, jest obecnie UML (Unified Modeling Language). Jest to język stale rozwijany od początku lat dziewięćdziesiątych i uzupełniany (od wersji 1.0 w roku 1997 do 1.5 w roku 2003, obecnie trwają prace nad wersją 2.0), stąd pewne jego składowe mają ugruntowane znaczenie i zastosowanie, gdy tymczasem inne wymagają dalszych badań, szukania rozwiązań.

Na początkowym etapie tworzenia systemu informatycznego określone są wymagania jakie ma on spełniać. Wymagania te można podzielić na wymagania funkcjonalne i нефункционалне. Wymagania funkcjonalne można wyrazić w notacji UML, natomiast нефункционалне już nie. Podczas całego procesu wytwarzania oprogramowania brane są zawsze pod uwagę wymagania funkcjonalne, gdyż spełnienie tych wymagań jest warunkiem koniecznym końcowego produktu. Tworzenie modelu ma na celu ustalenie związków między różnymi elementami modelu. Jednym z głównych elementów modelu jest klasa. Klasa posiada właściwości statyczne (strukturalne) i dynamiczne (behawioralne). Właściwości statyczne klasy są określone poprzez atrybuty, operacje oraz związki z innymi klasami. Atrybuty są określane przez nazwy i typy, natomiast operacje – poprzez nazwę, listę parametrów oraz metodę. Metoda jest implementacją operacji i specyfikuje algorytm lub procedurę generującą wynik operacji. Przy podejściu obiektowym, jak ma to miejsce w książce Boocha, Rumbaugh'a i Jacobsona [11], wyróżnia się w modelowaniu obiektowym trzy najważniejsze rodzaje związków między elementami modelu, w szczególności między klasami: zależności, uogólnienia i asocjacje. W pracy autor zajmie się związkiem uogólnienia.

Klasę opisuje też jej zachowanie, czyli "specyfikacja tego jak stan obiektu (instancji klasy) zmienia się w czasie w odpowiedzi na zewnętrzne zdarzenia i wewnętrzne przetwarzanie" [50]. Z punktu widzenia obserwatora zachowanie to ciąg reakcji obiektu wywołany ciągiem zdarzeń zewnętrznych. Jednym ze sposobów opisu zachowania klasy w języku UML są maszyny stanów, reprezentowane przez diagramy stanów.

Maszyna stanów to automat skończony z ograniczoną pamięcią. Konfigurację maszyny w danej chwili określa jej stan oraz zawartość pamięci. W przypadku obiektów pamięć jest reprezentowana przez atrybuty, którym można przypisać wartości określonych typów. Stan pamięci to wartościowanie określonych atrybutów.

Konfigurację obiektu wyznacza stan oraz nieujawnione na diagramie wartościowanie

¹<http://encyklopedia.wp.pl/>

atrybutów. Obiekt w trakcie działania zmienia konfigurację. Zmiana konfiguracji może się wyrażać poprzez zmianę stanu (oraz związaną z tym zmianą wartościowania) lub poprzez wewnętrzną zmianę wartości atrybutów. Te pierwsze są opisane poprzez tranzycje od jednego stanu do innego, natomiast drugie zmiany poprzez tranzycje wewnętrzne. Tranzycje opisują przy jakich warunkach i zdarzeniach zewnętrznych może nastąpić zmiana stanu (zmiana wartościowania atrybutów) i co może być efektem zewnętrznym tej zmiany (reakcja).

Związek uogólnienia jest nazywany również dziedziczeniem lub generalizacją. Związek uogólnienia między klasami to [50] związek taksonomiczny między bardziej ogólną klasą (zwaną nadklasą lub przodkiem) a pewną bardziej specyficzną klasą (zwaną podklasą lub potomkiem). Potomek posiada wszystkie właściwości przodka i może posiadać dodatkowe atrybuty, operacje i związki. Instancja podklasy jest również instancją nadklasy i dziedziczy jej właściwości. Termin uogólnienie wyrasta z postulatu Liskov [41] mówiący o tym, że:

Użycie w miejsce obiektu klasy ogólnej obiektu klasy specjalizowanej powinno być z punktu widzenia obserwatora zewnętrznego niezauważalne.

Relacja generalizacji ma duże znaczenie w wielu etapach tworzenia systemu. Zachodzenie tej relacji umożliwia ponowne wykorzystanie wcześniej stworzonych klas, poprzez stworzenie podklasy, uzupełnionej tylko o nowe właściwości. Z postulatu Liskov wynika też bezpośrednia możliwość użycia obiektu klasy specjalizowanej w miejscu obiektu klasy ogólnej. Upraszcza się też w ten sposób etap testowania i optymalizacji kodu w języku programowania obiektowego, oraz skraca się czas poprawiania błędów.

Dziedziczenie ma aspekt statyczny i dynamiczny. Pierwszy to dziedziczenie atrybutów, operacji i związków, drugi - dziedziczenie zachowania obiektu. Dla dziedziczonej operacji można przypisać inną metodę, która spełniałaby postulat Liskov. W przypadku wstawienia obiektu klasy potomnej w miejsce obiektu klasy bazowej i wywołaniu operacji, która ma dwie metody (jedną w klasie bazowej i drugą w klasie pochodnej), możliwe są dwie sytuacje: wywołana zostanie metoda z klasy bazowej lub wywołana zostanie metoda z klasy pochodnej. Mechanizm zastosowany w drugim przypadku nazywany jest polimorfizmem operacji. Definicja uogólnienia nie określa zależności między metodami potomka i przodka, tylko deklaruje potrzebę spełnienia postulatu Liskov.

Trudność zdecydowania, czy zachowanie klasy pochodnej jest zgodne pod względem dziedziczenia zachowania z zachowaniem klasy bazowej spowodowana jest trudnością ustalenia definicji dziedziczenia zachowań. Jak rozumieć dziedziczenie zachowania? Czy zachowanie maszyny stanowej podklasy mają być identyczne z maszyną nadklasy czy raczej z jakimś modelem, który również maszyna nadklasy reprezentuje? A może jednak tylko podobne? Jeśli podobne, to jak scharakteryzować to podobieństwo? Czy ważna jest struktura zachowań wewnątrz maszyny stanowej czy tylko jej wynik, a może tylko wyniki obserwowalne przez zewnętrznego obserwatora [22] [23]?

Jeśli w wymaganiach formalnych występują ilościowe zależności czasowe, mówimy wtedy o systemach czasu rzeczywistego. W przypadku relacji generalizacji, o ile w rozważaniach bezczasowych ważna jest tylko kolejność zachodzenia pewnych zdarzeń i reakcji na nie, to w systemach czasu rzeczywistego należy również w jakiś sposób porównać czasy tych elementów.

Przedmiotem pracy jest badanie dziedziczenia zachowań czasowych maszyn stanowych klas reprezentowanych diagramami stanów języka UML.

1.2 Tło problemu

Termin "dziedziczenia zachowań klas" (DZ) jest różnie definiowany, jednak za punkt wyjścia uznany został postulat Liskov. Istnieją dwa podejścia przy definiowaniu DZ z wykorzystaniem tego postulatu: logiczne i operacyjne.

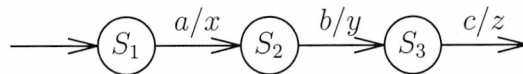
Pierwsze podejście polega na tym, że dla każdej operacji klasy zapisuje się zbiór warunków początkowych α i końcowych β jakie musi spełniać obiekt przed i po wykonaniu jego metody. Dziedziczenie zachowań zachodzi wówczas, gdy w klasie dziedziczącej zachowanie warunki początkowe α' są co najwyżej osłabione ($\alpha \Rightarrow \alpha'$) natomiast końcowe β' co najwyżej wzmocnione ($\beta' \Rightarrow \beta$) [41]. Wynika to stąd, że specjalizowana klasa musi "obsługiwać" co najmniej takie same sytuacje jak klasa ogólna, natomiast efektem obsługi klasy specjalizowanej nie może być sytuacja, która nie była przewidziana w klasie ogólnej. Tak postawiony problem DZ jest nierozstrzygalny, gdyż problem sprawdzenia prawdziwości powyższych implikacji jest w ogólnym przypadku nierozstrzygalny.

Zapisanie warunków logicznych może być dokonane na wiele sposobów. Niektórzy spośród autorów używają wcześniej stworzonych notacji jak algebry Larch [41], języka Larch/ML [62], rachunek lambda [39], język OCL [55], inni tworzą własną algebrę [40]. W ramach tych opracowań nie rozważa się czasu rzeczywistego.

Drugie podejście - operacyjne - polega na reprezentacji maszyn stanów podklasy i nadklasy w postaci diagramów stanów i sprawdzeniu zachodzenia zgodności zachowań jakie są generowane przez obydwie diagramy. W tym podejściu pojawia się kwestia jak porównywać maszyny stanowe podklasy i nadklasy. W literaturze spotyka się różne propozycje porównywania maszyn stanów.

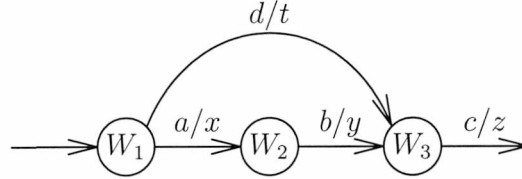
Na użytek rozważań przedstawionych we wstępie przyjęto następujące oznaczenia graficzne używane do reprezentowania maszyn stanowych: okręgi oznaczają stany, łuki to tranzycje, a opis na łukach a/x oznacza, że wystąpienie zdarzenia a powoduje zmianę stanu i wygenerowanie reakcji x .

Początkowo wyróżniano dwa możliwe uściślenia definicji dziedziczenia zachowań [18] (podobnie w [9]). Dla ich wyjaśnienia przyjęto fragment maszyny nadklasy jak na rys. (1). Niech MS_1 oznacza maszynę nadklasy, natomiast MS_2 – maszyną podklasy. Przyjęto następujące oznaczenia ($i \in \{1, 2\}$): Z_i – zbiór zdarzeń, które mogą powodować zmianę stanu w maszynie MS_i ; R_i – zbiór reakcji, które może wygenerować i -ta maszyna; ZCZ_i – zbiór takich ciągów zdarzeń ze zbioru Z_i , że każde zdarzenie z ciągu powoduje zmianę stanu maszyny MS_i , $C_i(CZ)$ – ciąg reakcji jakie generuje i -ta maszyna w odpowiedzi na ciąg zdarzeń CZ . Notacja $A|_X$ oznacza obcięcie ciągu A do elementów ze zbioru X , czyli pozostawienie w ciągu w tej samej kolejności tych i tylko tych elementów, które należą do zbioru X .



Rysunek 1: Maszyna nadklasy MS_1

Pierwsza definicja dziedziczenia zachowań, nazywana dziedziczeniem protokołu (ang. *protocol inheritance*), zakłada, że jeśli maszyna nadklasy zmienia swoje stany pod wpływem pewnego ciągu zdarzeń $CZ_1 \in ZCZ_1$ i generuje pewny ciąg reakcji $CR_1 = C_1(CZ_1)$, to maszyna podklasy pod wpływem tej samej sekwencji CZ_1 zdarzeń generuje taką samą sekwencję reakcji $CR_1 = C_2(CZ_1)$. Zależność taka musi zachowywać dla wszystkich $CZ_1 \in ZCZ_1$. Np. na rys. 1 i 2 ciąg zdarzeń $CZ_1 = abc$ generuje w każdej z maszyn MS_1 i MS_2 ciąg reakcji $CR_1 = xyz$.



Rysunek 2: Maszyna podklasy MS_2 dziedzicząca protokół

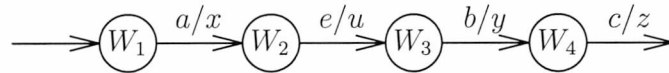
Druga definicja, nazywana dziedziczeniem obserwacyjnym (ang. *projection inheritance*), musi spełniać dwa poniższe warunki:

$$\forall CZ_2 \in ZCZ_2 \bullet (C_2(CZ_2)|_{R_1} = C_1(CZ_2|_{Z_1})) \quad (1)$$

$$ZCZ_1 = \{CZ_1 | \exists CZ_2 \in ZCZ_2 \bullet (CZ_1 = CZ_2|_{Z_1})\} \quad (2)$$

Pierwszy warunek oznacza, że dla każdego ciągu zdarzeń $CZ_2 \in ZCZ_2$, powodującego w maszynie podklasy pewien ciąg reakcji, obserwujemy jaki ciąg reakcji w maszynie nadklasy powoduje ciąg CZ_2 po obcięciu do zdarzeń tej maszyny $CZ_2|_{Z_1}$. Jeśli obserwator nie zauważa nowych elementów ciągu reakcji maszyny podklasy, to działania obydwóch maszyn muszą być dla niego nierozróżnialne, czyli $C_2(CZ_2)|_{R_1} = C_1(CZ_2|_{Z_1})$. Drugi warunek to stwierdzenie, że jeśli wszystkie ciągi zdarzeń wejściowych maszyny podklasy obetniemy do zbioru zdarzeń wejściowych maszyny nadklasy, to otrzymamy zbiór ciągów maszyny nadklasy.

W przykładzie na rys. 1 i 3 ciąg zdarzeń $CZ_2 = aeabc$ generuje w maszynie MS_2 ciąg reakcji $CR_2 = xuyz$, natomiast ciąg zdarzeń $CZ_1 = CZ_2|_{\{a,b,c\}} = abc$ generuje w maszynie MS_1 ciąg reakcji $CR_1 = xyz$ i zachodzi równość $CR_2|_{\{x,y,z\}} = CR_1$.



Rysunek 3: Maszyna podklasy MS_2 w dziedziczeniu obserwacyjnym

Ostatnio pojawiła się nowa, trzecia możliwość [60], nazwana dziedziczeniem "cyklu życia" (ang. *life-cycle inheritance*). (rys 4). W tym przypadku również zakłada się, że zewnętrzny obserwator dostrzega tylko reakcje, które występowały tylko w maszynie nadklasy, jednak porównanie nie jest dokonywane dla wszystkich ciągów zdarzeń rozważanych w dziedziczeniu obserwacyjnym. Ze zbioru zdarzeń nie występujących w maszynie nadklasy wybiera się podzbiór Z_x ($Z_x \subseteq Z_2 \setminus Z_1$). Spośród wszystkich ciągów zdarzeń

rozpatrywanych w dziedziczeniu obserwacyjnym wybiera się wszystkie te i tylko te, które składają się wyłącznie ze zdarzeń z maszyny nadklasy Z_1 lub zbioru Z_x . Niech zbiór takich ciągów oznacza ZCZ_x .

$$ZCZ_x = \{CZ_x | \exists CZ_x \in ZCZ_2 \bullet (CZ_x|_{Z_1 \cup Z_x} = CZ_x)\} \quad (3)$$

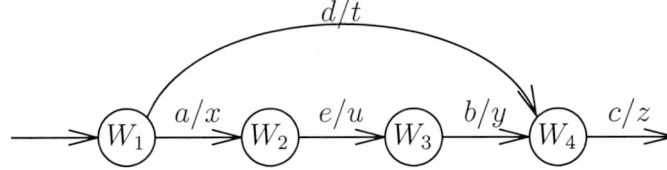
W tej definicji DZ spełnione muszą być następujące warunki:

$$\forall CZ_2 \in ZCZ_x \bullet (C_2(CZ_2)|_{R_1} = C_1(CZ_2|_{Z_1})) \quad (4)$$

$$ZCZ_1 = \{CZ_1 | \exists CZ_2 \in ZCZ_x \bullet (CZ_1 = CZ_2|_{Z_1})\} \quad (5)$$

Jeśli taki zbiór Z_x istnieje, że dla zewnętrznego obserwatora (takiego jak w dziedziczeniu obserwacyjnym) działanie obydwóch maszyn dla wszystkich ciągów ze zbioru ZCZ_x jest nierozróżnialne, to między maszynami zachodzi dziedziczenie cyklu-życia.

Przykładem dziedziczenia cyklu życia są rys. 1 i 4. Ponieważ $Z_2 = \{a, b, c, d, e\}$, zatem można wybrać zbiór $Z_x = \{e\}$ i otrzymamy zbiór jednoelementowy $ZCZ_x = \{aebc\}$. Dalsze rozumowanie przebiega jak dla dziedziczenia obserwacyjnego.



Rysunek 4: Maszyna podklasy MS_2 dziedzicząca cykl życia

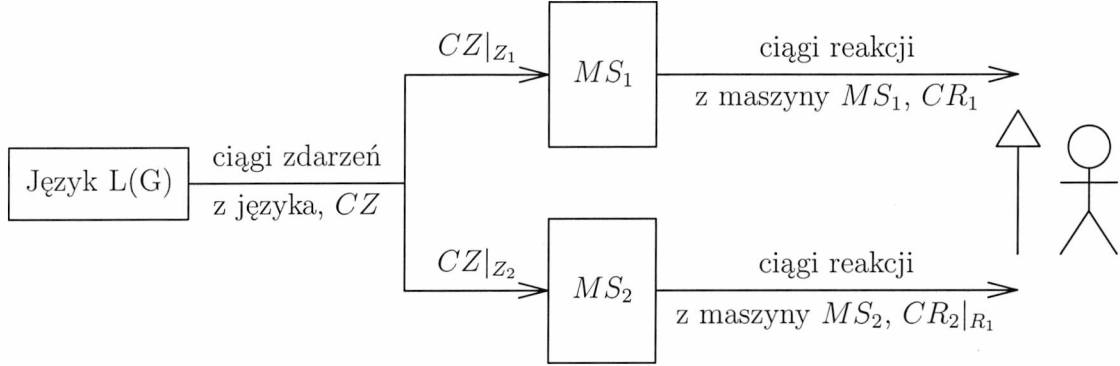
Korzystając z rozważań autorów w [23], możnaby wprowadzić między innymi pojęcie DZ maszyn stanowych parametryzowanych zbiorem sekwencji zdarzeń rys (5). Jeśli $L(G)$ oznacza język generowany przez pewną gramatykę G , to spełniony musiałby być warunek:

$$\forall CZ \in L(G) \bullet (C_2(CZ|_{Z_2})|_{R_1} = C_1(CZ|_{Z_1})) \quad (6)$$

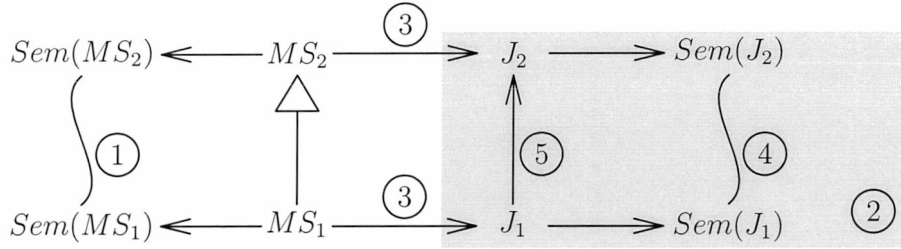
Sprawdzenie ciągów reakcji maszyn dokonywane byłoby tylko dla sekwencji zdarzeń możliwych do wygenerowania przez język $L(G)$.

Często rozwiązania w podejściu operacyjnym polegają na przekształceniu diagramu stanów na inny system spoza UML oraz jednocześnie przekształcenie problemu DZ diagramów UML na inny problem w innej dziedzinie. Autorzy w [18] proponują pięć kroków, które muszą być obecne w takiej metodzie przekształceń (rys. 6).

1. Przyjęcie wybranej lub stworzenie własnej definicji dziedziczenia zachowań wyrażonej w terminach semantyki języka UML.
2. Wybranie języka (spoza UML) do reprezentacji zachowań i dziedziny semantyki, która umożliwi zdefiniowanie dziedziczenia zachowań w tym języku.
3. Zdefiniowanie reguł przekształcania diagramów stanów UML do reprezentacji w wybranym języku.



Rysunek 5: Porównanie parametryzowane językiem zdarzeń



Rysunek 6: Przekształcanie maszyny stanów w język oraz przekształcanie problemu dziedziczenia zachowań w problem z dziedziny J.

4. Zdefiniowanie dziedziczenia zachowań w dziedzinie wybranej semantyki.
5. Sprawdzenie dziedziczenia przez porównanie reprezentacji zachowań w wybranym języku.

Kroki 3–5 muszą gwarantować, że spełnienie definicji z kroku 4 powoduje spełnienie definicji z kroku 1.

Występuje duża różnorodność reprezentacji do których można przekształcić diagramy UML, przykładami są: system transformacji grafów [35], algebra ACSR (Algebra of Communicating Share Resources) [46], CSP (Communicating Sequential Processes) [18], [49], LTS (Labelled Transitions Systems) [9]. Żadne z proponowanych reprezentacji nie uwzględnia czasu rzeczywistego.

Część rozwiązań opiera się na opracowaniu własnej notacji opisującej semantykę diagramów stanów. Takie rozwiązanie proponowane jest w [6], [5], [37], [24], [23]. Również w wymienionych opracowaniach czas nie został uwzględniony.

W rozważanych pracach do sprawdzenia DZ użyto następujących narzędzi: FDR [18], VERSA [46], PROMELA [37], narzędzia do CSP, NEPTUNE [10].

Próba wyrażenia dziedziczenia w terminach używanych w pierwszym podejściu w zastosowaniu do diagramów stanów natrafia na problemy interpretacyjne określić "osłabienie warunków początkowych" i "wzmocnienie warunków końcowych" [52], [53].

Innym sposobem podejścia do dziedziczenia zachowań w ramach diagramów stanów są próby określenia modyfikacji, które można dokonać w ramach diagramu stanów ma-

szyny klasy ogólnej, tak aby jej zachowanie zostało zachowane. Modyfikacje jakie zostały dopuszczone np. w [30] są dość oczywiste i obejmują tylko niektóre, najprostsze przypadki (np. łączenie dwóch maszyn w jedną z częściami równoległymi, pod warunkiem, że początkowe nie mają wspólnych zdarzeń i akcji). Dopiero ostatnio pojawiło się opracowanie umożliwiające, przy wybraniu odpowiedniej definicji dziedziczenia, dokonywanie mniej oczywistych modyfikacji w diagramie [60]. W artykułach tych również nie rozważa się czasu.

Autorowi pracy nie udało się znaleźć opracowań uwzględniających czas rzeczywisty w problemie DZ. Jedyne rozważania dotyczą czasu w postaci CTL (Computational Tree Logic) [10], jednak oznacza to, że czas nie jest rozważany w postaci ilościowej tylko w postaci jakościowej: "kiedyś", "zawsze" itp.

Problem dziedziczenia zachowań należy do gamy szeroko rozumianych problemów spójności modelu. Do badania spójności zaproponowano użycie języka LOTOS [4], OCL [10] z narzędziem RIDE [42], XML/XMI/DTD [32],[21] Object-Z [15],[49], RSDS [36].

Do tej grupy narzędzi należy UppAal, które jest powiązane z czasowymi automatami skończonymi oraz wspomaga rozwiązywanie problemu osiągalności w przestrzeni stanów. Używane automaty posiadają płaską strukturę, za to mają rozbudowane możliwości etykietowanych tranzycji. System ten wspomaga używanie całkowitoliczbowych zmiennych i zegarów, warunki wzbudzania tranzycji mogą być wyrażane zbiorem wyrażeń logicznych (porównanie zmiennych lub warunki czasowymi z użyciem zegarów) oraz ofertą (lub jej akceptacją) synchronizacji poprzez kanały. Część prac prezentujących użycie tego narzędzia do badania zgodności różnych diagramów UML [16, 34], sprawdzenia żywotności systemu itd. przekształca diagramy stanów UML bez części równoległych. Autorzy w [44] używając podobnego narzędzia o nazwie IF zapronowali automatyczny sposób przekształcania części równoległych tworząc dla każdej z nich osobny automat skończony, natomiast wspólne uruchomienie bądź zakończenie realizowane jest przez synchronizację kanałami. Przy automatycznych przekształceniach pojawia się problem tworzenia się niekontrolowanej liczby stanów i tranzycji. Prowadzić to może do sytuacji, że narzędzie nie może być skutecznie użyte [16] i dopiero po dodatkowej "ręcznej" optymalizacji osiąga się zadowalające efekty.

Oddzielną kwestią byłoby wykorzystanie czasowych sieci Petriego do badania przedstawionego w pracy problemu.

1.3 Cel pracy

Projektowanie systemu informatycznego odbywa się najczęściej w języku UML, zatem również kwestię dziedziczenia zachowań powinno się starać ujmować w notacji tego języka. Głównymi celami pracy są:

1. Opracowanie ogólnej definicji dziedziczenia zachowań czasowych. Definicja ta powinna uwzględniać wpływu czasu rzeczywistego na działanie maszyn stanowych.
2. Dokonanie wyboru kategorii maszyn stanów służących do reprezentowania zachowań i sprecyzowanie ich semantyki. Ponieważ notacja UML jest nadmiarowa, spośród elementów służących do reprezentacji diagramów stanów, wybrano tylko te,

które są niezbędne do rozważania problemu dziedziczenia zachowań. Norma UML bezpośrednio odnosi się do maszyn bezczasowych, wprowadzenie czasu wymaga rozszerzenia interpretacji zachowań maszyn stanowych.

3. Przyjęcie kryteriów porównawczych stosowanych przy określaniu dziedziczenia zachowań oraz opracowanie algorytmu stwierdzającego zachodzenie dziedziczenia zachowań dla wybranej kategorii diagramów stanów.

Dodatkowym celem było stworzenie programu implementującego opracowany algorytm, czyli sprawdzającego czy dwie zadane maszyny są w relacji dziedziczenia względem obranego środowiska, funkcji porównującej i zbioru zdarzeń obserwowalnych.

1.4 Zawartość pracy

Całość pracy podzielona została na 12 rozdziałów. Rozdział 2 wprowadza kwestię czasu i środowiska oraz definiuje pojęcie języka czasu rzeczywistego oraz sposób zapisanie zachowań środowiska i obiektu za pomocą tego języka. W rozdziale 3 zaproponowana zostaje definicja dziedziczenia zachowań czasowych oraz podział na różne rodzaje dziedziczeń. Następnie w rozdziale 4 znajduje się składnia i semantyka maszyny stanowej. Rozdział ten jest oparty na dokumentach standaryzacyjnych i zawiera te elementy języka UML, będą użyte w dalszej części pracy. W rozdziale 5 pojawia się opis wybranego modelu diagramu stanów. Po wprowadzeniu skwantowanego czasu model ten zostaje w rozdziale 6 ukonkretniony ze względu na możliwe różne interpretacje. Wykorzystanie tego modelu w celu opisu zachowań środowiska oraz obiektu jest opisane w rozdziałach, odpowiednio, 7 oraz 8, natomiast w rozdziale 9 wprowadzona zostaje klasa funkcji porównujących zachowania czasowe możliwe do stosowania w algorytmie. Sam algorytm prezentowany jest w rozdziale 10, po którym następuje rozdział 11 zawierający dowód poprawności algorytmu oraz oszacowanie jego złożoności. Tu również znajdują się przykładowe wyniki algorytmu oraz propozycja zastosowania prezentowanego rozwiązania dla diagramów stanów bezczasowych. Ostatecznie w rozdziale 12 znajdują się propozycje nowych problemów do rozwiązania.

Przy wprowadzaniu nowych oznaczeń przyjęta została następująca konwencja nazewnictwa: funkcje-konstruktory z dużej litery, natomiast oznaczenia i funkcje selekcji z małej litery.

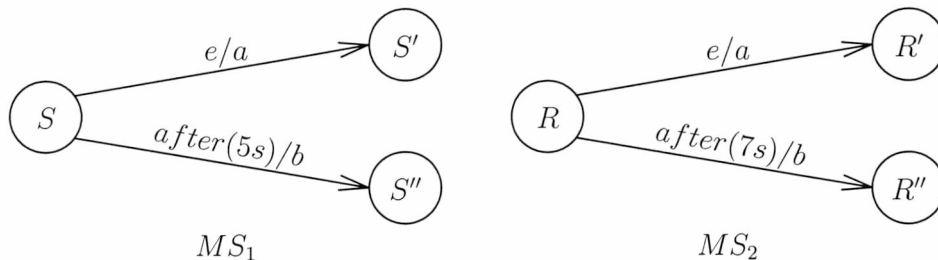
2 Język formalny etykietowany czasem

2.1 Kwestia czasu

Jak zostało wcześniej wspomniane podczas projektowania systemu należy wyodrębnić obiekty oraz ich cechy. Do opisu zachowania obiektu stosuje się zapis w postaci maszyny stanów, czyli stanów maszyny pomiędzy którymi jest możliwe przejście, jeśli istnieje połączenie (zwane tranzycją) z jednego stanu do drugiego. Dodatkowo, aby tranzycja mogła się wykonać, muszą być spełnione określone warunki przyporządkowane danej tranzycji.

Podczas wykonywania tranzycji uruchomiona może zostać jakaś akcja. Maszyna działa w pewnym otoczeniu, które może powodować spełnienie lub nie warunków zajścia tranzycji. Działania generowane przez środowisko nazywane są zdarzeniami, natomiast obserwowalne działania maszyny nazywane będą reakcjami. Jeśli w systemach czasu rzeczywistego w ramach tranzycji dopuszcza się możliwość uruchamiania ich upływem czasu (przeterminowaniem), to porównywanie zachowań obiektów nie powinno ograniczać się do badania następstw zdarzeń, ale uwzględniać również czas ich wystąpienia. Przykładowo dla maszyny stanowej z Rys (7) inne będzie zachowanie w stanie S , gdy po wejściu do tego stanu pojawi się zdarzenie e po upływie $4s$, a inne gdy to zdarzenie pojawi się po upływie $6s$. W pierwszym przypadku wygeneruje reakcję a oraz przejdzie do stanu S' , natomiast w drugim przypadku wygeneruje reakcję b oraz przejdzie do stanu S'' . Pozostaje pytanie, co się stanie ze zdarzeniem e w tym drugim przypadku. Można wyobrazić sobie dwie możliwości: zdarzenie zostanie obsłużone w przyszłości (oczywiście w stanie, w którym taka obsługa będzie w przyszłości możliwa) albo też zostanie pominięte (zignorowane). Pierwszą sytuację można rozwiązać na dwa sposoby. Pierwszy sposób to wprowadzić **komunikację synchroniczną między środowiskiem a maszyną stanów**, co oznacza że maszyna środowiska czeka w stanie, w którym chce wygenerować dane zdarzenie aż maszyna obiektu będzie w stanie umożliwiającym jego obsługę. Dopiero po tej sytuacji maszyna środowiska zmienia swój stan na kolejny. Innym sposobem jest wykorzystanie **kolejki zdarzeń odłożonych** w każdym stanie maszyny obiektu. Środowisko generuje zdarzenie i zmienia swój stan, a jeśli to zdarzenie nie może być natychmiast obsłużone przez obiekt, to zostaje odłożone do zbioru zdarzeń odroczonego.

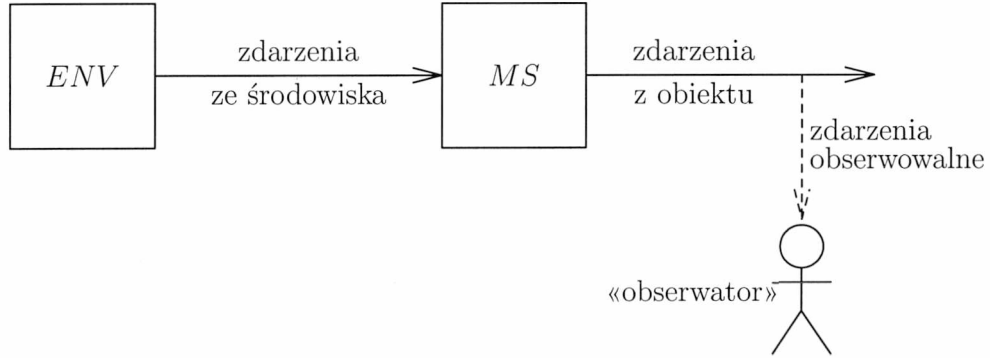
Autor pracy przyjął, że *komunikacja środowisko-obiekt* jest **asynchroniczna**, a zdarzenia wygenerowane przez środowisko, które obecnie nie mogą być obsłużone zostają pominięte przez obiekt.



Rysunek 7: Kwestia czasu pojawienia się zdarzenia

Jeśli wziąć pod uwagę, że obsługa zdarzenia, zmiana stanu, generowanie odpowiedzi itd. zajmuje pewien czas, to może się zdarzyć, że pewne zdarzenia zewnętrzne mogą nie zostać obsłużone, gdyż maszyna będzie zajęta. Oczywiście można przyjąć założenie, że wybieramy takie środowisko, które nie będzie generować nie obsłużonych zdarzeń. Można spróbować jednak potraktować **środowisko** jako dodatkowy **parametr** przy porównywaniu zachowań. Gdyby podchodzić do problemu dziedziczenia zachowań w sposób przedstawiony wcześniej, to wystarczy tylko skonstruować je tak, by spełniało założenie o nie "gubieniu" zdarzeń. Ponieważ istnieją problemy, w których środowisko

z założenia może generować nieskończony ciąg zdarzeń, zatem również sposób opisu środowiska musi dopuszczać taką możliwość. Dlatego dobrym, jako opis działania środowiska, wydaje się być **diagram stanów UML**, w którym przejścia między stanami mogą generować zdarzenia, natomiast każdy stan trwa pewien odcinek czasu. Dla dalszych rozważań obiekt oznaczany będzie jako maszyna stanowa obiektu (MS), natomiast środowisko jako maszyna stanowa środowiska (ENV).

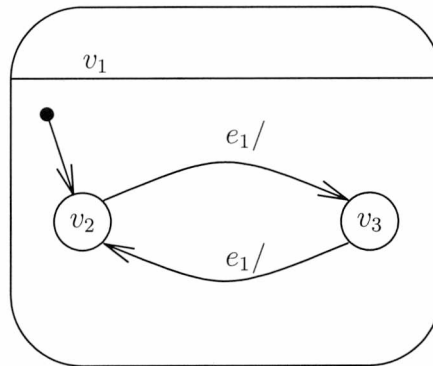


Rysunek 8: Maszyna środowiska a maszyna obiektu

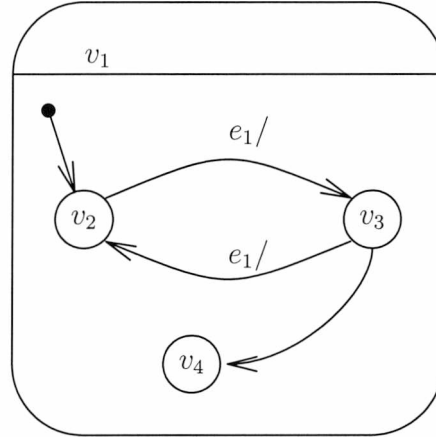
Maszyna obiektu podejmując akcję może wygenerować jakąś reakcję. Jednak porównując działania dwóch maszyn ze sobą nie wszystkie reakcje przez nie generowane mogą być istotne podczas porównania (Rys 8). Obserwator zewnętrzny może mieć percepcje obserwacji tylko pewnego ich podzbioru, który będzie nazywany *zdarzeniami obserwowalnymi*. Wszystkie pozostałe reakcje będą ignorowane przez obserwatora porównującego działania obydwóch maszyn.

2.2 Definicja języka formalnego etykietowanego czasem (JEC)

Należy zdefiniować formalny język opisujący pojawianie się zdarzeń z maszyny środowiska oraz reakcji maszyny obiektu na te zdarzenia. Nie wchodząc jeszcze w formalną specyfikację UML można zauważyć różnicę między poniższymi rysunkami (9) i (10).



Rysunek 9: Środowisko nieskończenie generujące zdarzenie e_1



Rysunek 10: Środowisko skończenie i nieskończenie generujące zdarzenie e_1

Rysunek (9) oznacza środowisko, które może wygenerować tylko nieskończony ciąg zdarzeń e_1 , natomiast rysunek (10) pokazuje środowisko, które albo będzie również w nieskończoność generowało taki sam ciąg zdarzeń jak pierwsze środowisko, albo nieparzystą ilość takich zdarzeń (po ich wygenerowaniu zatrzyma się). Na tym przykładzie widać, że język powinien dopuszczać istnienie ciągów skończonych jak i nieskończonych. Należy również zdefiniować odpowiednie operacje na takich ciągach.

W przypadku rozważania ciągów zdarzeń, gdy interesująca jest nie tylko kolejność występowania zdarzeń, ale również czas ich wystąpienia, pojawia się potrzeba zdefiniowania *języka formalnego etykietowanego czasem (JEC)* nad strukturą czasową $\langle Time, <, + \rangle$. Struktura ta składa się ze zbioru $Time$, który oznacza skwantowany, co najwyżej przeliczalny, zbiór chwil czasowych, relacji $<$ liniowego porządku, oraz operacji $+$ dodawania chwil czasowych.

Przez *alfabet* A rozumiemy pewien skończony zbiór symboli. *Literą* nazwiemy dowolny podzbiór tego alfabetu. *Literą czasową* nazwiemy parę $\langle A, t \rangle$ składającą się z litery i chwili czasowej.

W dalszych rozważaniach w celu uproszczenia notacji zapis ciągu (lub zbioru) z wielokropkiem oznaczać zarówno ciąg skończony jak i nieskończony. W zależności od rozważanego przypadku indeks dolny przy literach i czasach takich ciągów będzie rozważany w odpowiednim zakresie: dla ciągów skończonych od indeksu pierwszego elementu do indeksu ostatniego, w przypadku ciągu nieskończonego – od pierwszego do nieskończoności.

Należy wprowadzić pojęcie słowa

Definicja 1. *Słowem* jest dowolny ciąg skończony lub nieskończony liter czasowych takich, że czasy w kolejnych literach czasowych są większe od poprzednich:

$$\langle \langle A_1, t_1 \rangle, \langle A_2, t_2 \rangle, \dots \rangle \text{ jest słowem nad alfabetem } A \iff \forall_i ((A_i \subseteq A) \wedge (t_i < t_{i+1})) \quad (7)$$

Definicja 2. Przez zbiór $L_{A,Time}^\infty$ oznaczone są wszystkie skończone i nieskończone ciągi *liter czasowych*:

$$L_{A,Time}^\infty = \left\{ \langle \langle A_1, t_1 \rangle, \langle A_2, t_2 \rangle, \dots \rangle \mid A_1, A_2, \dots \in (2^A \setminus \emptyset) \wedge t_1, t_2, \dots \in Time \wedge t_1 < t_2 < \dots \right\} \quad (8)$$

Naturalnym wydawać by się mógł zapis par symbolu i czasu oraz dopuszczenie istnienia par o równym czasie. Jednak notacja $\langle \langle \{a, b, c\}, t \rangle \rangle$ w stosunku do notacji $\langle \langle a, t \rangle, \langle b, t \rangle, \langle c, t \rangle \rangle$ jest wygodniejsza przy porównywaniu ciągów. W ramach drugiej istnieją równoważne zapisy tej samej sytuacji np. $\langle \langle b, t \rangle, \langle c, t \rangle, \langle a, t \rangle \rangle$ co w kolejnych rozważaniach komplikuje porównywanie takich ciągów.

Teraz można już zdefiniować język formalny etykietowany czasem.

Definicja 3. *Językiem formalnym etykietowanym czasem* (JEC) jest dowolny podzbiór poprzednio zdefiniowanego zbioru z niepustymi zbiorami symboli. Czyli L_{CR} jest JEC, gdy:

$$L_{CR} \subseteq L_{A,Time}^\infty \wedge \forall_{W \in L_{CR}} \bullet \forall_{\langle A', t' \rangle \in W} \bullet (A' \neq \emptyset) \quad (9)$$

Ponieważ struktura czasowa będzie zawsze taka sama, więc dla skrócenia zapisu zamiast $L_{A,Time}^\infty$ będzie użyty symbol L_A^∞ .

Ponieważ struktura $Time$ jednoznacznie określa kolejność par w słowie, zamiast notacji ciągu $\langle \dots \rangle$ w wielu operacjach będzie używana notacja z rachunku zbiorów $\{ \dots \}$.

Oznaczenie 1. Konwersja notacyjna:

$$\{ \langle A_1, t_1 \rangle, \langle A_2, t_2 \rangle, \dots \} \in L \iff \langle \langle A_1, t_1 \rangle, \langle A_2, t_2 \rangle, \dots \rangle \in L \quad (10)$$

Zatem słowo W będzie traktowane jako zbiór liter czasowych. Stąd zapis $\langle a, t \rangle \in W$ oznacza istnienie takiej pary gdzieś w słowie. Ponieważ struktura czasowa $Time$ jest skwantowana, w dalszych rozważaniach zostanie przyjęte, że kwant czasu jest jednostką, zatem wartości czasu wyrażone są liczbami całkowitymi.

2.3 Zachowanie środowiska wyrażone za pomocą JEC

Zachowanie środowiska ENV polega na generowaniu zdarzeń w pewnych chwilach czasowych. Będzie ono określone przez parę: zbiór symboli oraz język formalny etykietowany czasem opartym na tym zbiorze. Zbiór symboli środowiska będzie nazywany zbiorem *zdarzeń*, które może generować dane środowisko. Litera czasowa będzie wówczas również nazywana *zdarzeniem czasowym*, a słowo należące do JEC środowiska *ciągami zdarzeń czasowych*.

Definicja 4. Środowisko ENV jest wyrażone przez parę:

$$ENV = \langle E, L \rangle$$

gdzie:

$$E - \text{zbiór zdarzeń środowiska}$$

$$L \subseteq L_E^\infty - \text{zbiór ciągów zdarzeń czasowych środowiska}$$
(11)

Nieskończony ciąg zdarzeń czasowych oznacza, że środowisko generuje pewne zdarzenia czasowe w nieskończoność, natomiast ciąg skończony oznacza, że po ostatnim zdarzeniu czasowym środowisko zatrzyma się i już nie wygeneruje żadnego zdarzenia czasowego. Stąd wniosek, że jeśli jakieś słowo należy do języka L , to nie wynika z tego, że pod słowo składające się z pewnej skończonej ilości początkowych zdarzeń czasowych rozważanego słowa musi również należeć do tego JEC.

Gdy nie będzie to powodowało niejednoznaczności zamiast zdarzenie czasowe będzie używane określenie *zdarzenie*, a zamiast ciągu zdarzeń czasowych - *ciąg zdarzeń*.

2.4 Zachowanie obiektu wyrażone za pomocą JEC

Rozważone teraz zostanie zachowanie obiektu. Na wejściu otrzymuje pewne zdarzenia czasowe, na wyjściu pojawiają się pewne zdarzenia będące reakcją na te zdarzenia i upływ czasu. Zatem zachowanie opisane będzie jako trójka: zbiór symboli wejściowych (dalej zwanych *zdarzeniami*, na które dana maszyna może reagować), zbiór symboli wyjściowych (dalej zwanych *reakcjami*, które maszyna może generować) oraz funkcji przekształcającej wejściowy JEC, oparty na pierwszym zbiorze w wyjściowy JEC, oparty na drugim zbiorze.

Definicja 5. Maszyna obiektu MS jest wyrażona przez trójkę:

$$MS = \langle E, R, f \rangle$$

gdzie:

$$E - \text{zbiór zdarzeń maszyny}$$

$$R - \text{zbiór reakcji maszyny}$$

$$f : L_E^\infty \rightarrow L_R^\infty - \text{funkcja całkowicie określona na zbiorze } L_E^\infty$$
(12)

Litera czasowa ze zbioru R nazywana będzie *reakcją czasową*, a ciąg takich liter *ciągiem reakcji czasowych*.

Funkcja f może jako argument otrzymać ciąg skończony lub nieskończony i jako wynik zwrócić także ciąg skończony lub nieskończony. Rozpatrując skończoność ciągu wejściowego i wyjściowego możliwe są cztery sytuacje. Jeśli funkcja f zwraca ciąg nieskończony dla skończonego ciągu wejściowego, to znaczy że mimo skończenia generowania zdarzeń przez środowisko, maszyna generuje reakcje spowodowane upływem czasu. Sytuacja, gdy dla wejściowego ciągu nieskończonego zwraca ciąg skończony oznacza, że maszyna kończy w pewnym momencie swoje działanie. Jeśli funkcja dla nieskończonego ciągu wejściowego zwraca nieskończony ciąg wyjściowy to niekoniecznie musi

oznaczać, że maszyna reaguje na generowane zdarzenia *explicite*. Tak naprawdę jej reakcje mogą wynikać nie ze zdarzeń, lecz z upływu czasu.

Gdy nie będzie to powodowało niejednoznaczności zamiast reakcja czasowa będzie używane określenie *reakcja*, a zamiast ciągu reakcji czasowych - *ciąg reakcji*.

3 Dziedziczenie zachowań czasowych

3.1 Składniki definicji

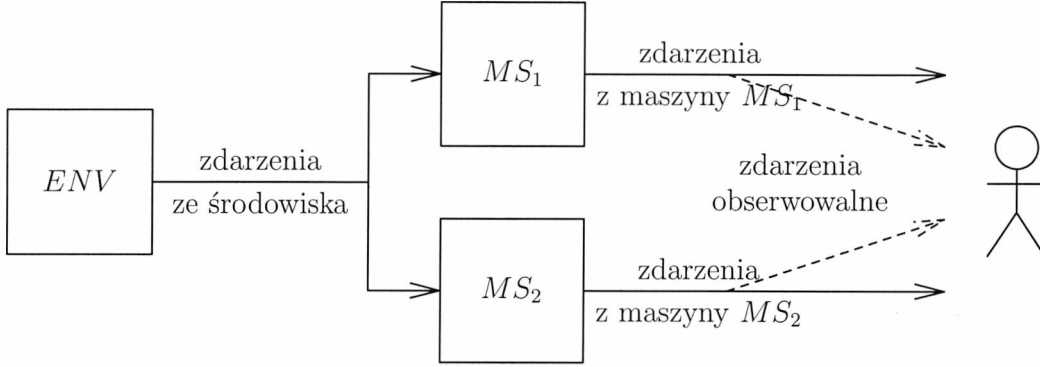
Rozważając dziedziczenie zachowań powinno się wyjść z postulatu Liskov mówiącego, że wstawienie obiektu specjalizowanego w miejsce pierwotnego nie powinno dla zewnętrznego obserwatora być widoczne. Postulat ten pozwala na różne podejścia do porównywania zachowań maszyn stanów. Restrykcyjne podejście polega na tym, że dana maszyna MS_2 dziedziczy zachowania innej maszyny MS_1 , jeśli wszystkie zdarzenia generowane przez maszynę MS_2 (*reakcje*) są **identyczne** do zdarzeń generowanych przez maszynę MS_1 . Identyfikacja ta musi dotyczyć wszystkich środowisk, czyli dla każdego wybranego środowiska reakcje maszyny MS_2 muszą być identyczne nie tylko **pod względem kolejności** ale i pod względem **czasu wystąpienia**. Mniej restrykcyjne podejście polega na tym, że identycznie mają się zachowywać tylko dla **pewnych środowisk** (kategorii środowisk), pewnych ciągów zdarzeń czasowych generowanych przez środowisko. Można również osłabić warunek dotyczący czasów wystąpienia reakcji. Można dopuścić pewne **odchylenia czasu wystąpienia reakcji** w maszynie MS_2 w stosunku do czasu wystąpienia tej reakcji w maszynie MS_1 .

Kolejność wystąpienia reakcji powinna być jednak zachowana, tak jak ma to miejsce w rozważaniach dziedziczenia zachowań, gdzie czas jest pomijany. Istotne jest dokładne zachodzenie następstw reakcji, tzn. jeśli w pierwszej maszynie następuje reakcja A, po pewnym czasie dwie reakcje B i C, a potem reakcja D, to zachowanie nie jest dziedziczone jeśli w innej maszynie występują reakcje A i B razem lub też najpierw B a po jakimś czasie C. Gdyby dopuścić taką możliwość, to rozpatrując trzy maszyny mogłaby nastąpić *inwersja* kolejności reakcji: w pierwszej mogłaby być kolejność A potem B, w drugiej A razem z B, w trzeciej najpierw B, potem A. Teoretycznie można by zezwalać na *łączenie się reakcji w grupy* w tym samym momencie czasowym, a zakazać rozdzielania ich, jednak wydaje się być to sztucznym podejściem i dalej się go nie rozpatruje. Kolejną kwestią są zdarzenia generowane przez maszyny, czyli reakcje. Obserwator może być zainteresowany tylko pewnym ich podzbiorem, a nie wszystkimi.

Przedyskutowane elementy zostały umieszczone na rys. (11). Niech:

$$\begin{aligned} ENV &= \langle E, L \rangle \\ MS_1 &= \langle E_1, R_1, f_1 \rangle \\ MS_2 &= \langle E_2, R_2, f_2 \rangle \\ ZO &- \text{zbiór zdarzeń obserwowalnych} \end{aligned} \tag{13}$$

Pozostaje jeszcze kwestia porównywania czasów reakcji porównywanych maszyn. W tym celu wprowadzone zostanie pojęcie funkcji porównującej dwa słowa z JCR. Jest



Rysunek 11: Porównanie działania dwóch obiektów

to funkcja logiczna, która zostanie oznaczona jako τ .

$$\tau : L_{ZO}^\infty \times L_{ZO}^\infty \rightarrow Bool \quad (14)$$

Funkcja τ musi zapewniać *zgodność strukturalną* polegającą na tym, że wymaga pary ciągów równej długości oraz tej samej kolejności reakcji w obydwóch ciągach.

Definicja 6. Niech $W_1 = \langle \langle A_1, t_1 \rangle, \langle A_2, t_2 \rangle, \dots \rangle$ i $W_2 = \langle \langle A'_1, t'_1 \rangle, \langle A'_2, t'_2 \rangle, \dots \rangle$. Zgodność strukturalna funkcji τ oznacza spełnienie dwóch warunków:

1. Dla ciągów o różnej długości wartością funkcji τ jest fałsz.

$$|W_1| \neq |W_2| \Rightarrow \neg \tau(W_1, W_2) \quad (15)$$

2. Kolejne zbiory zdarzeń w porównywanych słowach muszą być identyczne.

$$\exists_i \bullet (A_i \neq A'_i) \Rightarrow \neg \tau(W_1, W_2) \quad (16)$$

Oczywiście warunek zgodności strukturalnej odzwierciedla tylko podstawowy warunek dziedziczenia dla maszyn bezczasowych. Klasa funkcji spełniających ten warunek jest bardzo szeroka. W przypadku rozważania systemów czasu rzeczywistego funkcja powinna dodatkowo brać pod uwagę momenty reakcji czasowych. W rozdziale 9 nałożone zostaną dodatkowe warunki, jakie musi spełniać funkcja, aby można algorytmicznie wyliczać jej wartości.

3.2 Definicja dziedziczenia zachowań wyrażona poprzez JEC

Operując na słowie istotne mogą być tylko pewne litery alfabetu, stąd potrzeba obcięcia słowa do tych wybranych liter.

Definicja 7. Zapis $W|_X$ oznacza *obcięcie słowa W do zbioru X* , czyli powstanie nowego słowa poprzez zmodyfikowanie w słowie W elementów zbioru a tak, aby $a' = a \cap X$.

$$W|_X = \left\{ \langle a', t \rangle \mid \bigvee_{\langle a, t \rangle \in W} \bullet \left((a' = a \cap X) \wedge (a' \neq \emptyset) \right) \right\} \quad (17)$$

Definicja 8. Zapis $L|_X$ oznacza *obcięcie języka L do zbioru X* , czyli nowy język składający się ze wszystkich słów języka L obciętych do zbioru X .

$$L|_X = \left\{ W' \mid \bigvee_{W \in L} \bullet (W' = W|_X) \right\} \quad (18)$$

Definicja dziedziczenia zachowań przedstawia się następująco:

Definicja 9. Dziedziczenie przez maszynę $MS_2 = \langle E_2, R_2, f_2 \rangle$ zachowań maszyny $MS_1 = \langle E_1, R_1, f_1 \rangle$ w środowisku $ENV = \langle E, L \rangle$, przy funkcji porównującej τ i zbiorze zdarzeń obserwowalnych $ZO \subseteq R_1$ (co zapisane będzie jako $MS_1 \triangleleft^{\tau, ENV, ZO} MS_2$) zachodzi wtedy, gdy spełniony jest następujący warunek:

$$MS_1 \triangleleft^{\tau, ENV, ZO} MS_2 \iff \bigvee_{w \in L} \bullet \left((r_1 = f_1(w|_{E_1}) \wedge r_2 = f_2(w|_{E_2})) \Rightarrow \tau(r_1|_{ZO}, r_2|_{ZO}) \right) \quad (19)$$

Warunek ten oznacza, że każdy ciąg zdarzeń ze środowiska zostaje obcięty do, odpowiednio dla maszyn, zbiorów E_1 i E_2 . Tak zmodyfikowany ciąg jest interpretowany przez każdą z maszyn, które generują odpowiednio ciągi reakcji r_1 i r_2 . Obserwator zewnętrzny postrzegający tylko w zakresie zbioru obserwowalnego ZO (zbiór ten jest podzbiorem zbioru reakcji maszyny, której zachowanie ma być dziedziczone), porównuje te ciągi reakcji funkcją τ . Jeśli funkcja jest spełniona, to maszyna MS_2 dziedziczy zachowanie MS_1 .

Definicja ta jest ogólną definicją przyjętą przez autora. Dla ogólnego przypadku dowolnych środowisk, maszyn obiektów i funkcji porównującej problem DZ wydaje się być nierozwiązywalny. Stąd proponuje się przyjąć pewne ograniczenia, a w ramach tych ograniczeń proponuje się algorytm badania zgodności. Do opisu zachowań środowiska i obiektu proponuje się użyć pewne klasy diagramów stanów UML oraz nakłada się na funkcję porównującą pewne ograniczenia. Wszystkie te elementy zostaną opisane w kolejnych rozdziałach.

3.3 Rodzaje dziedziczenia zachowań

W poprzednim podrozdziale wprowadzone zostało pojęcie dziedziczenia zachowań. Rozważając maszyny stanowe dwóch obiektów dziedziczenie zachowań zależy od trzech czynników: środowiska, funkcji porównującej i zbioru zdarzeń obserwowalnych. Rozpatrując te elementy dziedziczenie można by podzielić dziedziczenie zachowań na następujące rodzaje:

1. Dziedziczenie zachowań ścisłe

$$MS_1 \triangleleft -MS_2 \quad (20)$$

Maszyna MS_2 zachowuje się dokładnie tak samo jak maszyna MS_1 , czyli dla dowolnego ciągu zdarzeń czasowych generuje taki sam ciąg reakcji czasowych.

2. Dziedziczenie zachowań z tolerancją czasową

$$MS_1 \triangleleft^\tau -MS_2 \quad (21)$$

Zachowanie maszyny MS_2 dla tych samych ciągów zdarzeń jest takie samo jak maszyny MS_1 co do kolejności generowanych reakcji, ale ich punkty czasowe mogą się różnić. Możliwe różnice opisuje czasowa funkcja porównująca τ .

3. Dziedziczenie zachowań względem środowiska

$$MS_1 \triangleleft^{ENV} -MS_2 \quad (22)$$

Zachowanie maszyn jest dokładne co do kolejności i czasu, ale jest rozpatrywane tylko dla ciągów zdarzeń, które są generowane przez środowisko ENV .

4. Dziedziczenie wybranego zbioru reakcji

$$MS_1 \triangleleft^{ZO} -MS_2 \quad (23)$$

Przyjmuje się, że jeśli na strzałce nie ma określonej funkcji czasowej to znaczy, że punkty czasowe odpowiadających zbiorów reakcji muszą być identyczne, a jeśli nie ma identyfikatora środowiska, to dziedziczenie ma zachodzić dla wszystkich środowisk.

Przypadek (1) dziedziczenia zachowań jest możliwy tylko dla niewiele różniących się maszyn. Przypadki (1) i (2) dla środowisk uniwersalnych wydają się mieć niewielkie zastosowanie praktyczne, gdyż dodanie w drugiej maszynie nowej tranzycji z reakcją nie istniejącą wśród reakcji maszyny pierwszej spowoduje nie dziedziczenie zachowania. Dodanie środowiska w rozważaniach na temat zgodności zachowań umożliwia analizowanie zachowania obiektu specjalizowanego tylko dla wybranych ciągów zdarzeń, środowisko to może być wręcz skonstruowane na podstawie zachowania obiektu pierwotnego.

4 Maszyna stanowa w UML

W tym rozdziale przedstawiona zostanie w skrócie składnia diagramu stanów języka UML i semantyka związanej z nim maszyny stanowej tego języka. W odniesieniu do kolejnych elementów diagramu stanów zobrazowana zostanie ich obecność lub brak w modelu używanym w tej pracy. Semantyka przedstawiona będzie na podstawie specyfikacji "OMG – Unified Modeling Language, v1.5" na stronach od 2-140 do 2-169 [45] (podobny opis znajduje się w [50]).

Używając diagramów UML należy zauważyć, że ich notacja jest nadmiarowa [51], oraz że pozwala tworzenie niestukturalnych diagramów (w których tranzycje przekraczają granice stanów), co utrudnia zrozumienie ich semantyki [6]. Stąd wielu autorów używa tylko wybranych elementów diagramów stanów oraz rozważa tylko diagramy strukturalne. Wprowadzenie tego założenia nie ogranicza możliwości zapisywania zachowań z użyciem diagramów stanów, a dodatkowo wymusza na twórcy diagramu zharmonizowane podejście.

W wielu opracowaniach, w celu uproszczenia notacji i interpretacji, używany jest płaski model, bez części równoległych [60], [37], [18] jednak przekształcenie modelu pełnego na płaski może spowodować "eksplozję" liczby stanów i tranzycji [3]. Wprowadzenie czasu byłoby dodatkowym czynnikiem zwielokrotniającym te liczby.

Ponieważ części równoległe są bardzo często używane w pracy założono, że maszyna może posiadać części równoległe oraz hierarchię stanów, jednak rozważane będą tylko diagramy strukturalne.

4.1 Składnia diagramów stanów w UML

Na początku ogólny opis maszyn stanów (fragmenty cytowań z normy UML posiadają poszerzony lewy i prawy margines) :

Pakiet maszyny stanowej (**ang.** *State Machine*) jest podpakietem pakietu Elementów Behawioralnych (**ang.** *Behavioral Elements*). Specyfikuje zbiór pojęć, które mogą być używane do modelowania zachowania dyskretnego poprzez systemy skończenie stanowe. Pojęcia te bazują na pojęciach zdefiniowanych w pakiecie Podstawa (**ang.** *Foundation*) jak i na pojęciach z pakietu Wspólne Zachowania (**ang.** *Common Behavior*). Umożliwia to integrację z innymi podpakietami w ramach Elementów Behawioralnych.

Maszyn stanów opisane w tym rozdziale są obiektowo zorientowanym wariantem diagramów stanów (**ang.** *statecharts*) Harela. Opis zawiera kilka pojęć podobnych do tych zdefiniowanych w ROOMcharts, wariacie diagramów stanów zdefiniowanych w języku modelowania ROOM. Główne różnice w stosunku do klasycznych diagramów stanów Harela są opisane na stronie 2.12.5.42–169.

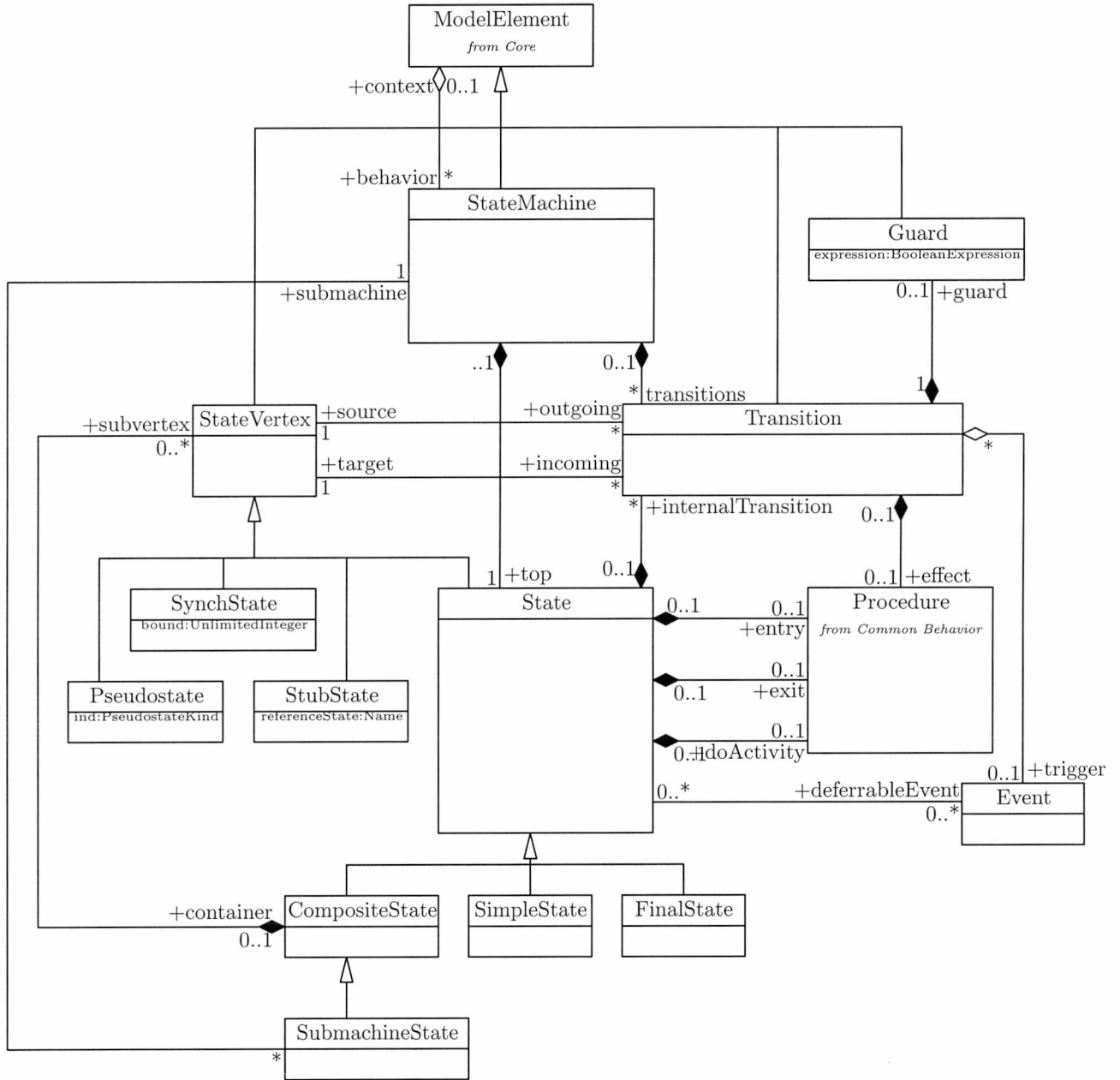
Maszyny stanów mogą być używane do specyfikacji zachowań różnorodnych modeli. Na przykład mogą być użyte do modelowania zachowań jednostkowych (np. instancji klas) lub do definiowania interakcji (kolaboracji) międzyjednostkowych.

Dodatkowo, sformułowania używane do opisu maszyn stanów dostarczają podstawy semantycznej dla diagramów aktywności. To znaczy, że diagramy aktywności są po prostu specjalnym rodzajem maszyn stanów.

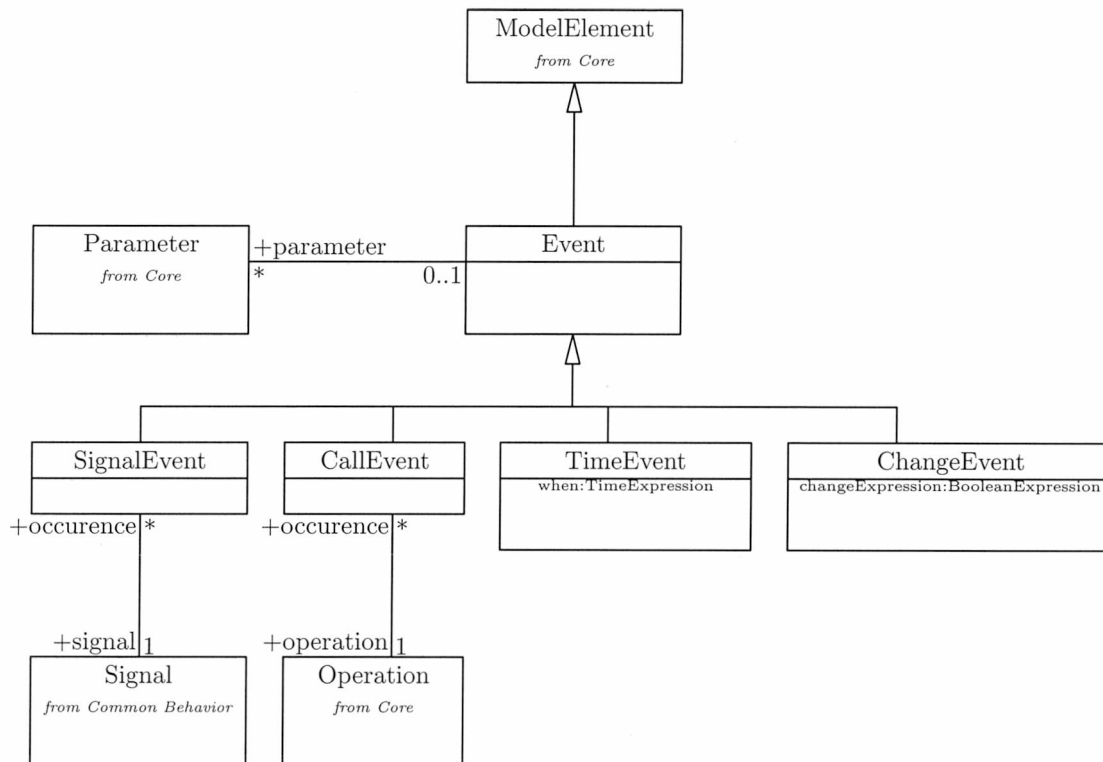
Abstrakcyjna składnia maszyn stanowych jest wyrażona graficznie na rysunku (12), pokrywając wszystkie podstawowe pojęcia diagramów maszyn stanów, takich jak stany

(**ang.** *state*) i tranzycje (**ang.** *transition*). Rysunek (13) opisuje abstracyjną składnię zdarzeń (**ang.** *event*), które mogą wzbudzać zachowanie maszyn stanów.

W pracy, w wybranej kategorii maszyn stanów używanych do modelowania zachowań, nie wszystkie elementy metamodelu UML będą używane. Diagram metaklas, po usunięciu nieużywanych elementów, znajduje się na rys. (14) i (15).



Rysunek 12: Metamodel maszyny stanów (State Machines - Main)



Rysunek 13: Metamodel zdarzeń (State Machines - Events)

4.2 Semantyka maszyny stanowej w UML

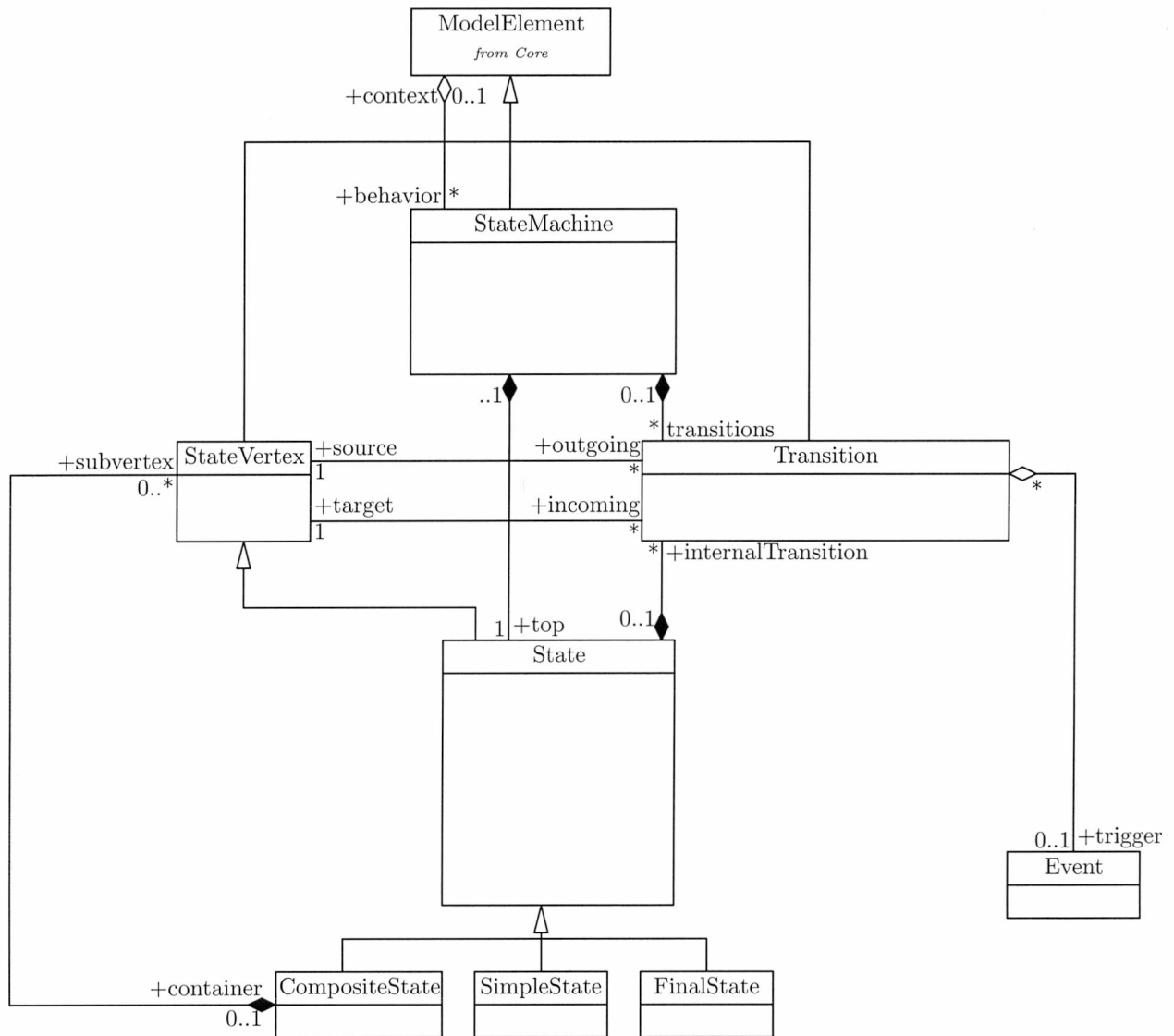
Szczegółowa semantyka, opisana za pomocą hipotetycznej maszyny w obrazowy sposób wyjaśnia znaczenie elementów maszyn stanów języka UML:

2.12.4 Szczegółowa semantyka (*ang.* Detailed Semantics)

Ta sekcja opisuje semantykę działania maszyn stanowych. Dla ułatwienia semantyka jest opisywana w terminach operacji hipotetycznej maszyny, która implementuje specyfikacje maszyny stanowej. (...)

W ogólnym przypadku kluczowymi komponentami hipotetycznej maszyny są:

- kolejka zdarzeń (*ang.* *event queue*), która przechowuje instancje napływających zdarzeń do czasu ich przetworzenia,
- mechanizm rozdzielania zdarzeń (*ang.* *event dispatcher mechanism*), który wybiera i wyjmuje z kolejki instancje zdarzeń do przetworzenia,
- procesor zdarzeń (*ang.* *event processor*), który przetwarza wybrane instancje zdarzeń stosownie do ogólnej semantyki maszyn stanów UML i specyficznej formy rozważanej maszyny stanów. Ten komponent będzie w dalszych rozważaniach uważany za właściwą "maszynę stanów".



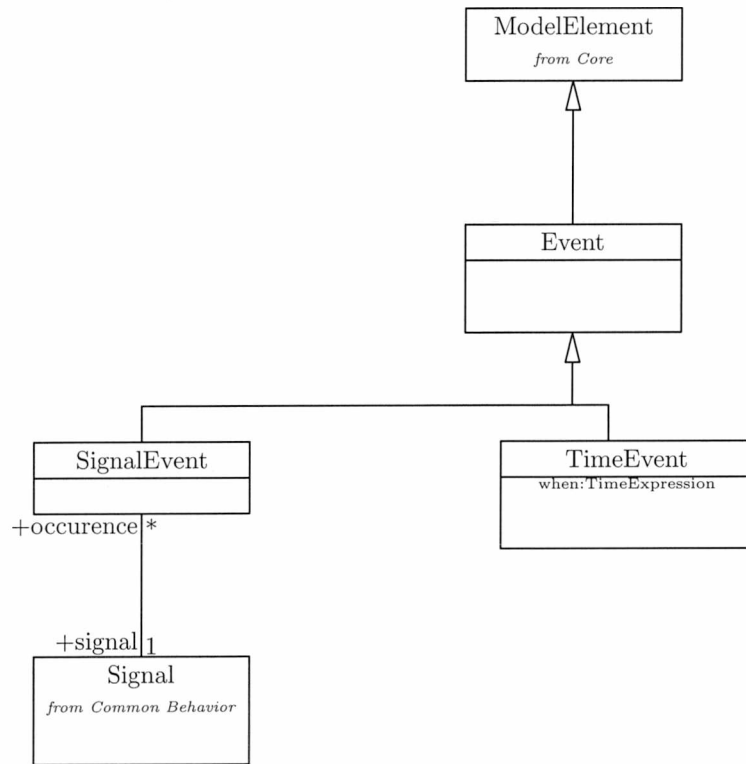
Rysunek 14: Metamodel maszyny stanowej używany w pracy

W pracy kolejną zdarzeń będzie opis środowiska. Mechanizm rozdzielania zdarzeń jest bardzo prosty, gdyż polega na pobieraniu ze środowiska instancji zdarzeń w momencie, w którym napłyną i w tym samym momencie przesyłaniu ich do procesora zdarzeń, czyli maszyny stanowej.

W specyfikacji UML występuje uwaga, że mimo intencji bardzo precyzyjnego określenia semantyki, wiele jej punktów dopuszcza różne interpretacje w zależności od zastosowań, a punkty te będą jasno identyfikowane w tekście. Następuje opis kolejnych elementów:

2.12.4.1 Zdarzenie

Zdarzenie (**ang.** *event*) jest generowane jako wynik pewnej akcji albo



Rysunek 15: Używane zdarzenia

wewnątrz systemu albo w środowisku otaczającym system. Zdarzenie jest wówczas dostarczane do jednego lub więcej miejsc docelowych. Przebieg transportu jest zależny typu akcji, miejsca docelowego, właściwości medium komunikacyjnego i wielu innych czynników. W pewnych przypadkach jest praktycznie natychmiastowy i niezawodny, gdy w innych może powodować różne opóźnienia transmisji, gubienie zdarzeń, zmiany uporządkowania czy powielenia. Nie rozważa się żadnych założeń na tym etapie. Dostarcza to pełnej elastyczności dla modelowania różnych typów ułatwień komunikacyjnych.

Zdarzenie jest odebrane (**ang.** *received*), gdy jest umiejscowione w kolejce swojego miejsca docelowego. Zdarzenie jest rozdzielone (**ang.** *dispatched*) kiedy jest wyjęte z kolejki zdarzeń i dostarczone do maszyny stanów do przetwarzania. W tym momencie nazywane jest ono bieżącym zdarzeniem (**ang.** *current event*). Ostatecznie jest skonsumowane (**ang.** *consumed*), gdy przetwarzanie zdarzenia jest zakończone. Skonsumowane zdarzenie nie jest już więcej dostępne do przetwarzania. Nie ma żadnych założeń co do odstępów czasowych między odebraniem, dostarczeniem i przetworzeniem.

W przyjętym w pracy modelu zdarzenia są generowane przez środowisko, a odebranie zdarzenia i jego dostarczenie dokonuje się w tej samej chwili. Skonsumowanie zdarzenia

dokonywane w najbliższym możliwym momencie. Różnica czasu między dostarczeniem a skonsumowaniem zdarzenia zostanie wyjaśniona później przy opisie uruchamiania tranzycji.

2.12.4.1 Stan (*ang. state*)

Stany aktywne (*ang. active states*)

Stan może być aktywny lub nieaktywny podczas uruchomienia. Stan staje się aktywny (*ang. active*) kiedy następuje wejście do niego jako wynik wykonania pewnej tranzycji, a staje się nieaktywny (*ang. inactive*), jeśli zostaje opuszczony w wyniku odpowiedniej tranzycji. Stan może być opuszczony i odwiedzony w wyniku tej samej tranzycji.

Wejście i wyjście ze stanu (*ang. state entry and exit*)

Kiedy stan jest odwiedzany, najpierw wykonywana jest akcja wejściowa tego stanu, a dopiero później jakakolwiek inna akcja. Konsekwentnie, kiedy stan jest opuszczany, wykonanie akcji wyjściowej jest ostatnią aktywnością dokonywaną w ramach tego stanu.

Jeśli inne aktywności w stanie są zdefiniowane, uruchamiane są one równolegle tuż po zakończeniu akcji wejściowej. Przy wychodzeniu aktywności te muszą zostać zakończone zanim zostanie wykonana akcja wyjściowa.

Rozumienie stanów aktywnych i nieaktywnych jest zachowane w przyjętym modelu. Zostały również wprowadzone akcje wejściowa i wyjściowa. Ich kolejność w stosunku do innych aktywności jest zgodna z powyższym opisem. Z akcjami tymi związany jest czas potrzebny na ich wykonanie.

Aktywność w stanie (*ang. Activity in state (do-activity)*)

Aktywność reprezentuje sekwencję wykonania akcji, która następuje kiedy maszyna stanowa jest w aktywnym stanie. Aktywność rozpoczyna się po wejściu do stanu, tuż po akcji wejściowej. Jeśli się zakończy gdy stan stale jest aktywny, powoduje powstanie zdarzenia ukończenia. Jeśli istnieje wychodząca tranzycja ukończenia (...), stan zostanie opuszczony. Jeśli stan zostanie opuszczony z powodu wzbudzenia tranzycji zanim ukończy się jego aktywność, zostanie ona przerwana przed jej zakończeniem.

Aktywności w stanie nie są w przyjętym modelu rozpatrywane. Gdyby założyć ich stały czas wykonania, to zmiany w notacji będą niewielkie, powodujące jednak niepotrzebne zwiększenie ilości pojęć i funkcji. Należało zauważyć, że w przypadku stanu złożonego trzeba by doprecyzować, czy przerwanie aktywności odbywa się w chwili wzbudzenia tranzycji, czy też w chwili jej uruchomienia.

Zdarzenia odroczone (*ang. deferred events*)

Stan może specyfikować zbiór typów zdarzeń, które mogą być odroczone w tym stanie. Zdarzenie, którego instancja nie wzbudza żadnej tranzycji w aktualnym stanie, nie będzie dostarczone jeśli jego typ zgadza się z jednym z typów zbioru zdarzeń odroczonego w danym stanie. Zamiast tego, zostanie ono w kolejce zdarzeń, z której będzie pobrane inne nieodroczone zdarzenie. Sytuacja taka będzie trwać, aż osiągnięty zostanie stan, w którym zdarzenie nie będzie dalej odroczone lub wzbudzi tranzycję w nowym stanie.

Kolejka zdarzeń odroczonego nie została zawarta w modelu. Przy rozpatrywaniu konfiguracji maszyny stanowej taka kolejka, w ogólnym przypadku dopuszczalna dla każdego stanu, powodowałaby dodatkowy element notacyjny.

Rozpatrując jednak taką sytuację należałoby się zastanowić nad pewnymi uściśleniami. Jak traktować sytuację, gdy w pewnym odstępie czasu nadejdzie instancja tego samego zdarzenia. Jeśli wtedy w kolejce będą dwie instancje zdarzenia, to znaczy że przy wejściu do stanu, w którym to zdarzenie wzbudza tranzycję, usunięta z kolejki będzie tylko jedna instancja. Druga możliwość to utożsamianie tych zdarzeń, czyli pamiętanie w kolejce tylko jednej instancji.

2.12.4.3 Stan złożony (*ang.* *CompositeState*)

Konfiguracje stanu aktywnego (*ang.* *active states configurations*)

Kiedy rozważamy stany złożone i równoległe, proste określenie "aktualny stan" może być niejasne. W hierarchicznej maszynie stanów więcej niż jeden stan może być w tej samej chwili aktywny. Jeśli maszyna stanów jest w prostym stanie, który jest zawarty w stanie złożonym, to wszystkie stany złożone zawierające bezpośrednio lub pośrednio ten prosty stan, są również aktywne. Co więcej, jeśli pewne z tych stanów złożonych w hierarchii mogą być równoległe, aktualny aktywny "stan" reprezentowany jest przez drzewo stanów, poczynając od pojedynczego najwyższego stanu w korzeniu, w dół do wszystkich prostych stanów będących liśćmi tego drzewa. Takie drzewo stanów będzie nazywane konfiguracją stanów.

Pomiędzy wykonywaniem tranzycji, następujące niezmienniki zawsze są spełnione w konfiguracji stanów:

- Jeśli stan złożony sekwencyjnie jest aktywny, to dokładnie jeden z jego podstanów jest aktywny.
- Jeśli stan złożony równoległe jest aktywny, to wszystkie jego podstany (regiony) są aktywne.

Chociaż nie jest to wprost napisane, to w niezmiennikach chodzi o podstany bezpośrednio rozważanego stanu aktywnego.

W przyjętym modelu znaczenie konfiguracji pozostaje zgodne z normą UML. Reprezentowana jest jednak w postaci zbioru stanów, a nie ich drzewa, gdyż kwestie, który stan jest podstanem innego są zapisane w modelu maszyny stanowej. Dodatkowo dla

każdego stanu zapamiętywany jest również pewien jego czas. W przypadku konfiguracji środowiska – czas pozostały do opuszczenia stanu, w przypadku maszyny obiektu – czas upływający od wejścia do stanu. W przyjętej reprezentacji konfiguracji maszyny stanowej pomijane są również stany wewnętrzne, w których zakończyła się już akcja wyjściowa.

Wchodzenie do stanu złożonego sekwencyjnie (ang. *entering a non-concurrent composite state*)

Przy wchodzeniu do stanu złożonego sekwencyjnie, rozróżniamy następujące przypadki:

- Wejście domyślne (**ang.** *default entry*): Graficznie jest pokazane poprzez przychodzącą tranzycję, która kończy się na zewnętrznej krawędzi stanu złożonego. W tym przypadku, domyślna tranzycja jest wzbudzana. Jeśli jest jakiś dozór na takiej tranzycji, to musi być spełniony (...). Akcja wyjściowa rozważanego stanu zostanie wykonana zanim wykona się akcja związana z tą tranzycją inicjującą.
- Wejście bezpośrednie (**ang.** *Explicit entry*): Jeśli tranzycja kończy się w podstanie stanu złożonego, podstan staje się aktywny, a jego akcja wyjściowa wykonuje się tuż po wykonaniu akcji wyjściowej stanu złożonego. Zasada ta działa rekursywnie, jeśli tranzycja kończy się w stanie wielokrotnie zagnieżdżonym w innych stanach.
- Wejście do stanu płytkiego wznowienia (**ang.** *shallow history entry*): (...)
- Wejście do stanu głębokiego wznowienia (**ang.** *deep history entry*): (...)

W proponowanym modelu w przypadku stanu złożonego istnieje tylko możliwość wejścia domyślnego. Wejście bezpośrednie, przekraczające krawędź stanu złożonego powoduje skomplikowanie wszelkich notacji i większość prac tego typu przejść nie dopuszcza. Przejścia takie są też najczęściej wyrazem braku strukturalności przy tworzeniu maszyny stanowej, co wydaje się być w sprzeczności z paradygmatami projektowania obiektowego.

Pozostałe elementy (stany wznowienia) nie zostały zawarte w modelu, jednak dodanie ich nie powinno stanowić żadnego problemu. W konfiguracji stanów należałoby wówczas dla stanów złożonych (posiadających stany wznowienia) pamiętać dodatkowy zbiór. Zbiór ten zawierałby stany, do których należy wejść przy wchodzeniu do stanu wznowienia.

Wchodzenie do stanu złożonego równolegle (ang. *entering a concurrent composite state*)

Jeśli stan złożony równolegle jest odwiedzany, każdy z jego podstanów (regionów) jest odwiedzany, albo domyślnie, albo bezpośrednio.

W przypadku stanu złożonego równolegle konsekwentnie dopuszczalne jest tylko wejście domyślne, aby zachować postulat strukturalności.

Opuszczanie stanu złożonego sekwencyjnie (ang. *exiting a non-concurrent state*)

Jeśli opuszczany jest stan złożony, wszystkie jego podstany są opuszczane rekursywnie. Oznacza to uruchomienie sekwencji akcji wyjściowych poczynając od stanu najbardziej wewnętrznego.

Opuszczanie stanu złożonego równolegle (ang. *exiting a concurrent state*)

Przy opuszczaniu stanu równoległego, każdy jego region jest opuszczany. Następuje wykonanie akcji wyjściowej w każdym regionie, a następnie akcji wyjściowej rozważanego stanu równoległego.

Powyższe zachowanie jest utrzymane w modelu. Z powodu wprowadzenia czasu, w przypadku stanu złożonego sekwencyjnie, zanim nastąpi wyjście z niego, musi upłynąć czas wykonania akcji wyjściowych jego podstanów, a następnie jego własnej akcji wyjściowej. W przypadku stanu złożonego równolegle, wyjścia z jego regionów są niezależne od siebie, zatem jego akcja wyjściowa rozpocznie się po zakończeniu wychodzenia z regionu, który najpóźniej został opuszczony.

2.12.4.4 Stan końcowy (ang. *final state*)

Jeśli stan końcowy jest aktywny, zawierający go stan złożony jest ukończony, to znaczy że spełnia warunki ukończenia. Jeśli ten stan jest stanem głównym maszyny, to cała maszyna się zatrzymuje.

Stany końcowe zostały w modelu zaimplementowane. Ponieważ są to stany specjalizowane, zakłada się że nie posiadają akcji wejściowych i wyjściowych.

2.12.4.5 Stan podmaszyny (ang. *SubmachineState*)

Stan podmaszyny są udogodnieniem, które nie wprowadza żadnej dodatkowej semantyki dynamicznej. Semantycznie jest równoważny stanowi złożonemu (...).

Stany podmaszyny nie są używane, gdyż nie wprowadzają nowej jakości.

2.12.4.6 Tranzycje (ang. *Transitions*)

Tranzycje wysokiego poziomu (ang. *High-level transitions*)

Tranzycje zapoczątkowane na granicy stanu złożonego są nazywane tranzycjami wysokiego poziomu lub grupowymi tranzycjami. Wzbudzone,

powodują wyjście ze wszystkich podstanów tego stanu złożonego. (...)

Tranzycje złożone (ang. *compound transitions*)

Tranzycja złożona jest pojęciem semantycznie wywiedzionym, reprezentującym "semantycznie kompletną" ścieżkę składającą się z jednej lub więcej tranzycji, rozpoczynających się ze zbioru stanów (...) i kończącej się w zbiorze stanów. (...)

Tranzycje wewnętrzne (ang. *Internal transitions*)

Wykonanie tranzycji wewnętrznej nie pociąga za sobą wykonania akcji wejściowej ani wyjściowej stanu w którym została zdefiniowana. Zachowanie to dotyczy również stanów zagnieżdżonych.

Tranzycje ukończenia i zdarzenia ukończenia (ang. *Completion transitions and completion events*)

Tranzycja ukończenia jest tranzycją bez bezpośredniego wyzwalacza (ang. *trigger*), chociaż może mieć zdefiniowany dozór. Kiedy wszystkie tranzycje i akcje wejściowe i aktywności w bieżącym stanie są zakończone, jest generowana instancja zdarzenia ukończenia. To zdarzenie jest pośrednim wyzwalaczem dla tranzycji ukończenia. Zdarzenie ukończenia jest przetwarzane przed jakimkolwiek innym kolejkowym zdarzeniem i nie posiada związanych z nim parametrów.

Ze względu na strukturalność maszyny stanowej, w proponowanym modelu istnieją tylko tranzycje wysokiego poziomu. Nie zostały wprowadzone tranzycje złożone, gdyż są rzadko używane.

Tranzycje wewnętrzne nie zostały wprowadzone, jednak wydają się być łatwe do wprowadzenia do modelu (ich działanie jest zbliżone do obecności sekcji równoległej w stanie),

Tranzycje ukończenia są wprowadzone w modelu i używane przy opisie środowiska, natomiast w maszynach porównywanych nie został ten typ tranzycji wprowadzony. Ich wprowadzenie do modelu opisu maszyny obiektów spowodowałoby potrzebę pamiętania w konfiguracji stanów czasu do zakończenia akcji wyjściowej ze stanu.

Tranzycje dozwolona (ang. *Enabled (compound) transitions*)

Tranzycja jest dozwolona do wykonania (krótko dozwolona), wtedy i tylko wtedy, gdy:

1. Wszystkie jej stany źródłowe są w konfiguracji aktywnych stanów.
2. Wyzwalacz odpowiada bieżącemu zdarzeniu. (...)
3. (...)

Zacytowane zasady, kiedy tranzycja jest dozwolona, są zachowane w proponowanym modelu. Trzeci warunek nie został zacytowany, gdyż odnosi się do nie używanych tranzycji złożonych.

Dozory (ang. *Guards*)

W prostej tranzycji z dozorem jest on obliczany zanim tranzycja jest wzbudzana.

W przyjętym modelu brak jest dozorów. Wprowadzenie dozorów związane jest z wprowadzeniem algebry do ich zapisania. Algebra taka komplikuje porównywanie zachowań.

Sekwencja wykonania tranzycji (ang. *Transition execution sequence*)

Każda tranzycja, oprócz tranzycji wewnętrznych, powoduje wyjście ze stanu źródłowego i wejście do stanu docelowego. Te dwa stany, które mogą być złożone, nazywają się głównym źródłem i głównym ujściem tranzycji.

Stanem ostatniego wspólnego przodka (ang. *least common ancestor (LCA)*) tranzycji jest najniższy stan złożony, który zawiera zarówno stan źródłowy jak i docelowy złożonej tranzycji. (...) Gdy tranzycja jest możliwa i wybrana do odpalenia, wykonują się w kolejności następujące kroki:

1. Stan głównego wyjścia jest właściwie opuszczony.
2. Akcje są wykonywane w kolejności liniowego porządku wzdłuż segmentu tranzycji. Akcje bliżej stany źródłowego są wykonywane wcześniej.
3. (...)
4. Następuje prawidłowe wejście do stan głównego docelowego.

W prezentowanym modelu wszystkie te właściwości są zachowane, oprócz wypunktowanej części, która dotyczy tranzycji złożonej.

2.12.4.7 Maszyna stanów (ang. *StateMachine*)

Przetwarzanie zdarzenia - krok wykonania (ang. *Event processing – run-to-completion step*)

Zdarzenia są rozdzielane i przetwarzane przez maszynę stanów, jedno w danym momencie. Kolejność wyciągania z kolejki nie jest zdefiniowana, pozostawiając otwartą możliwość modelowania różnych schematów opartych na priorytetach.

Semantyka przetwarzania zdarzeń oparta jest na założeniu kroku wykonania, interpretowane jako przetwarzanie ruszaj-do-zakończenia. Przetwarzanie ruszaj-do-zakończenia oznacza, że zdarzenie może być wyciągnięte z kolejki i dostarczone, jeśli przetwarzanie poprzedniego zdarzenia zostało całkowicie zakończone.

Ruszaj-do-zakończenia może być zaimplementowane na wiele sposobów. Dla klas aktywnych, może być zrealizowane przez pętlę zdarzeń uruchomioną

we własnych wątku współbieżnym, która czyta zdarzenia z kolejki. Dla klas biernych może być zaimplementowana jako monitor.

Przetwarzanie pojedynczego zdarzenia przez maszynę stanów jest znane jako krok ruszaj-do-zakończenia. Przez rozpoczęciem kroku ruszaj-do-zakończenia, maszyna stanów jest w stabilnej konfiguracji stanów z wszystkimi akcjami (ale nie koniecznie aktywnościami) zakończonymi. Ten sam warunek musi zachodzić po zakończeniu kroku ruszaj-do-zakończenia. Zatem, zdarzenie nigdy nie będzie przetwarzane, gdy maszyna stanów jest w jakiejś pośredniej i niespójnej sytuacji. Krok ruszaj-do-zakończenia jest przejściem między dwoma konfiguracjami stanów maszyny stanowej.

Założenie ruszaj-do-zakończenia upraszcza funkcję tranzycji w maszynie stanowej, ponieważ konflikty współbieżności są omijane podczas przetwarzania zdarzeń, pozwalając maszynie bezpiecznie zakończyć jej krok ruszaj-do-zakończenia.

Kiedy instancja zdarzenia jest rozsyłana, może powodować umożliwienie odpalenia więcej niż jednej tranzycji. Jeśli żadna tranzycja nie jest możliwa, a zdarzenie nie jest na liście zdarzeń odroczone, zdarzenie jest odrzucane i krok ruszaj-do-zakończenia zostaje zakończony.

W przypadku stanów równoległych jest możliwe odpalenie wielu tranzycji jako wynik tego samego zdarzenia - tyle ile może jedna tranzycja w każdym ze stanów równoległych w bieżącej konfiguracji stanów. W przypadku, gdy jedna lub więcej tranzycji są możliwe, maszyna stanów wybiera pewien podzbiór i go odpala. Które możliwe tranzycje zostają odpalone jest determinowane algorytmem selekcji tranzycji opisanym poniżej. Kolejność w jakiej wyselekcjonowane tranzycje zostaną odpalone nie jest zdefiniowana.

Każdy ortogonalny region w aktywnej konfiguracji stanów, który nie dekomponuje się na równoległe regiony (regiony "niższego poziomu") może odpalić co najwyżej jedną tranzycję jako wynik bieżącego zdarzenia. Kiedy wszystkie regiony ortogonalne zakończą wykonanie tranzycji, instancja aktualnego zdarzenia jest w pełni skonsumowana, a krok ruszaj-do-zakończenia zostaje zakończony.

Podczas tranzycji, pewna ilość akcji może zostać wykonana. Jeśli te akcje są synchroniczne, krok tranzycji jest niezakończony dopóki wywołane obiekty zakończą swoje własne kroki ruszaj-do-zakończenia.

Instancja zdarzenia może przybyć do maszyny stanów, która jest zablokowana w trakcie kroku ruszaj-do-zakończenia z jakiegoś innego obiektu w tym samym wątku, w sposób cykliczny. Taka instancja zdarzenia może być obsługiwana przez ortogonalne komponenty w maszynie stanów, które nie są zamrożone przez tranzycje w tym czasie.

Krok wykonania a równoległość (ang. *Run-to-completion and concurrency*)

Jest możliwe takie zdefiniowanie semantyki maszyny stanów, by kroki ruszaj-do-zakończenia były zastosowane równoległe w każdym z ortogonalnych regionów złożonego stanu, zamiast takiego do całej maszyny stanowej. To po-

zwoliliby na osłabienie wymagania serializowania zdarzeń. Jakkolwiek taka semantyka jest dość subtelna i trudna do zaimplementowania. Z tego powodu semantyka dynamiczna zdefiniowana w tym dokumencie bazuje na przesłance, że krok ruszaj–do–zakończenia zastosowano do całej maszyny stanów i zawiera równoległe kroki wykonywane przez równoległe regiony w konfiguracji aktywnych stanów.

W przypadku aktywnych obiektów, gdzie każdy obiekt ma swój własny wątek wykonania, jest bardzo ważne jasno rozróżnić notacje uruchamiania do zakończenia od pojęcia wywłaszczania wątków (**ang.** *thread pre-emption*). Mianowicie, obsługa zdarzenia ruszaj–do–zakończenia jest wykonywana przez wątek, który (w zasadzie) może być wywłaszczony (**ang.** *pre-empted*), a jego wykonanie zawieszone ze względu na wykonanie innego wątku w tym samym węźle przetwarzania. (Jest to zdeterminowane przez politykę przydzielania (**ang.** *scheduling policy*) w podstawowym wątku środowiska – nie ma żadnych założeń co do tej polityki). Kiedy ten zawieszony wątek otrzyma znowu czas procesora, wznowia swoje przetwarzanie zdarzenia z punktu wywłaszczenia (**ang.** *pre-emption*) i, ewentualnie, kończy to przetwarzania zdarzenia.

Tranzycje konfliktowe (ang.** *Conflicting transitions*)**

Jak już wcześniej wspomniano, jest możliwe, że więcej niż jedna tranzycja jest możliwa w maszynie stanów. Jeśli tak się zdarzy, takie tranzycje mogą być ze sobą w konflikcie. Na przykład, rozważmy przypadek, gdy dwie tranzycje rozpoczynają się z tego samego stanu, wzbudzone tym samym zdarzeniem, ale z różnymi dozorami. Jeśli wystąpi zdarzenie i obydwa dozory są prawdziwe (spełnione), tylko jedna tranzycja zostanie odpalona. Innymi słowy, w przypadku konfliktowym tranzycji tylko jedną z nich można odpalić podczas pojedynczego kroku ruszaj–do–zakończenia.

Powiemy, że dwie tranzycje są w konflikcie, jeśli obydwie wychodzą z tego samego stanu, lub, bardziej precyzyjnie, gdy część wspólna ich zbiorów stanów, które opuszczają nie jest pusta. Tylko tranzycje, które występują w rozdzielnymi ortogonalnych regionach mogą być odpalone równocześnie. To ograniczenie gwarantuje, że nowa konfiguracja stanów aktywnych powstała z wykonania zbioru tranzycji będzie dobrze skonstruowana. Wewnętrzne tranzycje w stanie mogą być w konflikcie tylko z tranzycjami, które powodują wyjście z tego stanu.

Priorytety odpalania (ang.** *Firing priorities*)**

W sytuacji, gdy istnieją tranzycje konfliktowe, wybór tranzycji do odpalenia bazuje w części na bezwzględnych priorytetach. Te priorytety rozwiązują pewne konflikty tranzycji, ale nie wszystkie. Priorytety konfliktowych tranzycji za podstawę biorą ich względne położenie w hierarchii stanów. Z definicji, tranzycja zaczynająca się z podstanu ma wyższy priorytet niż jakakolwiek tranzycja konfliktowa zaczynająca się w dowolnym z nadstanów.

Priorytet tranzycji jest zdefiniowany na podstawie jej stanu źródłowego. (...)

Generalnie, jeśli $t1$ jest tranzycją, której stanem źródłowym jest $s1$, a $t2$ ma źródło $s2$, to:

- Jeśli $s1$ jest bezpośrednim lub pośrednio zagnieżdżonym podstanem $s2$, to $t1$ ma wyższy priorytet niż $t2$.
- Jeśli $s1$ i $s2$ nie są w tej samej konfiguracji stanów, to nie ma żadnej różnicy między $t1$ i $t2$ w priorytetach .

Algorytm wyboru selekcji (ang. *Transition selection algorithm*)

Zbiorem tranzycji, które zostaną odpalone jest maksymalny zbiór tranzycji, który spełnia następujące warunki:

- Wszystkie tranzycje w zbiorze są możliwe.
- Nie ma tranzycji konfliktowych w tym zbiorze.
- Nie istnieje tranzycja poza zbiorem, która ma wyższy priorytet niż tranzycja w zbiorze (...).

Można to zaimplementować poprzez zachłanny algorytm przeglądu w przód konfiguracji stanów aktywnych. Stany w konfiguracji stanów aktywnych są odwiedzane startując od najbardziej zagnieżdżonych prostych stanów i posuwając się na zewnątrz do najwyższego stanu. Dla każdego stanu na danym poziomie, wszystkie tranzycje zaczynające się w nim są przetwarzane w celu stwierdzenia czy są możliwe. Takie odwiedzanie zapewnia, że zasada priorytetów nie będzie naruszona. Jedyną nietrywialną kwestią jest rozwiązanie konfliktów tranzycji.

W ramach przyjętego modelu większość przedstawionych postulatów jest zachowana. W przypadku równoległości w pracy przyjęto, że krok wykonania jest rozpatrywany osobno w każdym regionie ortogonalnym. Oznacza to, przed zakończeniem kroku wykonania w jednym regionie, możliwe jest wykonywanie kroku wykonania w innym regionie. Przy takim założeniu doprecyzowania wymaga działanie maszyny stanowej, co znajduje się w rozdziale 6.

2.12.4.8 Stany synchronizacyjne (ang. *Synch States*)

Stany synchronizacyjne postarczają środków do synchronizacji wykonania dwóch równoległych regionów.(...)

2.12.4.9 Stany łącznikowe (ang. *StubStates*)

Stany łącznikowe są pseudostanami oznaczającymi albo punkt wejścia do albo punkt wyjścia z podmaszyny.

Stany synchronizacyjne nie zostały wprowadzone. Stany łącznikowe również, gdyż jako pseudostany służą tylko do większej zwężłości zapisu dużej maszyny stanów.

Posumowując, przyjęty model maszyny stanowej zawiera stany proste i złożone, zarówno sekwencyjnie jak i równolegle. Nie zawiera stanów końcowych, stanów podmaszyny oraz stanów synchronizacyjnych czy łącznikowych. Z możliwych cech stany mają określony czas trwania akcji wejściowej oraz akcji wyjściowej. Dopuszczalne są tylko tranzycje nie przecinające granic stanów. Tranzycje te w przypadku maszyn środowiska wzbudzane są zakończeniem aktywności w stanie, natomiast w przypadku maszyny obiektu, sygnałem ze środowiska lub przeterminowaniem. Inaczej jest jednak rozumiany krok działania, gdyż w każdym z regionów równoległych może w dowolnym momencie wzbudzić się tranzycja. Pozostałe normy UML dotyczące działania maszyny stanowej (np. priorytet tranzycji wewnętrznych) zostały zachowane.

5 Model diagramu stanów

W maszynie stanowej język UML dopuszcza możliwość istnienia złożonych stanów. Mogą to być stany złożone sekwencyjnie jak i równolegle. Pomiedzy stanami istnieją przejścia – tranzycje. Z tranzycją mogą być związane warunki jej wzbudzenia. Taka struktura podowuje, że diagram stanów maszyny UML najlepiej zdefiniować dwustopniowo [29].

5.1 Graf diagramu stanów

Najpierw zdefiniowany zostanie *graf diagramu stanów* jako szóstka:

$$S = \langle BoxN, childB, typeB, defaultB, ArcN, Arc \rangle \quad (24)$$

gdzie:

- $BoxN$ jest skończonym zbiorem nazw wierzchołków
- $childB \subseteq BoxN \times BoxN$ jest hierarchii: $\langle b_1, b_2 \rangle \in childB$ oznacza, że b_2 jest bezpośrednim wierzchołkiem potomnym wierzchołka b_1 . Zbiór $BoxN$ oraz relacja $childB$ definiuje drzewo wierzchołków $\langle BoxN, childB \rangle$ o korzeniu w wierzchołku $root(\langle BoxN, childB \rangle)$. Niech $childB^*$ jest zwrotnym, przechodnim domknięciem relacji $childB$. Niech $childB(b)$ oznacza zbiór wierzchołków bezpośrednio potomnych wierzchołka b , natomiast $childB^*(b)$ oznacza zbiór wierzchołków pośrednio potomnych wierzchołka b . Analogicznie $childB^{-1}(b)$ oznacza bezpośredniego przodka, natomiast $childB^{*-1}(b)$ zbiór pośrednich przodków wierzchołka b .
- $typeB: BoxN \rightarrow \{SimS, SeqCS, ConCS, \}$ jest funkcją określającą rodzaj wierzchołka. Korzeń jest rodzaju $SeqCS$, liście są rodzaju $SimS$ lub $FinS$, zaś inne wierzchołki mogą być rodzaju $SeqCS$ lub $ConCS$. Te dwa rodzaje wierzchołki będą nazywał wierzchołkami złożonymi.

- $defaultB: BoxN \rightarrow 2^{BoxN}$ jest funkcją, która dla wierzchołków złożonych wyznacza początkowe wierzchołki w obrębie wierzchołków bezpośrednio potomnych danego wierzchołka. Dla wierzchołka rodzaju $SeqCS$ jest dokładnie jeden taki wierzchołek, natomiast $ConCS$ jest nim zbiór wszystkich bezpośrednich wierzchołków potomnych. Zachodzi oczywista zależność $defaultB(b) \subseteq childB(b)$.
- $ArcN$ jest skończonym zbiorem nazw łuków. $BoxN \cap ArcN = \emptyset$
- $Arc \subseteq BoxN \times ArcN \times BoxN$ jest skończonym zbiorem łuków. W trójce opisującej łuk $\langle b_1, a, b_2 \rangle \in Arc$, b_1 będzie nazywany *wierzchołkiem źródłowym*, natomiast b_2 *wierzchołkiem docelowym*. Zakładając, że nazwa łuku a jednoznacznie określa łuk, zatem do wyznaczania wierzchołka źródłowego i docelowego będą funkcje $source(a)$ i $sink(a)$.

W dalszych rozważaniach dla odróżnienia grafów i ich składowych będą używane dolne indeksy. Np. S_{ENV} oznacza graf środowiska, S_{MS_1} oznacza graf pierwszej maszyny, natomiast $BoxN_{S_{ENV}}$ oznacza $BoxN$ środowiska. Jeśli nie będzie to prowadziło do niejednoznaczności, indeksy te będą pomijane. We wszystkich dalszych rozważaniach zakładać się będzie, że łuki grafu nie mogą przecinać granic wierzchołków.

5.2 Diagram stanów

Graf diagramu stanów należy uzupełnić o elementy takie jak czas, symbole zdarzeń, reakcji itd. Do dalszych rozważań w diagramie stanów ujęte będą następujące właściwości:

- Każdy stan posiada czas wykonywania akcji wejściowej i czas opuszczania stanu. Pierwszy oznacza czas potrzebny od rozpoczęcia wchodzenia do danego stanu do momentu możliwego podjęcia następnych działań: wchodzenia do stanów wewnętrznych lub opuszczenia stanu. Zaczyna się od momentu zakończenia tranzycji kończącej się w tym stanie. Drugi czas to czas potrzebny do ostatecznego wyjścia ze stanu. Jeśli z jakiegoś powodu należy opuścić stan, to po zakończeniu wszelkich aktywności w stanach wewnętrznych, zaczyna biec czas wychodzenia ze stanu.
- Każdy łuk wraz z opisem nazywany będzie *tranzycją*. Podczas działania tranzycja może być *wzbudzona*, czyli przygotowana do wykonania lub *uruchomiona*. Wzbudzenie tranzycji oznacza, że do czasu uruchomienia nie mogą zostać wbudzone żadne inne tranzycje zawierające się lub wychodzące z jej stanu źródłowego. Z każdym łukiem związane jest co najwyżej jedno zdarzenie, zapisane jako zbiór. W przypadku symbolu parametrycznego $after(x)$, oznacza to zdarzenie czasowe polegające na przeterminowaniu równym x , które to przeterminowanie zaczyna biec od momentu rozpoczęcia wchodzenia do stanu źródłowego. Po tym czasie następuje wzbudzenie tranzycji związanej z tym zdarzeniem czasowym. Jeśli jest to inny symbol zdarzenie nieczasowego, to wzbudzenie tranzycji następuje w momencie nadejścia danego zdarzenia ze środowiska. Jeśli zbiór symboli wzbudzających tranzycję jest pusty to oznacza, że tranzycja zostanie wzbudzona od razu po zakończeniu wszelkich aktywności w stanach wewnętrznych stanu źródłowego. Każda

tranzycja ma też czas trwania tranzycji, czyli czas potrzebny na wykonanie tranzycji oraz co najwyżej jednoelementowy zbiór reakcji generowany po tym czasie. Czasem pojawienia się reakcji maszyny jest czas zakończenia wykonywania tranzycji. W ogólnym przypadku można rozszerzyć opis tranzycji o moment, w którym następuje reakcja, jednak ta zmiana nie wnosi nowej jakości do dalszych rozważań. Dla celów rozwiązywania konfliktów łuki muszą posiadać też wagi.

Diagram stanów będzie opisywany jako piątka:

$$MS = \langle S, BL, AL, EV, OU \rangle \quad (25)$$

gdzie:

- S jest grafem diagramu stanów
- $BL(b) = \langle ent(b), ext(b) \rangle$ dla $b \in BoxN$. $ent(b) \in Time$ oznacza czas wchodzenia do wierzchołka b (**ang. *entry time***), natomiast $ext(b) \in Time$ oznacza czas wychodzenia z wierzchołka b (**ang. *exit time***).
- $AL(tr) = \langle trt(tr), tre(tr), tro(tr), trw(tr) \rangle$ dla $tr \in ArcN$.
 - $trt(tr) \in Time$ oznacza czas trwania tranzycji od opuszczenia stanu źródłowego do momentu rozpoczęcia akcji wejściowej stanu docelowego, (**ang. *transition time***),
 - $tre(tr) \subset EV \cup \{after(x) | x \in R_+\}$, gdzie zachodzi $|tre(tr)| \leq 1$ jest zbiorem symboli zdarzeń (**ang. *transition event***), wyzwalającego daną tranzycję. Traktowanie zdarzenia jako zbioru ułatwia zapis braku zdarzenia.
 - $tro(tr) \in 2^{OU}$, $|tro(tr)| \leq 1$ oznacza zbiór zdarzeń wyjściowych generowanych po czasie $trt(tr)$ przez tranzycję tr (**ang. *transition outcome***). Jeśli $tro(tr) = \emptyset$, to oznacza brak generacji zdarzenia zewnętrznego (reakcji).
 - $trw(tr) \in \mathcal{N}$ oznacza wagę tranzycji (**ang. *transition weight***). Będzie on decydował o wybraniu tranzycji do wzbudzenia w momencie, gdy reguły UML nie będą jednoznacznie wyznaczały tranzycji do wzbudzenia. Każda tranzycja w ramach diagramu stanów ma mieć inny priorytet.
- EV jest zbiorem symboli zdarzeń ze środowiska obsługiwanych przez daną maszynę stanową. Poza tymi zdarzeniami są zdarzenia przeterminowania oraz zdarzenia zakończenia aktywności w stanie. (**ang. *EVents***)
- OU jest zbiorem symboli reakcji generowanych przez daną maszynę stanową. (**ang. *OUTcomes***)

Tu również będą stosowane dolne indeksy, jeśli zajdzie taka konieczność.

6 Konkretyzacja metamodelu diagramu stanów

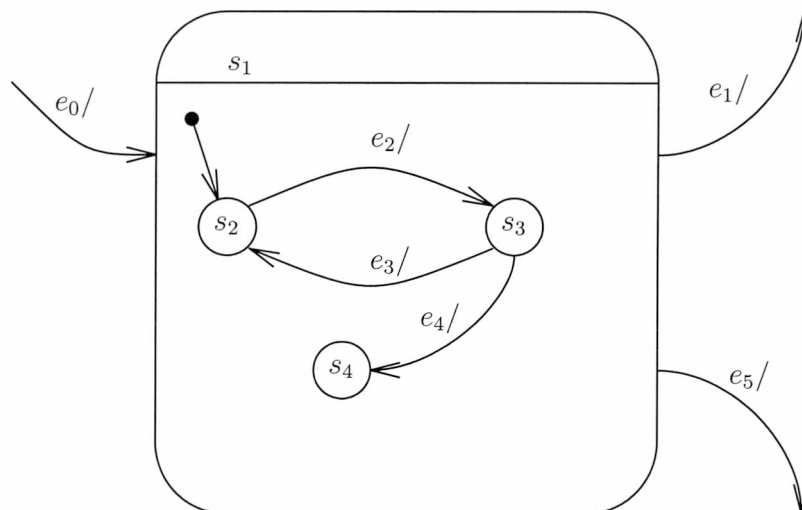
Interpretacja diagramów stanów UML z czasem rzeczywistym nie jest jednoznaczna [13], przedstawiona zatem zostanie konkretyzacja modelu diagramu stanów. W diagramach stanów języka UML występują stany złożone (które to złożenie może być zarówno sekwencyjne jak i równoległe). Każdy z tych stanów wykonuje jakąś akcję wejściową, która wymaga czasu. Również wejście do stanów potomnych pociąga za sobą wykonanie ich akcji wejściowych. Do opuszczenia stanu również potrzeba czasu, zależnego pośrednio od czasów zakończenia aktywności w podstanach. Należy doprecyzować zachowanie się maszyny w danej sytuacji w zależności od momentu i ilość napływających zdarzeń (co nie jest rozważane w specyfikacji UML).

6.1 Wzbudzenie tranzycji a uruchomienie tranzycji

Przejście z jednego stanu do drugiego, dokonuje się poprzez uruchomienie pewnej tranzycji. Powodem, dla którego następuje uruchomienie tranzycji może być zakończenie akcji wewnętrznych, upływ czasu, pojawienie się zdarzenia wyzwalającego daną tranzycję. Jednak zanim uruchomi się dane przejście w wielu przypadkach maszyna musi jeszcze zakończyć aktywności w stanie źródłowym tranzycji. Stąd będzie stosowane rozróżnienie na wzbudzenie/wyzwolenie tranzycji i uruchomienie tranzycji. Momentem wzbudzenia tranzycji jest najczęściej moment zajścia sytuacji umożliwiającej przejście po danej tranzycji (uruchomienie jej), natomiast czasem uruchomienia tranzycji jest moment, kiedy rozpoczyna się przejście, czyli ostateczne opuszczenie stan źródłowy tranzycji.

6.2 Kolejność i czas realizacji tranzycji zdarzeniowych

Jako przykład do dalszych rozważań w tym podrozdziale posłuży fragment maszyny zaprezentowany na rys. Rys (16).



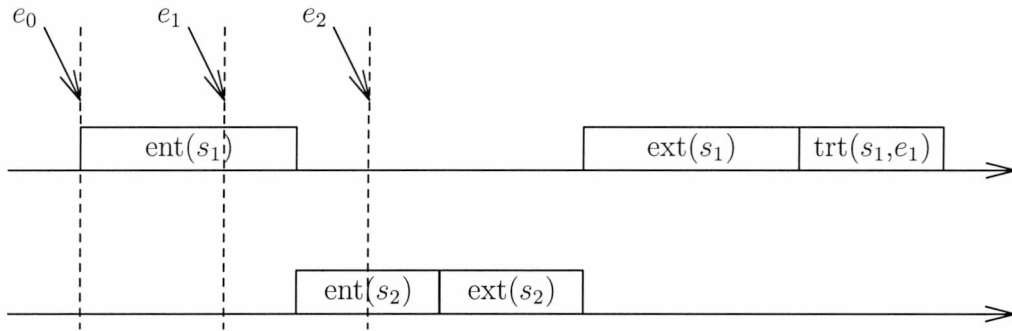
Rysunek 16: Stan złożony z wewnętrznymi tranzycjami

Stan s_1 zawiera stany s_2 i s_3 , do stanu s_1 dochodzi tranzycja wzbudzana zdarzeniem e_0 , natomiast wychodzi wzbudzana zdarzeniem e_1 oraz inna zdarzeniem e_5 . W stanach wewnętrznych istnieje tranzycja z s_2 do s_3 wzbudzana zdarzeniem e_2 oraz ze stanu s_3 do s_2 wzbudzana zdarzeniem e_3 (oprócz tego istnieje tranzycja ze stanu s_3 o zdarzeniu e_4). Należy zastanowić się jak zinterpretować różną kolejność występowania e_1 i e_2 . Po wejściu do stanu poprzez tranzycję ze zdarzeniem e_0 następuje wchodzenie do stanu s_1 oraz do wszystkich jego podstanów. Mamy trzy przypadki zachodzenia zdarzeń e_1 i e_2 : najpierw e_1 , potem e_2 ; najpierw e_2 , potem e_1 ; obydwa zdarzenia równocześnie.

Jak wynika z dotychczasowych rozważań na działanie maszyny ma wpływ nie tylko kolejność występowania tych zdarzeń, ale również czas pojawienia się ich. W kolejnych częściach rozważone zostaną wszystkie możliwe przypadki pojawienia się w stanie s_1 zdarzeń e_1 i e_2 , oraz w konsekwencji reakcje maszyny.

Przyjęto następujące oznaczenia: $ent(s)$ – czas wykonywania akcji wejściowej w stanie s , $ext(s)$ – czas wykonywania akcji wyjściowej w stanie s , $trt(s, e)$ – czas trwania tranzycji zaczynającej się w stanie s i wzbudzonej zdarzeniem e .

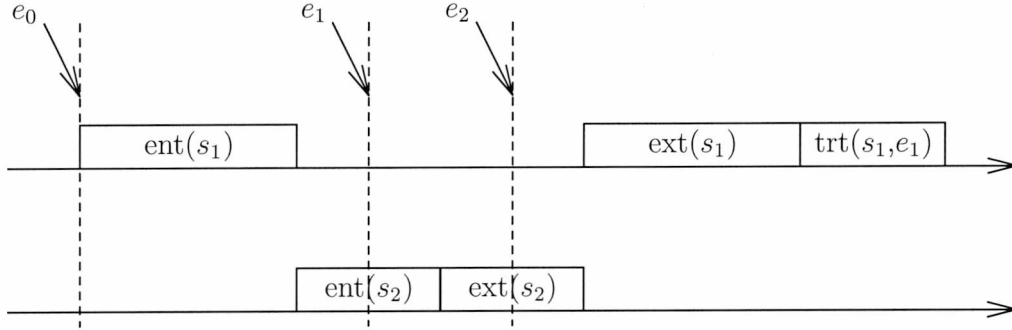
6.2.1 Nastęstwo: najpierw e_1 , potem e_2



Rysunek 17: Pojawienie się zdarzenia e_1 w czasie wejścia do stanu s_1

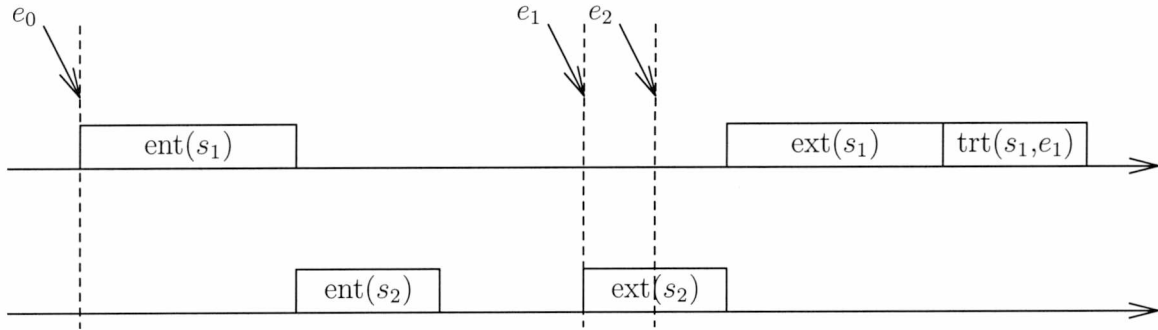
Jeśli zdarzenie e_1 nastąpi w czasie wchodzenia do stanu s_1 , to powinno jak najszybciej dojść do wyjścia z podstanów tego stanu. Wówczas czas wystąpienia zdarzenia e_2 nie ma znaczenia Rys (17). Gdyby przyjąć, że zdarzenie e_2 ma zostać w tym przypadku obsłużone w podstanie s_2 , to konsekwentne dalsze stosowanie takiej reguły spowodować mogłoby, że zdarzenie e_1 nigdy nie zostanie obsłużone. W momencie wystąpienia zdarzenia e_1 następuje wbudzenie tranzycji z nim związanej. Oznacza to, że żadna inna tranzycja w tym stanie jak i w stanach potomnych nie może już zostać wzbudzona (oczywiście do czasu zakończenia wykonywania wzbudzonej tranzycji). Tranzycja wzbudzona zostanie wykonana w najbliższym możliwym terminie, co w tym przypadku znaczy, że zacznie się wykonywać na początku przedziału zaznaczonego jako $trt(s_1, e_1)$. Jednocześnie wbudzenie tej tranzycji wymusza jak najszybsze wyjście z jej stanu źródłowego, czyli ze stanu s_1 . Ponieważ wyjście z tego stanu zależy od wyjścia z jego stanów podrzędnych, zatem rekurencyjnie wymusza to jak najszybsze rozpoczęcie akcji wyjściowej ze stanu s_2 . Jak nietrudno zauważyć, między wzbudzeniem tranzycji a jej uruchomieniem wykonana się bardzo dużo działań. W rozważanym przypadku nastąpi dokończenie

akcji wejściowej stanu s_1 , wykonanie wejścia do jego podstanu s_2 , wyjście ze stanu s_2 i ostatecznie wyjście ze stanu s_1 .



Rysunek 18: Pojawienie się zdarzenia e_1 w czasie wejścia do stanu s_2

W sytuacji Rys (18), gdy zdarzenie e_1 następuje w czasie wchodzenia do podstanu niczym w zachowaniu maszyny nie różni się od poprzedniej. Różny będzie tylko czas wzbudzenia tranzycji.



Rysunek 19: e_1 po czasie wejścia do stanu s_2

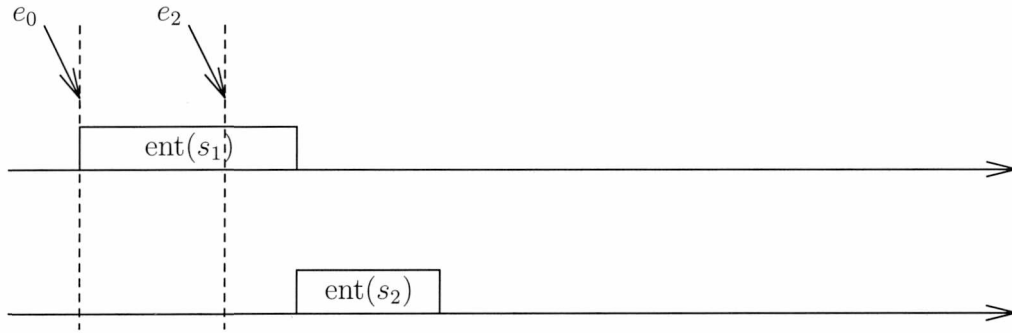
Na Rys (19) zdarzenie e_1 pojawi się już po wejściu do podstanu. Sytuacja taka różni się od poprzednich tylko czasem rozpoczęcia się wychodzenia z podstanu s_2 .

We wszystkich tych wypadkach pojawienie się jakiegokolwiek zdarzenia e_3, e_4, e_5 po zdarzeniu e_1 nie spowoduje zmiany zachowań czasowych oraz kolejności uruchamiania tranzycji.

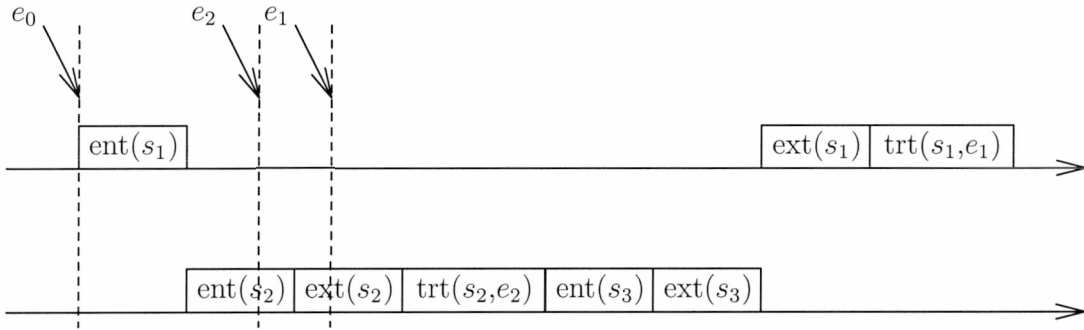
6.2.2 Następstwo: najpierw e_2 , potem e_1

Jeśli zdarzenie e_2 nadejdzie w czasie, gdy jeszcze nie zaczął się proces wchodzenia do stanu s_2 , to znaczy, że nie ma stanu, który może obsłużyć to zdarzenie. Zatem maszyna zignoruje to zdarzenie Rys (20). Dalsze rozważania są takie same jak dla poprzedniego punktu.

Jeśli zdarzenie e_2 nadejdzie w czasie wchodzenia do stanu s_2 , to po skończeniu wchodzenia do tego stanu, wykona się tranzycja wzbudzana tym zdarzeniem. Jeśli po zdarzeniu e_2 a przed rozpoczęciem wchodzenia do stanu s_3 pojawi się zdarzenie e_1 , to w stanie s_3 natychmiast po wejściu do niego nastąpi podjęcie akcji wychodzenia z tego stanu,



Rysunek 20: Pojawienie się zdarzenia e_2 w czasie wejścia do stanu s_1

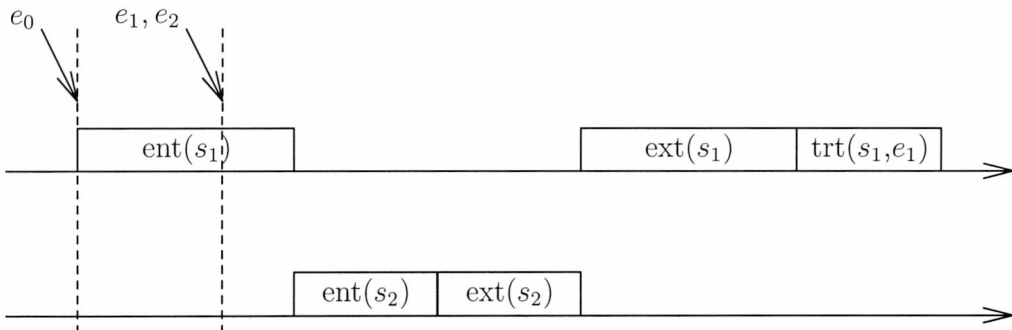


Rysunek 21: Pojawienie się zdarzenia e_2 w czasie wejścia do stanu s_2

a następnie wychodzenia ze stanu s_1 Rys (21). Jeśli zdarzenie e_1 nastąpi już w czasie wchodzenia do s_3 , to powstaje sytuacja jak z rys (18) po zamianie stanu s_2 na s_3 .

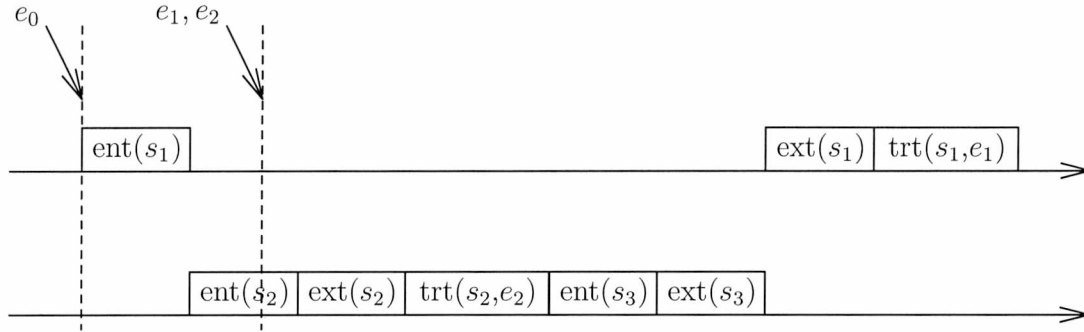
Jeśli zdarzenie e_2 nastąpi po czasie wchodzenia do stanu s_2 , to sytuacja będzie podobna do tej z rys. (21), z tym że wyjście ze stanu s_2 zostanie podjęte wraz z momentem otrzymania tego zdarzenia.

6.2.3 Następstwo: jednocześnie e_1 i e_2



Rysunek 22: Pojawienie się zdarzeń e_1 i e_2 w czasie wejścia do stanu s_1

Jeśli obydwa zdarzenia odbywają się jednocześnie w czasie wchodzenia do stanu s_1 (Rys 22), to oczywiście zdarzenie e_2 nie może być obsłużone. Zatem działanie jest takie samo jak w przypadku, gdy wystąpiłoby tylko zdarzenie e_1 .



Rysunek 23: Pojawienie się zdarzenia e_2 w czasie wejścia do stanu s_2

Jeśli obydwa zdarzenia pojawiają się, gdy stan s_2 już jest aktywny (Rys 23), to (ponieważ UML preferuje wykonanie tranzycji wewnętrznych) działanie będzie takie jakby najpierw wystąpiło zdarzenie e_2 a zaraz za nim zdarzenie e_1 . Oczywiście jeśli zdarzenia te nastąpią po wejściu do stanu s_2 , to wyjście z niego zacznie się dopiero w czasie nadejścia tych zdarzeń.

6.3 Preferencja tranzycji wewnętrznych

Zgodnie ze specyfikacją UML w przypadku, gdy stan s_1 zawiera w sobie stan s_2 , oraz z obydwoch zaczyna się tranzycja wzbudzana zdarzeniem e , to jeśli w bieżącej konfiguracji maszyny są aktywne obydwa stany, to przy nadejściu tego zdarzenia wzbudzona zostanie tylko tranzycja wychodząca ze stanu wewnętrznego s_2 . Założenie to jest zachowane w dalszych rozważaniach, chociaż w momencie, gdy czym innym jest wzbudzenie tranzycji a czym innym jej uruchomienie, równie uprawnione mogłoby być podejsie, że wzbudzone są obydwie tranzycje, natomiast najpier uruchomi się tranzycja ze stanu wewnętrznego, a dopiero później ta ze stanu zewnętrznego.

6.4 Preferencja tranzycji nieczasowych

Inną sytuacją jest przypadek, gdy ze stanu rozpoczynają się dwie tranzycje, jedna wzbudzana jakimś zdarzeniem nieczasowym, a druga upływem czasu. W przypadku, gdy maszyna ma mieć działanie deterministyczne należałoby przyjąć preferencje na rzecz jednego z typów tranzycji. W tym opracowaniu przyjmuje się preferencje tranzycji nieczasowych, chociaż odwrotne założenie zmieniłoby tylko nieznacznie definicje różnych pojęć i funkcji, nie modyfikując toku rozumowania. Podobne założenie przyjęte zostanie, jeśli w tym samym momencie wzbudzona może być tranzycja czasowa i nieczasowa w różnych stanach tej samej części równoległej. Pierwszeństwo wzbudzenia ma tranzycja nieczasowa. Oznacza to, że jeśli tranzycja czasowa jest ze stanu wewnętrznego, to nie zostanie wzbudzona, tylko nastąpi proces wychodzenia ze stanu zewnętrznego.

6.5 Kolejność realizacji tranzycji czasowych

Jeśli stan s_1 zawiera w sobie stan s_2 , oraz z obydwóch zaczynają się tranzycje czasowe, które w tym samym momencie obydwie ulegną przedawnieniu, wówczas wzbudzeniu ulegną obydwie tranzycje. Jednak analogicznie jak w poprzednim przypadku pierwszeństwo wykonania będzie miała tranzycja ze stanu wewnętrznego s_2 .

6.6 Moment reakcji maszyny podczas wykonywania tranzycji

Należałoby jeszcze doprecyzować, kiedy podczas tranzycji pojawia się reakcja maszyny. Dla uproszczenia rozważań przyjęto, że następuje to po czasie trwania tranzycji trt . W ogólnym przypadku można rozszerzyć opis tranzycji o moment, w którym następuje reakcja, jednak ta zmiana nie wnosi nowej jakości do dalszych rozważań.

6.7 Zbiór reakcji a nie kolekcja

W przypadku maszyny środowiska, która ma części równoległe, może pojawić się sytuacja, gdy w tym samym czasie różne części wygenerują to samo zdarzenie. Zgodnie ze specyfikacją UML, jeśli pojawi się jakieś zdarzenie, to w częściach równoległych uruchomione zostaną wszystkie tranzycje odpalane tym zdarzeniem. Jeśli symbole i czasy tych reakcji są identyczne można albo potraktować takie zdarzenia jako jedno zdarzenie, albo zabronić takiej sytuacji. W pracy przyjęto, że w tym samym momencie środowisko może wygenerować wiele zdarzeń o tym samym symbolu i zdarzenia te będą stanowiły zbiór, zatem powtarzające się zdarzenia będą traktowane jak jedno.

7 Model zachowania środowiska

7.1 Właściwości

Środowisko zostanie opisane za pomocą diagramu stanów języka UML, jednak pewne elementy wzoru (25) muszą spełniać określone właściwości.

Działanie maszyny środowiska ENV nie jest zależne od żadnych zdarzeń zewnętrznych, zatem $EV = \emptyset$. Maszyna taka powinna być w swym działaniu niezdeteminowana, a uruchomienia tranzycji powinno zależeć tylko od upływu czasu. Aby osiągnąć niedeterminizm można dopuścić zarówno istnienie tranzycji czasowych jak i wzbudzanych zakończeniem wewnętrznych aktywności. Jednak dopuszczenie tylko drugiego rodzaju tranzycji w połączeniu z czasem potrzebnym na opuszczenie stanu jest wystarczające. Z tego powodu dla każdego wierzchołka niekońcowego b zachodzi $ent(b) = 0$ (wejście jest natychmiastowe) oraz $ext(b) > 0$, a dla każdej tranzycji tr zachodzi $trt(tr) = 0$, $tre(tr) = \emptyset$. Żaden stan końcowy s nie posiada akcji wejściowej i wyjściowej, stąd zachodzi wtedy $ent(s) = 0$ oraz $ext(s) = 0$. Ponieważ maszyna ta ma być niezdeteminowana, zatem waga $trw(tr)$ nie jest używana. Maszyna o tak zdefiniowanych parametrach za każdym uruchomieniem może generować inną sekwencję zdarzeń czasowych.

7.2 Operacje na diagramie stanów środowiska

We wszystkich kolejnych rozważaniach założono, że istnieje pewna maszyna środowiska *ENV* spełniająca opisane właściwości. Analogicznie jak dla rozważania maszyn bezczasowych należy wprowadzić pojęcie konfiguracji środowiska. W przypadku opisywania maszyn bezczasowych wystarcza do tego celu zbiór stanów bieżących (aktywnych).

Ponieważ stany są w pewnej hierarchii drzewiastej, zatem jeśli w konfiguracji jest jakiś stan, to muszą w nim być również jego stany nadrzędne. Jednak nie będzie z tego wynikać, że w konfiguracji musi być co najmniej jeden stan podrzędny stanu złożonego. Jeśli jakiś stan, w rozpatrywanym momencie, zakończył swoją akcję wyjściową, to nie będzie ujęty w konfiguracji.

W przypadku czasowych maszyn stanów oprócz stanu należy pamiętać w jakiś sposób czas "przebywania w stanie". Ponieważ dla maszyn środowiska ma zachodzić dla każdego stanu s $ent(s) = 0$, natomiast zaraz po opuszczeniu wszystkich podstanów następuje wychodzenia ze stanu w czasie $ext(s) \neq 0$ (tranzycje wzbudzone są zakończeniem działalności w podstanach), zatem należy pamiętać ile czasu musi upłynąć (po opuszczeniu wszystkich podstanów), aby wyjść ze stanu. Czas ten będzie nazywany *czasem wyjścia ze stanu*.

Ostatecznie *konfiguracją środowiska* będzie zestaw aktywnych stanów wraz z informacją ile czasu pozostało do wyjścia ze stanu. Czas ten będzie dla każdego stanu względny do czasu zakończenia się aktywności w ramach jego stanów potomnych.

Definicja 10. *Konfiguracja środowiska jest zbiorem dwójek:*

$$Conf = \left\{ \langle s_1, t_1 \rangle, \langle s_2, t_2 \rangle, \dots, \langle s_n, t_n \rangle \right\}$$

gdzie dla każdego $i \in \{1, \dots, n\}$ (26)

$s_i \in BoxN$ - aktywny stan

$t_i \in Time$ - czas do wyjścia ze stanu s_i

Aby zmieniać konfigurację z jednej w drugą należy podać metodę takiej transformacji oraz sposób tworzenia konfiguracji początkowej. W tym celu wprowadzona zostanie funkcja wejścia do dowolnego stanu. Wejście do stanu wymusza wejście do wszystkich początkowych stanów wewnętrznych.

Definicja 11. *Wejście do stanu s wymusza rekurencyjnie wejście do wszystkich początkowych stanów wewnętrznych:*

$$Entry(s) = \emptyset$$

gdzie $TypeB(s) = FinS$

$$Entry(s) = \left\{ \langle s, ext(s) \rangle \right\}$$

gdzie $TypeB(s) = SimS$ (27)

$$Entry(s) = \left\{ \langle s, ext(s) \rangle \right\} \cup \bigcup_{s' \in defaultB(s)} Entry(s')$$

gdzie $TypeB(s) \in \{SeqCS, ConCS\}$

Konfigurację początkową środowiska ENV , wyznacza wejście do korzenia grafu tego środowiska. Zapisując to formalnie:

Definicja 12. Konfiguracja początkowa $Conf_0^{ENV}$ środowiska ENV to wejście do korzenia grafu stanów tego środowiska:

$$Conf_0^{ENV} = Entry\left(\text{root}\left(\left\langle BoxN_{ENV}, childB_{ENV} \right\rangle\right)\right) \quad (28)$$

Ponieważ w środowisku wszystkie tranzycje są wzbudzone i uruchamiane zakończeniem akcji wewnętrznych, zatem w stanie s , który nie posiada stanów wewnętrznych, rozpoczyna się od razu wyjście ze stanu. Czas wyjścia każdego stanu początkowo ma wartość $ext(s)$ jednak konfiguracja może ulec zmianie w krótszym czasie (z powodu zachowań w częściach równoległych). Zdefiniowana zostanie zatem funkcja selekcji, podającą ile czasu musi upłynąć do zakończenia akcji wejściowej dla zadanego stanu w rozpatrywanej konfiguracji:²

Definicja 13. Czasem do wyjścia $tte(C, s)$ (**ang. *Time To Exit***) określonym dla stanu s w konfiguracji C jest:

$$tte(C, s) = t \iff \exists_t \bullet \left(\langle s, t \rangle \in C \right) \quad (29)$$

Podobnie zachodzi potrzeba odpowiedzi na pytanie, czy zadany stan jest w bieżącej konfiguracji. W tym celu przeciążony zostanie zbiorowy symbol "należy":

Oznaczenie 2. Stan s należy do konfiguracji C oznacza, że:

$$s \in C \iff \langle s, tte(C, s) \rangle \in C. \quad (30)$$

Mimo, że stany w konfiguracji stanowią zbiór, to równocześnie wyznaczają pewne drzewo. Analogicznie jak w teorii grafów zdefiniowane zostanie pojęcie liści konfiguracji:

Definicja 14. Liściem konfiguracji środowiska będziemy nazwany taki wierzchołek konfiguracji, który nie jest przodkiem żadnego innego wierzchołka tej konfiguracji. Zbiór takich liści konfiguracji C oznaczony zostanie jako:

$$Leaf(C) = \{s | s \in C \wedge \neg \exists_{s'} \bullet (s' \in C \wedge s' \in childB(s))\} \quad (31)$$

Ponieważ konfiguracja środowiska zmieniać się może tylko poprzez zakończenie akcji wyjściowej ze stanu, zatem moment kolejnej zmiany, to najmniejszy czas do wyjścia wśród stanów - liści.

²We wszystkich wzorach z kwantyfikatorami, jeśli nie jest to podane wprost, to w przypadku stanu lub tranzycji zakłada się, że są one ze zbioru wszystkich stanów lub tranzycji rozważanej maszyny.

Definicja 15. Czas minimalny konfiguracji C to najkrótszy czas wyjścia spośród czasów wyjścia liści tej konfiguracji:

$$MinT(C) = \min_{s \in Leaf(C)} (tte(C, s)) \quad (32)$$

Po znalezieniu odpowiedniego czasu zmiana konfiguracji polega na wyjściu z odpowiednich stanów, wejściu do ewentualnych odpowiednich innych stanów oraz przesunięciu punktów czasowych na moment tych wszystkich zmian. Zdefiniowana zostanie najpierw ostatnia operacja, niezależna od pozostałych.

Definicja 16. Przesunięciem czasowym do zera konfiguracji C jest funkcja $ShTm0(C)$ (**ang. *Shift Time to zero***), przesuwająca czas liści konfiguracji do najbliższego wyzerowania:

$$ShTm0(C) = \left\{ \langle s, t' \rangle \mid \left(\langle s, t' \rangle \in C \wedge s \notin Leaf(C) \right) \vee \left(\langle s, t' \rangle \in C \wedge s \in Leaf(C) \wedge t' = t - MinT(C) \right) \right\} \quad (33)$$

Wykonując najpierw operację przesunięcia otrzymuje się w prosty sposób stany, które zostaną opuszczone. To te, których czas do wyjścia wynosi zero.

Definicja 17. Zbiorem liści zerowych konfiguracji C jest zbiór $Leaf0(C)$, taki że:

$$Leaf0(C) = \{s \mid s \in Leaf(C) \wedge tte(C, s) = 0\} \quad (34)$$

Ponieważ maszyna środowiska nie jest deterministyczna, zatem może istnieć więcej niż jeden zbiór tranzycji powodujących poprawną zmianę konfiguracji. Właściwości jakie powinien mieć ten zbiór, to aby dla każdego stanu ze zbioru liści zerowych była jedna tranzycja (jeśli taka istnieje) z niego wyprowadzająca. Zbiór tych tranzycji powinien być maksymalnie liczny.

Dodano zatem funkcję pomocniczą istnienia tranzycji ze stanu s :

Definicja 18. Funkcja logiczna $etf(s)$ (**ang. *exists transition from***) dla danego stanu s odpowiada, czy stan posiada jakąkolwiek tranzycję wyprowadzającą z niego:

$$etf(s) = \exists_{tr} \bullet (source(tr) = s) \quad (35)$$

Potrzebna jest funkcja sprawdzająca poprawność zbioru tranzycji w stosunku do danego zbioru stanów, która jest przedstawiona poniżej.

Definicja 19. Funkcja sprawdzająca *dopuszczalność zbioru tranzycji* $GTS(Tr, S)$ (**ang.** *Good Transition Set*) zdefiniowana rekursywnie ma postać (przy założeniu, że $s \in S$) następującą:

$$\begin{aligned}
GTS(\emptyset, \emptyset) &= true \\
GTS(Tr, \emptyset) &= false && \text{gdy } Tr \neq \emptyset \\
GTS(\emptyset, S) &= false && \text{gdy } etf(s) \\
GTS(Tr, S) &= GTS(Tr, S \setminus s) && \text{gdy } \neg etf(s) \\
GTS(Tr, S) &= false && \text{gdy } etf(s) \wedge \neg \exists_{tr \in Tr} \bullet (source(tr) = s) \\
GTS(Tr, S) &= GTS(Tr \setminus tr, S \setminus s) && \text{gdy } etf(s) \wedge (tr \in Tr) \wedge (source(tr) = s)
\end{aligned} \tag{36}$$

Po wybraniu zbioru tranzycji następuje ich wzbudzenie i uruchomienie. Ponieważ dla środowiska czas trwania tranzycji jest zerowy, zatem należy tylko wejść do stanu docelowego każdej z tranzycji. Definiuje to poniższa funkcja:

Definicja 20. *Opuszczenie stanów* $LS(C, Tr)$ (**ang.** *Leave State*) konfiguracji C za pomocą zbioru tranzycji Tr będziemy nazywali konfigurację po przekształceniach:

$$LS(C, Tr) = (C \setminus Leaf0(C)) \cup \bigcup_{tr \in Tr} Entry(sink(tr)) \tag{37}$$

Opisane wcześniej operacje prowadzą do nowej konfiguracji. Zmiana ta zdefiniowana w sposób formalny wymaga dodatkowych oznaczeń:

Definicja 21. *Bezpośrednia zmiana konfiguracji środowiska* C w C' po czasie t i względem zbioru tranzycji Tr co zapisane zostanie jako $C \xrightarrow{t/Tr} C'$ następuje gdy spełniony jest poniższy warunek:

$$\begin{aligned}
C \xrightarrow{t/Tr} C' &\iff \\
(t = MinT(C)) \wedge (C_0 = ShTm0(C)) \wedge GTS(Tr, Leaf0(C_0)) \wedge (C' = LS(C_0, Tr))
\end{aligned} \tag{38}$$

Powyższy zapis oznacza, że w maszynie będącej w konfiguracji C upłynął czas t . W tym okresie nie została wzbudzona *żadna nowa tranzycja*. Upływ tego czasu wzbudził zbiór tranzycji Tr , które spowodowały że obecnie maszyna środowiska jest w konfiguracji C' .

Bezpośrednia zmiana konfiguracji powoduje wygenerowanie pewnego zbioru reakcji R . Zbiór ten określony zostanie funkcją zależną od wybranego zbioru tranzycji i w szczególnym przypadku może być zbiorem pustym.

Zanim podany zostanie sposób tworzenia tego zbioru należy wprowadzić pojęcie dodawania słów czasowych. W celu połączenia dwóch słów należy pamiętać, że w słowie o kolejności liter decyduje czas. Właściwość ta musi zostać zachowana po połączeniu, jak również właściwość ostrej mniejszości wartości czasu między kolejnymi literami słowa. Operacja dodawania słów będzie zdefiniowana odmiennie od zwykłego sumowania

zbiorowego. Ponieważ sumę dwóch zbiorów oznacza symbol $\cup$ do operacji dodawania zostanie użyty symbol $+$. W przypadku ciągu dodawań, przez analogię, użyty zostanie symbol Σ .

Definicja 22. Dodawanie $Q_1 + Q_2$, gdzie Q_1 i Q_2 są słowami pewnego JEC jest określone następująco:

$$Q_1 + Q_2 = \left\{ \langle E, t \rangle \mid \begin{aligned} & (\langle E, t \rangle \in Q_1 \wedge \neg \exists E'. \langle E', t \rangle \in Q_2) \\ & \vee (\langle E, t \rangle \in Q_2 \wedge \neg \exists E'. \langle E', t \rangle \in Q_1) \\ & \vee (E = E_1 \cup E_2 \wedge \langle E_1, t \rangle \in Q_1 \wedge \langle E_2, t \rangle \in Q_2) \end{aligned} \right\} \quad (39)$$

Definicja 23. Dodawanie wielu słów oznaczać będzie symbol Σ :

$$\begin{aligned} \sum_{i=1}^n Q_i &= Q_1 + Q_2 + \dots + Q_n \\ \text{oraz} \quad \sum_{i=1}^{\infty} Q_i &= Q_1 + Q_2 + \dots \end{aligned} \quad (40)$$

Można teraz przejść do zdefiniowania zbioru reakcji czasowych. Ponieważ czas trwania tranzycji w maszynie środowiska równa się zero, zatem wszystkie reakcje następują w chwili zero:

Definicja 24. Funkcja $Re(Tr)$ dla zbioru tranzycji Tr podaje zbiór reakcji czasowych jaki wygeneruje środowisko i wyrażona jest wzorem:

$$Re(Tr) = \bigcup_{tr \in Tr} \left\{ A \mid (tro(tr) = A) \wedge (A \neq \emptyset) \right\} \quad (41)$$

Jeśli jakiś element zmiany konfiguracji nie jest ważny w rozważaniach będzie on pomijany przy zapisie, czyli np. zapis $C \xrightarrow{t} C'$ oznacza, że nie istotne jest jakie tranzycje zostały wzbudzone.

Definicja 25. Zbiór wszystkich możliwych zmian konfiguracji środowiska ENV oznaczmy jako CH_{ENV} .

Na tak zdefiniowane środowisko należy nałożyć pewne ograniczenie, które nie będzie tu zapisane formalnie: w grafie nie może istnieć cykl zawierający tylko tranzycje z pustym zbiorem reakcji. Inaczej mówiąc nie może się zdarzyć, że środowisko będzie w nieskończoność generować *tylko* upływ czasu. Powodowałoby to, że może powstać ścieżka zmian konfiguracji biegnąca w nieskończoność, którą nie możnaby przedstawić w postaci poprawnego słowa w JEC. Gdyby przyjąć, że w tym momencie taka sytuacja zostanie zapisana jako skończony ciąg symboli czasowych, to byłoby to niezgodne z założeniem, że taki ciąg oznacza zatrzymanie się maszyny w skończonym czasie.

7.3 Generowanie JEC przez diagram stanów reprezentujący środowisko

Maszyna stanowa środowiska ENV na początku jest w jakiejś konfiguracji początkowej $C_0 = C_0^{ENV}$. Po upływie czasu t_1 następuje zmiana konfiguracji:

$$C_0 \xrightarrow{t_1/Tr_1} C_1 \quad (42)$$

Zmiana ta powoduje wygenerowanie zbioru zdarzeń $Re(Tr_1) = R_1$. Po przejściu do nowej konfiguracji i upływie kolejnego odcinka czasowego t_2 następuje kolejna zmiana $C_1 \rightarrow C_2$ itd. Proces ten odbywa się bądź w nieskończoność, bądź aż do osiągnięcia konfiguracji, z której nie nastąpi żadna zmiana konfiguracji.

Aby zastosować definicję dziedziczenia zachowań ze wzoru (19) należy podać jaki JEC ze wzoru (11) generuje maszyna środowiska wyrażona diagramem stanów. Aby to wykonać należy wprowadzić kolejne pojęcia.

Bardzo podobnie jak JEC, przez $ET_{ENV}^\emptyset$ oznaczony zostanie zbiór wszystkich ciągów (wyrażonych jako zbiory) par: zbiór zdarzeń i czas, możliwych do wygenerowania przez środowisko ENV . Zbiór ten może zawierać zarówno ciągi nieskończone jak i skończone. Różnica w stosunku do języka jest taka, że zbiór zdarzeń może być zbiorem pustym. Zatem:

$$ET_{ENV}^\emptyset \subseteq 2^{2^{E_{ENV} \times Time}} \quad (43)$$

Zbiór można podzielić jest na dwa rozłączne zbiory $ET_{ENV,*}^\emptyset$ i $ET_{ENV,\infty}^\emptyset$, które zawierają odpowiednio zbiory skończone i nieskończone. Oczywiście:

$$ET_{ENV}^\emptyset = ET_{ENV,*}^\emptyset \cup ET_{ENV,\infty}^\emptyset \quad (44)$$

Jednak ponieważ ostatecznie tylko niepuste zbiory zdarzeń są istotne (tylko one mogą być obserwowalne), zatem dalsze rozważania ograniczone będą do zdarzeń czasowych $et = \langle E, t \rangle$ takich, że $E \neq \emptyset$. Takie ciągi zdarzeń czasowych generowanych przez środowisko ENV i obserwowalnych oznacza zbiór ET_{ENV} :

$$ET_{ENV} \subseteq 2^{2^{E_{ENV} \setminus \emptyset \times Time}} \quad (45)$$

Tutaj również zbiór można podzielić na dwie rozłączne części, analogicznie na zbiory $ET_{ENV,*}$ i $ET_{ENV,\infty}$.

$$ET_{ENV} = ET_{ENV,*} \cup ET_{ENV,\infty} \quad (46)$$

Zbiorem obliczeń środowiska ENV nazwany zostanie zbiór $Eval_{ENV}^\emptyset$, będący sumą obliczeń dla ciągów skończonych i nieskończonych:

$$Eval_{ENV}^\emptyset = Eval_{ENV,*}^\emptyset \cup Eval_{ENV,\infty}^\emptyset \quad (47)$$

Zbiór obliczeń $Eval_{ENV,*}^\emptyset$ dla ciągów skończonych wygląda następująco:

$$Eval_{ENV,*}^\emptyset = \left\{ \left\langle C_0 \rightarrow C_1, \dots, C_{n-1} \rightarrow C_n \right\rangle \mid \begin{array}{l} C_0 = Conf_0^{ENV} \wedge \bigwedge_{1 \leq i \leq n} C_{i-1} \rightarrow C_i \in CH_{ENV} \wedge \neg \exists_{C'} C_n \rightarrow C' \in CH_{ENV} \end{array} \right\} \quad (48)$$

natomiast dla ciągów nieskończonych $Eval_{ENV,\infty}^\emptyset$

$$Eval_{ENV,\infty}^\emptyset = \left\{ \left\langle C_0 \rightarrow C_1, \dots, C_{n-1} \rightarrow C_n, \dots \right\rangle \mid \right. \\ \left. C_0 = Conf_0^{ENV} \wedge \forall_{i \in \mathbf{N}} C_{i-1} \rightarrow C_i \in CH_{ENV} \right\} \quad (49)$$

Możemy teraz formalnie zdefiniować pierwsze zbiory zbiorów zdarzeń czasowych:

$$ET_{ENV,*}^\emptyset = \left\{ \left\{ \langle E_1, t_1 \rangle, \langle E_2, t_2 \rangle, \dots, \langle E_n, t_n \rangle \right\} \mid \right. \\ \left(\exists \langle C_0 \xrightarrow{p_1/Tr_1} C_1, C_1 \xrightarrow{p_2/Tr_2} C_2, \dots, C_{n-1} \xrightarrow{p_n/Tr_n} C_n \rangle \in Eval_{ENV,*}^\emptyset \right. \\ \left. \wedge \forall_{1 \leq i \leq n} (t_i = \sum_{j=1}^i p_j \wedge E_i = Re(Tr_i)) \right) \left. \right\} \quad (50)$$

$$ET_{ENV,\infty}^\emptyset = \left\{ \left\{ \langle E_1, t_1 \rangle, \langle E_2, t_2 \rangle, \dots \right\} \mid \right. \\ \left(\exists \langle C_0 \xrightarrow{p_1/Tr_1} C_1, C_1 \xrightarrow{p_2/Tr_2} C_2, \dots \rangle \in Eval_{ENV,\infty}^\emptyset \right. \\ \left. \wedge \forall_{i \in \mathbf{N}} (t_i = \sum_{j=1}^i p_j \wedge E_i = Re(Tr_i)) \right) \left. \right\} \quad (51)$$

Definicja ze wzoru (8) nie dopuszcza pustego zbioru w słowie czasowym. Jednak podczas tworzenia słów mogą powstawać ciągi par składające się ze zbiorów pustych. Do usuwania par z pustym zbiorem posłużmy poniższą funkcją.

Definicja 26. : Funkcja $ThEm(Q)$ (**ang.** ***Th**row **Em**pty*) przeszedłca ciąg par Q w poprawne słowo poprzez usunięcie par z pustymi literami:

$$ThEm(Q) = \left\{ \langle E, t \rangle \mid \langle E, t \rangle \in Q \wedge E \neq \emptyset \right\} \quad (52)$$

Teraz można formalnie zdefiniować zbiór ciągów zdarzeń obserwowalnych środowiska ENV.

$$ET_{ENV} = \left\{ cet \mid (\exists cet_x \in ET_{ENV}^\emptyset. ThEm(cet_x) = cet) \right\} \quad (53)$$

Definicja 27. Środowisko ENV wyrażone za pomocą JEC jest parą składającą się ze zbioru OU_{ENV} maszyny UML oraz wcześniej zdefiniowanego zbioru ciągów zdarzeń obserwowalnych środowiska zbiór ET_{ENV} , czyli

$$ENV = \langle OU_{ENV}, ET_{ENV} \rangle \quad (54)$$

8 Model zachowania obiektu

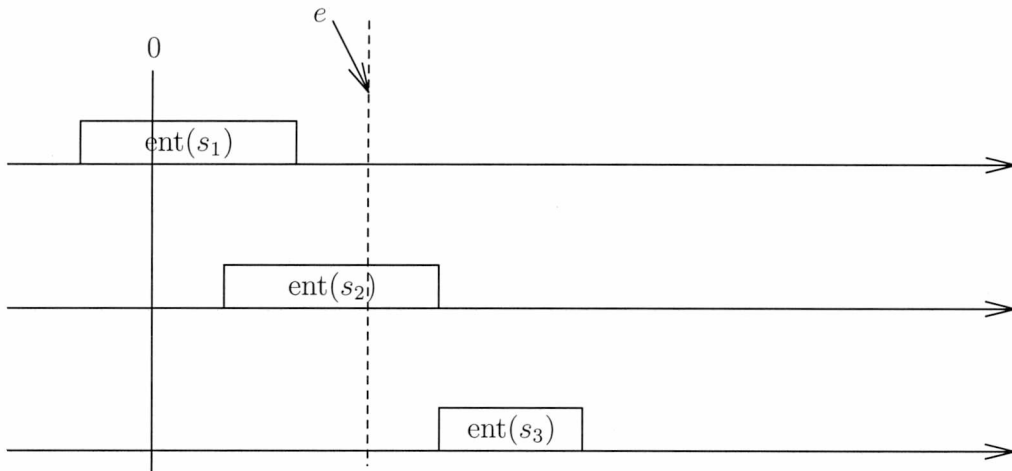
8.1 Właściwości

Podobnie jak dla środowiska również dla obiektu we wzorze (25) istnieją pewne ograniczenie. Dla każdego wierzchołka niekońcowego b zachodzi $ent(b) > 0$ oraz $ext(b) > 0$. Żaden stan końcowy s nie posiada akcji wejściowej i wyjściowej, stąd zachodzi wtedy $ent(s) = 0$ oraz $ext(s) = 0$. Dla dowolnego łuku tr zachodzi: $trt(tr) > 0$ oraz $tre(tr) \neq \emptyset$. Drugi warunek oznacza, że w maszynie obiektu nie ma tranzycji wzbudzanej zakończeniem aktywności wewnętrznej. Ograniczenie to można znieść, spowodowałoby to jednak skomplikowanie zapisu, późniejszych warunków itp.

Z założenia obiekt ma posiadać zachowanie zdeterminowane, czyli nie może posiadać w żadnym stanie dwóch tranzycji wzbudzanych tym samym zdarzeniem. Gdyby dopuścić niedeterminizm w porównywanym maszynach, zwiększyłoby to lawinowo rozmiar problemu oraz uniemożliwiłoby wyrażenie maszyny jako transformator JEC.

8.2 Operacje na diagramie stanów obiektu

Podobnie jak dla środowiska konfiguracją obiektu będzie zestaw aktywnych stanów wraz z pewnymi czasami. Tym razem będą to czasy zakończenia akcji wejściowej do nich liczone względem ostatniego zdarzenia ze środowiska.



Rysunek 24: Przykładowa konfiguracja maszyny obiektu

Definicja 28. Konfiguracją K maszyny obiektu jest n -tką:

$$K = \left\{ \langle s_1, t_1 \rangle, \langle s_2, t_2 \rangle, \dots, \langle s_n, t_n \rangle \right\}$$

gdzie dla każdego $i \in \{1, \dots, n\}$

$$s_i \in \text{Box}N \text{ oznacza stan aktywny}$$

$$t_i \in \text{Time} \text{ jest chwilą zakończenia akcji wejściowej stanu } s_i$$
(55)

Należy wyjaśnić, że obecność stanu w konfiguracji nie oznacza, że w chwili zerowej maszyna jest w tym stanie. Ta obecność oznacza, że w chwili zerowej (na podstawie upływu czasu i zdarzeń ze środowiska) wiadomo, że maszyna musiała wejść do tego stanu, lub będzie do tego zmuszona w przyszłości (np. poprzez rekurencyjne wchodzenie do podstanów, co trwa pewien czas). Dzięki takiemu podejściu nie trzeba w konfiguracji zapamiętywać dodatkowych informacji, np. tranzycji które zostały wzbudzone, ale jeszcze nie uruchomione, jak i wejść do stanów potomnych, które jeszcze się w bieżącej chwili nie rozpoczęły, ale muszą zostać wykonane.

Przyjęto założenie, że po rozpoczęciu akcji wejściowej do stanu, stan ten może już reagować na zdarzenia ze środowiska i ewentualnie wzbudzać (ale jeszcze nie uruchamiać) odpowiednie tranzycje. W związku z tym, jeśli czas z pary $\langle s, t \rangle \in K$ jest mniejszy lub równy $ent(s)$, to znaczy, że stan może, w rozważanej chwili, zareagować na odpowiednie zdarzenie wyprowadzające z niego. Jeśli jego wartość jest większa, to znaczy, że dopiero po upływie tego czasu będzie w stanie zareagować na zdarzenie (maszyna jeszcze nie zaczęła wchodzić do tego stanu). W przykładzie z rys. (24) stan s_1 w tej konfiguracji zawsze może zareagować na zdarzenie, natomiast stany s_2 i s_3 dopiero po pewnym czasie. Jeśli zdarzenie e napłynie w czasie pokazanym na rysunku, to stan s_2 będzie w stanie już na nie zareagować, a stan s_3 jeszcze nie.

Wprowadzone zostaną pomocnicze oznaczenia i funkcje:

Oznaczenie 3. Dla uproszczenia zapisu, $s \in K$ będzie oznaczać, że w konfiguracji K istnieje para z tym stanem:

$$(s \in K) \iff \exists t \bullet (\langle s, t \rangle \in K) \quad (56)$$

Oznaczenie 4. Funkcja czasu do zakończenia wejścia $tte(K, s)$ (**ang.** *Time To end of Entry*) w konfiguracji K stanu s oznaczać będzie czas z jakim dany stan jest w parze w danej konfiguracji:

$$tte(K, s) = t \iff (\langle s, t \rangle \in K) \quad (57)$$

Powodem zmiany konfiguracji może być zarówno upływ czasu jak i zdarzenie ze środowiska. W celu ścisłego zdefiniowania poprawnej zmiany konfiguracji w maszynie obiektu zdefiniowane zostanie kilka funkcji. Analogicznie jak dla środowiska najpierw wprowadzona zostanie funkcja wyznaczająca konfigurację stanów, jaka powstanie po wejściu do stanu. Jednak w tym przypadku moment rozpoczęcia wejścia do stanu uzależniony jest od czasu:

Definicja 29. Wejściem $Entry(s, t)$ do stanu s w chwili t nazywana jest konfiguracja jaka powstałaby po zakończeniu wejścia do danego stanu i wszystkich stanów potomnych;

$$Entry(s, t) = \left\{ \langle s, ent(s) + t \rangle \right\} \cup \bigcup_{s' \in childB(s)} Entry(s', ent(s) + t) \quad (58)$$

Początkowa konfiguracja maszyny wyznaczona jest przez wejście do korzenia:

Definicja 30. Konfiguracja początkowa maszyny stanowej MS oznaczona zostanie przez K_0^{MS} i jest ona wejściem do korzenia grafu maszyny obiektu:

$$K_0^{MS} = Entry(Root(\langle BoxN, ChildB \rangle), 0) \quad (59)$$

Jak zostało wspomniane wcześniej powodem wyjścia ze stanu może być upływ czasu, a konkretnie uruchomienie przeterminowanej tranzycji czasowej. W celu stwierdzenia, czy w pewnym momencie nie zachodzi taka sytuacja wprowadzona zostanie następująca funkcja pomocnicza:

Definicja 31. Przedziałem czasu przeterminowania $afp(s)$ (**ang. after period**) stanu s maszyny obiektu oznaczona zostanie wartość zdefiniowaną następująco:

$$afp(s) = \begin{cases} t & \text{gdy } \exists_{tr}. (source(tr) = s \wedge tre(tr) = \{after(t)\}) \\ \infty & \text{w p. p.} \end{cases} \quad (60)$$

Stany w rozważanej konfiguracji, które powinny uruchomić tranzycję czasową, jeśli upłynął czas równy t będą nazywane *stanami z aktywną tranzycją czasową* i wyznaczone przez funkcję $Swatt$:

Definicja 32. Zestaw stanów z konfiguracji K , które powinny wzbudzić tranzycje czasowe jeśli do momentu t nie pojawiło się żadne zdarzenie ze środowiska definiuje funkcja $Swatt$ (**ang. States with active time transition**):

$$Swatt(K, t) = \{s | (s \in K) \wedge tte(K, s) + afp(s) \leq t\} \quad (61)$$

Uruchamianie tranzycji w takich stanach nie następuje od razu wraz z upływem czasu $afp(s)$, gdyż najpierw należy pozakazać wewnętrzne aktywności. Inną sprawą jest to, że uruchomienie wcześniejszej tranzycji w stanie rodzicielskim spowoduje, że czasowa tranzycja wewnętrzna nie będzie uruchomiona, gdyż wcześniej nastąpi proces wychodzenia ze stanów potomnych. Poza tym, jak wcześniej przyjęto, pierwszeństwo wykonania będzie miała tranzycja wewnętrznego stanu. Zatem zanim będą analizowane zmiany w konfiguracji spowodowane wystąpieniem zdarzenia nieczasowego należy wykonać wszystkie przeterminowane tranzycje czasowe zaczynając od najwcześniejszych i najbardziej wewnętrznych. Po wejściu do nowych stanów może się okazać, że również

i w nich nastąpiło wzbudzenie tranzycji czasowych.

Przy wychodzeniu ze stanu należy najpierw wyjść ze stanów potomnych, które są w konfiguracji. Momentem rozpoczęcia wychodzenia ze stanu dla stanu bez aktywnych podstanów może być albo moment nadejścia zdarzenia, jeśli stan zakończył wcześniej akcję wejściową, albo w przeciwnym wypadku moment zakończenia tej akcji. Potrzebne jest poniższe oznaczenie:

Oznaczenie 5. Funkcja $mat(K, s, t)$ (**max tte**) oznacza najwcześniejszy moment, po jakim jest możliwe wychodzenie ze stanu s :

$$mat(K, s, t) = max(tte(K, s), t) \quad (62)$$

O ile dla stanu prostego czasem wyjścia jest wartość funkcji mat , o tyle dla stanów złożonych należy najpierw wyjść z podstanów, a dopiero później można rozpocząć wychodzenie z rozważanego stanu. Ten rekurencyjny proces jest zawarty w ostatecznym wzorze na czas wyjścia z dowolnego stanu:

Definicja 33. Czasem wyjścia $ExitT(K, s, t)$ (**ang. Exit Time**) ze stanu s w konfiguracji K w momencie t jest to wartość czasu zdefiniowana następująco:

$$ExitT(K, s, t) = \begin{cases} t & \text{gdy } s \notin K \\ mat(K, s, t) + ext(s) & \text{gdy } s \in Leaf(K) \\ \max_{s' \in childB(s)} ExitT(K, s', mat(K, s, t)) + ext(s) & \text{w p. p.} \end{cases} \quad (63)$$

Operację uruchomienia wszystkich przeterminowanych tranzycji czasowych zapisana zostanie za pomocą funkcji $RunTT$ i będzie się składać z odpowiedniego wykonania funkcji $RunMinTT$. W celu pozbycia się niejednoznaczności wykorzystana będzie również waga tranzycji.

Oznaczenie 6. Niech $tafp(K, s)$ (**tte afp**) oznacza moment czasu w którym nastąpi przeterminowanie wyprowadzające ze stanu s w konfiguracji K , $tra(s)$ (**ang. transition after**) oznaczać będzie tranzycję czasową zaczynającą się w stanie s i oznaczoną dla stanów, które taką tranzycję posiadają. Podobnie $nsa(s)$ (**ang. next state after**) oznaczać będzie stan docelowy tranzycji czasowej zaczynającej się w stanie s . Formalnie:

$$\begin{aligned} tafp(K, s) &= tte(K, s) + afp(s) \\ tra(s) &= tr \iff \exists_{x \in Time} \bullet (source(tr) = s \wedge ev(tr) = \{after(x)\}) \\ nsa(s) &= sink(tra(s)) \end{aligned} \quad (64)$$

Po uruchomieniu tranzycji ze stanu s , należy ten stan usunąć z konfiguracji. Oznaczenie tego usuwania znajduje się poniżej.

Oznaczenie 7. Usunięciem z konfiguracji K stanu s oznacza usunięcie z niego pary, w której występuje stan s :

$$K \setminus s = K \setminus \{\langle s, tte(K, s) \rangle\} \quad (65)$$

Najpierw należy uruchomić przeterminowane tranzycje czasowe. Istnienie takich tranzycji sprawdza poniższą funkcję.

Definicja 34. Funkcja $MinTm(K)$ (ang. *Minimal Time*) podaje, w którym najwcześniejszym momencie czasowym może nastąpić przeterminowanie tranzycji w stanie z konfiguracji K .

$$MinTm(K) = \min_{s \in K} (tafp(K, s)) \quad (66)$$

Natomiast tranzycję, która w pierwszej kolejności powinna być uruchomiona wyraża następująca funkcja.

Definicja 35. Funkcja $MinTmTr(K)$ (ang. *Minimal Time Transition*) podaje, która tranzycja spośród tranzycji czasowych powinna być wzbudzona w konfiguracji K . Oznaczona tylko, gdy $MinTm(K) \neq \infty$. Wprowadzone oznaczenia: $s = source(tr)$

$$\begin{aligned} MinTmTr(K) &= tr \\ &\iff \\ \forall_{s' \in K} \quad &(s' \neq s \Rightarrow (tafp(K, s') > tafp(K, s) \vee s \in childB^*(s')) \vee \\ &(s \notin childB^*(s') \wedge s' \notin childB^*(s) \wedge trw(tr(s')) > trw(tr(s)))) \end{aligned} \quad (67)$$

Oznacza to, że stan s jest tym, który ma największe przeterminowanie, i nie jest rodzicem innego takiego stanu, a jeśli i to byłoby niejednoznaczne, to o wyborze stanu decyduje waga sprawdzanej tranzycji czasowej.

Poniżej znajduje się funkcja generująca parę elementów wcześniej zdefiniowanych.

Definicja 36. Funkcję podającą parę: następująca przeterminowana tranzycja oraz moment jej wzbudzenia w konfiguracji K oznaczać będzie funkcja $GenMinTT(K)$.

$$GenMinTT(K) = \{\langle MinTmTr(K), MinTm(K) \rangle\} \quad (68)$$

Jeśli wartość funkcji $MinTm(K)$ jest mniejsza od wartości rozpatrywanego czasu t , to istnieje w konfiguracji K stan, w którym przed momentem t nastąpi wzbudzenie odpowiedniej tranzycji czasowej. Uruchomienie jej spowoduje zmianę konfiguracji: opuszczenie stanu źródłowego tranzycji, uruchomienie jej i wejście do jej stanu docelowego. Zmiany te opisuje kolejna funkcja:

Definicja 37. Funkcja $RunMinTT(K, t)$ (ang. *Run Minimal Time Transitions*) oznacza zmiany w konfiguracji K w momencie t poprzez uruchomienie tranzycji czasowych o najwcześniejszym czasie wzbudzenia. Niech $tr = MinTmTr(K)$, $s = source(tr)$, $s' = target(tr)$:

$$RunMinTT(K, t) = K \setminus s \setminus childB^*(s) \cup Entry(s', ExitT(K, s, tafp(K, s)) + trt(tr)) \quad (69)$$

Te zmiany należy wykonywać dopóty, dopóki istnieją jakieś przeterminowane tranzycje, zatem ostatecznie uruchomienie wszystkich tranzycji czasowych wyraża poniższy wzór rekurencyjnym.

Definicja 38. Konfigurację powstałą po uruchomieniu wszystkich przeterminowanych tranzycji czasowych w konfiguracji K w momencie t oznaczać będzie funkcja $RunTT(K, t)$.

$$RunTT(K, t) = \begin{cases} RunTT(RunMinTT(K, t), t) & \text{gdy } Swatt(K, t) \neq \emptyset \\ K & \text{w p. p.} \end{cases} \quad (70)$$

W celu późniejszego generowania zbioru reakcji maszyny wprowadzona zostanie funkcja generująca wszystkie tranzycje czasowe do ustalonego momentu t .

Definicja 39. Funkcję generującą zbiór par (tranzycja czasowa i moment jej wzbudzenia) wszystkich wzbudzonych tranzycji czasowych w konfiguracji K do momentu momentu t oznaczać będzie funkcja $GenTT(K, t)$.

$$GenTT(K, t) = \begin{cases} GenTT(RunMinTT(K, t), t) \cup GenMinTT(K) & \text{gdy } MinTm(K) < t \\ \emptyset & \text{w p. p.} \end{cases} \quad (71)$$

Po wzbudzeniu wszystkich przeterminowanych tranzycji należy wzbudzić tranzycje powodowane zdarzeniami. Ponieważ maszyna ma być zdeterminowana, zatem należy podać sposób wyboru tranzycji. W opisie diagramów stanów UML są podane warunki, jakim musi podlegać zbiór wybranych tranzycji, jednak w przypadku dalszej niejednoznaczności, dalsze zasady wyboru pozostawione są do doprecyzowania. Najpierw zostanie opisanych kilka pomocniczych funkcji:

Oznaczenie 8. Funkcja logiczna $acc(s, e)$ oznacza istnienie tranzycji ze stanu s wzbudzanej zdarzeniem e

$$acc(s, e) = \exists_{tr \in Tr} . (source(tr) = s \wedge tre(tr) = \{e\}) \quad (72)$$

Zbiór stanów źródłowych zbioru tranzycji T oznacza $ps(T)$:

$$ps(T) = \bigcup_{tr \in T} \{source(tr)\} \quad (73)$$

Zbiór zdarzeń wzbudzających zbiór tranzycji T oznacza $ev(T)$:

$$ev(T) = \bigcup_{tr \in T} tre(tr) \quad (74)$$

Spośród stanów musimy wybrać te, które w danym momencie t mogą zareagować na zdarzenie. Stany takie oznaczone zostaną przez $AS(K, t)$

Definicja 40. *Aktywne stany* $AS(K, t)$ (**ang.** **A**ctive **S**tate) z konfiguracji K w momencie t oznaczają te stany, które w rozważanym momencie t mogą zareagować na dowolne zdarzenie zewnętrzne:

$$AS(K, t) = \{s | (s \in K) \wedge tte(s) - ent(s) \leq t\} \quad (75)$$

W stanach aktywnych mogą zostać wzbudzone tranzycje. O tym, które tranzycje mogą być wzbudzone decydują zdarzenia ze środowiska. Jednak nie jest to informacja wystarczająca, aby stwierdzić, które zestawy wybrane do wzbudzenia są poprawne, a które nie. Istotny zestaw warunków jaki musi spełniać zestaw tranzycji podaje specyfikacja UML. W postaci formalnej zostało to zawarte w poniższej definicji.

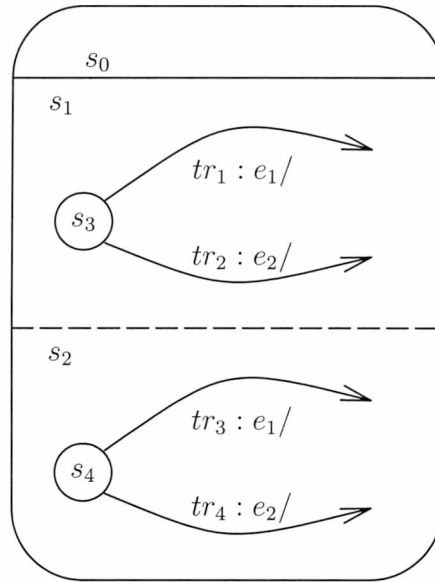
Definicja 41. Zbiór tranzycji Tr jest *dopuszczalny* dla konfiguracji K w momencie t i dla zdarzeń E , co oznaczone zostanie jako $AT(Tr, K, t, E)$ (**ang.** **A**dmissible **T**ransitions), wtedy gdy spełnia wszystkie poniższe warunki:

1. $\forall_{tr \in Tr} (tre(tr) \subseteq E)$
2. $\forall_{tr \in Tr} (source(tr) \in AS(K, t))$
3. $\forall_{tr, tr' \in Tr} (source(tr) = source(tr') \Rightarrow tr = tr')$
4. $\forall_{tr \in Tr} \neg \exists_{s \in AS(K, t)} (s \in childB^*(source(tr)) \wedge acc(s, tre(tr)) \wedge s \notin ps(Tr))$
5. $\forall_{e \in E} (e \notin ev(Tr) \Rightarrow \neg \exists_{s \in AS(K, t)} (s \notin ps(Tr) \wedge acc(s, e)))$

Warunek pierwszy oznacza, że każda tranzycja może być odpalana tylko zdarzeniem ze zbioru E . Drugi mówi, że mogą to być tylko tranzycje zaczynające się w aktywnych

stanach. Trzeci, że z każdego takiego aktywnego stanu może być odpalona maksymalnie jedna tranzycja. Jeśli to samo zdarzenie może uruchamiać tranzycje w jakimś stanie i w jego potomku, to priorytet ma tranzycja w potomnym stanie, co oznacza warunek czwarty. Jest to warunek priorytetu tranzycji wewnętrznej zgodny ze specyfikacją języka UML. Piąty mówi, jeśli w stanie s istnieje tranzycja wzbudzana zdarzeniem e ze zbioru E , to albo musi zostać ta tranzycja wybrana, albo musi istnieć inna wybrana tranzycja wychodząca z tego stanu.

W specyfikacji UML istnieje warunek uruchomienia tylko jednej tranzycji w ramach tej samej części równoległej. Jednak warunek ten nie dotyczy wzbudzania tranzycji. Jeśli dopuściliśmy tranzycje czasowe, to jak wcześniej było rozważane, może się zdarzyć przeterminowanie takiej tranzycji równocześnie w jakimś stanie jak i w jego stanie potomnym. Przyjęto założenie, że obydwie takie tranzycje powinny być wzbudzone, a rozwiązane to zostało poprzez preferowanie wewnętrznych tranzycji czasowych. Podobne założenie zostanie przyjęte przy rozważaniu tranzycji wzbudzanych różnymi zdarzeniami w tej samej chwili w jednej części równoległej. Oczywiście chociaż wzbudzone w tej samej chwili będą dwie tranzycje, to najpierw wykona się tranzycja wewnętrzna, a po odpowiednim czasie zewnętrzna.



Rysunek 25: Problem wyboru zestawu tranzycji

Jednak ten zestaw warunków nie określa do końca, które tranzycje zostaną uruchomione. Ponieważ z założenia maszyna obiektu ma być deterministyczna, zatem w przypadku jak na rys. 25 można wybrać poprawne zestawy: $\{tr_1, tr_3\}$, $\{tr_1, tr_4\}$, $\{tr_2, tr_3\}$, $\{tr_2, tr_4\}$. Należy zatem wprowadzić jakiś mechanizm powodujący zdeterminowany wybór spośród wielu zbiorów spełniających powyższe warunki ten najlepszy. Użyty do tego zostanie znak mniejszości i w jakiś sposób zdefiniowana odpowiednia relacja:

Oznaczenie 9. Zapis $Tr_1 < Tr_2$ oznacza, że zbiór relacji Tr_1 jest gorszy od zbioru Tr_2 . Relacja tak zdefiniowana powinna być relacją liniowego porządku.

Dalej zostanie podany przykładowy sposób definicji takiej relacji.

Pierwszym warunkiem decydowania o wyborze zbioru tranzycji będzie liczba tranzycji. Jeśli to nadal niejednoznacznie wybiera zbiór, zostaną użyte wagi tranzycji. Proponowany sposób, to porównanie największych wag dwóch zbiorów tranzycji. Jeśli są różne, to wybrany zostanie ten o wyższej wadze. Jeśli są równe, to do kolejnych porównań między zbiorami użyte zostaną pozostałe nie porównywane jeszcze tranzycje. I tak, aż do znalezienia tranzycji różniących się wagami.

Oznaczenie 10. Niech $maxw(Tr)$ oznacza maksymalną wagę spośród wszystkich wag zbioru tranzycji Tr :

$$maxw(Tr) = \max_{tr \in Tr} (trw(tr)) \quad (77)$$

Może teraz zdefiniować przykładową relację porządkującą:

$$\begin{aligned} Tr_1 < Tr_2 \iff & \\ & (|Tr_1| < |Tr_2|) \vee \left((|Tr_1| = |Tr_2|) \wedge (Tr_1 \neq \emptyset) \wedge (Tr_2 \neq \emptyset) \wedge \right. \\ & \left. ((maxw(Tr_1) < maxw(Tr_2)) \vee \exists_{w \in N} ((maxw(Tr_1) = maxw(Tr_2) = w) \wedge \right. \\ & \left. Tr_1 \setminus \{tr \in Tr_1 | trw(tr) = w\} < Tr_2 \setminus \{tr \in Tr_2 | trw(tr) = w\} \right) \right) \end{aligned} \quad (78)$$

Po tych rozważaniach można teraz zdefiniować funkcję zwracającą szukany zbiór tranzycji.

Definicja 42. Zbiór tranzycji $Tran(K, t, E)$ wzbudzanych w konfiguracji K w momencie t zbiorem zdarzeń E jest zdefiniowany następująco:

$$\begin{aligned} Tran(K, t, E) = Tr \iff & AT(Tr, K, t, E) \wedge \\ & \forall_{Tr'} \left(((Tr' \neq Tr) \wedge (AT(Tr', K, t, E)) \Rightarrow (Tr' < Tr)) \right) \end{aligned} \quad (79)$$

Po jednoznacznym wybraniu tranzycji do uruchomienia pozostaje je uruchomić. Kolejność uruchamiania zależy od położenia tranzycji w ramach stanów. Jeśli dwie tranzycje leżą w różnych częściach równoległych, to mogą być uruchamiane niezależnie. Jeśli jednak leżą w tej samej części, to najpierw muszą być uruchomione tranzycje wewnętrzne. Wprowadzone zostaną zatem funkcje potrzebne do określenia jak się zmieni konfiguracja po uruchomieniu wybranego zbioru tranzycji.

Definicja 43. Funkcja $RunOneET(K, t, tr)$ (ang. **Run One Event Transition**) podaje nową konfigurację powstałą po uruchomieniu tranzycji tr w momencie t w konfiguracji K :

$$\begin{aligned} RunOneET(K, t, tr) = & K \setminus source(tr) \cup \setminus childB^*(source(tr)) \\ & Entry(dest(tr), ExitT(K, source(tr), t) + trt(tr)) \end{aligned} \quad (80)$$

Nietrudno się zorientować, że ze zbioru tranzycji wzbudzonych najpierw muszą być

uruchomione tranzycje wewnętrzne, zanim uruchomiona zostanie tranzycja ze stanu zawierającego tranzycję wewnętrzną. Wprowadzona zostanie zatem funkcja określająca, czy dana tranzycja w zbiorze nie posiada tranzycji wewnętrznych stanu źródłowego.

Definicja 44. Funkcja logiczna $InTr(Tr, tr)$ (**ang. *Inner Transition***) zwraca prawdę, jeśli w zbiorze Tr nie istnieje tranzycja znajdująca się wewnątrz stanu źródłowego tranzycji tr :

$$InTr(Tr, tr) \iff \forall_{tr' \in Tr} . ((tr' \neq tr) \Rightarrow (tr \notin childB^*(source(tr')))) \quad (81)$$

Zdefiniować można teraz jak przekształci się konfiguracja po uruchomieniu wszystkich tranzycji z wybranego zbioru. Należy kolejno uruchomić wybrane tranzycje. W celu pozbycia się niejednoznaczności wykorzystana zostanie waga tranzycji do wyboru kolejności uruchomień. Wybór następnej tranzycji do wzbudzenia ze zbioru tranzycji Tr podaje poniższa funkcja:

Definicja 45. Funkcja $FiTr(Tr)$ (**ang. *First Transition***) podaje tranzycję, która w zbiorze Tr ma najwyższy priorytet wykonania:

$$FiTr(Tr) = tr \iff (tr \in Tr) \wedge InTr(Tr, tr) \wedge \neg \exists_{tr' \in Tr} . (InTr(Tr, tr') \wedge (trw(tr') > trw(tr))) \quad (82)$$

Możemy teraz zapisać uruchamianie wszystkich wybranych tranzycji.

Definicja 46. Nowa konfiguracja $RunET(K, t, Tr)$ (**ang. *Run Event Transition***) po uruchomieniu zbioru tranzycji Tr w momencie t w konfiguracji K jest zdefiniowana rekursywnie poniżej, gdzie $tr = FiTr(Tr)$:

$$\begin{aligned} RunET(K, t, \emptyset) &= K \\ RunET(K, t, Tr) &= RunET(RunOneET(K, t, tr), t, Tr \setminus \{tr\}) \end{aligned} \quad (83)$$

Pozostaje tylko przesunąć się do nowego miejsca na osi czasu. Jest to przesunięcie o czas t czasów wejścia wszystkich stanów w nowej konfiguracji. Dodanie czasu do konfiguracji definiuje poniższa funkcja.

Definicja 47. Dodanie $AdTm(K, t)$ czasu t do konfiguracji K oznacza dodanie wartości t do czasów wszystkich czasów wejść konfiguracji K :

$$AdTm(K, t) = \{\langle s, t_x + t \rangle \mid \langle s, t_x \rangle \in K\} \quad (84)$$

Ostatecznie należy zdefiniować bezpośrednią zmianę konfiguracji.

Definicja 48. Niech $K_x = RunTT(K, t)$. Bezpośrednią zmianą konfiguracji K na konfigurację K' , gdy po czasie t pojawił się ze środowiska zbiór zdarzeń E , oznaczona jako $K \xrightarrow{t/E} K'$, zdefiniowana jest następująco:

$$K \xrightarrow{t/E} K' \iff K' = AdTm(RunET(K_x, t, Tran(K_x, t, E)), -t) \quad (85)$$

Powyższy zapis oznacza, że od czasu rozpoczęcia poprzedniej zmiany konfiguracji upłynął czas t . W tym okresie nie została wzbudzona żadna nowa tranzycja zdarzeniowa (prócz tych z wcześniejszych zmian konfiguracji). Po upływie tego czasu ze środowiska napłynął zbiór zdarzeń E i spowodował uruchomienie pewnego zbioru tranzycji, które wygenerowały pewien zbiór reakcji czasowych. Zbiór ten zdefiniowany zostanie jako sumę dwóch zbiorów: reakcji podczas uruchomienia tranzycji czasowych oraz podczas tranzycji zdarzeniowych. Zbiór reakcji wywołanych uruchomieniem tranzycji czasowych utworzyć można na podstawie wzorów (69) oraz (70).

Definicja 49. Niech: $s_x = nsa(s)$, $tr = tra(s)$, $tr = MinTmTr(K)$, $s = source(tr)$, $s' = target(tr)$. $ReMinTT(K, t)$ (ang. **Reaction on Minimal Time Transitions**) oznacza reakcje w konfiguracji K w momencie t poprzez uruchomienie tranzycji czasowych o najwcześniejszym czasie wzbudzenia:

$$ReMinTT(K, t) = \langle tro(tr), Entry(s', ExitT(K, s, tafp(K, s)) + trt(tr)) \rangle \quad (86)$$

Wszystkie reakcje wywołane tranzycjami czasowymi wyraża wzór rekurencyjny:

Definicja 50. Reakcje powstałe po uruchomieniu wszystkich przeterminowanych tranzycji czasowych w konfiguracji K w momencie t wyznaczone są funkcją $ReTT(K, t)$:

$$ReTT(K, t) = \begin{cases} ReMinTT(K, t) + \\ ReTT(RunMinTT(K, t), t) & \text{gdy } Swatt(K, t) \neq \emptyset \\ \emptyset & \text{w p. p.} \end{cases} \quad (87)$$

Teraz zdefiniowany zostanie zbiór reakcji spowodowanych zdarzeniami nieczasowymi. Analogicznie do wzorów (80) i (83) powstają odpowiednie funkcje.

Definicja 51. Funkcja $ReOneET(K, t, tr)$ (ang. **Reaction of One Event Transition**) podaje reakcję powstałą po uruchomieniu tranzycji tr w momencie t w konfiguracji K :

$$ReOneET(K, t, tr) = \langle tro(tr), ExitT(K, source(tr), t) + trt(tr) \rangle \quad (88)$$

Definicja 52. Zbiór reakcji $ReET(K, t, Tr)$ (**ang.** *Reaction of Event Transitions*) po uruchomieniu zbioru tranzycji Tr w momencie t w konfiguracji K jest zdefiniowany rekursywnie poniżej, gdzie $tr = FiTr(Tr)$:

$$\begin{aligned} ReET(K, t, \emptyset) &= K \\ ReET(K, t, Tr) &= ReET(RunOneET(K, t, tr), t, Tr \setminus \{tr\}) + ReOneET(K, t, tr) \end{aligned} \quad (89)$$

Zatem wcześniej wspomniany zbiór reakcji to zbiór powstały poprzez poniższą funkcję.

Definicja 53. Zbiorem reakcji zmiany konfiguracji K , gdy po czasie t napłynął z zewnątrz zbiór zdarzeń E , będzie wynik funkcji $Re(K, t, E)$ zdefiniowej wzorem:

$$Re(K, t, E) = ReTT(K, t) + ReET(RunTT(K, t), t, E) \quad (90)$$

Czas w reakcjach powyższego zbioru jest względem konfiguracji K . Pozostało tylko wprowadzenie, analogicznie jak dla środowiska, pojęcia zbioru możliwych zmian konfiguracji.

Oznaczenie 11. Zbiór wszystkich możliwych zmian konfiguracji maszyny MS oznaczony zostanie jako CH_{MS} .

8.3 Opis działania maszyny stanowej obiektu

Na początku działania maszyna stanowa MS obiektu jest w pewnym stanie początkowym K_0^{MS} następnie po pewnym czasie t napływa zbiór zdarzeń E , który powoduje zmianę konfiguracji obiektu $K_0 \xrightarrow{t/E} K_1$. Jeśli zbiór E jest pusty, to znaczy, że wzbudzona zostanie jakaś tranzycja czasowa. Teraz ten moment napłynięcia zdarzeń uznajemy za czas 0. Po pewnym czasie t_2 liczonym względem pierwszego zbioru zdarzeń napływa drugi zbiór zdarzeń E_2 i następuje zmiana konfiguracji $K_1 \xrightarrow{t'/E'} K_2$. Proces ten biegnie bądź w nieskończoność bądź do momentu, gdy kolejna konfiguracja będzie pusta, czyli $K_x = \emptyset$, lub gdy powstała konfiguracja z której nie można otrzymać żadnej innej.

8.4 Diagram stanów maszyny jako transformator języka środowiska

Maszyna stanowa reagując na zdarzenia napływające ze środowiska generuje pewne reakcje. Na podstawie ciągu zdarzeń czasowych generuje pewne ciągi *reakcji czasowych* (**ang.** *Time Reaction*), dla tego samego ciągu zdarzeń czasowych zawsze ten sam ciąg reakcji czasowych. Jest zatem maszyna zdeterminizowaną środowiskiem.

Podobnie jak dla środowiska, aby zastosować definicję dziedziczenia zachowań ze wzoru (19) należy podać jak maszynę obiektu wyrażoną diagramem stanów ze wzoru (25) przekształcić na transformator JEC środowiska na JEC reakcji ze wzoru (12).

Należy wziąć pod uwagę fakt, że jeśli środowisko kończy wysyłanie zdarzeń, to po nąpłynięciu ze środowiska ostatniego zdarzenia zewnętrznego maszyna może nadal działać wykonując zmiany konfiguracji związane z upływem czasu i przeterminowaniem tranzykcji czasowych. Najpierw rozpatrzona zostanie sytuacja, gdy środowisko nie generuje już żadnych zdarzeń.

Definicja 54. Funkcja $F^\emptyset(K, t)$ dla konfiguracji K i momentu t podaje ciąg reakcji czasowych związanych z uruchomieniem odpowiednich tranzykcji czasowych. Niech $m = \text{MinTm}(K)$:

$$F^\emptyset(K, t) = \begin{cases} \emptyset & \text{gd } m = \infty \\ \text{TmEm}(\text{AdTm}(\text{Re}(K, m, \emptyset), t)) + F^\emptyset(K', t + m) & \text{w p. p.} \end{cases}$$

gdzie: $K \xrightarrow{m/\emptyset} K'$

(91)

Teraz można zdefiniować funkcję przekształcającą ciąg zdarzeń ze środowiska w ciąg zdarzeń wyjściowych. W tym celu dodatkowa funkcja rekursywna podana jest poniżej.

Definicja 55. Funkcja $F(CE, K, t)$ dla początkowej konfiguracji K w momencie t i ciągu zdarzeń czasowych SE definiuje ciąg reakcji czasowych maszyny i zdefiniowana jest wzorem rekursywnym:

$$\begin{aligned} F(\emptyset, K, t) &= F^\emptyset(K, t) \\ F(SE, K, t) &= \text{ThEm}(\text{AdTm}(\text{Re}(K, t_1 - t, E_1), t)) + F(\text{tail}(SE), K', t_1) \end{aligned}$$

gdzie:

$$\text{head}(SE) = \langle E_1, t_1 \rangle$$

$$K \xrightarrow{t_1 - t/E_1} K'$$
(92)

Wykonanie przekształcenia ciągów, to po prostu wykonanie funkcji F dla konfiguracji początkowej i chwili początkowej.

Definicja 56. Funkcja $f(SE)$ dla ciągu zdarzeń czasowych SE definiuje ciąg reakcji czasowych maszyny i zdefiniowana jest wzorem:

$$f(SE) = F(SE, K_0^{MS}, 0)$$
(93)

Wzory (92) i (91) wymagają kilku wyjaśnień. Ponieważ czasy w ciągu zdarzeń są czasami względem początku startu środowiska, natomiast w zmianach konfiguracji były rozpatrywane zawsze względem poprzedniej konfiguracji zatem należy wykonać odpowiednie odejmowanie. Ciągi reakcji też zawierają czasy względne, które ostatecznie należy zmienić na bezwzględne przez ich odpowiednie przesunięcie. Ostatecznie należy z tego ciągu usunąć pary z pustymi zbiorami.

W przedstawionych funkcjach zawarta jest sytuacja, gdy środowisko generuje zdarzenia w nieskończoność, a maszyna zakończy swoje działanie. Oznacza to, że od pewnego wykonania rekurencyjnego funkcji F reakcje będą zbiorami pustymi. Podczas wykonania

funkcji *ThEm* zostaną one usunięte z ciągu, który z nieskończonego zamieni się w skończony. Jeśli środowisko zakończy generowanie zdarzeń, należy jeszcze wykonać zmiany konfiguracji związane z upływem czasu.

9 Funkcje porównujące zachowania czasowe

W rozważaniach ogólnych założono, że funkcja porównująca musi posiadać tylko właściwość zgodności strukturalnej. Jednak w zastosowaniach praktycznych warunek ten jest zbyt ogólny i uniemożliwia znalezienie dobrego algorytmu. Dalej określone zostaną warunki jakie musi spełniać funkcja porównująca, aby była *dopuszczalna* dla proponowanego dalej algorytmu.

Wcześniej zostanie wprowadzona operacja dodawania czasu do ciągu.

Definicja 57. Funkcja $AdTm(Q, t_x)$ (**ang. *Add Time***) dodaje do czasu każdej reakcji w zbiorze Q wartość t_x :

$$AdTm(Q, t_x) = \{ \langle E, t + t_x \rangle \mid \langle E, t \rangle \in Q \} \quad (94)$$

Powyższa funkcja przesuwca wszystkie czasy reakcji o zadany przedział, zatem innymi słowy przesuwca punkt zerowy czasu, względem którego będą wyrażane czasy tych reakcji.

Definicja 58. Niech $W_1 = \langle \langle A_1, t_1 \rangle, \langle A_2, t_2 \rangle, \dots \rangle$ i $W_2 = \langle \langle A'_1, t'_1 \rangle, \langle A'_2, t'_2 \rangle, \dots \rangle$. Funkcja czasowa τ jest *dopuszczalna* jeśli spełnia następujące trzy właściwości:

1. Porównywanie ciągów czasowych nie zależy od punktu zerowego na osi czasu.

$$\forall_{W_1, W_2} \cdot \left(\forall_{t \in Time} \cdot \left(\tau(W_1, W_2) = \tau(AdTm(W_1, t), AdTm(W_2, t)) \right) \right) \quad (95)$$

2. Istnieje pewna stała oznaczona jako $Len(\tau)$, taka że jeśli dwa ciągi różnią się czasem na dowolnej pozycji o wartość większą niż $Len(\tau)$, to ciągi takie nie mogą być zgodne. Inaczej mówiąc, funkcja τ musi posiadać stałą $Len(\tau)$, taką że:

$$\forall_{W_1, W_2} \cdot \left(\exists_i \cdot (|t_i - t'_i| > Len(\tau)) \Rightarrow \neg \tau(W_1, W_2) \right) \quad (96)$$

3. Funkcja pozwala porównywać kolejne elementy ciągu parami, czyli:

$$\tau(W_1, W_2) = \tau(\langle \langle A_1, t_1 \rangle, \langle A'_1, t'_1 \rangle \rangle) \wedge \tau(\langle \langle A_2, t_2 \rangle, \langle A_3, t_3 \rangle, \dots \rangle, \langle \langle A'_2, t'_2 \rangle, \langle A'_3, t'_3 \rangle, \dots \rangle) \quad (97)$$

Pierwsza właściwość, jednorodności, oznacza że ciągi reakcji rozpatrywane są względem siebie, a nie względem jakiegoś punktu zerowego.

Druga oznacza, że jeśli minął czas $Len(\tau)$ i nie pojawił się w jednej z maszyn zbiór reakcji, który pojawił się w drugiej maszynie, to maszyny nie są zgodne. Innymi słowy,

nie można w nieskończoność czekać na reakcję w jednej z maszyn, gdy pojawiła się ona już w drugiej z nich. Takie podejście ma uzasadnienie praktyczne, gdyż obserwator obserwując jedną i drugą maszynę powinien "pozwalać" na niewielkie odstępstwa czasowe porównywanych reakcji.

Trzecia oznacza, że funkcja porównująca porównuje czasy wystąpień reakcji parami niezależne od siebie. Jeśli tylko jakaś pojedyncza para nie spełnia funkcji porównującej, to nie mogą jej spełniać żadne dwa ciągi posiadające na tych samych pozycjach taką parę.

Przykładem funkcji porównującej może być funkcja sprawdzająca, czy kolejne punkty czasowe są od siebie oddalone co najwyżej o δ jednostek (*funkcja porównująca dokładności δ*), którą została określona następująco (przyjmując wcześniejsze oznaczenia):

$$\tau_{\leq \delta}(W_1, W_2) \iff \forall_{1 \leq i \leq k}. (|t_i^1 - t_i^2| \leq \delta) \quad (98)$$

natomiast *funkcją porównującą dokładności zerowej* (inaczej *funkcją ścisłej dokładności*) oznaczoną przez $\tau_{=0}$ będzie funkcja:

$$\tau_{=0}(W_1, W_2) \iff \forall_{1 \leq i \leq k}. (t_i^1 = t_i^2) \quad (99)$$

Wprowadzone funkcje porównujące będą używane do porównywania ciągów reakcji czasowych. W ogólności funkcji takich jest nieskończenie wiele, jednak tylko pewne z nich mogą być użyteczne w dalszych badaniach dziedziczenia zachowań.

Autor rozważał zestaw warunków rozszerzających cechy 2. i 3. pozwalający poszerzyć klasę dopuszczalnych funkcji porównujących. Oznaczałoby to możliwość porównywania czasów nie parami, lecz x -krotnymi grupami. Podobnie parametr *Len* oznaczałby maksymalną dopuszczalną różnicę czasów między odpowiednimi elementami tych grup. Warunki te nie zmieniłyby w sposób istotny zasady działania proponowanego algorytmu.

10 Algorytm badania zgodności zachowań czasowych

10.1 Wstęp

Badając dziedziczenie zachowań dwóch maszyn względem jakiegoś środowiska (będącego również specyficzną maszyną), można tego dokonać poprzez śledzenie reakcji każdej z maszyn na pojawiające się zdarzenia. Jeśli maszyny mają w sobie pętle (ciąg tranzycji wychodzący z jednego stanu i do niego wracający), to należałoby porównywać ciągi nieskończone. Ponieważ środowisko jest również maszyną, zatem oczywistym wydaje się fakt, że istnieje skończona liczba kombinacji konfiguracji środowiska w stosunku do każdej z maszyn (i maszyn do siebie nawzajem). Jednak oprócz konfiguracji należy porównać ze sobą ciągi reakcji wygenerowane przez obydwie z maszyn. Do operowania na przedstawionych elementach (konfiguracjach i ciągach reakcji), wprowadzona zostanie odpowiednia notacja.

10.2 Operacje na ciągach reakcji

Wprowadzone pojęcie funkcji porównującej wymaga uzupełnienia o operacje, które zostaną użyte w algorytmie. Wprowadźmy najpierw pojęcie obcięcia słowa do momentu t .

Definicja 59. *Obcięciem słowa AC do momentu t nazywamy nowe słowo składające się tylko z tych liter czasowych, których czas jest mniejszy lub równy t :*

$$Cut(AC, t) = \{\langle A, t_x \rangle \mid (\langle A, t_x \rangle \in AC) \wedge (t_x \leq t)\} \quad (100)$$

Dla celów ujednolicenia późniejszych notacji zmieniony zostanie sposób zapisywanie, jaką wartość ma funkcja τ dla dwóch słów.

Oznaczenie 12. Słowo AC_1 ze słowem AC_2 jest *zgodne czasowo względem funkcji τ* co zostanie zapisane jako $AC_1 \stackrel{\tau}{=} AC_2$, gdy spełniony będzie warunek:

$$AC_1 \stackrel{\tau}{=} AC_2 \iff \tau(AC_1, AC_2) \quad (101)$$

Funkcja porównująca w różnych rozważaniach nie powinna brać pod uwagę aktualnych słów czasowych (ciągów reakcji czasowych), lecz tylko pewne ich fragmenty, stąd poniższe pojęcia.

Definicja 60. Słowo AC_1 ze słowem AC_2 jest *zgodne czasowo względem funkcji τ w momencie t* co zostanie zapisane jako $AC_1 \stackrel{\tau, t}{=} AC_2$, gdy spełniony będzie warunek:

$$AC_1 \stackrel{\tau, t}{=} AC_2 \iff \tau(Cut(AC_1, t), Cut(AC_2, t)) \quad (102)$$

Powyższa definicja oznacza zgodność od minus nieskończoności (na osi czasu) aż do momentu t . Jednak jeśli założyć, że do chwili t już żadne reakcje się nie pojawiają, to nie można stwierdzić niezgodności. Może się zdarzyć, że nowe reakcje w przyszłości spowodują zgodność czasową. Aby to wyrazić potrzeba nowych funkcji

Wprowadźmy funkcje określające pierwszy punkt czasowy w słowie oraz pierwszy zbiór liter.

Definicja 61. Funkcje pierwszego punktu czasowego FiT (**ang. *First Time***) oraz pierwszego zbioru liter FiL (**ang. *First Letter Set***): są następujące:

$$FiT(\{\langle A_1, t_1 \rangle, \langle A_2, t_2 \rangle, \dots\}) = t_1 \quad (103)$$

$$FiL(\{\langle A_1, t_1 \rangle, \langle A_2, t_2 \rangle, \dots\}) = A_1 \quad (104)$$

Można teraz zdefiniować potencjalną zgodność czasową.

Definicja 62. Słowo AC_1 ze słowem AC_2 jest *potencjalnie zgodne czasowo względem relacji τ w momencie t* co zostanie zapisane jako $AC_1 \stackrel{\tau, t}{\cong} AC_2$, gdy spełniony będzie warunek:

$$AC_1 \stackrel{\tau, t}{\cong} AC_2 \iff \exists_{AC'_1, AC'_2} ((Fit(AC'_1) > t) \wedge (Fit(AC'_2) > t) \wedge (AC_1 + AC'_1 \stackrel{\tau}{=} AC_2 + AC'_2)) \quad (105)$$

Powyższa definicja oznacza, że jeśli do obecnych ciągów reakcji można dodać takie ciągi reakcji, które zdarzą się po momencie t , że ciągi te będą zgodne względem funkcji τ , to wyjściowe ciągi są potencjalnie zgodne.

10.3 Pojęcie konfrontacji

Zanim algorytm zostanie zaprezentowany należy wprowadzić pojęcia konfrontacji (pierwotnej i wtórnej) oraz związanych z nimi funkcji.

Definicja 63. *Konfrontacja pierwotna* to szóstka $X = \langle K_1, AC_1, K_2, AC_2, V, t \rangle$ i oznacza sytuację po czasie t . W momencie t albo uruchomiła się tranzycja w środowisku, albo środowisko nie może już zmieniać swojej konfiguracji, natomiast w w momencie t wzbudziła się tranzycja czasowa w co najmniej jednej z maszyn MS_1 i MS_2 . W konfrontacji pierwotnej X maszyna MS_1 jest w konfiguracji K_1 , i podczas działania do czasu t wygenerowała te reakcje z ciągu AC_1 , których czas jest mniejszy od t , oraz wiadomo, że wygeneruje w przyszłości pozostałe reakcje z ciągu AC_1 . Analogiczne znaczenie dla maszyny MS_2 mają K_2 i AC_2 . Maszyna środowiska do tego czasu zmieniła się do konfiguracji V . Czas w ciągach AC_1 i AC_2 jest wyrażony względem początku działania porównywanych maszyn i środowiska, natomiast w konfiguracjach maszyn i środowiska względem momentu t .

Przez $ZKP(X, t_x, Tr)$ określona będzie zmiana konfiguracji pierwotnej. Zmiana ta może być spowodowana zmianą konfiguracji środowiska lub, gdy środowisko już się nie zmieni, uruchomieniem najwcześniejszej tranzycji czasowej w jednej z maszyn. Stąd poniższa definicja:

Definicja 64. Niech $X = \langle K_1, AC_1, K_2, AC_2, V, t \rangle$ Jeśli środowisko może zmienić swoją konfigurację poprzez $V \xrightarrow{t_x/Tr} V'$, to niech $R = Re(Tr)$. Jeśli nie może zmienić, to niech $t_x = \min(MinTm(K_1), MinTm(K_2)) \neq \infty$ oraz $R = \emptyset$. Niech $K_1 \xrightarrow{t_x/R} K'_1$, $K_2 \xrightarrow{t_x/R} K'_2$ $R_1 = Re(K_1, t_x, R)$, $R_2 = Re(K_2, t_x, R)$. *Zmianę konfrontacji pierwotnej* definiuje poniższy wzór:

$$ZKP(X, t_x, Tr) = \langle K'_1, AC_1 + AdTm(R_1, t), K'_2, AC_2 + AdTm(R_2, t), V', t + t_x \rangle \quad (106)$$

Jeśli środowisko nie zmieni w przyszłości swojej konfiguracji oraz $\min(MinTm(K_1), MinTm(K_2)) = \infty$, to funkcja ZKP jest nieokreślona.

Proponowany algorytm nie jest oparty w działaniu na konfrontacjach pierwotnych, lecz na konfrontacjach wtórnych, w których pamiętana jest tylko informacja niezbędna do stwierdzenia zachodzenia dziedziczenia zachowań.

Definicja 65. *Konfrontacja wtórna* (w skrócie *konfrontacja*) jest piątką $\langle K_1, AC_1, K_2, AC_2, V \rangle$ i oznacza sytuację albo po uruchomieniu się tranzycji w środowisku, albo, jeśli środowisko nie może już zmieniać swojej konfiguracji, po wzbudzeniu się tranzycji czasowej w co najmniej jednej z maszyn MS_1 i MS_2 . Ciągi AC_1 i AC_2 zawierają tylko te reakcje maszyn (odpowiednio MS_1 i MS_2), które są niezbędne do stwierdzenia zachodzenia zgodności w obecnej chwili i w przyszłości. W konfiguracjach K_1 i K_2 czasy stanów bez tranzycji czasowych są wyzerowane, jeśli w rozpatrywanym momencie nastąpiło już zakończenie akcji wejściowej. Czas we wszystkich elementach konfiguracji jest względem ostatniej zmiany powodującej wytworzenie tej konfrontacji.

Konfrontacja wtórna powstaje z konfrontacji pierwotnej poprzez następujące zmiany:

1. Przesunięcie czasu w ciągach reakcji AC_1 i AC_2 o wartość t .
2. Obcięcie tych ciągów do zdarzeń obserwowalnych.
3. Usunięcie z obydwóch powyższych ciągów tych początkowych części, które są zgodne z funkcją porównującą i których czas jest przed momentem zero (czyli przed przesunięciem przed momentem t).
4. Zmiana ujemnego czasu stanów, w konfiguracjach MS_1 i MS_2 , nie posiadających tranzycji czasowych na wartość zero.

Zanim zostanie wprowadzona formalna definicja tworzenia konfrontacji wtórnej, należy wprowadzić funkcję obcinającą w niej pary zgodnych reakcji czasowych OC .

Definicja 66. Obcięcie ciągów $OC(X)$ w konfrontacji wtórnej $X = \langle K_1, AC_1, K_2, AC_2, V \rangle$ określa poniższy wzór rekurencyjny:

$$OC(X) = \begin{cases} OC\left(\langle K_1, tail(AC_1), K_2, tail(AC_2), V \rangle\right) & \text{gdy } AC_1 \neq \langle \rangle \wedge AC_2 \neq \langle \rangle \wedge \\ & \wedge max(FiT(AC_1), FiT(AC_2)) \leq 0 \wedge \tau\left(\left\langle \langle A_1, t_1 \rangle, \langle A'_1, t'_1 \rangle \right\rangle\right) \\ X & \text{w p. p.} \end{cases} \quad (107)$$

Operacja OC modyfikuje ciągi AC_1 i AC_2 poprzez obcięcie pewnej wspólnej liczby ich elementów początkowych.

Funkcję wyzerowania odpowiednich czasów konfiguracji maszyny przedstawia poniższa definicja.

Definicja 67. Ustawienie czasów stanów z konfiguracji K , które nie posiadają zaczynających się w nich tranzycji czasowych, na wartość 0 oznacza funkcja $Set0(K)$:

$$Set0(K) = \{ \langle s, t \rangle \mid \langle s, t \rangle \in K \wedge ((aftp(s) < \infty) \vee (tte(s) > 0)) \} \cup \{ \langle s, 0 \rangle \mid s \in K \wedge ((aftp(s) = \infty) \wedge (tte(s) \leq 0)) \} \quad (108)$$

Zmiana konfrontacji pierwotnej na wtórną (*tworzenie wtórnej*) oznaczona będzie jako $TW(X)$. Formalna jej definicja znajduje się poniżej:

Definicja 68. Niech $X = \langle K_1, AC_1, K_2, AC_2, V, t \rangle$. Konfrontacją wtórną $X' = TW(X)$ powstałą z konfrontacji X jest konfrontacja powstała z poniższego wykonania funkcji:

$$TW(X) = OC\left(\langle Set0(K_1), AdTm(AC_1|_{ZO}, -t), Set0(K_2), AdTm(AC_2|_{ZO}, -t), V \rangle\right) \quad (109)$$

Warto zauważyć, że funkcja ta nie musi być różnowartościowa. Dla wielu konfrontacji pierwotnych może istnieć taka sama konfrontacja wtórna.

Funkcja zmiany konfiguracji wtórnej $ZKW(X, t_x, Tr)$ będzie, analogicznie jak dla zmiany konfiguracji pierwotnej, trochę inaczej wyglądała dla przypadku, gdy zmiana konfiguracji środowiska jest powodem tej zmiany, a inaczej, gdy powodem zmiany jest przeterminowanie tranzycji czasowej jednej z maszyn.

Definicja 69. Niech $X = \langle K_1, AC_1, K_2, AC_2, V \rangle$. Jeśli środowisko może zmienić swoją konfigurację poprzez $V \xrightarrow{t_x/Tr} V'$, to niech $R = Re(Tr)$. Jeśli nie może zmienić, to niech $t_x = \min(MinTm(K_1), MinTm(K_2)) \neq \infty$ oraz $R = \emptyset$. Niech $K_1 \xrightarrow{t_x/R} K'_1$, $K_2 \xrightarrow{t_x/R} K'_2$, $R_1 = Re(K_1, t_x, R)$, $R_2 = Re(K_2, t_x, R)$, $AC'_1 = AdTm(AC_1 + R_1|_{ZO}, -t_x)$, $AC'_2 = AdTm(AC_2 + R_2|_{ZO}, -t_x)$. Zmianę konfrontacji wtórnej definiuje poniższy wzór:

$$ZKW(X, t_x, Tr) = OC\left(\langle Set0(K'_1), AC'_1, Set0(K'_2), AC'_2, V \rangle\right) \quad (110)$$

Jeśli środowisko nie zmieni w przyszłości swojej konfiguracji oraz $\min(MinTm(K_1), MinTm(K_2)) = \infty$, to funkcja ZKW jest nieokreślona.

Jak wynika z definicji zmiana konfiguracji wtórnej polega na zmianach podobnych jak przy zmianie konfiguracji pierwotnej. Jednak należy jeszcze dokonać podobnych zmian jak przy tworzeniu konfiguracji wtórnej, czyli odciąć zgodne elementy ciągów reakcji i zmodyfikować czasy odpowiednich stanów konfiguracji maszyn obiektów.

10.4 Opis algorytmu

Algorytm polega na przeszukiwaniu wszczepionej przestrzeni konfrontacji wtórnych aż do napotkania niezgodnej z definicją DZ lub do czasu wytworzenia wszystkich możliwych konfrontacji zgodnych. Konfrontacje na początku mogą zależeć od zdarzeń napływających ze środowiska, a gdy środowisko przestanie generować zdarzenia, konfrontacje będą zależeć od upływu czasu.

Danymi wejściowymi dla algorytmu są trzy maszyny: jedna ENV - środowiska i dwie MS_1 i MS_2 - maszyny porównywanych obiektów. Oprócz tego jest wybrana relacja porównująca τ oraz zbiór zdarzeń obserwowalnych ZO . Maszyny są wyrażone diagramami

stanów języka UML.

W celu zapamiętania nowych konfrontacji w algorytmie istnieje kolejka **Q** konfrontacji jeszcze nie przebadanych oraz zbiór **X** konfrontacji poprawnie zweryfikowanych. Obydwie struktury na początku są puste. Kolejne kroki algorytmu wyglądają następująco:

1. Na początku algorytmu do kolejki **Q** trafia konfrontacja wtórna $X_0 = TW(\langle K_0^1, \emptyset, K_0^2, \emptyset, V_0, t \rangle)$, gdzie

$$\begin{aligned} K_0^1 & - \text{Konfiguracja początkowa maszyny } MS_1 \\ K_0^2 & - \text{Konfiguracja początkowa maszyny } MS_2 \\ V_0 & - \text{Konfiguracja początkowa środowiska } ENV \end{aligned} \quad (111)$$

2. Jeśli kolejka **Q** jest pusta to odpowiedź jest **TAK** - te maszyny są zgodne i następuje **koniec algorytmu**.
3. Pobranie konfrontacji X z kolejki **Q**. Niech jako $X = \langle K_1, AC_1, K_2, AC_2, V \rangle$. Jeśli $X \in \mathbf{X}$ to powrót do kroku 2. Jeśli nie, to dodać X do zbioru **X**. Jeśli $V = \emptyset$, to przejść do kroku 7.
4. Stworzyć zbiór **V** wszystkich możliwych konfiguracji środowiska osiągalnych z konfiguracji V :

$$\mathbf{V} = \left\{ V_i \mid \exists_{t_i} \bullet \exists_{Tr_i} \bullet (V \xrightarrow{t_i/Tr_i} V_i \in CH_{ENV}) \right\} \quad (112)$$

5. Dla każdego elementu $V' \in \mathbf{V}$ takiego, że $V \xrightarrow{t/Tr} V'$

- (a) Niech $X' = ZKW(X, t, Tr) = \langle K'_1, AC'_1, K'_2, AC'_2, V' \rangle$. Jeśli nie zachodzi warunek potencjalnej zgodności:

$$AC'_1 \stackrel{\tau_i 0}{\cong} AC'_2 \quad (113)$$

to odpowiedź jest **NIE** - maszyny te nie są zgodne i następuje **koniec algorytmu**.

- (b) Jeśli konfrontacja $X' \notin \mathbf{X}$ to wstawić ją na koniec kolejki **Q**.

6. Powrót do kroku 2.
7. Sprawdzenie możliwości stworzenia następnej konfiguracji poprzez wzbudzenie najbliższej przeterminowanej tranzycji czasowej w dowolnej z maszyn. Niech $t_x = \min(\text{MinTm}(K_1), \text{MinTm}(K_2))$.
8. Jeśli $t_x = \infty$, to znaczy, że w żadnej z maszyn nie pojawi się już żadna nowa reakcja. Sprawdzić, czy $AC_1 \stackrel{\tau}{=} AC_2$. Jeśli nie, to odpowiedź jest **NIE** - maszyny te nie są zgodne i następuje **koniec algorytmu**. Jeśli tak, to wrócić do kroku 2.
9. Jeśli $t_x \neq \infty$

- (a) Niech $X' = ZKW(X, t, \emptyset) = \langle K'_1, AC'_1, K'_2, AC'_2, V' \rangle$. Jeśli nie zachodzi warunek potencjalnej zgodności:

$$AC'_1 \stackrel{\tau, 0}{\cong} AC'_2 \quad (114)$$

to odpowiedź jest **NIE** - maszyny te nie są zgodne i następuje **koniec algorytmu**.

- (b) Jeśli konfrontacja $X' \notin \mathbf{X}$ to wstawić ją na koniec kolejki **Q**.

10. Powrót do kroku 2.

11 Właściwości algorytmu

11.1 Poprawność algorytmu

Algorytm uznaje się za poprawny wtedy gdy, *zatrzymuje się w skończonej liczbie kroków* i *zwraca poprawny wynik*, czyli w przypadku przedstawionego algorytmu daje odpowiedź TAK wtedy i tylko wtedy, gdy prawdą jest warunek (19). Tak jak wcześniej przyjęto, algorytm ma być poprawny przy przyjętej kategorii maszyn stanowych środowiska i porównywanych obiektów oraz przy ograniczeniach nałożonych na funkcję porównującą.

Aby sprawdzić niezachodzenie warunku dziedziczenia zachowań, należy znaleźć taki ciąg zdarzeń dla którego ciągu reakcji maszyny pierwszego i drugiego obiektu nie spełniają funkcji porównującej τ . W przypadku, gdy porównywane ciągi są nieskończone, musi w każdym z nich istnieć pewna i -ta reakcja czasowa, dla których funkcja porównująca jest niespełniona. Metoda, bezpośrednio wynikająca z definicji dziedziczenia zachowań, polegałaby na przeszukiwaniu wszczegółu grafu konfrontacji pierwotnych pamiętających reakcje każdej z maszyn obiektów na podstawie zdarzeń napływających ze środowiska. Jeśli zostanie znaleziona taka konfrontacja, że reakcje do chwili t napłynięcia ostatnio rozważanego zdarzenia nie będą potencjalnie zgodne w przyszłości, to znaleziony został przypadek falsyfikujący definicję dziedziczenia zachowań. Jednak tak skonstruowany algorytm w przypadku maszyn spełniających warunek dziedziczenia zachowań mógłby działać w nieskończoność. Zamiast operować na konfrontacjach pierwotnych, w algorytmie zostały użyte konfrontacje wtórne, zmiany konfrontacji wtórnych oraz badanie potencjalnej zgodności we względnym czasie 0. Należy zatem pokazać, że działanie na konfrontacjach wtórnych nie zmienia wartości odpowiednich potencjalnych zgodności, a dodatkowo powoduje, że liczba konfrontacji wtórnych wygenerowanych w trakcie działania algorytmu jest skończona.

W pierwszej części będzie udowodnione, że działanie na konfrontacjach wtórnych nie zmienia wartości odpowiednich potencjalnych zgodności. W drugiej pokazane zostanie, że jeśli algorytm się zatrzyma, to poda poprawny wynik, natomiast w ostatniej, że algorytm zawsze się zatrzyma.

Dla dalszych rozważań przydatne będą następujące lematy:

Lemat 1. Dla dowolnego $t \in Time$:

$$AdTm(AdTm(AC, t), -t) = AC \quad (115)$$

Dowód

Niech $AC = \langle \langle A_1, t_1 \rangle, \langle A_1, t_1 \rangle, \dots \rangle$. Zatem $AdTm(AC, t) = \langle \langle A_1, t_1 + t \rangle, \langle A_2, t_2 + t \rangle, \dots \rangle$. Ostatecznie $AdTm(AdTm(AC, t), -t) = \langle \langle A_1, t_1 + t - t \rangle, \langle A_2, t_2 + t - t \rangle, \dots \rangle = \langle \langle A_1, t_1 \rangle, \langle A_1, t_1 \rangle, \dots \rangle = AC$. ■

Lemat 2. Dla dowolnego $t \in Time$:

$$\left(AC_1 \stackrel{\tau, x}{\cong} AC_2 \right) \Leftrightarrow \left(AdTm(AC_1, t) \stackrel{\tau, x+t}{\cong} AdTm(AC_2, t) \right) \quad (116)$$

Dowód

Najpierw zostanie udowodniona implikacja:

$$\left(AC_1 \stackrel{\tau, x}{\cong} AC_2 \right) \Rightarrow \left(AdTm(AC_1, t) \stackrel{\tau, x+t}{\cong} AdTm(AC_2, t) \right) \quad (117)$$

Przyjęto oznaczenia $AC_1^2 = AdTm(AC_1, t)$ oraz $AC_2^2 = AdTm(AC_2, t)$. Jeśli lewa strona jest prawdą, to istnieją AC_1' i AC_2' , takie że $(FiT(AC_1') > x) \wedge (FiT(AC_2') > x)$ oraz $AC_1 + AC_1' \stackrel{\tau}{=} AC_2 + AC_2'$. Niech zatem $AC_1'' = AdTm(AC_1', t)$ oraz $AC_2'' = AdTm(AC_2', t)$. Ponieważ $(FiT(AC_1') > x) \wedge (FiT(AC_2') > x)$, zatem $(FiT(AC_1'') > x + t) \wedge (FiT(AC_2'') > x + t)$. Jeśli oznaczyć $AD_1 = AC_1 + AC_1'$, $AD_2 = AC_2 + AC_2'$ oraz $AD_1^2 = AC_1^2 + AC_1''$, $AD_2^2 = AC_2^2 + AC_2''$, to $AD_1^2 = AdTm(AD_1, t)$ oraz $AD_2^2 = AdTm(AD_2, t)$. Z właściwości 1 (95) funkcji porównującej wynika, że $AD_1 \stackrel{\tau}{=} AD_2 \Rightarrow AD_1^2 \stackrel{\tau}{=} AD_2^2$, czyli warunki definicji potencjalnej zgodności są spełnione zatem $AdTm(AC_1, t) \stackrel{\tau, x+t}{\cong} AdTm(AC_2, t)$ jest prawdą.

Dowód, że

$$\left(AdTm(AC_1, t) \stackrel{\tau, x+t}{\cong} AdTm(AC_2, t) \right) \Rightarrow \left(AC_1 \stackrel{\tau, x}{\cong} AC_2 \right) \quad (118)$$

Opiera się na lemacie 1 i pierwszej części dowodu, gdyż podstawiając za AC_1 wyrażenie $AdTm(AC_1, t)$ (analogicznie za AC_2 wyrażenie $AdTm(AC_2, t)$) i czas przesunięcia równy $-t$, natomiast za x wyrażenie $x - t$ otrzymujemy:

$$\begin{aligned} & \left(AdTm(AC_1, t) \stackrel{\tau, x-t}{\cong} AdTm(AC_2, t) \right) \Rightarrow \\ & \Rightarrow \left(AdTm(AdTm(AC_1, t), -t) \stackrel{\tau, x-t+t}{\cong} AdTm(AdTm(AC_2, t), -t) \right) = \\ & = \left(AC_1 \stackrel{\tau, x}{\cong} AC_2 \right) \end{aligned} \quad (119)$$

Co kończy dowód. ■

Lemat 3.

$$\begin{aligned}
 AC_1 \stackrel{\tau,0}{\cong} AC_2 \wedge AC_1 \neq \langle \rangle \wedge AC_2 \neq \langle \rangle \wedge \\
 \wedge \max(FiT(AC_1), FiT(AC_2)) \leq 0 \wedge \tau\left(\left\langle \langle A_1, t_1 \rangle \right\rangle, \left\langle \langle A'_1, t'_1 \rangle \right\rangle\right) \\
 \Downarrow \\
 tail(AC_1) \stackrel{\tau,0}{\cong} tail(AC_2)
 \end{aligned} \tag{120}$$

Dowód

Jeśli

$$AC_1 \stackrel{\tau,0}{\cong} AC_2 \tag{121}$$

to z definicji:

$$\exists_{AC'_1, AC'_2} \bullet \left(\tau(AC_1 + AC'_1, AC_2 + AC'_2) \wedge Fit(AC'_1) > 0 \wedge Fit(AC'_2) > 0 \right) \tag{122}$$

Jeśli oprócz tego $AC_1 \neq \langle \rangle \wedge AC_2 \neq \langle \rangle \wedge \max(FiT(AC_1), FiT(AC_2)) \leq 0$ to z definicji dodawania ciągów wynika, że pierwsze litery ciągów nie ulegną zmianie, a z właściwości 3 (97) funkcji porównującej wynika, że

$$\begin{aligned}
 \exists_{AC'_1, AC'_2} \bullet \left(\tau\left(\left\langle \langle A_1, t_1 \rangle \right\rangle, \left\langle \langle A'_1, t'_1 \rangle \right\rangle\right) \wedge \tau(tail(AC_1) + AC'_1, tail(AC_2) + AC'_2) \wedge \right. \\
 \left. \wedge Fit(AC'_1) > 0 \wedge Fit(AC'_2) > 0 \right)
 \end{aligned} \tag{123}$$

A to zgodnie z definicją oznacza, że:

$$tail(AC_1) \stackrel{\tau,0}{\cong} tail(AC_2) \tag{124}$$

co kończy dowód. ■

Twierdzenie 1. Niech $X = \langle K_1, AC_1, K_2, AC_2, V, t \rangle$. Jeśli $X' = TW(X) = \langle K'_1, AC'_1, K'_2, AC'_2, V \rangle$, to

$$AC_1|_{ZO} \stackrel{\tau,t}{\cong} AC_2|_{ZO} \Leftrightarrow AC'_1 \stackrel{\tau,0}{\cong} AC'_2 \tag{125}$$

Dowód

Najpierw zostanie udowodnione, że:

$$AC_1|_{ZO} \stackrel{\tau,t}{\cong} AC_2|_{ZO} \Rightarrow AC'_1 \stackrel{\tau,0}{\cong} AC'_2 \tag{126}$$

Z

$$AC_1|_{ZO} \stackrel{\tau,t}{\cong} AC_2|_{ZO} \tag{127}$$

na podstawie wzoru (116)

$$AdTm(AC_1|_{ZO}, -t) \stackrel{\tau,0}{\cong} AdTm(AC_2|_{ZO}, -t) \quad (128)$$

Oznaczmy $AD_1 = AdTm(AC_1|_{ZO}, -t)$ oraz $AD_2 = AdTm(AC_2|_{ZO}, -t)$. Czyli

$$AD_1 \stackrel{\tau,0}{\cong} AD_2 \quad (129)$$

Jeśli $AD_1 \neq \langle \rangle \wedge AD_2 \neq \langle \rangle \wedge$

$\wedge \max(FiT(AD_1), FiT(AD_2)) \leq 0 \wedge \tau\left(\left\langle \langle A_1, t_1 \rangle \right\rangle, \left\langle \langle A'_1, t'_1 \rangle \right\rangle\right)$, to

$$tail(AD_1) \stackrel{\tau,t}{\cong} tail(AD_2) \quad (130)$$

Warunki poprzedniego przejścia są identyczne jak warunki we wzorze (107). Wykorzystując rekurencyjność w kolejnych krokach dochodzi się do wniosku, że:

$$AC'_1 \stackrel{\tau,0}{\cong} AC'_2 \quad (131)$$

Dowód implikacji z prawej strony na lewą polega na uzupełnieniu ciągów AC'_1 i AC'_2 o te pary, które były usuwane w pierwszej części dowodu. ■

Lemat 4. Niech K jest konfiguracją dowolnej z maszyn obiektów. Wówczas:

$$Set0(Set0(K)) = Set0(K) \quad (132)$$

Dowód

Równanie jest oczywiste, skoro funkcja $Set0(K)$ zmienia na wartość zero czasu tylko tych stanów, które nie posiadają tranzycji czasowej i których czas był mniejszy od zera. Zatem na tak zmienionych wartościach czasów funkcja $Set0(Set0(K))$ nie dokona już żadnych zmian. ■

Twierdzenie 2. Jeśli wykonanie funkcji ZKP i ZKW jest określone dla czasu t i zbioru tranzycji Tr , to dla konfiguracji pierwotnej X zachodzi:

$$TW(ZKP(X, t, Tr)) = ZKW(TW(X), t, Tr) \quad (133)$$

Dowód

Niech

$$\begin{aligned} X &= \langle K_1^1, AC_1^1, K_2^1, AC_2^1, V_1, t_1 \rangle \\ ZKP(X, t, Tr) = X_1 &= \langle K_1^2, AC_1^2, K_2^2, AC_2^2, V_2, t_2 \rangle \\ TW(X) = X' &= \langle K_1^3, AC_1^3, K_2^3, AC_2^3, V_1 \rangle \\ TW(X_1) = X_2 &= \langle K_1^4, AC_1^4, K_2^4, AC_2^4, V_1 \rangle \\ ZKW(X', t, Tr) = X'' &= \langle K_1^5, AC_1^5, K_2^5, AC_2^5, V_2 \rangle \\ R_1 &= Re(K_1, t, Re(Tr)) \\ R_2 &= Re(K_2, t, Re(Tr)) \end{aligned} \quad (134)$$

Zgodnie z definicją:

$$X_1 = \langle K_1^1, AC_1^1 + AdTm(R_1, t_1), K_2^1, AC_2^1 + AdTm(R_2, t_1), V_1, t_1 + t \rangle \quad (135)$$

natomiast

$$X_2 = OC \left(\left\langle \begin{array}{l} Set0(K_1^1), AdTm((AC_1^1 + AdTm(R_1, t_1))|_{ZO}, -t_1 - t), \\ Set0(K_2^1), AdTm((AC_2^1 + AdTm(R_2, t_1))|_{ZO}, -t_1 - t), V_1 \end{array} \right\rangle \right) \quad (136)$$

oraz zgodnie z definicją:

$$X' = OC \left(\langle Set0(K_1^1), AdTm(AC_1^1|_{ZO}, -t_1), Set0(K_2^1), AdTm(AC_2^1|_{ZO}, -t_1), V_1 \rangle \right) \quad (137)$$

natomiast

$$X'' = OC \left(\langle Set0(K_1^3), AdTm(AC_1^3 + R_1|_{ZO}, -t), Set0(K_2^3), AdTm(AC_2^3 + R_2|_{ZO}, -t), V_2 \rangle \right) \quad (138)$$

Przy tych oznaczeniach należy dowieść, że $X_2 = X''$. Funkcja OC nie modyfikuje konfiguracji maszyn obiektów i środowiska. Zatem rozważając tylko konfiguracje z konfrontacji X_2 i X'' muszą zachodzić równości:

$$\begin{aligned} Set0(K_1^1) &= Set0(Set0(K_1^1)) \\ Set0(K_1^1) &= Set0(Set0(K_1^1)) \\ V_2 &= V_2 \end{aligned} \quad (139)$$

Pierwsze są prawdziwe na podstawie lematu 4 (132), trzecia jest tożsamością.

Funkcja OC jedynie obcina początki ciągów, pozostałe części zostawiając bez zmian. Należy zatem sprawdzić, czy dwie pary operacji z dowodzonego twierdzenia tworzą takie same pary ciągów rozpatrując sytuację bez obcinania oraz pokazać, że ostatecznie obetną te same pary reakcji czasowych.

Rozpatrując bez obcinania otrzyma się

$$\begin{aligned} AC_1^4 &= AdTm((AC_1^1 + AdTm(R_1, t_1))|_{ZO}, -t_1 - t) \\ &= AdTm(AC_1^1|_{ZO} + AdTm(R_1, t_1)|_{ZO}, -t_1 - t) \\ &= AdTm(AC_1^1|_{ZO}, -t_1 - t) + AdTm(AdTm(R_1, t_1)|_{ZO}, -t_1 - t) \\ &= AdTm(AC_1^1|_{ZO}, -t_1 - t) + AdTm(R_1|_{ZO}, -t) \\ AC_1^5 &= AdTm(AdTm(AC_1^1|_{ZO}, -t_1) + R_1|_{ZO}, -t) \\ &= AdTm(AdTm(AC_1^1|_{ZO}, -t_1), -t) + AdTm(R_1|_{ZO}, -t) \\ &= AdTm(AdTm(AC_1^1|_{ZO}, -t_1 - t) + AdTm(R_1|_{ZO}, -t) \end{aligned} \quad (140)$$

czyli $AC_1^4 = AC_1^5$. Analogicznie można pokazać, że $AC_2^4 = AC_2^5$.

Niech ciąg $AC_1^1 + AdTm(R_1, t_1)$ zostanie podzielony na trzy rozłączne części: AD_1 z reakcjami o czasach mniejszych równych t_1 , AE_1 z reakcjami o czasach większych od t_1 , ale mniejszych równych t_2 oraz AF_1 z reakcjami o czasach większych od t_2 , analogicznie $AC_2^1 + AdTm(R_2, t_1)$ na ciągi AD_2, AE_2 i AF_2 . Ciągi AD_1 i AD_2 są jednocześnie częściami ciągów AC_1^1 i AC_2^1 odpowiednio.

Dalej pokazane zostanie, że usunięcie pary podczas działania dwóch funkcji z prawej strony równania rozpatrywane go twierdzenia spowoduje usunięcie tej pary podczas funkcji TW lewej strony równania.

Zgodnie z definicją funkcji OC elementy z ciągów AF_1 i AF_2 przy żadnej z par funkcji z równania (133) nie mogą zostać usunięte.

Jeśli jakieś pary zostaną usunięte podczas wewnętrznej funkcji TW prawej strony równania (133), to będą również usunięte podczas zewnętrznej funkcji TW lewej strony równania, gdyż czasy w tych parach będą zmniejszone o $-t$, a wartość funkcji τ jest niezależna od ustalenia punktu zerowego czasu. Pary te mogą składać się tylko z elementów ciągów AD_1 i AD_2 . Pozostałe elementy ciągów AD_1 i AD_2 po wykonaniu funkcji TW z prawej strony równania oznaczone niech będą jako AD'_1 i AD'_2 .

Jeśli jakieś pary zostaną usunięte podczas funkcji ZKW prawej strony równania, to możliwe są trzy przypadki:

1. para składa się z elementu ciągu AD'_1 i elementu ciągu AE_2
2. para składa się z elementu ciągu AE_1 i elementu ciągu AD'_2
3. para składa się z elementu ciągu AE_1 i elementu ciągu AE_2

W każdym z nich para taka zostanie również usunięta podczas operacji TW lewej strony równania. Dla pierwszego przypadku wynika to z faktu, że musiały być usunięte wszystkie elementy ciągu AD_2 i para ta składa się z (dla pewnego i) z i -tego elementu ciągu AD'_1 oraz z i -tego elementu ciągu AE_2 . Niech $|AD_2| = n$, zatem dla funkcji z prawej strony jest to $n+i$ -ty element ciągu AD_1 czyli też ciągu $AD_1 + AE_1$ oraz $n+i$ -ty element ciągu $AD_2 + AE_2$. Analogicznie można rozpatrzeć pozostałe przypadki.

Podobne rozumowanie można przeprowadzić pokazując, że usunięcie pary podczas działania funkcji TW lewej strony równania spowoduje usunięcie tej pary podczas dwóch funkcji z prawej strony równania twierdzenia (133).

Rozpatrzyć należy cztery przypadki:

1. para składa się z elementu ciągu AD_1 i elementu ciągu AD_2
2. para składa się z elementu ciągu AD_1 i elementu ciągu AE_2
3. para składa się z elementu ciągu AE_1 i elementu ciągu AD_2
4. para składa się z elementu ciągu AE_1 i elementu ciągu AE_2

Para usunięta w pierwszym przypadku będzie usunięta podczas wykonywania funkcji TW prawej strony równania. Pozostałe zostaną usunięte podczas wykonania funkcji ZKW . ■

Poprzednie twierdzenie poprzez wielokrotne zastosowanie można rozwinąć do poniższego:

Twierdzenie 3. Jeśli kolejne zmiany konfrontacji (pierwotnej i wtórnej) są możliwe, to zachodzi:

$$TW(\overbrace{ZKP(\dots(ZKP(ZKP(X, t_1, Tr_1), t_2, Tr_2) \dots, t_n, Tr_n))}^{n \text{ razy}} = \overbrace{ZKW(\dots ZKW(ZKW(TW(X), t_1, Tr_1), t_2, Tr_2), \dots), t_n, Tr_n)}^{n \text{ razy}} \quad (141)$$

Twierdzenie to pozwala na to, aby zamiast wykonywać kolejne zmiany konfrontacji pierwotnych, a następnie stworzyć wtórną, najpierw stworzyć wtórną, a następnie wykonać, dla tych samych czasów i zbiorów tranzycji, zmiany konfrontacji wtórnych. W połączeniu z twierdzeniem 1 (125) pozwala to działać na konfrontacjach wtórnych i sprawdzać potencjalną zgodność ciągów w czasie względnym zamiast zmieniać konfrontacje pierwotne i badać zgodność w czasie będącym szóstym elementem tych konfrontacji. Na tej właściwości oparty jest zaprezentowany algorytm.

Twierdzenie 4. Algorytm jeśli się zatrzyma, poda poprawną odpowiedź, czyli TAK - jeśli maszyny obiektów są zgodne z wybranymi parametrami oraz NIE - jeśli maszyny nie są zgodne.

Dowód

Wynika to z poprzednich rozważań. Trzy przypadki, gdy odpowiedź jest NIE zachodzą wtedy, gdy istnieje przypadek konfrontacji wtórnej posiadającej ciągi reakcji, które niezależnie od następnych zmian konfrontacji, będą niezgodne. Z twierdzenia 1 (125) wynika, że niezgodność ta będzie dotyczyć również konfrontacji pierwotnych, czyli można znaleźć taki ciąg zdarzeń ze środowiska, przy którym porównywane maszyny są niezgodne.

Inną możliwością zatrzymania się algorytmu jest odpowiedź TAK w przypadku, gdy wszystkie konfrontacje wtórne prowadzą do innych już wcześniej przebadanych konfrontacji wtórnych. Oznacza to, że wszelkie możliwe zmiany prowadzą do tworzenia potencjalnie zgodnych ciągów reakcji i nie udało się znaleźć ciągów niezgodnych. ■

Pozostaje udowodnić, że algorytm zatrzyma się w skończonym czasie. Do tego celu posłużą poniższe lematy.

Lemat 5. Liczba możliwych konfiguracji maszyny środowiska, występujących w algorytmie, jest skończona.

Dowód

Ponieważ przy zmianie konfiguracji środowiska możliwe jest albo zmniejszenie czasu w liściach (do wartości większej niż zero) bądź wejście do stanów (z maksymalnym czasem równym czasowi akcji wyjściowej), zatem każdy stan ma tylko skończoną liczbę czasów z jaką może zaistnieć w konfiguracji. Ilość podzbiorów zbioru wszystkich stanów również jest ograniczona. ■

Lemat 6. Maksymalna wartość czasu jaką może posiadać każdy stan w konfiguracji maszyny obiektu w algorytmie jest ograniczona.

Dowód

Rozpatrując jakie wartości czasu może mieć stan maszyny klasy w konfiguracji obiektu, dla każdego stanu można podać maksymalną wartość dodatnią z jaką może wystąpić w konfiguracji. Osiągana jest ona przy wchodzeniu do tego stanu. Rozważmy stan s . Jeśli wejście do tego stanu nastąpiło z konfiguracji K' i powodowane było tranzycją tr od stanu s' do stanu s'' , a stan s jest podstanem (również pośrednim) stanu s'' , to maksymalna dodatnia wartość czasu w konfiguracji dla stanu s równa się:

$$ExitT(K', s, 1) + trt(tr) + \sum_{i=1}^k ent(s), \text{gdzie } s_1 = s'', s_k = s, \forall_{j=1}^{k-1} s_{j+1} \in childB(s_j) \quad (142)$$

co jest wartością skończoną. ■

Lemat 7. Minimalna wartość czasu jaką może posiadać każdy stan w konfiguracji maszyny obiektu w algorytmie jest ograniczona.

Dowód

Szukając dla każdego stanu najmniejszej wartości czasu jaką może osiągnąć w konfiguracji należy rozważyć dwa przypadki. Pierwszy, gdy stan posiada tranzycję czasową wyprowadzającą z niego oraz gdy stan takiej tranzycji nie posiada.

W pierwszym przypadku, jeśli stan posiada tranzycję czasową o czasie przeterminowania t_x , to najmniejszą wartość jaką ten stan może posiadać w konfiguracji to $-t_x$.

Jeśli w konfiguracji K stan s nie posiada tranzycji czasowej, to minimalna wartość czasu w konfiguracji po wykonaniu funkcji OC wynosi zero. ■

Lemat 8. Liczba możliwych konfiguracji maszyny obiektu, występujących w algorytmie, jest skończona.

Dowód

Wynika to z poprzednich trzech lematów. Zbiór możliwych podzbiorów zbioru stanów jest skończony, a każdy stan może posiadać ograniczony zbiór wartości czasu wejścia występujący w konfiguracji z danych stanem. ■

Lemat 9. Wartości czasu w ciągach reakcji pojawiających się w algorytmie są zbiorem skończonym.

Dowód

Wynika to z tego, maksymalna wartość czasu jest nie większa niż maksymalna wartość czasu dla stanów we wzorze (142). Minimalna wartość jest ograniczona przez właściwość 2 funkcji czasowej i wynosi $-Len(\tau)$. Jeśliby wartość ta podczas zmiany konfiguracji wtórnej przekroczyła wartość minimalną, to albo musi zostać usunięta podczas operacji OC , albo nie zajdzie warunek potencjalnej zgodności, zatem algorytm się natychmiast zatrzyma. ■

Lemat 10. Liczba możliwych ciągów reakcji każdej z maszyn, występujących w konfrontacjach wtórnych w algorytmie jest skończona.

Dowód

Wynika to z poprzedniego lematu, oraz ograniczonego zbioru możliwych reakcji. ■

Ostatecznie otrzymujemy następujące twierdzenie:

Lemat 11. Liczba możliwych konfrontacji wtórnych pojawiających się w algorytmie jest ograniczona.

Dowód

Wynika to stąd, że konfrontacja jest piątką elementów, z których każdy może przyjmować tylko skończoną liczbę wartości, co zostało pokazane w poprzednich twierdzeniach. ■

Twierdzenie 5. Algorytm zatrzymuje się w skończonym czasie.

Dowód

Wynika to ze skończonej liczby konfrontacji, jaką może wygenerować algorytm. Po stworzeniu nowej konfrontacji algorytm może się zatrzymać przy sprawdzaniu potencjalnej dokładności albo wstawić tę konfrontację do kolejki Q . Po wyciągnięciu konfrontacji z tej kolejki trafia ona do zbioru X (jeśli jej tam nie było), albo w następnym kroku wyciągana jest następna konfrontacja z kolejki. W ten sposób, jeśli algorytm nie zatrzyma się dając odpowiedź NIE, to wszystkie możliwe konfrontacje znajdą się w zbiorze X , a w konsekwencji nastąpi opróżnienie kolejki Q w krokach 2 i 3. Tym sposobem algorytm zatrzyma się w kroku 2. ■

11.2 Implementacja algorytmu

Algorytm został zaimplementowany w języku Java 1.4.2, w środowisku Eclipse 3.0, z narzędziem JUnit. Badane diagramy stanów obiektów i środowiska są zadawane jako pliki w formacie XML, którego specyfikacja znajduje się w pliku *StateMachine.dtd*. Specyfikacja ta nie wymusza strukturalności diagramów stanów i poprawności czasów, o które musi zadbać twórca diagramów.

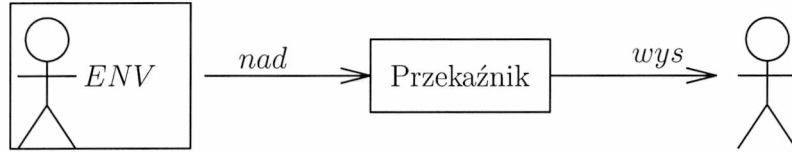
Algorytm zakłada, że zadane maszyny spełniają wszystkie wymogi wymienione opisach maszyn stanowych obiektów i środowiska.

11.3 Przykładowe zastosowania i wyniki działania algorytmu

Zostaną przedstawione dwa przykładowe zastosowania oraz wyniki działania algorytmu dla podanych przykładów. Przykłady te są przedstawicielami pewnej klasy zagadnień.

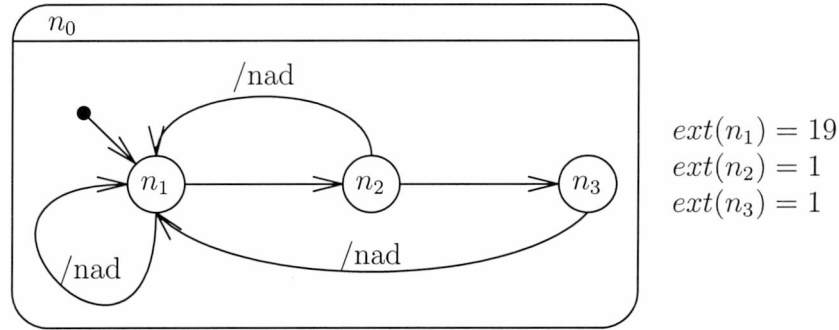
Pierwsza kategoria zagadnień związana jest z sytuacją, gdy istnieje specyfikacja pewnego systemu i na tej podstawie należy zbudować urządzenie, które tą specyfikację spełnia. Jednak do dyspozycji jest urządzenie, które spełnia wymagania funkcjonalne, ale nie spełnia wymagań wydajnościowych. Można wtedy zastanawiać się, czy połączenie w pewien sposób odpowiedniej liczby dostępnych urządzeń nie pozwoli na uzyskanie systemu o wymaganej wydajności.

Druga kategoria to urządzenie, które należy rozbudować np. o funkcję pomiaru diagnostycznego. Nasuwa się oczywiste pytanie, czy ta rozbudowa nie zakłóci podstawowych funkcji urządzenia. Zatem jest to problem dziedziczenia zachowań.

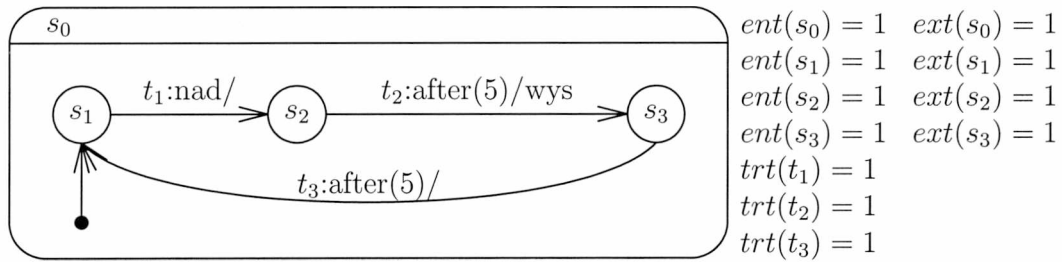


Rysunek 26: Pierwszy przykład

Przykładem należącym do pierwszej kategorii jest specyfikacja przełącznika odbierającego sygnały nadawane ze środowiska i po pewnym czasie przekazującego je dalej. Opisaną sytuację przedstawia rys. 26 Urządzenia dostępne, specjalizowane, dodatkowo liczą pewną sumę kontrolną odebranego sygnału.



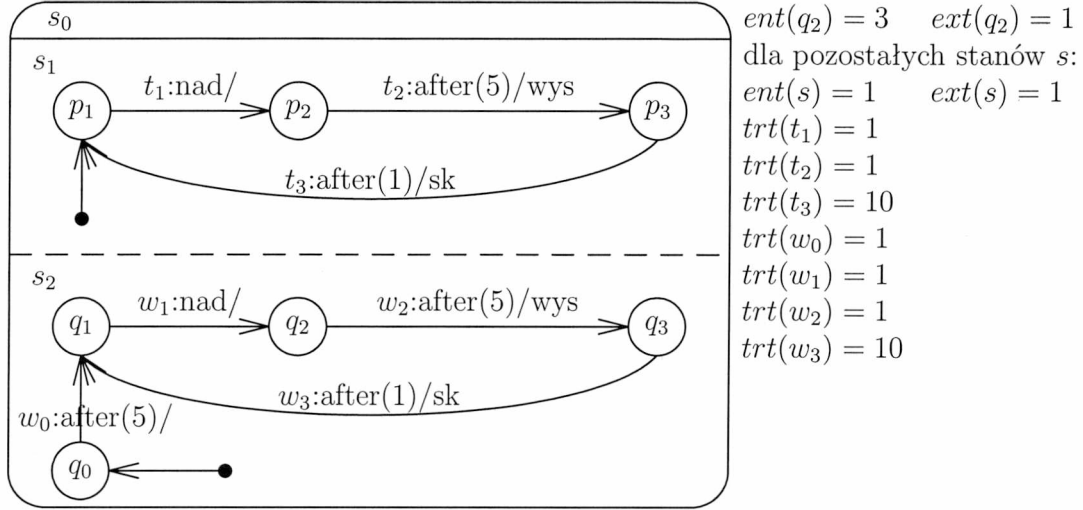
Rysunek 27: Środowisko - nadawca



Rysunek 28: Przełącznik A

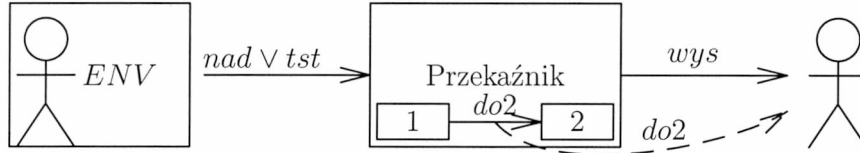
Środowisko przedstawione jest na rys. 27. Generuje ono sygnał nadawania po 19, 20 lub 21 jednostkach czasu. Diagram nadawcy pierwotnego, przełącznik A przedstawia rys. 28, natomiast przełącznika specjalizowanego B rys. 29.

Należy zbadać, czy w przedstawionym środowisku zachowanie przełącznika A jest dziedziczone w przełączniku B. Zdarzeniami obserwowalnymi jest tylko zdarzenie *wys*, natomiast funkcja porównująca to funkcja $\tau_{\leq \delta}$. Dla wartości $\delta = 1$ algorytm wytworzył 6 konfrontacji i stwierdził, że nie zachodzi dziedziczenie zachowań. Dla wartości $\delta = 2$



Rysunek 29: Przełącznik B

algorytm wytworzył 14 konfrontacji i stwierdził, że zachodzi dziedziczenie zachowań. Zmodyfikowanie środowiska w ten sposób, aby generowało sygnał nadawania co przedział $p \in \{19, \dots, 24\}$ spowodowało wygenerowanie odpowiednio 6 oraz 35 konfrontacji dla podanych wcześniej wartości δ .



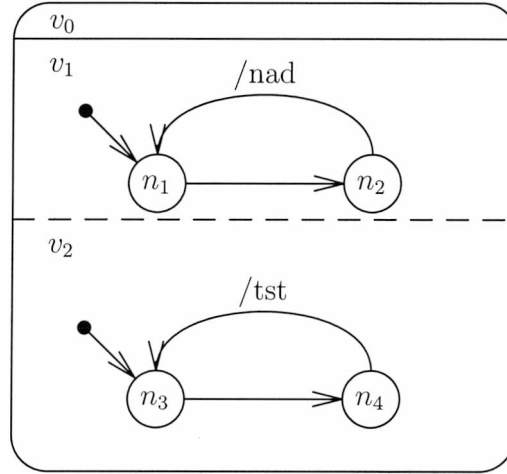
Rysunek 30: Drugi przykład

Drugim przykładem jest również przełącznik, jednak obserwator dostrzega przekazanie sygnału z pierwszej części przełącznika do drugiej (rys. 30). Wersja specjalizowana, która ma zastąpić pierwotną reaguje również na sygnał testowania i odpowiadając na niego wysyła sumę kontrolną.

Diagram pierwotnego przełącznika C przedstawiony jest na rys. 32, natomiast diagram przełącznika specjalizowanego na rys. 33. Urządzenia te działają w środowisku (rys. 31), które wysyła sygnał nadawania (co 20 jednostek) oraz testowania (co 51 jednostek).

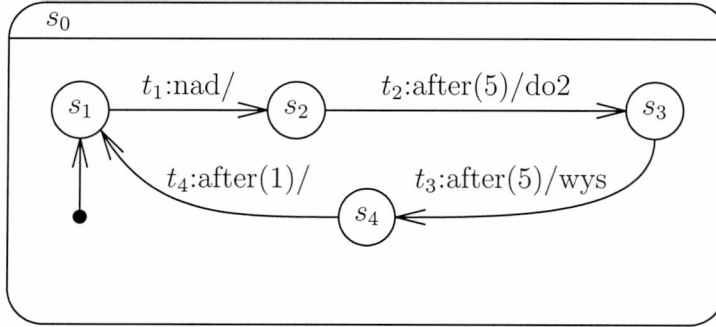
Przy badaniu dziedziczenia zachowań przyjęto, że zdarzenia obserwowalne, to nad oraz $do2$, a funkcja porównująca to funkcja $\tau_{\leq \delta}$. Dla wartości $\delta \in \{0, 1, 2\}$ algorytm wytworzył 15 konfrontacji i stwierdził, że nie zachodzi dziedziczenie zachowań. Dla wartości $\delta = 6$ algorytm wytworzył 273 konfrontacje i stwierdził, że zachodzi dziedziczenie zachowań. Zależność liczby konfrontacji i ilości czasu od parametru δ przedstawia tabela dla ENVNadTester.

Zmiana środowiska w ten sposób, aby sygnał nadawania pojawiał się co 20 lub 21 jednostek, natomiast sygnał testowania co 51 lub 52 jednostek, powoduje zwiększenie czasu trwania algorytmu (tabela dla ENVNadTester2). Podobna sytuacja zachodzi,



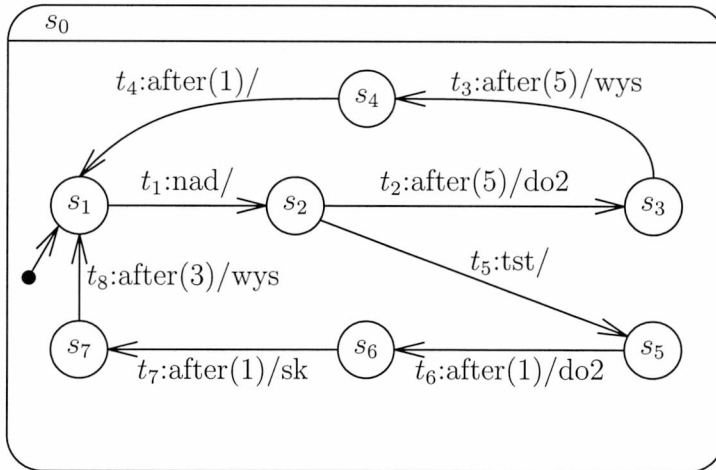
$$\begin{aligned}
 ext(v_0) &= 1 \\
 ext(v_1) &= 1 \\
 ext(v_2) &= 1 \\
 ext(n_1) &= 18 \\
 ext(n_2) &= 1 \\
 ext(n_3) &= 50 \\
 ext(n_4) &= 1
 \end{aligned}$$

Rysunek 31: Środowisko - nadawca i tester



dla wszystkich stanów s :
 $ent(s) = 1$ $ext(s) = 1$
dla wszystkich tranzycji t :
 $trt(t) = 1$

Rysunek 32: Przekaźnik C



dla wszystkich stanów s :
 $ent(s) = 1$ $ext(s) = 1$
dla wszystkich tranzycji t :
 $trt(t) = 1$

Rysunek 33: Przekaźnik D

gdy ustalone zostaną wartości na odpowiednio 20, 21 lub 22 oraz 51, 52 lub 53 (tabelka dla ENVNadTester3).

ENVNadTester			
δ	l. konf.	wynik	[s]
0	15	NIE	0,12
1	15	NIE	0,01
2	15	NIE	0,04
3	17	NIE	0,02
4	17	NIE	0,01
5	38	NIE	0,03
6	273	TAK	0,20

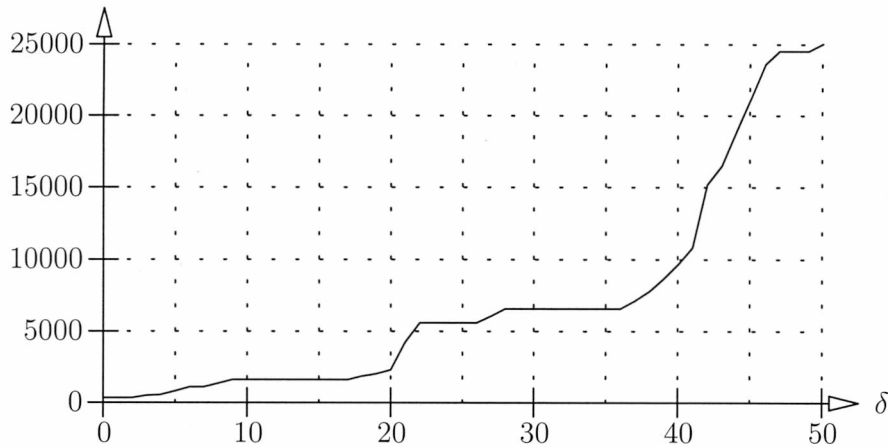
ENVNadTester2			
δ	l. konf.	wynik	[s]
0	88	NIE	0,17
1	88	NIE	0,08
2	88	NIE	0,07
3	115	NIE	0,06
4	123	NIE	0,09
5	166	NIE	0,09
6	811	TAK	0,58

ENVNadTester3			
δ	l. konf.	wynik	[s]
0	187	NIE	0,27
1	187	NIE	0,13
2	187	NIE	0,13
3	249	NIE	0,12
4	261	NIE	0,19
5	370	NIE	0,14
6	1515	TAK	0,93

W przypadku, gdy sygnał nadania może się pojawiać co 20, 21, 22 lub 23 jednostki, natomiast sygnał testowania niezależnie co 51, 52, 53 lub 54 jednostki, to bez względu na parametr δ maszyny są niezgodne (tabelka dla ENVNadTester4). Z wykresu tworzonej konfrontacji (rys. 34) i czasu działania (rys. 35) algorytmu wynika, że złożoność rośnie wykładniczo.

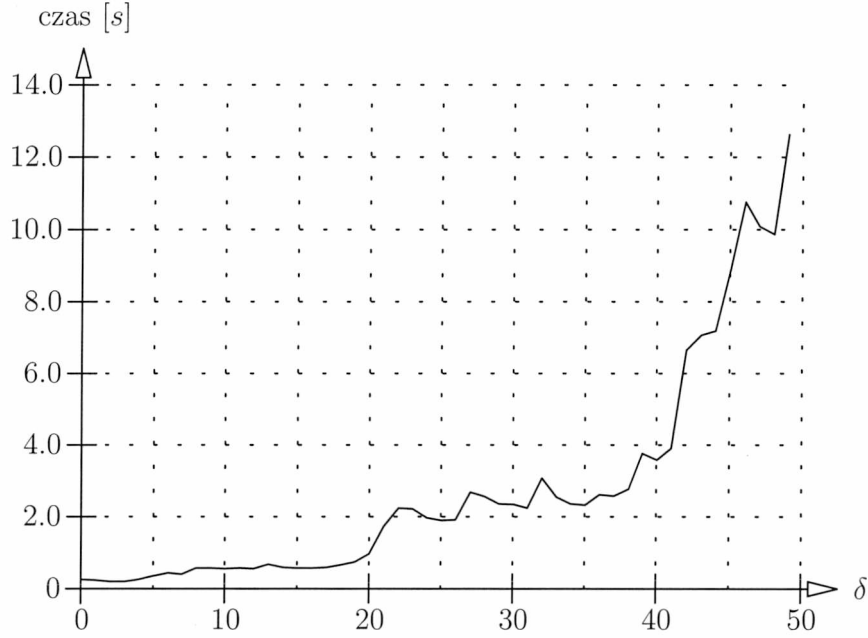
ENVNadTester4			
δ	l. konf.	wynik	[s]
0	356	NIE	0,27
10	1600	NIE	0,56
20	2309	NIE	0,97
30	6534	NIE	2,34
40	9640	NIE	3,58
50	24501	NIE	12,63

liczba konfrontacji



Rysunek 34: Zależność liczby konfrontacji od parametru δ w funkcji $\tau_{\leq \delta}$

Pliki z danymi dla pierwszego przykładu to: MSPrzekaznikA.xml, MSPrzekaznikB.xml, ENVNad2.xml, ENVNad3.xml, natomiast dla drugiego: MSPrzekaznikC.xml, MSPrzekaznikD.xml, ENVNadTester.xml, ENVNadTester2.xml, ENVNadTester3.xml, ENVNadTester4.xml.



Rysunek 35: Zależność ilości czasu od parametru δ w funkcji $\tau_{\leq \delta}$

11.4 Złożoność algorytmu

Zaproponowany algorytm ma teoretycznie dużą złożoność. Liczba kroków jaką należy wykonać zależy praktycznie od liczby konfrontacji wtórnych jakie musi wygenerować, natomiast liczba konfrontacji zależy od liczby konfiguracji maszyn obiektów i maszyny środowiska oraz ciągów reakcji niesprawdzonych co do zgodności.

Niech liczba stanów maszyny środowiska będzie oznaczona jako n_v , liczba stanów każdej z maszyn obiektów jako odpowiednio n_{MS_1} i n_{MS_2} . Najgorszy przypadek jest wtedy, gdy każdy ze stanów jest w części równoległej i może być lub nie być w konfiguracji maszyny. Wówczas liczba kombinacji wynosi dla każdej z maszyn odpowiednio: 2^{n_v} , $2^{n_{MS_1}}$ oraz $2^{n_{MS_2}}$. Dodatkowo każdy z tych stanów może mieć dowolny czas wejścia (występujący w konfiguracji) w zakresie od zera (dla stanu bez tranzycji czasowej) lub minus czas przeterminowania jego tranzycji czasowej do maksymalnej wartości dodatniej osiągananej podczas wejścia do tego stanu. Jeśli taki przedział oznaczony zostanie jako t_p to zakładając niezależność każdego stanu otrzyma się złożoność rzędu $t_p^{n_v} * 2^{n_v}$ dla konfiguracji środowiska i analogicznie dla pozostałych maszyn. Podobne rozumowanie można przeprowadzić dla ciągów reakcji występujących w konfrontacjach wtórnych. Przedziałem czasu czasu jest w tym przypadku okres od $-Len(\tau)$ do sumy czasu wyjścia ze stanu plus czas uruchomienia tranzycji.

W praktyce tak duża złożoność nie będzie występować, gdyż czasy w stanach podrzędnych zależą od czasu stanu nadrzędnego oraz od możliwych pętli tranzycji powodujących powrót do tego samego stanu. W algorytmie nie wszystkie konfrontacje zostaną stworzone, tylko te które wynikają z napływających zdarzeń ze środowiska. Również w przypadku ciągów liczba kombinacji jest mniejsza, gdyż tylko w jednym ciągu mogą pozostać elementy z czasem mniejszym od zera (pozostanie w drugim oznaczałoby nie zachodzenie potencjalnej zgodności, zatem zatrzymanie algorytmu).

Rozpatrując sytuację, gdy diagramy stanów użyte w algorytmie nie posiadają części równoległych, liczba możliwych kombinacji stanów w konfiguracji równa się liczbie stanów. Przyjmując taką samą interpretację oznaczenia t_p , liczba możliwych konfiguracji np. dla maszyny środowiska wynosi wówczas $t_p^{n_v}$. Jednak wystąpienie wszystkich tych czasów jest niemożliwe, gdyż np. po wyjściu ze stanu wewnętrznego i wejściu do stanu wewnętrznego powoduje, że czasy w stanach zewnętrznych zmniejszą się i będą to największe możliwe czasy do osiągnięcia w tej kombinacji stanów konfiguracji.

Wynika z tych rozważań, że złożoność algorytmu wzrasta wykładniczo wraz ze wzrostem liczby części równoległych, liczby tranzycji oraz przedziału czasu jaki może pojawiać w parach ze stanami i reakcjami.

Potwierdzają to przykłady, gdzie rozbudowa tylko sekwencyjnie złożonego środowiska nie powoduje szybkiego przyrostu generowanych konfrontacji. Natomiast tworzenie rozbudowanego środowiska złożonego równoległe, oraz dopuszczenie dużego przedziału tolerancji w funkcji porównującej może spowodować "eksplozję" generowania konfrontacji.

W praktycznych zastosowaniach czas działania przedstawionego algorytmu powinien być akceptowalny. Wynika to stąd, że w diagramach stanów liczba wszystkich części równoległych jest najczęściej kilka razy mniejsza od liczby wszystkich stanów a liczba tranzycji nie przekracza kilkunastu do kilkudziesięciu. Również czasy występujące w diagramach wyrażane są najczęściej niezbyt dużymi liczbami (lub do takich można je sprowadzić po podzieleniu przez wspólny mianownik). Także przedział tolerancji występujący w funkcji porównującej powinien być mały, gdyż obserwator nie powinien pozwalać na duże odstępstwa czasowe.

Z opracowań [16, 34] wynika, że również systemy do badania różnych cech czasowych automatów skończonych posiadają właściwości analogiczne dla prezentowanego algorytmu.

11.5 Zastosowanie algorytmu do badania zgodności zachowań bezczasowych

Przedstawiony algorytm można stosować również dla maszyn bez czasowych. W tym celu należy w diagramie bezczasowym uzupełnić wartości czasów dla wszystkich stanów i tranzycji, oraz stworzyć właściwe środowisko.

W przypadku dziedziczenia protokołu należy ustalić we wszystkich stanach i tranzycjach wszystkie czasy na wartość jednostkową. Graf środowiska powinien być kopią struktury diagramu maszyny pierwotnej, natomiast wszystkie czasy wyjścia ze stanów powinny być ustawione na jedną wartość M . Tą wartością jest co najmniej trzykrotność maksymalnej liczby zagnieżdżeń stanu w stanie. Celem takiego ustawienia jest, aby następne zdarzenie nadeszło, gdy zakończyły się akcje wejściowe we wszystkich stanach. Wybraną funkcją porównującą powinna być $\tau_{\leq M*N}$, gdzie N jest maksymalną liczbą krawędzi w jednym z diagramów stanów maszyn obiektów. Celem tego jest umożliwienie pozostawienia odpowiednio długiego ciągu reakcji reakcji do późniejszego porównania.

Zastosowanie do pozostałych definicji jest utrudnione znalezieniem odpowiedniego środowiska. Szczególnie definicja dziedziczenia cyklu życia oznacza znalezienie dowolne-

go środowiska, spełniającego podane w definicji właściwości.

Przy stosowaniu algorytmu dla maszyn bezczasowych, nie będą istniały tranzycje czasowe, co spowoduje, że nigdy nie wykonają się kroki (9) i (10) rozważanego algorytmu. Oprócz tego wykonując krok (5b) wszystkie stany otrzymają w konfiguracjach czas równy zero. Wynikiem tego będzie generowanie tych samych zestawów stanów w konfiguracjach zawsze z tymi samymi czasami.

12 Podsumowanie

12.1 Wyniki prac

W pracy przedstawiono rozważania wpływu czasu na porównywanie zachowań dwóch maszyn czasowych. Podana została ogólna definicja dziedziczenia zachowań czasowych maszyn stanów z wykorzystaniem języka czasu rzeczywistego. Istotą tej definicji jest porównywanie maszyn stanów względem środowiska. Definicja uwzględnia trzy elementy, względem których określa, czy ma miejsce dziedziczenie zachowań. Są to: środowisko generujące ciągi zdarzeń, funkcja porównująca zachowania czasowe oraz zbiór reakcji obserwowalnych.

Wybrano pewną kategorię diagramów stanów języka UML do reprezentacji zachowań obiektu. Kategoria ta nakłada wprawdzie ograniczenia na diagramy stanów, ale nie ogranicza ogólności rozważań. Ograniczenia te to strukturalność diagramów (zalecana cecha metodyczna) oraz wyeliminowanie elementów redundantnych pozwalających na różne zapisy tej samej maszyny stanowej. Oprócz tego rozważana jest tylko komunikacja asynchroniczna, a w ramach stanów nie są wprowadzone aktywności wewnętrzne ani tranzycje wewnętrzne. Nie są też rozważane, nieużywane w innych opracowaniach na temat dziedziczenia zachowań, takie elementy jak stany historyczne czy zdarzenia odroczone.

Diagramy stanów służą również do reprezentacji zachowań środowiska. Środowisko jest specyficzną maszyną, w której zmiany stanów mogą się odbywać tylko poprzez upływ czasu. Z tego powodu wybrane zostały te elementy diagramów stanów, które są wystarczające do reprezentacji wybranej kategorii środowisk.

Diagramy służą w pracy do reprezentacji zachowań czasowych, nieobecnych w oryginalnej dokumentacji. Dokonano rozszerzenia interpretacji zachowań diagramów stanów po wprowadzeniu do nich czasu rzeczywistego.

Dla wybranej kategorii diagramów stanów obiektów i diagramów stanów środowiska, oraz funkcji porównującej o określonych właściwościach zaprezentowano algorytm badający dziedziczenie zachowań. Przedstawiono uzasadnienie jego poprawności oraz oszacowanie złożoności obliczeniowej. Złożoność ta w ogólnym przypadku jest rzędu wykładniczego. Jednak, ponieważ rzeczywista złożoność zależy od struktury diagramów oraz występujących w nich wartości czasów, to w praktycznych zastosowaniach złożoność ta jest niższa.

Algorytm, po przyporządkowaniu odpowiednich czasów, może służyć również do porównywania maszyn bezczasowych.

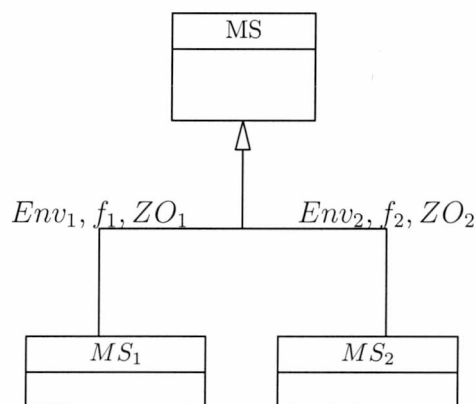
Opracowano program, który implementuje przedstawiony algorytm. Przedstawio-

no przykładowe wyniki programu, pokazujące dwa przypadki dziedziczenia zachowań. Na załączonym CDROM-ie znajduje się dokumentacja korzystania z algorytmu wraz z przykładowymi plikami.

12.2 Problemy na przyszłość

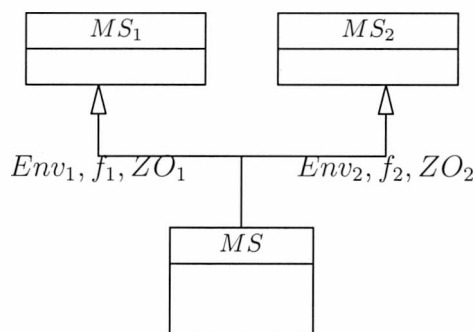
Problemów na przyszłość jest wiele i tu przedstawione zostaną tylko niektóre. Najpierw zaprezentowane zostaną problemy wyłaniające się w ramach przyjętej definicji i algorytmu. Stanowią one rozszerzenie problemów występujących w maszynach bezczasowych.

Pierwszy problem to kwestia, czy można i ewentualnie jak znaleźć maszynę stanową MS będącą uogólnieniem dwóch maszyn MS_1 i MS_2 . rys (36). Pozwoliłoby to na badanie zachowań w jednej maszynie zamiast w dwóch.



Rysunek 36: Generalizacja maszyn stanów

Drugi problem to stworzenie jednej maszyny, która dziedziczyłaby zachowania dwóch innych maszyn rys (37). Pozwoliłoby to, na podstawie postulatu Liskov, na użycie maszyny MS zarówno zamiast maszyny MS_1 jak i zamiast MS_2 .



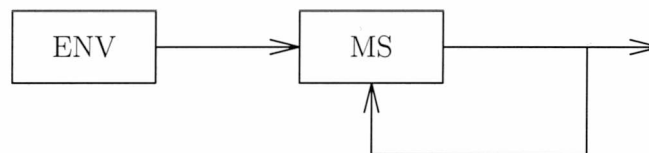
Rysunek 37: Dziedziczenie zachowań dwóch maszyn stanów

Dla obu przedstawionych problemów algorytm zaprezentowany w pracy można wykorzystać do sprawdzenia czy zachodzi dziedziczenie między maszynami MS i MS_1 , oraz między MS i MS_2 .

Problemem jest też kwestia, jakie modyfikacje (dodanie bądź usunięcie stanu bądź tranzycji) czasowego diagramu stanów są możliwe, tak aby jego zachowanie zostało dziedziczone przy zadanych parametrach dziedziczenia zachowań.

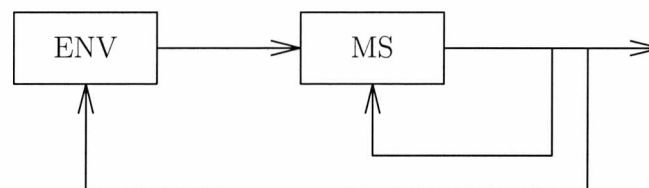
W pracy przyjęto założenie, że maszyny obiektów muszą być deterministyczne. W przypadku maszyn niedeterministycznych należałoby podać definicję dziedziczenia zachowań. Możliwe są różne interpretacje dziedziczenia zachowań dla tego przypadku. Szczególnie istotna byłaby interpretacja pojęcia dziedziczenia zachowań w sytuacji porównywania maszyny deterministycznej z niedeterministyczną.

W prezentowanej pracy środowisko jest specyficzną maszyną, która zmienia stan tylko pod wpływem czasu, natomiast maszyny obiektów sterowane są zdarzeniami ze środowiska lub upływem czasu. Krokiem dalej jest wprowadzenie sprzężenia zwrotnego w ramach maszyny obiektu (ma to sens w przypadku istnienia części równoległych), tzn. reakcje maszyny obiektu mogą być jednocześnie zdarzeniami dla tej maszyny (rys 38).



Rysunek 38: Sprzężenie zwrotne obiektu

Nie będzie to miało wpływu na ogólną definicję DZ, jednak w ramach zaprezentowanego algorytmu wprowadzi dodatkowy element komplikacji. Tak jak w oryginalnym algorytmie zapamiętane są reakcje, o których wiadomo, że zdarzą się w przyszłości, tak w nowej wersji algorytmu dodatkowo należałoby pamiętać zdarzenia, o których wiadomo, że w przyszłości maszyna wyśle do siebie (zdarzenia wewnętrzne). Przed rozpatrzeniem zdarzenia ze środowiska należałoby rozpatrzyć wcześniejsze od niego zdarzenia wewnętrzne. Zmiany konfiguracji wywołane przez te zdarzenia mogłyby wygenerować nowe zdarzenia wewnętrzne również wcześniejsze w stosunku do zdarzenia ze środowiska. Proces ten musiałby być zatem kontynuowany aż do momentu, gdy nie będzie już żadnych zdarzeń wewnętrznych maszyny obiektu wcześniejszych od zdarzenia ze środowiska. Dodatkowo w kroku 7 należałoby stworzyć dwie ścieżki postępowania. Jeśli żadna maszyna nie posiada wygenerowanych przez siebie zdarzeń do obsłużenia to działanie będzie jak w oryginalnym algorytmie, jeśli jednak takie są, to działanie będzie zbliżone jak do zdarzeń ze środowiska w kroku 5.



Rysunek 39: Sprzężenie między środowiskiem a obiektem

Istotniejszą modyfikacją modelu byłoby wprowadzenie dodatkowo sprzężenia zwrot-

nego między maszyną obiektu a maszyną środowiska (39). Wówczas środowisko mogłoby reagować na reakcje obiektu w podobny sposób jak maszyna obiektu na zdarzenia ze środowiska. To podejście spowoduje jednak potrzebę zmiany definicji dziedziczenia zachowań, gdyż zmieniłaby się definicja środowiska. Z tego rodzaju zmianą związana jest możliwość wprowadzenia komunikacji synchronicznej między środowiskiem a maszyną obiektu.

Przedstawiona definicja dziedziczenia zachowań nie posiada właściwości przechodności dla dowolnych parametrów dziedziczenia, czyli jeśli maszyna MS_2 dziedziczy zachowanie MS_1 przy pewnych parametrach, natomiast MS_3 dziedziczy zachowania MS_2 przy innych parametrach, to nie wiadomo czy MS_3 dziedziczy zachowanie MS_1 i przy jakich parametrach. Należałoby znaleźć takie klasy środowisk i funkcji porównujących, dla których taka przechodność jest możliwa lub takich, że parametry dziedziczenia przez MS_3 zachowań MS_1 można wyrazić jako funkcję parametrów pozostałych dziedziczeń.

Problemem na przyszłość jest też opracowanie reguł przekształcenia problemu dziedziczenia zachowań dla wybranych kategorii maszyn stanowych do problemu możliwego do rozwiązania w narzędziach rodzaju UPPAAL lub IF i porównanie efektywności tego podejścia z algorytmem prezentowanym w pracy.

Skorowidz

- środowisko, 15
- JEC, 13
- acc, 56
- AdTm, 59, 63
- afp, 52
- aktywne stany, 56
- alfabet, 13
- Arc, 35
- ArcN, 35
- AS, 56
- AT, 56
- bezpośrednia zmiana konfiguracji środowiska, 46
- bezpośrednia zmiana konfiguracji maszyny, 60
- BoxN, 34
- CH, 47
- childB, 34
- ciąg reakcji, 16
- ciąg reakcji czasowych, 15
- ciąg zdarzeń, 15
- ConCS, 34
- Cut, 65
- czas do wyjścia, 44
- czas do zakończenia wejścia, 51
- czas minimalny konfiguracji, 45
- czas wyjścia ze stanu maszyny, 53
- defaultB, 35
- diagram stanów, 36
- dodawanie słów czasowych, 47
- dopuszczalność funkcji porównującej, 63
- dopuszczalność zbioru tranzycji, 46
- dopuszczalny zbiór tranzycji, 56
- dziedziczenie zachowań w środowisku, 18
- ent, 36
- Entry, 43, 52
- ET, 48
- etf, 45
- EV, 36
- ev, 56
- ExitT, 53
- ext, 36
- F, 62
- f, 62
- F0, 62
- FiL, 65
- FinS, 34
- FiT, 65
- FiTr, 59
- GenMinTT, 54
- GenTT, 55
- graf diagramu stanów, 34
- GTS, 46
- InTr, 59
- język formalny etykietowany czasem, 13
- konfiguracja środowiska, 43
- konfiguracja maszyny obiektu, 50
- konfiguracja początkowa środowiska, 44
- konfiguracja początkowa maszyny, 52
- konfrontacja pierwotna, 66
- konfrontacja wtórna, 67
- L, 14
- Leaf, 44
- Leaf0, 45
- Len, 63
- liść konfiguracji środowiska, 44
- litera, 13
- litera czasowa, 13
- LS, 46
- maszyna obiektu, 15
- mat, 53
- maxw, 58
- MinT, 45
- MinTm, 54
- MinTmTr, 54

nsa, 53
 obcięcie ciągów, 67
 obcięcie języka, 18
 obcięcie słowa, 18
 OC, 67
 opuszczenie stanów za pomocą zbioru
 tranzycji, 46
 OU, 36

 przedział czasu przeterminowania, 52
 przesunięcie czasowe do zera, 45
 ps, 56

 Re, 47, 61
 reakcja, 16
 reakcja czasowa, 15
 ReET, 61
 ReMinTT, 60
 ReOneET, 60
 ReTT, 60
 root, 34
 RunET, 59
 RunMinTT, 55
 RunOneET, 58
 RunTT, 55

 słowo, 13
 SeqCS, 34
 Set0, 67
 ShTm0, 45
 SimS, 34
 sink, 35
 source, 35
 Swatt, 52

 tafp, 53
 ThEm, 49
 Time, 13
 tra, 53
 Tran, 58
 tranzycja, 35
 tre, 36
 tro, 36
 trt, 36
 trw, 36

 tte, 44, 51
 TW, 68
 tworzenie wtórnej, 68
 typeB, 34

 wejście do stanu środowiska, 43
 wejście do stanu maszyny, 52
 wierzchołek źródłowy, 35
 wierzchołek docelowy, 35

 zbiór liści zerowych, 45
 zbiór reakcji zmiany konfiguracji, 61
 zbiór zdarzeń, 14
 zbiór zdarzeń czasowych, 14
 zdarzenie, 15
 zdarzenie czasowe, 14
 zgodność strukturalna, 17
 ZKW, 68
 zmiana konfrontacji pierwotnej, 66
 zmiana konfrontacji wtórnej, 68

13 Literatura

Literatura

- [1] 11th IEEE International Workshop on Rapid System Prototyping. *RSP 2000*, Paris, jun 2000. Springer-Verlag Berlin Heidelberg.
- [2] R. Alur, C. Courcoubetis, D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1983.
- [3] T. Aoshima, T. Ando, N. Yonezaki. Consistency checking of behavioural modeling in UML statechart diagrams. Patrz Information Modeling [31], strony 179–196.
- [4] L. Apvrille, P. de Saqui-Sannes, C. Lohr, P. Senac, J.-P. Courtiat. A new UML profile for real-time system formal design and validation. Patrz *UML2001* [58], strony 287–301.
- [5] M. Beeck. A concise compositional statecharts semantics definition. Patrz FORTE [19], strony 335–350.
- [6] M. Beeck. Formalization of UML–statecharts. Patrz *UML2001* [58], strony 406–421.
- [7] B. Berthomieu, M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Softw. Eng.*, 17(3):259–273, 1991.
- [8] B. Berthomieu, M. Menasche. An enumerative approach for analyzing time petri nets. *IFIP Congress Series*, 9:41–46, 1983.
- [9] P. Bhaduri, R. Venkatesh. Formal consistency of models in multi-view modeling. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 149–159. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [10] J.-P. Bodeveix, T. Millan, Ch. Percebois, Ch. Le Camus, P. Bazex, L. Feraud. Extending OCL for verifying UML models consistency. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 75–90. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [11] G. Booch, J. Rumbaugh, I. Jacobson. *UML przewodnik użytkownika*. Inżynieria oprogramowania. Wydawnictwo Naukowo–Techniczne, Warszawa, 2002.

- [12] G. Butler. Easy verification of behavioural subtyping in common cases. *Information Processing Letters*, 55:57–58, 1995.
- [13] A. David, M. O. Möller. From HUPPAAL to UPPAAL: A translation from hierarchical timed automata to flat timed automata. Raport instytutowy RS-01-11, BRICS, March 2001.
- [14] A. David, M. O. Möller, W. Yi. Formal verification of uml statecharts with real-time extensions. *FASE '02: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, strony 218–232. Springer-Verlag, 2002.
- [15] J. Derrick, D. H. Akehurst. A framework for UML consistency. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 30–45. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [16] K. Diethers, U. Gortz, M. Huhn. Model Checking UML Statecharts with Time. In *Jurjens et al. [19]*, strony 35–51, 2002.
- [17] G. Engels, R. Heckel, M. J. Küster. Rule-based specification of behavioral consistency based on the uml meta-model. Patrz *UML2001* [58], strony 272–286.
- [18] G. Engels, J. M. Küster, R. Heckel, L. Groenewegen. A methodology for specifying and analyzing consistency of object-oriented behavioral models. *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, strony 186–195. ACM Press, 2001.
- [19] *FORTE/PSTV 2000*. Kluwer, 2000.
- [20] S. Graf, L. Mounier, redaktorzy. *Model Checking Software, 11th International SPIN Workshop, Barcelona, Spain, April 1-3, 2004, Proceedings*, wolumen 2989 serii *Lecture Notes in Computer Science*. Springer, 2004.
- [21] C. Gryce, A. Finkelstein, Ch. Nentwich. Lightweight checking for UML based software development. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 124–132. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [22] D. Harel. On the behavior of complex object-oriented systems. Patrz Hofmann i Schürr [27], strony 11–15.

- [23] D. Harel, O. Kupferman. On object systems and behavioral inheritance. *IEEE Transactions on Software Engineering*, 28(9):889–903, sep 2002.
- [24] D. Harel, O. Kupferman, M. Y. Vardi. On the complexity of verifying concurrent transition systems. Patrz Mazurkiewicz i Winkowski [43], strony 258–272.
- [25] J. H. Hausmann, R. Heckel, S. Sauer. Extended model relations with graphical consistency conditions. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 61–74. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [26] B. Hnatkowska, Z. Huzar, L. Kuźniarz, L. Tuzinkiewicz. A systematic approach to consistency within UML based software development process. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 16–29. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [27] P. P. Hofmann, A. Schürr, redaktorzy. *OMER - Object-Oriented Modeling of Embedded Real-Time Systems, GI-Workshops OMER-1 & OMER-2 May 28/29, 1999 & May 10-12, 2001, Herrsching am Ammersee, Germany*, wolumen 5 serii LNI. GI, 2001.
- [28] Z. Huzar. *Wprowadzenie do języka UML*. Wojskowa Akademia Techniczna, Warszawa, 1999.
- [29] Z. Huzar. Semantyka czasu rzeczywistego map stanów w uml. Patrz Systemy czasu rzeczywistego 2002 [57], strony 3–22.
- [30] S.-H. Hwang, Y. Tsujino, N. Tokura. A stepwise inheritance framework for object behavior models. *IEICE Trans. INF. & Syst.*, E80-D(5):573–583, may 1997.
- [31] 12th *European-Japanese Conference on Information Modelling and Knowledge Bases*, Krippen, Germany, May 27–30 2002. Kluwer.
- [32] U. Kelter, M. Monecke, Y. Piao. Checking consistency constraints in xml-based development environments.
- [33] R. Klimek. Zastosowanie logiki temporalnej w specyfikacji i weryfikacji oprogramowania – w stronę czasu rzeczywistego. Patrz Systemy czasu rzeczywistego 2000 [56], strony 61–115.
- [34] A. Knapp, S. Merz, Ch. Rauh. Model checking timed UML state machines and collaborations. W. Damm, E.-R. Olderog, redaktorzy, *7th Intl. Symp. Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT 2002)*, wolumen 2469

serii *Lecture Notes in Computer Science*, strony 395–414, Oldenburg, Germany, Wrzesień 2002. Springer-Verlag.

- [35] S. Kuske. A formal semantic of UML state machines based on structured graph transformation. Patrz *UML2001* [58], strony 241–256.
- [36] K. Lano, D. Clark, K. Androutsopoulos. Formalising inter-model consistency of the UML. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 133–148. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [37] D. Latella, I. Majzik, M. Massink. Automatic verification of behavioural subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, 11:637–664, 1999.
- [38] R. Levinson, G. Ellis. Multilevel hierarchical retrieval. *Knowledge-Based Systems*, 5(3):233–244, sep 1992.
- [39] P. Lincoln, J. C. Mitchell. Algorithmic aspects of type inference with subtypes. *POPL '92: Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, strony 293–304. ACM Press, 1992.
- [40] B. Liskov, J. M. Wing. A new definition of the subtype relation. *ECOOP '93: Proceedings of the 7th European Conference on Object-Oriented Programming*, strony 118–141. Springer-Verlag, 1993.
- [41] B. Liskov, J. M. Wing. Specifications and their use in defining subtypes. *OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, strony 16–28. ACM Press, 1993.
- [42] W. Liu, S. Easterbrook, J. Mylopoulos. Rule based detection of inconsistency in UML models. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 106–123. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [43] Antoni W. Mazurkiewicz, Józef Winkowski, redaktorzy. *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, wolumen 1243 serii *Lecture Notes in Computer Science*. Springer, 1997.
- [44] I. Ober, S. Graf, I. Ober. Validation of uml models via a mapping to communicating extended timed automata. Patrz Graf i Mounier [20], strony 127–145.

- [45] Object Management Group, www.omg.com. *OMG Unified Modeling Language Specification Version 1.5 formal/03-03-01*, March 2003.
- [46] M.-H. Park, K.-S. Bang, J.-Y. Choi. Equivalence checking of two statechart specifications. Patrz *RSP2000* [1], strony 46–51.
- [47] S. Polovina, J. Heaton. An introduction to conceptual graphs. *AI Expert*, may 1992.
- [48] A. Półrola. Weryfikacja własności tctl dla sieci petriego z czasem. *Materiały IX Konferencji Systemy Czasu Rzeczywistego (SCR'02)*, strony 141–153. Instytut Informatyki Politechniki Śląskiej, 2002.
- [49] H. Rasch, H. Wehrheim. Consistency between UML classes and associated state machines. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 46–60. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [50] J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, United States of America., 2005.
- [51] A. J. H. Simons. the compositional properties of UML statechart diagrams, 2000.
- [52] J. L. Sourrouille. A framework for the definition of behavior inheritance. *JOOP*, 9(1):17–21, 1996.
- [53] J. L. Sourrouille. Should subclasses inherit all states and transitions? *the Report on Object Analysis and Design*, 2(6):19–21, mar-apr 1996.
- [54] J. L. Sourrouille. UML behavior: Inheritance and implementation in current object-oriented languages. Patrz *UML99* [59], strony 457–72.
- [55] J. L. Sourrouille, G. Caplat. Checking UML model consistency. Ludwik Kuzniarz, Gianna Reggio, Jean Louis Sourrouille, Zbigniew Huzar, redaktorzy, *Blekinge Institute of Technology, Research Report 2002:06. UML 2002, Model Engineering, Concepts and Tools. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials*, strony 1–15. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002.
- [56] *VII Konferencja Systemy Czasu Rzeczywistego*, Kraków, wrzesień 25–28 2000. Katedra Automatyki Akademii Górniczo-Hutniczej.
- [57] *IX Konferencja Systemy Czasu Rzeczywistego*, Ustroń, wrzesień 16–19 2002. Instytut Informatyki Politechniki Śląskiej w Gliwicach.

- [58] UML 2001 LNCS 2185. *UML 2001*, Heidelberg, 2001. Springer-Verlag Berlin Heidelberg.
- [59] UML '99 LNCS 1723. *UML '99 - Unified Modeling Language. Beyond the Standard. Second International Conference*, Berlin, 1999. Springer-Verlag Berlin.
- [60] W.M.P. van der Aalst. Inheritance of dynamic behavior in uml.
- [61] J.M. Wing, J. Ockerbloom. Respectful type converters. *IEEE Transactions on Software Engineering*, 26(7):579–593, jul 2000.
- [62] A. M. Zaremski, J. M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, oct 1997.