AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI

Piotr Szwed

ANALIZA POPRAWNOŚCI OPROGRAMOWANIA WSPÓŁBIEŻNEGO Z WYKORZYSTANIEM FUNKCJI OBSERWACJI

Rozprawa doktorska

Promotor: prof. dr hab. inż. Tomasz Szmuc

Kraków 1999

Panu prof. dr hab. inż. Tomaszowi Szmucowi,
promotorowi pracy, składam serdeczne podziękowania
za liczne i cenne dyskusje, słowa zachęty oraz cierpliwość

Spis treści

1. Wstęp

Tematem pracy jest specyfikacja wymagań opisujących poprawność systemów współbieżnych oraz ich analiza z wykorzystaniem metod automatycznej weryfikacji. Zagadnienia te będą szczegółowo opisane w kolejnych rozdziałach, natomiast niniejszy rozdział stanowi wprowadzenie do problematyki będącej przedmiotem pracy.

Rozważania wstępne rozpoczniemy od charakterystyki dziedziny oraz omówienia roli badań nad poprawnością systemów współbieżnych i tworzeniem zautomatyzowanych metod jej dowodzenia. Dalej podane będą motywacje, którymi kierował się autor przy wyborze opisanego w pracy kierunku badań szczegółowych. W kolejnym podrozdziale zostaną sformułowane cel i teza pracy. Na zakończenie omówiona będzie zawartość poszczególnych rozdziałów niniejszej rozprawy.

1.1 Ogólna charakterystyka dziedziny pracy

1.1.1 Systemy współbieżne

Systemy współbieżne to systemy, które przejawiają równoległe działanie w czasie. Zadaniem badań nad współbieżnością jest opis potencjalnej równoległości i rozwiązywanie zagadnień związanych z problemami synchronizacyjnymi i komunikacyjnymi [Ben-Ari 82]. Badania te zazwyczaj traktowane są jako dziedzina informatyki; ich przedmiotem jest wówczas działanie sprzętu komputerowego i oprogramowania. Jednakże rozwijane teorie stosują się także do innych systemów obejmujących systemy telekomunikacyjne, produkcyjne, mechaniczne, obiegu dokumentów, itp.

W informatyce systemy współbieżne rozważane są w opozycji do programów sekwencyjnych. Te drugie [Lamport 94] mogą być traktowane jako funkcje odwzorowujące wejścia w wyjścia. W przypadku systemów współbieżnych (systemów reaktywnych [HD 85]) opis działania systemu musi obejmować zbiór możliwych jego zachowań.

1.1.2 Rola weryfikacji i walidacji w cyklu życia oprogramowania

Typowy cykl życia oprogramowania (model kaskadowy) obejmuje następujące fazy [Calvez 93, HS 92]:

- analizy wymagań
- projektowania (wstępnego i szczegółowego)
- implementacji modułów i testowania
- integracji modułów i testowania
- testów jakościowych
- użytkowania i konserwacji

W zależności od metody rozwoju oprogramowania fazy te mogą się częściowo przeplatać. Na przykład ostatnia wersja standardu przyjętego w armii amerykańskiej [US 94] wydziela etapy analizy wymagań systemu i projektu architektury. W ich wyniku zostaje przeprowadzony podział systemu na komponenty, dla których jest przeprowadzana dalsza analiza wymagań i projektowanie. Rezultatem wczesnych faz cyklu życia oprogramowania (analizy wymagań i projektowania) jest zazwyczaj zbiór dokumentów. Rezultatem fazy implementacji jest kod źródłowy oprogramowania.

Standardy rozwoju oprogramowania zalecają zastosowanie metod analizy wymagań i projektowania oprogramowania wspartych automatycznymi technikami oferowanymi przez

systemy CASE. Za najbardziej rozpowszechnione należy uznać obecnie metody analizy strukturalnej i metody obiektowe.

IEEE Standard Glossary of Software Terminology [IEEE 83] definiuje następująco pojęcia weryfikacji i walidacji.

- *Weryfikacja* to proces służący ocenie czy produkt danej fazy rozwoju spełnia warunki narzucone na początku tej fazy.
- *Walidacja* to proces służący ocenie, czy oprogramowanie powstałe w wyniku cyklu rozwoju oprogramowania jest zgodne z wymaganiami.

Nieformalnie [Boehm 84] weryfikacja służy udzieleniu odpowiedzi na pytanie, „czy produkt jest budowany w właściwy sposób" natomiast walidacja „czy budowany jest właściwy produkt".

Podstawowym celem weryfikacji i walidacji jest możliwie wczesna identyfikacja potencjalnych problemów i błędów projektowych. Porównania kosztów odszukiwania i usuwania błędów dla różnych faz rozwoju wskazują, że dla dużych systemów koszty eliminacji błędów we wczesnych fazach mogą być nawet stukrotnie mniejsze niż w fazie konserwacji. W przypadku systemów małych i średnich różnice te są mniejsze (kilkukrotne). Dla niektórych typów oprogramowania wysoki poziom zaufania użytkownika do poprawnego działania jest niezbędnym warunkiem ich akceptacji. Dotyczy to zwłaszcza systemów, którym stawiane są ostre wymagania ze względu na bezpieczeństwo człowieka (medycznych, lotniczych, sterowania reaktorami atomowymi) lub bezpieczeństwo działania (systemy transakcji finansowych).

W [Boehm 84] wymieniono szereg technik weryfikacji i walidacji stosujących się do wczesnych faz cyklu życia oprogramowania. Warto zwrócić uwagę, że zasadnicza część rzeczywistych specyfikacji ma nieformalny (często słowny) charakter lub używa metod graficznych zdefiniowanych formalnie jedynie pod względem składniowym. Stąd też wiele z praktycznie stosowanych technik weryfikacji i walidacji (w tym oferowanych przez narzędzia automatyczne) ukierunkowanych jest na analizę spójności tego typu specyfikacji.

Weryfikacja aspektów dynamicznych, które odgrywają szczególnie istotną rolę w przypadku systemów współbieżnych, obejmuje następujące techniki:

- konstrukcję prostych modeli manualnych opisujących wybrane aspekty działania systemu z użyciem pewnego formalizmu matematycznego i badanie ich własności;
- tworzenie prostych modeli automatycznych wykorzystujących komputer do analizy formuł opisujących modelowany system;
- manualne dowody poprawności systemu w oparciu o metody formalne podtrzymywane często przez narzędzia automatycznego dowodzenia;
- szczegółowe modele automatyczne oparte na technikach symulacyjnych.

Tradycyjnie weryfikacja oparta na metodach formalnych przeciwstawiana jest testowaniu, które nie daje gwarancji, że program zachowuje się poprawnie. (Odpowiedzią na ten zarzut są rozwijane techniki minimalizacji prawdopodobieństwa, że przy testowaniu błędy zostaną pominięte [MMNPNMV 92].)

Z kolei metody formalne dowodzenia poprawności systemów mają ograniczone zastosowanie; ich jakość i wiarygodność, o ile nie są podtrzymywane technikami automatycznymi, uzależniona jest od ludzkich błędów. Prawdopodobieństwo tych

ostatnich rośnie na ogół wraz z długością dowodu. Zaletą właściwie użytych metod formalnych jest pewność ich rezultatów. Pewność ta może jednak być osłabiona, jeśli uwzględnimy fakt, że posługują się one abstrakcyjnym opisem rzeczywistego systemu. Model stworzony do celów weryfikacyjnych zawiera z reguły znaczne uproszczenia i może nie uwzględniać wielu istotnych aspektów systemu.

Zarzutem często stawianym metodom formalnym jest niezrozumiałość specyfikacji dla inżynierów oprogramowania. Stąd naturalna jest tendencja widoczna w narzędziach CASE do specyfikacji oprogramowania za pomocą języków graficznych o ściśle zdefiniowanej składni. Na podstawie stworzonych specyfikacji budowany jest następnie model semantyczny (najczęściej w postaci modułu wykonywalnego), którego zachowanie może być obserwowane (podczas symulacji) lub weryfikowane. Proces generacji modelu pozwala dodatkowo wykryć potencjalne niespójności w specyfikacji.

1.1.3 Poprawność

Definicje poprawności dla programów sekwencyjnych pochodzą od Floyda [Floyd 67]. Mówiąc nieformalnie, program jest uważany za poprawny jeśli zatrzymuje się i zwraca właściwą odpowiedź.

Program S uważany jest za *częściowo poprawny*, jeśli uruchomiony dla danych wejściowych spełniających prewarunek P zatrzyma się i obliczone dane będą spełniały postwarunek Q lub też nie zatrzyma się. Warunkiem dodatkowym *całkowitej poprawności* jest zatrzymanie programu.

Systemy współbieżne wymagają innych definicji poprawności, zwłaszcza, że większość z nich nigdy się nie powinna zatrzymać. W [Lamport 77] zaproponowano podział własności związanych z poprawnością systemów współbieżnych na dwie podstawowe klasy: *bezpieczeństwa* i *żywotności*.

Własność bezpieczeństwa wyrażona jest nieformalnie jako „nic złego nie może się zdarzyć”. Niepożądaną sytuacją może być, np.: blokowanie lub naruszenie zasady wykluczania dla sekcji krytycznej.

Własność żywotności opisuje żądanie, by po spełnieniu pewnego warunku w końcu nastąpiło pożądane zdarzenie, jak: zakończone sukcesem zatrzymanie programu, uzyskanie dostępu do zasobu lub aktywacja procesu składowego.

Własność częściowej poprawności jest zaliczana do klasy bezpieczeństwa, poprawność całkowita wymagająca zakończonego właściwą odpowiedzią zatrzymania programu należy do klasy własności żywotności. Ich definicje przyjmują elegancką postać w logice temporalnej.

1.1.4 Poprawność procesu projektowania

Przedmiotem zainteresowania wielu prac poświęconych systemom współbieżnym jest poprawność procesu ich tworzenia. Często stosowaną metodą tworzenia systemów, zgodną modelem kaskadowym, jest stopniowe uściślanie (ang. *stepwise refinement*) [Wirth 71]. Proces projektowania rozpoczyna się od wstępnej definicji systemu S_0 nazywanego jego specyfikacją. Jest ona opisem najbardziej abstrakcyjnym, pomijającym nieistotne szczegóły jego zachowania. Specyfikacja pozostawia do późniejszego rozstrzygnięcia wiele elementów, które są ustalane w wyniku decyzji projektowych.

Poszczególne etapy projektowania systemu owocują kolejno powstającymi specyfikacjami. Składają się one na ciąg: $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n$

W każdym etapie realizowane jest przejście pomiędzy specyfikacjami S_{i-1} i S_i . W jego wyniku dochodzi do uściślenia specyfikacji S_{i-1} poprzez dodanie kolejnych szczegółów i podjęcie decyzji projektowych o sposobie implementacji wybranych elementów specyfikacji wejściowej. Wynikiem ostatniego etapu jest stworzenie *implementacji* systemu S_n.

Ogólnie, można rozważyć dwa rodzaje decyzji, które są podejmowane podczas przejścia pomiędzy specyfikacjami. Część decyzji może dodawać informacje o tym *co* system powinien robić, uściślając w ten sposób cel działania systemu. (Zazwyczaj podejmowane są one w fazie analizy wymagań.) Innego typu decyzje ustalają *jak* system powinien implementować wymagane funkcje.

Dla etapów projektowania, w których uściślenie następuje poprzez podjęcie decyzji *jak* system powinien realizować wymagania opisy wejściowe i wyjściowe nazywane są często *specyfikacją* i *implementacją.* Głównym celem formalnych teorii opisujących systemy współbieżne jest dowodzenia poprawności przejścia pomiędzy specyfikacją S_{i-1} i implementacją S_i. Stąd często uściślanie traktowane jest jako dowolna transformacja specyfikacji, która zgodnie z przyjętymi kryteriami może zostać uznana za poprawną.

Relacje poprawnościowe opisujące zależności pomiędzy specyfikacją i implementacją można najogólniej scharakteryzować jako relacje równoważności i częściowego uporządkowania [Glabbeek 90, GG 98]. Odpowiadają one dwóm sytuacjom:

- Mimo, że opis implementacji S_i zawiera znacznie więcej dodatkowych szczegółów dotyczących jej zachowania (akcji, zmiennych, instrukcji) mogą być one traktowane jako wewnętrzne i nieobserwowalne. Po ich abstrakcji (ukryciu) specyfikacje S_{i-1} oraz S_i mogą być uznane za równoważne (w ramach pewnego modelu nadającego semantykę specyfikacjom).
- Tworząc opis implementacji S_i dokonano ustaleń, które były pozostawione jako alternatywy do rozstrzygnięcia w dalszych fazach rozwoju systemu. Z tego powodu specyfikacje S_{i-1} i S_i nie mogą być traktowane jako równoważne. Stosowną relacją pomiędzy klasami systemów opisanymi przez specyfikacje jest relacja częściowego porządku określająca warunki, kiedy system S_i może być uznany za poprawną implementacją specyfikacji S_{i-1}.

Relacje równoważności pomiędzy systemami budowane są najczęściej wokół zależności pomiędzy zbiorami obliczeń systemu (ścieżek) lub relacji bisymulacji [Milner 89]. Są one krótko scharakteryzowana w następnym rozdziale pracy.

Relacje częściowego uporządkowania definiowane są różnie, w zależności od przyjętej metody formalnej. Obejmują one takie relacje jak symulacja i uściślanie [KMP 94], zgodność z wymaganiami wyrażonymi jako predykaty [Hoare 85], formuły logiki temporalnej [Pnueli 77; MP 80], poprawność względna [Szmuc 89b].

Jak można zauważyć, zaletą poprawności rozpatrywanej jako równoważność specyfikacji jest ten sam język użyty do opisu specyfikacji i implementacji. Cechy tej nie mają pozostałe formalizmy. Wyjątkiem są algebraiczne metody poprawności względnej.

1.2 Motywacje

Inspiracją do podjęcia badań będących przedmiotem tej rozprawy były prace nad zastosowaniem algebraicznych metod poprawności względnej [Szmuc 88, 89a, 89b, 91, 92, 93, 97a, 97b] do weryfikacji systemów opisanych językiem LACATRE [Schwarz 92;

MSSS 94; SS 97c]. Systemy te zazwyczaj były modelowane w postaci zbioru komunikujących się maszyn skończenie stanowych [SS 94, 95a, 95b, 96; 97a, 97b; SSSS 94, 96].

W algebraicznych metodach poprawności względnej przedmiotem weryfikacji jest sekwencyjny opis systemu współbieżnego w postaci procesu Pawlaka [Pawlak 69]. Proces ten definiowany jest jako relacyjna struktura z wyodrębnionym zbiorem stanów początkowych i końcowych. Stany procesu w zależności od interpretacji mogą odpowiadać wektorom etykiet instrukcji i wartościom zmiennych.

Wymagania stawiane badanemu systemowi definiowane są również w postaci procesu sekwencyjnego; nazywany jest on procesem kryterialnym.

Proces kryterialny i weryfikowany mogą być traktowane jako specyfikacja S_{i-1} oraz implementacja S_i etapu uściślania. Odwzorowanie pomiędzy tymi dwiema warstwami zdefiniowane jest w postaci relacji łączącej wybrane, tzw. charakterystyczne stany procesu weryfikowanego ze stanami procesu kryterialnego.

W ujęciu nieformalnym, relacja poprawności pomiędzy dwoma specyfikacjami opiera się na wymaganiu, aby stany charakterystyczne w procesie weryfikowanym były osiągane w kolejności zgodnej ze strukturą przejść procesu kryterialnego.

1.2.1 Wybór akcji jako dziedziny analizy

Algebraiczne metody badania poprawności (patrz podrozdział 2.6) definiują model poprawności oparty na asocjacji w dziedzinie stanów. Alternatywną podstawą do formułowania zagadnienia poprawności jest oparcie się na akcjach, które mogą być utożsamiane z przejściami w procesie.

Pojęcie akcji występuje w wielu specyfikacjach, zwłaszcza opartych na dekompozycji funkcjonalnej. Mogą one przybierać nieco inne nazwy, np.: funkcja, proces transformujący dane, metoda. Przy opisie aspektów funkcjonalnych aplikacji w sposób naturalny przyjmuje się, że na daną akcję składają się inne akcje lub też mówi się o akcjach polegających na przesyłaniu komunikatów, których kolejność opisana jest pewnymi scenariuszami.

Możliwy jest oczywiście opis procesu, w którym poszczególnym stanom przyporządkowane zostaną symbole akcji (czy etykiety instrukcji). Takie rozwiązanie widoczne było w wielu przykładach zamieszczonych w pracy [Szmuc 89b]. Z punktu widzenia modelowania zachowania programu istotne jest, czy tak utworzony stan należy interpretować jako fakt uzyskania gotowości do wykonania akcji, jej wykonywania czy też jej zakończenia. Oparcie się na jednym z powyższych założeń może prowadzić do różnych modeli semantycznych systemu współbieżnego.

Wybór akcji jako dziedziny analizy podyktowany był przede wszystkim pragnieniem ułatwienia specyfikacji wymagań odzwierciedlających zależności występujące przy dekompozycji funkcjonalnej. W wielu przypadkach specyfikacja wymagań oraz zależność pomiędzy różnymi warstwami specyfikacji przybiera prostszą i bardziej ekspresywną postać, jeśli prowadzona jest dla akcji.

1.2.2 Model poprawności sformułowany bezpośrednio dla systemu procesów

W ujęciu podstawowym weryfikacja systemu procesów współbieżnych poprzedzona jest fazą tworzenia procesu stanowiącego sekwencyjny opis działania systemu. Proces ten powstaje przez wykonanie z przeplotem wszystkich przejść w procesach składowych.

Doświadczenia zebrane w trakcie prac nad zastosowaniem algebraicznych metod poprawności względnej do weryfikacji oprogramowania współbieżnego zadanego w postaci zbioru komunikujących się maszyn skończenie stanowych skłaniają autora do wniosku, że wygodna z punktu widzenia formalnego modularność i podział na etapy weryfikacji (por. podrozdział 2.6) są w praktycznych zastosowaniach mało wygodne.

Alternatywnym rozwiązaniem jest sformułowanie modelu poprawności bezpośrednio dla systemu procesów i prowadzenie weryfikacji równolegle z budowa procesu sekwencyjnego. Rezultaty prac opartych na takim podejściu zaprezentowano w [SS 94; SSSS 94; SS 95a, 95b].

Rozwiązanie to jest preferowane przez autora ze względu na fakt, że weryfikacja procesu sekwencyjnego skonstruowanego na podstawie zbioru procesów współbieżnych daje trudniejsze w interpretacji rezultaty dotyczące źródeł błędów. Struktura procesu opisującego zachowanie systemu na ogół bardzo jest odległa od struktur procesów składowych.

Z punktu widzenia zastosowań oba podejścia są w praktyce równoważne. Jeśli generacja procesu sekwencyjnego jest możliwa do przeprowadzenia w akceptowalnym czasie i w ramach dostępnych zasobów, wówczas połączenie generacji z równoczesną weryfikacją jest również możliwe (na ogół w czasie zwiększonym w niewielkim stopniu). Największym problemem, który ogranicza w praktyce stosowalność narzędzi automatycznej weryfikacji dla systemu procesów współbieżnych jest złożoność obliczeniowa algorytmu generacji procesu sekwencyjnego opisującego system będący przedmiotem analizy.

Z tego względu wszelkie (ewentualne) redukcje powinny zostać przeprowadzone wcześniej bezpośrednio w procesach składowych. Możliwość ich przeprowadzenia uzależniona jest oczywiście od znajomości asocjacji pomiędzy specyfikacją weryfikowaną a kryterialną i jej postaci.

1.2.3 Instancje sterowania

Zastosowanie algebraicznych metod badania poprawności względnej dla procesów wchodzących w skład systemów reagujących na zdarzenia zewnętrzne lub przesyłających dane za pośrednictwem buforów prowadzi do problemów związanych właściwym doborem kryterium poprawności.

Wynikają one z tego, że proces kryterialny interpretowany jest w sposób nie pozwalający na łatwy zapis wymagań dla systemów, w których w wyniku otrzymanych z zewnątrz zdarzeń pojawiają się nowe instancje sterowania, a ich obsługa przeplatana jest z przetwarzaniem instancji „zmagazynowanych” wewnątrz systemu w procesach modelujących bufory (kolejki komunikatów lub semafory).

W konsekwencji prowadzi to na ogół do mało czytelnych specyfikacji procesu kryterialnego. Jego struktura staje się bardzo skomplikowana i mało związana z rzeczywistą ścieżką przetwarzania informacji. Jest to zjawisko bardzo niekorzystne z punktu widzenia zastosowania w cyklu życia oprogramowania, ponieważ uniemożliwia bezpośrednie przekształcenie wymagań w proces kryterialny, ale zmusza do tworzenia sztucznego procesu będącego pochodną wymagań i struktury przetwarzania zdefiniowanej w kolejnej warstwie specyfikacji.

1.2.4 Symetryczna postać zagadnienia poprawności

W obecności rozszerzeń w definicji procesu kryterialnego [Szmuc 93, 97a, 97b] oryginalne metody weryfikacji poprawności względnej definiują poprawność dla dwóch różnych obiektów. Utrudnia to prowadzenie wieloetapowej weryfikacji wymagań w ciągu stopniowego uściślania, gdzie zakłada się, że specyfikacja danej warstwy stanowi kryterium poprawności dla warstwy następnej.

Celowym wydaje się dążenie do ujednolicenia postaci obiektów modelujących specyfikacje weryfikowaną i kryterialną.

Założenie, że przedmiotem weryfikacji poprawności powinien być system procesów skłania do wniosku, że równie ogólny model powinien zostać przyjęty dla specyfikacji kryterialnej.

1.2.5 Zakres badań szczegółowych

Rozważane w pracy rozwiązania są metodologicznie bliskie zasadom konstrukcji modeli oraz narzędzi weryfikacji zaproponowanych dla algebraicznych metod poprawności względnej. Najbardziej istotną różnicą jest wybór podstawowego elementu opisu. Jak już wspomniano, nie są nim stany lecz akcje.

Problem postawiony jest podobnie, jak w metodach poprawności względnej. Badane są relacje poprawnościowe pomiędzy dwoma modelami specyfikacji: weryfikowanym oraz kryterialnym.

Reprezentacją zachowania analizowanych specyfikacji jest ciąg wektorów wykonanych akcji. (Wektory te mogą być traktowane jako wielozbiory.) Wybór opisu zachowania wpływa na postać konstruowanych modeli. Są one definiowane jako zbiór nierówności i równań liniowych.

Badanie istnienia relacji poprawnościowych pomiędzy dwoma specyfikacjami wymaga zdefiniowania odwzorowania przekształcającego reprezentacje zachowania modelu weryfikowanego w dziedzinę zachowań modelu kryterialnego. Odwzorowanie to nazywane jest w pracy *funkcją obserwacji*.

Rozważane są trzy typy funkcji obserwacji:

- homomorficzna (liniowa)
- niehomomorficzna
- warunkowa

Homomorficzna funkcja obserwacji przekształca akcje występujące w badanym systemie w zbiory równolegle wykonanych akcji. W ten sposób, podobnie jak w algebrach procesów, reprezentacja zachowania bardziej szczegółowego jest przekształcana w reprezentację bardziej abstrakcyjną, z której usuwane są nieistotne elementy (nieobserwowalne akcje) przy zachowaniu ogólnej struktury.

Specyfikacja *niehomomorficznej funkcji obserwacji* pozwala na wyrażenie wymagań dotyczących uściślania akcji, czyli zastępowania akcji traktowanych atomowo na danym poziomie abstrakcji zbiorami akcji, które składają się na pewien proces. Dokonywane obserwacje nie polegają wyłącznie na zmianie etykiet i ukrywaniu pojedynczych akcji występujących w modelu weryfikowanym, ale są uzależnione od historii wykonania.

Warunkowa funkcja obserwacji stosuje się do rozszerzonej postaci modelu weryfikowanego uwzględniającej definicję danych. Postać takich modeli jest na ogół mniej obszerna i bliższa rzeczywistym językom specyfikacji. Prowadzone obserwacje pozwalają na rozróżnienie akcji dokonywanych dla różnych wartości zmiennych.

1.3 Cel pracy i teza

Omówione w poprzednim podrozdziale motywacje, jak również doświadczenia autora skłaniają do sformułowania następującego celu pracy:

1. Określenie i zbadanie modelu poprawności względnej oprogramowania współbieżnego, w którym zapis odwzorowania pomiędzy rzeczywistym i wymaganym zachowaniem analizowanej specyfikacji wyrażony jest poprzez funkcje obserwacji.
2. Analiza zastosowań trzech typów funkcji obserwacji: homomorficznej, niehomomorficznej (funkcji z pamięcią historii wykonania) oraz funkcji obserwacji warunkowej dla algebraicznego modelu danych.
3. Opracowanie metod automatycznej analizy poprawności oprogramowania współbieżnego dla modelu opartego na funkcjach obserwacji, a następnie wykonanie prototypowej implementacji algorytmów oraz przeprowadzenie testów.

W wyniku przeprowadzonych badań formułuje się następującą tezę:

> Zaproponowane modele poprawności względnej wykorzystujące różne typy funkcji obserwacji mogą znaleźć bezpośrednie zastosowanie w fazach analizy wymagań i projektowania oprogramowania współbieżnego. Opracowane metody analizy poprawności mogą zostać w efektywny sposób użyte do przeprowadzenia pełnej lub częściowej automatycznej weryfikacji.

Powyższa teza jest następnie dowodzona w kolejnych rozdziałach niniejszej rozprawy.

1.4 Zawartość pracy

Praca podzielona jest na dziewięć rozdziałów (wliczając w to bieżący rozdział wprowadzający oraz podsumowanie).

W **rozdziale 2** zamieszczono przegląd literatury związanej z modelowaniem systemów współbieżnych, różnymi metodami specyfikacji wymagań i ich dowodzenia lub automatycznej weryfikacji.

Pierwszy z podrozdziałów poświęcony jest sieciom Petriego. Reprezentacja macierzowa sieci Petriego jest bliska modelom stosowanym w pracy. Różnicą jest poziom szczegółowości. Modele formułowane w pracy utożsamiają akcje z łukami sieci, a nie z tranzycjami. Bliższe są opisanym dalej specyfikacjom systemów w postaci zbioru synchronicznie komunikujących się maszyn skończenie stanowych.

Kolejny fragment rozdziału 2 poświęcony jest opierającym się na akcjach algebrom procesów: CSP i CCS. Wiele z pojęć definiowanych dla tych algebr można odnaleźć w dalszych rozdziałach pracy. Ścieżkowy opis zachowania systemów, bliski modelom semantycznym CSP, jest często wykorzystywany przy analizie przykładów obrazujących omawiane zagadnienia.

Wymagania opisujące poprawność systemów współbieżnych formułowane są często z użyciem logiki temporalnej. Podejście to jest przedstawione nieco szerzej (a zwłaszcza metody automatycznej weryfikacji własności opisanych za pomocą liniowej logiki temporalnej).

Najobszerniejszą część rozdziału stanowi przegląd prac dotyczących algebraicznych metod badania poprawności względnej. Obejmuje on: definicję procesu, postać zagadnienia poprawności w postaci pary procesów weryfikowanego i kryterialnego oraz relacji kryterialnej, definicje pojęć poprawności częściowej i całkowitej oraz metody dowodzenia poprawności wykorzystujące konstrukcję procesu sprzężonego. Lektura tego podrozdziału pozwoli przy przejściu do dalszych części pracy stwierdzić, jak poszczególne pojęcia i konstrukcje zostały przeformułowane dla zmienionych modeli specyfikacji i postaci zagadnienia poprawności.

Na zakończenie omówiono typową postać algorytmów automatycznej weryfikacji (walidacji) opartych na przeszukiwaniu w głąb drzewa stanów procesu oraz metody przeszukiwania częściowego, w których dąży się do ograniczenia wpływu eksplozji stanów na jakość analizy. Metody te nie są bezpośrednio stosowane w implementowanych algorytmach; są cytowane dla wskazania możliwych kierunków przyszłego rozwoju.

Dalsze rozdziały stanowią właściwy opis własnych prac autora.

Rozdział 3 stanowi wprowadzenie do szczegółowych zagadnień omawianych w pracy. Pokazuje na wybranych przykładach ideę specyfikacji wymagań z użyciem funkcji obserwacji i postać zagadnienia poprawności. Definiuje homomorficzną (liniową) i niehomomorficzną funkcję obserwacji.

Rozdział 4 omawia konstrukcję modelu liniowego dla procesu sekwencyjnego, sposób odzwierciedlenia w modelu specyficznych zależności występujących w interpretacji zachowania procesu kryterialnego dla algebraicznych metod poprawności względnej oraz konstrukcję modelu opisującego system synchronicznie komunikujących się procesów. Zwłaszcza ten ostatni model jest Uznany w pracy za najbardziej interesujący (jako stosowny do opisu zbioru komunikujących się maszyn skończenie stanowych). Opisane są zasady modelowania pozwalające na rozróżnienie pomiędzy blokowaniem i poprawnym zakończeniem działania (osiągnięciem stanu końcowego).

Dalej wprowadzone są pojęcia *rozwiązania* odpowiadającego wektorowi (wielozbiorowi) wykonanych akcji oraz dualne pojęcie *stanu* modelu. Następnie omówione są zasady wykonania modelu przez wykonanie ciągu *przejść elementarnych* odpowiadających jednej lub dwu synchronizowanym akcjom. Wykonanie modelu opiera się na przeplocie; jako następne przejście wybierane jest w sposób niedeterministyczny jedno spośród kilku przejść dopuszczalnych w danym stanie.

W rozdziale omówiono także zbieżność rozważanych modeli z sieciami Petriego.

Rozdział 5 koncentruje się na liniowej (homomorficznej) funkcji obserwacji. Definiuje on poprawność względną ciągu rozwiązań (jako reprezentacji zachowania systemu), poprawność częściową (definiowaną jako poprawność wszystkich ciągów rozwiązań) oraz poprawność całkowitą. W przypadku poprawności całkowitej warunki zaostrzane są o konieczność dokonania obserwacji wszystkich akcji specyfikacji kryterialnej oraz o brak dywergencji. (Pojęcie dywergencji zapożyczone zostało z formalizmu CSP.)

Pomocniczymi pojęciami, które mogą być stosowane przy badaniu własności systemów są pojęcia: poprawności potencjalnej i spójności wymagań.

Formalną konstrukcją służącą dowodzeniu poprawności jest, podobnie jak dla algebraicznych metod poprawności względnej, *proces sprzężony*. Jego definicja dostosowana jest do postaci rozważanych modeli. Podane twierdzenia uzasadniają zastosowanie procesu sprzężonego w weryfikacji poprawności.

Rozdział 6 omawia zagadnienia związane z niehomomorficzną obserwacji. Przedstawia on zmodyfikowaną definicję poprawności względnej ciągu rozwiązań (która jest podstawową dla dalszych definicji poprawności częściowej i całkowitej).

Niehomomorficzna funkcja obserwacji przekształca zbiory akcji występujące w modelu weryfikowanym w akcje modelu kryterialnego. Konstruując definicję poprawności względnej ciągu rozwiązań przyjęto założenie, że atomowość akcji modelu kryterialnego przenosi się na rozłączność czasową odpowiadających im zbiorów akcji modelu weryfikowanego. (Czyli, że aktywność podprocesów specyfikacji weryfikowanej, które odwzorowywane są w kolejne akcje kryterialne nie może się przeplatać w modelu semantycznym opartym na przeplocie.)

Formalnie wymaganie to opisane jest przez dwa pojęcia: *odwrotnej dopuszczalności* i *lokalnej osiągalności*. Odwrotna dopuszczalność jest własnością łatwą do sprawdzenia. Konstrukcją służącą badaniu lokalnej osiągalności (która musi uwzględniać informację o historii wykonania) służy *macierz lokalnej osiągalności*.

Macierz lokalnej osiągalności rozszerza stan procesu sprzężonego konstruowanego dla niehomomorficznej funkcji obserwacji. Podobnie jak dla funkcji liniowej podano twierdzenia uzasadniające taką właśnie konstrukcję.

W rozdziale 7 przedstawiono trzy przykłady zastosowań: system pięciu filozofów, protokół z bitem potwierdzenia oraz mniej akademicki przykład specyfikacji bankomatu. Każdy z przedstawionych systemów analizowany był z użyciem prototypowego oprogramowania implementującego opisane w pracy algorytmy.

Rozdział 8 zawiera opis zmodyfikowanego modelu specyfikacji weryfikowanej uwzględniającego definicję danych. Model ten odpowiada popularnym rozszerzeniom dla maszyn skończenie stanowych. Rozważane typy danych to: zmienne wyliczeniowe, licznikowe oraz kolejki komunikatów.

Dla tak zdefiniowanego modelu definiowana jest warunkowa funkcja obserwacji. Istotnym jej elementem jest składowa nazywana *funkcją ekspansji*. Jej zadaniem jest rozróżnienie pomiędzy akcjami wykonywanymi dla różnych wartości zmiennych. Rozróżnienie to zapisywane jest w sposób ogólny – w postaci predykatów, których argumentami są wartości zmiennych. Funkcja ekspansji rozszerza pierwotną przestrzeń rozwiązań specyfikacji weryfikowanej przekształcając ją w przestrzeń akcji, które są dalej poddane bezpośrednio obserwacji za pomocą liniowej lub niehomomorficznej funkcji obserwacji.

W przedstawionej wersji postać zagadnienia jest niesymetryczna – definicje danych nie mogą wystąpić w modelu kryterialnym. W dalszej części rozdziału przedstawiono konstrukcję zmodyfikowanego procesu sprzężonego stosującego się do rozważanego zagadnienia.

Rozdział 9 zawiera krótkie podsumowanie.

W dodatkach **A**, **B** i **C** podano algorytmy: badania spójności wymagań oraz weryfikacji dla liniowej i niehomomorficznej funkcji obserwacji.

2. Przegląd modeli oraz metod specyfikacji i weryfikacji poprawności systemów współbieżnych

2.1 Sieci Petriego

Sieci Petriego są bardzo rozpowszechnionym narzędziem opisu systemów. Umożliwiają one modelowanie takich własności jak: współbieżność, niedeterminizm, przepływy danych, reakcja na asynchroniczne pojawiające się zdarzenia, rozproszenie elementów. W ciągu trzydziestu kilku lat powstało wiele różnych definicji sieci Petriego, często ukierunkowanych na wygodne modelowanie różnych aspektów zachowania systemów.

Charakterystyczną cechą wspólną wszystkich modeli jest to, że równorzędnymi elementami opisu są zarówno stany systemu, jak i akcje (nazywane tu tranzycjami).

Literatura dotycząca sieci Petriego jest bardzo bogata. Przegląd różnych typów sieci można znaleźć w [Reisg 85; Murata 89].

Podstawowymi modelami sieci są:

- sieci warunków i zdarzeń (sieci C/E)
- sieci miejsc i tranzycji (sieci PT)
- sieci kolorowe [Jensen 91, 95-96; Sacha 95]
- sieci obiektowe [Lakos 94, 95]
- sieci czasowe [GMMP 91; Jensen 95-96; Sacha 95].

Modelem, który był przedmiotem największej liczby prac są sieci miejsc i tranzycji. Zostanie on krótko omówiony ze względu na to, że stosowany w pracy model specyfikacji jest mu dość bliski.

2.1.1 Definicja

Sieć Petriego jest dwudzielnym grafem skierowanym, którego wierzchołki należą do dwóch zbiorów: *miejsc* i *tranzycji*. Łuki mogą łączyć wyłącznie miejsca z tranzycjami i tranzycje z miejscami.

Stan systemu opisanego przez sieć nazywany jest *znakowaniem*. Jest to funkcja, która przydziela miejscom wartości nieujemne. Ze względu na symbolikę graficzną wartości te nazywane są żetonami. Żetony mogą być traktowane jak abstrakcyjne obiekty modelujące spełnienie warunków logicznych, obecność obliczonych danych określonego typu, stan sterowania procesów składowych, warunki synchronizacji. Znakowanie oznaczane jest przez M. Jest ono wektorem liczb całkowitych nieujemnych o wymiarze równym liczbie miejsc. Przez $M(p)$ oznaczali będziemy liczbę żetonów w miejscu p.

Łukom mogą być przypisane wagi. Dla modeli wielu systemów (zwłaszcza opisanych warunki logicznymi) wagi nie występują – domyślnie przyjmujemy, że mają one wartość 1.

Tranzycje mogą być utożsamiane ze wystąpieniem zdarzenia, wykonaniem instrukcji, transformacją danych, dokonaniem przekształcenia warunków logicznych, uruchomieniem mechanizmu synchronizacji.

Def. 2-1 Sieć Petriego

Sieć Petriego zdefiniowana jest jako krotka $PN = (P, T, F, W, M_0)$, gdzie:

- P jest skończonym zbiorem miejsc,
- T jest skończonym zbiorem tranzycji,
- $F \subset (P \times T) \cup (T \times P)$ jest skończonym zbiorem łuków,
- $W : F \rightarrow \mathrm{N}$ jest funkcją przydzielającą łukom wagi,
- $M_0 : P \rightarrow \mathrm{N} \cup \{0\}$ jest znakowaniem początkowym.

Przez $N = (P, T, F, W)$ oznaczana jest często sieć bez określonego znakowania początkowego. Przez (N, M_0) sieć z ustalonym znakowaniem początkowym.

■

Często spotykany wariant definicji sieci pomija funkcję W (jej domyślne wartości wynoszą 1) lub nadaje ograniczenie na maksymalną liczbę żetonów dopuszczalnych w danym miejscu.

Wykonanie (*odpalenie*) tranzycji jest zmianą stanu sieci (znakowania) polegającą na usunięciu żetonów z miejsc wejściowych i wprowadzenie nowych żetonów do miejsc wyjściowych. Liczba usuwanych i dodawanych żetonów określona jest przez wagi łuków. Dana tranzycja t usuwa $W(p, t)$ żetonów z miejsc wejściowych i dodaje $W(t, p)$ żetonów do miejsc wyjściowych.

Odpalenie tranzycji nie może prowadzić do znakowań ujemnych. Stąd tranzycja t jest dopuszczalna, jeśli liczba żetonów w miejscu wejściowym p spełnia $M(p) \geq W(t, p)$.

Zbiór dopuszczalnych tranzycji dla danego znakowania może zawierać więcej niż jeden element. W ten sposób sieć Petriego może opisywać niedeterminizm modelowanego systemu.

2.1.2 Własności sieci

Większość dynamicznych własności sieci definiuje się poprzez własności znakowań osiągalnych ze znakowania początkowego M_0 w wyniku wykonania ciągów tranzycji $\rho = \langle t_1, t_2,\ldots, t_i, \ldots\rangle$.

Tutaj przytoczone zostaną własności najbardziej typowe:

- osiągalność znakowania,
- ograniczoność i bezpieczeństwo,
- żywotność.

Szerszy przegląd i literaturę odnoszącą się do poszczególnych własności znaleźć można w [Murata 89].

Osiągalność

Znakowanie M_n jest osiągalne z M_0, jeżeli istnieje ciąg tranzycji $\rho = \langle t_1, t_2,\ldots, t_n\rangle$, który przeprowadza znakowanie M_0 w M_n. Zbiór wszystkich znakowań osiągalnych z M_0 oznaczany jest jako $R(N , M_0)$. Zbiór wszystkich ciągów tranzycji dopuszczalnych oznaczany jest jako $L(N , M_0)$

Problem czy $M_n \in R(N , M_0)$ jest rozstrzygalny. Problem równości sieci, czyli czy $L(N , M_0) = L(N' , M_0')$ jest w ogólnym przypadku nierozstrzygalny.

Ograniczoność i bezpieczeństwo

Sieć nazywana jest ograniczoną, jeżeli

$$\exists\, k \in \mathrm{N} \,.\forall\, M \in R(N\,,\, M_0)\,.\forall\, p\,.\, M(p) \le k\,.$$

Sieć nazywana jest bezpieczną, jeżeli $k = 1$.

Miejsca w sieci Petriego często opisują bufory składujące obliczane dane, pośrednie magazyny wyrobów dla procesu produkcyjnego, itp. W rzeczywistych systemach są one zawsze ograniczone. Własność ograniczoności lub bezpieczeństwa gwarantuje, że nie dojdzie do niekorzystnej sytuacji, jak przekroczenie rozmiarów bufora.

Żywotność

Żywotność opisuje brak blokowania w systemie. Sieć jest żywa, jeśli dla każdego znakowania ze zbioru $R(N\,,\, M_0)$ istnieje dopuszczalna tranzycja.

Dla systemów współbieżnych żywotność jest często rozważana jako własność tranzycji. Tranzycja t jest martwa, jeśli nie może nigdy zostać odpalona, tranzycja t jest *potencjalnie wykonywalna*, jeśli istnieje ciąg $\rho \in L(N\,,\, M_0)$ zawierający tę tranzycję.

Tranzycja jest żywa, jeśli jest potencjalnie wykonywalna dla każdego znakowania osiągalnego z M_0.

2.1.3 Reprezentacja macierzowa sieci Petriego

Wiele własności sieci Petriego bada się z wykorzystaniem ich macierzowej reprezentacji [Murata 89]. Macierzowa reprezentacja pozwala na przejrzyste sformułowanie warunków osiągalności oraz na dowodzenie własności strukturalnych opisanych przez niezmienniki dotyczące miejsc i przejść [Reisg 85].

Jak każdy graf, sieć Petriego może zostać opisana za pomocą macierzy incydencji [Deo 74]. Ze względu na strukturę połączeń macierz incydencji A sieci nie jest macierzą kwadratową lecz macierzą o rozmiarach $|\,P\,| \times |\,T\,|$. Każdy z elementów macierzy $A(p\,,\, t)$ opisuje zmianę liczby żetonów w miejscu p w wyniku wykonania tranzycji t. Stąd element $A(p\,,\, t)$ może być obliczony na podstawie funkcji W wag łuków jako

$$A(p\,,\, t) = A(p\,,\, t)^{+} - A(p\,,\, t)^{-}, \text{ gdzie}$$

$$A(p\,,\, t)^{+} = \begin{cases} W(p,t) \text{ jeżeli } (p,t) \in F \\ 0 \text{ w.p.p} \end{cases}$$

$$A(p\,,\, t)^{-} = \begin{cases} W(t,p) \text{ jeżeli } (t,p) \in F \\ 0 \text{ w.p.p} \end{cases}$$

Ciąg tranzycji ρ_n prowadzący ze znakowania M_0 do M_n zapisany jest w modelu liniowym jako wektor u_n, którego i–ta składowa odpowiada liczbie wykonanych tranzycji t_i w ciągu ρ.

Stąd otrzymujemy równanie stanu sieci Petriego postaci:

$$M_n = M_0 + A\, u_n\,.$$

Rozważane w dalszych częściach pracy modele specyfikacji są formułowane wyłączanie jako modele liniowoalgebraiczne. Jak będzie pokazane, są one sprowadzalne do postaci macierzowej sieci Petriego

2.1.4 Grafy znakowań osiągalnych i grafy pokrycia

Podstawowe własności dynamiczne sieci mogą być badane przez konstrukcję grafów znakowań osiągalnych lub grafów pokrycia. Graf znakowań osiągalnych $\Gamma = (H, P)$ jest grafem skierowanym, którego wierzchołkom ze zbioru H przypisane są osiągalne znakowania, natomiast łukom ze zbioru P tranzycje. Każdy wierzchołek, któremu odpowiada znakowanie M połączony jest k łukami z następnymi znakowaniami osiągalnymi, gdzie k jest liczbą tranzycji dopuszczalnych dla znakowania M.

W ogólnym przypadku grafy znakowań osiągalnych są nieskończone. Rozważmy znakowanie M i ciąg tranzycji ρ przeprowadzający M w znakowanie osiągalne M' spełniające $M' \geq M$. Oznaczmy $\Delta M = M' - M$. Wykonanie ciągu ρ dla znakowania M' spowoduje osiągnięcie znakowania $M'' = M' + \Delta M$. Jeżeli istnieje miejsce p, dla którego $\Delta M(p) > 0$, wówczas w wyniku wykonania kolejnych ciągów tranzycji ρ liczba żetonów w miejscu p będzie rosła nieograniczenie. Za każdym razem wszystkie tranzycje ciągu ρ będą dopuszczalne.

Graf pokrycia jest skończonym grafem reprezentującym zbiór znakowań osiągalnych. Jego wierzchołkom przypisane są wektory znakowań, których elementy przyjmują wartości ze zbioru $N \cup \{0\} \cup \{\omega\}$, gdzie ω jest symbolem nieograniczonej wartości.

Rozszerzona algebra liczb całkowitych dla symbolu ω jest następująca:

$$\omega + n = \omega,\ \omega - n = \omega,\ n < \omega,\ \text{gdzie } n \in C.$$

W [Murata 89] można znaleźć szczegółowy algorytm tworzenia grafu pokrycia. Dowód jego skończoności zamieszczony jest w [Reisg 85].

Dowolne znakowanie przypisane węzłom grafu pokrycia jest albo znakowaniem osiągalnym, jeśli nie zawiera symbolu ω, albo *pokrywa* znakowanie osiągalne, czyli na pozycjach oznaczonych symbolem ω mogą pojawiać się dowolnie duże wartości.

Konstrukcja grafu pokrycia dla danej sieci może być niejednoznaczna. Dwóm różnym sieciom o różnej strukturze i różnym zachowaniu może być przypisany ten sam graf pokrycia [Reisg 85].

Konstrukcja grafu pokrycia pozwala na rozstrzyganie o skończoności zbioru znakowań osiągalnych, własnościach bezpieczeństwa (ograniczoności sieci) oraz częściowo o własnościach opisujących żywotność.

2.2 Komunikujące się maszyny skończenie-stanowe

W podrozdziale zostanie omówiony model komunikujących się maszyn skończenie-stanowych. Opisana w dalszych częściach pracy specyfikacja będąca przedmiotem weryfikacji poprawności jest najczęściej reprezentacją zbioru synchronicznie komunikujących się maszyn.

Maszyny skończenie-stanowe stanowią jeden z najstarszych i najbardziej rozpowszechnionych modeli. W wielu metodach specyfikacji wymagań i projektowania oprogramowania są one używane jako jeden z elementów opisu dynamicznych własności systemów. Często są one uzupełnieniem dla metod skupiających się na opisach związków strukturalnych pomiędzy przetwarzanymi danymi i realizujących to przetwarzanie elementach specyfikacji.

Typowym przykładem są metody analizy strukturalnej [WM 85; Yourdon 88; Perez 90], gdzie za pomocą maszyn skończenie stanowych opisywane są dynamiczne własności

procesów (tzw. procesy sterujące). Podobnie w metodach obiektowych [CY 1990, Booch 94] maszyny skończenie stanowe opisują diagramy stanów obiektów. Rozszerzona definicja maszyn skończenie stanowych w ujęciu Harela [Harel 88] jest elementem metody UML [FS 97].

Rozmaitość zastosowań pociąga za sobą różnorodność definicji. Ich wspólną cechą jest to, że system scharakteryzowany jest przez skończony zbiór stanów. Przejście pomiędzy stanami jest zazwyczaj możliwe w wyniku pojawienia się zdarzenia wejściowego (otrzymania bodźca). Równocześnie przy przejściu może być generowane zdarzenie wyjściowe.

Typowa definicja maszyny skończenie stanowej [BH 93] jest następująca:

$FSM = (S, s_0, I, O, F_S, F_O)$, gdzie:

- S jest skończonym zbiorem stanów;
- $s_0 \in S$ jest stanem początkowym;
- I jest skończonym zbiorem zdarzeń wejściowych;
- O jest skończonym zbiorem zdarzeń wyjściowych;
- $F_S : S \times I \rightarrow S$ jest funkcją określającą następny stan na podstawie aktualnego stanu i oraz zdarzenia wejściowego;
- $F_O : S \times I \rightarrow O \cup \{\varepsilon\}$ jest funkcją określającą generowane zdarzenie wyjściowe przy przejściu pomiędzy stanami; przez ε oznaczane jest zdarzenie puste.

Zdefiniowana w powyższy sposób maszyna zawsze zmienia stan pod wpływem bodźca (zdarzenia wejściowego) przechodząc w jednoznacznie zdefiniowany stan następny.

Z punktu widzenia wielu zastosowań tego typu definicja jest zbyt słaba, ponieważ nie pozwala na modelowanie systemów dokonujących przejść spontanicznych i nie pozwala na modelowanie niedeterminizmu. Alternatywną postacią jest definicja oparta na relacji przejścia:

$FSM = (S, s_0, I, O, T)$, gdzie

$T \subset S \times (I \cup \{\varepsilon\}) \times S \times (O \cup \{\varepsilon\})$

Maszyna zdefiniowana relacyjnie może wykonywać przejścia spontaniczne, jeśli do dziedziny relacji T należy para (s_i, ε) oraz może modelować niedeterminizm, jeżeli $| T((s_i, in_j)) | > 1$.

Element relacji nazywany jest często regułą przejścia. Zdarzenia wejściowe i wyjściowe są zdefiniowane abstrakcyjnie. W zależności od modelu, zdarzenie wejściowe może być interpretowane jako spełnienie pewnego warunku logicznego, natomiast zdarzenie wyjściowe jako wykonanie pewnej operacji (rozkazu). Stąd element relacji przejścia może być traktowany jako rozkaz dozorowany, a wykonanie maszyny podobne jest działania niedetrministycznej konstrukcji **do-od** [Dijkstra 76].

Specyfikacje systemów najczęściej nie są formułowane w postaci izolowanej maszyny skończenie stanowej, lecz jako zbiór maszyn sprzężonych wejściami i wyjściami. Stanowią one zbiór komunikujących się maszyn.

Zazwyczaj rozważane są dwa zasadniczo odmienne modele komunikacji pomiędzy maszynami:

- sprzężenie synchroniczne,

- sprzężenie asynchroniczne.

2.2.1 Synchroniczny model komunikacji

W synchronicznym modelu komunikacji zdarzenia wyjściowe maszyny pełniącej rolę nadawcy odwzorowywane są bezpośrednio na zdarzenia wejściowe odbiorcy. Dwie komunikujące się maszyny mogą wykonywać przejścia niezależnie od siebie z wyjątkiem przejść, kiedy bezpośrednio się komunikują.

Rozważmy dwie maszyny $FSM_1 = (S_1, s_{01}, I_1, O_1, T_1)$ i $FSM_2 = (S_2, s_{02}, I_2, O_2, T_2)$. Załóżmy, że istnieje relacja $\phi_{12} \subset O_1 \times I_2$ opisująca komunikację pomiędzy maszynami.

Maszyna FSM_1 pełniąca rolę nadawcy może wykonać przejście $((s_{1i}, \bullet),(s_{1j}, out)) \in T_1$ tylko wtedy, kiedy maszyna FSM_2 pełniąca nadawcy znajduje się w stanie s_{2k} takim, że $((s_{1i}, in),(s_{1j},\bullet)) \in T_2$ oraz spełnione jest $(out, in) \in \phi_{12}$.

Model komunikacji podobny jest więc do mechanizmu spotkań w języku Ada [Barnes 84]. Mimo, że nie odnoszący się jawnie do maszyn skończenie stanowych, powyższy model komunikacji używany jest w CCS [Milner 89] i CSP [Hoare 85]. Elementami relacji ϕ_{12} są pary akcji, którym przypisano etykiety i koetyekiety postaci $(\overline{com}, com)$ dla CCS lub (com!, com?) dla CSP. Zwłaszcza ten drugi sposób zapisu jest preferowany przy definiowaniu sygnałów wejściowych i wyjściowych dla maszyn skończenie stanowych. Spotkać go można między innymi w [Holzmann 91] oraz [Shaw 92, 94].

2.2.2 Asynchroniczny model komunikacji

W synchronicznym modelu komunikacji zdarzenia wyjściowe generowane przez nadawcę są natychmiast konsumowane przez maszynę pełniącą rolę odbiorcy. W przypadku komunikacji asynchronicznej mają one bardziej trwały charakter. Przesyłane zdarzenia zazwyczaj nazywane są *komunikatami*. Model zbioru komunikujących się maszyn zakłada, że są one połączone *kolejkami komunikatów* o skończonej lub nieskończonej pojemności działającymi w trybie FIFO. Kolejki komunikatów nazywane są także kanałami. Nadawca może wygenerować komunikat wyjściowy niezależnie od odbiorcy i umieścić go w jego kolejce wejściowej. Odbiorca odbiera komunikaty w takiej kolejności, w jakiej zostały umieszczone w kolejce.

Model asynchronicznej komunikacji jest chętnie używany w przypadku modelowania protokołów komputerowych. Typowym przykładem zastosowań jest język SDL [RS 82; SSR 89; BH 93], który definiuje system jako zbiór komunikujących się procesów opisanych przez maszyny skończenie stanowe. Podobnie modelowany jest system w przypadku języka Promela [Holzmann 91]. Z tej ostatniej pracy zaczerpnięte zostały poniższe definicje:

Def. 2-2 Kolejka komunikatów

Kolejka komunikatów MQ jest trójką postaci $MQ = (V, N, C)$, gdzie V – jest zbiorem komunikatów (słownikiem), N – maksymalną liczbą komunikatów w kolejce, C – zwartością kolejki, czyli co najwyżej N–elementowym ciągiem elementów z V.

■

Niech M będzie dowolnym zbiorem kolejek komunikatów. Zakładamy, że dla dwóch dowolnych kolejek ich zbiory komunikatów są rozłączne. Przez C_j^i oznaczany będzie j–ty element i–tej kolejki, przez $|C^i|$ liczbę elementów kolejki.

Zakładając, że kolejkom przydzielone są numery od 1 do $|M|$, alfabet całego systemu zdefiniowany jest jako: $V = \bigcup_{m=1}^{|M|} V^m \cup \{\varepsilon\}$, gdzie V^m oznacza słownik *m*-tej kolejki.

Def. 2-3 Maszyna skończenie stanowa

Maszyna skończenie stanowa *FSM* zdefiniowana jest jako *FSM* = (*S*, s_0, *M*, *A*, *T*), gdzie

- *S* jest skończonym zbiorem stanów;
- $s_0 \in S$ jest stanem początkowym
- *M* jest zbiorem kolejek komunikatów
- *A* jest zbiorem akcji obejmującym akcje wejściowe, wyjściowe i akcję wewnętrzną (ε);
- *T* jest relacją przejścia: $T \subset S \times (V \cup \{\varepsilon\}) \times S$

■

Argumentami relacji przejścia są pary postaci (*s* , *act*), gdzie *s* jest stanem, natomiast *act* jest akcją. Rozróżniamy trzy rodzaje akcji: wewnętrzne (oznaczane przez ε) wejściowe związaną z odebraniem komunikatu oraz wyjściowe związaną z wysłaniem komunikatu.

W przypadku, kiedy (*s* , ε) należy do dziedziny relacji *T* możliwe jest dokonanie spontanicznego przejścia.

Systemem komunikujących się maszyn nazywana jest para ($\{ FSM^i \}$, *M*), gdzie $\{ FSM^i \}$ jest zbiorem maszyn, *M* jest sumą wszystkich zbiorów kolejek komunikatów elementów składowych.

Algorytm wykonania systemu jest następujący

Załóżmy, że $|\{ FSM^i \}| = p$; niech s^i oznacza stan *i*–tej maszyny.

1. Dla $1 \le i \le p$ inicjuj: $s^i = s_0^i$ oraz dla $1 \le i \le |M|$ podstaw: $C^i = \langle\,\rangle$
2. Wybierz dowolną maszynę *i* oraz regułę przejścia T^i, taką że

   $T^i(s^i, a) \neq \varnothing$ i akcja *a* jest wykonalna
3. Jeśli brak takiej reguły → STOP
4. Wykonaj akcję *a* i przejdź do 2.

Dla danej akcji *a* niech *m*(*a*) oznacza przesyłany komunikat, *d*(*a*) kolejkę komunikatów, na której wykonywane są operacje.

Dana akcja *a* uznana jest za wykonalną w stanie s^i, jeżeli:

- jest akcją wewnętrzną, wówczas *a* = ε ;
- jest akcją wejściową i zachodzi *m*(*a*) = $C_1^{d(a)}$;
- jest akcją wyjściową i zachodzi $|C^{d(a)}| < N^{d(a)}$.

W wyniku wykonania akcji wejściowej pierwszy element ciągu komunikatów w kolejce jest usuwany, w wyniku wykonania akcji wyjściowej do ciągu dodawany jest komunikat.

Modele komunikacji synchronicznej i asynchronicznej są w pewnym stopniu sobie równoważne. Synchroniczna komunikacja może być traktowana w modelu asynchronicznym jako komunikacja asynchroniczna poprzez kanał o zerowej pojemności.

Z kolei w modelu synchronicznym kanał może być modelowany jako odrębna maszyna skończenie stanowa. Tego typu przykłady można odnaleźć w [Shaw 92].

Opisany powyżej model komunikacji asynchronicznej zakłada, że kolejki komunikatów nie są maszynami skończenie stanowymi lecz odrębnymi elementami modelu. Umożliwia to przeprowadzenie kompozycji maszyn, czyli połączenia dwóch lub większej liczby maszyn w maszynę bardziej złożoną. Podczas złożenia stany kolejek wewnętrznych uczestniczących wyłącznie w komunikacji pomiędzy maszynami składowymi nie rozszerzają wektora stanu maszyny złożonej.

Dla dwóch maszyn postaci $FSM = (S_1, s_{01}, M_1, A_1, T_1)$ i $FSM = (S_2, s_{02}, M_2, A_2, T_2)$ maszyna FSM_{12} otrzymana w wyniku ich kompozycji ma postać: $FSM_{12} = (S_{12}, s_{012}, M_{12}, A_{12}, T_{12})$, gdzie:

- $S_{12} = S_1 \times S_2$
- $s_{012} = (s_{01}, s_{02})$
- $M_{12} = M_1 \cup M_2$
- $A_{12} = A_1 \cup A_2$
- $((s_{11},s_{12}),a, (s_{11},s_{12})) \in T_{12} \Leftrightarrow (s_{11}, a, s_{12}) \in T_1 \vee (s_{21}, a, s_{22}) \in T_2$

Otrzymana maszyna złożona może zostać poddana redukcji. W [Holzmann 91] przytoczono za [KS 90] algorytm redukcji maszyn skończenie stanowych opierający się na znalezieniu relacji równoważności pomiędzy stanami.

Maszyna skończenie stanowa obserwowana z zewnątrz może być traktowana jako obiekt, który w odpowiedzi na ciąg zdarzeń wejściowych generuje ciąg zdarzeń wyjściowych. Zadaniem redukcji dla danej maszyny *FSM* jest stworzenie równoważnej maszyny FSM_r, która w odpowiedzi na dany ciąg wejściowy wygeneruje ten sam ciąg wyjściowy.

Redukcja odbywa się poprzez poszukiwanie relacji równoważności E pomiędzy stanami i zastąpienie minimalizowanej maszyny *FSM* maszyną zredukowaną FSM_r, której stany odpowiadają klasom równoważności. Generowane przez akcje przejścia dla zredukowanej maszyny zastępowane są przejściami pomiędzy stanami reprezentującymi klasy równoważności.

Dwa stany s_{11} i s_{12} uważane są za równoważne $(s_{11}, s_{12}) \in E$ jeżeli dla każdej akcji a dopuszczalnej w obu stanach jej wykonanie spowoduje przejście z $s_{11} \xrightarrow{a} s_{21}$ oraz $s_{12} \xrightarrow{a} s_{22}$ oraz $(s_{21}, s_{22}) \in E$. Relacja ta podobna jest więc do relacji mocnej bisymulacji CCS [Milner 89].

2.2.3 Rozszerzenia modelu

Podstawowy model maszyny skończenie-stanowej w wielu formalizmach podlega rozmaitym rozszerzeniom. Znanym rozszerzeniem jest model Harela [Harel 88] definiujący hierarchiczne maszyny skończenie-stanowe. Pozwala on na przeprowadzenie dekompozycji stanów występujących w opisie na wyższym poziomie abstrakcji przez zastąpienie ich maszynami zdefiniowanymi w kolejnej, bardziej szczegółowej warstwie specyfikacji.

Wiele rozszerzeń podstawowego modelu związanych jest z potrzebą zamodelowania procesów opisanych konkretnym językiem specyfikacji, który uwzględnia obecność zmiennych, a zwłaszcza zmiennych sterujących. Prowadzi to zazwyczaj do wyróżnienia dodatkowych wewnętrznych akcji, które dokonują operacji na zmiennych oraz

predykatów, które służą do testowania ich wartości. Najczęściej zakłada się, że zmienne mogą przyjmować skończony zbiór wartości, stąd mogą być również traktowane jako maszyny skończenie stanowe.

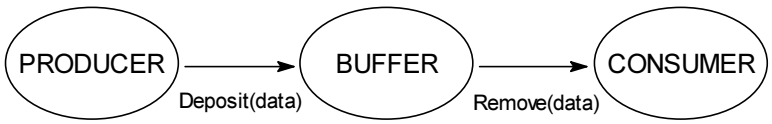
Równocześnie rozszerzeniu ulega opis komunikacji. Komunikacja nie jest traktowana wyłącznie jako działanie synchronizacyjne z użyciem abstrakcyjnych obiektów, lecz jako przekazywanie wartości należących do skończonych zbiorów połączone z przypisaniem ich zmiennym. Typową formą zapisu komunikacji jest postać *channel*!*value* dla nadawcy oraz *channel*?*variable* dla odbiorcy. Tego typu składnia używana jest w języku Promela [Holzmann 91]. Równoważną pod względem siły wyrazu graficzną składnię oferuje język SDL [SSR 89; BH 93].

Dodanie zmiennych o skończonych zbiorach wartości umożliwia przede wszystkim redukcję wielkości specyfikacji. Model maszyny rozszerzony o zmienne może zostać dalej przez kompozycję przekształcony do postaci jednej maszyny. Jego siła obliczeniowa jest taka sama, jak w przypadku podstawowej definicji maszyny. Zaletą jest zbliżenie do rzeczywistych języków programowania.

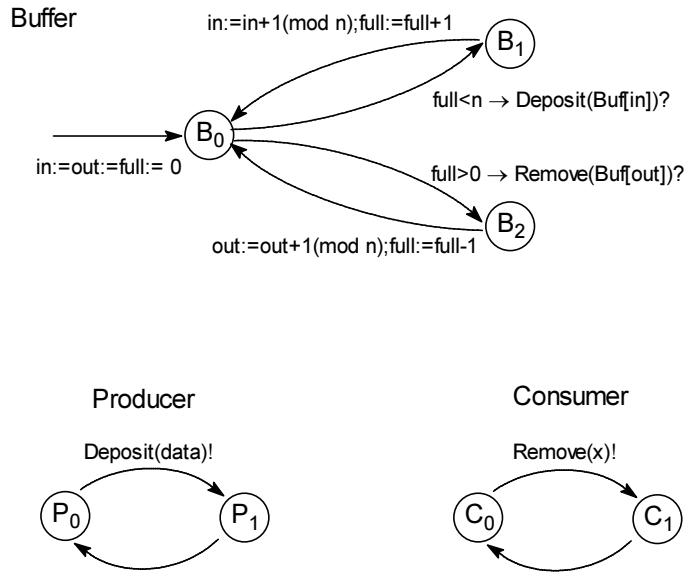
2.2.4 Model CRSM

Omówiony powyżej sposób modelowania operacji na zmiennych zastosowano w modelu CRSM (*Communicating Real-Time State Machines*) [Shaw 92, 94; RS 94]. Model CRSM stosowany jest przede wszystkim do opisu systemów czasu rzeczywistego, dlatego zakłada wyłącznie synchroniczną komunikację miedzy maszynami. Kolejki komunikatów (lub semafory) występujące w systemie są jawnie wyspecyfikowane przez odpowiednio zdefiniowane maszyny.

Przykład specyfikacji w postaci zbioru maszyn CRSM przedstawiony jest na Rys. 2-1 oraz Rys. 2-2. System składa się z trzech komponentów: producenta wysyłającego dane, bufora oraz konsumenta. Proces *Buffer* realizuje bufor cykliczny. Składuje on dane w *n*–elementowej tablicy *Buf*. Zmienne *in*, *out* służą do indeksowania bufora. Zmienna *full* przechowuje liczbę jego elementów.



Rys. 2-1 System trzech komunikujących się maszyn



Rys. 2-2 Specyfikacje maszyn składowych

Każdemu przejściu w modelu CRSM przypisany jest rozkaz dozorowany postaci: *dozór* → *sekwencja operacji*. Dozór jest predykatem logicznym biorącym za argumenty wartości zmiennych. Operacjami występującymi w ciągu mogą być komunikacje lub operacje wewnętrzne. Możliwe jest także przypisanie przejściu operacji pustej.

Aspekt czasu rzeczywistego odzwierciedlony jest w specyfikacjach CRSM poprzez możliwość przypisania przejściom przedziału czasu $[t_{min}, t_{max}]$, gdzie $t_{min}, t_{max} \geq 0$. Wartość t_{min} opisuje minimalny czas od momentu osiągnięcia stanu, po którym przejście może zostać wykonane; wartość t_{max} oznacza czas, po którego upływie przejście staje się zabronione.

Model CRSM umożliwia symulację wykonania systemu. Wykonanie odbywa się poprzez niedetrministyczny wybór przejścia, którego dozór przybiera wartość prawdy w danym stanie systemu. Konieczność uwzględnienia czasów przejść powoduje, że dodatkowym elementem wektora stanu stają się liczniki czasu przypisane każdej maszynie. Są one zerowane w momencie osiągnięcia stanu i inkrementowane wraz z upływem symulowanego czasu. Po przekroczeniu przez wartość licznika wartości t_{min} przypisanej przejściu staje się ono dopuszczalne, jeśli dozór przyjmuje wartość prawdy. Przekroczenie czasu t_{max} powoduje, że przejście staje się niedopuszczalne.

Model CRSM był używany jako pośrednia warstwa opisu systemu przy weryfikacji aplikacji opisanych językiem projektowania czasu rzeczywistego LACATRE [SS 94, 95a, 95b, 96; 97a, 97b; SSSS 94, 96].

Systemy modelowane za pomocą zbioru synchronicznie komunikujących się maszyn skończenie stanowych mogą być zazwyczaj opisane za pomocą sieci Petriego. Wydaje się jednak, że w praktycznie stosowanych językach specyfikacji przeważają modele maszyn skończenie stanowych. W odróżnieniu od sieci Petriego, maszyny skończenie stanowe pozwalają na rozdzielenie opisu sterowania od mechanizmów synchronizacji. Traktując modele procesów rozłącznie ułatwiają graficzną specyfikację systemu poprzez definicję poszczególnych komponentów. Diagram systemu w postaci sieci Petriego złożonej z kilkudziesięciu miejsc i tranzycji przewyższa na ogół możliwości percepcyjne człowieka.

Standardowe typy sieci Petriego nie pozwalają na modelowanie komunikacji asynchronicznej.

2.3 Algebry procesów

Ważne miejsce w opisie i analizie systemów współbieżnych odgrywają podejścia algebraiczne. Ich przykładem są algebry procesów takie, jak CCS [Milner 89], CSP [Hoare 85], ACP [BK 86]. Ukierunkowane są one na opis systemów złożonych z równolegle działających komunikujących się komponentów.

Podstawowymi pojęciami w podejściu algebraicznym są: *proces* rozumiany jako matematyczna abstrakcja interakcji systemu z otoczeniem oraz *akcja* (zdarzenie). Zbiór akcji, które proces może wykonać nazywany jest jego *alfabetem*.

Typowymi składnikami algebry procesów [LB-GG 94] są:

- zbiór operatorów i reguł syntaktycznych pozwalających konstruować procesy
- odwzorowanie, które nadaje semantykę wyrażeniom opisującym procesy
- zbiór reguł algebraicznych, które opisują syntaktyczne zasady przekształcania wyrażeń
- pojęcia opisujące równoważność lub częściowe uporządkowanie pomiędzy procesami.

Term w algebrze procesów reprezentuje proces, który jest przekształcany w inny proces w wyniku wykonania jednej lub kilku akcji. Typową cechą algebr procesów jest ich modularność pozwalająca traktować system, jako zbiór komponentów. Algebry procesów definiują zestaw operatorów, które są używane do konstruowania złożonych procesów z prostych procesów składowych.

Podstawowy zbiór operatorów jest bardzo podobny. Obejmują one:

- operator prefiksowy opisujący kolejność akcji,
- wybór pomiędzy alternatywnymi zachowaniami,
- złożenie równoległe,
- restrykcję pozwalającą na ukrywanie nazw akcji i abstrahowanie od szczegółów,
- operator zmiany etykiet
- rekursję dla opisu procesów o nieskończonym zachowaniu.

Dla porównania w poniższej tabeli zestawiono składnię operatorów CSP i CCS, do których następują odwołania w przykładach zamieszczonych w dalszych częściach pracy.

Operator	CSP	CCS	opis
prefiksowy	$a \rightarrow P$	$a.P$	Proces, który wykonuje akcję a; następnie zachowuje się jak proces P.
wybór	$P \mid Q$	$P + Q$	Proces, który zachowuje się jak proces P albo Q.
złożenie równoległe	$P \parallel Q$	$P \mid Q$	Złożenie procesów, wymagana jest synchronizacja akcji składających się na komunikację lub akcji o tych samych etykietach (CSP).
ukrywanie (restrykcja)	$P \setminus \{l_1,\ldots,l_n\}$	$P \setminus \{l_1,\ldots,l_n\}$	Proces, z którego definicji usunięto wszystkie etykiety ze zbioru $L = \{l_1,\ldots,l_n\}$ zastępując je etykietą

			pustą (lub specjalną etykietą τ w CCS).
zmiana etykiet	$f(P)$	$P[f]$	Proces, w którego definicji etykiety akcji zostały odwzorowane przez funkcję f w nowe symbole.
rekursja	$P = \mu X{:}A.F(X)$	$P = \text{fix}(X{=}F(X))$	Proces P jest najmniejszym procesem spełniającym równanie $X{=}F(X)$, gdzie $F(X)$ jest funkcją odwzorowującą proces w wyrażenie. A jest alfabetem.

Komunikacja jest podstawowym mechanizmem współdziałania równoległych procesów. Można wyróżnić komunikację dwustronną i wielostronną (synchronizację). Mechanizmy komunikacji dwustronnej pozwalają na synchronizację jedynie dwóch procesów wykonujących pary dopełniających się akcji $\langle a!, a?\rangle$ (CSP) lub $\langle \overline{a}, a\rangle$ (CCS). Proces wykonujący akcję oznaczoną $a!$ lub $\overline{a}$ pełni rolę aktywną (nadawcy). W CCS efektem wykonania wewnętrznej komunikacji $\langle \overline{a}, a\rangle$ w złożonym systemie jest przekształcenie jej w ukrytą akcję oznaczaną symbolem τ.

Mechanizm synchronizacji zdefiniowany jest w CSP. Złożony system opisany jako: $a \to P \parallel a \to Q \parallel a \to R$ jest zgodnie z regułami algebraicznymi przekształcany na: $a \to (P \parallel Q \parallel R)$.

Podczas specyfikacji złożonych systemów istotna jest możliwość reprezentowania ich na różnych poziomach abstrakcji. Jednym z głównych celów wprowadzenia metod formalnych, do których należą algebry procesów, jest dowodzenie poprawności specyfikacji. Zazwyczaj wymagania przyjmują postać bardziej abstrakcyjną, wolną od wielu szczegółów, które zostały wprowadzone w implementacji.

Abstrakcję w algebrach procesów umożliwiają operatory restrykcji i zmiany etykiet oraz reguły syntaktyczne opisujące łączenia procesów przy równoczesnym ukrywaniu wewnętrznych komunikacji. Proces $P \setminus \{a\}$ reprezentuje proces, z którego definicji usunięto akcję $\{a\}$. W wyniku konwersji stała się ona nieobserwowalna lub nierozróżnialna. W wyniku restrykcji akcja nie może być już używana do komunikacji (lub synchronizacji) z innymi procesami. W formalizmie CCS restrykcja dokonuje się także w sposób automatyczny dla komunikacji wewnętrznych pomiędzy komponentami złożonych systemów. Wynik restrykcji jest rozmaicie traktowany. W CSP, który opiera się przede wszystkim na ścieżkowym modelu semantycznym wynikiem restrykcji jest usunięcie akcji z opisu systemu. W CCS opierającym się na modelu etykietowanego systemu przejść rezultatem restrykcji akcji a jest jej konwersja na ukrytą akcję τ.

Operator zmiany etykiet pozwala na zmianę alfabetu procesu, przy równoczesnym zachowaniu jego struktury. Dzięki temu możliwe jest dowodzenie równości lub równoważności procesów.

2.3.1 CSP

Modele semantyczne

Zasadniczym modelem semantycznym dla formalizmu CSP jest model ścieżkowy [BHR 84; Hoare 85]. Ścieżką procesu o alfabecie A nazywany jest ciąg akcji ze zbioru A, który można otrzymać aktywując proces. Ciąg ten może być ciągiem pustym, ciągiem skończonym lub nieskończonym.

Model ścieżkowy *traces*(*P*) jest parą (*A*,*D*) gdzie *A* oznacza alfabet procesu, natomiast *D* jest zbiorem wszystkich ścieżek procesu.

Model ścieżkowy (*A*,*D*) spełnia:

1. $\langle\,\rangle \in D$

2. $\forall\, s,t\,.\, s\hat{}t \in D \Rightarrow s \in D$ (przez ^ oznacza się operator konkatenacji ścieżek)

Bardziej złożonym modelem pozwalającym na opis zarówno procesów deterministycznych jak i niedetrministycznych jest model niepowodzeń (ang. *failures model*) [BR 85; Hoare 85; OH 86].

Model niepowodzeń *failures*(*P*) zdefiniowany jest jako zbiór par postaci (*s*, *X*), gdzie $X \subset 2^{A}$. Zbiór *X* jest takim zbiorem zdarzeń oferowanych przez otoczenie procesu, że proces *P* po wykonaniu ścieżki *s* zostanie zablokowany w pierwszym kroku z powodu braku możliwości synchronizacji z jakimkolwiek zdarzeniem ze zbioru *X*.

Dywergencja

Model ten uzupełniany jest często o *dywergencję* [Hoare 85; BR 85]. Dywergencja jest niepożądanym zjawiskiem pojawiającym się w algebrze CSP w wyniku zastosowania operatora restrykcji do rekurencyjnie zdefiniowanych procesów.

Warunkiem istnienia unikalnych rozwiązań równań rekurencjnych jest dozorowana przez operator prefiksowy postać wyrażeń opisujących procesy. Równanie $X = a \rightarrow X$ posiada unikalne rozwiązanie oznaczane przez $\mu X{:}A.a \rightarrow X$, natomiast rozwiązaniem $X = X$ może być dowolny proces. Efektem zastosowania operatora restrykcji usuwającego akcje występujące w dozorach może być degeneracja równań rekurencyjnych:

$$(\mu X{:}\, A.a \rightarrow X) \setminus \{a\} = \mu X{:}\, (A \setminus \{a\}).X.$$

W algebrze CSP wprowadzono specjalne oznaczenie na proces $\mu X{:}\, A.X$ – jest on zdefiniowany jako $CHAOS_{\mathrm{A}}$. Proces $CHAOS_{\mathrm{A}}$ jest procesem zachowującym się w najbardziej nieprzewidywalny sposób. Jego ścieżką może być dowolny ciąg symboli ze zbioru *A* (alfabetu). Może on odmówić wykonania dowolnej akcji.

Dywergencją nazywana jest dowolna ścieżka procesu, po której wykonaniu proces zachowuje się chaotycznie: $divergences(P) = \{s \mid s \in traces(P) \wedge (P/s) = CHAOS\}$.

Dywergencja powstaje w wyniku usunięcia z definicji procesu akcji składających się na pętle. W formalizmie CCS [Milner 89] takie akcje zamieniane są na akcje ukryte τ (widoczne lecz nierozróżnialne). Obecność dywergencji nie jest tam traktowana jako element wymagający formalizacji algebraicznej, ale raczej jako zjawisko braku sprawiedliwości w działaniu systemu.

Inne operatory

Formalizm CSP definiuje szerszy niż podano powyżej zestaw operatorów. Między innymi obejmuje on:

- operator sekwencyjnego złożenia procesów *P*;*Q* – proces, który zachowuje się jak *P* do momentu zaobserwowania specjalnego zdarzenia ✓ (ang. *tick*) symbolizującego zakończenie działania, a następnie jak proces *Q*
- niedeterministycznego wyboru pomiędzy możliwymi implementacjami $P \sqcap Q$,,
- przeplotu $P \,|||\, Q$ (równoległe złożenie procesów bez synchronizacji)

- szereg operatorów stosujących się do ścieżek; np.: $s \uparrow A$ oznacza ścieżkę, z której usunięto symbole ze zbioru A.

Dla każdego z operatorów zdefiniowanych w algebrze podano obszerny zbiór reguł przekształcania wyrażeń.

Specyfikacja wymagań

Specyfikacja wymagań w CSP opiera się na pojęciu *obserwacji*. Obserwacja procesu jest to zdefiniowany w skończony sposób eksperyment, któremu proces może zostać poddany. [BHR 84; OH 86]. Specyfikacją nazywany jest zbiór obserwacji, które proces musi przejawiać. Proces P jest zgodny ze specyfikacją S, co jest zwykle zapisywane jako P **sat** S, jeżeli każda obserwacja procesu P jest dopuszczona przez specyfikację S. (Czyli S zawiera wszystkie możliwe obserwacje procesu.)

W przypadku *specyfikacji behawioralnej* jest ona zbiorem predykatów zdefiniowanych dla modelu semantycznego (modelu ścieżkowego lub modelu niepowodzeń).

Tak więc dla modelu ścieżkowego

$$P \textbf{ sat } S(t) \Leftrightarrow \forall\, t \in \mathit{traces}(P)\,.\,S(t),$$

natomiast dla modelu niepowodzeń:

$$P \textbf{ sat } S(t, X) \Leftrightarrow \forall\, (t, X) \in \mathit{failures}(P)\,.\,S(t, X).$$

Model ścieżkowy może być używany przede wszystkim do weryfikacji własności bezpieczeństwa. Nie pozwala on jednak na dowodzenie braku blokowania, ponieważ ścieżka pusta należy do zbioru ścieżek każdego procesu, w tym procesu blokującgo *STOP*.

Model niepowodzeń pozwala na dowodzenie własności żywotności. Typowym sposobem specyfikacji wymagań związanych z żywotnością jest:

dla $(s, X) \in \mathit{failures}(P)$ zachodzi $a \notin X$.

Algebra CSP definiuje szereg aksjomatów i reguł pozwalających na dowodzenie poprawności złożonego procesu na podstawie własności spełnianych przez jego komponenty.

2.3.2 CCS

Formalizm CCS [Milner 89], mimo podobieństwa składni, różni się od CSP modelem semantycznym, co ma wpływ między innymi na zestaw reguł opisujących zasady przekształcania wyrażeń.

Zasadniczym modelem dla CCS jest etykietowany system przejść (ang. *labelled transition system*) zdefiniowany jako: $(S, T, \{ \overset{t}{\rightarrow} . t \in T \})$, gdzie S oznacza zbiór stanów, T jest zbiorem etykiet, natomiast $\overset{t}{\rightarrow} \subset S \times S$ jest relacją przejścia zdefiniowaną dla każdej etykiety.

Zbiorem stanów S jest dla algebry CCS zbiór wyrażeń E opisujących procesy, zbiorem etykiet – suma alfabetów procesów Act, natomiast relacje $\overset{a}{\rightarrow}$ dla $a \in Act$ opisane są regułami przekształcania wyrażeń.

Na przykład:

$$\frac{}{a.E \overset{a}{\rightarrow} E} \equiv (a.E, E) \in \overset{a}{\rightarrow}$$

$$\frac{E \xrightarrow{a} E'}{E|F \xrightarrow{a} E'|F} \equiv (E,E') \in \xrightarrow{a} \Rightarrow (E|F, E'|F) \in \xrightarrow{a}$$

Inne rozważane modele semantyczne obejmują systemy przejść oparte na zmodyfikowanych definicjach relacji pomiędzy wyrażeniami. Przykładem tego rozważany jest system $(S, L^*, \{ \overset{s}{\Rightarrow} . t \in T \})$, gdzie L^* jest zbiorem wszystkich ciągów symboli należącego do zbioru L, natomiast $(P,Q) \in \overset{s}{\Rightarrow}$ jeśli z P można osiągnąć Q w wyniku wykonania ciągu symboli $s \in L^*$.

Relacje równoważności

Podstawowym zastosowaniem formalizmu CCS jest dowodzenie równoważności systemów. W praktyce wymagania poprawnościowe definiowane są jako istnienie relacji równoważności pomiędzy stanami dwóch systemów (czyli wyrażeniami opisującymi procesy). W szczególności, systemy te mogą reprezentować specyfikacje różniące się poziomem abstrakcji. Definicja jednego z nich może być bardziej złożona – obejmować kilka komponentów i dodatkowe akcje. Na potrzeby dowodu dodatkowe akcje i komunikacje pomiędzy komponentami są ukrywane.

Podstawowe relacje to:

- równoważność ~ (największa mocna bisymulacja)
- równoważność obserwacyjna ≈ (największa słaba bisymulacja)
- równość = (obserwacyjna kongruencja)

Większość definicji relacji równoważności wykorzystuje rozmaicie zdefiniowane relacje bisymulacji.

Relacja S pomiędzy procesami nazywana jest mocną bisymulacją, jeśli $(P, Q) \in S$ implikuje, że dla każdego $a \in Act$

$$1. \forall P \xrightarrow{a} P' . \exists Q' . (Q \xrightarrow{a} Q' \wedge (P', Q') \in S)$$

$$2. \forall Q \xrightarrow{a} Q' . \exists P' . (P \xrightarrow{a} P' \wedge (P', Q') \in S)$$

Równoważność systemów (jako suma wszystkich mocnych bisymulacji) jest wymaganiem, by dla równych stanów drzewa akcji (w tym ukrytych) były równe.

Słaba bisymulacja i równoważność obserwacyjna pomija ukryte (nieobserwowalne) akcje. Dwa procesy P i Q są uważane za obserwacyjnie równoważne, jeżeli każde obserwowane zachowanie procesu P może być zaobserwowane dla Q (i na odwrót.)

Relacja obserwacyjnej kongruencji (równości) wymaga uzgodnienia nieobserwowalnych akcji, ale ignoruje ich liczbę. Jest to stosunkowo mocna relacja, ponieważ równe procesy muszą być podobne zarówno pod względem obserwowanego zachowania, jak i struktury ukrytych akcji.

Alternatywną metodą opisu wymagań poprawnościowych dotyczących obserwowanego zachowania procesów jest w formalizmie CCS logika, w której wprowadzono dodatkowe operatory modalne: możliwości $\langle a \rangle$ i konieczności $[a]$.

Specyfikacja postaci $P \models \langle a \rangle F$ mówi, że proces P może wykonać akcję a i osiągnąć stan, w którym spełniona jest formuła logiczna F. Specyfikacja $P \models [a]F$ jest równoważna

$P \models \neg \langle a \rangle \neg F$. Jak pokazano w [Milner 89], dwa procesy są obserwacyjnie równoważne wtedy i tylko wtedy, gdy spełniają te same specyfikacje:

$$P \approx Q \text{ wtw. } \forall F . (P \models F \Leftrightarrow Q \models F)$$

Podstawą wszystkich relacji równoważności zdefiniowanych w CCS są rozmaicie definiowane relacje bisymulacji. Badanie relacji równoważności i rządzących nimi praw dla różnych modeli równoległości było przedmiotem wielu prac. Obszerne ich zestawienie znaleźć można między innymi w [GG 98].

2.4 Logika temporalna

2.4.1 Opis ogólny

Logika temporalna [Pnueli 77; MP 81; MP 91; Lamport 94; Klimek 98, 99] jest często używanym narzędziem specyfikacji wymagań poprawnościowych dla systemów współbieżnych.

Formuły logiki temporalnej zbudowane są z predykatów biorących za argumenty stany systemu połączonych operatorami logicznymi oraz operatorami modalnymi. Stan systemu jest interpretowany jako przypisanie poszczególnym procesom etykiet instrukcji i nadanie zmiennym wartości. Określone dla stanów predykaty są wyrażeniami zbudowanymi z etykiet instrukcji, zmiennych i stałych. Znaczenie predykatu $[\ \Psi\ ](s)$ jest funkcją, która przypisuje mu zależną od stanu s wartość prawdy lub fałszu.

Modelem dla logiki temporalnej (liniowej) jest obliczenie, czyli nieskończony ciąg kolejnych stanów systemu $\sigma = (s_0, s_1,\ldots)$ reprezentujących jego wykonanie.

Podstawowe operatory modalne to: $\Box$ „zawsze” (ang. *always*) oraz $\Diamond$ „w końcu” (ang. *eventually*).

Znaczenie tych operatorów określone jest jako:

$$[\Box\ \Psi]_\sigma \equiv \forall\ i\ .\ [\ \Psi\ ](s_i) = \mathit{true}$$

$$[\Diamond\ \Psi]_\sigma \equiv \exists\ i\ .\ [\ \Psi\ ](s_i) = \mathit{true}$$

Dalszymi, często wprowadzanymi operatorami modalnymi są: $\bigcirc$ „w następnym stanie” (ang. *next-state*) oraz U „dopóki” (ang. *until*). Wyrażenie $\bigcirc\ \Psi$ jest asercją, że formuła Ψ będzie prawdziwa w następnym stanie; wyrażenie $\Psi_1\ U\ \Psi_2$ oznacza, że istnieje stan s_k, w którym formuła Ψ_2 będzie spełniona i dla $i < k$ spełniona jest formuła Ψ_1.

Pragnąc zastosować logikę temporalną do określenia własności programu wyspecyfikowanego z użyciem abstrakcyjnego języka programowania buduje się model semantyczny programu P, w postaci zbioru nieskończonych ciągów jego stanów (ciągi skończone mogą być rozszerzone przez powtórzenie ostatniego stanu). Wyrażenie $P \models \Psi$ oznacza, że formuła Ψ przyjmuje wartość prawdy dla każdego ciągu $\sigma = \langle s_0, s_1,\ldots\rangle$ reprezentującego wykonanie programu P.

Typowym zabiegiem jest więc przypisanie semantyki do programu P przez przekształcenie go do postaci struktury relacyjnej określonej przez zbiór stanów i relację przejścia pomiędzy nimi. Podobne podejście można zaobserwować w [KMP 94] oraz w pracach poświeconych weryfikacji systemów wyspecyfikowanych z użyciem języka Promela [Holzmann 97; GH 93; Schoot 97; VT 98].

2.4.2 Niezmienniczość dla powtórzeń

Logika temporalna w ostatnich latach była przedmiotem intensywnych badań. Istnieje wiele jej odmian różniących się min. modelami, zestawem operatorów oraz regułami wnioskowania. Najbardziej istotne różnice dotyczą modelu semantycznego (logika liniowa i rozgałęziona – ang. *linear time*, *branching time*) oraz użycia operatora $\bigcirc$.

Częstym zastosowaniem logiki temporalnej jest dowodzenie, że abstrakcyjny program P_A jest implementowany przez konkretny program P_C. (Por. 2.5.1). Przejście od programu P_A do P_C może być rozpatrywane jako etap projektowania systemu (uściślenie specyfikacji).

Typowym przykładem jest takiego przejścia jest zastąpienie atomowej instrukcji wysokiego poziomu przez kilka rozkazów maszynowych. Otrzymane obliczenie σ_C systemu P_C (lub jego obraz za pośrednictwem pewnego przekształcenia) nie jest bezpośrednio obliczeniem programu P_A lecz wariacją zawierającą powtórzenia stanów. Powoduje to, że formuły zbudowane z użyciem operatora $\bigcirc$ (w następnym stanie), specyfikujące własności obliczeń σ_A systemu P_A mogą być nie spełnione dla obliczeń σ_C programu P_C.

Pożądaną własnością formuł opisujących poprawność jest *niezmieniczość dla powtórzeń* (ang. *invariance under stuttering*) [Lamport 83]. Formuła logiki temporalnej Ψ jest niezmiennicza dla powtórzeń, jeśli dla każdego ciągu stanów σ reprezentującego zachowanie systemu dodanie lub usunięcie skończonej liczby powtórzeń dowolnego stanu nie zmieni jej wartości.

2.4.3 Własności opisujące poprawność systemów współbieżnych

Logika temporalna pozwala na wygodny zapis własności opisujących poprawność systemów współbieżnych. Ich lista jest bardzo obszerna; szczegółowe definicje poszczególnych własności znaleźć można w [MP 81; Szmuc 89b; HS 92; Klimek 98].

Własność bezpieczeństwa (wyrażona nieformalnie jako „nic złego nie może się zdarzyć") może być zapisana jako $P \models \Box \neg w'$, gdzie w' jest predykatem opisującym niepożądaną systuację. Po podstawieniu $w = \neg w'$ otrzymamy postać $P \models \Box w$ opisującą wymaganie, by predykat w był niezmiennikiem wszystkich obliczeń programu P.

Typowe przykłady własności bezpieczeństwa to:

- częściowa poprawność (ang. *partial correctness*),
- wzajemne wykluczanie (ang. *mutual exclusion*),
- brak blokowania (ang. *deadlock freedom*).

Własność żywotności wyrażona jest nieformalnie jako żądanie, by po spełnieniu pewnego warunku w wyniku wykonania skończonej liczby przejść nastąpiło pożądane zdarzenie, jak zakończone sukcesem zatrzymanie programu, uzyskanie dostępu do zasobu lub aktywacja procesu składowego.

Własność żywotności wyrażana jest z użyciem logiki temporalnej jako $P \models \Box (p \Rightarrow \Diamond q)$, gdzie p jest warunkiem, natomiast q jest zdaniem prawdziwym dla stanu, w którym zaszło pożądane zdarzenie.

Typowymi przykładami własności żywotności są:

- całkowita poprawność (ang. *total correctness*)
- dostępność (ang. *accessibility*)

- brak g³odowania (ang. *freedom of individual starvation*).

Jak można zauważyć, podstawowe własności charakteryzujące poprawność opisane są bez użycia operatora $\bigcirc$.

2.4.4 Weryfikacja własności opisywanych logiką temporalną

Specyfikacja wyrażona z użyciem logiki temporalnej jest sformułowana jako lista własności, które powinien spełniać poprawnie zbudowany program. Formalnie, każda z tych własności jest wyrażeniem logiki temporalnej, a specyfikacja jest koniunkcją tych wyrażeń.

Zastosowania logiki temporalnej w cyklu życia oprogramowania współbieżnego obejmują metody aksjomatyczne rozwijania specyfikacji oraz zagadnienia weryfikacji [Lamport 94].

Rozwój oprogramowania w opierający się na metodach aksjomatycznych może być traktowany jako ciąg przekształceń specyfikacji $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n$. Specyfikacja S_i implementuje specyfikację S_{i-1} jeśli własności opisane przez S_i pociągają za sobą własności opisane przez S_{i-1}, co może być stwierdzone przez dowodzenie implikacji $S_i \Rightarrow S_{i-1}$. Zaletą tego podejścia jest użycie tego samego języka na wszystkich poziomach specyfikacji. Przykłady zastosowania powyższej metody można znaleźć w [Klimek 98].

Drugim typowym podejściem jest weryfikacja własności programu opisanego pewnym językiem [MP 91; KMP 94]. Dowody poprawności mogą być wykonywane ręcznie lub z użyciem narzędzi automatycznej weryfikacji.

Użyteczność metod aksjomatycznych i manualnych dowodów wydaje się mocno ograniczona ze względu na złożoność rzeczywistych specyfikacji. Większą rolę praktyczną należy raczej przypisać metodom automatycznej weryfikacji (walidacji), dlatego też krótko zostaną omówione ich podstawy teoretyczne.

Przedstawiony tu materiał pochodzi z prac omawiających narzędzie automatycznej weryfikacji SPIN [GH 93; Holzmann 97; VB 98]. Analizowany program zapisany jest z użyciem języka Promela i przekształcany do postaci tzw. *struktury Kripke*. Wymagane własności oprogramowania zapisane są z użyciem liniowej logiki temporalnej nie zawierającej operatora 7.

Struktura Kripke zdefiniowana jest jako $K = (Props, S, T, s_0, L)$, gdzie *Props* jest zbiorem atomowych zdań, S jest zbiorem stanów, $T \subset S \times S$ jest relacją przejścia, s_0 jest stanem początkowym, natomiast $L : S \rightarrow 2^{Props}$ jest funkcją, która przypisuje każdemu stanowi s zbiór zdań prawdziwych w tym stanie.

Obliczeniem w strukturze Kripke nazywany jest ciąg stanów $\sigma = \langle s_0, s_1, \ldots \rangle$ spełniających $(s_{i-1}, s_i) \in T$. Odpowiadającą mu ścieżką nazywany jest ciąg akcji (przejść) $\rho = \langle \alpha_1, \alpha_2, \ldots \rangle$, gdzie $\alpha_i = (s_{i-1}, s_i) \in T$.

W pracach [VW 86, 94] pokazano, że dowolna formuła liniowej logiki temporalnej będąca kombinacją własności bezpieczeństwa i żywotności nie zawierająca operatora $\bigcirc$ (w następnym stanie) może zostać sformalizowana w postaci tzw. niedeterministycznego *automatu Büchi*.

Automat Büchi jest zdefiniowany jako $B = (S , \Sigma , \Delta , s_0 , F)$, gdzie S jest zbiorem stanów, Σ alfabetem, $\Delta \subset S \times \Sigma \times S$ jest relacją przejścia, $s_0 \in S$ stanem początkowym, natomiast $F \subset S$ jest zbiorem stanów akceptujących.

Dla danej formuły logiki temporalnej Ψ opisującej zachowanie programu przekształconego do postaci struktury Kripke $K = (Props, S, T, s_0, L)$ konstruowany jest automat Büchi B_Ψ,

którego przejścia są etykietowane zdaniami ze zbioru *Props*. Automat B_Ψ akceptuje nieskończone obliczenie $\sigma = \langle s_0, s_1, \ldots \rangle$, wtw. jeśli istnieje nieskończony ciąg stanów automatu π taki, że

1. etykieta *i*–tego przejścia w ciągu π odpowiada zdaniu prawdziwemu w stanie s_i ciągu σ
2. pewien stan ze zbioru stanów akceptujących *F* pojawia się nieskończenie często w π

Analogicznie, automat B_Ψ akceptuje ciąg przejść (akcji) struktury Kripke, jeśli akceptuje odpowiadający mu ciąg stanów.

Struktura Kripke *K* może być rozważana jako automat Büchi B_K, którego każdy stan jest stanem akceptującym (przekształcenie podano min. w [VB 98]) stąd zagadnienie poprawności może być rozpatrywane jako problem inkluzji języków dwóch automatów: $L(B_K) \subset L(B_\Psi)$.

W praktyce [VW 86; CVWY 92; Holzmann 91], zagadnienie to jest przeformułowywane do klasycznej postaci dowodu nie wprost. Sprawdzana formuła Ψ jest negowana i konstruowany jest automat formalizujący jej negację $B_{\neg\Psi}$. Oryginalne zagadnienie poprawności przekształcane jest do postaci problemu $L(B_K) \cap L(B_{\neg\Psi}) = \varnothing$. Zgodnie z argumentami w [VW 86; Holzmann 97] takie rozwiązanie jest bardziej efektywne ze względu na zużycie zasobów podczas weryfikacji.

Proces weryfikacji polega na konstrukcji *synchronicznego produktu* $K \times B_{\neg\Psi}$ struktury Kripke opisującej badany program i automatu reprezentującego negację formuły Ψ. Każde przejście automatu $K \times B_{\neg\Psi}$ jest postaci $(s_i, s_i^b) \xrightarrow{(\alpha,P)} (s_j, s_j^b)$, gdzie przejście $s_i \xrightarrow{\alpha} s_j$ jest przejściem w badanej strukturze Kripke, natomiast przejście $s_i^b \xrightarrow{P} s_j^b$ jest takim przejściem automatu $B_{\neg\Psi}$, że zdanie *P* przyjmuje wartość prawdy w stanie s_i.

Automat $K \times B_{\neg\Psi}$ akceptuje dokładnie te ciągi stanów $\sigma = \langle s_0, s_1, \ldots \rangle$ struktury *K*, które spełniają $\neg\Psi$, stąd formuła Ψ może być dowiedziona przez sprawdzenie, że język akceptowany przez automat $K \times B_{\neg\Psi}$ jest pusty. Istnienie obliczenia $\sigma = \langle s_0, s_1, \ldots \rangle$, któremu odpowiada ciąg stanów $\langle (s_0, s^b_0), (s_1, s^b_1), \ldots (s_f, s^b_f), \ldots \rangle$ akceptowany przez synchroniczny produkt $K \times B_{\neg\Psi}$ służy jako kontrprzykład dowodzący nie spełnienia formuły.

Specyfikacja pożądanych własności programu dana jest zazwyczaj w postaci koniunkcji formuł logiki temporalnej należących do pewnego zbioru $\{\Psi_i\}$. Na podstawie tego zbioru konstruowany jest zbiór automatów odpowiadających zanegowanym formułom składowym i każda z nich sprawdzana jest równolegle [Holzmann 97]. Narzędzie SPIN implementuje algorytm przeszukiwania w głąb przestrzeni stanów programu modelowanego za pomocą struktury Kripke *K* konstruując równocześnie produkty $K \times B_{\neg\Psi}$ dla poszczególnych formuł opisujących wymagane własności. Z myślą o praktycznych zastosowaniach stworzono efektywne algorytmy przeszukiwania częściowego i redukcji przeplotu.

SPIN jest typowym narzędziem *walidacji* czyli badania zgodności ze specyfikacją produktu końcowego. Specyfikacja i implementacja opisywane są innymi językami (logiki temporalnej i językiem Promela). Nie jest możliwe bezpośrednie zastosowanie zbadanego systemu jako weryfikowalnej specyfikacji dla produktu powstającego w kolejnej fazie projektowania.

2.5 Uściślanie specyfikacji

Jak wspomniano we wstępie, rozwój systemów współbieżnych często dokonywany przez stopniowe uściślanie specyfikacji (ang. *stepwise refinment*). Idea uściślania pierwszy raz postulowana była w [Wirth 71]. Zaproponowanym sposobem budowy oprogramowania jest przeprowadzenie szeregu kroków uściślających. W każdym kroku wybrane instrukcje programu są dekomponowane, czyli zastępowane instrukcjami bardziej szczegółowymi.

W ujęciu bardziej ogólnym, pod pojęciem uściślania rozumiana jest dowolna transformacja specyfikacji, która może zostać uznana za poprawną, ze względu na to, że przekształcona specyfikacja powiązana jest z oryginalną relacją implementacji: częściowego uporządkowania lub równoważności [BRR 90].

2.5.1 Relacje symulacji i uściślania

W pracy [KMP 94] przedstawiono koncepcję zastosowania logiki temporalnej do badania relacji symulacji i uściślania pomiędzy systemami.

Podstawowym elementem opisu systemu jest skończony zbiór zmiennych V (definiujących zarówno dane jak i stan sterowania systemu). Stanem systemu s jest interpretacja zmiennych ze zbioru V, czyli przypisanie zmiennym $u \in V$ wartości $s[u]$ należących do ich dziedziny. Zbiór stanów oznaczany jest przez Σ.

Jako model obliczeniowy przyjęto sprawiedliwy system przejść $S = (V, \Theta, T, J, C)$, gdzie V jest zbiorem zmiennych (słownikiem), Θ jest warunkiem początkowym, T zbiorem przejść (akcji), $J \subset T$ i $C \subset T$ są zbiorami specyfikującymi wymagania sprawiedliwości. Każde przejście $\tau \in T$ jest funkcją: $\Sigma \rightarrow 2^{\Sigma}$ odwzorowującą stan w zbiór jego następników.

Zbiory J i C służą do wykluczenia ze zbiorów obliczeń systemu obliczeń nie spełniających wymagań sprawiedliwości. Za niedopuszczalne uważane jest nieskończone obliczenie, w którym akcja $\tau \in J$ jest stale dopuszczalna, ale wykonana jedynie skończoną liczbę razy oraz podobnie takie obliczenie, w którym akcja $\tau \in C$ jest dopuszczalna w nieskończenie wielu stanach, ale wykonana jedynie skończoną liczbę razy.

Jak pokazano w pracy, zbiór wszystkich dopuszczalnych przejść systemu S może zostać wyrażony jako odpowiednio skonstruowana formuła liniowej logiki temporalnej $comp_S$. Dla model logiki temporalnej σ spełniona jest następującą zależność:

($\sigma \models comp_S$) wtw. (σ jest obliczeniem systemu S).

Symulacja

Najprostszą z relacji pomiędzy systemami jest relacja *symulacji*. Niech S^A oznacza bardziej abstrakcyjną wersję systemu (specyfikację), natomiast S^C wersję konkretną (implementację).

System S^C symuluje system S^A, co jest zapisywane jako $S^C \preceq S^A$, jeśli każde obliczenie S^C należy do zbioru obliczeń S^A.

Definicja ta pozwala na pomijanie dodatkowych zmiennych lokalnych wprowadzanych w systemie S^C (stan traktowany jest jako interpretacja wszystkich zmiennych należących do słownika V, a nie tylko zmiennych zdefiniowanych w systemie).

Warunkiem pozwalającym na weryfikację relacji symulacji w oparciu o logikę temporalną jest:

$$(S^C \preceq S^A) \text{ wtw. } (comp_{S^C} \Rightarrow comp_{S^A}) \text{ wtw. } (S^C \models comp_{S^A})$$

Uściślanie

Relacja uściślania zdefiniowana jest dla systemów, których opis został rozszerzony przez wyróżnienie w zbiorze zmiennych V podzbioru zmiennych lokalnych $L \subset V$. W przypadku specyfikacji zmienne lokalne traktowane są jako pomocnicze elementy opisu, które nie wymagają bezpośredniej implementacji. W przeciwieństwie do nich, zmienne ze zbioru $V \setminus L$ traktowane są jako jawny (obserwowalny) element wymagań. Rozszerzony system nazywany jest modułem.

Obserwacją dla modułu M nazywany jest taki ciąg stanów σ, dla którego istnieje obliczenie σ_c modułu M, różniące się od σ co najwyżej wartościami zmiennych lokalnych. W kategoriach logiki temporalnej zbiór wszystkich obserwacji zdefiniowany jest jako:

$$obs_{\mathrm{M}}: \exists L: comp_{\mathrm{M}}$$

Pojęcie obserwacji służy do zdefiniowania relacji uściślania i równoważności pomiędzy modułami.

Moduł M^C *uściśla* moduł M^A, co zapiszemy jako $M^C \trianglelefteq M^A$ jeśli każda obserwacja modułu M^C należy do zbioru obserwacji M^A. Moduły są *równoważne*, jeśli uściślają się wzajemnie.

Warunkiem opisującym relację uściślania w kategoriach logiki temporalnej jest:

$$(M^C \trianglelefteq M^A) \text{ wtw. } (obs_{M^C} \Rightarrow obs_{M^A}) \text{ wtw. } (comp_{M^C} \Rightarrow obs_{M^A}) \text{ wtw. } (M^C \models obs_{M^A})$$

Niech α będzie odwzorowaniem $E(V^C) \to L^{\mathrm{A}}$ (podstawieniem). Funkcja ta zastępuje zmienne lokalne modułu M^A przez wyrażenia na zmiennych ze zbioru V^C. Funkcja α indukuje odwzorowanie modułów $m_\alpha : \Sigma^{\mathrm{C}} \to \Sigma^{\mathrm{A}}$. W stanie $s^{\mathrm{A}} = m_\alpha(s^{\mathrm{C}})$ wartości zmiennych lokalnych zostały zastąpione przez obliczone wartości wyrażeń na zmiennych modułu M_{C}.

Odwzorowanie modułów m_α nazywane jest *odwzorowaniem uściślającym* jeżeli przekształca obliczenia modułu M^C w obliczenia modułu M^A.

Weryfikacja relacji uściślenia $M^C \trianglelefteq M^A$ polega więc na określeniu podstawienia α i udowodnieniu, że indukowane przez nie odwzorowanie modułów m_α jest odwzorowaniem uściślającym, co jest równoważne weryfikacji formuły logiki temporalnej:

$$comp_{M^C} \Rightarrow comp_{M^A}[\alpha].$$

2.5.2 Uściślanie akcji

Odmienne podejście do uściślania specyfikacji zaprezentowane zostało w przeglądowej pracy [GG 98]. Głównym przedmiotem jej zainteresowania są akcje, rozumiane jako element opisujący działanie systemu, który może być traktowany atomowo dla danego poziomu abstrakcji. Przy przejściu od opisów bardziej abstrakcyjnych do bardziej szczegółowych możliwe jest zastępowanie akcji procesami różnego typu (sekwencyjnymi, równoległymi, nieskończonymi).

Uściślanie akcji definiowane jest w postaci operatora, który biorąc za argumenty opis oryginalnego systemu oraz definicje procesów przypisanym akcjom składowym pozwala wywieść zachowanie nowego systemu z zachowania oryginalnego systemu oraz zachowania procesów podstawionych w miejsce akcji.

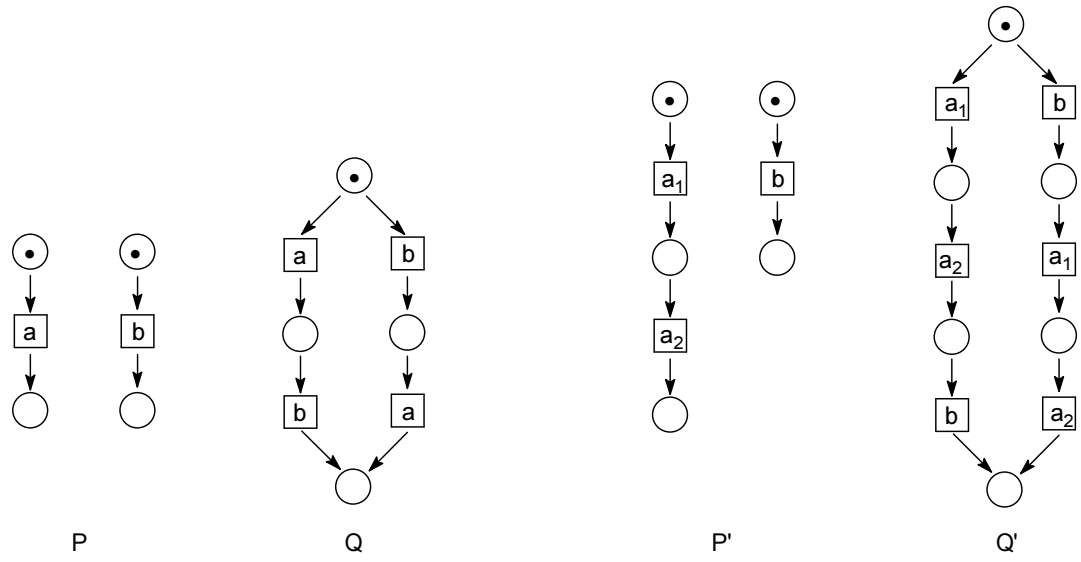
W [GG 98] pominięto pojęcie poprawności, lecz skupiono się na badaniu czy relacje równoważności systemów (obejmujące szerokie spektrum od równoważności ścieżek do rozmaicie zdefiniowanych relacji bisymulacji) są zachowywane przy uściślaniu. W tym celu rozważano grupę modeli nadających semantykę językowi bliskiemu CCS [Milner 89]

uwzględniającemu jednak rozróżnienie pomiędzy blokowaniem a poprawnym zatrzymaniem procesu (zdarzenie ✓).

Najbardziej interesująca z punktu widzenia podejścia stosowanego w tej rozprawie jest analiza uściślania dla modeli, dla których reprezentacja zachowania oparte jest na przeplocie. Jak pokazano, dla modeli z przeplotem nie są zachowane żadne relacje równoważności. Przykładem tego są systemy opisane jako $P = a \mid b$ oraz $Q = a;b + b;a$, których zachowanie jest nierozróżnialne – oba mogą wykonać sekwencje akcji $\langle a, b\rangle$ lub $\langle b, a\rangle$.

Po zastąpieniu akcji a przez proces postaci $(a_1;a_2)$ otrzymane specyfikacje $P' = (a_1;a_2) \mid b$ oraz $Q' = (a_1;a_2);b + b;(a_1;a_2)$ nie mogą zostać uznane za równoważne, ponieważ P' może wykonać ciąg akcji $\langle a_1, b, a_2\rangle$, który nie jest możliwy dla Q'. Wyklucza to wszelkie relacje oparte na równości zbiorów ścieżek czy też bisymulacje.

Rys. 2-3 pokazuje przykłady sieci Petriego reprezentujące systemy przed i po uściśleniu.



Rys. 2-3 Systemy przed uściśleniem (P,Q) i po zastąpieniu akcji a przez $a_1;a_2$ (P',Q')

Wskazane metody definicji operatora uściślania dla modeli opartych na przeplocie to ograniczenie akcji, które mogą zostać poddane uściślaniu lub ograniczenie klas rozważanych systemów.

W [GG 98] przebadano przede wszystkim uściślanie dla modeli opartych na etykietowanych zbiorach zdarzeń, pomiędzy którymi zachodzą relacje przyczynowości oraz konfliktu. Udowodniono między innymi, że zachowane są relacje równoważności oparte na równości rodziny wielozbiorów zdarzeń częściowo uporządkowanych przez relację przyczynowości (ang. *pomset*). Modele te są dość odległe od rozważanych w tej pracy, z tego powodu nie będą dokładniej omawiane.

Stosowany w pracy model zachowania oparty jest na przeplocie; uściślanie akcji przez zastępowanie ich procesami jest przedmiotem weryfikacji i może wystąpić jedynie w specyfikacji będącej implementacją. Nie jest dowodzona równoważność specyfikacji ale istnienie relacji implementacji. Przyjęte są także pewne ograniczenia dotyczące rozważanych modeli. (Por. rozdział 6 omawiający niehomomorficzną funkcję obserwacji.)

2.6 Algebraiczne metody analizy poprawności względnej

Analiza poprawności względnej oprogramowania współbieżnego była od kilku lat przedmiotem badań prowadzonych w Katedrze Automatyki AGH. [Szmuc 88, 89a, 89b, 91, 92, 93, 97a, 97b]. W trakcie prac teoria ulegała pewnym modyfikacjom i rozszerzeniom

Przedstawiony w niniejszym rozdziale materiał omawia zaawansowany stan badań, ale pomija wpływ pewnych później wprowadzonych zmian na postać definicji istotnych pojęć. (Ich obecność jest jednakże zaznaczona.) Taki wybór jest spowodowany chęcią lepszego wyjaśnienia własności ogólnych, które nieco zacierają się w definicjach uwzględniających wszystkie przypadki szczegółowe.

Charakterystyczną cechą modelu poprawności przyjętego w omawianej teorii jest jego względny charakter. Poprawność nie jest rozpatrywana, jako własność ogólna (taka jak bezpieczeństwo czy żywotność sieci Petriego) ale jako zgodność systemu z wymaganiami. Takie podejście jest zbieżne z definicją poprawności oprogramowania objętą standardem IEEE [IEEE 86].

Metody analizy poprawności względnej opisują poprawność dla modelu złożonego z trzech komponentów:

- procesu weryfikowanego w postaci struktury relacyjnej specyfikującej zachowanie badanego systemu
- procesu kryterialnego opisującego wymagane własności
- relacji kryterialnej definiującej odwzorowanie pomiędzy dwoma procesami

Poprawność (ujęta nieformalnie) definiowana jest jako własność zachowania procesu weryfikowanego polegająca na tym, że jego wybrane stany (nazywane stanami charakterystycznymi) osiągane są w kolejności zgodnej ze stanami procesu kryterialnego. Zadaniem relacji kryterialnej jest definicja asocjacji pomiędzy zbiorem stanów charakterystycznych a zbiorami stanów procesu kryterialnego.

Z punktu widzenia zastosowań najbardziej interesujące wydaje się użycie opisywanego modelu poprawności w fazach analizy wymagań i projektowania cyklu życia oprogramowania (chociaż można wskazać także zastosowania w fazie testowania). Proces kryterialny reprezentuje tu opis warstwy bardziej abstrakcyjnej (specyfikacji), natomiast proces weryfikowany reprezentuje wynik analizy możliwego zachowania specyfikacji bardziej szczegółowej (implementacji). Podejście to mieści się więc w modelu stopniowego uściślanie specyfikacji.

Zapewnienie jakości cyklu życia oprogramowania stawia wymóg opisu sposobu przydziału wymagań sformułowanych w wyższej warstwie do obiektów pojawiających się w kolejnej warstwie (ang. *requirements traceability*) [Boehm 84; US 94]. Relacja kryterialna jest formalną postacią tego opisu.

Niniejsza rozprawa mieści się w nurcie badań nad poprawnością względną procesów, bazując zarówno na podobnym modelu poprawności, jak i używając zbliżonego zestawu pojęć. Istotne modyfikacje dotyczą przede wszystkim dziedziny opisu i sposobu definiowania różnych pojęć.

2.6.1 Oznaczenia

Symbolem $\subset$ oznaczać będziemy inkluzję nieostrą; inkluzja ostra będzie zaznaczana jawnie jako $A \subset B \wedge A \neq B$.

Dla dowolnej relacji $R \subset X \times Y$ jej dziedzinę oznaczać będziemy jako *Dom*(*R*) natomiast przeciwdziedzinę jako *Ran*(*R*).

Dla dowolnej relacji $R \subset X \times Y$ i elementu $x \in X$ przez $R(x)$ oznaczać będziemy $R(x) = \{y \mid (x, y) \in R\}$. Dla dowolnego podzbioru A ⊂ X oznaczmy $R(A) = \bigcup_{x \in A} R(x)$.

Dla dowolnej relacji $R \subset X \times Y$ niech $R^{-1} = \{(y, x) \mid (x, y) \in R\}$

2.6.2 Pojęcie procesu

Centralnym pojęciem występującym w pracy jest pojęcie procesu sekwencyjnego. Jego definicja zaproponowana przez Pawlaka [Pawlak 69; BRS 77] stanowi rozszerzenie definicji automatu. Proces jest zdefiniowany jako relacyjna struktura (graf) z wyróżnionymi zbiorami stanów (węzłów) początkowych i końcowych.

Def. 2-4 Proces

Procesem nazywamy czwórkę $P = (S, B, F, T)$, gdzie

S – jest przeliczalnym zbiorem stanów,

$B \subset S$ – jest zbiorem stanów początkowych,

$F \subset S$ – jest zbiorem stanów końcowych,

$T \subset S \times$ S jest relacją przejścia,

spełniające warunek $B \subset Dom(T) \wedge F \cap Dom(T) = \varnothing$

■

Proces jest pojęciem ogólnym, które w zależności od znaczenia przypisywanego stanom i przejściom może służyć do modelowania różnych zjawisk. W odniesieniu do oprogramowania często podaje się następujące interpretacje stanu:

1. wartościowanie danych i etykieta aktualnej instrukcji (stan rejestru licznika instrukcji) dla procesu opisującego program sekwencyjny;
2. wektor danych i etykiet instrukcji składowych zadań dla systemu współbieżnego;
3. zawężenie stanu systemu do wektora wybranych danych i etykiet instrukcji.

Możliwe są również inne znaczenia. Czytelnik łatwo dalej dostrzeże, że w definicji procesu sprzężonego (Def. 2-9) stan jest interpretowany jako etap pewnego algorytmu.

2.6.3 Pojęcia poprawności całkowitej i częściowej

Elementami składowymi wprowadzonego w tym rozdziale modelu poprawności jest proces weryfikowany, oznaczany zwykle jako *P*, proces kryterialny oznaczony jako *P'* oraz relacja kryterialna *k*. Zakłada się, że proces weryfikowany *P* reprezentuje sekwencyjny opis zachowania systemu współbieżnego.

Def. 2-5 Semiobliczenie i obliczenie

Semiobliczeniem nazywamy dowolny *n*–elementowy ($n \geq 1$) ciąg stanów ($s_1, s_2, \ldots, s_n$), spełniający dla n ≥ 2 następujący warunek: $\forall i < n\,(s_i, s_{i+1}) \in T$. Semiobliczenie dwuelementowe nazywane będzie *przejściem*. W celu wyróżnienia stanu początkowego semiobliczenia stosować będziemy oznaczenie $sc(s_1, \bullet)$; podobnie stanu końcowego $sc(\bullet, s_n)$ oraz obu stanów $sc(s_1, s_n)$.

Dowolne semiobliczenie $sc(s_1,\bullet)$, gdzie $s_1 \in B$ nazywać będziemy *obliczeniem*, jeśli jest skończone i jego ostatni element $s_n \in F$ lub jest nieskończone.

Przez $Z(sc)$ oznaczymy zbiór wszystkich elementów semiobliczenia

■

Def. 2-6 Semiobliczenia dolne i górne

Dane są dwa procesy $P = (S, B, F, T)$ i $P' = (S', B', F', T')$ oraz relacja $k \subset S \times S'$. Dla dowolnej pary $(s, s') \in k$ zdefiniowane zostanie semiobliczenie dolne i górne ze względu na tę parę.

Semiobliczeniem dolnym ze względu na parę $(s, s') \in k$ nazywamy dowolne semiobliczenie $sc(s,\bullet)$ spełniające warunek:

$$\forall s_1 \in Z(sc(s,\bullet)) \,.\, \neg\exists s'_1 \in k(s_1) \,.\, (s', s'_1) \in T' \,.$$

Zbiór wszystkich semiobliczeń dolnych związanych z parą (s, s') oznaczymy przez $SL(s, s')$; jego uogólnienie dla zbiorów $X \subset S$ oraz $Y \subset S'$ oznaczony zostanie jako $SL(X,Y)$.

$$SL(X,Y) = \bigcup_{(s,s') \in X \times Y \cap k} SL(s,s')$$

Semiobliczeniem górnym ze względu na parę $(s, s') \in k$ nazywamy dowolne semiobliczenie $sc(s,\bullet)$ spełniające warunek:

$$\forall s_0 \in Z(sc(\bullet,s)) \,.\, \neg\exists s'_0 \in k(s_0) \,.\, (s'_0, s') \in T' \,.$$

Analogicznie zdefiniowany jest zbiór wszystkich semiobliczeń górnych związanych z parą (s, s') oznaczony przez $SU(s, s')$ oraz jego uogólnienie $SU(X,Y)$ dla zbiorów $X \subset S$ oraz $Y \subset S'$

■

Zdefiniujmy domknięcia zbiorów $SL(s_0, s')$ oraz $SU(s_n, s')$ jako:

$$\overline{SL}\,(s_0, s') = \{\, (s_0) \,\} \cup \{\, sc(s_0, s_n) \mid sc(s_0, s_{n-1}) \in SL(s_0, s') \,\}$$

$$\overline{SU}\,(s_n, s') = \{\, (s_n) \,\} \cup \{\, sc(s_0, s_n) \mid sc(s_1, s_n) \in SU(s_n, s') \,\}$$

oraz analogiczne uogólnienie dla zbiorów $X \subset S$ oraz $Y \subset S'$:

$$\overline{SL}(X,Y) = \bigcup_{(s,s') \in X \times Y \cap k} \overline{SL}(s,s'),$$

$$\overline{SU}(X,Y) = \bigcup_{(s,s') \in X \times Y \cap k} \overline{SU}(s,s').$$

Def. 2-7 Definicja częściowej poprawności

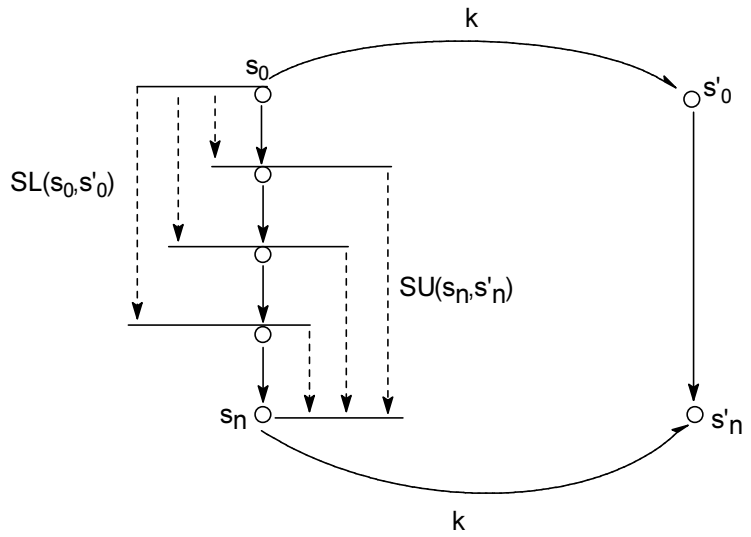
Dane są dwa procesy $P = (S, B, F, T)$ i $P' = (S', B', F', T')$ oraz relacja $k \subset S \times S'$. Proces P jest częściowo poprawny w sensie kryterium (k, P') wtw. jeśli spełnione są poniższe warunki:

1. $SU(B, S') = SU(B, B')$ oraz $SL(F, S') = SL(F, F')$
2. Dla każdego $s_n' \in Ran(T') \cap Ran(k)$ istnieją stany s_0, s_n oraz s_0' takie, że: $s_0' \in T'^{-1}(s_n') \cap Ran(k)$ oraz spełnione jest: $\overline{SL}\,(s_0, s_0') \cap \overline{SU}\,(s_n, s_n') \neq \varnothing$

■

Warunek (1) definicji oznacza, że dla każdego semiobliczenia procesu weryfikowanego jako pierwsze pojawiać się będą wyłącznie te stany charakterystyczne, które przez relację *k* są odwzorowane w zbiór *B'* procesu kryterialnego. Podobnie, ostatnimi stanami charakterystycznymi wszystkich semiobliczeń muszą być te, które odwzorowane są w zbiór *F'*.

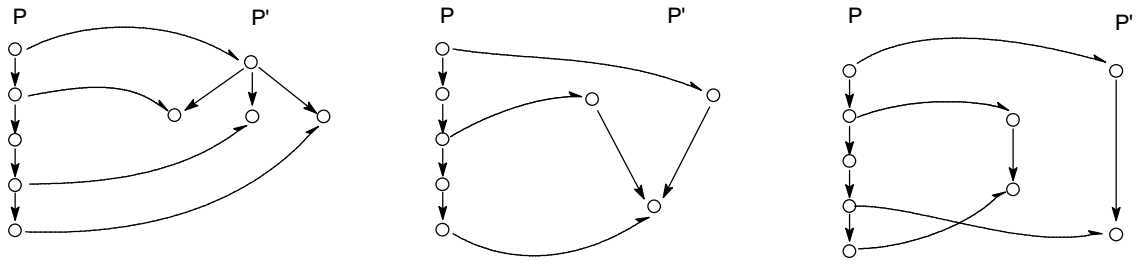
Warunek (2) dotyczy stanów procesu kryterialnego, które nie są stanami początkowym i należą do przeciwdziedziny relacji *k*. Stany te powinny być osiągalne w wyniku przejść należących do relacji *T'* zawężonej do przeciwdziedziny relacji *k*. (Czyli $T_R' = \{(s_1', s_2') \in T' \mid s_1', s_2' \in Ran(k)\}$). Dla takich przejść żąda się, by istniało semiobliczenie procesu weryfikowanego, które jest przecięciem domknięć zbiorów semiobliczeń dolnych i górnych związanych ze stanami wyznaczającymi przejście kryterialne (Rys. 2-1).



Rys. 2-1 Ilustraczna graficzna warunku (2) definicji poprawności

Warunek (2) definicji poprawności zezwala na wystąpienie ciągu stanów charakterystycznych, z którymi nie jest bezpośrednio związane semiobliczenie procesu kryterialnego. (Rys. 2-2 a i b). W szczególności dany ciąg może być odwzorowany w przejścia równoległe (Rys. 2-2 c). Jak można jednak zauważyć, żąda się, by osiągniecie danego stanu procesu kryterialnego s_2' było poprzedzone wcześniejszym osiągnięciem jednego z jego stanów poprzednich s_1' spełniających $s_2' \in T'(s_1')$.

Dopuszczalność odwzorowań, których przykłady daje rysunek, jest uzasadniona biorąc pod uwagę założenie, że proces weryfikowany reprezentuje zachowanie systemu współbieżnego. Niezachowanie kolejności stanów lub odwzorowanie semiobliczenia w przejścia równoległe procesu kryterialnego łatwo wyjaśnić zmianami aktywności składowych procesów (zadań) lub wykonaniem operacji synchronicznej (np.: komunikacji) przez dwa procesy



(a) (b) (c)

Rys. 2-2 Różne warianty odwzorowania semiobliczeń w przejścia procesu kryterialnego

Oznaczmy przez $SC(X, Y)$ oznacza zbiór dolnych lub górnych semiobliczeń w odniesieniu do zbiorów $X \subset S$ oraz $Y \subset S'$. Przez $CH(\ SC(X, Y)\ )$ oznaczymy zbiór wszystkich stanów procesu kryterialnego, które posłużyły do utworzenia semiobliczeń ze zbioru $SC(X, Y)$.

$$CH(\ SC(X, Y)\ ) = \{\ s' \mid \exists\ s\ .\ SC(s, s') \subset SC(X, Y)\ \}$$

Wniosek 2-1

Jeśli proces jest częściowo poprawny w sensie kryterium (P',k), wówczas:

1. $CH(\ SU(B, S')\ ) \subset B'$
2. $CH(\ SL(F, S')\ ) \subset F'$
3. $Ran\ (k) \subset S'$

■

Def. 2-8 Całkowitej poprawności

Dane są dwa procesy $P = (S, B, F, T)$ i $P' = (S', B', F', T')$ oraz relacja $k \subset S \times S'$. Proces P jest częściowo poprawny w sensie kryterium (k, P') wtw. jeśli spełnione są poniższe warunki:

1. $CH(\ SU(\ B, S'\ )\ ) = B'$ oraz $CH(\ SL(\ F, S'\ )\ ) = F'$
2. Dla każdego $s_n' \in Ran(\ T'\ )$ istnieją stany s_0, s_n oraz s_0' takie, że: $s_0' \in T'^{-1}(\ s_n'\ )$ oraz spełnione jest: $\overline{SL}\ (\ s_0, s_0'\ ) \cap \overline{SU}\ (\ s_n, s_n'\ ) \neq \varnothing$

■

Wniosek 2-2

Jeśli proces jest całkowicie poprawny w sensie kryterium (P',k), wówczas:

1. $CH(\ SU(B, S')\ ) = B'$
2. $CH(\ SL(F, S')\ ) = F'$
3. $Ran\ (k) = S'$

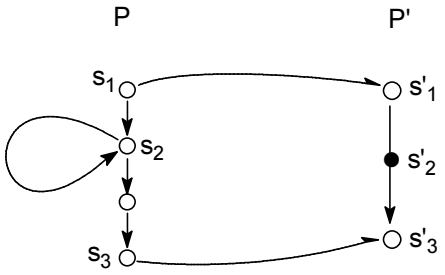
■

Równość (3) wypływa bezpośrednio ze zmienionego warunku (2) definicji.

W pracy [Szmuc 89b, 91] pokazano przez konstrukcję odpowiednich procesów, że zaproponowane pojęcia częściowej i całkowitej poprawności względnej są rozszerzeniem klasycznego pojęcia definiowanego dla programów sekwencyjnych.

Konieczność zatrzymania programu w przypadku poprawności całkowitej jest w tym modelu zastąpiona przez wymaganie zgodności pomiędzy nieskończonymi pętlami procesu weryfikowanego i kryterialnego, które jest bardziej adekwatne dla oprogramowania współbieżnego. Problemem związanym z taką interpretacją jest jednak fakt, że nie wszystkie pętle muszą być odwzorowane przez relację *k* (Rys. 2-3). W cytowanej pracy sugerowane jest w takim przypadku rozszerzanie procesu kryterialnego o dodatkowe stany pośrednie (s_2' na przykładowym rysunku.)

Poszukując innych podobieństw można wskazać zbieżność pojęcia częściowej poprawności z własnością bezpieczeństwa oraz całkowitej poprawności z własnością żywotności formułowanych z użyciem logiki temporalnej (por. podrozdział 2.4.). W stosunku do równoważności własności całkowitej poprawności i żywotności można jednak wysunąć analogiczne zarzuty dotyczące pętli.



Rys. 2-3 Pętla w procesie weryfikowanym dla której brak jest odwzorowania w procesie kryterialnym

Problem uzgadniania pętli może zostać rozwiązany przez dodatkowe założenia dotyczące sprawiedliwości systemu, podobnie jak zaproponowano to w [KMP 94] (por. 2.5.1) Mogą one prowadzić one do wykluczenia ze zbiorów obliczeń procesu *P* nieskończonego ciągu $\langle s_1, s_2, s_2, \ldots \rangle$. W cytowanych pracach założenia takie nie występują; wydają się one kłopotliwe do wyspecyfikowania dla rzeczywistych systemów poddanych weryfikacji.

2.6.4 Narzędzia formalne weryfikacji poprawności

Rozdział ten definiuje konstrukcje formalne, które są podstawą do implementacji algorytmów weryfikacji poprawności. Obejmują one: *proces sprzężony* opisujący zasady równoległej analizy przebiegu procesu weryfikowanego i kryterialnego oraz *twierdzenie o lokalnej testowalności* określające warunki konieczne i wystarczające, jakie musi spełniać proces sprzężony, aby proces weryfikowanego był poprawny w sensie kryterium.

Def. 2-9 Proces sprzężony

Niech $P = (S, B, F, T)$ i $P' = (S', B', F', T')$ będą procesami a k relacją $k \subset S \times S'$. Proces sprzężony jest czwórką $\hat{P} = (\hat{S}, \hat{B}, \hat{F}, \hat{T})$, gdzie

1. $\hat{S} \subset S \times M$, $M = 2^{S'} \times 2^{S'} \times 2^{S'}$ jest zbiorem stanów
2. $\hat{B} = \{(s, (pS', pB', pF')) \mid s \in B \wedge pS' = (B' \cup T'(k(s)) \setminus k(s) \wedge pB' = B' \cap k(s) \wedge pF' = k(s) \}$; $\hat{B}$ jest zbiorem stanów początkowych
3. $\hat{F} = \{(s, (pS', pB', pF')) \mid s \in F\}$ jest zbiorem stanów końcowych
4. $\hat{T}$ jest relacją.

   Dla każdej pary $((s, (pS', pB', pF')), (s_1, (pS_1', pB_1', pF_1'))) \in \hat{T}$ spełnione są następujące warunki:

   a) $(s, s_1) \in T$

   b) $pS_1' = pS' \cup T'(k(s_1)) \setminus k(s_1)$

   c) $pB_1' = pB' \cup (B' \cap k(s_1))$

   d) $pF_1' = pF' \cup k(s_1) \setminus (T'^{-1}(k(s_1)) \setminus F_*')$

■

Stan procesu sprzężonego jest parą, której pierwszy element należy do zbioru stanów procesu weryfikowanego, natomiast drugi określa tzw. stan pamięci. Zbudowany jest on z trzech składowych (pS', pB', pF'). Składowe te odpowiedzialne są za przechowywanie informacji o historii odwzorowania stanów charakterystycznych procesu weryfikowanego w stany procesu kryterialnego oraz o stanach oczekiwanych. Składowa pS' określa następne dopuszczalne stany, składowa pB' osiągnięte stany początkowe procesu kryterialnego, natomiast składowa pF' definiuje zbiór stanów, które mogą poprzedzać poprawnie osiągane stany przyszłe.

Zbiór F_*' jest zbiorem tzw. stanów pętlących procesu kryterialnego. Zauważmy, że dla procesu o skończonej liczbie stanów, obliczenie nieskończone może pojawić się jedynie w obecności pętli. Każde nieskończone obliczenie jest elementem pewnego zbioru, który może być reprezentowany przez skończone wyrażenie zbudowane ze skończonych ciągów elementów. Ostatnim elementem takiego ciągu jest pierwszy element pętli oznaczany symbolem „*" [Szmuc 91].

Proces sprzężony opisuje sposób odwzorowania przez relację kryterialną k przejść w procesie weryfikowanym w zbiory przejść w procesie kryterialnym. W szczególności, kiedy dany stan s_1 procesu weryfikowanego nie należy do dziedziny relacji k, stan składowej pamięci procesu sprzężonego nie ulega zmianie. Osiągnięcie stanu charakterystycznego w procesie weryfikowanym związane jest ze zmianą składowej pamięci. Zmiana ta dla procesu poprawnego w sensie kryterium nie jest jednak dowolna. Poniżej przedstawione twierdzenie formułuje warunki, jakie muszą spełniać przejścia w procesie kryterialnym.

Twierdzenie 2-1 O lokalnej testowalności

Niech $P = (S, B, F, T)$ i $P' = (S', B', F', T')$ będą procesami a k relacją $k \subset S \times S'$. Niech $\hat{P}$ będzie procesem sprzężonym. Proces P jest całkowicie poprawny w sensie kryterium (k, P') wtw. jeśli proces sprzężony spełnia następujące warunki:

1. $\forall\, s \in B\,.\;\; k(s) \subset B'$
2. $\forall\, (s, (\, pS', pB', pF'\,)) \in \hat{F} \;\; pF' \subset F'$
3. Dla dowolnego przejścia $(\,(s, (\, pS', pB', pF'\,)), (s_1, (\, pS_1', pB_1', pF_1'\,))\,) \in \hat{T}$ spełniony jest warunek $k(s_1) \subset pS'$
4. Wystąpiły wszystkie stany początkowe: $\bigcup_{(s,(pS',pB',pF'))\in\hat{F}} pB' = B'$
5. Wystąpiły wszystkie stany końcowe: $\bigcup_{(s,(pS',pB',pF'))\in\hat{F}} pF' = F' \cup F_*' \cup \overline{F_*'}$,

   gdzie $\overline{F_*'}$ jest pewnym, w szczególności pustym podzbiorem stanów tych obliczeń, które nie należą do F_*' i występują w obliczeniach kończących się w stanach ze zbioru F_*'.

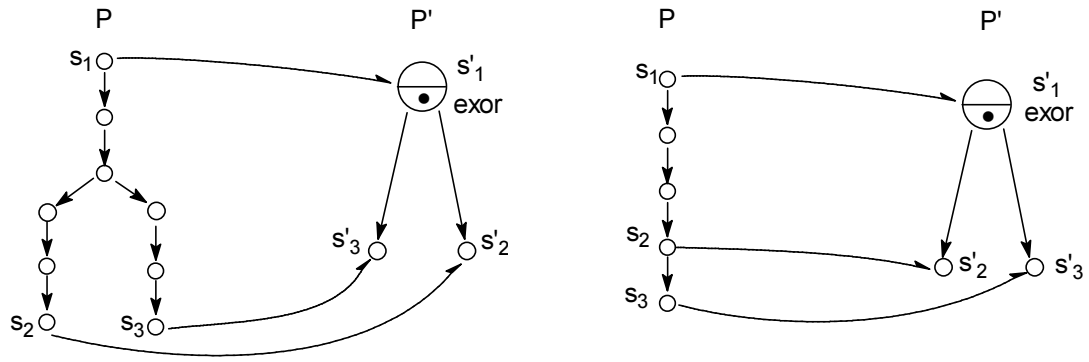
■

W przypadku częściowej poprawności warunki (4) i (5) są pomijane. Dowód twierdzenia dla przypadków częściowej i całkowitej poprawności zamieszczony jest w [Szmuc 89b, 91]. Polega on na dekompozycji procesu weryfikowanego na zbiór procesów, których odpowiednie stany początkowe i końcowe odpowiadają stanom charakterystycznym. Dowód wykorzystuje opis procesu weryfikowanego w postaci wyrażenia zbudowanego z symboli procesów i zdefiniowanych w cytowanych pracach operatorów algebry procesów. Stworzone wyrażenie jest rzutowane w zbiór stanów procesu kryterialnego.

2.6.5 Rozszerzenia w definicji procesu kryterialnego

Podstawowa relacyjna struktura procesu kryterialnego poddana została w kolejnych pracach [Szmuc 93, 97a, 97b] modyfikacjom. Rozszerzenia te nie będą tutaj formalnie opisywane, ponieważ taka postać nie będzie używana bezpośrednio w pracy. Opis formalny prowadzony jest w kategoriach funkcji klasyfikującej typy stanów ze względu na sposób ich osiągania oraz opuszczania.

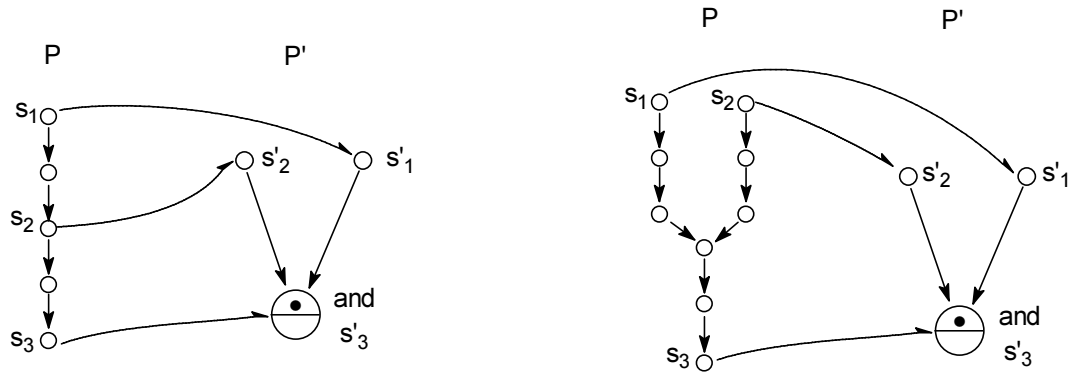
Podstawowym rozszerzeniem jest modelowanie niedeterministycznego wyboru (*exor*) jednego z przejść w procesie kryterialnym. Rys. 2-4 przedstawia poprawny i niepoprawny wariant odwzorowania fragmentu procesu weryfikowanego w przejścia procesu kryterialnego. Charakter funkcji wyjściowej stanu s_1' powoduje, że po osiągnięciu stanu charakterystycznego związanego ze stanem s_2' przejście do stanu s_3' traktowane jest jako niepoprawne.



(a) *(b)*

Rys. 2-4 Poprawne (a) i niepoprawne (b) warianty odwzorowania dla węzła exor

Stan opisany funkcją wejściową typu *and* wprowadzony został przede wszystkim w celu modelowania zjawisk synchronizacji procesów. Przed osiągnięciem wyróżnionego stanu konieczne jest wcześniejsze osiągnięcie wszystkich stanów poprzedzających. Uwzględniając te uwagi, odwzorowanie na Rys. 2-5 (a) jest poprawne, natomiast na Rys. 2-5 (b) jest niepoprawne, ponieważ dla semiobliczenia $sc(s_1, s_3)$ stan s_2' nie jest osiągany przed stanem s_3'.



(a) *(b)*

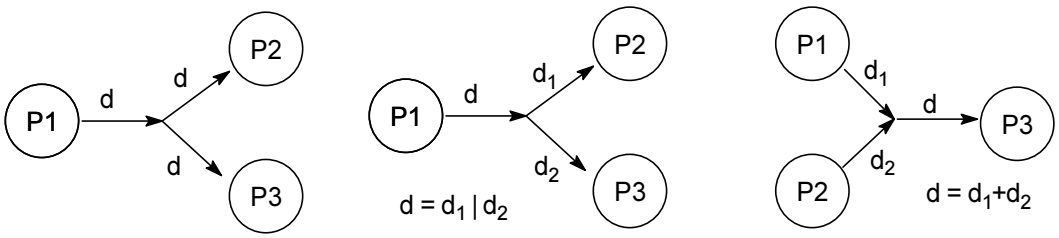
Rys. 2-5 Poprawne (a) i niepoprawne (b) warianty odwzorowania dla węzła and

Innym, możliwym do wskazania, zastosowaniem wprowadzonych rozszerzeń jest lepsze modelowanie wymagań wypływających z interpretacji diagramów przepływu danych (DFD) [Yourdon 88; Perez 90], a w szczególności definicji słowników danych. Na Rys. 2-6 przedstawiono kilka wariantów specyfikacji DFD które mogą być modelowane z użyciem rozmaitych typów węzłów.

W przypadku (a) jest to podstawowy typ węzła (*or*). Zakończenie procesu *P1* prowadzi do pojawienia się danej *d* na wyjściu. Procesy *P2* i *P3* mogą zostać uaktywnione w dowolnej kolejności.

W przypadku (b) modelowanie powinno być prowadzone z wykorzystaniem węzła *exor*, ponieważ dana *d* została zdefiniowana w słowniku jako jedna z wartości d_1 lub d_2. W zależności od obliczonego rezultatu uaktywniony powinien zostać jeden z procesów *P2* i *P3*.

Przypadek (c) opisuje sytuację, kiedy dana *d* jest rekordem złożonym z danych d_1 i d_2. Wówczas uaktywnienie procesu *P3* musi zostać poprzedzone wcześniejszym skompletowaniem rekordu, czyli zakończeniem działania procesów *P1* i *P3*.



(a) *(b)* *(c)*

Rys. 2-6 Warianty specyfikacji DFD

2.6.6 Weryfikacja oprogramowania współbieżnego

Zasadnicza procedura zastosowania opisanego modelu poprawności do weryfikacji oprogramowania składa się z trzech etapów:

1. konstrukcji procesu sekwencyjnego opisującego system współbieżnych procesów;
2. redukcji procesu sekwencyjnego (opcjonalnie)
3. właściwej weryfikacji otrzymanego procesu sekwencyjnego względem zdefiniowanego kryterium.

Etap 1

W pracach [Szmuc 89b, 97a, 97b] podano definicję procesu sekwencyjnego opisującego system procesów. Początkowo zasady jego konstrukcji uwzględniały dzieloną pamięć procesów, w ostatnich pracach uwzględniono także mechanizmy komunikacji. Pod tym względem rozważany model jest bliski modelowi synchronicznie komunikujących się maszyn skończenie stanowych. (Por. rozdział 2.2.1)

Proces sekwencyjny opisujący system procesów budowany jest przez wykonanie przejść poszczególnych procesów składowych z przeplotem oraz wspólne wykonanie przejść synchronicznych. Łatwo tu odnaleźć analogie do konstrukcji drzewa stanów oisągalnych sieci Petriego.

Etap 2

Etap redukcji procesu sekwencyjnego opisującego system współbieżny opisany został w [Szmuc 89b]. Polega on na usuwaniu ze zbioru S stanów procesu weryfikowanego i elementów zbioru $S \setminus Dom(k)$ przy równoczesnym scalaniu przejść. Procedura redukcji opiera się na analizie wyrażeń ścieżkowych opisujących proces. Etap ten nie jest wymagany do przejścia do etapu następnego, ponieważ z punktu widzenia zarówno narzędzi formalnych jak i opracowanych algorytmów zredukowany proces weryfikowany jest szczególnym przypadkiem obiektu występującym w ogólnym modelu.

Etap 3

Przedmiotem weryfikacji jest niedeterministyczny proces sekwencyjny opisujący system współbieżny. Weryfikacja polega na budowie procesu sprzężonego $\hat{P}$ i zastosowaniu twierdzenia o lokalnej dla badania jego własności.

Konstrukcja modułu realizującego weryfikację oraz zastosowane algorytmy zostały szczegółowo przedstawione w [SS 91]. Tutaj omówione zostaną one jedynie w sposób opisowy.

Jak łatwo zauważyć, definicja procesu sprzężonego jest rekurencyjna; nie jest możliwe podanie bezpośredniego odwzorowania, które przekształca zbiór stanów procesu weryfikowanego S w zbiór stanów procesu sprzężonego $\hat{S}$. Badanie poprawności wymaga więc w praktyce analizy semiobliczeń procesu weryfikowanego postaci $sc(s_1, \bullet) = (s_1, s_2, \ldots, s_n)$ rozpoczynających się w stanie początkowym ($s_1 \in B$) i równoczesnej konstrukcji odpowiadających im semiobliczeń procesu sprzężonego $sc(\hat{s}_1, \bullet) = (\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_n)$ zgodnie z warunkiem (4) Def. 2-9

Analiza semiobliczenia $sc(s_1, \bullet)$ ulega zakończeniu, jeśli konstruowane równolegle semiobliczenie procesu sprzężonego jest obliczeniem. Opisane jest to następującym warunkiem:

Semiobliczenie $sc(\hat{s}_1, \hat{s}_n) = (\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_n)$ procesu sprzężonego jest obliczeniem wtw.

1. $\hat{s}_n \in \hat{F}$ lub
2. $\exists \hat{s} \in Z(sc(\hat{s}_1, \hat{s}_{n-1}))\,.\,\hat{s} = \hat{s}_n$

Pierwszy składnik warunku związany jest z osiągnięciem stanu końcowego procesu weryfikowanego; drugi składnik związany jest z osiągnięciem stanu pseudo-końcowego (pętli w procesie sprzężonym). Zbiór stanów, które spełniają drugi składnik warunku oznaczany był w warunku (5) twierdzenia o lokalnej testowalności jako $\overline{F_*'}$.

Realizacja praktyczna algorytmu nie musi w ogólnym przypadku zakładać bezpośrednio analizy semiobliczeń. Możliwą wariacją w stosunku algorytmu podstawowego jest konstrukcja skończonego ciągu $\langle \hat{P}_0, \hat{P}_1, \hat{P}_2, \ldots, \hat{P}_n \rangle$ zbieżnego do $\hat{P}$. Może ona polegać na zapamiętywaniu osiągniętych stanów procesu sprzężonego i sprawdzaniu w *i*-tej iteracji, czy ostatni stan $\hat{s}_{ki}$ semiobliczenia $sc(\hat{s}_1, \hat{s}_{ki})$ spełnia warunek $\hat{s}_{ki} \in \hat{S}_{i-1}$. W takim wypadku analiza danego semiobliczenia nie jest dalej kontynuowana. Takie właśnie rozwiązanie przyjęto w pracy przy konstrukcji procesu sprzężonego dla zmienionej dziedziny modelu poprawności.

2.6.7 Analiza procesów wchodzących w skład modelu poprawności

Definicja poprawności w wersji podstawowej formułuje wymagania dotyczące relacyjnych zależności pomiędzy dwiema analogicznie zdefiniowanym strukturami procesów weryfikowanego i kryterialnego. W przypadku modelowania własności quasi-determinizmu oraz synchronizacji definicja ta uległa zmianie [Szmuc 89b, 93]; zmodyfikowane także zostały warunki opisujące przejścia w procesie sprzężonym, niemniej ogólna postać zagadnienia pozostała niezmieniona.

Analiza przeprowadzona w tym podrozdziale opiera się na założeniu, że zarówno proces weryfikowany *P* jak i proces kryterialny *P'* reprezentują pewne specyfikacje zachowania. Omówione zostaną podstawowe różnice pomiędzy zasadami interpretacji tych specyfikacji wynikające z konstrukcji procesu sprzężonego.

Proces weryfikowany jest interpretowany jako niedeterministyczny proces sekwencyjny. Zakłada się, że każdym momencie wyróżniony jest dokładnie jeden stan aktywny. Jeśli stanem aktywnym jest s_i, wówczas wraz z wykonaniem przejścia $(s_i, s_{i+1}) \in T$ następuje wyróżnienie stanu s_{i+1} jako stanu aktywnego. W przypadku rozgałęzień możliwy jest

niedeterministyczny wybór pomiędzy możliwymi przejściami. Obserwowane zachowanie procesu weryfikowanego rozpoczyna się w momencie uaktywnienia jednego ze stanów początkowych; jest ono wynikiem arbitralnej decyzji podejmowanej poza systemem opisanym przez proces.

Interpretacja zachowania **procesu kryterialnego** (mimo użycia analogicznej postaci definicji) jest zasadniczo odmienna. W danym momencie analizy (a dokładniej weryfikacji) wyróżniony jest pewien zbiór stanów aktywnych opisany składową *pF'* procesu sprzężonego. Zbiór ten może być w szczególności zbiorem pustym. Równocześnie przechowywana jest informacja o przyszłych dopuszczalnych stanach w postaci składowej *pS'*. Jak łatwo zauważyć, w ten sposób pośrednio opisany jest zbiór dopuszczalnych przejść, który można zdefiniować jako $(pF' \times pS') \cap T'$.

Definicja zbioru *pS'* pozwala w przypadku rozgałęzień typu *or* na wykonanie wszystkich przejść wychodzących z danego węzła (co odpowiada w przybliżeniu zasadom interpretacji diagramów przepływów danych DFD).

W przypadku węzłów typu *exor* (przypadek quasi-determinizmu) formuły opisujące składową pamięci *pS'* są odpowiednio modyfikowane [Szmuc 89b], aby wykonanie jednego z przejść (np.: (s_1', s_2') na Rys. 2-4) powodowało usunięcie ze zbioru *pS'* wszystkich stanów następujących po danym węźle (s_2' dla omawianego przykładu.)

Podobnie dla stanów typu *and* (z synchronizacją na wejściu) stan ten może się pojawić w zbiorze *pS'* jeśli wszystkie stany poprzedzające należą do zbioru *pF'*.

Charakterystyczną własnością konstrukcji procesu sprzężonego jest to, że w jego poprawnym semiobliczeniu stany początkowe procesu kryterialnego mogą być osiągane jednokrotnie (wynika to ze sposobu przekształacania składowej *pS'*). Własność ta uniemożliwia związanie stanu początkowego procesu kryterialnego z asynchronicznie pojawiającym się zdarzeniem generowanym przez otoczenie systemu. W modelu przyjętym w pracy założenie to zostanie osłabione, aby dopuścić możliwość takiej definicji wymagań. Wydaje się to bardzo naturalne dla wielu nieformalnych specyfikacji – zwłaszcza wykorzystujących konstrukcje analizy strukturalnej.

2.6.8 Inne spojrzenie na zagadnienie poprawności względnej

Autorowi bliższa jest intuicja poprawności względnej wynikająca z formuł definiujących proces sprzężony niż bezpośrednio z definicji. Interpretacja procesu sprzężonego pozwala traktować poprawność względną jako pewną własność semiobliczeń procesu weryfikowanego.

Omówione powyżej własności sugerują inną konstrukcję modelu poprawności – równoważną warunkom twierdzenia o lokalnej testowalności. Opiera się ona na założeniu, że zachowanie procesu kryterialnego traktowanego jako specyfikacja można opisać w postaci semiobliczenia pewnego niedeterministycznego procesu sekwencyjnego.

Model poprawności będzie w pracy formułowany w kategoriach własności odwzorowania przekształcającego semiobliczenia (lub inne bliskie im reprezentacje zachowania) procesu weryfikowanego w semiobliczenia procesu sekwencyjnego opisującego potencjalne zachowanie przejawiane przez proces kryterialny.

Poniższa definicja obrazuje ideę konstrukcji niedeterministycznego procesu sekwencyjnego opisującego zachowanie najprostszego modelu procesu kryterialnego (bez węzłów *exor* i *and*).

Def. 2-10 Proces sekwencyjny opisujący zachowanie procesu kryterialnego

Niech $P' = (S', B', F', T')$ oznacza proces kryterialny.

Proces opisujący zachowanie procesu kryterialnego jest czwórką $\widetilde{P} = (\widetilde{S}, \widetilde{B}, \widetilde{F}, \widetilde{T})$, gdzie

1. $\widetilde{S} \subset M$, $M = 2^{S'} \times 2^{S'} \times 2^{S'}$ jest zbiorem stanów
2. $\widetilde{B} = \{ (ps', pb', pf') \mid ps' = (B' \cup T'(pb') \setminus pb' \wedge pb' \in 2^{B'} \wedge pf' = pb' \}$; $\widetilde{B}$ jest zbiorem stanów początkowych
3. $\widetilde{F} = \{((ps', pb', pf') \mid ps' = \varnothing \wedge pb' = B' \wedge pf' \subset F' \}$ jest zbiorem stanów końcowych
4. $\widetilde{T}$ jest relacją. Dla każdej pary $((ps', pb', pf') , (ps_1', pb_1', pf_1')) \in \widetilde{T}$ spełnione są następujące warunki:
   a) istnieje niepusty podzbiór stanów *reached* osiągniętych w danym przejściu: $reached \subset ps' \cup (B' \setminus pb') \wedge reached \neq \varnothing$
   b) $ps_1' = ps' \cup T'(reached) \setminus reached$
   c) $pb_1' = pb' \cup (B' \cap reached)$
   d) $pf_1' = pf' \cup reached \setminus (T'^{-1}(reached))$

■

Jak łatwo zauważyć (przez odpowiednie podstawienie $k(s) = reached$, $pS' = ps'$, itd.) zdefiniowany powyżej proces $\widetilde{P}$ pokrywa wszystkie stany i przejścia składowej pamięci procesu sprzężonego $\hat{P}$ wprowadzając równocześnie dodatkowe stany i przejścia. Zachowanie składowej pamięci procesu sprzężonego można opisać odpowiednio skonstruowanym procesem postaci $\hat{P}_M = (\hat{S}_M, \hat{B}_M, \hat{F}_M, \hat{T}_M)$. Proces ten jest w istocie podprocesem procesu $\widetilde{P}$, który w przypadku zachowania poprawności spełnia $\hat{S}_M \subset \widetilde{S}$, $\hat{B}_M \subset \widetilde{B}$, $\hat{F}_M \subset \widetilde{F}$, $\hat{T}_M \subset \widetilde{T}$.

Jest to uzasadnione, ponieważ proces kryterialny może być traktowany jako specyfikacja wymagań dla rodziny procesów weryfikowanych, które mogą powstać jako rezultat ich uściślania dla kolejnej warstwy abstrakcji opisu oprogramowania. Naturalnym rezultatem decyzji projektowych jest dodawanie ograniczeń, które jednak powinny być w zgodzie z ogólnymi własnościami zdefiniowanymi w poprzedniej warstwie.

Def. 2-11 Poprawność semiobliczenia względem procesu sekwencyjnego opisującego zachowanie procesu kryterialnego

Niech $P = (S, B, F, T)$ i $P' = (S', B', F', T')$ będą procesami a k relacją $k \subset S \times S'$.

Niech $\widetilde{P}$ będzie procesem opisującym zachowanie specyfikacji zdefiniowanym dla P'

Semiobliczenie $sc(s_1, s_n) = (s_1, s_2,\ldots, s_n)$, gdzie $s_1 \in B$, jest poprawne w sensie kryterium jeśli istnieje semiobliczenie procesu $\widetilde{P}$ postaci $sc(\widetilde{s}_1, \widetilde{s}_m) = (\widetilde{s}_1, \widetilde{s}_2,\ldots, \widetilde{s}_m)$, gdzie $m \le n$ oraz zachowująca uporządkowanie funkcja J: $\{k \in \mathrm{N} \mid k \le n\} \to \{k \in \mathrm{N} \mid k \le m\}$ takie, że:

1. $k(s_1) = pB_1' \wedge J(1) = 1$
2. Dla dowolnego $1 < i \le n$ zachodzą następujące warunki:

   a) $k(s_i) = \varnothing \quad \Rightarrow \quad J(i) = J(i-1)$

   b) $k(s_i) \neq \varnothing \quad \Rightarrow \quad J(i) = J(i-1) + 1$

   oraz istnieje zbiór *reached* służący do wyznaczenia przejścia $(\widetilde{s}_{J(i-1)}, \widetilde{s}_{J(i)})$ (zgodnie z Def. 2-10) taki, że *reached* = $k(s)$
3. Jeżeli $s_n \in F$ to dla odpowiadającego mu $\widetilde{s}_m = (ps'_m, pb'_m, pf'_m)$ zachodzi:

   $pf_m' \subset F' \quad \wedge \quad pS_n' = B' \setminus pb'_m$

■

Def. 2-12 Poprawności częściowej procesu względem procesu sekwencyjnego opisującecgo zachowanie procesu kryterialnego

Niech $P = (S, B, F, T)$ i $P' = (S', B', F', T')$ będą procesami a k relacją $k \subset S \times S'$.

Proces P jest częściowo poprawny względem kryterium (k, P') wtw. jeżeli wszystkie semiobliczenia są poprawne.

■

W przypadku całkowitej poprawności należy dodać są warunki analogiczne do (5,6) twierdzenia o lokalnej testowalności.

Wniosek 2-3

Warunki twierdzenia o lokalnej testowalności oraz Def. 2-12 są sobie równoważne.

■

Łatwo zauważyć bezpośrednie analogie pomiędzy konstrukcją procesu sprzężonego $\hat{P}$ i sposobem jego wykorzystania w twierdzeniu o lokalnej testowalności, a procesem $\widetilde{P}$ oraz jego wykorzystaniem w powyższych definicjach. Dla dowolnego semiobliczenia procesu weryfikowanego $sc(s_1, s_n)$ oraz odpowiadającego mu semiobliczenia procesu $\widetilde{P}$ postaci $sc(\widetilde{s}_1, \widetilde{s}_m)$ można stworzyć odpowiednie semiobliczenie procesu sprzężonego $\hat{P}$ postaci $sc(\hat{s}_1, \hat{s}_n)$, gdzie $\hat{s}_i = (s_i, \hat{s}_{J(i)})$. Jak wynika z Def. 2-11 dla poprawnego (niepoprawnego) semiobliczenia poszczególne pary $(\hat{s}_i, \hat{s}_{i+1})$ także spełniają (nie spełniają) warunki twierdzenia o lokalnej testowalności.

Zaproponowana wyżej modyfikacja definicji poprawności bliska jest relacjom symulacji i uściślania omawianym w podrozdziale 2.5.1. Podobnie, konstrukcja procesu sprzężonego bliska jest koncepcji produktu synchronicznego dwóch automatów stosowanych przy weryfikacji własności opisanych logiką temporalną (Por. 2.4.4). Występują tu oczywiste różnice: proces kryterialny traktowany jak automat nie realizuje negacji. W zamian za to akceptuje wszystkie stany, dzięki czemu konstrukcja jest efektywna przy badaniu inkluzji języków.

Rozwijane w pracy podejście zakłada użycie odwzorowania przekształcającego reprezentację zachowania procesu weryfikowanego w postaci ciągu wektorów akcji (przejść) w analogiczną reprezentację zachowanie procesu kryterialnego. Podejście to wydaje się autorowi tym bardziej właściwe w obecności rozszerzeń definicji stanów.

Traktując proces kryterialny jako specyfikację, dla której opis zachowania reprezentowany jest przez proces $\widetilde{P}$ łatwo skonstruować przykłady wykorzystujące węzły typu *and* oraz *exor*, w których nie wszystkie stany są osiągalne, tzn. $\bigcup_{(ps',pb',pf')\in\widetilde{S}} pf' \neq S'$. Oznacza to, że nie istnieje żaden proces całkowicie poprawny w sensie kryterium (k, P').

Badanie tego zjawiska (określanego dalej jako brak spójności wymagań) nie będzie centralnym motywem pracy, ponieważ nie stosuje się do niego bezpośrednio model oparty na porównaniu zachowania dwóch procesów. Można tu znaleźć bliskie analogie do badania własności żywotności sieci Petriego.

2.7 Problemy zastosowań metod automatycznej analizy

Wiele opracowanych teoretycznie metod automatycznej analizy specyfikacji opiera się na całkowitym lub częściowym przeglądzie zbioru stanów pewnego procesu osiągalnych ze stanu początkowego. Przykładem jest analiza własności sieci Petriego w oparciu o grafy znakowań osiągalnych, czy opisane zasady weryfikacji poprawności względnej wykorzystujące konstrukcję procesu sprzężonego. Podobne algorytmy tworzone są dla weryfikacji własności opisanych logiką temporalną dla systemów komunikujących się maszyn skończenie stanowych [Holzmann 91, 97; GH 93]. W większości przypadków do stwierdzenia, że system posiada wymagane własności niezbędny jest pełny przegląd przestrzeni stanów osiągalnych.

Podstawową trudnością ograniczającą praktyczne zastosowanie metod automatycznych jest problem eksplozji przestrzeni stanów. Ich liczba na ogół nie jest uzależniona od wielkości specyfikacji, lecz od jej struktury. Złożoność obliczeniowa algorytmów badania własności systemów w oparciu o analizę zbioru stanów osiągalnych jest na ogół bardzo duża (PSPACE zupełna). W wielu przypadkach przestrzeń stanów badanego systemu może być nieskończona, na przykład dla modeli zakładających asynchroniczną komunikację pomiędzy procesami połączonych kanałami o nieskończonych pojemnościach. Dla takiego przypadku istnieją formalne dowody, że wyrokowanie o takich własnościach jak brak blokowania jest nierozstrzygalne [Holzmann 91].

Opisane wyżej problemy zaowocowały algorytmami częściowego przeszukiwania przestrzeni stanów oraz opracowaniem różnych heurystyk przeszukiwania [Holzmann 85; West 86, 89; LCL 87]. Metodami stosującymi się do większych systemów (a także modeli uwzględniających czas) są metody przeszukiwania losowego lub metody symulacyjne [Shaw 92; RS 94; Sacha 95].

Racjonalność zastosowania konkretnego typu algorytmu zależy od wielkości przestrzeni stanów badanego systemu. W [Holzmann 91] podano przykładowe liczby:

- pełny przegląd – dla około 10^5 stanów
- przeszukiwanie częściowe – systemy do 10^8 stanów
- przeszukiwanie losowe – dla większych systemów

Wraz z rozwojem sprzętu komputerowego liczby te mogą się zwiększyć, np.: o jeden lub dwa rzędy wielkości, ale nie zmieni to faktu, że w przypadku dużych systemów rozmiar przestrzeni stanów będzie zawsze uniemożliwiał jej pełne przebadanie.

Przedstawione w dalszych rozdziałach pracy algorytmy weryfikacji oparte są wyłącznie na zupełnym przeglądzie przestrzeni stanów. Naturalnym jest, że możliwości ich zastosowania będą poddane ogólnym ograniczeniom wynikającym ze złożoności obliczeniowej rozważanych problemów.

Opiszemy krótko podstawowe strategie ograniczania wpływu eksplozji przestrzeni stanów na jakość analizy systemów. Ich zastosowanie może w przyszłości pozwolić na wykorzystanie omówionych w pracy metod weryfikacji do częściowego badania własności większych systemów.

Algorytmy przeglądu zupełnego zbioru stanów osiągalnych dążą do generacji skończonego zbioru semiobliczeń badanego procesu rozpoczynających się w danym stanie początkowym. W zależności od algorytmu zapisywane są wybrane informacje o obliczonych stanach i wykonanych przejściach.

Typowy algorytm wyznaczania zbioru stanów osiągalnych przeszukuje w głąb drzewo semiobliczeń rozpoczynających się od początkowego semiobliczenia $sc = \langle s_0 \rangle$. Osiągnięte stany dodawane są do pewnego zbioru S_r. Jeśli stan wyznaczony jako następny spełnia $s \in S_r$ lub należy do zbioru stanów bieżącego semiobliczenia $Z(sc)$, oznacza to, że dalsza analiza jego następników została już przeprowadzona lub będzie dokonana później. W takim przypadku ścieżka przeszukiwania nie jest dalej kontynuowana.

Poniżej przedstawiony jest pseudokod typowej rekurencyjnej procedury implementującej algorytm przeszukiwania w głąb drzewa stanów procesu..

Podstaw $sc = \langle s_0 \rangle$, $S_r = \{ s_0 \}$

```
procedure explore;
begin
      if sc = ⟨ ⟩ then STOP;
      podstaw pod s ostatni stan należący do semiobliczenia sc ;
      dodaj s do S_r ;
      wyznacz zbiór T(s) wszystkich następników s ;
      for all s' ∈ T(s) do
            sprawdź poprawność stanu s' lub przejścia (s , s')
            if s' ∉ Z(sc) ∧ s' ∉ S_r then
            begin
                  dodaj s' do sc ;
                  explore;
            end;
      end;
      usuń ostatni element semiobliczenia sc ;
end;
```

Efektywność algorytmu zależy od dostępnej pamięci M, wielkości przestrzeni stanów badanego systemu $|S|$ i rozmiarów reprezentacji stanu $size(s)$. W przypadku, kiedy $M / size(s) < |S|$ algorytm przeglądu zupełnego pozwala pokrywa pokrycie pewnego podzbioru S_r przestrzeni stanów. Przebadanymi stanami będą te, które wybrane zostaną

jako pierwsze. W typowych sytuacjach będą należały do pewnej gałęzi procesu poprzedzonej długim nierozgałęzionym ciągiem stanów. Pokrycie zbiorów stanu procesu przez wyznaczony zbiór S_r jest więc bardzo nieregularne i potencjalnie wiele stanów i przejść kryjących błędy może zostać pominięte.

Lepsze pod tym względem są efekty zastosowania algorytmów sterujących częściowym przeglądem stanów, których celem jest zapewnienie większej regularności pokrycia. Ich ideą jest zastąpienie przeglądu wszystkich następników $T(s)$ badanego stanu s jedynie pewnym jego podzbiorem. W przypadku metod przeszukiwania losowego podzbiór ten jest jednoelementowy.

Do metod przeglądu częściowego należą:

- ograniczenia głębokości przeszukiwania (długości rozważanych semiobliczeń)
- heurystyki poszukiwania blokowania, np.: faworyzujące operacje odbioru komunikatów dla modeli asynchronicznej komunikacji
- heurystyki wyboru przejść oparte na funkcji kosztów obliczanej dla danych stanów
- przeszukiwanie probabilistyczne – oparte na przydziale poszczególnym przejściom prawdopodobieństw ich wykonania
- metody częściowego uporządkowania bazujące na priorytetach przydzielanych procesom składowym występującym w modelu. Ich zadaniem jest ograniczenie przeplotu pomiędzy procesami. W zależności od wybranej heurystyki, po wykonaniu każdego przejścia priorytety procesów są zmniejszane (system zachowuje się bardziej sprawiedliwie) lub zwiększane.
- strategie wyboru losowego analizowanego podzbioru zbioru następnych stanów.

Opisane powyżej metody opierają się na ograniczeniu liczby rozważanych przejść w badanym procesie. Drugim kierunkiem zwiększenia efektywności algorytmów wykorzystujących analizę własności zbioru stanów osiągalnych jest zmniejszenie wielkości danych reprezentujących stany przechowywanego w wyznaczanym zbiorze S_r. Na takim założeniu oparta została konstrukcja zaproponowanego przez G. Holzmanna [Holzmann 88, 91, 95; HP 89] algorytmu *supertrace*.

Budowa algorytmu opiera się na prostym pomyśle – jeżeli do przechowywania stanów w zbiorze S_r dysponujemy pamięcią o rozmiarze M i używamy reprezentacji stanu s_r wielkości $size(s_r)$, wówczas największą liczbę stanów zdołamy umieścić w pamięci, jeżeli $size(s_r) = 1$, czyli gdy każdy osiągnięty stan będzie reprezentowany przez jeden bit. Możemy wówczas pokryć $8 \cdot M$ stanów przestrzeni S.

Zbiór stanów osiągniętych S_r traktowany jest więc jak tablica bitowa; zapalenie bitu oznacza, że stan został już osiągnięty. Do odwzorowania stanu s w indeks tablicy ze zbioru $\{1,\ldots, 8 \cdot M\}$ używana jest funkcja mieszająca (ang. *hash function*).

Oczywiście, odwzorowanie takie jest niejednoznaczne i jakość pokrycia (mierzonego jako $|S_r| / |S|$) zależy od postaci funkcji mieszającej. Fakt wyznaczenia w trakcie analizy stanu s, któremu odpowiada zapalony bit w tablicy traktowany jest jako napotkanie stanu, którego następniki były już wcześniej analizowane. Tak być jednak nie musi, ze względu na niejednoznaczność funkcji mieszającej. Kilkukrotne uruchomienie algorytmu z różnymi funkcjami mieszającymi zwiększa jakość pokrycia. Efektywność algorytmu supertrace jest uzasadniana empirycznie. Niewątpliwie uzależniona jest przede wszystkim od doboru funkcji mieszających i stopnia ich dostosowania do konkretnego systemu.

Algorytm supertrace znalazł zastosowanie w pakiecie SPIN [Holzmann 97]. Pewna jego odmiana stosowana jest także w komercyjnym pakiecie Telelogic SDT [Telelogic 97].

3. Ogólna charakterystyka przyjętych rozwiązań

Celem tego rozdziału jest przedstawienie na wybranych przykładach sposobu formułowania zagadnienia poprawności względnej. W rozważanych specyfikacjach relacja poprawności będzie definiowana jako inkluzja zbiorów ścieżek procesów etykietowanych przekształconych przez ścieżkową funkcję obserwacji.

Następnie zostaną wprowadzone i omówione definicje liniowej funkcji obserwacji stanowiącej uogólnienie funkcji ścieżkowej oraz funkcji niehomomorficznej.

3.1 Proces etykietowany

Zgodnie z wcześniejszą zapowiedzią, głównym przedmiotem zainteresowania omawianego podejścia są akcje i zależności pomiędzy nimi. Akcje utożsamiane są ze zmianą stanu procesu. Akcja prosta związana jest z wykonaniem przejścia, akcja złożona z wykonaniem pewnego zbioru przejść. Przy opisie akcji, a zwłaszcza w celu prezentacji zależności pomiędzy nimi na przykładach, wygodnie posługiwać się pojęciami procesu etykietowanego i ścieżki.

Def. 3-1 Proces etykietowany

Procesem etykietowanym nazywamy trójkę $P_L = (P, A, \sigma)$, gdzie $P = (S, B, F, T)$ jest procesem, A zbiorem etykiet (alfabetem) równolicznym z relacją opisującą przejścia procesu T, natomiast σ jest wzajemnie jednoznaczną funkcją przypisującą łukom symbole etykiet: $\sigma: T \rightarrow A$.

■

Wprowadzona definicja procesu etykietowanego różni się od definicji przyjętej w [Szmuc 97b] oraz [Hoare 85; Milner 89] dodatkowym wymaganiem, aby każdemu przejściu była przypisana unikalna etykieta. W związku z tym specyfikacje dyskutowane w [GG 98] (Q i Q' na Rys 2.3) uznawane są za niepoprawne. Nie jest też wprowadzona etykieta pusta lub też wyróżnione przejście ukryte (jak τ w CCS).

Funkcja pozwalająca na przypisanie tej samej etykiety różnym przejściom (w tym etykiety oznaczającej akcję ukrytą) traktowana będzie jako szczególny przypadek odwzorowania służącego do zdefiniowania funkcji obserwacji. W konsekwencji, takie odwzorowania będą traktowane zazwyczaj jako zapis wymagań poprawnościowych pomiędzy dwoma różnymi specyfikacjami.

Niech $S(A)$ oznacza zbiór wszystkich ciągów utworzonych z symboli alfabetu A. Symbolem $\langle\ \rangle$ oznaczana jest ciąg pusty.

Funkcja *head*: $S(A) \rightarrow S(A)$ zdefiniowana jest jako:

$$head(\langle \alpha_1, \alpha_2, \ldots \alpha_n \rangle) = \langle \alpha_1 \rangle$$

$$head(\langle\ \rangle) = \langle\ \rangle$$

Funkcja *tail*: $S(A) \rightarrow S(A)$ zdefiniowana jest jako:

$$tail(\langle \alpha_1, \alpha_2, \ldots \alpha_n \rangle) = \langle \alpha_2, \ldots \alpha_n \rangle$$

$$tail(\langle\ \rangle) = \langle\ \rangle$$

Operator konkatenacji • jest funkcją przekształcającą $S(A) \times S(A) \rightarrow S(A)$ zdefiniowaną jako:

$$\langle \alpha_1, \alpha_2, \ldots \alpha_k \rangle \bullet \langle \alpha_{k+1}, \alpha_{k+2}, \ldots \alpha_n \rangle = \langle \alpha_1, \alpha_2, \ldots \alpha_k, \alpha_{k+1}, \alpha_{k+2}, \ldots \alpha_n \rangle$$

Oznaczmy przez $\# t$ liczbę elementów ciągu t; oznaczmy przez $t{\downarrow}\alpha$ liczbę wystąpień symbolu α w ciągu t.

Def. 3-2 Ścieżka

1. Ścieżką procesu etykietowanego P_L nazywamy n–elementowy ciąg etykiet $t = \langle \alpha_1, \alpha_2, \ldots \alpha_n \rangle$ spełniający dla $n > 2$ następujący warunek: $\forall i < n . \left(\sigma^{-1}(\alpha_i) = (s_{k_1}, s_{k_2}) \wedge \sigma^{-1}(\alpha_{i+1}) = (s_{k_3}, s_{k_4}) \Rightarrow s_{k_2} = s_{k_3} \right)$.
2. Ścieżką początkową procesu P_L nazywamy ścieżkę pustą lub ścieżkę postaci $t = \langle \alpha_1, \alpha_2, \ldots \alpha_n \rangle$, spełniającą $\sigma^{-1}(\alpha_1) = (s_{k_1}, s_{k_2}) \wedge s_{k_1} \in B$.
3. Ścieżką końcową procesu P_L nazywamy ścieżkę postaci $t = \langle \alpha_1, \alpha_2, \ldots \alpha_n \rangle$, spełniającą $\sigma^{-1}(\alpha_n) = (s_{k_1}, s_{k_2}) \wedge s_{k_2} \in F$

■

Oznaczmy przez $L(P)$ zbiór wszystkich ścieżek procesu. Funkcje *head* oraz *tail* przekształcają $L(P) \rightarrow L(P)$, natomiast operator konkatenacji w ogólnym przypadku przekształca $L(P) \times L(P) \rightarrow S(A)$.

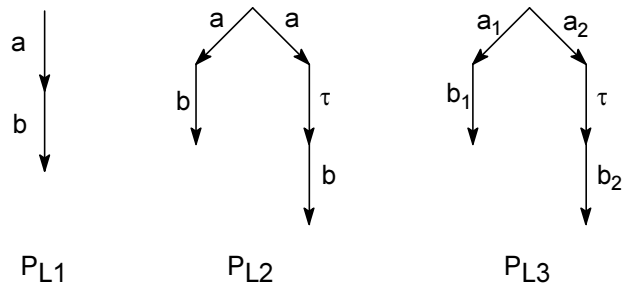
Oznaczmy przez $L_b(P)$ zbiór wszystkich scieżek początkowych procesu natomiast przez $L_f(P)$ zbiór wszystkich scieżek końcowych.

Rozważmy procesy przedstawione na Rys. 3-1. Proces P_{L1} jest procesem etykietowanym w sensie Def. 3-1, natomiast proces P_{L2} jest z nią niezgodny. Proces P_{L3} jest poprawioną wersją procesu P_{L2}. Symbolem τ oznaczono pewną akcję ukrytą.

Zbiór ścieżek początkowych procesu P_{L1} jest postaci: $L_b(P_{L1}) = \{\langle\,\rangle, \langle a \rangle, \langle a, b \rangle\}$,

natomiast dla procesu P_{L2} jest on równy: $L_b(P_{L2}) = \{\langle\,\rangle, \langle a \rangle, \langle a, b \rangle, \langle a, \tau \rangle, \langle a, \tau, b \rangle\}$.

W zależności od przyjętych założeń zbiory ścieżek i procesy mogą zostać uznane za równoważne, jak w CSP, lub też stwierdza się ich odmienność jak w CCS (ze względu na to, że po wykonaniu akcji oznaczonej symbolem *a* w procesie P_{L1} możliwe jest wykonanie jedynie akcji $\langle b \rangle$ natomiast w procesie P_{L2} akcji $\langle b \rangle$ lub $\langle \tau, b \rangle$).



Rys. 3-1 Przykłady procesów etykietowanych

W zaproponowanym podejściu przyjmuje się odmienne założenia. Modelem semantycznym procesów są zbiory ścieżek początkowych. Zbiór ścieżek początkowych procesu P_{L3} jest postaci $L_b(P_{L3}) = \{\langle\,\rangle, \langle a_1\rangle, \langle a_1, b_1\rangle, \langle a_2\rangle, \langle a_2, \tau\rangle, \langle a_2, \tau, b_2\rangle\}$. Utożsamienie zbiorów $L_b(P_{L3})$ oraz $L_b(P_{L1})$ jest zadaniem obserwatora, który posługując się zdefiniowaną przez siebie funkcją obserwacji odwzorowuje zbiór ścieżek w zbiór ciągów symboli wybranego alfabetu.

3.2 Pojęcie obserwacji i jej rola w specyfikacji wymagań

W podrozdziale zostanie zdefiniowana ścieżkowa funkcja obserwacji oraz wyjaśniona na przykładach koncepcja stosowanego w pracy modelu poprawności względnej procesów.

3.2.1 Ścieżkowa funkcja obserwacji

Niech f_{12} będzie funkcją częściową odwzorowującą zbiór etykiet A_1 w A_2. Oznaczmy przez *Dom* f_{12} dziedzinę funkcji f_{12}. Przez analogię do relacji kryterialnej funkcja f_{12} nazywana będzie funkcją kryterialną.

Def. 3-3 Ścieżkowa funkcja obserwacji π_t

Dla danej funkcji kryterialnej $f_{12} : A_1 \to A_2$, $P_L = (P, A_1, \sigma)$ oraz A_2 zdefiniujemy funkcję π_t odwzorowującą $S(A_1) \to S(A_2)$. Funkcja ta zostanie zdefiniowana rekurencyjnie.

Niech $t = \langle \alpha_1, \alpha_2, \ldots \alpha_n\rangle \in S(A_1)$.

1. $\pi_t(t) = \pi_t(head(t)) \bullet \pi_t(tail(t))$ jeśli $\# t \geq 2$ ($\bullet$ oznacza operator konkatenacji)
2. $\pi_t(\langle\,\rangle) = \langle\,\rangle$
3.a. $\pi_t(\langle a\rangle) = \langle f_{12}(a)\rangle$ jeśli $a \in Dom\, f_{12}$
3.b. $\pi_t(\langle a\rangle) = \langle\,\rangle$ jeśli $a \notin Dom\, f_{12}$

■

Działanie ścieżkowej funkcji obserwacji π_t polega na zastąpieniu w obserwowanej ścieżce symboli alfabetu A_1 odpowiednimi symbolami alfabetu A_2 oraz usunięciu symboli nie należacych do dziedziny funkcji f_{12}.

Definiując dla obserwowanego procesu P_{L3} (Rys. 3-1) funkcję f_{12}: $\{a_1, a_2, b_1, b_2\} \to \{a, b\}$ jako:

$f_{12}(a_1) = f_{12}(a_2) = a$ oraz

$f_{12}(b_1) = f_{12}(b_2) = b$,

otrzymamy wynik $\pi_t(L_b(P_{L3})) = \{\langle\,\rangle, \langle a\rangle, \langle a, b\rangle\}$, a więc równość zbioru obserwowanych ścieżek ze zbiorem $L_b(P_{L1})$ ścieżek początkowych procesu P_{L1}.

Działanie ścieżkowej funkcji obserwacji π_t podobne jest do operatorów zmiany etykiet oraz restrykcji zdefiniowanych w algebrach procesów CSP i CCS. W obu tych formalizmach podstawowym celem zastosowań operatorów działających na etykietach była zmiana struktury specyfikacji. Tutaj zakładamy, że zastosowaniem funkcji π_t jest jedynie zmiana obserwowanych akcji.

Ważną zaletą ścieżkowej funkcji obserwacji π_t jest to, że jest przekształceniem homomorficznym ze względu na operator konkatenacji „$\bullet$” ciągów symboli. Istotnie, bezpośrednio z definicji wynika, że jeżeli t_1 oraz t_2 są ciągami symboli alfabetu A_1, wówczas spełnione jest: $\pi_t(t_1 \bullet t_2) = \pi_t(t_1) \bullet \pi_t(t_2)$. Własność ta może znaleźć zastosowanie

przy dowodzeniu poprawności procesów. W szczególności, na podobnych rozwiązaniach opiera się koncepcja dowodzenia poprawności w formalizmie CSP.

3.2.2 Specyfikacja wymagań i poprawność

Podobnie jak dla algebraicznych metod badania poprawności względnej (por. 2.6) zakłada się, że wymagania specyfikowane są w formie procesu kryterialnego. Podstawową różnicą pomiędzy oryginalną postacią zagadnienia poprawności względnej, a przedstawioną tu konstrukcją jest sposób ustalania odwzorowania pomiędzy dwoma warstwami specyfikacji: weryfikowaną i kryterialną. Dla oryginalnego sformułowania problemu była to asocjacja w dziedzinie stanów zadana w postaci relacji kryterialnej. W pracy zakłada się, że asocjacja ta ustalana jest dla akcji.

Naturalną metodą powiązania dwóch warstw specyfikacji jest oparcie się na dobranych do problemu funkcjach obserwacji, o których zakładamy, że w logiczny sposób odwzorowują zbiór ścieżek weryfikowanego procesu w zbiór ciągów symboli alfabetu procesu kryterialnego.

Idea poprawności może być rozpatrywana w kategoriach eksperymentu. Obserwator śledzi zachowanie procesu weryfikowanego i posługując się funkcją obserwacji dokonuje interpretacji ścieżki kolejno wykonywanych akcji zamieniając ją na ciąg akcji należących do alfabetu procesu kryterialnego. Dopóki otrzymany ciąg wyjściowy jest ścieżką początkową procesu kryterialnego, obserwator może twierdzić, że badany proces zachowuje się poprawnie. Eksperyment ten może trwać w nieskończoność lub do momentu, kiedy analizowana ścieżka zostanie zakończona wraz z osiągnięciem stanu końcowego. W takim przypadku obserwator sprawdza dodatkowe wymaganie, czy zaobserwowany ciąg symboli stanowi również ścieżkę końcową procesu kryterialnego.

Istotnym założeniem jest, obserwator nie wie *a priori*, jaki jest zbiór ścieżek badanego procesu, ale posługuje się jedynie funkcją odwzorowującą w siebie alfabety dwóch procesów, na podstawie której jest w stanie zdefiniować ogólnie funkcję obserwacji.

Uwzględniając powyższe założenia, pierwsze przybliżenie idei poprawności może być sformułowane następująco:

Def. 3-4 Częściowa poprawność π_t

Dane są dwa procesy etykietowane:

1. weryfikowany $P_L = (P, A, \sigma)$
2. kryterialny $P_L' = (P', A', \sigma')$

Dana jest funkcja częściowa $f_{12} : A \rightarrow A'$ (funkcja kryterialna). Niech π_t oznacza ścieżkową funkcję obserwacji generowaną przez funkcję f_{12} zgodnie z Def. 3-3.

Proces P_L jest częściowo poprawny w sensie kryterium (π_t, P_L') wtw. jeśli:

1. $\forall t \in L_b(P_L) \,.\, \pi_t(t) \in L_b(P_L')$
2. $\forall t \in L_f(P_L) \,.\, \pi_t(t) \in L_f(P_L')$

■

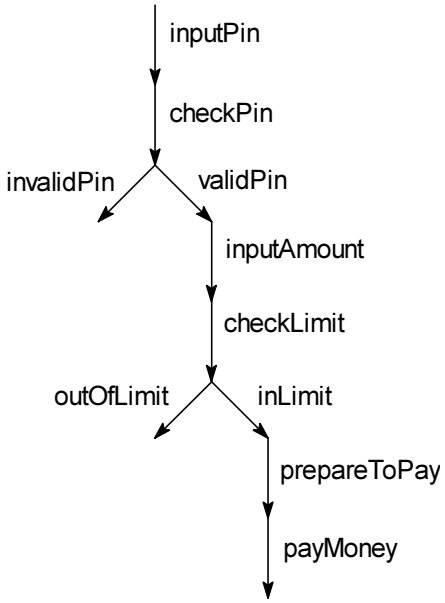
Pomijając różnice związane z dziedzinami odwzorowania łączącymi dwie specyfikacje, powyższy model poprawności jest w ogólnym przypadku słabszy niż oryginalny sformułowany w pracach [Szmuc 88, 89a, 89b, 91, 92]. Istotną różnicą jest fakt, że w tym modelu nie możemy powiązać z wykonaniem akcji po stronie procesu weryfikowanego

obserwacji, której wynikiem byłby zbiór akcji po stronie procesu kryterialnego odzwierciedlających równolegle przetwarzane własności. Wynika to z założenia, że f_{12} jest funkcją, a nie relacją.

Przykład 3-1

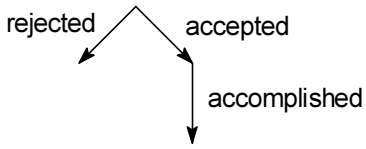
Rozważmy bardzo uproszczony proces opisujący zachowanie bankomatu (Rys. 3-2), którego inspiracją były przykłady zamieszczone w [BH 93]. Użytkownik bankomatu (klient) posługuje się kartą, na której zakodowany jest jej numer identyfikacyjny PIN a także limit wypłat przypadający na kartę. Operacja wypłaty pieniędzy powinna zostać dokonana, tylko wtedy, gdy klient zna kod identyfikacyjny karty oraz żądana kwota jest mniejsza niż limit.

Klient po wsunięciu karty do bankomatu podaje na klawiaturze kod PIN (akcja *inputPin*). Kod ten jest sprawdzany w wyniku akcji *checkPin* wykonywanej np.: przez inny proces. Niepoprawny wynik weryfikacji kodu symbolizowany jest przez akcję *invalidPin*; wynik poprawny przez akcję *validPin*. Dalej użytkownik wprowadza kwotę, którą chciałby wypłacić (akcja *inputAmount*). Symbole *outOfLimit* oraz *inLimit* odpowiadają otrzymaniu odpowiednio: niepoprawnego i poprawnego rezultatu akcji *checkLimit*. Podczas akcji *payMoney* następuje właściwa wypłata; jest ona poprzedzana akcją *prepareToPay*.



Rys. 3-2 Proces opisujacy zachowanie bankomatu

Dla tak zdefiniowanego procesu określimy wymagania dotyczące jego zachowania. Pragniemy wykazać, że proces opisujący działanie bankomatu zachowuje się tak jak pewien bardzo ogólny proces wykonujący pewną akcję warunkowo. Symbol *rejected* opisuje działanie, w wyniku którego stwierdzono, że warunek wykonania akcji nie został spełniony; symbolem *accepted* opisujemy sprawdzenie warunku, natomiast przejście etykietowane symbolem *accomplished* związane jest z wykonaniem akcji obłożonej warunkiem. Proces opisujący wymagania przedstawiono na Rys. 3-3.



Rys. 3-3 Proces opisujący wymagania

Dla powyżej przedstawionych procesów zdefiniujemy funkcję częściową f_{12}, której zadaniem jest odwzorowanie alfabetów procesów.

$f_{12}(invalidPin) = rejected$

$f_{12}(outOfLimit) = rejected$

$f_{12}(inLimit) = accepted$

$f_{12}(payMoney) = accomplished$

Łatwo sprawdzić, poprzez analizę ścieżek procesu weryfikowanego, że proces ten jest poprawny. Przykładowo: dla wybranych ścieżek procesu weryfikowanego zachodzi następujące odwzorowanie:

$\pi_t(\langle inputPin, checkPin, invalidPin\rangle) = \langle rejected\rangle$

$\pi_t(\langle inputPin, checkPin, validPin, inputAmount, checkAmount, outOfLimit\rangle) = \langle rejected\rangle$

$\pi_t(\langle inputPin, checkPin, validPin, inputAmount, checkAmount, inLimit\rangle) = \langle accepted\rangle$

$\pi_t(\langle inputPin, checkPin, validPin, inputAmount, checkAmount, inLimit, prepareToPay\rangle) = \langle accepted\rangle$

$\pi_t(\langle inputPin, checkPin, validPin, inputAmount, checkAmount, inLimit, prepareToPay, payMoney\rangle) = \langle accepted, accomplished\rangle$

3.3 Liniowa funkcja obserwacji

3.3.1 Wprowadzenie

Rozważmy inny typ funkcji kryterialnej. Niech g_{12} będzie funkcją częściową, która odwzorowuje zbiór symboli alfabetu A_1 w rodzinę zbiorów symboli alfabetu A_2.

$$g_{12} : A_1 \rightarrow 2^{A_2}$$

Zastosowanie takiej funkcji pozwala na powiązanie z obserwowanym przejściem pewnego zbioru równolegle wykonywanych przejść. (W typowym przypadku zbiór ten będzie jednak jednoelementowy.)

Dla tak zdefiniowanej funkcji kryterialnej można próbować stworzyć model odwzorowania opisującego obserwację, który ścieżkę będzie przekształcał w zbiór ścieżek. Jednakże trudno takie odwzorowanie opisać w postaci bardziej ogólnej, jako odwzorowanie ciągu symboli w zbiór ciągów symboli, ponieważ ponieważ dla danego przejścia złożonego $\alpha \in Dom\, g_{12}$ spełniającego $|\, g_{12}(\alpha)\,| > 1$ nie można stwierdzić, do której z wyznaczonych wcześniej ścieżek $S \subset S(A_2)$ należy dołączać wybrany symbol $\beta \in |\, g_{12}(\alpha)\,|$. Taka decyzja byłaby możliwa przy założeniu, że wyznaczane zbiory należą zawsze do zbioru ścieżek procesu kryterialnego, ale w ten sposób definicja funkcji obserwacji straciłaby na ogólności.

Ta trudność była głównym powodem zmiany sposobu opisu w stosunku do modelu używanego w oryginalnym sformułowaniu metod badania poprawności względnej. Przyjęto założenie, że stosowaną reprezentacją ścieżki będzie wektor opisujący wykonane przejścia, natomiast oba procesy: weryfikowany i kryterialny będą modelowane w sposób, który umożliwi przetwarzanie wektorów. Alternatywną reprezentacją mógłby być wielozbiór [Jensen 91], ale ze względu na prostotę zapisu wielu formuł użycie wektorów wydawało się bardziej atrakcyjne.

Reprezentacja ścieżki w postaci wektora prowadzi w oczywisty sposób do utraty informacji. Strata ta jednak nie jest tak istotna, jeśli zakładamy, że zachowanie procesu weryfikowanego będzie analizowane w systematyczny sposób polegający na generacji ciągów wektorów różniących się jedynie pojedynczymi akcjami. W takim przypadku ścieżka może zostać odtworzona poprzez analizę danego ciągu.

3.3.2 Definicja wektorowej funkcji obserwacji π_{Lin}

Dla danych dwóch alfabetów A_1 oraz A_2 niech $x_1 \in X(A_1) = R^{n_1}$ oznacza wektor w przestrzeni n_1 wymiarowej, gdzie $n_1 = |A_1|$, natomiast $x_2 \in X(A_2) = R^{n_2}$ oznacza wektor w przestrzeni n_2 wymiarowej, gdzie $n_2 = |A_2|$.

Załóżmy, że symbole obu alfabetów, są uporządkowane, tzn. istnieje funkcja $J_1: A_1 \to \{1,...,n_1\}$ oraz $J_2: A_2 \to \{1,...,n_2\}$.

Współrzędną danego wektora $x_{J(\alpha)}$ związaną z danym symbolem $\alpha \in A_i$ oznaczać będziemy na ogół x_α z pominięciem oznaczenia funkcji porządkującej.

Dla danego ciągu t symboli należących do alfabetu A jego reprezentacja w postaci wektora $x(t)$ jest określona jako: $x_\alpha(t) = t\downarrow\alpha$, gdzie $\alpha \in A$.

Def. 3-5 Wektorowa liniowa funkcja obserwacji π_{Lin}

Niech g_{12} będzie funkcją częściową $g_{12} : A_1 \to 2^{A_2}$. Zdefiniujemy macierz L o wymiarach $n_2 \times n_1$. Jej elementy oznaczane przez $L(i, j)$ przyjmują wartości:

$$L(J_2(\beta), J_1(\alpha)) = \begin{cases} 1 & \text{jeżeli } \beta \in g_{12}(\alpha) \\ 0 & \text{w p.p.} \end{cases}$$

Zdefiniujemy funkcję π_{Lin} odwzorowującą $X(A_1) \to X(A_2)$. Załóżmy, że $x_1 \in X(A_1)$ oraz $x_2 \in X(A_2)$.

Wektor x_2 spełniający $x_2 = \pi_{Lin}(x_1)$ określony jest jako: $x_2 = L\, x_1$

■

Na Rys. 3-4 pokazano macierz L liniowej funkcji obserwacji dla przykładu 4.1. Kolumny macierzy etykietowane są symbolami alfabetu procesu opisującego zachowanie bankomatu, natomiast wiersze macierzy odpowiadają etykietom procesu kryterialnego.

Zauważmy, że akcjom ukrytym odpowiadają zerowe kolumny macierzy (np.: *inputPin*, *checkPin*). Akcja *rejected* może zostać zaobserwowana w wyniku wykonania *invalidPin* lub *outOfLimit*). W przypadku odwzorowania danej akcji w zbiór akcji równoległych, w odpowiadającej jej kolumnie będzie znajdować się kilka wyrazów niezerowych.

	inputPin	*checkPin*	*invalidPin*	*validPin*	*inputAmount*	*checkLimit*	*outOfLimit*	*inLimit*	*prepareToPay*	*payMoney*
rejected	0	0	**1**	0	0	0	**1**	0	0	0
accepted	0	0	0	0	0	0	0	**1**	0	0
accomplished	0	0	0	0	0	0	0	0	0	**1**

Rys. 3-4 Macierz L liniowej funkcji obserwacji

Ważną cechą funkcji obserwacji π_{Lin} jest jej homomorficzny charakter. Zauważmy, że jeżeli t_1, t_2 są pewnymi ciągami symboli alfabetu A_1, wówczas zachodzi:

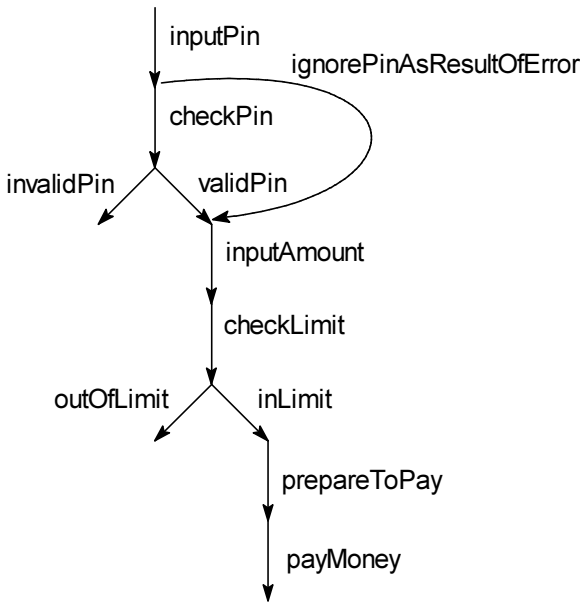
$$\pi_{Lin}(x(t_1 \bullet t_2)) = \pi_{Lin}(x(t_1)) + \pi_{Lin}(x(t_2)).$$

3.4 Niehomomorficzna funkcja obserwacji

3.4.1 Wprowadzenie

Przykład 3-2

Rozważmy zmodyfikowany proces opisujący działanie bankomatu (Rys. 3-5), w którym w wyniku błędu projektanta lub programisty wprowadzono niepoprawne przejście oznaczone etykietą *ignorePinAsResultOfError*. Analiza poprawności rozważanego procesu przy takich samych jak wcześniej definicjach procesu kryterialnego i funkcji odwzorowującej w siebie dwa alfabety prowadzi do stwierdzenia, że proces ten jest również poprawny.



Rys. 3-5 Zmodyfikowany proces opisujący błędnie działający bankomat

W szczególności spełnione jest:

$\pi_t(\langle inputPin, ignorePinAsResultOfError, inputAmount, checkAmount, inLimit \rangle) = \langle accepted \rangle$

Rezultat ten, całkowicie sprzeczny z intuicją poprawnego zachowania bankomatu prowadzi do propozycji dwóch alternatywnych rozwiązań:

1. zmiany definicji procesu kryterialnego i funkcji f_{12} odwzorowującej alfabety procesów tak, aby uwidocznić fakt, że najpierw powinna nastąpić operacja sprawdzenia kodu, a następnie limitu wypłat;
2. prowadzenia bardziej złożonych obserwacji, które swój wynik uzależniały będą nie tylko od aktualnie wybranej akcji ale także od akcji wykonywanych wcześniej.

Rozwiązanie pierwsze wydaje się niewygodne. Zmusza ono do zastąpienia abstrakcyjnych wymagań postacią bardziej szczegółową, zbliżoną w wyniku kolejnych przekształceń do struktury procesu weryfikowanego. Prowadzi to na ogół do spłaszczenia i zatarcia hierarchii specyfikacji, w której wymagania stają się procesem definiującym szczegółowo zachowanie całego systemu, natomiast opis kolejnej warstwy (np.: projektu oprogramowania) jest opisem niewiele bardziej złożonym. Wprowadzenie dodatkowych szczegółów do opisu wymagań tylko w celu ich weryfikacji, jest także niewłaściwe, ponieważ przeczy zasadzie trwałości rezultatów analizy wymagań.

Z punktu widzenia zastosowań w cyklu wytwarzania oprogramowania bardziej naturalne jest przyjęcie założenia, że proces opisujący wymagania ma postać ustaloną, natomiast projektant konstruując następną warstwę specyfikacji definiuje także sposób odwzorowania jej zachowania stosowny do podjętych decyzji projektowych. (W pewnym sensie odwzorowanie to jest również częścią specyfikacji.)

Powyższe argumenty skłoniły autora do oparcia poprawności na bardziej złożonych obserwacjach, które uwzględniają historię wykonania procesu weryfikowanego.

Dla omawianych przykładów pragnęlibyśmy jawnie zdefiniować, że akcja *accepted* procesu kryterialnego może zostać zaobserwowana w wyniku wcześniejszego wykonania akcji oznaczonych jako *validPin* i *inLimit* procesu weryfikowanego.

Dla dwóch alfabetów A_1 oraz A_2 zdefiniujmy funkcję kryterialną h_{12} jako funkcję częściową odwzorowującą $2^{A_1} \to A_2$.

Przy takim założeniu możemy wyspecyfikować wymaganie, aby warunkiem zarejestrowania przez obserwatora danego symbolu α alfabetu A_2 było wcześniejsze pojawienie się w analizowanej ścieżce wszystkich symboli z pewnego zbioru $B \in h_{12}^{-1}(\alpha)$.

Dla omawianych wcześniej przykładów (Przykład 3-1 oraz Przykład 3-2) funkcja h_{12} może zostać zdefiniowana następująco:

$h_{12}(\{\mathit{invalidPin}\}) = \mathit{rejected}$

$h_{12}(\{\mathit{outOfLimit}\}) = \mathit{rejected}$

$h_{12}(\{\mathit{inLimit}, \mathit{validPin}\}) = \mathit{accepted}$

$h_{12}(\{\mathit{prepareToPay}, \mathit{payMoney}\}) = \mathit{accomplished}$

Jak łatwo można sprawdzić, taka definicja chroni obserwatora przed zarejestrowaniem akcji *accepted* dla ścieżki postaci:

$\langle \mathit{inputPin}, \mathit{ignorePinAsResultOfError}, \mathit{inputAmount}, \mathit{checkAmount}, \mathit{inLimit} \rangle$

ponieważ nie wystąpił w niej symbol *validPin*.

Podobnie, jak dla funkcji kryterialnej g_{12}, kłopotliwą cechą funkcji kryterialnej h_{12} jest fakt, że trudno na jej podstawie podać konstruktywną metodę definiowania funkcji

obserwacji odwzorowującej w siebie ciągi symboli dwóch alfabetów. Dla procesów cyklicznych symbole akcji z danego zbioru mogą powtarzać się w ścieżce wielokrotnie. Podanie równań definiujących obserwowany ciąg wyjściowy zmusza do analizy liczby wystąpień zbiorów symboli w rozpatrywanej ścieżce.

3.4.2 Definicja wektorowej funkcji obserwacji π_{Min}

Definicja funkcji π_{Min} zostanie wprowadzona przy złożeniu, że odwzorowanie h_{12} jest różnowartościowe.

Def. 3-6 Wektorowa funkcja obserwacji π_{Min}

Niech h_{12} będzie różnowartościową funkcją częściową odwzorowującą $2^{A_1} \to A_2$; niech, $x_1 \in X(A_1) = R^{n_1}$ oraz $x_2 \in X(A_2) = R^{n_2}$ będą wektorami przestrzeni określonych w 3.3.2.

Zdefiniujemy funkcję $\pi_{Min} : X(A_1) \to X(A_2)$.

Oznaczmy przez $x_i(\alpha)$ współrzędną wektora x_i związaną z symbolem α. Niech $D(\alpha) = \{B \subset A_1 \mid h_{12}(B) = \alpha\}$

Współrzędne wektora $x_2 = \pi_{Min}(x_1)$ spełniają:

$$x_2(\alpha) = \min\{ x_1(\beta) \mid \beta \in h_{12}^{-1}(\alpha)\}$$

■

Zgodnie z definicją – akcja α zostanie zarejestrowana dokładnie tyle razy, ile razy w ścieżce reprezentowanej przez wektor pojawią się wszystkie akcje ze zbioru odwzorowanego przez funkcję h_{12}.

Wektorowa funkcja obserwacji π_{Min} często definiowana będzie w postaci macierzy, której wiersze odpowiadają elementom dziedziny funkcji h_{12}.

Def. 3-7 Reprezentacja macierzowa funkcji π_{Min}

Podobnie jak przy określaniu macierzowej postaci liniowej funkcji obserwacji, załóżmy, że symbole obu alfabetów, są uporządkowane, tzn. istnieje funkcja $J_1: A_1 \to \{1,...,n_1\}$ oraz $J_2: A_2 \to \{1,...,n_2\}$.

Zdefiniujemy macierz M o wymiarach $n_2 \times n_1$, której elementy oznaczane będą przez $M(i, j)$.

$$M(J_2(\alpha), J_1(\beta)) = \begin{cases} 1 & \text{jeżeli } \beta \in h_{12}^{-1}(\alpha) \\ 0 & \text{w p.p.} \end{cases}$$

Współrzędne wektora $x_2 = \pi_{Min}(x_1)$ opisane są wówczas prostą formułą:

$$x_2(\alpha) = \min\{ x_1(\beta) \mid M(J_2(\alpha), J_1(\beta)) \neq 0\}$$

■

3.4.3 Definicja funkcji obserwacji π

Def. 3-8 Funkcja obserwacji π

Niech π_{Min} będzie odwzorowaniem przekształcającym przestrzeń $X \to P$ oraz π_{Lin} niech przekształca $P \to X'$. Złożenie tych funkcji

$$\pi = \pi_{Lin} \cdot \pi_{Min}$$

nazywane będzie funkcją obserwacji.

■

Przestrzeń $P = R^r$ nazywana będzie *przestrzenią liniowej obserwacji*.

Jeżeli wszystkie wiersze macierzy M są liniowo niezależne, wówczas funkcja obserwacji jest zapisana w postaci standardowej. Jeżeli wszystkie wiersze macierzy M zawierają dokładnie jeden element równy 1, wówczas funkcja obserwacji jest funkcją *liniową* (homomorficzną). Jeżeli obie macierze są macierzami jednostkowymi, wówczas funkcja obserwacji jest funkcją *identycznościową*.

Na Rys. 3-6 i Rys. 3-7 pokazano macierze opisujące funkcję obserwacji dla przykładu 4.2. Macierz M (Rys. 3-6) jest zapisem funkcji $h_{1P} : 2^{A_1} \rightarrow A_P$, gdzie A_P jest niejawnie zdefiniowanym alfabetem przestrzeni liniowej obserwacji P (tutaj A_P = {*badPin*, *badLimit*, *pinAndLimitOK*, *moneyPayed*}). Macierz L przekształca przestrzeń liniowej obserwacji w przestrzeń wektorów procesu kryterialnego zgodnie z odwzorowaniem $g_{P2} : A_P \rightarrow 2^{A_2}$

	inputPin	*ignorePinAsResultOfError*	*checkPin*	*invalidPin*	*validPin*	*inputAmount*	*checkLimit*	*outOfLimit*	*inLimit*	*prepareToPay*	*payMoney*
badPin	0	0	0	**1**	0	0	0	0	0	0	0
badLimit	0	0	0	0	0	0	0	**1**	0	0	0
pinAndLimitOk	0	0	0	0	**1**	0	0	0	**1**	0	0
moneyPayed	0	0	0	0	0	0	0	0	0	**1**	**1**

Rys. 3-6 Macierz M opisująca odwzorowanie π_{Min}

	badPin	*badLimit*	*pinAndLimitOk*	*moneyPayed*
rejected	**1**	**1**	0	0
accepted	0	0	**1**	0
accomplished	0	0	0	**1**

Rys. 3-7 Macierz L opisująca odwzorowanie π_{Lin}

3.4.4 **Zapis wektorowej funkcji obserwacji** π

W typowych przykładach pojawiających się w pracy specyfikacja funkcji obserwacji π będzie dokonywana na dwa sposoby:

1. Poprzez zapis bezpośredni współrzędnych wektorów lub z użyciem macierzy L i M.

2. W postaci wyrażenia z użyciem operatorów „∘” oraz „⊕” będącego zapisem funkcji kryterialnych.

Bezpośredni zapis współrzędnych (bliski definicji) dla przykładu 3.1 podany jest w postaci:

$$x(\mathit{rejected}) = x(\mathit{invalidPin}) + x(\mathit{outOfLimit})$$

$$x(\mathit{accepted}) = \min\{\ \mathrm{x}(\mathit{inLimit}), x(\mathit{validPin})\}$$

$$x(\mathit{accomplished}) = \min\{\ x(\mathit{prepareToPay}), x(\mathit{payMoney})\ \}$$

Jak łatwo zauważyć analizując definicję, wyrażenia typu $\min\{\ x(\mathit{invalidPin})\ \}$ zastąpiono po prostu wartością $x(\mathit{invalidPin})$.

Reprezentacja w postaci wyrażenia tej samej funkcji ma postać

$$\mathit{rejected} = \mathit{invalidPin} \oplus \mathit{outOfLimit}$$

$$\mathit{accepted} = \mathit{validPin} \circ \mathit{inLimit}$$

$$\mathit{accomplished} = \mathit{prepareToPay} \circ \mathit{payMoney}$$

Najczęściej przy specyfikacji wymagań preferowana będzie postać wyrażenia ponieważ najlepiej oddaje ona zależności logiczne pomiędzy akcjami. Operator „∘”, który powinien być czytany jako „i” może być interpretowany jako koniunkcja dla pewnej logiki wielowartościowej. Podobnie operator „⊕”, który powinien być czytany jako „lub” czy też „albo” może być interpretowany jako alternatywa. Konstruując wyrażenie zakładamy, że operator „∘” ma większy priorytet niż „⊕”. Oba operatory te są przemienne i łączne.

Standardową postacią zapisu jest alternatywa koniunkcji symboli, czyli:

$$\alpha_{2,i} = \alpha_{1,1} \circ \ldots \circ \alpha_{1,k} \oplus \ldots \oplus \alpha_{l,m1} \circ \ldots \circ \alpha_{1,mk}$$

Pojedynczy składnik odpowiada zbiorowi B, takiemu, że $h_{12}(B) = \alpha$.

3.5 **Zastosowanie funkcji obserwacji do specyfikacji i badania poprawności**

Sformułowanie i analiza zagadnienie poprawności wykorzystującego wektorową funkcję obserwacji jest centralnym motywem pracy. Jego idea podobna jest do rozwiązań opisanych w podrozdziale 3.2.2. Model poprawności zakłada istnienie obserwatora, który ciąg wektorów opisujący zachowanie procesu weryfikowanego $\langle x_0, x_1, x_2, \ldots\rangle$ tłumaczy na obserwowany ciąg wektorów $\langle \pi(x_0), \pi(x_2),\ \pi(x_2), \ldots\rangle$ i sprawdza, czy jest on dopuszczalnym przejawem zachowania procesu kryterialnego.

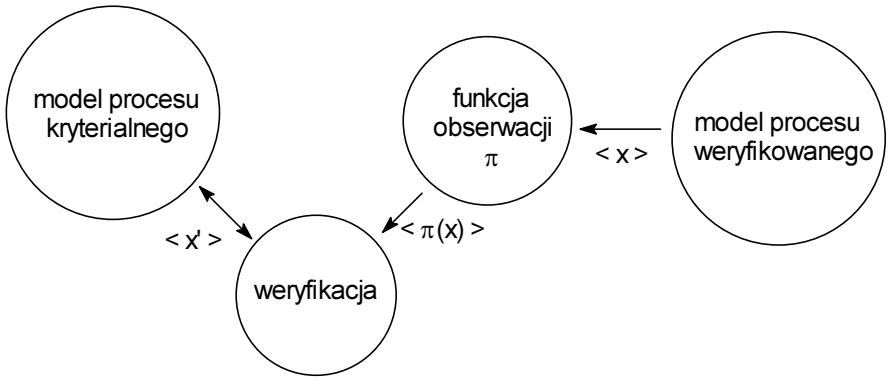
Ze względu na to, że analiza poprawności ukierunkowana jest na analizę wektorów, które mogą reprezentować zarówno ścieżki jak i semiobliczenia stworzony zostanie model specyfikacji procesu w postaci nierówności i równań liniowych abstrahujący od konkretnej specyfikacji, a równocześnie zapewniający wygodne przetwarzanie wektorów.

Model ten będzie bardziej ogólny niż proces lub proces etykietowany zdefiniowany w [Szmuc 97b]. Jego zadaniem będzie między innymi specyfikacja zachowania przejawianego przez procesy współbieżne.

Pojęcie poprawności zdefiniowane zostanie dla dwóch modeli reprezentujących specyfikację procesu weryfikowanego i kryterialnego. Zaletą takiego podejścia jest

możliwość zastosowania analizy poprawności do specyfikacji nieformalnych, którym można nadać ścisłą semantykę przez budowę odpowiedniego modelu.

Na Rys. 3-8 przedstawiono poglądowy schemat zależności pomiędzy obiektami występującymi w analizie poprawności.



Rys. 3-8 Obiekty wystepujace w analizie poprawności

Koncepcja poprawności dla homomorficznej funkcji obserwacji (por. rozdział 5) jest pod wieloma względami podobna do koncepcji relacji symulacji i uściślania opisanych w podrozdziale 2.5.1.

Podana w rozdziale 6 definicja poprawności dla niehomomorficznej funkcji obserwacji będzie nakładała ostrzejsze warunki na sposób obliczania akcji modelu kryterialnego. W praktyce, warunki te ograniczają przeplot akcji należących do zbiorów odwzorowanych w akcje procesu kryterialnego, pomiędzy którymi zachodzi relacja przyczynowości (następstwa). Dla funkcji obserwacji zaproponowanej dla przykładu 4.2 oznacza to, np.: że akcja *prepareToPay* nie może poprzedzić akcji *inLimit*.

Zmodyfikowana funkcja obserwacji (tzw. funkcja obserwacji warunkowej) zostanie zaproponowana dla modelu liniowego rozszerzonego o dane. Jej definicja podana zostanie w rozdziale 8.

4. Model specyfikacji

Celem tego rozdziału jest zdefiniowanie modelu stosowanego w zagadnieniu poprawności. Zaproponowana postać pozwala na ujednolicenie sposobu opisu następujących obiektów:

- niedeterministycznego procesu sekwencyjnego (procesu weryfikowanego),
- procesu kryterialnego,
- systemu komunikujących się procesów.

Założono, że model ten powinien możliwie wiernie opisywać zachowanie wymienionych obiektów, a także pozwolić na ułatwienia przy specyfikacji wymagań w postaci procesu kryterialnego, którego stany początkowe mogą być wielokrotnie osiągane (wieloinstancyjnego procesu kryterialnego).

Założona postać funkcji obserwacji π jako funkcji odwzorowującej wektory skłoniła autora do wprowadzenia opisu modelu w postaci równań i nierówności liniowych. Jak już wspomniano, alternatywną mogłaby być reprezentacja wykorzystująca wielozbiory, ale postać wektorowa wydaje się prostsza w zapisie przy takiej samej sile ekspresji.

4.1 Postać modelu

Opis zasad tworzenia modelu poprzedzimy założeniem, że rozważany jest proces postaci $P = (S, B, F, T)$ lub związany z nim proces etykietowany $P_L = (P, A, \sigma)$ wyznaczający ogólną topologię przejść. Nie opisane wcześniej rozszerzenia stosujące się do procesu kryterialnego lub systemu procesów zostaną wprowadzone bezpośrednio przy omówieniu zasad modelowania różnych typów węzłów lub komunikacji. Zasady te zostaną następnie zastosowane do szczególnych przypadków modelu, jakimi są niedeterministyczny proces sekwencyjny, proces kryterialny i system (zbiór) procesów.

Główną ideą modelowania specyfikacji jest jej transformacja do postaci zapisanych macierzowo ograniczeń nierównościowych i równościowych. Zazwyczaj stanom procesu (węzłom grafu) odpowiada jedno lub więcej ograniczeń nierównościowych, natomiast synchronicznym przejściom ograniczenia równościowe. Jak zostanie dalej pokazane, stworzony model może zostać sprowadzony do macierzowego opisu sieci Petriego.

Def. 4-1 Model liniowy procesu

Dla danego procesu $P = (S, B, F, T)$ i procesu etykietowanego $P_L = (P, A, \sigma)$ model liniowy zdefiniowany jest jako czwórka:

$ML = (X, x_0, A_I x \le b, A_E x = 0)$, gdzie

- X jest przestrzenią rozwiązań związanych z przejściami $X = R^n$, $n = |T| = |A|$; wektor $x \in X$, spełniający $\forall i = 1,...,n \,.\, x_i \ge 0$ nazywany jest rozwiązaniem
- Wektor $x_0 \in X$ jest rozwiązaniem początkowym, w szczególności może być nim wektor zerowy
- $A_I x \le b$ jest zbiorem ograniczeń nierównościowych;
- $A_E x = 0$ jest zbiorem ograniczeń równościowych

■

Współrzędne wektorów będą oznaczane jako $x_k(i)$ lub x_i w zależności od kontekstu. W pierwszym przypadku oznacza to i-tą współrzędną wektora x_k ; w drugim i-tą

współrzędną wektora x. Symbolem x_α oznaczać będziemy współrzędną wektora związana z przejściem etykietowanym symbolem α.

Dla uproszczenia oznaczeń przez A_i będziemy oznaczać i-ty wiersz macierzy $A = [A_I\ A_E]^T$; przynależność wiersza do odpowiedniej podmacierzy określona będzie poprzez zbiory indeksów: I ograniczeń nierównościowych oraz E ograniczeń równościowych $I \cap E = \varnothing$. Przez b_i będziemy oznaczać współrzędną wektora b; dla $i \in E$ spełnione jest $b_i = 0$.

Spośród zbioru ograniczeń nierównościowych będziemy w niektórych momentach opisu jawnie wydzielać ograniczenia związane z kolejnością osiągania i opuszczania węzłów (stanów). Są one postaci $A_i\ x \le 0$. Przez $N \subset I$ oznaczymy zbiór *ograniczeń węzłowych*. Jest on zdefiniowany jako: $N = \{i \in I \mid b_i = 0\}$.

4.2 Modelowanie węzłów

Definiując sposób opisu węzłów za pomocą nierówności i równań liniowych założymy, że modelowany jest proces etykietowany. Ze względu na fakt, że funkcja etykietująca σ jest wzajemnie jednoznaczna (patrz Def. 3-1) analogicznie budowany jest model dla zwykłego procesu.

Dla danego stanu s_i procesu etykietowanego P_L oznaczmy przez $In(s_i)$ zbiór symboli przejść wejściowych, natomiast przez $Out(s_i)$ zbiór symboli przejść wyjściowych.

$$Out(s_i) = \{\alpha \in A \mid \exists\ (s_i, s_{i+1}) \in T\ .\ \alpha = \sigma(\ (s_i, s_{i+1})\ )\ \}$$

$$In(s_i) = \{\alpha \in A \mid \exists\ (s_{i-1}, s_i) \in T\ .\ \alpha = \sigma(\ (s_{i-1}, s_i)\ )\ \}$$

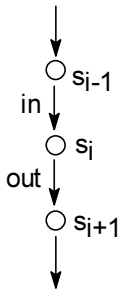
4.2.1 Węzeł prosty

Na Rys. 4-1 przedstawiono fragment specyfikacji procesu. Rozważmy węzeł s_i oraz przejścia $in = \sigma(s_{i-1}, s_i)$ i $out = \sigma(\ s_i, s_{i+1})$. Kierunek grafu i brak rozgałęzień przy węźle s_i narzuca ograniczenie, aby liczba przejść *out* była zawsze mniejsza lub równa liczbie przejść *in*.

Stosowne ograniczenie może być zapisane jako

$$x_{out} - x_{in} \le 0,$$

gdzie x_{in} oraz x_{out} są odpowiednimi współrzędnymi wektora rozwiązania.

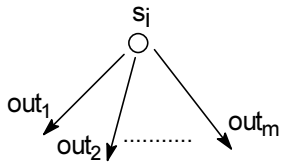


Rys. 4-1 Węzeł bez rozgałęzień

4.2.2 Węzeł początkowy

Przez węzeł początkowy rozumiany jest tu stan $s_i \notin Ran\ (\ T\ )$ (Rys. 4-2). W typowych przypadkach węzłowi początkowemu $s_i \in B$ nie będą przypisywane ograniczenia, które

można związać wyłącznie z danym węzłem. W ten sposób będą modelowane zbiory stanów początkowych.



Rys. 4-2 Węzeł początkowy

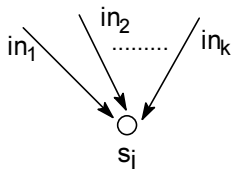
W przypadku, kiedy $s_i \notin B$ zostanie mu jednak przypisane ograniczenie postaci:

$$\sum_{out \in Out(s_i)} x_{out} \leq 0 .$$

Uniemożliwi ono wykonanie jakiejkolwiek z akcji ze zbioru $Out(s_i)$. Brak ograniczenia tej postaci dla przypadku, gdy $s_i \in B$ spowoduje, że przejścia rozpoczynające się w stanie początkowym będą mogły być zawsze wykonane. Przy modelowaniu procesu sekwencyjnego wprowadzone zostaną ograniczenia liczby takich przejść.

4.2.3 Węzeł końcowy

Przez węzeł końcowy rozumiany jest tu stan $s_i \notin Dom(T)$. Podobnie, jak dla węzła początkowego, w przypadku kiedy $s_i \in F$, nie będą mu przypisywane ograniczenia.



Rys. 4-3 Węzeł końcowy

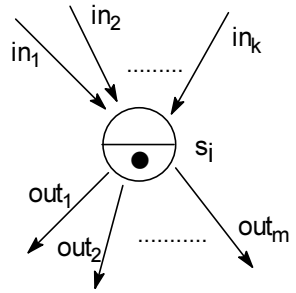
W przypadku, kiedy $s_i \notin F$ przypisane mu zostanie *i*-te ograniczenie węzłowe postaci:

$$\sum_{in \in In(s_i)} (-x_{in}) \leq 0$$

Po wykonaniu dowolnej z akcji ze zbioru $In(s_i)$ wyrażenie $y_i = b_i - A_i x$ przyjmie wartość dodatnią i nie będzie możliwe wykonanie przejścia, które tę wartość mogłoby zmniejszyć. W ten sposób będzie wykrywana stała aktywność stanu nie będącego stanem końcowym.

4.2.4 Węzeł *or-exor*

Węzeł typu *or-exor* jest jedynym typem węzła występującym w grafie niedeteministycznego procesu sekwencyjnego. Po osiągnięciu stanu s_i (Rys. 4-4) za pośrednictwem jednego z przejść ze zbioru $In(s_i)$ możliwe jest wybranie dokładnie jednej z dróg z zbioru $Out(s_i)$.



Rys. 4-4 Reprezentacja graficzna węzła or-exor

Oznacza to, że liczba przejść opuszczających węzeł jest mniejsza lub równa od liczby przejść prowadzących do węzła. Algebraiczny zapis tych reguł w postaci ograniczenia liniowego może być przedstawiony w postaci:

$$\sum_{out \in Out(s_i)} x_{out} - \sum_{in \in In(s_i)} x_{in} \leq 0,$$

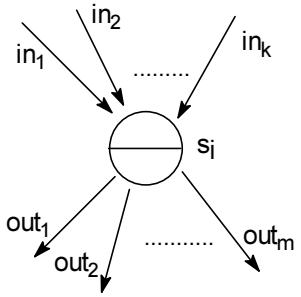
Jak można zauważyć, wykonanie jednego z przejść wyjściowych ze zbioru $Out(s_i)$ jest możliwe, jeśli $y_i = b_i - A_i\, x > 0$. Liczba przejść wejściowych nie jest w tym przypadku ograniczona.

Jeśli nie zostanie zaznaczone graficznie inaczej, węzeł typu *or-exor* będzie domyślnym typem węzła we wszystkich przykładowych specyfikacjach.

4.2.5 Węzeł *or-or*

Węzeł typu *or-or* był domyślnym typem dla węzłów występujących w specyfikacji procesu kryterialnego opisanych w podrozdziale 2.6.7. Na Rys. 4-5 przedstawiono reprezentację graficzną tego węzła.

Przyjęte reguły interpretacji związane z węzłem typu *or-or* mówią, że aby wykonać jedno z przejść wyjściowych ze zbioru $Out(s_i)$ konieczne jest wcześniejsze osiągnięcie węzła s_i poprzez jedno z przejść wejściowych ze zbioru $In(s_i)$. Podstawową różnicą w stosunku do węzła *or-exor* jest jednak to, że po uaktywnieniu węzła, mimo wykonania wszystkich przejść z pewnego zbioru $O \subset Out(s_i)$ dalej jest możliwe wykonanie dowolnego przejścia $out \in Out(s_i) \setminus O$.



Rys. 4-5 Reprezentacja graficzna węzła or-or

Algebraiczny zapis tych reguł może być przedstawiony w postaci m ograniczeń nierównościowych, gdzie $m = |\, Out(s_i)\, |$.

$$\forall out \in Out(s_i).(x_{out} - \sum_{in \in In(s_i)} x_{in} \leq 0)$$

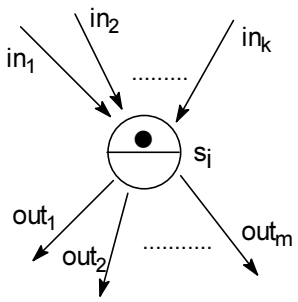
Interpretacja powyższych formuł wskazuje, że węzeł typu *or-or* jest modelowany jako *m* węzłów typu *or-exor* , które osiągane są równocześnie w wyniku wykonania jednego z przejść wejściowych.

4.2.6 Węzeł *and-or*

Węzeł typu *and-or* został wprowadzony w celu modelowania wymagań dotyczących synchronizacji procesów. W podrozdziale 2.6.5 pokazano także jego interpretację dla diagramów DFD. Osiągnięcie stanu s_i z synchronizacją na wejściu (Rys. 4-6) jest możliwe jedynie poprzez jednoczesne wykonanie wszystkich przejść wejściowych ze zbioru $In(s_i)$. Narzuca to następujący zbiór warunków:

$$\forall in_1, in_2 \in In(s_i).(x_{in_1} = x_{in2}),$$

co może być zapisane w postaci k–1 prostych ograniczeń równościowych, gdzie $k = |\ In(s_i)\ |$.



Rys. 4-6 Reprezentacja graficzna węzła and-or

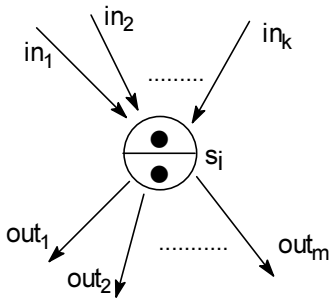
Ograniczenia węzłowe są zapisane podobnie, jak dla węzła *or-or* w postaci $m = |\ Out(s_i)\ |$ ograniczeń nierównościowych:

$$\forall out \in Out(s_i).(x_{out} - x_{in} \leq 0),$$

gdzie *in* jest dowolnym przejściem wejściowym $in \in In(s_i)$.

4.2.7 Węzeł *and-exor*

Węzeł ten (Rys. 4-7) jest w opisany podobnymi ograniczeniami równościowymi opisującymi zależności pomiędzy akcjami wejściowymi jak dla węzła *and-or*.



Rys. 4-7 Reprezentacja graficzna węzła and-exor

Warunek wyjściowy zapisywany jest jako:

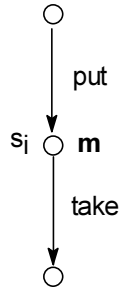
$$\sum_{out \in Out(s_i)} x_{out} - x_{in} \leq 0 ,$$

gdzie *in* jest dowolnym przejściem wejściowym $in \in In(s_i)$

4.2.8 Węzeł o ograniczonej pojemności

Węzeł o ograniczonej pojemności nie występował w pracach dotyczących zagadnienia poprawności względnej z powodu braku reprezentacji dla akumulacji instancji sterowania. Przypadek wielokrotnego uruchamiania procesu opisywany był raczej przez pętle. Typową specyfikacją, gdzie występują analogiczne konstrukcje są sieci Petriego o ograniczonej pojemności miejsc.

W przykładach przedstawionych w pracy węzły o ograniczonej pojemności będą najczęściej modelowały zjawisko buforowania pomiędzy producentem wykonującym akcję *put* a konsumentem wykonującym akcję *take* (Rys. 4-8).



Rys. 4-8 Węzeł o ograniczonej pojemności

Węzeł ten opisany jest dwoma ograniczeniem postaci:

a) $x_{take} - x_{put} \leq 0$

b) $-x_{take} + x_{put} \leq m$.

Warunek (a) jest analogiczny jak dla węzła prostego, który może być traktowany jako węzeł o nieskończonej pojemności. Warunek (b) związany jest z ograniczoną pojemnością węzła. Szczególnym przypadkiem jest węzeł o pojemności 1, który modeluje zachowanie semafora binarnego.

Ograniczenie na pojemność węzła może zostać przypisane dowolnemu węzłowi. Jeśli dany węzeł s_i zostanie opisany przez *k* ograniczeń nierównościowych postaci:

$A_i\, x \leq 0,\ i = 1,\ldots,k$

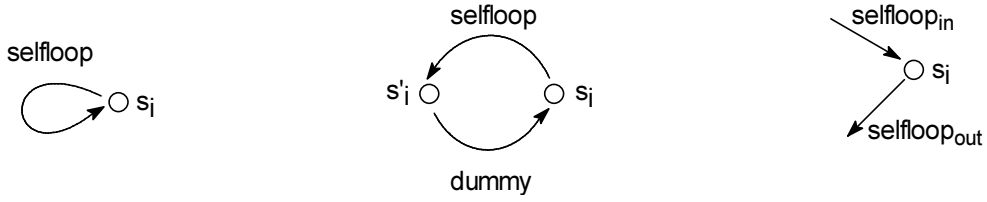
wówczas dodanie ograniczeń na pojemność węzła zostanie odzwierciedlone przez *k* ograniczeń nierównościowych postaci:

$(-A_i)\, x \leq m ,\ i = 1,\ldots,k.$

Tak prosta operacja jest uzasadniona w następujący sposób: dla danego ograniczenia węzłowego $i \in N$ zmienna $y_i = b_i - A_i\, x$ ($b_i = 0$) określa liczbę przejść wyjściowych, które można wykonać. Ograniczenie na pojemność węzła w istocie dotyczy różnicy między liczbą przejść wejściowych a wyjściowych - stąd zmiana znaku przy wektorze ograniczenia. Jak łatwo zauważyć, różnica między liczbą przejść wejściowych a wyjściowych odpowiada liczbie żetonów zgromadzonych w miejscu dla sieci Petriego.

4.2.9 Modelowanie oczek

Przez oczko (ang. *self-loop*) związane z danym stanem s_i rozumiane jest przejście postaci $(s_i, s_i) \in T$. Problem oczek jest, jak się wydaje, jedyną słabością modelu liniowego. Akcja ***selfloop*** związana z oczkiem (Rys. 4-9a) spełnia ***selfloop*** $\in In(s_i) \cap Out(s_i)$, przez co zastosowanie reguł opisu węzła prowadzi do zerowania współczynnika występującego przy ograniczeniu. Prostą operacją stosującą się do oczek (opisaną także przy budowie macierzowego modelu sieci Petriego [Murata 89]) jest wprowadzenie do pętli dodatkowego węzła oraz przejścia oznaczonego na Rys. 4-9b symbolem ***dummy***. W ten sposób na ogół będą modelowane oczka w procesie weryfikowanym. Jest to rozwiązanie niekorzystne, ponieważ w sztuczny sposób zwiększa złożoność obliczeniową algorytmów analizujących model.



(a) (b) (c)

Rys. 4-9 Różne sposoby modelowania oczek

Takie rozwiązanie jest jednakże niedopuszczalne dla procesu kryterialnego, ponieważ w rzeczywistości akcja ***dummy*** nie jest elementem wymagań i brak jest składowej funkcji obserwacji, która pozwalałaby na jej obliczenie.

Z tego powodu w przypadku procesu kryterialnego stosowane będzie rozwiązanie przedstawione na Rys. 4-9c. Akcja tworząca oczko dekomponowana jest na dwie akcje synchroniczne (sprzężone) ***selfloop***$_{in}$ oraz ***selfloop***$_{out}$; pierwsza z nich związana jest ze sztucznie dodanym stanem początkowym, druga ze stanem końcowym. Zgodnie z wcześniej opisanymi regułami, stanom tym nie odpowiadają żadne ograniczenia. Sprzężenie pomiędzy akacjami opisane będzie dodatkowym ograniczeniem równościowym: $x_{selfloop_{in}} = x_{selfloop_{out}}$.

Składowa funkcji obserwacji związana z akcją ***selfloop*** jest odpowiednio modyfikowana do postaci:

$$x_2(selfloop_{in}) = x_2(selfloop_{out}) = \sum_{B \in D\ (\alpha)} \min\{x_1(\beta) \mid \beta \in B\}$$

4.3 Model procesu sekwencyjnego

Dany proces sekwencyjny postaci $P_S = (S_S, B_S, F_S, T_S)$ modelowany będzie jako proces $P = (S, B, F, T)$ przez dodanie do zbioru stanów sztucznego stanu początkowego *INIT* oraz dodatkowych przejść.

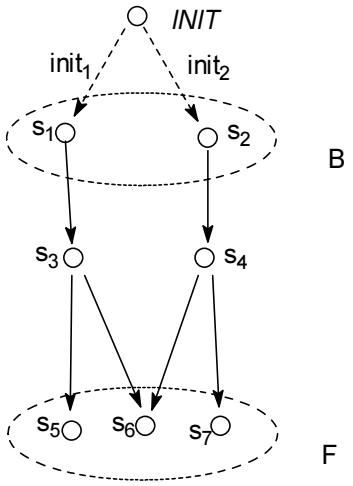
$$S = S_S \cup \{INIT\}$$

$$B = \{INIT\}$$

$$F = F_S$$

$$T = T_S \cup \{INIT\} \times B.$$

Celem wprowadzonego rozszerzenia jest odzwierciedlenie opisanych w podrozdziale 2.6.7 zasad interpretacji procesu sekwencyjnego, a zwłaszcza faktu, że jego obserwowane zachowanie rozpoczyna się wraz z arbitralnym uaktywnieniem jednego ze stanu początkowych.



Rys. 4-10 Rozszerzenie procesu sekwencyjnego

Wszystkie węzły rozszerzonego procesu *P* modelowane są jak węzły typu *or-exor*. Istotną własnością, która musi zostać uwzględniona jest ograniczenie liczby osiągniętych stanów początkowych w danym semiobliczeniu. Ograniczenie to zapisywane jest jako nierówność

$$\sum_{out \in Out(INIT)} x_{out} \leq 1$$

Stanowi *INIT* jako elementowi zbioru stanów początkowych *B* nie jest przypisywane żadne ograniczenie.

4.4 Model procesu kryterialnego

Model procesu kryterialnego występującego w oryginalnej postaci zagadnienia poprawności względnej skonstruowany jest analogicznie do procesu sekwencyjnego przez dodanie sztucznego stanu początkowego *INIT* i przejść początkowych. Dla zapewnienia zachowania podobnego do procesu $\widetilde{P}$ opisanego w podrozdziale 2.6.8 wprowadzony jest inny typ ograniczeń dla przejść prowadzących do stanów początkowych. Są one modelowanej jako *m* nierówności, gdzie *m* = | *Out*(*INIT*) | , postaci

$$\forall out \in Out(INIT).(x_{out} \leq 1)$$

4.4.1 Różnice w interpretacji procesu kryterialnego

Pomiędzy zasadami interpretacji procesu kryterialnego występującego w oryginalnym modelu poprawności, a zachowaniem jego modelu liniowo-algebraicznego istnieją pewne różnice związane ze zjawiskiem akumulacji sterowania w węzłach (jest ono podobne do akumulacji żetonów w miejscach sieci Petriego). Główną przyczyną różnic jest to, że proces kryterialny przede wszystkim specyfikował wymagania względem procesu sekwencyjnego opisującego system współbieżny, w związku z tym jego struktura odpowiadała mocniej strukturze procesu sekwencyjnego, a nie jego specyfikacji.

Rozważmy przykład na Rys. 4-11a. W oryginalna interpretacji procesu kryterialnego zakłożono, że w wyniku przejścia w procesie weryfikowanym osiągany jest pewien stan,

np.: stan s_i. W kategoriach opisu prowadzonego dla akcji oznaczało to równoczesne wykonanie akcji in_1 oraz in_2. Osiągnięcie następnego stanu s_{i+1} wiązało się z wykonaniem operacji *out*.

Interpretacja modelu liniowego procesu pozwala na przesunięcie operacji in_1 oraz in_2 w czasie; w przypadku kiedy obie zostały wcześniej wykonane – akcja *out* może zostać wykonana dwukrotnie.

Jak wydaje się, taka interpretacja lepiej oddaje oczekiwane zachowanie procesu kryterialnego modelującego diagramy przepływów danych DFD (Rys. 4-11 b). Traktując diagram DFD jako opis wymagań, mamy prawo oczekiwać, że w fazie projektowania odpowiednim procesom zostanie nadana kolejność aktywacji zgodna z zasadami przyczynowości opisanymi przez połączenia pomiędzy procesami, np.: ⟨ in_1, *out*, in_2, *out* ⟩ lub ⟨ in_1, in_2, *out*, *out* ⟩, ale nie ⟨ *out*, in_1, in_2, *out* ⟩

(*a*) (*b*)

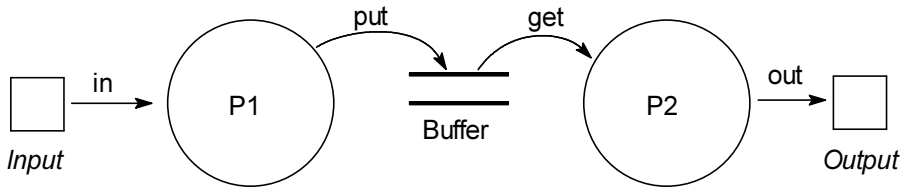
Rys. 4-11 (a) Fragment procesu kryterialnego (b) specyfikacja DFD

4.4.2 Proces wieloinstancyjny

Wieloinstancyjny proces kryterialny jest konstrukcją mającą przede wszystkim na celu ułatwienie specyfikacji kryterium poprawności. Przesłanką do jego wprowadzenia są zaobserwowane problemy związane z opisem poprawnego zachowania aplikacji, w której następuje buforowanie przesyłanych komunikatów pomiędzy aktywnymi procesami.

Nazwa *proces wieloinstancyjny* sugeruje, że może być on traktowany jako specyfikacja dynamicznie tworzonych instancji procesu kryterialnego występującego w oryginalnym zagadnieniu poprawności. Utworzenie instancji procesu związane jest zazwyczaj z pojawieniem się zdarzenia na wejściu systemu.

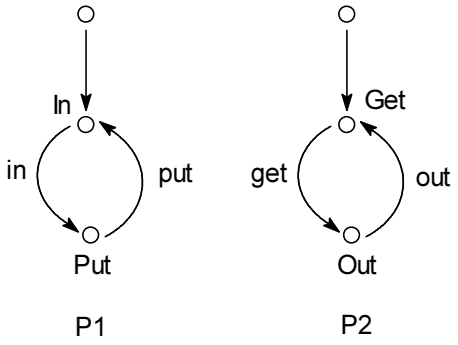
Rozważmy specyfikację DFD przedstawioną na Rys. 4-12 . Zadaniem procesu *P1* jest reakcja na asynchroniczne komunikaty pojawiające się na wejściu i ich zapis do bufora; zadaniem procesu *P2* jest pobieranie komunikatów z bufora i ich wyprowadzanie na zewnątrz aplikacji. Zakładamy tu, że bufor ma pojemność nieograniczoną.



Rys. 4-12 Specyfikacja dwóch procesóww łańcuchu przetwarzania

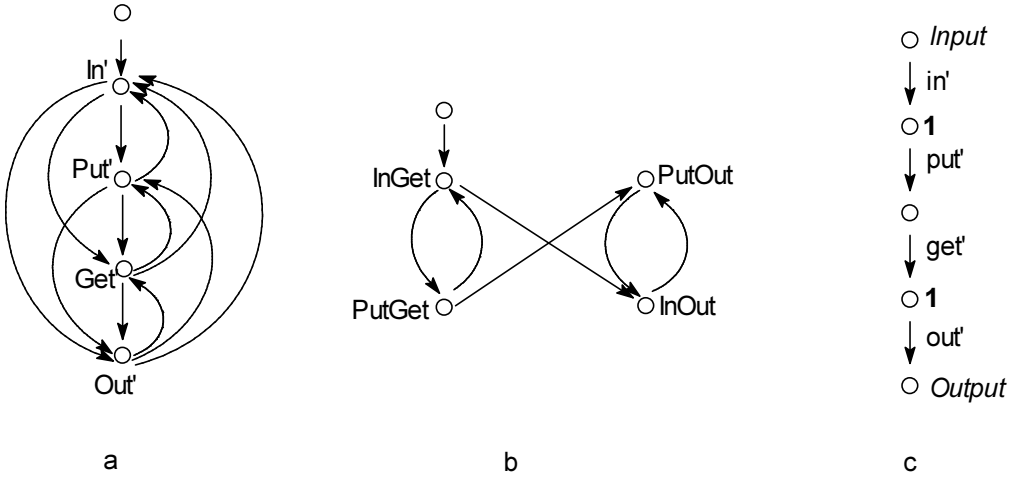
Kolejny schemat (Rys. 4-13) przedstawia prostą specyfikację procesów *P1* oraz *P2* w postaci maszyn skończeniestanowych służącą w tym przypadku jako specyfikacja weryfikowana. Łuki oznaczono etykietami wykonywanych akcji, natomiast stanom

przypisano symbole związane z etykietami następnych akcji (pisane dla odróżnienia z dużej litery).



Rys. 4-13 Specyfikacja procesów (weryfikowana)

Rys. 4-14 przedstawia trzy procesy kryterialne stosujące się do tej samej specyfikacji. Pierwszy z nich (a) usiłuje odtwarzać strukturę przesyłania informacji; drugi (b) jest znacznie prostszy, ale struktura ta jest na nim niewidoczna. Oba zdefiniowane są dla modelu zakładającego odwzorowanie stanów.



Rys. 4-14 Warianty procesów kryterialnych

Proces przedstawiony na Rys. 4-14 c reprezentuje wieloinstancyjny proces kryterialny. Jego konstrukcja dokładnie odpowiada strukturze przesyłania informacji. Stan początkowy *Input* związany jest jawnie z wejściem systemu. W odróżnieniu od zasad interpretacji oryginalnego procesu kryterialnego stan ten może być osiągany wielokrotnie, czyli akcję *in'* można wykonywać bez żadnych ograniczeń. Węzłom procesu kryterialnego związanym z aktywnością procesów *P1* lub *P2* przypisano ograniczenie na pojemność. W ogólnym przypadku nie jest to wymagane.

Jak łatwo zauważyć, proces wieloinstancyjny odpowiada strukturalnej pętli. Brak ograniczeń opisujących jego stany początkowe i końcowe powoduje, że w wyniku wykonania pojedynczej instancji procesu zbiór stanów aktywnych wraca do wartości początkowej.

4.5 Model systemu procesów

4.5.1 Komunikacja

Konstruując model systemu procesów ograniczono się do opisu komunikacji, nie uwzględniając innych mechanizmów synchronicznej zmiany stanów procesów (jak bezpośrednio dzielona pamięć). Nie jest to istotnym ograniczeniem, ponieważ pamięć dzielona jest zazwyczaj w systemach współbieżnych chroniona; rozmaite mechanizmy ochrony, np.: semafory modelowane są przez procesy. Zmiana zawartości pamięci dzielonej nie pociąga za sobą bezpośrednio zmiany stanu zbioru procesów. Taka zmiana może nastąpić dopiero po jej odczycie przez jeden z procesów mających potencjalny dostęp do dzielonego zasobu.

Definicja systemu komunikujących się procesów bazuje na konwencji stosowanej dla maszyn skończenie stanowych CRSM [Shaw 92] przejętej za CSP [Hoare 85]. Opis komunikacji nie uwzględnia wartości przesyłanych komunikatów, lecz traktuje komunikacje wiążące się z przesłaniem rozróżnialnych wartości mających wpływ na przebieg sterowania jako rozróżnialne przejścia.

Model systemu komunikujących się procesów może zostać zastosowany zarówno jako specyfikacja procesu weryfikowanego, jak i kryterialnego. Sposób interpretacji zachowania modelu jest niezależny od jego postaci.

Def. 4-2 System komunikujących się procesów

Systemem komunikujących się procesów nazywana jest para ($\mathbf{P}_L$, ϕ), gdzie

1. $\mathbf{P}_L$ – jest zbiorem rozłącznych sekwencyjnych procesów etykietowanych

   $\mathbf{P}_L = \{ P_{Li} \mid P_{Li} = (P_i, A_i, \sigma_i), i=1,\ldots,m\}$, gdzie
   $\forall i, j \le m, i \ne j \,.\, (S_i \cap S_j = \varnothing \wedge A_i \cap A_j = \varnothing)$;

2. ϕ – jest funkcją częściową opisującą komunikacje $\phi : \bigcup_i A_i \rightarrow \bigcup_i A_i$

■

Założenie, że zależność pomiędzy komunikacjami jest opisywana funkcją ma wpływ na budowę modelu. Oznacza to, że dopuszczalne są komunikacje pokazane na Rys. 4-15, natomiast relacyjna zależność z Rys. 4-16 a. jest niedopuszczalna.

Rys. 4-15 Dopuszczalne warianty synchronizacji przejść związanych z komunikacją

Takie założenie, różne np.: od koncepcji opisu komunikacji przedstawionych w [Szmuc 97b] jest uzasadniane tym, że mechanizmy komunikacji implementowane dla systemów współbieżnych nie pozwalają na synchroniczną realizację relacyjnych zależności pomiędzy komunikacjami. W praktycznych rozwiązaniach używany jest w

takim przypadku asynchroniczny kanał komunikacji [Shaw 92, 94; SSSS 94; SS 95a, 95b] , który może zostać opisany jako bierny proces (Rys. 4-16 b).

(a) (b)

Rys. 4-16 (a) Zależność relacyjna pomiędzy komunikacjami zastąpiona przez (b) kanał komunikacji

System procesów modelowany jest jako proces $P = (S, B, F, T)$ będący prostą sumą procesów składowych: $S = \bigcup_i S_i$, $T = \bigcup_i T_i$, itd. Istotnym elementem modelu jest opis synchronicznych komunikacji jako ograniczeń równościowych.

Dla danego symbolu $\alpha \in Ran(\phi)$ oraz zbioru $D(\alpha) = \phi^{-1}(\alpha)$ ograniczenie równościowe związane z komunikacją α przybiera postać:

$$x_\alpha - \sum_{\beta \in D(\alpha)} x_\beta = 0 .$$

Macierz A_I opisująca system komunikujących się procesów ma postać macierzy blokowo-diagonalnej.

$$A_I = \begin{bmatrix} A_1 & 0 & 0 & 0 & 0 \\ 0 & A_2 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & A_{m-1} & 0 \\ 0 & 0 & 0 & 0 & A_m \end{bmatrix}$$

Macierze A_i, $i = 1,\ldots,m$ leżące na przekątnej są macierzami procesów składowych. Komunikacje opisane są za pomocą macierzy A_E. Struktura macierzy A_I dla procesów sekwencyjnych odpowiada strukturze macierzy incydencji grafu [Deo 74]. Dla stanów z rozgałęzieniem typu *or* (co może dotyczyć jedynie procesu kryterialnego) struktura ta jest odmienna.

4.5.2 Modelowanie funkcji

Problem modelowania funkcji wydaje się bardzo istotny ze względu na możliwość rozszerzenia zakresu zastosowań rozważanego modelu. Zwłaszcza dotyczy to wszelkich specyfikacji opartych na dekompozycji funkcjonalnej (jak metody analizy strukturalnej). Funkcje występują także w metodach obiektowych [CY 1990; Booch 94] (pod nazwą *metod* lub *usług*); jednakże ich specyfikacje są często mało systematyczne lub słabo poddające się analizie (np.: do ich opisu używany jest często pseudokod). Pod tym względem wyjątkowy jest obiektowy SDL [BH 93] używający podstawowej nieobiektowej wersji języka [RS 82; SSR 89] do opisu funkcji w postaci maszyny skończenie-stanowej.

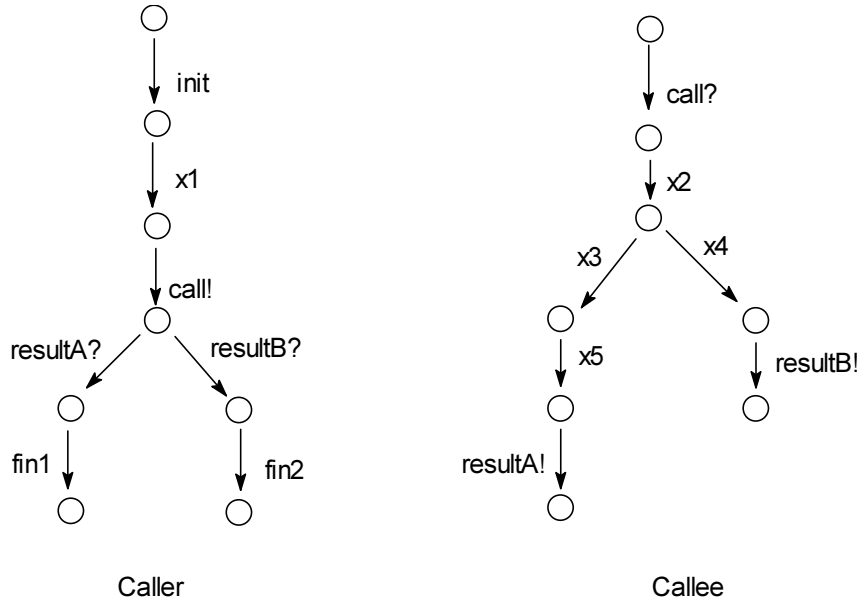
Podobne rozwiązanie jest stosowane i w tym przypadku. Funkcje modelowane są jako szczególnego rodzaju procesy, zgodnie z propozycjami przedstawionymi w [SSSS 96;

SS 96, 97a]. Zaproponowany model jest wystarczająco elastyczny, by wyrazić ich wywołanie oraz przeszukać drzewo wywołań funkcji.

Podstawową ideą modelowania funkcji jest przedstawienie ich w postaci specyfikacji procesu sekwencyjnego, który ma co najmniej jeden wyodrębniony stan początkowy (kilka stanów początkowych odpowiada przesyłaniu różnych parametrów) oraz co najmniej jeden stan końcowy (kilka stanów końcowych odpowiada różnym zwracanym wartościom).

Odpowiednie sprzężenia pomiędzy podprocesem wołającym i podprocesem reprezentującym funkcję pozwalają modelować wywołanie i oczekiwanie na zakończenie działania funkcji. Z założenia – podproces reprezentujący funkcję jest nieaktywny dopóki nie zostanie uaktywniony przez podproces wołający. Oznacza to, że zachowuje się podobnie jak opisany wcześniej wieloinstancyjny proces kryterialny.

Rysunek Rys. 4-17 pokazuje przykład wykorzystania funkcji w modelu. Proces *Caller* wykonuje przejście *call!* współbieżnie z przejściem *call?* procesu *Callee*, a następnie czeka na przesłanie rezultatu (komunikacje *resultA* lub *resultB*).



Rys. 4-17 Przykład wywołania funkcji

Opisany sposób reprezentacji funkcji ma swoje ograniczenia, których nie da się bezpośrednio ominąć w tym modelu. Procesy, w które zostały przekształcone funkcje zachowują się wiarygodnie, tylko wtedy, gdy nie są wołane współbieżnie z różnych wątków sterowania. Jest tak, ponieważ wektor rozwiązania x (jego współrzędnymi są liczby poszczególnych przejść) agreguje informację pochodzącą z różnych wątków.

Zakładając, że do systemu zostanie wprowadzony drugi proces *Caller2* o strukturze podobnej do procesu *Caller*, możemy spotkać się z sytuacją, że wygenerowana zostanie ścieżka, w której jeden z procesów wywoła funkcję, następnie proces ją reprezentujący osiągnie stan bliski końca, po czym drugi z procesów wywoła funkcję i natychmiast otrzyma jej rezultat.

Niestety, jedynym racjonalnym remedium jest rozróżnianie wątków. W rozważanym modelu oznaczałoby to konieczność zastąpienia n–elementowego wektora rozwiązania x macierzą X o rozmiarach $n \times r$, gdzie r jest maksymalną liczbą wątków równą liczbie statycznych procesów. Model taki nie był tworzony, ponieważ w ogólnej postaci powinien

obejmować szerszą klasę problemów (np.: dynamicznie tworzone procesy dla metod obiektowych).

Model funkcji będzie stosowany przy pełnej świadomości jego ograniczeń, jako wygodniejszy niż rozwijanie przejść wołającego procesu w drzewa. W przypadku konieczności użycia danej funkcji przez dwa współbieżne procesy – raczej do modelu dodana będzie jej kopia.

4.6 Rozwiązania i stany

Rozwiązaniem dla modelu liniowego specyfikacji $ML = (\mathrm{X}, \mathrm{A_I}\, x \le \mathrm{b}, \mathrm{A_E}\, x = 0)$ nazywany jest wektor $x \in \mathrm{X}$. Wektor $x_\mathrm{f} \in \mathrm{X}$ spełniający ograniczenia modelu nazywany jest rozwiązaniem dopuszczalnym. Zbiór $\mathrm{X_F} = \{x \in \mathrm{X} \mid x \le \mathrm{b}, \mathrm{A_E}\, x = 0\}$ nazywany jest zbiorem rozwiązań dopuszczalnych.

Wektor $y = \mathrm{b} - \mathrm{A_I}\, x$ nazywany jest *stanem* systemu. Stan y_f jest dopuszczalny, jeżeli istnieje dopuszczalne rozwiązanie x_f takie, że $y_\mathrm{f} = \mathrm{b} - \mathrm{A_I}\, x_\mathrm{f}$. Stanami dopuszczalnymi są jedynie wektory o nieujemnych wartościach współrzędnych. Zbiór wszystkich stanów oznaczany będzie przez Y.

Przez y(x) oznaczana będzie funkcja, która zgodnie z podanym wzorem przypisuje rozwiązaniom stany.

Zgodnie z terminologią programowania liniowego ograniczeniem aktywnym w rozwiązaniu x nazywany i-te ograniczenie, dla którego zachodzi: $y_\mathrm{i} = \mathrm{b_i} - \mathrm{A_i}\, x = 0$.

Jak wcześniej stwierdzono, spośród ograniczeń nierównościowych wyróżnione będą te, które są bezpośrednio związane ze węzłami (stanami procesów występujących w specyfikacji), a nie pojemnościami tych węzłów. Dany stan specyfikacji $s_\mathrm{k} \notin B \cup F$ jest jednoznacznie identyfikowany przez zbiór ograniczeń węzłowych $N_\mathrm{k} \subset N$. (Jedynie dla węzła wyjściowego *or* zbiór ten zawiera więcej niż jeden element.)

Mówimy, że stan s_k jest aktywny jeśli istnieje $i \in N_\mathrm{k}$ takie, że i-te ograniczenie jest nieaktywne, czyli $\mathrm{b_i} - \mathrm{A_i}\, x > 0$. W szczególności dla $x = 0$, żaden stan $s_\mathrm{k} \notin B \cup F$ nie jest aktywny.

Def. 4-3 Przejście elementarne

Przejściem elementarnym $v \in \mathrm{X}$ nazywany jest wektor spełniający:

1. $v > 0$, gdzie $v(i) \in \{0,1\}$;
2. $\mathrm{A_E}\, v = 0$;
3. jeśli $v = v_1 + v_2$, to żaden z v_1 i v_2 nie spełnia równocześnie warunków (1) i (2) .

Zbiór wszystkich przejść elementarnych będzie oznaczany przez V.

■

Def. 4-4 Dopuszczalne przejście elementarne

Dla danego stanu systemu y przejście elementarne v jest dopuszczalne, jeżeli:

$$y - A_i\, v \geq 0.$$

Zbiór wszystkich dopuszczalnych przejść elementarnych w stanie y oznaczany będzie przez $V_F(y)$.

■

Podobnie dla danego rozwiązania x przejście elementarne v jest dopuszczalne, jeżeli $y(x) - A_i\, v \geq 0$. Analogicznie przez $V_F(x)$ oznaczany będzie zbiór wszystkich przejść elementarnych dla rozwiązania x .

Def. 4-5 Rozwiązanie osiągalne

Mówimy, że rozwiązanie $x_n \in X_F$ jest osiągalne z $x_0 \in X_F$ jeżeli

1. $x_n = x_0$
2. $x_n = x_0 + v$, gdzie $v \in V_F(x_0)$
3. $x_n = x_{n-1} + v$, gdzie $v \in V_F(x_{n-1})$ oraz rozwiązanie x_{n-1} jest osiągalne z x_0.

■

Def. 4-6 Stan osiągalny

Dopuszczalny stan y_n jest osiągalny z dopuszczalnego stanu y_0 jeżeli

1. $y_n = y_0$
2. $y_n = y_0 - A_I\, v$, gdzie $v \in V_F(y_0)$
3. $y_n = y_{n-1} - A_I\, v$, gdzie $v \in V_F(y_{n-1})$ oraz stan y_{n-1} jest osiągalny z y_0.

■

Wyrażenie $\Delta y = A_I\, v$ nazywane jest dopuszczalną zmianą stanu w stanie y_0 jeżeli $v \in V_F(y_0)$.

Wniosek 4-1

Rozwiązanie x_n jest osiągalne z x_0 (lub stan y_n jest osiągalny z y_0) jeżeli istnieje ciąg rozwiązań $\langle x_1,\ldots,x_n\rangle$ (lub stanów $\langle y_1,\ldots,y_n\rangle$) oraz ciąg przejść elementarnych $\langle v_1,\ldots,v_{n-1}\rangle$ takich, że $v_i \in V_F(x_{i-1})$ (lub $v_i \in V_F(y_{i-1})$).

■

Def. 4-7 Zbiór przejść równoległych dopuszczalnych w stanie y_0

Zbiór przejść $V_{\|} \subset V_F(y_0)$ jest zbiorem przejść równoległych w stanie y_0 jeśli dla każdego podzbioru $V_i \subset V_{\|}$ zachodzi

1. $y_0 - A_I\, \Delta x_i \geq 0$, gdzie $\Delta x_i = \sum_{v \in V_i} v$
2. $V_{\|} \setminus V_i \subset V_F(y_0 - A_I\, \Delta x_i)$

Równoważna definicja dla rozwiązań jest następująca:

Def. 4-8 Zbiór przejść równoległych dopuszczalnych w rozwiązaniu x_0

Zbiór przejść $V_{\|} \subset V_F(x_0)$ jest zbiorem przejść równoległych w rozwiązaniu x_0 jeśli dla każdego podzbioru $V_i \subset V_{\|}$ zachodzi

1. $b - A_I (x_0 + \Delta x_i) \geq 0$, gdzie $\Delta x_i = \sum_{v \in V_i} v$
2. $V_{\|} \setminus V_i \subset V_F(x_0 + \Delta x_i)$

■

Wniosek 4-2

Jeżeli zbiór $V_{\|} \subset V_F(x_0)$ jest zbiorem przejść równoległych, to rozwiązanie $x_0 + v_{\|}$, gdzie $v_{\|} = \sum_{v \in V_{\|}} v$ jest osiągalne w wyniku dowolnej permutacji przejść elementarnych ze zbioru $V_{\|}$

■

Twierdzenie 4-1

1. Niech V oznacza pewien podzbiór zbioru przejść elementarnych dopuszczalnych w stanie y_0 , $V \subset V_F(x_0)$.
2. Niech I oznacza zbiór indeksów wierszy macierzy A_I.
3. Dla dowolnego $v_r \in V$ oraz $i \in I$ oznaczmy przez

$$Y_{ir}^{+} = \sum_{j \leq n \wedge A_i(j) v_r(j) > 0} A_i(j)\, v_r(j)$$

gdzie $A_i(j)$, $v_r(j)$ oznaczają j-ty element odpowiednich wektorów. Zmienna Y_{ir}^{+} oznacza sumę wyrazów postaci $A_i(j) \cdot v_r(j)$ przyjmujących jedynie wartości dodatnie.

Zbiór V jest zbiorem przejść równoległych $\Leftrightarrow \forall i \in I \,.\, \sum_{v_r \in V} Y_{ir}^{+} \leq y_0(i)$ (i)

■

Dowód

Zdefiniujmy $Y_{ir}^{-} = \sum_{j \leq n \wedge A_i(j) v_r(j) < 0} A_i(j)\, v_r(j)$ i zauważmy, że $A_i v_r = Y_{ir}^{+}$, jeżeli $A_i v_r > 0$ albo $A_i v_r = Y_{ir}^{-}$, jeżeli $A_i v_r < 0$. Zachodzi także zależność $A_i v_r = Y_{ir}^{+} + Y_{ir}^{-}$.

Jeśli V jest zbiorem przejść równoległych, wówczas dla dowolnego $i \in I$ oraz dowolnego zbioru $V_1 \subset V$ zachodzi:

$$\sum_{v_r \in V_1} (Y_{ir}^{+} + Y_{ir}^{-}) + \sum_{v_r \in V \setminus V_1} (Y_{ir}^{+} + Y_{ir}^{-}) \leq y_0(i) \qquad \text{(ii) oraz}$$

$$\sum_{v_r \in V_1} (Y_{ir}^{+} + Y_{ir}^{-}) \leq y_0(i) \qquad \text{(iii)}$$

W szczególności obie nierówności muszą zachodzić dla zbioru

$$V_1 = \{\, v_r \in V \mid A_i v_r = Y_{ir}^{+} \,,\, A_i v_r > 0 \,\},$$

a zatem (iii) $\Rightarrow \sum_{v_r \in V_1} Y_{ir}^{+} \le y_0(i)$.

Jeśli zachodzi nierówność (i) , wówczas dla dowolnego zbioru $V_1 \subset V$ zachodzi (ii) oraz (iii) ponieważ $Y_{ir}^{+} + Y_{ir}^{-} \le Y_{ir}^{+}$.

■

Def. 4-9 Dopuszczalne równoległe przejście złożone

Przejściem złożonym v_c dopuszczalnym w stanie y_i nazywany jest wektor spełniający następujące warunki:

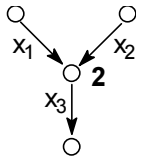
1. $v_c > 0$ oraz $v_c(i) \in \{0, 1\}$,
2. $v_c = \sum_{v_r \in V_{||}} v_r$, gdzie $V_{||} \in 2^{V_F(y_i)}$
3. zbiór $V_{||}$ jest zbiorem przejść równoległych

Analogicznie można mówić o dopuszczalnym równoległym przejściu złożonym w rozwiązaniu x_i przez podstawienie $y_i = y(x_i)$.

■

Dopuszczalne równoległe przejścia złożone są istotne z punktu widzenia zastosowania w badaniu poprawności. Oczekuje się, że każda zaobserwowana zmiana rozwiązania procesu kryterialnego będzie dopuszczalnym przejściem złożonym. Twierdzenie 4-1 określa warunki, jakie muszą zostać sprawdzone, aby sprawdzić, czy dla dwóch kolejnych rozwiązań x_i oraz x_{i+1} $v' = \pi(x_{i+1}) - \pi(x_i)$ jest dopuszczalnym przejściem złożonym dla rozwiązania $x_i' = \pi(x_i)$.

Rozważmy fragment specyfikacji procesu (Rys. 4-18). Węzeł środkowy ma pojemność ograniczoną do 2. Dla rozwiązania $x = [x_1{:}1, x_2{:}0, x_3{:}0]$ przejście złożone $[x_1{:}1, x_2{:}1, x_3{:}1]$ nie jest dopuszczalnym przejściem równoległym, mimo, że wszystkie przejścia elementarne ($[x_1{:}1, x_2{:}0, x_3{:}0]$, $[x_1{:}0, x_2{:}1, x_3{:}0]$ oraz $[x_1{:}0, x_2{:}0, x_3{:}1]$) są w tym rozwiązaniu dopuszczalne, ponieważ po wykonaniu przejścia elementarnego $[x_1{:}1, x_2{:}0, x_3{:}0]$ przejście $[x_1{:}0, x_2{:}1, x_3{:}0]$ jest niedopuszczalne. Łatwo sprawdzić, że w takim przypadku warunki określane przez Twierdzenie 4-1 nie są spełnione.



Rys. 4-18 Fragment specyfikacji procesu

4.7 Wykonanie modelu

Wykonanie modelu $ML = (X, A_I x \le b, A_E x = 0)$ polega na generacji ciągu rozwiązań $s(x_0,\bullet) = \langle x_0, x_1,\ldots,x_i,\ldots \rangle$ począwszy od pewnego rozwiązania początkowego $x_0 \in X_F$. W szczególności wybranym rozwiązaniem początkowym może być wektor zerowy $x_0 = 0$. Dla każdej pary kolejnych rozwiązań (x_{i-1}, x_i) należących do ciągu spełnione jest $v_i = x_i - x_{i-1} \in V_F(x_{i-1})$. Jeżeli dla danego x_{i-1} zachodzi $| V_F(x_{i-1}) | > 1$, wówczas arbitralnie

jest dokonywany niedeterministyczny wybór pomiedzy dopuszczalnymi przejściami elementarnymi. Ciąg $s(x_0, x_n)$ jest skończony jeśli $V_F(x_n) = \varnothing$.

Ciągi postaci $s(x_0,\bullet)$ zawierają wektory o współrzędnych całkowitoliczbowych. Przyjęty model jest z tego punktu widzenia zbyt szeroki, ponieważ opisuje ograniczenia dla wektorów o współrzędnych rzeczywistych. Taka postać została przyjęta z myślą o ewentualnych przyszłych zastosowaniach, w których zjawisko niedeterminizmu może zostać usunięte przez wprowadzenie prawdopodobieństw. Ograniczenia dotyczące rozgałęzień pozostają niezmienne również dla rozwiązań niecałkowitoliczbowych. Rozważmy dwa ciągi $s_1(x_{10},\bullet)$ oraz $s_2(x_{20},\bullet)$ różniące się począwszy od i+1 wyrazu. Ciąg postaci $s(z_0,\bullet)$, gdzie $z_i = a_1 \cdot x_{1i} + a_2 \cdot x_{2i}$, $a_1 + a_2 = 1$ jest również ciągiem rozwiązań dopuszczalnych, jako kombinacja liniowa rozwiązań dopuszczalnych należących do zbioru wypukłego.

4.7.1 Algorytm wykonania modelu

Poniżej przedstawiony zostanie algorytm wykonania modelu, czyli generacji sekwencji rozwiązań. Pierwszym etapem algorytmu jest wyznaczanie zbioru przejść elementarnych. Zbiór ten jest zapisywany postaci macierzy B_V, której liniowo niezależne wierszowe wektory są przejściami elementarnymi ze zbioru V .

1. Wyznacz zbiór przejść elementarnych V .
2. Podstaw $x_i := x_0$.
3. Wyznacz $V_F(x_i)$ zbiór przejść elementarnych dopuszczalnych w rozwiązaniu x_i .
4. Jeśli zbiór $V_F(x_i) = \varnothing$ przejdź do 7 .
5. Wybierz dowolne przejście $v_i \in V_F(x_i)$.
6. Podstaw $x_{i+1} = x_i + v_i$; podstaw $i := i$+1; przejdź do 3.
7. Dokonaj analizy rozwiązania x_i kończącego sekwencję.

4.7.2 Procedura wyznaczania macierzy przejść elementarnych

Zgodnie z Def. 4-3 każde przejście elementarne v jest najmniejszym nieujemnym wektorem spełniającym $A_E\, v = 0$. Z definicji wynika także, że zbiór wszystkich przejść elementarnych V jest zbiorem wektorów liniowo niezależnych. Zadaniem algorytmu jest wyznaczenie wszystkich liniowo niezależnych niezerowych wektorów o nieujemnych składowych spełniających zbiór ograniczeń równościowych.

Zauważmy także, że macierz A_E ma specyficzną postać związaną z modelowaniem węzłów typu *and* lub komunikacji pomiędzy procesami (por. 4.5).

Procedura wyznaczania macierzy B_V opiera się na rozwiązaniu zaczerpniętym z [Murata 89; Hohn 58] .

Załóżmy, że macierz A_E ma wymiary $m \times n$. Niech r będzie rzędem macierzy A_E. Macierz A_E zostanie podzielona na cztery podmacierze: A_{11} - nieosobliwą macierz rzędu r, podmacierz A_{12} o wymiarach $(n–r)\times r$ oraz macierze A_{21} o wymiarach $r\times(m–r)$ i A_{22} o wymiarach $(n\text{-}r)\times(m\text{-}r)$.

$$A_E = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{matrix} \updownarrow r \\ \updownarrow m-r \end{matrix} \quad \text{(i)}$$

Zbiór $(m–r)$ liniowo niezależnych rozwiązań układu $A_E\, v = 0$ dany jest w postaci macierzy

$$B_V = [-A_{12}^T (A_{11}^T)^{-1} ; I_\mu], \text{ (ii)}$$

gdzie I_μ jest macierzą jednostkową rzędu $\mu = m - r$. W źródłowej pracy przeprowadzony był nieco inny podział macierzy; tu ze względu na zastosowany algorytm wymieniono kolumny pierwszą z drugą, stąd zmienione są kolumny macierzy B_V . Nie jest to istotna zmiana, ponieważ dla wyznaczenia podziału (i) niezbędna jest permutacja wierszy i kolumn A_E.

Określanie podziału na podmacierze zgodnie z (i) oraz wyznaczanie czynnika $(A_{11}^T)^{-1}$ dokonywane w trakcie $r \le m$ kroków algorytmu bazującego na eliminacji Gaussa-Jordana [BF 85] z częściowym wyborem elementu głównego. Przyjęta została najprostsza metoda ze względu na dobre uwarunkowanie macierzy A_E.

Algorytm

1. Podstaw i := 1 , $A^{(0)} = A_E$; $r_i := m$.
2. Jeśli $i = r_i$, to STOP.
3. Jeśli i-ty wiersz jest zerowy, wówczas zamień wiersze i oraz r_i , następnie podstaw $r_i := r_i - 1$ i przejdź do 2. (Zmienna r_i jest licznikiem niezerowych wierszy macierzy $A^{(i-1)}$.)
4. Określ jako element centralny $A_{ik}^{(i-1)}$, dla którego wyrażenie $\sum_{j=0}^{r_i} |A_{jk}^{(i-1)}|$ przyjmuje wartość maksymalną. (Jako element centralny wybierany jest ten, dla którego suma wyrazów w kolumnach jest maksymalna.)
5. Wykonaj i-ty krok eliminacji Gaussa-Jordana, aby wyznaczyć nową macierz $A^{(i)}$ oraz podmacierz o wymiarach $i \times i$ macierzy A_{11}^{-1}.
6. Podstaw $r_{i+1} := r_i$; $i := i+1$; przejdź do 2.

Otrzymana w wyniku eliminacji macierz A_{11}^{-1} oraz odpowiednie macierze określające permutacje wierszy i kolumn pozwalają na obliczenie poszukiwanej macierzy B_V ze wzoru (ii).

Procedura ta jest wykonywana jednorazowo dla zazwyczaj niewielkiej macierzy A_E , dlatego jej złożoność obliczeniowa jest pomijalna.

4.7.3 Procedura wyznaczania zbioru dopuszczalnych przejść elementarnych

Zbiór dopuszczalnych przejść elementarnych w rozwiązaniu x_r wyznaczany jest jako suma:

- dopuszczalnych przejść początkowych
- przejść, które są określane na podstawie zbioru aktywnych stanów specyfikacji

Zbiór dopuszczalnych przejść początkowych V_{S0} określany jest dla rozwiązania początkowego x_0. W kolejnych rozwiązaniach x_{r-1} , x_r zbiór V_{Si} wyznaczany jest jako

$$V_{Sr} = \{v \in V_{Sr-1} \mid y(x_r) - A\, v \ge 0\}$$

W szczególności dla procesów sekwencyjnych zbiór ten może być pusty począwszy od pewnego rozwiązania.

W 4.1 zdefiniowano zbiór ograniczeń węzłowych N jako tych ograniczeń nierównościowych, dla których $b_i = 0$. Dla danego rozwiązania x_r wyznaczony jest zbiór ograniczeń nieaktywnych (czyli ograniczeń związanych z aktywnymi stanami specyfikacji) $AN \subset N$ jako: $AN = \{i \in N \mid A_i\, x_r < 0\}$

Wszystkie dopuszczalne przejścia związane z aktywnymi stanami określone są przez zbiór *AN* i na ogół będą wiązały się ze zwiększeniem wartości A_i *v*.

Określmy wektor $z \in X$ jako

$$z_j = \begin{cases} 1 & \text{gdy } \exists i \in AN \,.\, A_{ij} > 0 \\ 0 & \text{w p.p.} \end{cases}$$

Współrzędne tego wektora określają te przejścia atomowe, które prowadziły będą do zwiększenia wartości A_i *v* .

Zbiór przejść elementarnych związanych z aktywnymi stanami specyfikacji V_{Ar} w rozwiązaniu x_r określany jest jako:

$$V_{Ar} = \{B_{Vi} \mid B_{Vi} \cdot z = \mid B_{Vi} \mid_{\Sigma} \ \wedge y(x_r) - A_I\, B_{Vi} \geq 0\},$$

gdzie „·" oznacza iloczyn skalarny wektorów, $\mid x \mid_{\Sigma}$ jest normą postaci $\sum_{j=0}^{n} \mid x_j \mid$.

Ostatecznie zbiór $V_F(x_r)$ jest sumą zbiorów V_{Sr} oraz V_{Ar}.

4.7.4 Rozróżnienie pomiędzy rozwiązaniem końcowym i blokowaniem

Def. 4-10 Rozwiązanie końcowe

Osiągalne rozwiązanie x_n nazywane jest końcowym jeśli zachodzi:

1. $V_F(x_n) = \varnothing$
2. $\forall i \in N \,.\, A_i\, x_n = 0$

■

Pierwszy warunek powyższej definicji oznacza, że w danym rozwiązaniu x_n dalsza generacja rozwiązań jest niemożliwa. Drugi warunek wiąże się ze sprawdzaniem, czy dla danego rozwiązania x_n kończącego ciąg system znalazł się w pożądanym stanie *y*¸ w którym żaden ze stanów specyfikacji nie jest aktywny. Niespełnienie tego warunku, czyli sytuacja, gdy $\exists\, i \in N \,.\, A_i\, x_n < 0$ utożsamiana jest z blokowaniem (ang. *deadlock*).

Zdefiniujmy następujące predykaty:

$$\textit{final}(x) \quad \Leftrightarrow \quad V_F(x) = \varnothing \wedge \forall i \in N \,.\, A_i\, x = 0$$

$$\textit{deadlock}(x) \quad \Leftrightarrow \quad V_F(x) = \varnothing \wedge \exists\, i \in N \,.\, A_i\, x < 0$$

4.8 Podobieństwo modelu do sieci Petriego

Model liniowy *ML* = (X, $A_I\, x \leq b$, $A_E\, x = 0$) może zostać sprowadzony do macierzowej reprezentacji sieci Petriego.

Oznaczmy przez μ liczbę wierszy macierzy B_V. Niech $U = R^{\mu}$ będzie przestrzenią wektorową o wymiarze μ. Macierz B_V^T opisuje przekształcenie liniowe przetrzeni $U \rightarrow X$.

Zapiszmy ograniczenia równościowe w postaci

$$A_E\, B_V^T\, u = 0$$

Macierz będąca wynikiem iloczynu $A_E\, B_V^T$ jest macierzą zerową, stąd ograniczenia równościowe mogą zostać usunięte z przekształconego modelu.

Niech x_0 będzie wybranym rozwiązaniem początkowym (np.: wektorem zerowym), natomiast x_i pewnym rozwiązaniem osiągalnym z x_0. Oznacza to, że do osiagniecia x_i wykonano pewną skończoną liczbę przejść elementarnych, stąd:

$$x_i = x_0 + \sum_{j \le \mu} u_j \mathrm{B}_{Vj}^T, \text{ gdzie } u_j \in \{0\} \cup \mathrm{N}\,.$$

Oznaczmy przez $M_0 = \mathrm{b} - \mathrm{A_I}\, x_0$ oraz przez $M_i = \mathrm{b} - \mathrm{A_I}\, x_i = M_0 - \mathrm{A_I}\, \mathrm{B_V}^{\mathrm{T}}\, u_{\cdot}$. Stąd mamy:

$$M_i = M_0 + \mathrm{A_u}\, u, \text{ gdzie } \mathrm{A_u} = -\mathrm{A_I}\, \mathrm{B_V}^{\mathrm{T}} \text{ (i)}$$

Łatwo zauważyć, że powyższe równanie opisuje równanie stanu sieci Petriego, gdzie M_0 jest znakowaniem początkowym, natomiast M_i jest znakowaniem osiągalnym z M_0 w wyniku wykonania wektora tranzycji u. (Por. 2.1.3)

Model liniowy jest więc sprowadzalny do sieci Petriego i własności odpowiednich algorytmów (skończoność i złożoność obliczeniowa) są zachowywane.

Sieci Petriego opisują szerszą grupę modelów niż te, które zostały przyjęte jako obiekty występujące w zagadnieniu poprawności. Zakłada się, że obiekty te w stosunku do klasy ogólnych sieci Petriego są poddane następującym ograniczeniom:

1. Każde przejście opisane jest przez wektory o wartościach ze zbioru {0,1}, a zatem łuki tranzycji mają wagę 1.
2. Specyfikacja weryfikowana jest zbiorem procesów sekwencyjnych lub komunikujących się procesów sekwencyjnych, które mogą być opisane siecią ograniczoną. Jej zachowanie jest wiec reprezentowane przez graf stanów osiągalnych, a nie graf pokrycia.
3. W przypadku modelu specyfikacji kryterialnej dopuszczany jest nieograniczony zbiór stanów osiągalnych.

4.9 Proces opisujący wykonanie modelu

4.9.1 Własności ciągów rozwiązań

Podobnie jak dla semiobliczeń (por. 2.6.3, Def. 2-5) przez $s(z_0,\bullet)$ oznaczali będziemy ciąg wektorów rozpoczynających od wektora z_0 , przez $s(z_0, z_n)$ ciąg zakończony wektorem z_n.

Przez $Z(s)$ oznaczany będzie zbiór wszystkich elementów ciągu.

Niech $s(v_1, v_n)$ będzie ciągiem przejść elementarnych. Para $(x_0, s(v_1, v_n))$ jednoznacznie określa ciąg rozwiązań postaci $\langle x_0, x_1, \ldots, x_n\rangle$, gdzie $x_i = x_{i-1} + v_i$.

Niech $s(x_0, x_n)$ będzie ciągiem rozwiązań. Ciąg ten jednoznacznie określa ciąg stanów postaci $s(y_0, y_n)$, gdzie $y_i = \mathrm{b} - \mathrm{A_I}\, x_i$. Oczywiście, ciąg stanów nie określa jednoznacznie ciągu rozwiązań, ponieważ dwa kolejne stany mogą być połączone równoległymi przejściami.

Def. 4-11 Stan pętlący

Rozważmy ciąg rozwiązań osiągalnych $s(x_0, x_n)$ i odpowiadający mu ciąg stanów $s(y_0, y_n)$.

Stan y_n nazywany jest pętlącym, jeżeli $\exists\, y_i \in Z(s(y_0, y_{n-1}))\,.\, y_i = y_n$.

■

Jak łatwo zauważyć, wektor $x_c = x_n - x_i$ spełnia równość $\mathrm{A_I}\, x_c = 0$.

Wektorem pętli podstawowej nazywany jest taki wektor x_c, który spełnia zależność $A_I x_c = 0$ oraz może być on wyrażony jako suma elementów ciągu przejść elementarnych nie zawierającego powtórzeń. Dla zbioru *r* różnych pętli podstawowych, ich kombinacja liniowa $x_l = \sum_r k_r x_{c_r}$, również spełnia zależność $A_I x_l = 0$.

Def. 4-12 Relacja (ostrego) pokrywania

Rozważmy dwa stany y_1 oraz y_2 , gdzie $y_1, y_2 \in R^m$.

Określmy relację ostrego pokrywania $y_1 \prec y_2$ jako:

(a) $\forall\, i = 1,\ldots,m$. ($y_1(i) \le y_2(i)$, gdy $y_1(i) > 0$) $\wedge$ ($y_1(i) = y_2(i)$, gdy $y_1(i) = 0$)

(b) $\exists\, i$: $1 \le i \le m$. $y_1(i) < y_2(i)$

■

Wniosek 4-3

1. Jeśli $y_1 = y_2$ lub $y_1 \prec y_2$ lub $y_2 \prec y_1$, to $V_F(y_1) = V_F(y_2)$;
2. Jeśli $y_1 = y_2$ lub $y_1 \prec y_2$, lub $y_1 < y_2$ (czyli $y_1 \le y_2$) wówczas każde złożone przejście dopuszczalne w y_1 jest również dopuszczalne w y_2 .

■

Stan y_2 spełniający zależność $y_1 \prec y_2$ nazywany będzie stanem pokrywającym. Może on również być traktowany jako rodzaj stanu pętlącego, ponieważ zbiory aktywnych stanów specyfikacji utożsamianych z ograniczeniami węzłowymi są w nich jednakowe i identyczne są w obu zbiory przejść dopuszczalnych.

Zauważmy, że jeśli występuje ograniczenie na pojemność węzła, wówczas związane z nim ograniczenia węzłowe i ograniczenia na pojemność są postaci $A_i x \le 0$ oraz $A_j x \le b(j)$, gdzie $A_j = -A_i$, $b(j)$ jest liczbą ograniczającą pojemność węzła. Dla żadnej z par stanów osiągalnych (y_1, y_2) nie może wówczas zachodzić równocześnie $y_1(i) < y_2(i)$ i $y_1(j) < y_2(j)$.

4.9.2 Konstrukcja procesu opisującego wykonanie modelu

Niech $ML = (X, x_0, A_I x \le b, A_E x = 0)$ będzie modelem liniowym specyfikacji. Niech *n* będzie wymiarem przestrzeni X, załóżmy, że macierz A_I ma rozmiar $m \times n$.

Opisana zostanie konstrukcja procesu sekwencyjnego postaci $\tilde{P} = (\tilde{S}, \tilde{B}, \tilde{F}, \tilde{T})$ opisującego zachowanie modelu liniowego. Proces ten jest skonstruowany jako granica ciągu procesów $\Theta = \langle P^{(0)},\ldots,P^{(i)},\ldots,P^{(n)} \rangle$ postaci $P^{(i)} = (S^{(i)}, B^{(i)}, F^{(i)}, T^{(i)})$, gdzie $S^{(i)} = R^n$.

Załóżmy, że istnieje relacja równoważności $Similar \subset R^n \times R^n$, która opisuje pewne klasy „podobnych” stanów. Jej postać zostanie omówiona dalej.

Proces rozpoczynający ciąg opisany jest następującymi zależnościami

1. $S^{(0)} = B^{(0)} = \{(x_0)\}$
2. $F^{(0)} = \begin{cases} B^{(0)} & \text{jeżeli } V_F(x_0) = \varnothing \wedge \forall i \in N . A_i x_0 = 0 \\ \varnothing & \text{w p.p.} \end{cases}$
3. $T^{(0)} = \varnothing$

Przejście od $P^{(i-1)}$ do $P^{(i)}$ może nastąpić jeśli

$$\exists\, s_k \in S^{(i-1)} \setminus F^{(i-1)} \,.\, s_k = (x_k) \wedge \exists\, v \in V_F(y_k) \,.\, s_i = (x_k + v) \notin S^{(i-1)}$$

Wówczas konstruowany jest nowy proces $P^{(i)}$ postaci:

1. $S^{(i)} = S^{(i-1)} \cup \{ s_i \}$, $s_i = (x_k + v)$
2. $B^{(i)} = B^{(i-1)}$
3. $T^{(i)} = T^{(i-1)} \cup \{(s_k, s_i)\}$
4. Zbiór $F^{(i)}$ modyfikowany jest następująco:
   a) Jeśli $V_F(y_i) = \varnothing \wedge \forall\, i \in N \,.\, A_I\, x_i = 0$ to $F^{(i)} = F^{(i-1)} \cup \{ s_i \}$
   b) Jeśli $V_F(y_i) \neq \varnothing$ oraz $\exists\, s_r \in S^{(i-1)} \,.\, (s_i, s_r) \in \mathit{Similar}$, to $F^{(i)} = F^{(i-1)} \cup \{ s_i \}$
   c) Jeżeli nie zachodzi żaden z powyższych przypadków $F^{(i)} = F^{(i-1)}$

■

Omówmy relację *Similar* . Jej zadaniem jest zapewnienie zbieżności ciągu Θ. Definiowana będzie ona w zależności od celu konstrukcji procesu. W szczególnym przypadku może być ona pusta, wówczas, jeśli występują pętle, ciąg Θ jest nieskończony.

Załóżmy, że relacja *Similar* jest zdefiniowana jako $\mathit{Similar} = \mathit{Equal} \cup \mathit{Covers}$, gdzie

a) $\mathit{Equal} = \{(s_1,s_2) \mid s_1 = (x_1), y_1 = y(x_1), s_2 = (x_2), y_2 = y(x_2) \wedge y_1 = y_2 \}$

b) $\mathit{Covers} = \{(s_1,s_2) \mid s_1 = (x_1), y_1 = y(x_1), s_2 = (x_2), y_2 = y(x_2) \wedge (y_1 \prec y_2 \vee y_2 \prec y_1) \}$

Załóżmy, że x_1 jest „nowym” rozwiązaniem, natomiast x_2 rozwiązaniem należącym do osiągniętego wcześniej zbioru.

Przypadek (a) opisuje dojście do stanu y_1, który został wcześniej osiągnięty w innym ciągu rozwiązań. Dalsza generacja procesu w kierunku kolejnych rozwiązań osiągalnych z x_1 jest przerywana, ponieważ można je wyprowadzić z analizy kolejnych rozwiązań następujących po x_2 .

Podobny warunek opisuje (b) ale tutaj zachodzi pewna różnica – w ogólnym przypadku nie wszystkie rozwiązania osiągalne z x_1 są opisywane przez drzewo rozpoczynające się od rozwiązania x_2. Co więcej, w tym przypadku nie istnieje skończony proces opisujący zachowanie modelu [Murata 89; Hack 76].

Wniosek 4-4

Dla zdefiniowanej jak wyżej relacji *Similar* ciąg procesów Θ jest skończony.

■

Uzasadnienie. W tym przypadku stosuje się lemat zamieszczony w [Reisg 85] dotyczący osiągalnych znakowań sieci Petriego. Mówi on, że każdy nieskończony ciąg różnych znakowań sieci Petriego ma nieskończony podciąg silnie rosnący, a zatem każdy zbiór nieuporządkowanych znakowań jest ograniczony.

Wynika z tego, że zbiór osiągalnych parami rozłącznych stanów, czyli takich, że nie zachodzi $y_1 \prec y_2$ lub $y_2 \prec y_1$ jest skończony.

W przypadku specyfikacji weryfikowanej wymagać będziemy, aby zbiór stanów osiągalnych Y był skończony, ponieważ tylko w takim przypadku problem poprawności będzie rozstrzygalny. Zakładamy więc, że przypadek (b) nie będzie zachodził.

W przypadku osiągnięcia stanów pokrywających podczas analizy specyfikacji weryfikowanej następowało będzie odcinanie dalszych gałęzi grafu. Oznacza to, że weryfikacja poprawności będzie jedynie częściowa; wartościowym jej rezultatem będzie więc jedynie negatywny wynik (znalezienie błędów popartych kontrprzykładem).

Podczas konstrukcji algorytmu weryfikacji poprawności opartego o proces sprzężony używana będzie znacznie bardziej złożona relacja *Similar* . Relacja w powyższej postaci jest uznawana za podstawową, przy konstrukcji procesu sekwencyjnego opisującego wykonanie modelu.

Można oczywiście zdać pytanie, dlaczego nie stworzyć procesu, którego stany będą po prostu podzbiorem zbioru wszystkich stanów Y modelu. Taki opis nie będzie wykorzystywany, ponieważ dziedziną analizy jest zbiór akcji, a nie stanów. Potencjalnie, kilka równoległych przejść elementarnych (etykietowanych różnymi symbolami) może łączyć ze sobą pary stanów y_1 i y_2. Zasadą analizy jest jednakże ich rozróżnianie, ponieważ przejściom tym mogą towarzyszyć różne obserwacje.

4.10 Podsumowanie

Zdefiniowany w rozdziale model liniowy traktowany jest jako abstrakcyjna postać formalnej specyfikacji wymagań oraz specyfikacji systemu weryfikowanego. Opis procesów w postaci macierzowej nie powinien wydawać się czymś zaskakującym, biorąc pod uwagę, że zazwyczaj są one strukturami relacyjnymi (grafami), a typową reprezentacją grafów używaną do przetwarzania obliczeniowego jest postać macierzy incydencji [Deo 74].

W rozdziale podano zasady budowy modelu macierzowego dla:

- procesu sekwencyjnego,
- specyficznych elementów występujących w konstrukcji procesu kryterialnego dla oryginalnego modelu poprawności względnej bazującego na stanach,
- systemu komunikujących się procesów.

Następnie przedstawiono algorytm wykonania modelu czyli generacji ciągu rozwiązań osiągalnych począwszy od rozwiązania początkowego.

Postać modelu podobna jest do reprezentacji macierzowej sieci Petriego. Odpowiednikiem akcji (przejścia atomowego) jest w sieciach Petriego łuk, odpowiednikiem przejścia elementarnego tranzycja, odpowiednikiem stanu znakowanie. Różnicą między opisanym modelem i sieciami Petriego jest stopień szczegółowości. W przypadku sieci Petriego wyznaczanie tranzycji jest elementem etapu modelowania systemu, natomiast dla modelu liniowego jest elementem wykonania systemu. Dzięki temu opis procesów w modelu liniowym jest bliższy modelowanej rzeczywistości.

Wprowadzone w dalszych rozdziałach pojęcia poprawności zdefiniowane będą dla par specyfikacji opisanych przez model weryfikowany *ML* i kryterialny *ML'*. Dzięki temu będą pojęciami bardziej ogólnymi i stosującymi się do szerszej klasy systemów, niż gdyby podjąć próbę zdefiniowania ich dla konkretnego języka specyfikacji.

Podobna abstrakcja opisu cechowała specyfikacje występujące w algebraicznym modelu poprawności względnej. Opis ten jednak nie umożliwiał łatwej specyfikacji zależności między akcjami (przejściami) .

Posługując się modelem liniowym korzystamy z tego, że każdemu rozwiązaniu reprezentującemu zbiór semiobliczeń może być jednoznacznie przypisany stan. Dzięki

temu mamy możliwość elastycznego wykorzystania obu pojęć do opisu zależności pomiędzy modelami systemów i ich dowodzenia.

5. Model poprawności dla liniowej funkcji obserwacji

W rozdziale tym zaproponowana zostanie postać zagadnienia poprawności dla homomorficznej funkcji obserwacji. W postawionym problemie poprawności będą występowały trzy obiekty:

- model liniowy specyfikacji weryfikowanej *ML*;
- model specyfikacji kryterialnej *ML'*;
- funkcja obserwacji π opisująca odwzorowanie pomiędzy dwiema warstwami specyfikacji.

W pierwszej części rozdziału zdefiniowane zostaną pojęcia opisujące poprawność. Definicje tych pojęć mają charakter bardziej ogólny i poza definicją poprawności ciągu rozwiązań stosują się także do niehomomorficznej funkcji obserwacji. W kolejnym podrozdziale podane zostaną warunki poprawności dla liniowej funkcji obserwacji.

Weryfikacja poprawności, podobnie, jak dla algebraicznych metod poprawności wzglednej (por. rozdział 2.6) prowadzona będzie w oparciu o konstrukcję procesu sprzężonego. Zaproponowana postać dostosowana jest do zmienionej dziedziny opisu zachowania systemu i postaci wymagań.

5.1 Pojęcia związane z poprawnością

W rozdziale 3 podano definicje liniowej funkcji obserwacji oraz zaprezentowano ideę jej wykorzystania do specyfikacji poprawności. Funkcja ta traktowana jest jako uogólnienie ścieżkowej funkcji obserwacji, ponieważ pozwala na odwzorowanie akcji modelu weryfikowanego w zbiory równolegle wykonanych akcji modelu kryterialnego.

Ścieżkowy model semantyczny procesu jest tu zastąpiony przez ciąg wektorów akcji. Poszczególne współrzędne wektorów podają liczbę akcji wykonanych od początku ścieżki.

Podstawowym pojęciem jest poprawność ciągu rozwiązań modelu weryfikowanego *ML*. Wykorzystując tę definicję wprowadza się poprawność częściową i całkowitą. Dalszymi omawianymi pojęciami są: poprawność potencjalna oraz spójność modelu wymagań.

5.1.1 Definicja częściowej poprawności

Def. 5-1 Poprawność względna ciągu rozwiązań

Dane są dwa modele specyfikacji:

$ML = (X, x_0, A_I\, x \le b, A_E\, x = 0)$ – model specyfikacji weryfikowanej oraz

$ML' = (X', x_0', A_I'\, x' \le b', A_E'\, x' = 0)$ – model specyfikacji kryterialnej.

Dana jest liniowa funkcja obserwacji $\pi : X \rightarrow X'$ zdefiniowana jak w podrozdziale 3.3.2, (Def. 3-5).

Zakładamy, że $\pi(x_0) = x_0'$

Ciąg rozwiązań osiągalnych $s(x_0, \bullet) = \langle x_0, x_1, x_2, \ldots, x_i, x_{i+1}, \ldots \rangle$ jest poprawny, jeśli:

1. Dla każdej pary rozwiązań (x_i, x_{i+1}) przejście w specyfikacji kryterialnej $v_c' = \pi(x_{i+1}) - \pi(x_i)$ spełnia:
   a) $v_c' = 0$ lub

b) $v_c' \in V_F(\pi(x_i))$ i v_c' jest dopuszczalnym równoległym przejściem złożonym w $\pi(x_i)$.

2. Jeżeli ciąg $s(x_0, \bullet)$ jest skończony, wówczas jego ostatnie rozwiązanie x_n jest rozwiązaniem końcowym, czyli spełnia: $V_F(x_n) = \varnothing \wedge \forall i \in N \,.\, A_i\, x_n = 0$ (zdefiniowany w podrozdziale 4.7.4 predykat *final*(x_n)).

3. Jeśli x_n rozwiązaniem końcowym to $x_n' = \pi(x_n)$ spełnia *final*(x_n').

■

Zdefiniujmy predykat *correct*($s(x_0, \bullet)$, π, *ML*'), który przyjmuje wartość prawdy, jeśli ciąg $s(x_0, \bullet)$ jest poprawny względem modelu *ML*'.

Oznaczmy przez S(*ML*) zbiór wszystkich ciągów rozwiązań modelu *ML*.

Def. 5-2 Poprawność względna częściowa

Dane są dwa modele specyfikacji *ML* oraz *ML*' określone jak w Def. 5-1 oraz funkcja obserwacji π.

Model *ML* jest częściowo poprawny względem *ML*' jeżeli:

$$\forall\, s(x_0, \bullet) \in \mathrm{S}(ML) \,.\, correct(s(x_0, \bullet), \pi, ML')$$

■

Definicja żąda więc, aby wszystkie ciągi rozwiązań były poprawne względem kryterium (π, *ML*') określonym przez funkcję obserwacji π i model kryterialny *ML*'.

Definicja ta może być traktowana jako uogólnienie relacji **sat** CSP [Hoare 85] (Por. 2.3.1). Pomijając różnice dotyczące wybranego elementu opisu (akcje zamiast stanów) można także zauważyć jej podobieństwo do relacji symulacji i uściślania zdefiniowanych w podrozdziale 2.5.1.

5.1.2 Dywergencja

Załóżmy, że istnieje ciąg przejść elementarnych $s_c(v_1, v_c)$ tworzący pętlę podstawową; oznaczmy przez $x_i = \sum_{j \le i} v_j$.

Załóżmy, że funkcja π jest funkcją liniową (homomorficzną) oraz załóżmy, że $\pi(x_c) = 0$. Ponieważ $x_i \le x_c$, więc również zachodzi $\pi(x_i) = 0$

Rozważmy ciąg postaci:

$$s(x_0, \bullet) = \langle x_0, \ldots, x_s, x_s + x_1, \ldots, x_s + x_{c-1}, x_s + x_c, x_s + x_c + x_1, \ldots, x_s + x_c + x_{c-1}, x_s + 2\,x_c, \ldots \rangle .$$

Począwszy od rozwiązania x_s funkcja obserwacji π przyjmuje stałą wartość. Oznaczmy $y_s' = y(\pi(x_s))$.

Obserwator posługujący się funkcją π do wyznaczania kolejnych rozwiązań modelu *ML*' dostrzeże, że obliczane są kolejne rozwiązania, a więc nie nastąpił przypadek blokowania.

Załóżmy, że $V_F(y_b') \neq \varnothing$. Obserwator wówczas nie potrafi określić, czy w przyszłości zaobserwowane będą jakiekolwiek przejścia ze zbioru $V_F(y_b')$. Załóżmy, że $V_F(y_b') = \varnothing$ i y_b' jest poprawnym stanem końcowym modelu *ML*'; wówczas obserwator nie będzie mógł w skończonym czasie stwierdzić, czy obserwowany system kiedykolwiek się zatrzyma lub też będzie usiłował wykonać niedopuszczalne przejście.

Oczywiście dla zwykłych przejść w modelu *ML* również obserwator może zanotować powtarzające się wartości funkcji π, jest to naturalne ze względu na mechanizm ukrywania podobny do działania operatora restrykcji „/" w CCS i CSP. Liczba tych powtarzających się wartości jest jednak ograniczona, np.: przez wymiar przestrzeni X. Tu jednak takiego ograniczenia nie ma.

Podobne zjawisko zostało opisane dla formalizmu CSP w [BR 85] oraz [Hoare 85] jako dywergencja (Por. 2.3.1). Wskazanym źródłem dywergencji jest ograniczenie zbioru obserwowalnych operacji przez zastosowanie operatora „/" usuwającego pewien zbiór symboli z alfabetu procesu.

Z punktu widzenia obserwatora zjawisko dywergencji może być utożsamiane z brakiem sprawiedliwości (ang. *fairness*) systemu. Zauważmy, że w przypadku występowania dywergencji istnieje nieskończona obserwacja ciągu rozwiązań dla której pewien zbiór akcji modelu kryterialnego jest stale dopuszczalny, ale żadna z tych akcji nie została wykonana.

Występowanie dywergencji może być stwierdzone w bardzo wielu modelowanych specyfikacjach. Dywergencja pojawiała się będzie zwłaszcza w wyniku nie uwzględniania parametrów czasowych przejść. Przyjęty model wykonania systemu komunikujących się procesów opiera się na operacji przeplotu. Z tego powodu brak jest w nim założeń dotyczących szeregowania procesów, które mogłyby zapewnić sprawiedliwość.

W pracy [KMP 94] (patrz 2.4.1) budując model systemu na potrzeby logiki temporalnej zdefiniowano sprawiedliwy system przejść (ang. *fair transition system*). Pragnąc abstrahować od czasu i mimo tego wyspecyfikować warunki sprawiedliwości założono jawnie, że niedopuszczalne są nieskończone obliczenia procesu, dla których przejścia z pewnego zbioru są stale dopuszczalne (lub dopuszczalne cyklicznie) i są wykonywane jedynie skończoną liczbę razy. Oczywiście, odpowiednio skonstruowane zbiory akcji spełniające powyższe założenia mogą eliminować ciągi rozwiązań zawierające dywergencję jako niedopuszczalne.

Mimo, że nie określone jawnie, na podobnej zasadzie występowanie niektórych dywergencji w weryfikowanych specyfikacjach może być również ignorowane.

Def. 5-3 Dywergencja ciągu rozwiązań

Niech $s(x_0, \bullet) = \langle x_0, x_1, x_2, \ldots, x_i, x_{i+1}, \ldots \rangle$ będzie dowolnym ciągiem rozwiązań osiągalnych. Niech π będzie dowolną funkcją obserwacji. Mówimy, że ciąg ten zawiera dywergencję, jeśli istnieje podciąg $s_{div}(x_{div\,1}, x_{div\,n})$ ciągu $s(x_0, \bullet)$ taki, że $s(x_0, \bullet) = s' \bullet s_{div}(x_{div\,1}, x_{div\,n}) \bullet s''$ i zachodzi:

1. $y(x_{div\,1}) = y(x_{div\,n}) \vee y(x_{div\,1}) \prec y(x_{div\,n})$
2. $\forall\, i : div_1 \le i \le div_n \,.\, \pi(x_i) = \pi(x_{div\,1})$

■

Przez „•" oznaczono operator konkatenacji ciagów. Definicja dywergencji stosuje sie także do nieliniowej funkcji obserwacji.

Zdefiniujmy predykat *divergent*$(s(x_0, \bullet), \pi)$, który przyjmuje wartość prawdy, jeżeli dla danej funkcji obserwacji π ciąg rozwiązań zawiera dywergencję.

5.1.3 Definicja całkowitej poprawności

Def. 5-4 Poprawność względna całkowita

Dane są dwa modele specyfikacji *ML* oraz *ML*' określone jak w Def. 5-1 oraz funkcja obserwacji π.

Model *ML* jest częściowo poprawny względem *ML*' jeżeli jest:

1. Częściowo poprawny:

   $\forall\ s(x_0, \bullet) \in \mathrm{S}(ML)\ .\ correct(s(x_0, \bullet), \pi, ML')$

2. Pokrywa całą przestrzeń X':

   $\forall\ i \le dim\ \mathrm{X}'\ .\ (\exists\ s(x_0,x_\mathrm{k}) \in \mathrm{S}(ML)\ .\ x' = \pi(x_\mathrm{k}) \wedge x_\mathrm{i}' > 0),$

   gdzie *dim* X' oznacza wymiar przestrzeni X' .

3. Nie zawiera dywergencji:

   $\forall\ s(x_0,\bullet) \in \mathrm{S}(ML)\ .\ \neg\ divergent(\ s(x_0,\bullet), \pi\ )$

■

Pewną alternatywą dla warunku (2) powyższej definicji może być żądanie, aby zbiór wszystkich ciągów rozwiązań modelu *ML* pokrył całą *przestrzeń obserwacji* P (Por. 3.4.3) Kwestią umowy jest jak traktować operator $\oplus$ w definicji funkcji obserwacji. Jeśli wskazuje on na alternatywny sposób implementacji (jak operator $\sqcap$ CSP [Hoare 85]) wówczas postać warunku (2) jest wystarczająca. Jeżeli jednak funkcję obserwacji należy traktować jako odwzorowanie pomiędzy istniejącą implementacją a jej specyfikacją – wówczas bardziej adekwatne może być sprawdzanie pokrycia całej przestrzeni obserwacji.

5.1.4 Definicja potencjalnej poprawności

Dla analizy własności systemu interesujące jest pytanie, jak bliski jest on poprawności, czyli czy istnieje pewien podzbiór zbioru rozwiązań, o których wiemy, że spełniają warunki poprawności.

Def. 5-5 Poprawność względna potencjalna

Niech $S \subset \mathrm{S}(ML)$ będzie pewnym zbiorem rozwiązań.

Model *ML* jest potencjalnie poprawny względem *ML*' jeżeli jest:

1. Częściowo poprawny:

   $\forall\ s(x_0, \bullet) \in S\ .\ correct(s(x_0, \bullet), \pi, ML')$

2. Pokrywa całą przestrzeń X'

   $\forall\ i \le dim\ \mathrm{X}'\ .\ (\exists\ s(x_0,x_\mathrm{k}) \in S\ .\ x' = \pi(x_\mathrm{k}) \wedge x_\mathrm{i}' > 0),$

   gdzie *dim* X' oznacza wymiar przestrzeni X'.

■

Poprawność potencjalna niesie pewne istotne informacje: własność ta oznacza, że jeśli do modelu *ML* nie spełniającego warunków poprawności wprowadzimy pewne dodatkowe ograniczenia zacieśniające zbiór rozwiązań osiągalnych – wówczas będzie się on zachowywał poprawnie.

W szczególności, ograniczenia te mogą być niewyrażalne w przyjętym modelu – mogą dotyczyć czasów wykonania poszczególnych operacji lub relacji porządku ustalającej wybór pomiędzy nimi zgodnie z przydziałem priorytetów współbieżnym zadaniom.

5.1.5 Spójność modelu wymagań

Interesujące jest pytanie, czy dany model *ML* jest poprawny dla kryterium (π_I , *ML*), gdzie π_I jest identycznościową funkcją obserwacji. Z definicji wynika, że tak może nie być, ponieważ dla rozwiązań modelu prowadzących do stanów blokujących stwierdzona będzie ich niepoprawność.

Kolejnym interesującym pytaniem jest, czy dla danego modelu *ML*' procesu istnieje taka funkcja obserwacji π oraz model weryfikowany *ML*, że model *ML* jest całkowicie poprawny względem kryterium (π , *ML*'). Pytanie to można sprowadzić do badania czy istnieje podzbiór zbioru rozwiązań $S \subset S(ML')$, taki że nie prowadzi do blokowania oraz pokrywa przestrzeń rozwiązań X'. Problem podobny jest więc do problemu żywotności i braku zakleszczeń.

Def. 5-6 Spójność modelu

Niech $S \subset S(ML)$ będzie pewnym zbiorem rozwiązań, natomiast π_I identycznościową funkcją obserwacji.

Wymagania są *częściowo spójne* jeżeli:

1. $\forall\ s(x_0, \bullet) \in S\ .\ correct(s(x_0, \bullet), \pi_I, ML)$
2. $\forall\ i \le dim\ X'\ .\ (\exists\ s(x_0,x_k) \in S\ .\ x' = \pi(x_k) \wedge x_i' > 0)$,

   gdzie *dim* X' oznacza wymiar przestrzeni X' .

Wymagania są *całkowicie spójne*, jeżeli cały zbiór rozwiązań S(*ML*) spełnia warunki częściowej spójności.

■

5.2 Warunki poprawności

W rozdziale tym przedstawione zostaną twierdzenia określające warunki konieczne poprawności ciągów rozwiązań dla homomorficznej funkcji obserwacji, tj. takiej, że wszystkie wiersze macierzy M opisującej odwzorowanie π_{Min} zawierają co najwyżej jeden element różny od 0.

Twierdzenie 5-1 I warunek konieczny poprawności dla pętli

Załóżmy, że zbiór stanów Y modelu weryfikowanego jest skończony. Niech $s_c(v_1, v_c)$ będzie ciągiem przejść elementarnych tworzącym pętlę podstawową

$$s_c(v_1, v_c) = \langle\ v_1, v_2, \ldots, v_c\ \rangle$$

Oznaczmy przez $x_i = \sum_{j \le i} v_j$, $x_0 = 0$. Zachodzi oczywiście, $A_I\ x_c = 0$.

Rozważmy ciąg nieskończony, który począwszy od skończonego rozwiązania x_s osiągalnego z rozwiązania początkowego składa się z kolejnych przejść należących do pętli $s_c(v_1, v_c)$:

$$s(x_0, \bullet) = \langle x_0, \ldots, x_s, x_s + x_1, \ldots, x_s + x_{c-1},\ x_s + x_c,\ x_s + x_c + x_1, \ldots, x_s + x_c + x_{c-1},\ x_s + 2\,x_c, \ldots\ \rangle.$$

Teza: warunkiem koniecznym poprawności nieskończonego ciągu rozwiązań $s(x_0, \bullet)$ jest, aby:

$$A_I' \, x_c' \leq 0 \text{ , gdzie } x_c' = \pi(x_c)$$

■

Dowód

Rozważmy podciąg ciągu $s(x_0, \bullet)$ postaci $\sigma = \langle x_s, x_s + x_c, x_s + 2\, x_c, \ldots, x_s + m\, x_c, \ldots \rangle$.

Utwórzmy ciąg wartości funkcji obserwacji dla ciągu σ. Ma on postać:

$$\sigma' = \langle \pi (x_s), \pi (x_s) + x_c', \pi (x_s) + 2\, x_c', \ldots, \pi (x_s) + m\, x_c', \ldots \rangle.$$

Niech $y' = y(\pi (x_s))$. Jeżeli podciąg $s(x_0, x_s)$ ciagu $s(x_0, \bullet)$ prowadzący do rozwiązania x_s był poprawny, wówczas $y' \geq 0$.

Załóżmy, że $\exists\, k \,.\, A_k' \, x_c' > 0$, gdzie $k \in I'$. Wówczas istnieje takie $m \geq 1$, że $y_k' - m\, A_k' \, x_c' < 0$, a zatem $y_m' = y(\pi (x_s + m\, x_c))$ jest stanem niedopuszczalnym. Wynika z tego, że istnieje takie przejście w ciągu $s(x_0, \bullet)$, które prowadzi do niedopuszczalnego przejścia po stronie procesu kryterialnego.

■

Powyższy warunek dotyczy poprawności kolejnych przejść. Kolejny warunek dotyczy rozwiazań zawierających pętle i prowadzących do stanu końcowego oraz ich obrazu przez funkcję π .

Twierdzenie 5-2 II warunek poprawności dla pętli

Załóżmy, że zbiór stanów Y dla specyfikacji weryfikowanej *ML* jest skończony.

Niech $s_c(v_1, v_c)$, x_i, x_c oraz $s(x_0, \bullet)$ będą określone tak jak w założeniach poprzedniego twierdzenia (Twierdzenie 5-1). Załóżmy dodatkowo, że $| V_F(x_s) | > 1$ oraz że istnieje ciąg przejść postaci $s_f(v_{f_1} , \; v_{f_k})$ taki, że

$$v_{f_1} \notin Z(s_c(v_1, v_c)) \wedge v_{f_1} \in V_F(x_s)$$

prowadzący do osiągnięcia stanu końcowego. Ciąg ten spełnia więc

$$\forall\, i \in N \,.\, A_i (x_s + x_f) = 0 \wedge V_F(x_s + x_f) = \varnothing \,,$$

$$\text{gdzie } x_f = \sum_{j \leq k} v_{f_k} \,.$$

Teza: warunkiem koniecznym poprawności dowolnego ciągu rozwiązań generowanego przez ciąg przejść postaci

$$\rho(m) = s_s(\bullet, \bullet) \bullet (s_c(v_1, v_c))^m \bullet s_f(v_{f_1} , \; v_{f_k}), \text{ gdzie } m \geq 0$$

jest, $A_I' \, x_c' = 0$, gdzie $x_c' = \pi(x_c)$, $s_s(\bullet, \bullet)$ oznacza ciąg przejść elementarnych generujących ciąg rozwiązań $s(x_0, x_s)$.

■

Dowód. Zbiór stanów Y jest skończony, więc jeśli istnieje ciąg prowadzący do stanu końcowego, wówczas istnieje taki ciąg skończony, a więc wektor x_f jest ograniczony.

Założenie, że $|V_F(x_s)| > 1$ i ciąg $s_f(v_{f_1}, v_{f_k})$ jest ciągiem prowadzącym do stanu końcowego jest wystarczająco ogólne, ponieważ rozwiązanie x_s może być osiągane w wyniku wykonania części przejść należących już do pętli.

Załóżmy, że ciąg rozwiązań generowany przez $\rho(0) = s_s(\bullet, \bullet) \bullet s_f(v_{f_1}, v_{f_k})$ jest poprawny. Wówczas spełnione jest

$$\forall\, i \in N' .\ A_i' \pi(x_a + x_f) = 0 \wedge V_F(\pi(x_s + x_f)) = \varnothing, \text{ (i)}$$

Rozważmy obraz ciągu rozwiązań generowany przez $\rho(m)$, gdzie $m \geq 1$. Oczekujemy, że spełnione jest

$$\forall\, m \geq 1 .\ \forall\, i \in N' .\ A_i' \pi(x_b + m{\cdot}x_c + x_f) = 0 . \text{ (ii)}$$

Obliczmy wartość $\pi(x_b + m{\cdot}x_c + x_f)$ jako:

$$\pi(x_b + m{\cdot}x_c + x_f) = \pi(x_b + x_f) + m\, \pi(x_c) \text{ (iii)}$$

Wartość y' dla obrazu rozwiązania końcowego wyrażona jest zależnością:

$$y' = b' - A_I'(\pi(x_b + m{\cdot}x_c + x_f)) = -\, m\, A_I' \pi(x_c)$$

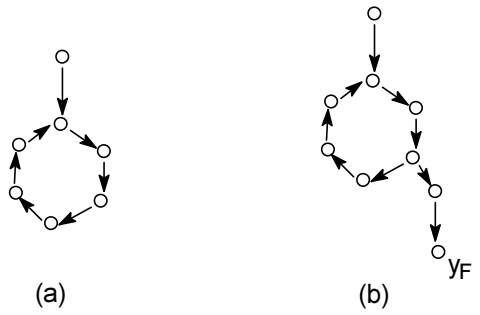
Stąd, jeżeli $\exists\, i \in N' .\ A_i' \pi(x_c) < 0$, wówczas $y' \geq 0$, a więc nie jest poprawnym stanem końcowym.

■

W przypadku, kiedy nie istnieje droga prowadząca do stanu końcowego, wówczas dla poprawności ciągu zawierającego pętle nie jest wymagane, spełnienie II warunku pętli (Twierdzenie 5-2). Wraz z wykonywaniem pętli możemy więc osiągać kolejne stany pokrywające

$$y(\pi(x_s)) < y(\pi(x_s + \cdot x_c)) \prec \ldots \prec y(\pi(x_s + m{\cdot}x_c)) \ldots$$

modelu specyfikacji kryterialnej (co jest poprawne jedynie w przypadku, kiedy zbiór stanów Y' jest nieskończony). Aby jednak taki ciąg mógł wystąpić, musi istnieć współrzędna stanu związana z węzłem o nieograniczonej pojemności. Oznacza to, że przejścia w specyfikacji kryterialnej wychodzące z tego węzła nie będą osiągane, a więc na ogół warunki całkowitej poprawności nie będą zachowane.



Rys. 5-1 Dwa przypadki pętli (a) bez rozgałęzienia, (b) z rozgałęzieniem prowadzącym do stanu końcowego

Warunki wystarczające braku poprawności związanych z osiąganiem stanów pokrywających w dla modelu procesu kryterialnego określa następujące twierdzenie.

Twierdzenie 5-3 Warunek wystarczający braku poprawności dla stanów pokrywających

Niech $s_1(x_0, x_n) = \langle x_0,\ldots,x_k,\ldots, x_n \rangle$ będzie dowolnym poprawnym ciągiem rozwiązań prowadzącym do stanu końcowego, czyli spełnione jest *final*(x_n) oraz *final*($\pi(x_n)$).

Niech $s_2(x_0, x_r)$ będzie ciągiem rozwiązań, dla którego spełnione jest:

1. $y(x_k) = y(x_r)$
2. $y(\pi(x_k)) < y(\pi(x_r))$

Teza: Istnieje niepoprawne rozwiązanie osiągalne z x_r.

■

Dowód. Rozważmy ciąg przejść elementarnych postaci $s(v_1,v_{n-k})$ takich, że $v_i = x_{k+i+1} - x_{k+i}$. Ciąg ten jest dopuszczalnym ciągiem kolejnych przejść następujących po osiągnięciu obu rozwiązań. Oznaczmy $x_i = \sum_{j \le n-k} v_j$, przez x_{n-k} wartość $x_{n-k} = x_n - x_k$.

Rozszerzmy ciąg $s_2(x_0, x_r)$ do ciągu $s_{2n}(x_0, x_{rn})$ takiego, że

$$s_{2n}(x_0, \bullet) = s_2(x_0, x_r) \bullet s_2(x_{r+1}, x_{r+n-k}), \text{ gdzie } x_{r+i} = x_r + x_i.$$

Rozwiązanie x_{r+n-k} jest również rozwiązaniem końcowym, ponieważ dla każdego $1 \le i \le n\text{-}k$ zachodzi $y(x_{r+i}) = y(x_{k+i})$.

Zauważmy, że jeżeli ciąg $s_1(x_0, x_n)$ był poprawny, wówczas $\forall\, i \in N'$. b' - A_I' $\pi(x_n) = 0$, stąd zachodzi $\forall\, i \in N'$. $y(\pi(x_k))$ - A_I' $\pi(x_{n-k}) = 0$.

Jeżeli $y(\pi(x_k)) \prec y(\pi(x_r))$, to $\exists\, i \in N'$. $y_k' < y_r'$, gdzie $y_k' = y(\pi(x_k))$ i $y_r' = y(\pi(x_r))$, stąd również y_k' - A_I' $\pi(x_{n-k}) < y_r'$ - A_I' $\pi(x_{n-k})$, a więc w rozwiązaniu końcowym x_{r+n-k} będzie istniała niezerowa składowa $y(\pi(x_{r+n-k}))$.

■

5.3 Weryfikacja poprawności dla liniowej funkcji obserwacji

Podobnie jak dla oryginalnego modelu poprawności względnej weryfikacja poprawności oparta jest o konstrukcję procesu sprzężonego opisującego wspólne wykonanie specyfikacji weryfikowanej i kryterialnej. Analiza własności tego procesu pozwoli określić, całkowitą, częściową lub potencjalną poprawność procesu weryfikowanego.

5.3.1 Konstrukcja procesu sprzężonego

Niech *ML* = (X, x_0, A_I $x \le b$, A_E $x = 0$) będzie modelem liniowym specyfikacji. Oznaczmy prze Y przestrzeń stanów modelu ML. Zakładamy, że zbiór Y jest skończony.

Podobnie niech *ML*' = (X', x_0', A_I' $x' \le b'$, A_E' $x' = 0$) będzie modelem specyfikacji kryterialnej, Y' jego przestrzenią stanów. Niech π oznacza liniową funkcję obserwacji. Zakładamy, że $\pi(x_0) = x_0'$.

Proces sprzężony postaci $\hat{P} = (\hat{S}, \hat{B}, \hat{F}, \hat{T})$ jest skonstruowany jako granica ciągu procesów $\Theta = \langle P^{(0)},\ldots, P^{(i)},\ldots, P^{(n)},\ldots\rangle$ postaci $P^{(i)} = (S^{(i)}, B^{(i)}, F^{(i)}, T^{(i)})$, gdzie $S^{(i)} \subset X \times \mathbf{Y}$, $\mathbf{Y} = Y \times Y'$.

Spełnione jest więc:

1. $\hat{S} = \bigcup_{i=0}^{\infty} S^{(i)}$

2. $\hat{B} = \bigcup_{i=0}^{\infty} B^{(i)}$

3. $\hat{F} = \bigcup_{i=0}^{\infty} F^{(i)}$

4. $\hat{T} = \bigcup_{i=0}^{\infty} T^{(i)}$

Poszczególne stany procesu będą zazwyczaj zapisywany jako pary postaci (x , (y , y')).

Proces rozpoczynający ciąg opisany jest następującymi zależnościami

1. $S^{(0)} = B^{(0)} = \{ (x_0 , (y_0 , y_0'))\}$, gdzie $y_0 = y(x_0)$, $y_0' = y(x_0')$

2. $F^{(0)} = \begin{cases} B^{(0)} & \text{jeżeli } V_F(x_0) = \varnothing \wedge \forall i \in N . A_i x_0 = 0 \\ \varnothing & \text{w p.p.} \end{cases}$

3. $T^{(0)} = \varnothing$

Przejście od $P^{(i-1)}$ do $P^{(i)}$ następuje jeśli

$\exists\, s_k \in S^{(i-1)} \setminus F^{(i-1)} . s_k = (x_k, (y_k, y_k')) \wedge \exists\, v \in V_F(y_k) . s_i = ((x_k + v), (y_i, y_i')) \notin S^{(i-1)}$, gdzie $y_i = y(x_k + v)$, $y_i' = y(\pi(x_k + v))$

Wówczas konstruowany jest nowy proces $P^{(i)}$ postaci:

1. $S^{(i)} = S^{(i-1)} \cup \{ s_i \}$, $s_i = ((x_k + v), (y_i, y_i'))$
2. $B^{(i)} = B^{(i-1)}$
3. $T^{(i)} = T^{(i-1)} \cup \{(s_k, s_i)\}$
4. Zbiór $F^{(i)}$ modyfikowany jest następująco:
   a) Jeśli $V_F(y_i) = \varnothing \wedge \forall\, i \in N . A_I x_i = 0$ to $F^{(i)} = F^{(i-1)} \cup \{ s_i \}$
   b) Jeśli $V_F(y_i) \neq \varnothing$ oraz istnieje $s_r \in S^{(i-1)}$, $s_r = (x_r,(y_r, y_r'))$ spełniające: $(y_r = y_i) \wedge (y_r' \leq y_i')$ to $F^{(i)} = F^{(i)} \cup \{ s_i \}$
   c) Jeśli $V_F(y_i) \neq \varnothing$ oraz istnieje $s_r \in S^{(i-1)}$, $s_r = (x_r,(y_r, y_r'))$ spełniające: $(y_r \prec y_i) \wedge (y_r' \leq y_i')$ to $F^{(i)} = F^{(i)} \cup \{ s_i \}$
   d) Jeżeli nie zachodzi żaden z powyższych przypadków $F^{(i)} = F^{(i-1)}$

■

Zauważmy, że z założenia o skończoności zbioru stanów Y wynika, że warunek 4.c nigdy nie będzie zachodził. Trudno przed przystąpieniem do konstrukcji procesu sprzężonego to sprawdzić, więc warunek ten badany jest w algorytmie. Specyfikację, dla której zachodzi powyższy warunek traktujemy jako częściowo weryfikowalną.

Def. 5-7 Poprawność przejścia procesu sprzężonego

Niech $(s_k, s_r) \in T^{(i)}$ będzie przejściem w procesie $P^{(i)}$. Oznaczmy $s_k = (x_k, (y_k, y_k'))$ oraz $s_r = (x_r,(y_r, y_r'))$. Oczywiście zachodzi $v = (x_r - x_k) \in V_F(y_k)$.

Mówimy, że przejście (s_k, s_r) jest poprawne, jeżeli $v_c' = \pi(x_r) - \pi(x_k)$ spełnia $v_c' = 0$ lub v_c' jest dopuszczalnym przejściem złożonym w y_k' (czyli także w $\pi(x_k)$)

■

Twierdzenie 5-4 Skończoność procesu sprzężonego

Proces sprzężony $\hat{P}$, którego każde przejście jest poprawne jest skończony.

■

Dowód. Zauważmy, że dowolne semiobliczenie początkowe procesu sprzężonego $sc(s_0,\bullet)$ jest wyznaczane przez pewien podciąg Θ' ciągu Θ postaci:

$$\Theta' = \langle P^{(sc\ 0)}, P^{(sc\ 1)},\ldots, P^{(sc\ i)},\ldots, P^{(sc\ n)},\ldots\rangle$$

taki, że $P^{(sc\ 0)} = P^{(0)}$. Współrzędne x_i kolejnych stanów semiobliczenia $sc(s_0,\bullet)$ tworzą ciąg rozwiązań osiągalnych.

Wykażemy, że każde semiobliczenie początkowe procesu sprzężonego jest skończone (czyli jest obliczeniem). Niech $\Gamma = \langle (y_0, y_0'),\ldots, (y_i, y_i'),\ldots \rangle$ będzie ciągiem składowych $Y \times Y'$ semiobliczenia $sc(s_0,\bullet)$. Z poprawności wszystkich przejść wynika, że $\forall\, i\,.\, (y_i \geq 0 \wedge y_i' \geq 0)$.

Aby ciąg Γ był nieskończony, wówczas na podstawie lematu dotyczącego znakowań sieci Petriego [Reisg 85] musi zawierać silnie rosnący podciąg nieskończony Γ'. Niech $r = Dim\ Y + Dim\ Y'$. Zatem wśród co najwyżej $r + 2$ wyrazów ciągu Γ' musi pojawić się para (y_s, y_s') oraz (y_t, y_t') taka, że $s < t$ oraz $[y_s; y_s']^T \prec [y_t; y_t']^T$. Taki przypadek jest jednak wykryty przez warunki 4.b i 4.c, a zatem ciąg Γ jest skończony.

Ponieważ każde semiobliczenie procesu sprzężonego jest skończone i reprezentuje drogę w grafie acyklicznym, stąd na podstawie lematu Koeniga w [Reisg 85] proces sprzężony jest skończony.

Wynika stąd bezpośrednio, że ciąg procesów Θ używany do wyznaczania procesu sprzężonego jest również skończony .

■

5.3.2 Zastosowanie procesu sprzężonego w weryfikacji poprawności

Wniosek 5-1

Z zasad konstrukcji procesu sprzężonego wynika, że jeśli każde jego przejście jest poprawne, wówczas spełniony jest następujący warunek:

$$\forall\, s_k \in (\hat{S} \setminus \hat{F}) : s_k = (x_k,(y_k, y_k')) \,.\, \forall v \in V_F(y_k) \,.\, \exists\, s_r \in \hat{S} \,.\, s_r = (x_r,(y_r, y_r')),$$

spełniające $y_r = y_k - A_I\, v$ i $y_r' = y_k' - A_I'\, \pi(v)$.

■

Def. 5-8 Relacja podobieństwa rozwiązań

Zdefiniujmy relację $<\!\approx \subset X \times X$ jako:

$x_1 <\!\approx x_2 \Leftrightarrow \quad y_1 = y_2 \wedge y_1' \leq y_2'$,

gdzie $y_i = y(x_i), y_i' = y(\pi(x_2))$

■

Relacja $<\!\approx$ jest zwrotna i przechodnia, ponieważ relacje = i ≤ są zwrotne i przechodnie.

Wniosek 5-2

1. Jeśli $x_1 <\approx x_2$ to $V_F(x_1) = V_F(x_2)$ oraz zachodzi $\forall\ v \in V_F(x_1)\ .\ (x_1 + v\ ) <\approx (x_2 + v)$.
2. Jeżeli $\pi(v)$ było dopuszczalnym przejściem równoległym w y_1' to $\pi(v)$ jest również dopuszczalnym przejściem równoległym w y_2'.

■

Uzasadnienie

Równość $y(x_1 + v\ ) = y(x_2 + v)$ zachodzi, ponieważ $y(x_i + v\ ) = y(x_i) - A_I\ v$. Jeżeli $y_1' \le y_2'$ stąd zachodzi $y_1' - \Delta y' \le y_2' - \Delta y'$, gdzie $\Delta y' = A_I'(\pi(v))$.

Prawdziwość stwierdzenia (2) wynika bezpośrednio z tego, że $y_1' \le y_2'$.

■

Oznaczmy $\hat{F}_* = \{\ s_r \in \hat{F}\ |\ s_r = (x_r,(\ y_r, y_r'))\ \wedge\ V_F(y_r) \neq \varnothing\ \}$. Elementami tego zbioru są stany, które zostały osiągnięte co najmniej dwukrotnie w trakcie generacji ciągu procesów lub stany je pokrywające.

Wniosek 5-3

1. Zbiór $\hat{F}_*$ spełnia następujący warunek:

$$\forall\ s_f = (x_f,(\ y_f, y_f')) \in \hat{F}_*\ .\ \exists\ s_k = (x_k,(\ y_k, y_k')) \in \hat{S} \setminus \hat{F}\ .\ x_k <\approx x_f$$

2. Wynika stąd, że jeśli przejście $(s_k, s_{k+1}) \in \hat{T}$ jest poprawne, gdzie

$$(s_k, s_{k+1}) = (\ (x_k,(\ y_k, y_k'))\ ,\ (x_{k+1},(\ y_{k+1}, y_{k+1}'))\ ),$$

$$x_{k+1} = x_k + v,$$

wówczas przejście z rozwiązania x_f do rozwiązania $x_f + v$ jest poprawne oraz spełnione jest:

$$x_k + v <\approx x_f + v.$$

■

Uzasadnienie

Stwierdzenie (1) wynika bezpośrednio z zasad konstrukcji procesu sprzężonego.

Z równości stanów y_f i y_k wynika, że zbiory przejść dopuszczalnych są identyczne. $V_F(y_f) = V_F(y_k)$. Jeśli $\pi(v) = 0$, wówczas przejście to jest poprawne, jeśli $\pi(v) \neq 0$, wówczas wektor $y_{fv}' = (\ \pi(\ x_f + v)\ ) = y_f' + (y_{k+1}' - y_k')$ spełnia warunek:

$y_{k+1}' = y_{fv}'$, jeśli $y_k' = y_f'$ albo

$y_{k+1}' < y_{fv}'$, jeśli $y_k' < y_f'$.

W obu przypadkach, jeśli $v_c' = \pi(v)$ było dopuszczalnym przejściem złożonym w y_k', to będzie dopuszczalnym przejściem złożonym w y_f'.

■

Twierdzenie 5-5

Jeśli zbiór osiągalnych stanów Y modelu *ML* jest skończony i proces sprzężony zawiera wszystkie przejścia poprawne, to dla każdego rozwiązania osiągalnego x_i zachodzi:

$$\exists\, s_k \in \hat{S} : s_k = (x_k,(\, y_k, y_k')\,.\, x_k <\approx x_i$$

■

Dowód. Jeśli rozwiązanie x zostało wyznaczone podczas konstrukcji procesu sprzężonego, wówczas ta zależność zachodzi.

Rozważmy dowolny ciąg rozwiązań osiągalnych, który zawiera rozwiązania nie wyznaczone przez proces sprzężony. Jego podciąg początkowy musi tworzyć semiobliczenie procesu sprzężonego. Stąd załóżmy, że ma on postać:

$$s(x_0,\bullet) = \langle\, x_0,\ldots, x_f, x_{f+1},\ldots, x_{f+i-1}, x_{f+i},\ldots \,\rangle$$

Załóżmy, że x_f jest ostatnim rozwiązaniem należącym do ciągu rozwiązań wyznaczonego przez semiobliczenie procesu sprzężonego, wówczas na mocy poprzedniego wniosku (Wniosek 5-3) warunek określony w tezie zachodzi dla x_{f+1}.

Załóżmy, że rozwiązanie x_{f+i-1} spełnia warunek określony w tezie, czyli istnieje stan procesu sprzężonego $s_k = (x_k,(\, y_k, y_k')$ spełniający $x_k <\approx x_{f+i-1}$.

Jeżeli stan $s_k \in \hat{F}_*$ wówczas istnieje również stan $s_r = (x_r,(\, y_r, y_r') \in \hat{S} \setminus \hat{F}$ spełniający $x_r <\approx x_k$ stąd $x_r <\approx x_{f+i-1}$, a więc możemy się ograniczyć do przypadku, kiedy $s_k \in \hat{S} \setminus \hat{F}_*$.

Jeżeli $V_F(y_k) = \varnothing$, wówczas na mocy równości stanów y_k oraz y(x_{f+i-1}) przejście do następnego rozwiązania jest niemożliwe.

Jeżeli $V_F(y_k) \neq \varnothing$, wówczas przejście $v = x_{f+i} - x_{f+i-1}$ musi spełniać $v \in V_F(x_k) = V_F(x_{f+i-1})$. Stąd rozwiązanie $x_r = (x_k + v)$ jest wyznaczone w trakcie konstrukcji procesu sprzężonego (Wniosek 5-1), a zatem istnieje stan procesu sprzężonego $s_r = (x_r,(\, y_r, y_r'')) \in \hat{S}$ dla którego zachodzi $x_r <\approx x_{f+i}$.

■

Twierdzenie 5-6

Z poprawności wszystkich przejść procesu sprzężonego wynika, że dla dowolne przejście elementarne pomiędzy dwoma kolejnymi rozwiązaniami osiągalnymi model *ML* jest poprawne w sensie kryterium (π, *ML'*)

■

Dowód. Rozważmy parę kolejnych rozwiązań osiągalnych (x_{f+i-1} ,x_{f+i}). Na mocy poprzedniego twierdzenia, istnieje stan procesu sprzężonego $s_k \in \hat{S} \setminus \hat{F}_*$, $s_k = (x_k, (y_k, y_k'')$, dla którego zachodzi: $x_k <\approx x_{f+i-1}$, stąd zgodnie z wnioskiem 5-2 przejście do rozwiązania x_{f+i} jest poprawne.

■

5.3.3 Sprowadzalność do stanu końcowego

Zgodnie z II warunkiem poprawności dla pętli (Twierdzenie 5-2), jeśli $\exists\, s_f \in \hat{F}_*\,.\, s_f = (x_f, (\, y_f, y_f''))) \wedge \exists\, s_k \in \hat{S}\,.\, s_k = (x_k,(\, y_k, y_k'))$ spełniające $y_f = y_k \wedge y_f'' > y_k''$ i stan końcowy jest osiągalny z y_k na drodze $s_f(\bullet\,,\bullet)$ to na tej drodze z y_f'' nie da się osiągnąć stanu końcowego modelu *ML'*.

Stąd w algorytmie badania poprawności znaczy się stany procesu sprzężonego, z których można osiągnąć stan końcowy i sprawdza dodatkowy warunek czy istnieją stany y_i, z

których można osiągnąć stan końcowy, którym odpowiadają pokrywające się wartości składowej stanu procesu sprzężonego: $(\bullet,(y_i, y_{i1}'))$, $(\bullet,(y_i, y_{i2}')) \in \hat{S} \wedge y_{i1}' \prec y_{i2}'$.

5.4 Podsumowanie

W rozdziale opisano model poprawności względnej dla liniowej funkcji obserwacji. Przedmiotem analizy jest ciąg rozwiązań modelu weryfikowanego *ML* . Zadaniem funkcji obserwacji jest przekształcenie rozpatrywanego ciągu rozwiązań w jego *obserwację*, czyli w ciąg wektorów należących do dziedziny rozwiązań modelu kryterialnego *ML'*.

Wymagania poprawnościowe zastosowane do ciągu będącego obserwacją modelu weryfikowanego żądają, by po usunięciu z niego powtarzających się wartości oraz rozwinięciu przejść złożonych modelu *ML'* w dowolne permutacje składowych przejść elementarnych otrzymać ciąg rozwiązań osiągalnych modelu kryterialnego.

Poprawność częściową analizowanego systemu zdefiniowano jako poprawność wszystkich ciągów jego rozwiązań. Wymagania poprawnościowe dopuszczają powtarzające się wartości rozwiązań obliczanych za pośrednictwem funkcji obserwacji. W ten sposób zdefiniowana poprawność częściowa jest własnością niezmienniczą w przypadku powtórzeń (ang. *invariant under stuttering*). Własnością tą zazwyczaj charakteryzuje się modele logiki temporalnej opisujące badane systemy [Lamport 83] .

Modele dopuszczające niezmienniczość w przypadku powtórzeń są bardziej kłopotliwe z punktu widzenia weryfikacji własności żywotności (a więc całkowitej poprawności). Problemem jest, że obserwując powtarzające się stany modelu (tu rozwiązania) nie mamy pewności, czy ciąg ten jest skończony czy nieskończony.

Możliwość wystąpienia nieskończonego ciągu powtarzających się rozwiązań w obserwacji nazwana została *dywergencją*. Nazwa została zapożyczona z CSP [Hoare 85] ze względu na podobne źródła jej występowania – nieobserwowalność akcji wchodzących w skład pętli. Występowanie dywergencji może być traktowane jak brak sprawiedliwości.

Zaproponowanymi warunkami poprawności całkowitej są:

- poprawność częściowa
- brak dywergencji
- zaobserwowanie wszystkich akcji zdefiniowanych w modelu kryterialnym.

Dalszymi zaproponowanymi pojęciami są poprawność potencjalna (czyli spełnienie warunków poprawności przez podzbiór zbioru wszystkich ciągów modelu weryfikowanego) oraz spójność wymagań modelu *ML*' (rozstrzygająca o istnieniu modelu weryfikowanego *ML* poprawnego względem *ML'*).

Opisana metoda badania poprawności jest inspirowana metodą stworzoną dla modelu poprawności względnej bazującego na stanach i relacji kryterialnej [Szmuc 89a, 89b, 91, 93]. Oparta jest ona na konstrukcji procesu sprzężonego opisującego wspólne wykonanie modelu weryfikowanego i kryterialnego. Ze względu na różnice rozpatrywanych modeli postać procesu sprzężonego jest zasadniczo odmienna od postaci zdefiniowanej dla modelu bazującego na stanach.

Udowodniono, że skonstruowanie procesu sprzężonego, którego wszystkie przejścia spełniają warunki poprawności jest wystarczające do wnioskowania o poprawności wszystkich przejść w ciągach rozwiązań modelu weryfikowanego.

Zaimplementowane algorytmy badania spójności wymagań i weryfikacji poprawności podane są w Dodatku A oraz B. Algorytm weryfikacji jest w istocie rozwinięciem algorytmu badania spójności wymagań. Obejmuje on odwzorowanie wyznaczonych rozwiązań modelu weryfikowanego za pośrednictwem funkcji obserwacji oraz sprawdzanie warunków poprawności przejścia w równolegle rozwijanym ciągu rozwiązań modelu kryterialnego.

Na podkreślenie zasługuje fakt, że przedmiotem weryfikacji nie jest proces sekwencyjny opisujący badany system lecz model zbioru komunikujących się procesów współbieżnych. Dzięki temu modele weryfikowane i kryterialne są bliskie rzeczywistym specyfikacjom tworzonym w trakcie prac nad budową oprogramowania. Pojęciem podstawowym występującymi w specyfikacjach są akcje, które mogą być utożsamiane z wywołaniem funkcji, wykonaniem instrukcjami, przesyłaniem zdarzeń.

Jako typowy obszar zastosowań można wskazać problem weryfikacji koncepcji działania systemu zdekomponowanego na współbieżne elementy składowe. Zazwyczaj w takim przypadku dysponujemy definicjami zdarzeń wejściowych i wyjściowych poszczególnych komponentów, specyfikacjami procesów opisujących poszczególne komponenty oraz definicją struktury systemu (asocjacje między komponentami związane z przesyłaniem komunikatów). Tak jest np.: w przypadku dekompozycji prowadzonej za pomocą metody SART [WM 85; Yourdon 88; Perez 90], podobnie prowadzone są opisy zachowania systemu w metodach obiektowych [CY 1990].

Projektując system zazwyczaj opisuje się scenariusze współdziałania komponentów opisujące przesyłane zdarzenia. Zbiór tych scenariuszy składający się na koncepcję działania systemu zazwyczaj będzie pełnić rolę specyfikacji kryterialnej. Naszkicowaną tutaj ideę specyfikacji wymagań zaprezentowano szerzej w pracach [SSSS 96; SS 96; 97a].

6. Model poprawności dla niehomomorficznej funkcji obserwacji

Definicję pojęć związanych z poprawnością dla niehomomorficznej funkcji obserwacji poprzedzimy dyskusją wprowadzającą, której celem jest przedstawienie intuicji i założeń, które miały wpływ na budowę modelu poprawności. Pojęcia występujące w modelu poprawności dla niehomomorficznej funkcji obserwacji omówione zostaną w sposób opisowy i porównawczy; ich zastosowanie pokazane zostanie na przykładach.

Formalna definicja poprawności zostanie wprowadzona w następnym podrozdziale.

Kolejny podrozdział omawiał będzie zagadnienia związane z weryfikacją poprawności dla niehomomorficznej funkcji obserwacji. Opisana zostanie konstrukcja procesu sprzężonego służącego do badania poprawności oraz przedstawione będą twierdzenia uzasadniające zastosowanie zaproponowanej postaci procesu sprzężonego w dowodzeniu poprawności.

6.1 Dyskusja poprawności dla niehomomorficznej funkcji obserwacji

6.1.1 Rozszerzenie możliwości specyfikacji odwzorowania wymagań dla niehomomorficznej funkcji obserwacji

W rozdziale 3 zdefiniowano ogólną postać niehomomorficznej funkcji obserwacji (Def. 3-8). Zasadniczą jej cechą jest możliwość wyrażenia wymagań dotyczących dekompozycji akcji modelu kryterialnego na zbiory obserwowalnych akcji modelu weryfikowanego.

Niehomomorficzna funkcja obserwacji pozwala na zapis odwzorowania pomiędzy modelem powstałym w wyniku dekompozycji hierarchicznej a modelem warstwy bardziej abstrakcyjnej. Struktura tej dekompozycji jest jednak bardziej złożoną konstrukcją niż zastosowanie prostej operacji syntaktycznej polegającej na zastąpieniu wybranego przejścia z góry określonym ciągiem przejść.

Postać niehomomorficznej funkcji obserwacji:

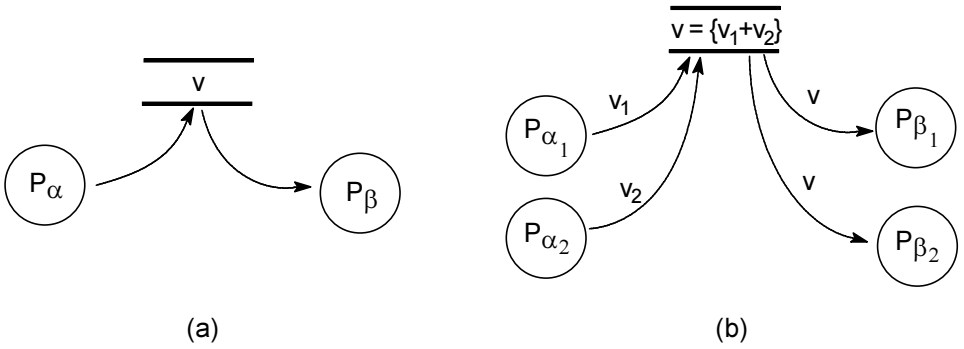
- nie narzuca ograniczeń na kolejność wykonywanych akcji składowych modelu weryfikowanego;
- pozwala na to, aby daną akcję modelu weryfikowanego traktować jako komponent wielu akcji modelu specyfikacji kryterialnej reprezentujących równolegle przetwarzane własności;
- umożliwia dekompozycję danej akcji kryterialnej na zbiór akcji wykonywanych przez różne procesy wchodzące w skład specyfikacji weryfikowanej.

Implikacją zastosowania niehomomorficznej funkcji obserwacji jako elementu wymagań, jest zezwolenie na niedeterminizm przy wyborze sposobu implementacji akcji procesu kryterialnego. Forma niedeterminizmu jest podobna do działania operatora „⊓” CSP [Hoare 85].

Przykład 6-1

Rozważmy prosty przykład specyfikacji DFD przedstawiony na Rys. 6-1a. Proces oznaczony jako P_α oblicza magazynowaną wartość v; natomiast proces P_β używa rezultatu obliczeń.

Rys. 6-1b pokazuje rezultaty dekompozycji. Dana v zdefiniowana jest w słowniku danych jako rekord złożony z pól v_1 oraz v_2. Proces P_α zdekomponowany jest na podprocesy P_{α_1} oraz P_{α_2} odpowiednio obliczające v_1 oraz v_2; natomiast P_β jest podzielony na procesy P_{β_1} oraz P_{β_2} używające obliczonej przez proces P_α wartości danej v .



Rys. 6-1 Dwa poziomy specyfikacji z użyciem diagramów DFD

Z postaci diagramu DFD wynika, że procesy wchodzące w skład par $(P_{\alpha_1}, P_{\alpha_2})$ i $(P_{\beta_1}, P_{\beta_2})$ mogą działać niezależnie. Konkretna implementacja narzuci im pewne uporządkowanie, lecz nie jest ono istotne z punktu widzenia zgodności z wymaganiami. Stąd, oznaczając przez *act* akcję związaną z wykonaniem procesu P_{act} możemy wyspecyfikować funkcję obserwacji dla rozważanego systemu jako:

$\alpha = \alpha_1 \circ \alpha_2$

$\beta = \beta_1 \circ \beta_2$

6.1.2 Kolejność operacji

Definicja poprawności dla omówionej w podrozdziale 2.6 oryginalnej postaci modelu poprawności względnej bazującego na stanach w sposób nieformalny określa wymagania poprawnościowe, jako *osiąganie stanów charakterystycznych w kolejności zgodnej z relacją przejścia procesu kryterialnego*. Podobne własności cechują poprawność dla modelu opartego na homomorficznej (liniowej) funkcji obserwacji. Obserwowane akcje powinny być wykonywane w kolejności zgodnej z akcjami (przejściami) procesu kryterialnego.

Dla funkcji homomorficznej wykonanie obserwowalnej akcji procesu weryfikowanego pociąga za sobą natychmiastowe zaobserwowanie akcji procesu kryterialnego. Dla poprawnego systemu ich kolejność jest więc również zgodna z relacją przejścia procesu sekwencyjnego opisującego zachowanie procesu kryterialnego (z dokładnością do przejść równoległych, które mogą być wyrażone jako dowolna permutacja przejść elementarnych).

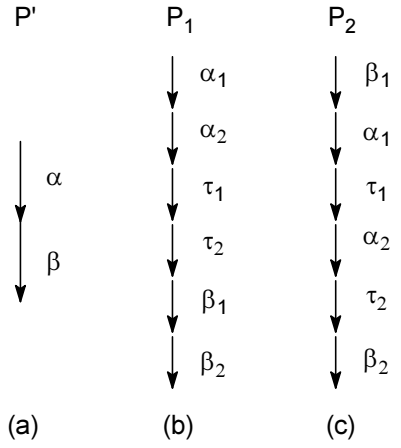
Podstawową różnicą pomiędzy zachowaniem niehomomorficznej i homomorficznej funkcji obserwacji jest brak *natychmiastowości* odwzorowania. Zaobserwowanie danej akcji procesu kryterialnego może się wiązać z uprzednim wykonaniem kilku akcji procesu weryfikowanego, które są rozłożone w czasie.

Rozważany model poprawności abstrahuje od czasu, ograniczając się do założenia o uporządkowaniu czasowym przejść. Niemniej, można postawić fundamentalne pytanie, które w istotny sposób rzutuje na to, jakie przejawy zachowania weryfikowanego systemu będziemy uważać za poprawne.

Czy postać procesu kryterialnego powinna narzucać jakieś ograniczenia na zbiór momentów, w których mogą być wykonywane akcje procesu weryfikowanego ze zbioru, który jest odwzorowywany na wybraną akcję procesu kryterialnego? Czy struktura specyfikacji kryterialnej powinna rzutować na wzajemne relacje pomiędzy porządkiem wykonania akcji z poszczególnych zbiorów ?

Konstruując model poprawności dla niehomomorficznej funkcji obserwacji przyjęto, że na powyższe pytania należy udzielić odpowiedzi twierdzącej.

Rozważmy proces kryterialny *P'* i dwie wersje procesu weryfikowanego P_1 i P_2 przedstawione na Rys. 6-2 . Porównajmy zachowanie systemów dla funkcji obserwacji określonej jak w poprzednim przykładzie (Przykład 6-1).



Rys. 6-2 Proces kryterialny i dwie wersje procesu weryfikowanego

Stosując bezpośrednio definicję poprawności dla homomorficznej funkcji obserwacji uznamy oba procesy P_1 i P_2 za poprawne.

Traktując rozważane procesy jako modele wyjściowej specyfikacji DFD opisanej w ostatnim przykładzie, stwierdzimy raczej, że sekwencja akcji realizowana przez proces P_2 powinna zostać uznana za niepoprawną. Proces P_{β_1} (element specyfikacji DFD) używa w tym drugim przypadku danej *v* przed jej obliczeniem przez procesy P_{α_1} i P_{α_2} .

Dla uniknięcia podobnych sytuacji definiując poprawność dla niehomomorficznej funkcji obserwacji wprowadzono dodatkowe wymagania. Założono, że atomowość i relacje przyczynowości (następstwa) pomiędzy akcjami występującymi w specyfikacji kryterialnej muszą być odzwierciedlane przez rozłączność przedziałów czasowych, w których wykonywane są odpowiadające im zbiory akcji modelu weryfikowanego. Nie dotyczy to zbiorów odpowiadających akcjom równoległym i akcjom, pomiędzy którymi zachodzi możliwość wyboru (konflikt). To założenie w połączeniu z założeniem o jednoznacznym etykietowaniu przejść symbolami akcji (Def. 3-1) pozwala na dowodzenie uściślania akcji w modelach wykorzystujących przeplot (Por 2.5.2).

Formalnie, przedstawione wymagania opisane są przez dwie własności, które powinny spełniać poprawne ciągi rozwiązań. Są nimi:

- odwrotna dopuszczalność (ang. *reverse enabledness*)
- lokalna osiągalność (ang. *local reachability*).

6.1.3 Idea odwrotnej dopuszczalności

Odwrotna dopuszczalność jest pojęciem odnoszącym się do akcji procesu weryfikowanego. Jeśli akcja *act* jest akcją specyfikacji kryterialnej, to przez *reverseEnabled*(*act*) oznaczany będzie maksymalny podzbiór obserwowalnych akcji specyfikacji weryfikowanej, który może zostać użyty do obliczenia akcji *act*.

Przeanalizujmy jeszcze raz definicję ogólnej wektorowej funkcji obserwacji (Def 3-8). Funkcja ta była generowana przez złożenie dwóch odwzorowań pomiędzy alfabetami:

$$h_{1P} : 2^{A_1} \to A_P$$

$$g_{P2} : A_P \to 2^{A_2}$$

Przez A_P oznaczono niejawnie zdefiniowany alfabet przestrzeni liniowej obserwacji P.

Zbiór *reverseEnabled*(*act*) jest określony jako relacyjny przeciwobraz złożenia funkcji g_{1P} h_{1P} użytych do definicji funkcji obserwacji. Niech $\mathrm{D}(act) = \{A | act \in g_{P2}(h_{1P}(A))\}$.

$$reverseEnabled(act) = \bigcup_{A \in \mathrm{D}(act)} A$$

(Formalna definicja stosująca się bezpośrednio do wektorów przedstawiona zostanie dalej.)

Dla przykładu przedstawionego na Rys. 6-2 zachodzi więc:

reverseEnabled(α) = $\{\alpha_1 , \alpha_2\}$

reverseEnabled(β) = $\{\beta_1 , \beta_2\}$

Ideą rozszerzenia definicji poprawności o badanie własność odwrotnej dopuszczalności jest sprawdzanie dla każdego przejścia procesu weryfikowanego $x_1 \to x_2$, czy wykonana obserwowalna akcja była odwrotnie dopuszczalna ze względu na dopuszczalne akcje procesu kryterialnego w danym rozwiązaniu $\pi(x_1)$.

Wprowadzenie do modelu wymagań dotyczących odwrotnej dopuszczalności powoduje, że proces P_2 przedstawiony na Rys. 6-2 będzie uznany za niepoprawny, ponieważ jego pierwszym przejściem jest akcja β_1, która w rozwiązaniu początkowym $x_0 = 0$ nie jest odwrotnie dopuszczalna – jedyną akcją dopuszczalną procesu kryterialnego jest wówczas α, natomiast zbiorem akcji odwrotnie dopuszczalnych jest $\{\alpha_1 , \alpha_2\}$.

6.1.4 Idea lokalnej osiągalności

Pojęcie lokalnej osiągalności odnosi się do badanego ciągu rozwiązań. Tu jego idea zostanie przedstawiona na przykładzie ścieżki $t = \langle act_0,\ldots, act_n\rangle$.

Załóżmy, że w wyniku wykonania ostatniej akcji act_n zaobserwowano akcję α procesu kryterialnego. Dla poprawnego ciągu rozwiązań akcja α była dopuszczalnym następnym przejściem procesu kryterialnego dla ostatnich $k \geq 0$ elementów ścieżki przekształconych przez funkcję obserwacji, czyli dla podścieżki $t_{k,n-1} = \langle act_{n-1-k} ,\ldots, act_{n-1} \rangle$.

Niech $t_{k,n} = \langle act_{n-1-k} ,\ldots, act_{n-1}, act_n \rangle$. Oznaczmy przez $Z(t)$ zbiór symboli należących do ścieżki.

Zaobserwowana akcja α była lokalnie osiągalna, jeżeli istnieje zbiór $D_\alpha \subset Z(t_{k,n})$ spełniający:

$$D_\alpha \in Dom\, h_{1P} \wedge \alpha \in g_{P2}(h_{1P}(D_\alpha)),$$

czyli do wyznaczenia akcji α można użyć ścieżki $t' = t_{k,n} \uparrow D_\alpha$ powstałej przez usunięcie ze ścieżki $t_{k,n}$ symboli spoza zbioru D_α.

Badając zasadę lokalnej osiągalności do procesu P_1 przedstawionego na Rys. 6-2b stwierdzamy, że jest ona zachowana.

Przykład 6-2

Na Rys. 6-3 przedstawiono przykład specyfikacji, która narusza zasadę lokalnej osiągalności dla funkcji obserwacji postaci:

$init' = init$

$\alpha_1' = \alpha_1 \circ \alpha_2$

$\beta' = \beta$

$\gamma' = \gamma$

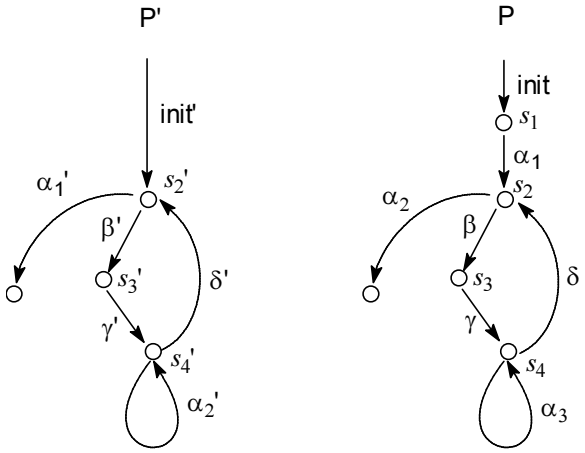
$\alpha_2' = \alpha_1 \circ \alpha_3$

$\delta' = \delta$

Rozważmy ścieżkę weryfikowanego systemu zapisaną jako wyrażenie regularne: $\langle init, \alpha_1 \rangle \bullet \langle \beta, \gamma, \alpha_3, \delta \rangle^*$ czyli $\langle init, \alpha_1, \beta, \gamma, \alpha_3, \delta, \beta, \gamma, \alpha_3, \delta \ldots \rangle$

Odpowiadającą jej ścieżką procesu kryterialnego jest $\langle init', \beta', \gamma', \alpha_2', \delta' \rangle \bullet \langle \beta', \gamma', \delta' \rangle^*$.

Stosując wymagania opisujące poprawność dla homomorficznej funkcji obserwacji stwierdzamy, że jest to ścieżka poprawna. Jak można zauważyć, własność odwrotnej dopuszczalności jest również zachowana,



Rys. 6-3 Brak lokalnej osiągalnosci

Jednakże segment początkowy $t = \langle init, \alpha_1, \beta, \gamma, \alpha_3 \rangle$ badanej ścieżki narusza zasadę lokalnej osiągalności.

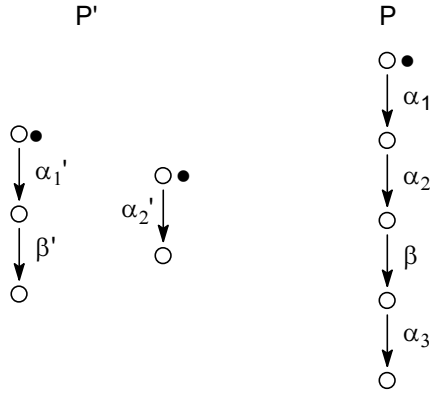
1. Po wykonaniu akcji *init* procesu weryfikowanego dopuszczalnymi akcjami procesu kryterialnego były α_1' oraz β'. Stąd w stanie s_1 akcjami odwrotnie dopuszczalnymi były: α_1, α_2 oraz β. Akcja α_1 została zarejestrowana jako składowa kryterialnej akcji α_1'.

2. Po wykonaniu akcji β i zaobserwowaniu β' proces kryterialny osiągnął stan oznaczony jako s_3'. W stanie tym wykonanie akcji α_1' nie jest już możliwe (jest ona niedopuszczalna), stąd akcja α_1 procesu weryfikowanego nie jest już odwrotnie dopuszczalna jako składowa akcji α_1'. Jedyną odwrotnie dopuszczalną akcją jest γ.
3. Po wykonaniu akcji γ akcjami odwrotnie dopuszczalnymi są: α_1 i α_3 (jako składowe dopuszczalnej akcji α_2') oraz akcja δ , która jest bezpośrednio odwzorowana na akcję δ'.
4. Po wykonaniu α_3 obserwator posługując się funkcją obserwacji zarejestruje wykonanie akcji α_2'. Oczekiwane jest więc, że towarzyszyło jej wykonanie akcji α_1 i α_3. Istotnie tak było dla rozważanej ścieżki, ale akcje nie były wykonane w tym samym *lokalnym kontekście*. Akcja α_1 nie była wykonana jako potencjalna składowa akcji α_2' procesu kryterialnego, lecz jako składowa akcji α_1'. Od osiągnięcia stanu s_4, kiedy akcje α_1 i α_3 stały się odwrotnie dopuszczalne ze względu na składową α_2' wykonana została jedynie akcja α_3, a więc nie wszystkie komponenty akcji α_2' zostały skompletowane. Akcja α_2' była dopuszczalna jedynie dla obrazu ostatniego segmentu $t_{\mathrm{kn}} = \langle \alpha_3 \rangle$ badanej ścieżki t. Łatwo zauważyć, że $Z(t_{\mathrm{kn}}) \notin \mathrm{Dom}\ h_{1\mathrm{P}}$

Oznacza to, że zaobserwowana akcja α_2' nie jest *lokalnie osiągalna*. Taki sam błędny rezultat byłby stwierdzony byłby dla granicznego przypadku, dla którego pary akcji (β , γ) oraz (β' , γ') zostałyby połączone w jedną akcję przez usunięcie stanów s_3 i s_3'.

Przykład 6-3

Rozważmy fragment specyfikacji przedstawiony na Rys. 6-4 . Kropkami oznaczono węzły grafów procesów, które są aktywne dla pewnego rozwiązania. Po stronie procesu kryterialnego dopuszczalne są dwa *równoległe* przejścia α_1' oraz α_2'.



Rys. 6-4 Przykład specyfikacji spełniającej własność lokalnej osiągalności

Załóżmy, że funkcja obserwacji ma postać:

$\alpha_1' = \alpha_1 \circ \alpha_2$

$\beta' = \beta$

$\alpha_2' = \alpha_1 \circ \alpha_3$

Powyższy fragment specyfikacji uważany jest za poprawny. Prześledźmy zachowanie weryfikowanego systemu:

1. W rozwiązaniu początkowym (kiedy aktywne są zaznaczone węzły) dopuszczalnymi przejściami procesu kryterialnego są α_1' oraz α_2'. Akcjami odwrotnie dopuszczalnymi są *reverseEnabled*(α_1') = {α_1 , α_2}, *reverseEnabled*(α_2') = {α_1 , α_3}
2. Po wykonaniu akcji α_1 i α_2 zmienił się zbiór akcji odwrotnie dopuszczalnych na *reverseEnabled*(β') = {β} oraz *reverseEnabled*(α_2') = {α_1 , α_3}. Akcja α_1 została zarejestrowana jako przejście wchodzące w skład akcji procesu kryterialnego α_1' a także równoległej akcji α_2'.
3. Następnie, po wykonaniu akcji β akcje α_1 i α_3 są dalej odwrotnie dopuszczalne ze względu na dopuszczalne przejście procesu kryterialnego α_2' : *reverseEnabled*(α_2') = {α_1 , α_3} . Akcja α_1 została wykonana wcześniej w tym samym lokalnym kontekście zbioru odwrotnie dopuszczalnego ze względu na α_2'.
4. Ostatecznie przejście α_3 domyka akcję α_2' procesu kryterialnego. Akcja α_2' była dopuszczalna dla całej badanej ścieżki $t = \langle \alpha_1 , \alpha_2, \beta, \alpha_3 \rangle$, stąd $t_{k,n} = \langle \alpha_1 , \alpha_2, \beta, \alpha_3 \rangle$. Ścieżka $t' = t_{k,n} \uparrow \{\alpha_1 , \alpha_3\} = \langle \alpha_1 , \alpha_3 \rangle$ i zawiera wystarczający zbiór akcji, by wyznaczyć przejście α_2'.

6.1.5 Porównanie z pojęciami występującymi w oryginalnym modelu poprawności

Omówiony w porozdziale 2.6 model poprawności względnej zakładał, ze odwzorowanie pomiędzy dwoma warstwami specyfikacji jest określane z użyciem relacji kryterialnej *k* łączącej pary stanów dwóch procesów: weryfikowanego *P* i kryterialnego *P'*.

Niech *SC*(*P*) oznacza zbiór wszystkich semiobliczeń procesu P. Na podstawie relacji *k* można wyznaczyć relację $ks \subset SC(P) \times T'$, gdzie *T'* oznacza zbiór przejść procesu weryfikowanego. Elementami relacji *ks* są pary postaci ($sc(s_i , s_{i+1}),(s_i' , s_{i+1}')$) spełniające:

$$sc(s_i , s_{i+1}) = \overline{SL}\,(s_i, s_i') \cap \overline{SU}\,(s_{i+1}, s_{i+1}')$$

(Symbolami $\overline{SL}\,(s_i, s_i')$ i $\overline{SU}\,(s_{i+1}, s_{i+1}')$ oznaczono domknięcia zbiorów semiobliczeń dolnych i górnych związanych z odpowiednimi parami, (patrz: Def 2-6).

Zauważmy, że w ogólnym przypadku przeciwobrazem $ks^{-1}(t)$ przejścia $t = (s_i' , s_{i+1}')$ może być zbiór semiobliczeń SC_t. Powyższa relacyjna zależność może zostać zapisana za pomocą odpowiednio skonstruowanej niehomomorficznej funkcji obserwacji dla modeli liniowych procesów.

Każdemu semiobliczeniu $sc_k \in Dom\ ks$ przypisany jest jeden wiersz M_k macierzy M opisującej przekształcenie π_{Min} oraz wartość 1 w *k*-tej kolumnie macierzy L i wierszu odpowiadającym wybranemu przejściu. Tak przekształcony problem jest równoważny problemowi poprawności dla niehomomorficznej funkcji obserwacji. (Chociaż oczywiście jego budowa mijałaby się z celem, ze względu na nakład obliczeniowy konieczny do zbudowania relacji *ks* .)

Poszukując analogii pomiędzy modelami, łatwo spostrzec podobieństwo pomiędzy zbiorem semiobliczeń górnych związanych z parą stanów $SL(s_i, s_i')$ a sumą zbiorów

$$\bigcup_{(s_i',s_k')\in T''} reverseEnabled(\sigma((s_i',s_k'))),$$

gdzie σ jest funkcją etykietującą przejścia.

Podobnie bliskie analogie można znaleźć pomiędzy pojęciem lokalnej osiągalności i sposobem wykorzystania w definicji poprawności zbioru semiobliczeń dolnych.

6.1.6 Porównanie poprawności i pojęcia makrohomorfizmu

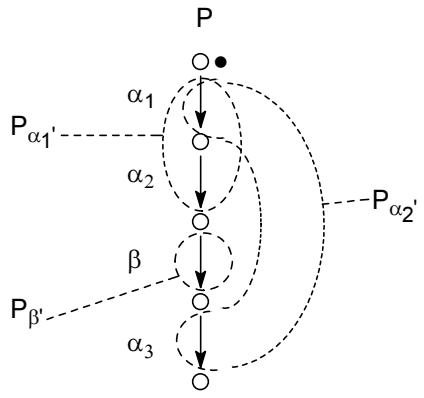
W [Szmuc 89b] pokazano równoważność pomiędzy poprawnością a tzw. *makrohomomorfizmem* procesów. Równoważność tę dowodzono poprzez odpowiednio prowadzoną dekompozycję procesu weryfikowanego na procesy składowe i jego zapis w postaci wyrażenia algebraicznego, którego termami były nazwy procesów uzyskanych w wyniku dekompozycji. Wprowadzona algebra procesów bazowała na operacjach na relacyjnych zbiorach semiobliczeń i przypominała składniowo wyrażenia regularne automatów skończonych. Dekompozycja procesu weryfikowanego na podprocesy prowadzona była podobnie do wydzielania semiobliczeń będących elementami par relacji *ks*.

Makrohomomorfizm procesów został zdefiniowany jako relacja pomiędzy procesami weryfikowanym i kryterialnym. Relacja ta była generowana przez pewną funkcję *g* odwzorowująca zbiór symboli podprocesów powstałych w wyniku dekompozycji procesu weryfikowanego w zbiór stanów procesu kryterialnego. Z definicji, relacja pomiędzy procesami była relacją makrohomomorficzną, jeżeli funkcja *g* w sposób homomorficzny odwzorowywała proces określony nad wyrażeniem opisującym strukturę dekompozycji (rodzaj języka automatu skończonego) w proces kryterialny.

Dowodzenie związków pomiędzy rozważanym w pracy modelem poprawności a pewną postacią makrohomorfizmu przekracza ramy tej pracy. Przede wszystkim, do jego określenia konieczny byłby wybór odpowiedniej algebry procesów bazującej na akcjach i analiza wynikających stąd konsekwencji. Jednakże, zamierzeniem autora jest zachowanie intuicji związków pomiędzy poprawnością dla niehomomorficznej funkcji obserwacji i istnieniem pewnego rodzaju relacji makrohomomorficznej.

Przykładem tego może być proponowana dekompozycja procesu *P* zdefiniowanego na Rys. 6-4 wykorzystująca definicję funkcji obserwacji (Przykład 6-3). (Proces *P* jest tu traktowany podobnie jak dla algebraicznych metod poprawności względnej – jako sekwencyjny opis zachowania systemu współbieżnego.)

Przedstawiona na Rys. 6-5 dekompozycja wydziela podprocesy zawierające przejścia wchodzące w skład zbiorów odwzorowanych przez niehomomorficzną funkcję obserwacji w poszczególne akcje procesu kryterialnego.



Rys. 6-5 Dekompozycja procesu w oparaciu o funkcję obserwacji

Dla zbioru symboli procesów powstałych w wyniku dekompozycji $\{ P_{\alpha 1'}, P_{\beta'}, P_{\alpha 2'}\}$ odpowiednie odwzorowanie w zbiór *akcji* procesu kryterialnego określone jest przez liniowy (tu identycznościowy) składnik funkcji obserwacji:

$g(P_{\alpha 1'}) = \alpha_1'$

$g(P_{\beta'}) = \beta'$

$g(P_{\alpha 2'}) = \alpha_2'$

Zapis procesu P w postaci wyrażenia algebraicznego CSP [Hoare 85] może mieć postać:

$P = (P_{\alpha 1'} ; P_{\beta'}) \parallel P_{\alpha 2'}$, gdzie

- operator ; jest operatorem sekwencji,
- operator || jest operatorem kompozycji (wspólne elementy alfabetów argumentów są uzgodnione, zdarzenia nieuzgodnione mogą być wykonywane z przeplotem.

Definicje poszczególnych procesów mają postać:

$P_{\alpha 1'} = \alpha_1 \rightarrow \alpha_2 \rightarrow TICK_{P\alpha 1'}$

$P_{\beta'} = \beta \rightarrow TICK_{P\beta'}$

$P_{\alpha 2'} = \alpha_1 \rightarrow \alpha_3 \rightarrow TICK_{P\alpha 2'}$

Symbolem $TICK_i$ oznaczono zakończone sukcesem wykonanie procesu P_i odpowiadające ścieżce $\langle \ldots , \checkmark \rangle$.

Analizując wyrażenie $P = (P_{\alpha 1'} ; P_{\beta'}) \parallel P_{\alpha 2'}$ możemy sprawdzić, że funkcja *g* odwzorowuje je w sposób homomorficzny w specyfikację procesu *P'*, ponieważ akcja α_2' jest wykonywana równolegle z sekwencją akcji $\langle \alpha_1', \beta' \rangle$

Problem weryfikacji dla niehomomorficznej funkcji obserwacji może być rozpatrywany także jako badanie makrohomomorfizmu, przez dynamiczne rozwijanie wyrażenia opisującego weryfikowany proces. Istotne jest to, że struktura dekompozycji nie jest z góry zadana, lecz określona jest jej specyfikacja wynikająca z definicji funkcji obserwacji.

Dla rozważanego przykładu zdefiniujmy

$PSPEC_{\alpha 1'} = \alpha_1 \rightarrow \alpha_2 \rightarrow TICK_{P\alpha 1'} \sqcap \alpha_2 \rightarrow \alpha_1 \rightarrow TICK_{P\alpha 1'}$

$PSPEC_{\beta'} = \beta \rightarrow TICK_{P\beta'}$

$PSPEC_{\alpha 2'} = \alpha_1 \rightarrow \alpha_3 \rightarrow TICK_{P\alpha 2'} \sqcap \alpha_3 \rightarrow \alpha_1 \rightarrow TICK_{P\alpha 2'}$

W specyfikacji używany jest operator „⊓" oznaczający niedeterministyczny wybór pomiędzy jedną z możliwych implementacji.

Alternatywnie możemy tę samą specyfikację wyrazić jako:

$traces(PSPEC_{\alpha 1'}) = \{ \langle \alpha_1, \alpha_2, \checkmark \rangle , \langle \alpha_2, \alpha_1, \checkmark \rangle \}$

$traces(PSPEC_{\beta'}) = \{ \langle \beta, \checkmark \rangle \}$

$traces(PSPEC_{\alpha 2'}) = \{ \langle \alpha_1, \alpha_3, \checkmark \rangle , \langle \alpha_3, \alpha_1, \checkmark \rangle \}$

Niech A_{obs} oznacza zbiór obserwowalnych akcji systemu weryfikowanego. Dla omawianego przykładu $A_{obs} = \{\alpha_1, \alpha_2, \alpha_3, \beta\}$. Dynamicznie obliczana struktura dekompozycji zdefiniowana jest poprzez wymagania:

$P_{\alpha 1'}$ **sat** $traces(P_{\alpha 1'}) \uparrow A_{obs} \subset traces(PSPEC_{\alpha 1'})$

$P_{\beta'}$ **sat** $traces(P_{\beta'}) \uparrow A_{obs} \subset traces(PSPEC_{\beta'})$

$P_{\alpha 2'}$ **sat** $traces(P_{\alpha 2'}) \uparrow A_{obs} \subset traces(PSPEC_{\alpha 2'})$

W rzeczywistości, funkcja obserwacji implikuje bardziej ogólną postać specyfikacji procesów powstałych w wyniku dekompozycji. Uogólniona specyfikacja powinna także zezwalać na pętle wewnątrz procesów, a więc na wariacje z powtórzeniami wewnątrz ścieżek.

6.2 Definicja poprawności dla niehomomorficznej funkcji obserwacji

Definicję poprawności poprzedzimy wprowadzeniem pojęć pomocniczych. Będziemy zakładali, że rozpatrywane są dwa modele liniowe systemów:

$ML = (X= R^n , x_0, A_I\, x \le b, A_E\, x = 0)$ – model specyfikacji weryfikowanej;

$ML' = (X'= R^{n'} , x_0', A_I'\, x' \le b', A_E'\, x' = 0)$ – model specyfikacji kryterialnej.

Rozpatrywana będzie niehomomorficzna funkcja obserwacji : $X \to X'$ zdefiniowana jako złożenie dwóch odwzorowań:

$\pi_{Min} : X \to P = R^r$ (przestrzeń P nazywana jest przestrzenią liniowej obserwacji)

$\pi_{Lin} : P \to X'$

Dalej zakładali będziemy, że oba odwzorowania opisane są macierzami. (Por. podrozdział 3.4) Odwzorowanie π_{Min} opisane jest macierzą M o wymiarach $r \times n$; natomiast odwzorowanie π_{Lin} macierzą L o wymiarach $n' \times r$.

6.2.1 Operacje pseudo-algebraiczne na wektorach

Macierz M specyfikuje odwzorowanie $h_{1P} : 2^{A_1} \to A_P$ użytege przy definicji nieliniowego składnika funkcji obserwacji. Jak łatwo zauważyć, jej wiersze M_i , $i = 1,\ldots,r$ odpowiadają zbiorom należącym do dziedziny $Dom(h_{1P})$. Część operacji, mimo, że formalnie dotyczy wektorów, przeprowadzana jest na wektorowych reprezentacjach wielozbiorów. Z tego powodu, aby uprościć część formuł, a zwłaszcza uniknąć zbyt wielu poziomów iteracji, wprowadzone zostaną dodatkowe operatory dla wektorów.

Niech $x, y, r \in R^m$ będą wektorami o nieujemnych współrzędnych.

Zdefiniujmy operator *restrykcji* : $R^m \times R^m \to R^m$ jako:

$$x \uparrow y = z \;\Leftrightarrow\; \forall\, 0 \le i \le m\,.\; \begin{cases} z(i) = x(i)\,, & \text{jeśli } y(i) \neq 0\,; \\ z(i) = 0\,, & \text{jeśli } y(i) = 0 \end{cases}$$

Zakładamy, że operator $\uparrow$ ma wyższy priorytet niż operacje dodawania wektorów. Dla nieujemnych argumentów operator ten jest rozdzielny względem dodawania, tzn.: $(x + y) \uparrow z = x \uparrow z + y \uparrow z$.

Operacja *dzielenia wektorów* odwzorowująca $R^m \times R^m \to N \cup \{0\}$ zdefiniowana jest jako:

$$x / y = k \;\Leftrightarrow\; 1.\; x = k\,y + r\,,\; r \ge 0$$

$$2.\; \exists\, i : 0 \le i \le n\,.\; (r(i) - y(i) < 0 \wedge y(i) > 0)$$

Jeśli $x / y = k$, wówczas wektor $r = x - k\,y$ nazywany jest resztą z dzielenia. Oznaczany będzie on jako $r(x / y)$.

Zdefiniujmy operację konwersji wektora na zbiór niezerowych indeksów $R^m \to 2^N$

$$idxset(x) = A \Leftrightarrow A = \{i \in \mathrm{N} \mid x(i) \neq 0\}$$

■

Oznaczmy przez $e_i \in \mathrm{R}^n$ i-ty wersor, czyli wektor spełniający $e_i(i)=1$; $e_i(k)=0$ dla $k \neq i$.

6.2.2 Definicje odwrotnej dopuszczalności i lokalnej osiągalności

Def. 6-1 Zbiór obserwowalnych akcji modelu ML

Zbiorem obserwowalnych akcji (przejść) atomowych dla funkcji obserwacji π nazywali będziemy zbiór wektorów postaci

$$O_\pi = \{e_i \in \mathrm{R}^n \mid \exists k \,.\, i \in idxset(M_k) \wedge \exists j \,.\, L_{ik} \neq 0\}$$

■

Def. 6-2 Zbiór akcji odwrotnie dopuszczalnych

Zdefiniujmy zbiór $IDX_P(v')$ jako:

$$IDX_P(v') = \bigcup_{\substack{i=1,n' \\ v'(i)\neq 0}} idxset(L_i)\,.$$

Jest to zbiór współrzędnych akcji w przestrzeni P, które mogą doprowadzić do zmiany jednej ze współrzędnych v' .

Niech $IDX_X(v') = \bigcup_{i \in IDX_P(v')} idxset(M_i)\,.$

Jest to zbiór współrzędnych akcji w przestrzeni X, które mogą doprowadzić do zmiany jednej ze współrzędnych v'.

Zdefiniujemy zbiór akcji atomowych odwrotnie dopuszczalnych ze względu na przejście v':

$$RE_X(v') = \{e_i \in \mathrm{X} \mid i \in IDX_X(v')\}\,.$$

Zbiór akcji atomowych odwrotnie dopuszczalnych dla danego rozwiązania $x \in$ X modelu *ML* zdefiniowany jest jako:

$$RE(x) = \bigcup_{v' \in V_F(\pi(x))} RE_X(v')$$

■

Zgodnie z definicją, dla danego rozwiązania $x \in$ X wyznaczany jest jego obraz poprzez funkcję obserwacji $x' = \pi(x)$. Dla rozwiązania x' wyznaczany jest zbiór przejść dopuszczalnych $V_F(\pi(x))$ modelu *ML'*. Zbiór ten jest podstawą do wyznaczenia zbioru akcji odwrotnie dopuszczalnych w przestrzeni P – czyli tych akcji, których zaobserwowanie może prowadzić do zaobserwowania przejścia ze zbioru $V_F(\pi(x))$. Następnie odwrotna dopuszczalność w przestrzeni P propaguje się dalej na przestrzeń rozwiązań modelu *ML*.

Przedstawiona poniżej definicja *lokalnej osiągalności* będzie nieco bardziej złożona, niż przedstawiono w dyskusji. Uwzględniała ona będzie także przypadek szczególny, kiedy obserwowana akcja procesu kryterialnego tworzy „oczko". Będziemy więc żądali, aby zaobserwowanie danej akcji procesu kryterialnego było poprzedzone wykonaniem wszystkich akcji składowych modelu weryfikowanego począwszy od rozwiązania, kiedy:

- stały się one odwrotnie dopuszczalne,
- w przypadku akcji obserwowanych wielokrotnie po rozwiązaniu, od którego stały się odwrotnie dopuszczalne – także od momentu kiedy ostatni raz zaobserwowano rozważaną akcję.

Pojęcie lokalnej osiągalności stosuje się raczej do przeciwdziedziny funkcji π_{Min} , czyli przestrzeni liniowej obserwacji P. Lokalna osiągalność w przestrzeni rozwiązań modelu kryterialnego jest pochodną lokalnej osiągalności w przestrzeni P.

Def. 6-3 Lokalna osiągalność dla ciągu rozwiązań

Niech $s(x_0 , x_n)$ będzie ciągiem kolejnych rozwiązań modelu *ML*.

Załóżmy, że przejściu $\Delta x_n = x_n - x_{n-1}$ odpowiada zaobserwowanie przejścia *v*' modelu *ML'*, czyli $v' = \pi(x_n) - \pi(x_{n-1}) \neq 0$. Załóżmy, że przejście to jest poprawnym przejściem równoległym w rozwiązaniu $\pi(x_{n-1})$. Z nierówności $v' \neq 0$ wynika, że $\Delta p = \pi_{Min}(x_n) - \pi_{Min}(x_{n-1}) \neq 0$.

Rozważmy *k*–tą składową wektora Δp dla której zachodzi $\Delta p(k) = 1$.

Niech $le(k) = \max_{i \in \{0,\ldots,n-1\}} \{i | \exists v' \in V(ML'). \forall 0 \leq j < i . v' \in V_F(\pi(x_{n-j})) \wedge k \in IDX_P(v')\}$

Oznaczmy przez $lo(k)$:

$$lo(k) = \begin{cases} \max_{i \in \{1,\ldots,n-1\}} \{i | \Delta p_i = \pi_{Min}(x_i) - \pi_{Min}(x_{i-1}) \wedge \Delta p_i(k) \neq 0\}, \text{ jeśli } p = \pi_{Min}(x_{n-1}) \ \wedge \ p(k) \neq 0 \\ 0 \text{ w p.p.} \end{cases}$$

Niech $lr(k) = \max \{le(k) , lo(k)\}$ (i)

Zdefiniujmy funkcję $z(k)$ jako: $z(k) = x_n - x_{n-1r(k)}$.

Akcja *k* w przestrzeni P jest *lokalnie osiągalna* w ciągu rozwiązań $s(x_0 , x_n)$, jeśli zachodzi:

$z(k) \uparrow M_k \geq M_k$ (ii)

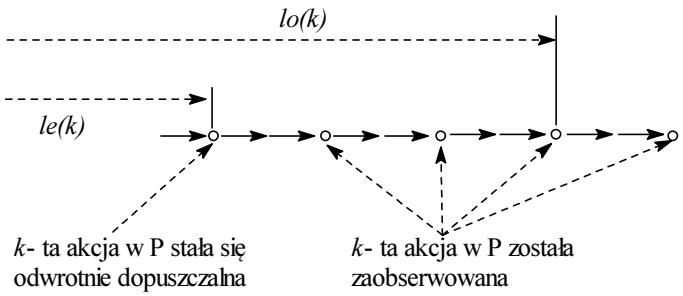
Przejście $\Delta p = \pi_{Min}(x_n) - \pi_{Min}(x_{n-1})$ jest lokalnie osiągalne, jeśli każda z jego akcji jest lokalnie osiągalna.

Przejście $v' = \pi(x_n) - \pi(x_{n-1})$ w modelu *ML'* jest lokalnie osiągalne w ciągu rozwiązań $s(x_0 , x_n)$ jeśli $v' = 0$ lub $v' \neq 0$ i przejście $\Delta p = \pi_{Min}(x_n) - \pi_{Min}(x_{n-1})$ jest lokalnie osiągalne.

■

Omówmy krótko konstrukcję definicji. Wartość $le(k)$ (patrz Rys. 6-6) jest indeksem rozwiązania należącego do ciągu $s(x_0 , x_n)$, w którym *k*–ta akcja w przestrzeni P stała się odwrotnie dopuszczalna i pozostała taka aż do końca ciągu. Wartość taka istnieje, ponieważ założyliśmy, ze $v' = \pi(x_n) - \pi(x_{n-1})$ było poprawnym przejściem równoległym.

Wartość $lo(k)$ jest indeksem rozwiązania, w którym ostatni raz zaobserwowano zmianę *k*–tej składowej w przestrzeni P lub ma wartość 0. W typowym przypadku zachodziło będzie $le(k) \geq lo(k)$. Wyjątkiem jest sytuacja, kiedy w specyfikacji kryterialnej występuje oczko. Wówczas dana akcja *k* w przestrzeni P może być wielokrotnie obserwowana od momentu, kiedy stała się odwrotnie dopuszczalna. Rys. 6-6 obrazuje taką sytuację.



Rys. 6-6 Ilustracja graficzna definicji lokalnej osiągalności

Funkcja $lr(k)$ opisana zależnością (i) odpowiada za wydzielanie z ciagu rozwiązań $s(x_0, x_n)$ końcowego podciągu postaci $s(x_{n\text{-}1r(j)}, x_n)$. Zależność (ii) jest równoważna zastosowaniu operatora restrykcji w odniesieniu do podciągu reprezentowanego przez wektor $z(k) = x_n - x_{n\text{-}1r(k)}$. Oczekujemy, że po usunięciu elementów spoza zbioru opisanego przez k-ty wiersz macierzy M pozostałe elementy wektora są wystarczające do zaobserwowania k-tej akcji.

Zauważmy jeszcze, że nie będzie nigdy zachodził warunek $z(k) \uparrow M_k \geq 2M_k$, ponieważ uniemożliwia to definicja $le(k)$. W typowych przypadkach spełnione będzie: $z(k) \uparrow M_k = M_k$. Nierówność (ii) możemy także zastąpić warunkiem $(z(k) \uparrow M_k) / M_k = 1$.

6.2.3 Poprawność względna dla niehomomorficznej funkcji obserwacji

Przedstawiona poniżej definicja poprawności względnej ciągu rozwiązań stanowi modyfikację definicji poprawności dla liniowej funkcji obserwacji (Def. 5-1).

Def. 6-4 Poprawność względna ciągu rozwiązań

Dane są dwa modele specyfikacji

$ML = (X, x_0, A_I\, x \leq b, A_E\, x = 0)$ – model specyfikacji weryfikowanej) oraz

$ML' = (X', x_0', A_I'\, x' \leq b', A_E'\, x' = 0)$ – model specyfikacji kryterialnej)

Dana jest niehomomorficzna funkcja obserwacji $\pi : X \rightarrow X'$

Zakładamy, że $\pi(x_0) = x_0'$.

Ciąg rozwiązań osiągalnych $s(x_0, \bullet) = \langle x_0, x_1, x_2, \ldots, x_i, x_{i+1}, \ldots \rangle$ jest poprawny, jeśli:

1. Dla każdej pary rozwiązań (x_i, x_{i+1}) spełnione są następujące warunki:
   a) Oznaczmy przez $\Delta x = x_{i+1} - x_i$. Warunkiem *odwrotnej dopuszczalności* jest $\forall i.(\Delta x(i) \neq 0 \wedge e_i \in O_\pi \Rightarrow e_i \in RE_X(x_i))$
   b) Przejście w specyfikacji kryterialnej $v_c' = \pi(x_{i+1}) - \pi(x_i)$ spełnia:
      $v_c' = 0$ lub
      $v_c' \in V_F(\pi(x_i))$ i v_c' jest dopuszczalnym przejściem złożonym w $\pi(x_i)$.
   c) Przejście v_c' jest *lokalnie osiągalne* w ciągu rozwiązań $s(x_0, x_{i+1})$
2. Jeżeli ciąg $s(x_0, \bullet)$ jest skończony, wówczas jego ostatnie rozwiązanie x_n jest rozwiązaniem końcowym, czyli spełnia: $V_F(x_n) = \varnothing \wedge \forall i \in N \cdot A_i\, x_n = 0$ (zdefiniowany w 4.7.4 predykat *final*(x_n)).
3. Jeśli x_n rozwiązaniem końcowym to $x_n' = \pi(x_n)$ spełnia *final*(x_n').

■

Analogicznie, jak w definicji poprawności dla liniowej funkcji obserwacji zdefiniujmy predykat *correct* ($s(x_0, \bullet)$, π, *ML*'), który przyjmuje wartość prawdy, jeśli ciąg $s(x_0, \bullet)$ jest poprawny względem modelu *ML*' dla niehomomorficznej funkcji obserwacji.

Definicje poprawności częściowej i całkowitej (oraz poprawności potencjalnej) podane w podrozdziale 5.1 przenoszą się bez zmian dla niehomomorficznej funkcji obserwacji. Bazują one wyłącznie na definicji poprawności ciągu rozwiązań, która jako jedyna uległa zmianie.

Tak więc model *ML* jest częściowo poprawny względem *ML*' dla niehomomorficznej funkcji obserwacji, jeśli wszystkie jego ciągi rozwiązań są poprawne.

Model *ML* jest całkowicie poprawny jeśli jest częściowo poprawny, wszystkie akcje procesu kryterialnego są obserwowalne oraz nie żadne z rozwiązań nie zawiera dywergencji.

6.3 Weryfikacja poprawności dla niehomomorficznej funkcji obserwacji

6.3.1 Macierz lokalnej osiągalności

Warunki określające lokalną osiągalność wydają się trudne w testowaniu, ponieważ dla analizowanego ciągu rozwiązań $s(x_0, x_n)$ wymagają sprawdzenia pewnej liczby jego elementów poprzedzających ostatnie rozwiązanie, co jest nieefektywne z punktu widzenia złożoności obliczeniowej algorytmu weryfikacji. Nasuwającym się rozwiązaniem jest stworzenie pewnej reprezentacji *historii wykonania* modelu weryfikowanego, która akumulowała będzie informacje o wykonanych przejściach.

Formalną konstrukcją służącą temu celowi jest macierz lokalnej osiągalności. Jej zadaniem jest przechowywanie informacji:

- umożliwiających szybkie rozstrzyganie o spełnieniu warunków lokalnej osiągalności,
- reprezentujących ścieżkę dojścia do stanu procesu weryfikowanego, co jest podstawą do stwierdzenia równoważności osiąganych stanów ze względu na możliwe *przyszłe* obserwacje.

Macierz lokalnej osiągalności R jest macierzą o rozmiarach $r \times n$ (czyli równych rozmiarom macierzy M opisującej odwzorowanie π_{Min}). Każdy wiersz macierzy R przechowuje informacje o ostatnio wykonanych odwrotnie dopuszczalnych akcjach procesu weryfikowanego.

W miarę wykonywania kolejnych przejść v_i w procesie weryfikowanym do wybranych wierszy macierzy R odpowiadających odwrotnie dopuszczalnym zbiorom akcji w przestrzeni P dodawane są wektory $v_i \uparrow M_k$. Zmiana zbioru przejść odwrotnie dopuszczalnych lub zaobserwowanie akcji przestrzeni P powoduje wyzerowanie odpowiednich wierszy.

Zadaniem macierzy lokalnej osiągalności jest więc akumulowanie informacji o wykonanych akcjach weryfikowanego procesu i ich klasyfikacja w postaci rodziny zbiorów, które mogą potencjalnie stać się argumentami odwzorowania h_{1P} generującego nieliniowy składnik funkcji obserwacji π_{Min} .

Def. 6-5 Konstrukcja ciągu macierzy lokalnej osiągalności

Dla danego ciągu rozwiązań $s(x_0,\bullet) = \langle x_0, x_1,\ldots, x_n,\ldots \rangle$ zbudujemy ciąg macierzy lokalnej osiągalności $\rho(s(x_0,\bullet)) = \langle R^{(0)}, R^{(1)},\ldots, R^{(n)},\ldots \rangle$.

Oznaczmy $IDX_P(x_k) = \bigcup_{v' \in V_F(\pi(x_k))} IDX_P(v')$. Jest to zbiór współrzędnych (indeksów) macierzy M odpowiadających odwrotnie dopuszczalnym akcjom w rozwiązaniu x_k.

1. Macierz początkowa $R^{(0)}$ opisana jest następującymi równaniami
   a) Jeśli $k \notin IDX_P(x_0)$ wówczas $R_k^{(0)} = \mathbf{0}$
   b) Jeśli $k \in IDX_P(x_0)$ wówczas:
      $$R_k^{(0)} = r(x_0 / M_k) \uparrow M_k$$
2. Dla danej macierzy $R^{(i-1)}$ i przejścia $\Delta x_i = x_i - x_{i-1}$ macierz następna macierz $R^{(i)}$ obliczana jest jako:
   a) Jeśli $k \notin IDX_P(x_i)$, wówczas $R_k^{(i)} = \mathbf{0}$
   b) Jeśli dla $\Delta p = \pi_{Min}(x_i) - \pi_{Min}(x_{i-1})$ istnieje k, że $\Delta p(k) \neq 0$, wówczas $R_k^{(i)} = \mathbf{0}$
   c) Jeśli $k \in IDX_P(x_i)$ i warunek (b) nie zachodzi wówczas:
      $$R_k^{(i)} = R_k^{(i-1)} + \Delta x_i \uparrow M_k$$

■

Dla uproszczenia zapisu w algorytmie będziemy posługiwać się funkcją *lrm* , która oblicza kolejną macierz na podstawie macierzy poprzedniej i wykonanego przejścia zgodnie z (2). Tak więc będziemy pisali:

$$R^{(i)} = lrm(R^{(i-1)}, \Delta x_i).$$

6.3.2 Badanie lokalnej osiągalności

Zastosowanie macierzy lokalnej osiągalności upraszcza procedurę sprawdzania czy własność ta jest zachowana.

Wniosek 6-1

Niech $s(x_0 , x_n)$ będzie ciągiem kolejnych rozwiązań modelu *ML*. Załóżmy, że jego podciąg $s(x_0 , x_{n-1})$ jest poprawny.

Niech $s(R^{(0)} , R^{(n-1)})$ będzie odpowiadającym mu ciągiem macierzy lokalnej osiągalności skonstruowanym zgodnie z Def. 6-5.

Załóżmy, że przejściu $\Delta x_n = x_n - x_{n-1}$ odpowiada zaobserwowanie przejścia modelu *ML'*, czyli wektor $\Delta x' = \pi(x_n) - \pi(x_{n-1})$ spełnia $\Delta x' \neq 0$. Załóżmy, że przejście to jest dopuszczalnym przejściem równoległym.

Obliczeniu wektora $\Delta x'$ z użyciem działającej globalnie funkcji obserwacji powinno odpowiadać obliczenie lokalne – z wykorzystaniem danych zgromadzonych w macierzy $R^{(i-1)}$.

Niech $\Delta p_n \in P$ będzie wektorem, którego współrzędne spełniają:

$$\forall\, k \in \{1,\ldots,r\} \,.\, \Delta p_n(k) = (R_k^{(n-1)} + \Delta x_n \uparrow M_k) / M_k \qquad \text{(i)}$$

Niech $z' \in X'$ będzie wektorem zdefiniowanym jako:

$$z' = \pi_{Lin}(\Delta p_n) = L\, \Delta p_n \qquad \text{(ii)}$$

Warunkiem, aby przejście $\Delta x'$ było lokalnie osiągalne jest spełnienie warunku $\Delta x' = z'$.

■

Uzasadnienie Równoważność warunków wynika bezpośrednio z zasady konstrukcji macierzy lokalnej osiągalności. Rozważmy *k*–tą akcję w przestrzeni P. Zauważmy, że odpowiadający jej *k*–ty wiersz macierzy może mieć niezerowe elementy wyłącznie wtedy, jeżeli akcja jest odwrotnie dopuszczalna. A więc, $R_k^{(le(k))} = \mathbf{0}$. Z zasad konstrukcji macierzy R wynika, że jej *k*–ty wiersz jest zerowany w momencie zaobserwowania *k*–tej akcji w przestrzeni P. Stąd zachodzi również $R_k^{(lo(k))} = \mathbf{0}$, o ile taka macierz istnieje. Z rozdzielności operatora $\uparrow$ względem sumy nieujemnych wektorów wynika, że $R_k^{(n-1)} = (x_{n-1} - x_{lr(k)}) \uparrow M_k$, a zatem: $R_k^{(n-1)} + \Delta x_n \uparrow M_k = (x_n - x_{lr(k)}) \uparrow M_k$.

Stąd warunek $(R_k^{(n-1)} + \Delta x_n \uparrow M_k) / M_k = 1$ jest równoważny warunkowi (ii) definicji lokalnej osiągalności.

□

Zdefiniujmy funkcję *lrtrans*, która oblicza wartość z' zgodnie z zależnościami (i) oraz (ii):

$$z' = lrtrans(R , \Delta x).$$

Zastosowanie macierzy lokalnej osiągalności pozwala zamienić warunek 1.c definicji poprawności względnej ciągu rozwiązań (Def. 6-4) na równoważną asercję postaci:

Przejście v_c' jest lokalnie osiągalne w ciągu rozwiązań $s(x_0 , x_{i+1})$ jeśli

$$v_c' = lrtrans(R^{(i)} , v_i),$$

gdzie macierz $R^{(i)}$ jest macierzą lokalnej osiągalności odpowiadającej *i*-temu rozwiązaniu.

6.3.3 Pokrywanie dla macierzy lokalnej osiągalności

Def. 6-6 Relacja pokrywania dla macierzy lokalnej osiągalności

Niech $R^{(1)}$ i $R^{(2)}$ będą dwoma macierzami lokalnej osiągalności o wymiarach $r \times n$. Mówimy, że macierz $R^{(2)}$ pokrywa macierz $R^{(1)}$, jeśli:

$$\forall k \in \{1 \ldots r\} \,.\, R_k^{(1)} = R_k^{(2)} \vee R_k^{(1)} \prec R_k^{(2)}$$

$$\exists k \in \{1 \ldots r\} \,.\, R_k^{(1)} \prec R_k^{(2)}$$

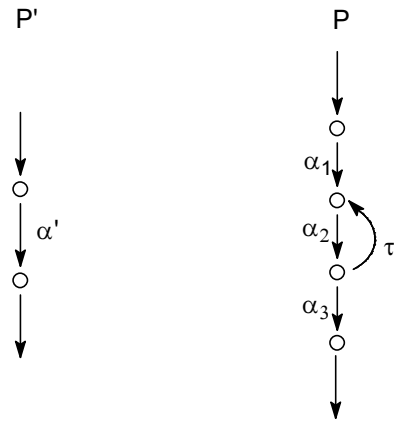
Pokrywanie macierzy zapisywać będziemy analogicznie, jak dla wektorów jako $R^{(1)} \prec R^{(2)}$

■

Przykład fragmentu specyfikacji, dla której będzie występowało pokrywanie dla macierzy lokalnej osiągalności przedstawiono na Rys. 6-7. Załóżmy, że funkcja obserwacji ma postać: $\alpha' = \alpha_1 \circ \alpha_2 \circ \alpha_3$. Łatwo zauważyć, że wiersz $R_{\alpha'}$ (związany z akcją α') macierzy lokalnej osiagalności może przybierać wartości:

$\{[\alpha_1 : 0; \alpha_2 : 0; \alpha_3 : 0], [\alpha_1 : 1; \alpha_2 : 0; \alpha_3 : 0], [\alpha_1 : 1; \alpha_2 : 1; \alpha_3 : 0],$

$[\alpha_1 : 1; \alpha_2 : 2; \alpha_3 : 0], [\alpha_1 : 1; \alpha_2 : s; \alpha_3 : 0], \ldots\}$, gdzie s jest dowolnie dużą liczbą.



Rys. 6-7 Specyfikacja, dla której występuje pokrywanie w macierzy lokalnej osiągalności

Wystąpienie pokrywania dla macierzy lokalnej osiągalności nie jest zjawiskiem negatywnym z punktu widzenia poprawności częściowej (poprawność całkowita będzie na ogół wykluczona ze względu na wystąpienie dywergencji).

Istotne informacje, ze względu na możliwości decydowania o przyszłych obserwacjach, to przede wszystkim rozróżnienie pomiędzy wartością zerowymi i niezerowymi.

Łatwo sprawdzić korzystając z definicji funkcji *lrm* i *lrtrans* oraz definicji macierzy lokalnej osiagalności, że zachodzą poniższe zależności:

Niech $R^{(ij)}$ będą macierzami lokalnej osiągalności dla $i, j = 1, 2$; niech v będzie przejściem elementarnym (lub ogólniej wektorem o współrzędnych ze zbioru {0, 1}).

1. $R^{(11)} \prec R^{(21)} \wedge R^{(12)} = lrm(R^{(11)},v) \wedge R^{(22)} = lrm(R^{(21)},v) \Rightarrow$

   $\Rightarrow R^{(12)} = R^{(22)} \vee R^{(12)} \prec R^{(22)}$

1. $R^{(11)} \prec R^{(21)} \Rightarrow lrtrans(R^{(11)}, v) = lrtrans(R^{(21)}, v)$

Najistotniejsze jest tu założenie, że współrzędne wektora v przyjmują wartości ze zbioru {0, 1}. Jeśli $R^{(11)} \prec R^{(21)}$ to dodanie do k–tego wiersza macierzy $R^{(11)}$ i $R^{(21)}$ wektora $v \uparrow M_k$ spowoduje albo zmianę pewnej zerowej wartości na wartość 1, albo powiększenie niezerowej wartości o 1 albo wyzerowanie całego wiersza w wyniku zaobserwowania k–tej akcji w przestrzeni P.

Def. 6-7 Równoważność macierzy lokalnej osiągalności

Zdefiniujmy relację równoważności:

$$R^{(1)} \approx R^{(2)} \Leftrightarrow R^{(1)} = R^{(2)} \vee R^{(1)} \prec R^{(2)} \vee R^{(2)} \prec R^{(1)}$$

■

Wniosek 6-2

Niech $R^{(ij)}$ będą macierzami lokalnej osiągalności dla $i, j = 1, 2$, niech v będzie wektorem o współrzędnych ze zbioru {0, 1}.

1. $R^{(11)} \approx R^{(21)} \wedge R^{(12)} = lrm(R^{(11)}, v) \wedge R^{(22)} = lrm(R^{(21)}, v) \Rightarrow R^{(12)} \approx R^{(22)}$

1. $R^{(11)} \approx R^{(21)} \Rightarrow lrtrans(R^{(11)}, v) = lrtrans(R^{(21)}, v)$

■

6.3.4 Ekstrapolacja poprawności dla ciągów rozwiązań

Podobnie, jak dla relacji określającej podobieństwo rozwiązań dla homomorficznej funkcji obserwacji (Def. 5-8), sformułujemy twierdzenie, które pozwalało będzie rozciągnąć własność poprawności określoną dla przejścia w pewnym ciągu na przejście w innym ciągu. Podstawą do rozstrzygania poprawności będzie w tym przypadku spełnienie warunków analogicznych, jak dla podobieństwa rozwiązań oraz równoważność macierzy lokalnej osiągalności. Istotne założenie dotyczy także osiągalnych wartości macierzy lokalnej osiągalności.

Twierdzenie 6-1 O ekstrapolacji poprawności

1. Niech x_i będzie rozwiązaniem osiągalnym modelu *ML* w pewnym poprawnym ciągu $s_1(\bullet, x_i)$; oznaczmy przez y_i odpowiadającą mu wartość stanu, przez y_i' wartość stanu modelu *ML'* odpowiadającą rozwiązaniu $\pi(x_i)$, przez $R^{(i)}$ macierz lokalnej osiągalności w rozwiązaniu x_i wyznaczoną w ciągu $\rho(s_1(\bullet, x_i))$. Zapiszmy zdefiniowane wartości jako parę $(x_i, (y_i, y_i', R^{(i)}))$.

2. Niech v będzie przejściem dopuszczalnym w stanie y_i. Oznaczmy $x_{i+1} = x_i + v$ i obliczmy wartości $(x_{i+1}, (y_{i+1}, y_{i+1}', R^{(i+1)}))$. Załóżmy, że przejście pomiędzy rozwiązaniami $x_i \rightarrow x_{i+1}$ spełnia warunki poprawności przejścia określone zgodnie z Def. 6-4, a więc:

   a) Przejście v jest odwrotnie dopuszczalne:

$$\forall i.(\, v(i) \neq 0 \wedge e_i \in O_\pi \Rightarrow e_i \in RE_X(x_i)\,)$$

b) Przejście w specyfikacji kryterialnej $v_c' = \pi(x_{i+1}) - \pi(x_i)$ spełnia:

$v_c' = 0$ lub

$v_c' \in V_F(\pi(x_i))$ i v_c' jest dopuszczalnym przejściem złożonym w $\pi(x_i)$.

c) Przejście v_c' jest lokalnie osiągalne w ciągu rozwiązań $s(x_0, x_{i+1})$, czyli spełnia:
$v_c' = lrtrans(R^{(i)}, v_i)$

3. Niech x_j będzie rozwiązaniem osiągalnym modelu *ML* w pewnym ciągu poprawnym $s_2(\bullet, x_j)$; załóżmy, że para $(x_j, (y_j, y_j', R^{(j)}))$ spełnia następujące warunki:

$y_i = y_j$

$y_j' \geq y_i'$

$R^{(j)} \approx R^{(i)}$

4. Oznaczmy przez *RS* zbiór wszystkich macierzy lokalnej osiągalności, które mogą być wyznaczone dla dowolnych ciągów rozwiązań $s(\bullet,\bullet) \in S(ML)$ modelu *ML*:

$$RS = \bigcup_{s(\bullet,\bullet)\in S(ML)} \bigcup_{R\in Z(\rho(s(\bullet,\bullet)))} R$$

Załóżmy, że każda macierz $R \in RS$ spełnia poniższy niezmienniczy warunek:

$$\forall k \in \{1,\ldots,r\}.\exists NR_k \subset \{1,\ldots,n\}.\forall nr \in NR_k.R_k(i) = 0 \wedge M_k(i) \neq 0$$

□

Z powyższych założeń wynika, że

1. przejście $x_j \rightarrow x_{j+1} = x_j + v$ jest poprawnym przejściem
2. oraz zachodzi:

a) $y_{i+1} = y_{j+1}$

b) $y_{j+1}' \geq y_{i+1}'$

c) $R^{(j+1)} \approx R^{(i+1)}$

■

Dowód

Łatwo zauważyć podobieństwo do wniosku 5-2 dla homomorficznej funkcji obserwacji. Analogiczny dowód dla homomorficznej funkcji obserwacji opiera się na założeniu, że $v_c' = \pi(x_{i+1}) - \pi(x_i)$ spełnia $v_c' = \pi(v)$. Podstawową różnicą pomiędzy homomorficzną i niehomomorficzną funkcją obserwacji jest sposób obliczania v_c'.

Załóżmy wpierw, że spełnione jest:

$v_c' = \pi(x_{j+1}) - \pi(x_j) = \pi(x_{j+1}) - \pi(x_j)$.

Bezpośrednio z założenia (2) i (3) wynika, że przejście *v* jest odwrotnie dopuszczalne dla rozwiązania x_j, ponieważ zbiór dopuszczalnych przejść w modelu ML spełnia

$V_F(y_i') \subset V_F(y_i')$. Podobnie, spełniony jest warunek odwrotnej dopuszczalności, ze względu na równoważność macierzy $R^{(j)} \approx R^{(i)}$ i równość *lrtrans*($R^{(j)}$, *v*) = *lrtrans*($R^{(i)}$, *v*).

Założenie $\pi(x_{j+1}) - \pi(x_j) = \pi(x_{i+1}) - \pi(x_i)$ gwarantuje nam spełnienie warunków 2a,b tezy podobnie jak dla homomorficznej funkcji obserwacji; warunek 2c wynika bezpośrednio z zasad konstrukcji macierzy lokalnej osiągalności (i równoważności $R^{(j)} \approx R^{(i)}$)

Udowodnijmy teraz, że $\pi(x_{j+1}) - \pi(x_j)$ = *lrtrans*($R^{(j)}$, *v*), czyli że zmiana wartości funkcji obserwacji może być obliczona na podstawie macierzy lokalnej osiągalności i wykonanego przejścia. Zauważmy, że w tym celu wystarczy ograniczyć się do obliczenia zmian w przestrzeni liniowej obserwacji P, czyli przeanalizować jedynie niehomomorficzny składnik odwzorowania π_{Min} .

Niech $p = \pi_{Min}(x)$. Rozważmy *k*-tą składową wektora *p* ; zachodzi wówczas $p(k) = x / M_k$. Zauważmy także, że jeśli $p' = \pi_{Min}(x + v)$, to zachodzi:

$$p'(k) = (r(x / M_k) + v \uparrow M_k) / M_k = (r((x \uparrow M_k) / M_k) + v \uparrow M_k) / M_k .$$

Czyli: $p'(k) - p(k) = 1$ jeśli $r((x \uparrow M_k) / M_k) + v \uparrow M_k \geq M_k$ oraz $p'(k) - p(k) = 0$ jeśli istnieje niezerowy element wektora M_k, który przyjmuje wartość 0 dla wektora występującego po lewej stronie nierówności $r((x \uparrow M_k) / M_k) + v \uparrow M_k$.

Obliczmy wartość $r((x_j \uparrow M_k) / M_k)$ posługując się macierzami lokalnej osiągalności należącymi do ciągu $\rho(s_2(\bullet , x_j))$. Z zasad konstrukcji macierzy lokalnej osiągalności wynika, że ich *k*-te wiersze są zerowane, kiedy *k*-ta składowa w przestrzeni obserwacji przestaje być odwrotnie dopuszczalna lub zaobserwowano *k*–tą akcję w przestrzeni P.

Stąd można obliczyć wartość $r((x_j \uparrow M_k) / M_k)$ posługując się macierzami zawartymi w podciągu

$$\rho' = \langle R_k^{(j_1)}, R_k^{(j_2)}, \ldots, R_k^{(j_{n-1})}, R_k^{(j_t)} \rangle,$$

dla których jest spełnione

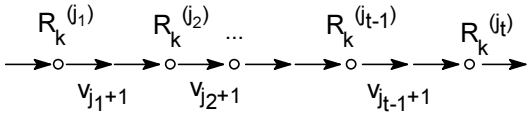
$$\forall s < t . R_k^{(j_s+1)} = 0 \text{ oraz } R_k^{(j_t)} = R_k^{(j)} .$$

Mamy więc:

$$r((x_j \uparrow M_k) / M_k) = \sum_{s<t} r((R_k^{(j_s)} + v_{j_s+1} \uparrow M_k) / M_k) + R_k^{(j)}, \qquad \text{(i)}$$

gdzie v_{j_s+1} jest przejściem pomiędzy $x_{j_s} \to x_{j_s+1}$.

Zauważmy, że wykorzystujemy tu założenie o poprawności ciągu $s_2(\bullet , x_j)$. Do obliczenia prawej strony możemy się posłużyć powyższą formułą, ponieważ w ciągu tym zachowana jest odwrotna dopuszczalność. Dla rozwiązań, w których *k*-ta akcja w przestrzeni P nie jest odwrotnie dopuszczalna odpowiadający jej wiersz macierzy przyjmuje wartość zerową i nie jest możliwe wykonywanie akcji ze zbioru określonego przez wektor M_k.



Rys. 6-8 Ilustracja do dowodu twierdzenia

Wyznaczmy w zbiorze indeksów $J = \{j_1,\ldots, j_{t-1}\}$ podzbiór J' odpowiadający tym rozwiązaniom, po których nastąpiła zmiana k-tej składowej wektora w przestrzeni obserwacji.

$$r(\,(x_j \uparrow M_k) / M_k) = \sum_{s\in J'} r((R_k^{(s)} + v_{s+1} \uparrow M_k) / M_k) + \sum_{s\in J\setminus J'} R_k^{(s)} + R_k^{(j)} \quad \text{(ii)}$$

Dla każdego $s \in J'$ spełnione jest więc:

$$R_k^{(s)} + v_{s+1} \uparrow M_k \geq M_k \text{ oraz} \quad \text{(iii)}$$

$$r_{s+1} = r((R_k^{(s)} + v_{s+1} \uparrow M_k) / M_k) = (R_k^{(s)} + v_{s+1} \uparrow M_k) - M_k . \quad \text{(iv)}$$

Skorzystajmy teraz z założenia (4). Wynika z niego, że dla każdego wiersza wektora $R_k^{(s)}$ występującego w obu ciągach istnieje zbiór akcji NR_k taki, że

$$\forall nr \in NR_k . R_k^{(s)}(nr) = 0 \wedge M_k(nr) = 1 \quad \text{(v)}$$

Na mocy (iv), (v) oraz z założenia, że wektory są nieujemne i wektory przejść elementarnych mają składowe jedynie ze zbioru {0, 1} mamy:

$$\forall s \in J' . \forall nr \in NR_k . v_{s+1}(nr) = 1, \text{ stąd}$$

$$\forall s \in J' . \forall nr \in NR_k . r_{s+1}(nr) = 0 \quad \text{(vi).}$$

Z (vi) oraz (ii) wynika, że wektor $z_j = r(\,(x_j \uparrow M_k) / M_k)$ spełnia również zależność:

$$\forall\ nr \in NR_k\ .\ z_j(nr) = 0 \quad \text{(vii)}$$

Skorzystajmy z tego ostatniego rezultatu, aby obliczyć k–tą składowej wektora $\Delta p' = \pi_{Min}(x + v) - \pi_{Min}(x)$ i porównać z wartością $\Delta p''$ wyznaczoną na podstawie macierzy lokalnej osiągalności $R^{(j)}$ i przejścia v .

Rozważmy dwa przypadki:

Załóżmy, że $\Delta p'(k) = 1$ i $\Delta p'' = 0$. Z drugiej równości wynika, że nie zachodzi $R_k^{(j)} + v \uparrow M_k \geq M_k$, a zatem $\exists\, s_0 \in NR_k\ .\ v(s_0) = 0$. Z drugiej strony, z założenia, że $\Delta p'(k) = 1$ wynika, że $r(\,(x_j \uparrow M_k) / M_k) + v \uparrow M_k \geq M_k$, a zatem wektor $z_j = r(\,(x_j \uparrow M_k) / M_k)$ musi spełniać : $z_j(s_0) \geq 0$. To z kolei prowadzi do sprzeczności z (vii).

Załóżmy, że $\Delta p'(k) = 0$ i $\Delta p'' = 1$. To założenie prowadzi natychmiast do sprzeczności, ponieważ $R_k^{(j)}$ jest użyty do obliczenia $\Delta p''$ i równocześnie na mocy (ii) wchodzi w skład sumy $z_j = r(\,(x_j \uparrow M_k) / M_k)$.

■

6.3.5 Konstrukcja procesu sprzężonego

Podobnie, jak dla homomorficznej funkcji obserwacji, proces sprzężony jest podstawowym narzędziem badania poprawności. Istotną różnicą jest tu rozszerzenie składowej stanu procesu sprzężonego o macierz lokalnej osiągalności.

Stan procesu sprzężonego dla niehomomorficznej funkcji obserwacji, będzie zapisywany jako para $(x\ ,\ (\ y\ ,\ y'\ ,\ R)\ )$, gdzie x jest rozwiązaniem modelu *ML*, y odpowiadającym mu stanem, y' stanem modelu kryterialnego, natomiast R jest macierzą lokalnej osiągalności. Macierz R pozwala na zapisanie historii dojścia do stanu modelu weryfikowanego y. Na podstawie zawartości macierzy można określić przyszłe obserwacje dokonywane w wyniku wykonania przejść osiągalnych z danego stanu y specyfikacji weryfikowanej.

Tak więc stany $(x_1 , (y_1 , y_1', R^{(1)}))$ i $(x_2 , (y_2 , y_2', R^{(2)}))$ dla różnych (nierównoważnych) macierzy $R^{(1)}$ i $R^{(2)}$ będą rozróżnialne nawet dla $y_1 = y_2$ i $y_1' = y_2'$ ze względu na to, że dane przejście elementarne v dopuszczalne w obu stanach y_1 i y_2 może być źródłem obserwacji różnych przejść procesu kryterialnego.

Biorąc pod uwagę fakt, że w przedstawionym dalej algorytmie weryfikacji (patrz dodatek C) osiągnięte stany procesu sprzężonego są przechowywane, rozszerzenie składowej stanu o macierz może się wydawać bardzo nieefektywne. Macierz lokalnej osiągalności R o wymiarach $r \times n$ w implementacji algorytmu reprezentowana jest jako rzadki wektor o wymiarze $r \cdot n$, zazwyczaj kilka wierszy macierzy R ma niezerowe elementy i mogą się one pojawiać jedynie w kilku kolumnach odpowiadających niezerowym wartościom elementów wierszy macierzy M. Stąd, w praktycznych zastosowaniach jest ona konstrukcją efektywną.

Niech $ML = (X, x_0, A_I\, x \le b, A_E\, x = 0)$ będzie modelem liniowym specyfikacji. Oznaczmy przez Y przestrzeń stanów modelu ML. Zakładamy, że zbiór stanów osiągalnych modelu jest skończony.

Podobnie niech $ML' = (X', x_0', A_I'\, x' \le b', A_E'\, x' = 0)$ będzie modelem specyfikacji kryterialnej, Y' jego przestrzenią stanów. Niech π oznacza niehomomorficzną funkcję obserwacji. Zakładamy, że $\pi(x_0) = x_0'$.

Proces sprzężony postaci $\hat{P} = (\hat{S}, \hat{B}, \hat{F}, \hat{T})$ jest skonstruowany jako granica ciągu procesów $\Theta = \langle P^{(0)},\ldots, P^{(i)},\ldots, P^{(n)},\ldots\rangle$ postaci $P^{(i)} = (S^{(i)}, B^{(i)}, F^{(i)}, T^{(i)})$, gdzie $S^{(i)} \subset X \times \mathbf{Y}$, natomiast $\mathbf{Y} = Y \times Y' \times (\mathbf{R}^r)^n$.

Podobnie, jak dla liniowej funkcji obserwacji spełnione są zależności:

$$\hat{S} = \bigcup_{i=0}^{\infty} S^{(i)}$$

$$\hat{B} = \bigcup_{i=0}^{\infty} B^{(i)}$$

$$\hat{F} = \bigcup_{i=0}^{\infty} F^{(i)}$$

$$\hat{T} = \bigcup_{i=0}^{\infty} T^{(i)}$$

Proces rozpoczynający ciąg opisany jest następującymi równaniami:

1. $S^{(0)} = B^{(0)} = \{ (x_0 , (y_0 , y_0', R^{(0)}))\}$, gdzie $y_0 = y(x_0)$, $y_0' = y(x_0')$, macierz początkowa jest określona wzorem Def. 6-5 (1).
2. $F^{(0)} = \begin{cases} B^{(0)} & \text{jeżeli } V_F(x_0) = \varnothing \wedge \forall i \in N . A_i x_0 = 0 \\ \varnothing & \text{w p.p.} \end{cases}$
3. $T^{(0)} = \varnothing$

Przejście od $P^{(i-1)}$ do $P^{(i)}$ następuje jeśli:

$\exists\, s_k \in S^{(i-1)} \setminus F^{(i-1)} . s_k = (x_k,(y_k, y_k', R^{(k)})) \wedge$

$\exists\, v \in V_F(y_k) . s_i =((x_k + v), (y_i, y_i', R^{(i)})) \notin S^{(i-1)}$, gdzie:

$y_i = y(x_k + v)$,

$y_i' = y(\pi(x_k + v))$,

$R_i = lrm(R^{(k)}, v)$

Wówczas konstruowany jest nowy proces $P^{(i)}$ postaci:

1. $S^{(i)} = S^{(i-1)} \cup \{ s_i \}$, $s_i = ((x_k + v), (y_i, y_i', R^{(i)}))$
2. $B^{(i)} = B^{(i-1)}$
3. $T^{(i)} = T^{(i-1)} \cup \{(s_k, s_i)\}$
4. Zbiór $F^{(i)}$ modyfikowany jest następująco:

   a) Jeśli $V_F(y_i) = \varnothing \wedge \forall\, i \in N\,.\, A_I\, x_i = 0$ to $F^{(i)} = F^{(i-1)} \cup \{ s_i \}$

   b) Jeśli $V_F(y_i) \neq \varnothing$ oraz istnieje stan $s_t \in S^{(i-1)}$, $s_t = (x_t, (y_t, y_t', R^{(t)}))$ spełniający:

   $(y_t = y_i) \wedge (y_t' \leq y_i') \wedge (R^{(t)} = R^{(i)} \vee R^{(t)} \prec R^{(i)})$ to $F^{(i)} = F^{(i)} \cup \{ s_i \}$

   c) Jeśli $V_F(y_i) \neq \varnothing$ oraz istnieje stan $s_t \in S^{(i-1)}$, $s_t = (x_t, (y_t, y_t', R^{(t)}))$ spełniający:

   $(y_t \prec y_i) \wedge (y_t' \leq y_i') \wedge (R^{(t)} = R^{(i)} \vee R^{(t)} \prec R^{(i)})$ to $F^{(i)} = F^{(i)} \cup \{ s_i \}$

   d) Jeżeli nie zachodzi żaden z powyższych przypadków $F^{(i)} = F^{(i-1)}$

■

Podobnie, jak w przypadku liniowej funkcji obserwacji ograniczamy się do weryfikacji dla skończonego zbioru stanów osiągalnych modelu ML; stąd specyfikację traktujemy jako częściowo weryfikowalną, jeśli zachodził będzie warunek 4.c .

Def. 6-8 Poprawność przejścia procesu sprzężonego

Niech $(s_k, s_r) \in T^{(i)}$ będzie przejściem w procesie $P^{(i)}$. Oznaczmy $s_k = (x_k, (y_k, y_k', R^{(k)}))$ oraz $s_r = (x_r, (y_r, y_r', R^{(r)}))$.

Mówimy, że przejście (s_k, s_r) jest poprawne, jeżeli para (x_k, x_r) spełnia warunki poprawności określone w Def. 6-4 pkt.1. :

■

Twierdzenie 6-2 Skończoność procesu sprzężonego

Proces sprzężony $\hat{P}$, którego każde przejście jest poprawne jest skończony.

■

Dowód. Dowód jest niewielką modyfikacją dowodu twierdzenia o skończoności procesu sprzężonego dla liniowej funkcji obserwacji (Tw. 5–4). Zwróćmy uwagę, że uwzględnić należy wyłącznie rozszerzenie składowej stanu o macierz lokalnej osiągalności R. Macierz ta może być przedstawiona w postaci wektora o rozmiarach $r \cdot n$. Podobnie, jak dla wektora y' jej elementy mogą przyjmować jedynie nieujemne wartości i nieograniczony wzrost elementów macierzy jest dozorowany przez warunki na pokrywanie, stąd dowód przebiegał będzie analogicznie.

6.3.6 Zastosowanie procesu sprzężonego w weryfikacji poprawności

Idea zastosowania procesu sprzężonego w weryfikacji poprawności dla niehomomorficznej funkcji obserwacji jest podobna jak dla funkcji liniowej. W rozdziale 5 zdefiniowano relację $<\approx$ określającą podobieństwo rozwiązań (Def. 5-8) . Dla danych dwóch rozwiązań

x_i oraz x_j spełnienie warunku $x_i <\approx x_j$ pozwalało przenieść poprawność rozwiązań osiągalnych z x_i na poprawność wszystkich rozwiązań osiągalnych z x_j . Podstawą do stwierdzenia istnienia relacji $<\approx$ pomiędzy rozwiązaniami było spełnienie odpowiednich warunków przez wartości odpowiadające składowej stanu procesu sprzężonego.

W przypadku niehomomorficznej funkcji obserwacji nie możemy bezpośrednio zbudować relacji pomiędzy rozwiązaniami, ponieważ uzależnieni jesteśmy także od historii dojścia do danego stanu. Natomiast możliwe jest określenie relacji pomiędzy składowymi stanu, które tę historię dojścia opisują.

Def. 6-9 Relacja pokrywania

Niech Y będzie zbiorem stanów modelu *ML*; Y' będzie zbiorem stanów modelu *ML*', *RS* zbiorem wszystkich macierzy lokalnej obserwacji.

Zdefiniujmy relację $<\approx \subset (\mathrm{Y} \times \mathrm{Y'} \times RS) \times (\mathrm{Y} \times \mathrm{Y'} \times RS)$ jako:

$$(y_1,\ y_1',\ \mathrm{R}^{(1)}) <\approx (y_2,\ y_2',\ \mathrm{R}^{(2)}) \Leftrightarrow$$
$$y_1 = y_2 \wedge y_1' \le y_2' \wedge \mathrm{R}^{(1)} \approx \mathrm{R}^{(2)}$$

■

Relacja $<\approx$ pomiędzy składową pamięci procesu sprzężonego występuje w założeniach twierdzenia o ekstrapolacji poprawności (Twierdzenie 6-1).

Wniosek 6-3

Z zasad konstrukcji procesu sprzężonego wynika, że jeśli każde jego przejście jest poprawne, wówczas spełniony jest następujący warunek:

$$\forall\ s_k \in (\hat{S} \setminus \hat{F}) : s_k = (x_k,(\ y_k, y_k', \mathrm{R}^{(k)})) .\ \forall v \in V_F(\ y_k) .\ \exists\ s_r \in \hat{S} .\ s_r = (x_r,(\ y_r, y_r', \mathrm{R}^{(r)})) ,$$

gdzie $y_r = y_k - \mathrm{A_I}\ v$ i $y_r' = y_k' - \mathrm{A_I}'\ \pi(v)$ oraz $\mathrm{R}^{(r)} = lrm(\mathrm{R}^{(k)}, v)$.

■

Podobnie, jak dla procesu sprzężonego dla homomorficznej funkcji obserwacji zdefiniujmy $\hat{F}_* = \{\ s_r \in \hat{F} \mid s_r = (x_r,(\ y_r, y_r', \mathrm{R}^{(r)})) \wedge \mathrm{V}_F(y_r) \neq \varnothing\ \}$. Elementami tego zbioru są stany, które zostały osiągnięte co najmniej dwukrotnie w trakcie generacji ciągu procesów lub stany je pokrywające.

Wniosek 6-4

Zbiór $\hat{F}_*$ spełnia następujący warunek:

1. $\forall\ s_f = (x_f,(\ y_f, y_f', \mathrm{R}^{(f)})) \in \hat{F}_* .\ \exists\ s_k = (x_k, (\ y_k, y_k', \mathrm{R}^{(k)})) \in \hat{S} \setminus \hat{F}$.

   $(\ y_k, y_k', \mathrm{R}^{(k)}) <\approx (\ y_f, y_f', \mathrm{R}^{(f)})$

2. Jeśli przejście $(s_k, s_{k+1}) \in \hat{T}$ jest poprawne, gdzie

   $(s_k, s_{k+1}) = (\ (x_k,(\ y_k, y_k', \mathrm{R}^{(k)})) , (x_{k+1},(\ y_{k+1}, y_{k+1}', \mathrm{R}^{(k+1)})))$,

oraz spełniony jest warunek (4) twierdzenia o ekstrapolacji poprawności

wówczas dla $v = x_{k+1} - x_k$ przejście z rozwiązania x_f do rozwiązania $x_f + v$ jest poprawne oraz spełnione jest:

$$(\ y_{k+1}, y_{k+1}', \mathrm{R}^{(k+1)}) <\approx (\ y_{f+1}, y_{f+1}', \mathrm{R}^{(f+1)}),$$

gdzie $x_{f+1} = x_f + v$, $y_{f+1} = y(x_{f+1})$, $y_{f+1}' = y(\pi(x_{f+1}))$, $R^{(f+1)} = lrm(R^{(f)}, v)$

■

Uzasadnienie

Stwierdzenie (1) wynika bezpośrednio z zasad konstrukcji procesu sprzężonego.

Stwierdzenie (2) wynika bezpośrednio z (1) oraz twierdzenia o ekstrapolacji poprawności..

□

Twierdzenie 6-3

Jeśli zbiór osiągalnych stanów Y modelu *ML* jest skończony i wszystkie przejścia procesu sprzężonego są poprawne, to dla każdego rozwiązania osiągalnego x_n w pewnym ciągu $s(x_0 , x_n)$:

$$\exists\, s_k \in \hat{S}\,,\ s_k = (x_k\,,\ (\,y_k, y_k', R^{(k)})) \,.\, (\,y_k, y_k', R^{(k)}) <\approx (\,y_n, y_n', R^{(n)}),$$

gdzie $R^{(n)}$ jest macierzą lokalnej osiągalności odpowiadającą rozwiązaniu x_n obliczoną w ciągu $\rho(\,s(x_0, x_n)\,) = \langle\, R^{(0)}, R^{(1)}, \ldots, R^{(n)} \,\rangle$

■

Dowód. Jeśli rozwiązanie x_n zostało wyznaczone podczas konstrukcji procesu sprzężonego, wówczas ta zależność zachodzi.

Rozważmy dowolny ciąg rozwiązań osiągalnych, który zawiera rozwiązania nie wyznaczone przez proces sprzężony. Jego podciąg początkowy musi tworzyć semiobliczenie procesu sprzężonego. Stąd załóżmy, że ma on postać:

$$s(x_0,\bullet) = \langle\, x_0, \ldots, x_f, x_{f+1}, \ldots, x_{f+i-1}, x_{f+i}, \ldots \,\rangle$$

Załóżmy, że odpowiadający mu ciag macierzy lokalnej osiągalności ma postać

$$\rho(\,s(x_0,\bullet)\,) = \langle\, R^{(0)}, R^{(1)}, \ldots, R^{(f)}, R^{(f+1)}, \ldots, R^{(f+i-1)}, R^{(f+i)}, \ldots \rangle$$

Załóżmy, że x_f jest ostatnim rozwiązaniem należącym do ciągu rozwiązań wyznaczonego przez semiobliczenie procesu sprzężonego, wówczas na mocy poprzedniego wniosku (Wniosek 6-4) warunek określony w tezie zachodzi dla x_{f+1}.

Załóżmy, że rozwiązanie x_{f+i-1} spełnia warunek określony w tezie, czyli istnieje stan procesu sprzężonego $s_k = (x_k\,,\ (\,y_k, y_k', R^{(k)}))$ spełniający:

$$(\,y_k, y_k', R^{(k)}) <\approx (\,y_{f+i-1}, y_{f+i-1}', R^{(f+i-1)}).$$

Jeżeli stan $s_k \in \hat{F}_*$ wówczas istnieje również stan $s_r = (x_r,(\,y_r, y_r', R^{(r)})) \in \hat{S} \setminus \hat{F}$ spełniający $(\,y_r, y_r', R^{(r)}) <\approx (\,y_k, y_k', R^{(k)})$ stąd $(\,y_r, y_r', R^{(r)}) <\approx (\,y_{f+i-1}, y_{f+i-1}', R^{(f+i-1)})$, a więc możemy się ograniczyć do przypadku, kiedy $s_k \in \hat{S} \setminus \hat{F}_*$.

Jeżeli $V_F(y_k) = \varnothing$, wówczas na mocy równości stanów y_k oraz $y(\,x_{f+i-1}\,)$ przejście do następnego rozwiązania jest niemożliwe.

Jeżeli $V_F(y_k) \neq \varnothing$, wówczas przejście $v = x_{f+i} - x_{f+i-1}$ musi spełniać $v \in V_F(x_k) = V_F(x_{f+i-1})$. Stąd stan $s_{k+1} = (x_r,(\,y_{k+1}, y_{k+1}', R^{(k+1)}))$ jest wyznaczony w trakcie konstrukcji procesu sprzężonego. Zachodzi zatem $(\,y_{k+1}, y_{k+1}', R^{(k+1)}) <\approx (\,y_{f+i-1}, y_{f+i-1}', R^{(f+i-1)})$.

□

Wniosek 6-5

Niech $\hat{P}$ będzie procesem sprzężonym. Oznaczmy przez $s(x_0, \bullet)$ dowolny ciąg osiągalnych rozwiązań modelu *ML*. Niech $\rho(s(x_0, x_i))$ będzie odpowiadającym mu ciągiem macierzy lokalnej osiągalności.

$$\rho(s(x_0, \bullet)) = \langle R^{(0)}, R^{(1)}, \ldots, R^{(i)}, \ldots \rangle$$

Oznaczmy przez $Z(\rho(s(x_0, \bullet)))$ zbiór wszystkich macierzy lokalnej osiągalności należących do ciągu $\rho(s(x_0, \bullet))$.

Jeśli zbiór osiągalnych stanów Y modelu *ML* jest skończony i wszystkie przejścia procesu sprzężonego są poprawne, to

$$\forall R \in Z(\rho(s(x_0, \bullet))) . \exists s_k \in \hat{S}, s_k = (x_k, (y_k, y_k', R^{(k)})) . R^{(k)} \approx R$$

■

Wniosek ten jest bezpośrednią konsekwencją powyższego twierdzenia.

Oznaczmy przez $RS(\hat{P}) = \{R | \exists s \in \hat{S}.s = (x,(y,y',R))\}$. Zbiór $RS(\hat{P})$ jest zbiorem wszystkich macierzy lokalnej osiągalności wyznaczonych w trakcie budowy procesu sprzężonego.

Wniosek 6-6

Z definicji relacji równoważności pomiędzy macierzami oraz poprzedniego wniosku wynika, że jeśli wszystkie macierze zbioru $RS(\hat{P})$ spełniają warunek (4) twierdzenia o ekstrapolacji poprawności (Twierdzenie 6-1), to spełniają go wszystkie macierze należące do zbioru *RS*. ■

Twierdzenie 6-4

Załóżmy, że

1. wszystkie przejścia procesu sprzężonego są poprawne,
2. każda macierz $R \in RS(\hat{P})$ spełnia warunek:

$$\forall k \in \{1,\ldots,r\}.\exists NR_k \subset \{1,\ldots,n\}.\forall nr \in NR_k.R_k(i) = 0 \wedge M_k(i) \neq 0$$

Dowolne przejście elementarne pomiędzy dwoma kolejnymi rozwiązaniami osiągalnymi modelu *ML* jest poprawne w sensie kryterium (π, ML')

■

Dowód. Rozważmy parę kolejnych rozwiązań osiągalnych (x_{i-1}, x_i) w pewnym ciągu $s(x_0, x_i)$. Niech $\rho(s(x_0, x_i))$ będzie odpowiadającym mu ciagiem macierzy lokalnej osiagalności.

Na mocy twierdzenia 6-3, istnieje stan procesu sprzężonego $s_k \in \hat{S} \setminus \hat{F}_*$, $s_k = (x_k,(y_k, y_k', R^{(k)}))$, dla którego zachodzi $(y_k, y_k', R^{(k)}) <\approx (y_{i-1}, y_{i-1}', R^{(i-1)})$. Założenie (2) powyższego twierdzenia jest przekształconym założeniem (4) twierdzenia o ekstrapolacji poprawności oraz konsekwencją wniosku 6-5.

Stąd zgodnie z twierdzeniem o ekstrapolacji poprawności przejście do rozwiązania x_i jest poprawne.

□

6.3.7 Dyskusja rozstrzygalności

Dowód twierdzenia o ekstrapolacji poprawności (Twierdzenie 6-1) oparto między innymi na założeniu, że dla każdej możliwej do wyznaczenia macierzy lokalnej osiągalności jej dowolny *k*–ty wiersz nie zawierał elementów niezerowych w kolumnach pewnego zbioru $NR_k \subset idxset(M_k)$.

Narusznie tego warunku najczęściej prowadzi do stwierdzenia niepoprawności podczas weryfikacji. W przypadku, kiedy warunek nie jest spełniony i weryfikacja nie zwróciła informacji o błędach, jej wynik musi być uznany za nierozstrzygalny.

Nierozstrzygalność dotyczy szczególnych przypadków specyfikacji, kiedy zbiór akcji określony przez wektor M_k odwzorowany jest w „oczko" w specyfikacji kryterialnej.

Przykład takiej specyfikacji przedstawiono na Rys. 6-9 . Załóżmy, że funkcja obserwacji ma postać:

$$\alpha' = \alpha_1 \circ \alpha_2 \circ \alpha_3$$

Możliwe jest tu wyznaczenie miedzy innymi następujących wierszy macierzy lokalnej osiągalności odpowiadających akcji α' :

$R^{(a)} = [\alpha_1 : 1; \alpha_2 : 0; \alpha_3 : 1]$,

$R^{(b)} = [\alpha_1 : 2; \alpha_2 : 0; \alpha_3 : 1]$,

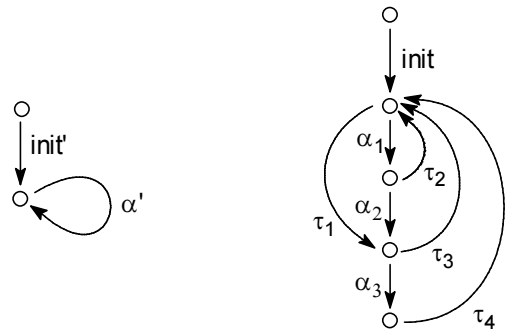
$R^{(c)} = [\alpha_1 : 1; \alpha_2 : 0; \alpha_3 : 2]$,

$R^{(d)} = [\alpha_1 : 1; \alpha_2 : 1; \alpha_3 : 0]$,

$R^{(e)} = [\alpha_1 : 2; \alpha_2 : 1; \alpha_3 : 0]$,

…

Równocześnie, co można sprawdzić „manualnie" specyfikacja jest poprawna.



Rys. 6-9 Specyfikacja poprawna, dla której weryfikacja w oparciu o proces sprzężony jest nierozstrzygalna

Przykład przedstawiony na Rys. 6-10 jest niewielką modyfikacją poprzedniego. Dodano w nim dwie akcje β i γ , które są przez funkcję obserwacji odwzorowywane w β' i γ'.

Przypadek ten jest błędny, co można prześledzić dla ścieżki

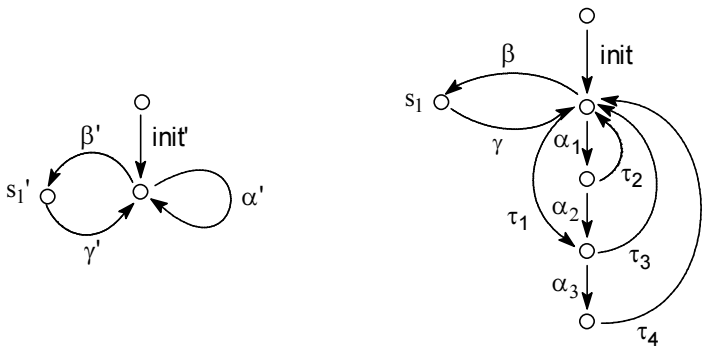
$$t = \langle init, \tau_1, \alpha_3 , \tau_4, \beta, \gamma , \alpha_1, \alpha_2 \rangle .$$

Ścieżka ta narusza własność lokalnej osiągalności, ponieważ obliczenie α' za pomocą funkcji obserwacji nie odpowiada obliczeniu z wykorzystaniem macierzy R (akcja α_3 nie

została wykonana w lokalnym kontekście). Błędna ścieżka tej postaci zostanie wykryta przez skonstruowany algorytm *jeżeli zostanie wygenerowana*. Tak jednak być nie musi. Jeżeli wcześniej przeszukiwano ścieżki postaci :

$$t' = \langle init, \beta, \gamma, \ldots \rangle ,$$

wówczas ścieżka t zostanie obcięta do $t'' = \langle init, \tau_1, \alpha_3, \tau_4, \beta \rangle$, ponieważ osiągnięty zostanie poprzednio wyznaczony stan procesu sprzężonego $(\bullet, (s_1, s_1', \mathbf{0}))$.



Rys. 6-10 Specyfikacja niepoprawna, dla której weryfikacja w oparciu o proces sprzężony jest nierozstrzygalna

Jak łatwo zauważyć, usunięcie przejścia τ_1 spowoduje, że specyfikacja stanie się poprawna.

Równocześnie wiersze macierzy lokalnej osiągalności związane z akcją α' przyjmowały będą wartości typu:

$R^{(a)} = [\alpha_1 : 1; \alpha_2 : 0; \alpha_3 : 0]$,

$R^{(b)} = [\alpha_1 : 2; \alpha_2 : 0; \alpha_3 : 0]$,

$R^{(c)} = [\alpha_1 : 1; \alpha_2 : 1; \alpha_3 : 0]$,

$R^{(d)} = [\alpha_1 : 2; \alpha_2 : 1; \alpha_3 : 0]$,

…

W możliwych do wyznaczenia macierzach akcji α_3 zawsze będzie odpowiadała wartość zerowa, będzie ona zawsze domykała obserwację przejścia α'. W tym przypadku spełnione będą warunki opisujące rozstrzygalność.

6.3.8 Uporządkowanie dla macierzy lokalnej osiągalności

Niech k będzie indeksem obserwowalnej akcji w przestrzeni P. Warunek rozstrzygalności problemu poprawności dla niehomomorficznej funkcji obserwacji można sprowadzić do żądania, aby obserwacja k–tej akcji była zawsze domykana przez to samo przejście. Spełnienie tego założenia wymaga, by istniało pewne uporządkowanie wierszy macierzy lokalnej osiągalności. Uporządkowanie to określone jest przez relację inkluzji zbiorów akcji akumulowanych w wierszach.

Niech RS_k oznacza zbiór k-tych wierszy macierzy $R \in RS$. Niech

$$IDXR_k = \bigcup_{R_k \in RS_k} idxset(R_k)$$

Warunek rozstrzygalności można wyrazić jako:

$\exists I_{\text{sup}} \in IDXR_{\text{k}} \,.\, I_{\text{sup}} = sup_{\subseteq}\, IDXR_{\text{k}}$

Wymagamy więc, aby dla każdego wiersza macierzy istniał kres górny zbioru akcji opisanych przez jego elementy. (Kres dolny również istnieje i jest nim co najmniej zbiór pusty.)

Analiza wielu przykładów wskazuje, że poza szczególnymi przypadkami podobnymi do tego z Rys. 6-9 (podproces, którego nie można opuścić lub do którego nie można powrócić odwzorowany w „oczko") warunkiem poprawności częściowej jest znacznie ostrzejsze wymaganie dotyczące uporządkowania:

$$\forall k \in \{1,\ldots,r\}.\forall R_k^1, R_k^2 \in RS_k.(idxset(R_k^1) \subseteq idxset(R_k^2) \vee idxset(R_k^1) \supseteq idxset(R_k^2)),$$

czyli, że zbiory niezerowych elementów poszczególnych wierszy macierzy lokalnej osiągalności są dobrze uporządkowane.

Powyższe stwierdzenie należy traktować jako hipotezę. Własność ta jest łatwa do zaobserwowania dla pojedynczych procesów sekwencyjnych (bez komunikacji). W tym przypadku wydaje się możliwe przeprowadzenie dowodu indukcyjnego.

Zauważmy, że powyższy warunek nie zabrania wystąpienia pokrywania; spełniony jest na przykład przez specyfikację z Rys. 6-10 po usunięciu akcji τ_1.

Wymaganie, by zbiory akcji opisanych przez wiersze macierzy lokalnej osiągalności były dobrze uporządkowane wydaje się łatwe do uzasadnienia. Traktując zbiór akcji procesu weryfikowanego, jako rezultat hierarchicznej dekompozycji mamy prawo oczekiwać, że będą one wykonywane w określonej kolejności.

W przypadku dekompozycji funkcjonalnej i specyfikacji funkcji z użyciem instrukcji strukturalnych wysokiego poziomu takie uporządkowanie jest zazwyczaj automatycznie narzucane przez samą składnię języka. Znacznie trudniejszym do analizy jest stwierdzenie uporządkowania w przypadku, kiedy obserwowane akcje składające się na pewien wymagany podproces należą do różnych współbieżnych procesów składowych.

Problem badania uporządkowania może stać się jednym z kierunków dalszych prac dotyczących specyfikacji wymagań poprawnościowych z użyciem niehomomorficznej funkcji obserwacji.

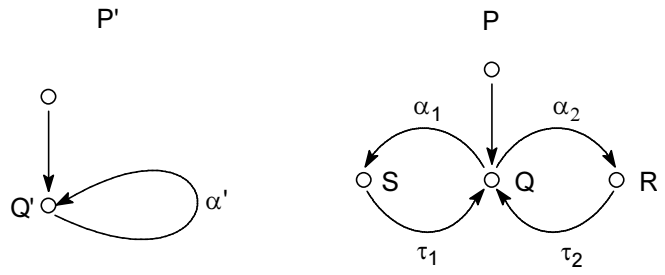
6.4 Deterministyczna zgodność funkcji obserwacji

Niech h będzie odwzorowaniem, które posłużyło nam do zdefiniowania π_{Min} – niehomomorficznego składnika funkcji obserwacji (Por. 3.4.2). Niech $D \subset Dom(h)$ będzie pewnym zbiorem obserwowalnych akcji. Zgodnie z zaproponowaną w podrozdziale 6.1.6 ideą dekompozycji weryfikowanego procesu, pragniemy powiązać ze zbiorem D zarówno podproces procesu weryfikowanego P_D jak i jego specyfikację $PSPEC_D$.

Przedstawione w podrozdziale 6.1.6 specyfikacje podprocesów otrzymywanych w wyniku dekompozycji zakładają, że ich ścieżki powinny być skonstruowane jako wariacje (z powtórzeniami) akcji należących do zbioru $D \setminus \{act\}$. Przy takich założeniach, celem funkcji obserwacji jest odwzorowanie fragmentu drzewa operacji występującego w specyfikacji weryfikowanej w pojedynczą akcję procesu reprezentującego specyfikację kryterialną.

W ogólnym przypadku struktura opisana przez funkcję obserwacji nie musi wystąpić w strukturze obserwowanego procesu. Wówczas obserwacja danej akcji procesu kryterialnego może nie zostać nigdy dokonana lub będzie się pojawiać chaotycznie.

Rozważmy przykład procesów przedstawiony na Rys. 6-11 i załóżmy, że funkcja obserwacji ma postać: $\alpha' = \alpha_1 \circ \alpha_2$. Prowadząc nieskończoną obserwację zachowania procesu *P* możemy nigdy nie zarejestrować pojawienia się symbolu α', zarejestrować skończony ciąg symboli α' lub ciąg nieskończony.



Rys. 6-11 Funkcja obserwacji nie odzwierciedlająca struktury podprocesów

Wynika to stąd, że podprocesy mają strukturę opisaną w formalizmie CSP jako:

$$Q = \alpha_1 \rightarrow \tau_1 \rightarrow Q \mid \alpha_2 \rightarrow \tau_2 \rightarrow Q$$

$$Q' = \alpha' \rightarrow Q'$$

Podczas gdy funkcja obserwacji implikuje specyfikację strukturę pewnego podprocesu *P* postaci:

$$QSPEC = (\alpha_1)^n \rightarrow \alpha_2 \rightarrow QSPEC \sqcap (\alpha_1)^n \rightarrow \alpha_1 \rightarrow QSPEC$$

Podstawową różnicą jest operator | oznaczający niedeterministyczny wybór pomiędzy podjęciem akcji α_1 lub α_2.

W typowym przypadku strukturalny niedeterminizm w procesie powstałym w wyniku dynamicznie rozwijanej dekompozycji będzie naruszał własność lokalnej osiągalności. Tak jednak nie będzie dla przedstawionego przykładu, ponieważ akcje α_1 i α_2 będą *stale* odwrotnie dopuszczalne ze względu na akcję α'.

Trudno dla opisanego przykładu jednoznacznie negatywnie ustosunkować się do tego zjawiska, ponieważ otrzymany w wyniku dekompozycji proces istnieje i ma postać

$$S = \tau_1 \rightarrow \alpha_2 \rightarrow \tau_2 \rightarrow \alpha_1 \rightarrow S$$

lub

$$R = \tau_2 \rightarrow \alpha_1 \rightarrow \tau_1 \rightarrow \alpha_2 \rightarrow R.$$

Z tego powodu przyjęto, że opisywane zachowanie nie jest traktowane jako brak poprawności. Może być ono jednak wykrywane w trakcie weryfikacji. Zjawisko to będzie określone jako brak *deterministycznej zgodności* funkcji obserwacji z procesem weryfikowanym.

Jak łatwo zauważyć, chaotyczny charakter wykonywanych obserwacji uzależniony jest w takim przypadku jedynie od postaci funkcji obserwacji oraz struktury procesu weryfikowanego i może być rozpatrywany w oderwaniu od specyfikacji procesu kryterialnego. Ze względu na operator niedeterministycznego wyboru w specyfikacji procesu CSP *Q*, akcja α' może zostać zaobserwowana jedynie dla niedeterministycznych ciągów rozwiązań.

Def. 6-10 Ciąg deterministyczny

Niech $s(v_1,\bullet) = \langle v_{J(1)},\ldots, v_{J(i)},\ldots \rangle$ będzie ciągiem przejść elementarnych rozpoczynających się od rozwiązania x_0.

Ciąg $s(v_1,\bullet)$ generuje ciąg rozwiązań postaci $s(x_0,\bullet) = \langle x_0, x_1,\ldots, x_i,\ldots\rangle$, gdzie $x_i = x_0 + \sum_{j\le i} v_i$.

Ciąg $s(x_0,\bullet)$ nazywany jest deterministycznym, jeżeli zachodzi

$$\forall\, x_s, x_t \in Z(s(x_0,\bullet))\ .\ y(x_s) = y(x_t) \Rightarrow v_{J(s)} = v_{J(t)}$$

■

Ciągiem niedeterministycznym będziemy nazywali ciąg rozwiązań niezgodny z powyższą definicją.

Def. 6-11 Zgodność funkcji obserwacji z procesem weryfikowanym

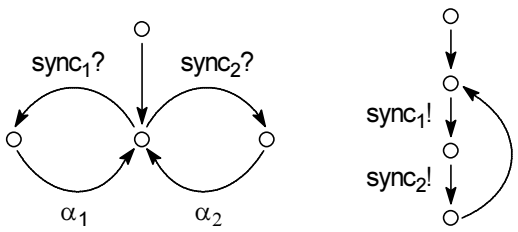
Niech $P = R^m$ będzie przestrzenią liniowej obserwacji. Funkcja obserwacji jest zgodna z procesem weryfikowanym, jeśli z tego, że istnieje ciąg niedeterministyczny $s_{nd}(x_0, x_r)$ taki, że $p_r = \pi_{Min}(x_r)$ i zachodzi $p_r(i) \neq 0$ wynika, że istnieje ciąg deterministyczny $s_d(x_0, x_s)$ taki, że $p_s = \pi_{Min}(x_s) \neq 0$ i $p_s(i) \neq 0$.

■

Będziemy mówić, że *i*-ta akcja procesu kryterialnego jest *deterministycznie obserwowalna*, jeżeli istnieje ciąg deterministyczny $s_d(x_0, x_s)$, taki że $x' = \pi(x_s)$ i zachodzi $x'(i) \neq 0$.

W ogólnym przypadku, konstrukcja algorytmu weryfikacji (patrz: dodatek C) nie pozwala na rozstrzyganie o zgodności funkcji obserwacji z procesem weryfikowanym i deterministycznej obserwowalności, ponieważ budowano go z myślą o weryfikacji. W tym celu stworzono jego zmodyfikowaną wersję, która generuje wyłącznie ścieżki deterministyczne i tylko dla niech prowadzi zgrubną weryfikację.

Warto zwrócić uwagę, że brak deterministycznej zgodności nie może zostać rozstrzygany przez bezpośrednią analizę struktury wybranego procesu, ale musi być rozpatrywany dla całego systemu. Przykładem tego może być Rys. 6-12 , na którym przedstawiono system, do którego dołączono proces wymuszający szeregowanie akcji poprzez synchronizację (komunikację). Dodanie nowego procesu powoduje rozszerzenie wymiaru przestrzeni stanów systemu, stąd akcje α_1 i α_2 nie są wykonywane w tych samych stanach i akcja α' jest obserwowalna dla deterministycznych ścieżek systemu.



Rys. 6-12 Synchronizacja wymuszająca szeregowanie akcji

Brak deterministycznej zgodności zazwyczaj będzie źródłem wykrytych błędów (braku poprawności) lub powodował będzie, że poprawność systemu będzie nierozstrzygalna.

6.5 Podsumowanie

Podstawową różnicą pomiędzy niehomomorficzną i liniową funkcją obserwacji jest rozszerzenie możliwości specyfikacji wymagań. Poprzez odwzorowanie zbiorów akcji modelu weryfikowanego w zbiory równoległych akcji modelu kryterialnego umożliwia ona zapis wymagań dla hierarchicznej dekompozycji pozostawiając konkretnej implementacji (czyli specyfikacji weryfikowanej) określenie *jak* ma zostać przeprowadzona. Podejście takie zgodne jest z zasadami zstępującego cyklu (ang. *top–down*) budowy oprogramowania.

W rozdziale zaproponowano warunki poprawności względnej dla niehomomorficznej funkcji obserwacji. Podobnie jak dla funkcji liniowej przedmiotem analizy jest ciąg rozwiązań modelu weryfikowanego *ML* . Zadaniem funkcji obserwacji jest skonstruowanie obserwacji badanego ciągu, czyli ciągu rozwiązań należących do dziedziny rozwiązań modelu kryterialnego *ML'*.

Model poprawności dla niehomomorficznej funkcji obserwacji został rozszerzony w stosunku do modelu dla funkcji liniowej. Analiza typowych oczekiwań dotyczących hierarchicznej dekompozycji skłoniła do wprowadzenia dodatkowych pojęć charakteryzujących poprawność badanego ciągu rozwiązań. Pojęciami tymi są:

- odwrotna dopuszczalność akcji modelu weryfikowanego,
- lokalna osiągalność obserwowanych akcji.

Sprawdzenie warunków lokalnej osiągalności wymaga analizy historii dojścia do pewnego rozwiązania. Zaproponowaną jej reprezentacją jest macierz lokalnej osiągalności.

W dalszej części pracy zaproponowano formalną konstrukcję umożliwiającą badanie poprawności modelu weryfikowanego dla niehomomorficznej funkcji obserwacji. Podobnie, jak dla liniowej funkcji obserwacji jest nią proces sprzężony opisujący wspólne wykonanie modelu weryfikowanego i kryterialnego. Różnicą w stosunku do modelu liniowego jest uwzględnienie historii dojścia do pewnego stanu modelu weryfikowanego. Reprezentowana jest ona przez macierz lokalnej osiągalności.

Udowodniono, że skonstruowanie procesu sprzężonego, którego

- wszystkie przejścia spełniają warunki poprawności oraz
- wszystkie wyznaczone macierze lokalnej osiągalności spełniają warunek dotyczący istnienia kresu górnego zbiorów ich elementów

jest wystarczające do wnioskowania o poprawności wszystkich przejść w ciągach rozwiązań modelu weryfikowanego.

Zaproponowany algorytm badania poprawności wykorzystuje opisane wyżej konstrukcje. Należy on do metod przeglądu zupełnego. W wyniku jego działania konstruowany jest zbiór wszystkich stanów procesu sprzężonego i badana poprawność przejść dopuszczalnych w każdym stanie. Algorytm ten został szczegółowo opisany w dodatku C.

7. Przykłady

Celem tego rozdziału jest przedstawienie trzech przykładów ilustrujących zastosowanie wprowadzonych wcześniej modeli poprawności opartych na funkcji obserwacji w specyfikacji wymagań. Każdy z opisanych przykładów został poddany analizie poprawności z wykorzystaniem stworzonego oprogramowania implementującego podane w pracy algorytmy.

Pierwsze dwa przykłady to klasyczne problemy: pięciu filozofów i protokołu z bitem potwierdzenia. Trzeci przykład pochodzący od autora to rozwinięty opis systemu sterującego bankomatem przedstawionego w rozdziale 3.

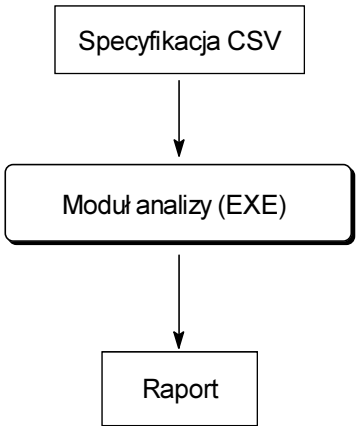
7.1 Opis implementacji

Przed przystąpieniem do omówienia badanych przykładów opiszemy krótko zasady działania oprogramowania użytego do ich analizy. Oprogramowanie to ma prototypowy charakter; głównym celem jego powstania było przetestowanie opracowanych algorytmów – z tego powodu nie zapewnia ono środowiska graficznego do definiowania występujących w problemie poprawności procesów i funkcji obserwacji.

Specyfikacje zagadnień (analizy poprawności i badania spójności) wymagają zdefiniowania zapisanych macierzowo zależności. W tym celu wykorzystywany jest arkusz kalkulacyjny Excel.

Na potrzeby programu zdefiniowano pewną nieformalną składnię języka specyfikacji. Definicje problemów są nieco bogatsze niż wynika z definicji modeli – miedzy innymi obejmują one także tablice symboli identyfikujących poszczególne akcje, co zapewnia większą czytelność specyfikacji i rezultatów jej analizy.

Przygotowany arkusz zapisywany jest w pliku w formacie tekstowym CSV; plik ten stanowi wejście dla modułu analizy (Rys. 7-1).



Rys. 7-1 Wejście i wyjście oprogramowania

W wyniku działania programu sporządzany jest raport z jego przebiegu. Jest on dosyć szczegółowy – opisywane są w nim kolejne przejścia, osiągnięte stany, zbiory dopuszczalnych akcji, itd. W miarę możliwości informacje te podawane są w sposób symboliczny (nazwy akcji a nie ich indeksy). Z tego powodu pliki raportujące przebieg oprogramowania są stosunkowo duże (czasem nawet rzędu kilkudziesięciu MB).

W przypadku napotkania sytuacji błędnej (blokowanie, niepoprawne przejście) podawana jest ścieżka, dla której błąd został zaobserwowany. Pełni ona rolę kontrprzykładu.

Na zakończenie w raporcie podawane jest zestawienie zbiorcze (liczba przejść poprawnych, niepoprawnych, pętli, dywergencji, itd.) oraz wydawany werdykt.

Jak można zauważyć, macierze specyfikujące problemy mogą być stosunkowo duże (dla przedstawionych dalej przykładów miały one rozmiary rzędu 50-70), ale są zazwyczaj bardzo rzadkie. Podobnie, wektory opisujące rozwiązania i stany zawierają często wiele elementów zerowych.

Implementacja opisanych wcześniej algorytmów wykorzystuje wyłącznie rzadką reprezentację danych, z tego powodu zużycie pamięci zostało znacznie ograniczone. Standardowe struktury używane w implementacji algorytmów (rzadkie wektory i macierze, zbiory) zaimplementowano w oparciu o zestaw kilku szablonów C++.

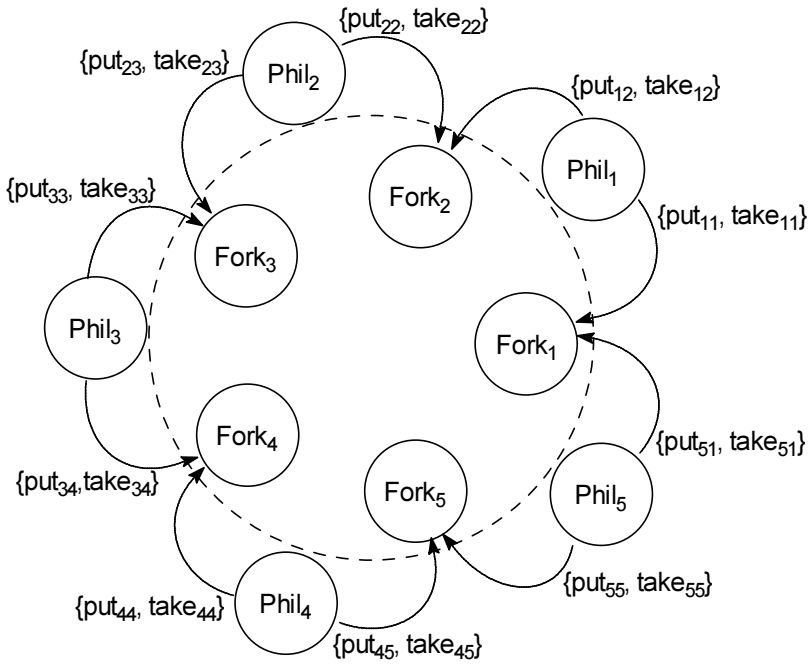
Struktury danych są pod tym względem dość efektywne – ale rzadka reprezentacja danych pozwala jedynie na wielomianową redukcję zużycia zasobów. Największe testowe specyfikacje, które udało się w pełni przeanalizować na średniej klasy sprzęcie (Pentium 200 MHz, 48 MB RAM) generowały około 50 000 przejść. Trudno jest podać wydajność obliczeniową (liczoną np.: w przejściach na sekundę) ponieważ równocześnie za pomocą stosunkowo wolnych funkcji formatowanego wyjścia generowane były pliki o rozmiarach osiągających 80 MB. Analiza problemów tej wielkości trwała zazwyczaj kilka minut.

W przypadku dużych problemów bardzo znaczne przyspieszenie dało się zaobserwować po zastąpieniu nieuporządkowanej kolekcji reprezentującej zbiór osiągniętych stanów (procesu sekwencyjnego lub sprzężonego) przez tablicę indeksowaną funkcją mieszającą (ang. *hash table*).

7.2 Problem pięciu filozofów

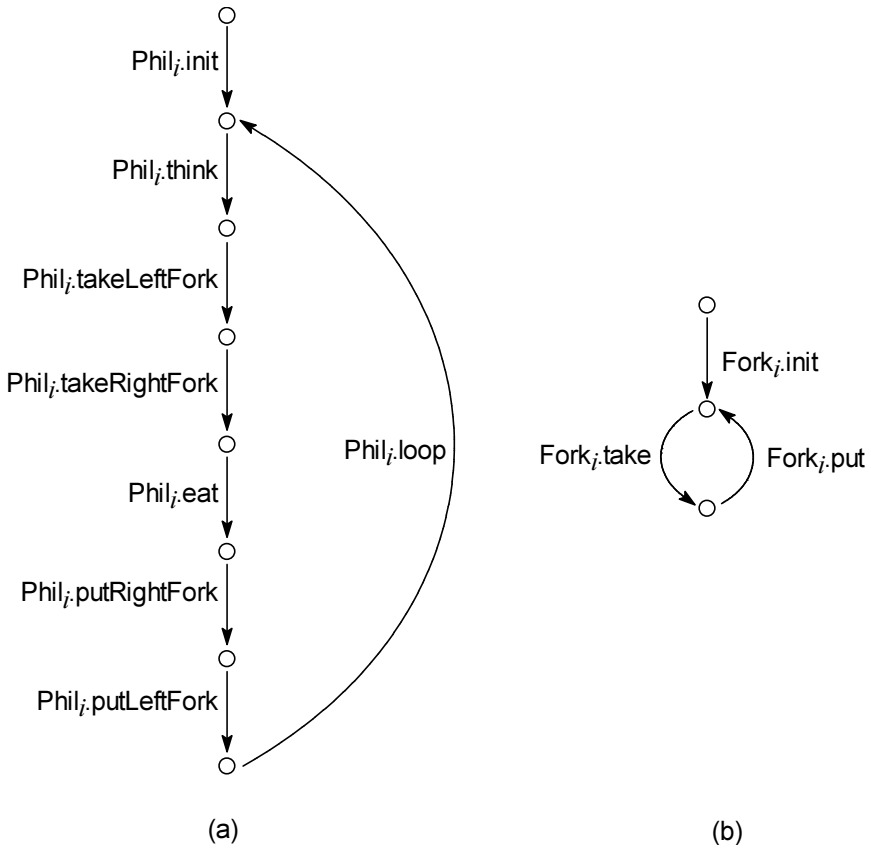
Opis problemu pięciu filozofów można znaleźć w różnych pracach [Dijkstra 68; Szmuc 89b; Hoare 85;]. Poniższy opis najbliższy jest ostatniej pracy. Pięciu filozofów zgromadzono przy okrągłym stole, na którym stoi miska spaghetti. Każdy z filozofów ma stałe miejsce przy stole. Pomiędzy filozofami położono pięć widelców (Rys. 7-2). Główną czynnością filozofa jest myślenie, przerywane od czasu do czasu jedzeniem. Aby przystąpić do jedzenia, filozof musi podnieść dwa widelce. Zakładamy, że robi to w ustalonej kolejności: wpierw sięga po lewy, a następnie po prawy widelec. Po zaspokojeniu głodu filozofowie odkładają widelce w odwrotnej kolejności.

Filozofowie dzielą widelce z sąsiadami. Jeśli któryś z widelców jest używany, filozof, który chciałby przystąpić do jedzenia musi czekać (być może z jednym widelcem w ręce), aż niezbędny widelec zostanie zwolniony.



Rys. 7-2 Problem pięciu filozofów

Na Rys. 7-3 przedstawiono procesy opisujące poszczególnych filozofów (*Phil*$_i$) i widelce (*Fork*$_i$). Proces *Fork*$_i$ może być traktowany jako rodzaj semafora binarnego chroniącego dzielony zasób.



Rys. 7-3 Procesy opisujące:(a) zachowanie filozofa (b) widelec

Zdefiniujmy funkcję komunikacji ϕ odpowiedzialną za sprzężenia pomiędzy procesami (Patrz podrozdział 4.5.1) . Dla *i*-tego filozofa, gdzie $i = 1,\ldots,5$ ma ona postać:

$\phi(Phil_i.\text{takeLeftFork}) = Fork_i.\text{take}$

$\phi(Phil_i.\text{takeRightFork}) = Fork_j.\text{take}$, gdzie $j = \mathbf{mod}(i+1, 5)$

$\phi(Phil_i.\text{putLeftFork}) = Fork_i.\text{put}$

$\phi(Phil_i.\text{putRightFork}) = Fork_j.\text{put}$, gdzie $j = \mathbf{mod}(i+1, 5)$

7.2.1 Proces sekwencyjny systemu

Opisany zbiór procesów oraz funkcję komunikacji zamodelowano w postaci zbioru ograniczeń nierównościowych i równościowych składających się na model liniowy ML_1. Rozwiązaniem początkowym był wektor przejść początkowych spełniający $x(Phil_i.\text{init}) = 1$ oraz $x(Fork_i.\text{init}) = 1$.

Raport z przebiegu oprogramowania do generacji procesu sekwencyjnego ujęto w poniższej tabeli.

Liczba wykonanych przejść	10795
Liczba stanów osiągalnych	2622
Liczba ciągów rozwiązań prowadzących do stanów pętlących	3022
Liczba ciągów rozwiązań prowadzących do stanów blokujących	5
Liczba stanów końcowych	0

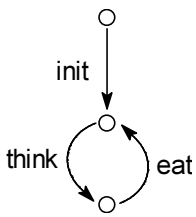
Równocześnie dla zbioru ciągów rozwiązań wolnych od blokowania wszystkie przejścia zostały wykonane, czyli stwierdzono potencjalne istnieje procesu całkowicie poprawnego względem tej specyfikacji.

Blokowanie związane było ze znanym z literatury opisem ścieżki zachowania systemu, która prowadzi do stanu, gdy każdy z filozofów ujął po jednym widelcu blokując w ten sposób wszystkich sąsiadów.

W [Hoare 85] podano także rozwiązanie pozwalające na wyeliminowanie przypadku blokowania. Polega ono na ograniczeniu do czterech liczby filozofów, którzy mogą równocześnie używać widelców. Po dodaniu odpowiedniego ograniczenia w modelu ML_2 blokowanie zostało usunięte.

7.2.2 Weryfikacja poprawności systemu pięciu filozofów

Poprawiony model systemu ML_2 został poddany weryfikacji poprawności względem bardzo prostego procesu kryterialnego przedstawionego na Rys. 7-4. Proces ten opisuje wymagane zachowanie systemu z punktu widzenia wybranego (tu pierwszego) filozofa. Jego głównym pragnieniem jest, aby móc na przemian myśleć i jeść, bez względu na to, co robią inni współbiesiadnicy.



Rys. 7-4 Proces kryterialny

Funkcja obserwacji rzutująca zachowanie systemu na proces kryterialny jest w tym przypadku prostą funkcją homomorficzną zdefiniowaną jako:

$init = \mathrm{Phil}_1.init$

$eat = \mathrm{Phil}_1.eat$

$think = \mathrm{Phil}_1.think$

Wynik weryfikacji ujęto w poniższej tabeli

Liczba wykonanych przejść	10640
Liczba niepoprawnych przejść	0
Liczba stanów procesu sprzężonego	2592
Liczba ciągów rozwiązań prowadzących do stanów pętlących	2962
Liczba ciągów rozwiązań zawierających dywergencje	549
Liczba ciągów rozwiązań prowadzących do stanów blokujących	0
Liczba stanów końcowych	0

Zaobserwowano wszystkie przejścia w procesie kryterialnym; jednakże ze względu na obecność dywergencji specyfikacja jest częściowo poprawna. Ten fakt wymaga komentarza. Wybrany filozof postawiony w roli obserwatora stwierdza, że nawet jeżeli czas jedzenia poszczególnych filozofów jest ograniczony, nic nie wskazuje, aby system przejawiał jakąkolwiek tendencję prowadzącą go do uzyskania dostępu do obu widelców. W zależności od tego, czy on, jego sąsiedzi (i ich sąsiedzi) są szybsi lub bardziej zdecydowani może w skończonym czasie zebrać oba widelce lub nigdy tego nie osiągnąć będąc tym samym skazany na głodowanie.

Istnienie dywergencji powoduje, że z punktu widzenia obserwatora system nie realizuje zasady sprawiedliwości (ang. *fairness*). Mogą w nim wystąpić nieskończone ścieżki wykonania, w których kryterialna akcja *eat* jest stale dopuszczalna i oczekiwana, ale nie zostanie nigdy wykonana.

7.3 Protokół z bitem potwierdzenia

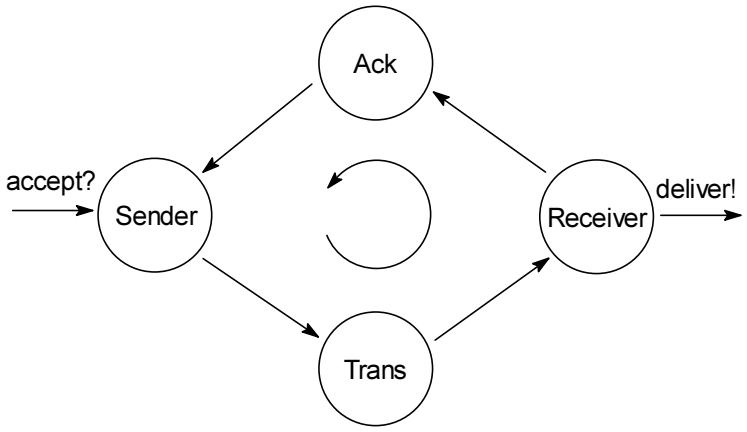
Protokół z bitem potwierdzenia (ang. *alternating bit protocol*) jest znanym protokołem przesyłania informacji [Milner 89; Szmuc 89b]. Przedstawiona tu wersja wzorowana jest na opisie zawartym w pracy Milnera.

Stronami w protokole są dwa procesy *Sender* pełniący rolę nadawcy oraz *Receiver* pełniący rolę odbiorcy. Połączone są one przez dwa asynchroniczne kanały komunikacyjne *Trans* i *Ack* (reprezentujące na przykład sieć komputerową).

Idea działania protokołu przedstawiona jest na Rys. 7-5. Wymiana informacji odbywa się według określonego scenariusza. Nadawca po otrzymaniu zlecenia z komunikatem do wysłania (*accept*) kieruje komunikat do kanału *Trans* dodając znacznik, którym jest uzgodniona wartość bitu potwierdzenia (np.: 0). Odbiorca odbiera komunikat i jeśli przesłana wartość bitu jest zgodna z oczekiwaniami, wówczas dostarcza komunikat na

zewnątrz (*deliver*) oraz wysyła bit potwierdzenia do kanału *Ack*. Nadawca odbiera bit potwierdzenia; jeśli odpowiada on ostatnio wysłanemu komunikatowi, wówczas jest gotowy do przesyłania następnego komunikatu, któremu zostanie przypisana zmieniona wartość bitu.

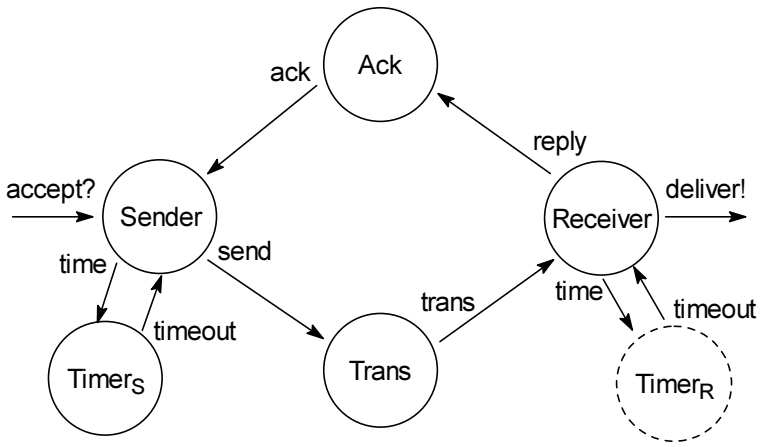
Bity potwierdzenia stosowane są na wypadek, gdyby media komunikacji okazały się zawodne i prowadziły do utraty lub duplikacji przesyłanych komunikatów.



Rys. 7-5 System transmisji danych

W [Milner 89] rozważano system, którego ogólny diagram przedstawiono na Rys. 7-6. System został wzbogacony o dodanie dwóch liczników czasu ($Timer_S$ i $Timer_R$), które służą procesom nadawczym i odbiorczym do stwierdzenia, czy odpowiedź nadeszła w oczekiwanym czasie. W przypadku stwierdzenia jej braku należy powtórzyć transmisję. Założono, że kanały komunikacji reprezentowane są przez nieskończone bufory, w których możliwa jest utrata lub duplikacja przesłanej wiadomości.

Dla takiego systemu przedstawiono formalny dowód, że zachowuje się on jak bufor o jednostkowej pojemności, czyli, że będzie na przemian wykonywał operacje *accept* i *deliver* .



Rys. 7-6 Komponenty systemu (procesy)

7.3.1 Modele procesów składowych

Modele procesów składowych w różnych wersjach przedstawiono na kolejnych rysunkach. Dla uczynienia ich bardziej czytelnymi przyjęto konwencję, zgodnie z którą wybrane węzły oznaczane są etykietami. Powtórzenie etykiet przy różnych węzłach oznacza, że reprezentują one ten sam stan.

Proces nadawczy *Sender*

W algebrze CCS proces ten zdefiniowany jest jako:

$$Accept(b) \equiv accept.\ Send(b)$$

$$Send(b) \equiv \overline{send}_b\,.\,\overline{time}\,.Sending(b)$$

$$Sending(b) \equiv timeout.Send(b) + ack_b.timeout.Accept(1-b) + ack_{(1-b)}.\ Sending(b),$$

gdzie $b \in \{0, 1\}$.

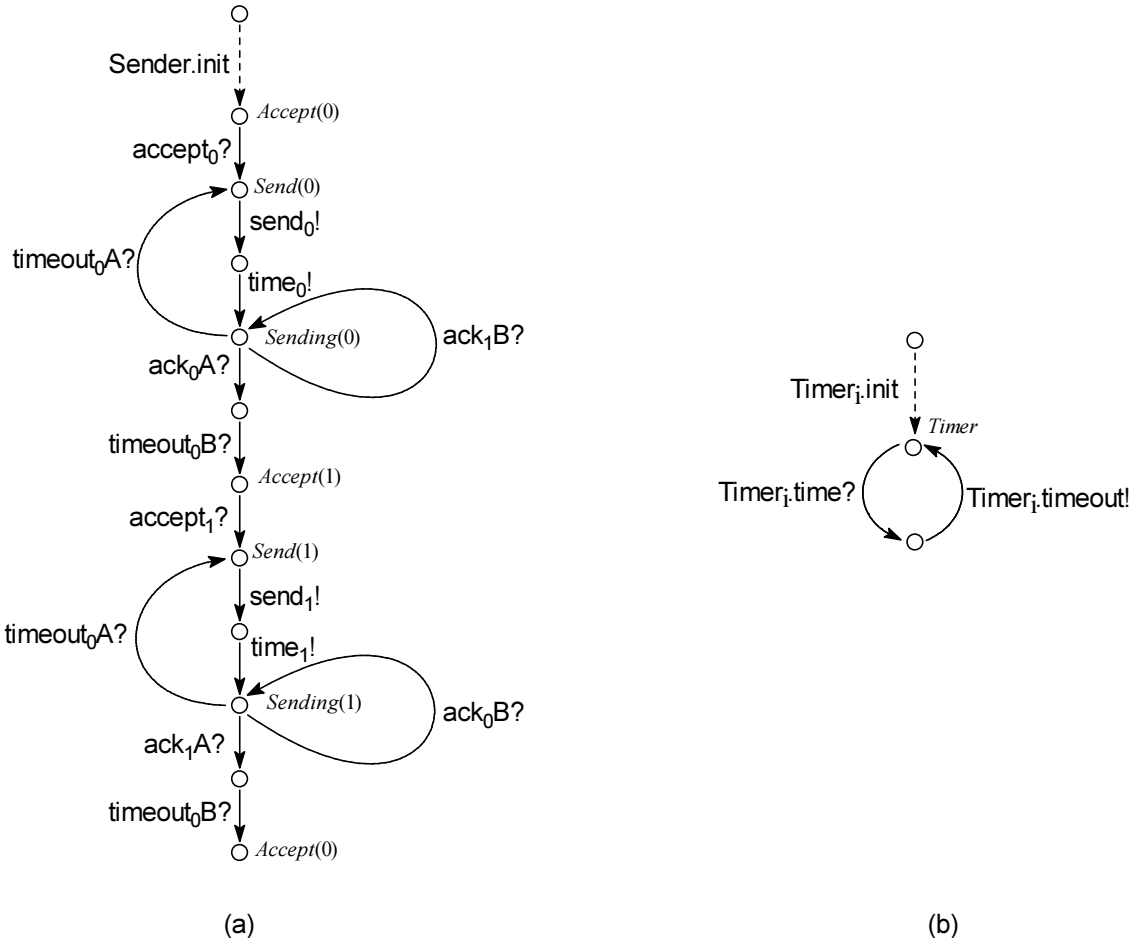
Proces ten przedstawiono na Rys. 7-7 a. Warto zwrócić uwagę na to, że niejawną składową stanu procesu jest zmienna binarna *b*. Jej wartością są indeksowane rozróżnialne przejścia i stany, stąd reprezentacja procesu jest znacznie bardziej obszerna niż definicja CCS. Występujące w CCS symbole etykiet i koetykiet zostały zastąpione na rysunkach symbolami „!" oraz „?".

Załóżmy, że zmienna binarna *b* ma wartość 0. Proces *Sender* pracuje według następującego scenariusza:

1. czeka na zlecenie z komunikatem do wysłania, po jego otrzymaniu wysyła komunikat oznaczony wartością *b* do kanału *Trans* i ustawia licznik czasu;
2. jeśli otrzyma sygnał *timeout*, wówczas powtarza wysyłkę komunikatu, natomiast jeśli otrzyma wcześniej potwierdzenie z wartością *b*, wówczas przechodzi w stan oczekiwania na następny komunikat ustawiając $b := b - 1$;
3. jeśli inny otrzyma bit potwierdzenia, niż oczekiwano (czyli b–1) wówczas powtórnie wysyła komunikat.

Na Rys. 7-7 b przedstawiono proces $Timer_i$ ($i \in \{S, R\}$) zdefiniowany uprzednio jako:

$$Timer_i \equiv time.\ \overline{timeout}\,.Timer_i$$



Rys. 7-7 Procesy: (a) Sender *(b)* Timer_i *(i = S, R)*

Proces odbiorczy *Receiver*

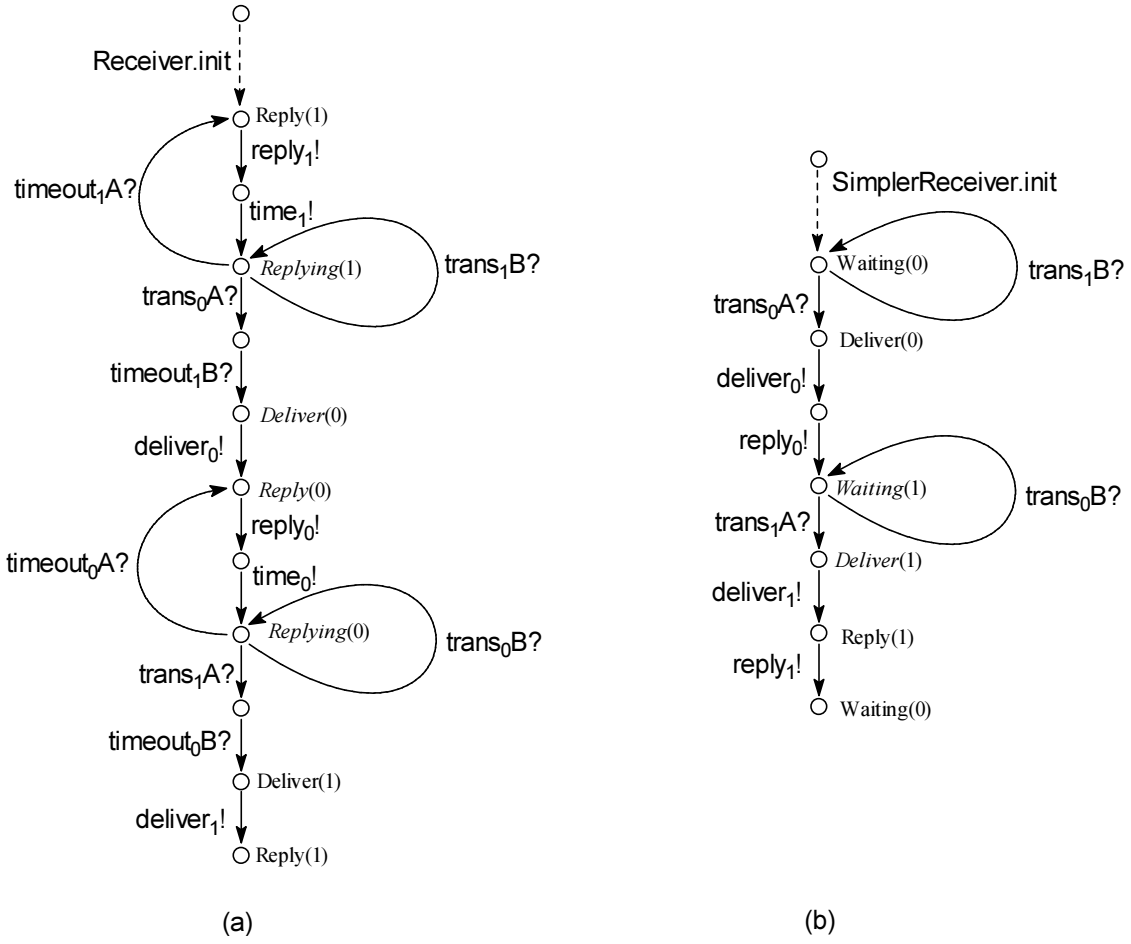
Rys. 7-8 pokazuje dwie wersje procesu *Receiver* . Pierwsza z nich (Rys. 7-8 a) jest zgodna z definicją pochodzącą z pracy Milnera zapisaną jako:

$$Reply(b) \equiv \overline{reply}_b \,.\, \overline{time} \,.Sending(b)$$

$$Replying(b) \equiv timeout.Reply(b) + trans_{(1-b)}.timeout.Deliver(1-b) + trans_b.Replying(b)$$

$$Deliver(b) \equiv \overline{deliver} \,.Reply(b) \text{ , gdzie } b \in \{0, 1\}.$$

Druga (*SimpleReceiver*) jest wersją uproszczoną. Proces ten nie korzysta z licznika czasu. Rola licznika czasu po stronie odbiorcy jest raczej dyskusyjna, ponieważ nie ma żadnego praktycznego uzasadnienia. System traktowany jako całość powinien reagować na dostarczony z zewnątrz komunikat (operacja *accept*) i dopiero wtedy przejawiać aktywność. Jak wynika z przeprowadzonych testów, w wersji z licznikiem czasu *TimerR* system będzie generował ruch w sieci nawet jeśli operacja *accept* nie została wykonana, zapełniając w ten sposób nieograniczenie bufor *Trans*.



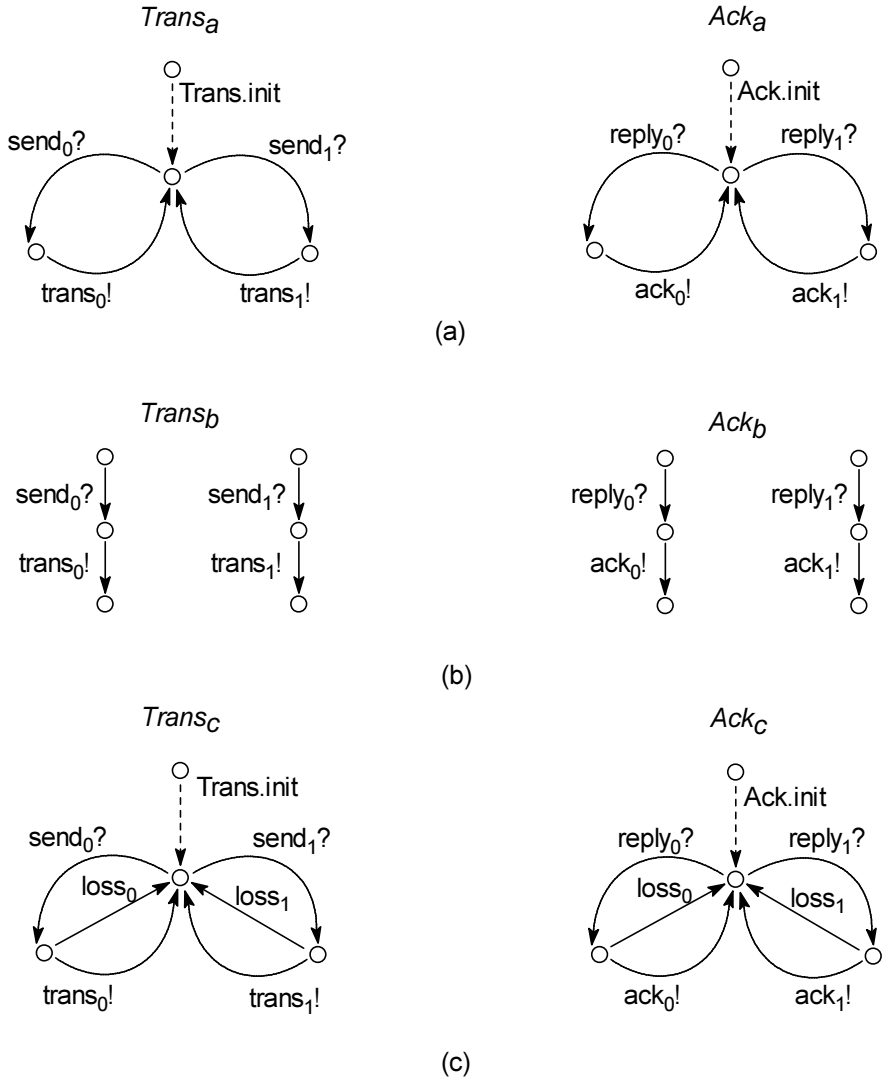
Rys. 7-8 Dwie wersje procesu Receiver *(a) proces komunikujący się z licznikiem czasu (b) wersja uproszczona*

Kanały komunikacji *Trans* **i** *Ack*

W opisie Milnera zakładano, że bufory mają nieograniczoną pojemność i zachowują się jak kolejki FIFO. Procesy opisujące bufory tego typu nie są możliwe do zamodelowania w postaci modelu liniowego. Przeanalizowano więc trzy wersje procesów pełniących rolę kanałów komunikacyjnych przedstawione na Rys. 7-9.

Rys. 7-9 a pokazuje model buforów o pojemności jeden. Na Rys. 7-9 b przedstawiono kanały o nieskończonej pojemności, ale nie zachowujące się jak kolejki FIFO. Model podobny jest do kolorowych sieci Petriego, gdzie jedno miejsce jest zdolne do przechowywania żetonów z palety {0,1}. Miejsce takie zostało zamodelowane jako dwa węzły. Oczywiście w takim przypadku możliwa jest jedynie częściowa weryfikacja, ponieważ zbiór stanów badanego procesu jest nieskończony.

Rys. 7-9 c prezentuje bufory o pojemności jeden ale z możliwą utratą zawartości (przejścia $loss_i$).



Rys. 7-9 Różne wersje procesów Trans i Ack

Przebadane wersje systemu

Podczas testów przebadano poprawność czterech wersji systemu, których komponenty zostały zestawione w poniższej tabeli.

Wersja	Sender i TimerS	Trans i Ack	Receiver	TimerR
I (ML_I)	Rys. 7-7 a, b	Rys. 7-9 a	Rys. 7-8 a	Rys. 7-7 b
II (ML_{II})	Rys. 7-7 a, b	Rys. 7-9 a	Rys. 7-8 b	brak
III (ML_{III})	Rys. 7-7 a, b	Rys. 7-9 b	Rys. 7-8 b	brak
IV (ML_{IV})	Rys. 7-7 a, b	Rys. 7-9 c	Rys. 7-8 b	brak

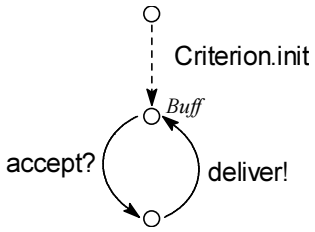
Poprawność każdego z systemów była weryfikowana dla prostego procesu kryterialnego reprezentującego bufor o pojemności jeden przedstawionego na Rys. 7-10. W każdym z przypadków używana była ta sama funkcja obserwacji π postaci:

$$Criterion.init = \Pi\circ (X.init) ,$$

$$accept = accept_0 + accept_1$$

$$deliver = deliver_0 + deliver_1$$

gdzie $\Pi\circ$ oznacza uogólniony operator „$\circ$”; X dowolny proces.



Rys. 7-10 Proces kryterialny

Rozwiązaniem początkowym weryfikowanego modelu ML_i był wektor spełniający $x_0(X.init) = 1$, gdzie X jest symbolem dowolnego procesu, natomiast rozwiązanie początkowe dla modelu kryterialnego ML' spełniało $x_0'(Criterion.init) = 1$.

7.3.2 Wynik weryfikacji dla wersji I

Pierwsza testowana wersja składała się z procesów *Sender*, $Timer_S$, *Receiver*, $Timer_R$, $Trans_a$ oraz Ack_a. Kanały przesyłowe modelowane były jako bufory o pojemności 1.

Wynikiem przeprowadzonych testów była stwierdzenie potencjalnej poprawności systemu; nie zaobserwowano żadnych nieprawidłowych przejść, jednakże specyfikacja nie była poprawna, ponieważ zaobserwowanie 16 ciągów rozwiązań prowadzących bezpośrednio do stanów blokujących. Najkrótszy spośród nich przedstawiony jest poniżej.

1. ($Sender.accept_0$)
1. ($Trans.send_0$, $Sender.send_0!$)
1. ($TimerS.time?$, $Sender.time_0!$)
1. ($TimerS.timeout!$, $Sender.timeout_0a$)
1. ($Ack.reply_1?$, $Receiver.reply_1!$)
1. ($TimerR.time?$, $Receiver.time_1!$)
1. ($TimerR.timeout!$, $Receiver.timeout_1b$)

Problem stwarza sekwencja przejść elementarnych (5, 6, 7), która prowadzi do wysyłania zbędnego potwierdzenia i zapełnienia bufora. Co więcej, analogiczne zjawisko mogłoby powstać dla bufora o dowolnej (ale ograniczonej) pojemności, ponieważ po wykonaniu ciągu przejść (5, 6, 7) przejście (5) będzie dopuszczalne dopóki pojemność bufora będzie na to pozwalała.

7.3.3 Wynik weryfikacji dla wersji II

W drugiej z testowanych wersji system składał się z procesów *Sender*, $Timer_S$, *SimpleReceiver*, $Trans_a$ oraz Ack_a. Różnicą w stosunku do wersji poprzedniej było usunięcie licznika czasu po stronie odbiorczej i przez to znaczne uproszczenie procesu (Rys. 7-8 b).

Raport z przebiegu weryfikacji ujęto w poniższej tabeli.

Liczba wykonanych przejść	148
Liczba niepoprawnych przejść	0
Liczba stanów procesu sprzężonego	84
Liczba ciągów rozwiązań prowadzących do stanów pętlących	18
Liczba ciągów rozwiązań zawierających dywergencje	12
Liczba ciągów rozwiązań prowadzących do stanów blokujących	0
Liczba stanów końcowych	0

Wynikiem weryfikacji jest częściowa poprawność specyfikacji związana z zaobserwowaniem dywergencji. Ich źródłem jest pętla związana z sygnałem *timeout* pochodzącym z od licznika czasu. Omówmy przykład krótkiego ciągu rozwiązań zawierającego dywergencję:

1. po otrzymaniu pierwszego komunikatu proces *SimpleReceiver* wysyła potwierdzenie do kanału *Ack* (sygnał *reply*) i przechodzi w stan *Waiting*(1) (Rys. 7-8 b);
2. proces *Trans* nie odbiera sygnału potwierdzenia lecz sygnał *timeout* i wysyła powtórnie komunikat oznaczony bitem 0
3. proces *SimpleReceiver* otrzymuje komunikat oznaczony bitem 0 lecz ignoruje go
4. system wraca do kroku 2

Sytuacja taka jest możliwa dla nietrafnie dobranych parametrów licznika czasu. W tym przypadku stwierdzenie dywergencji może być traktowane jako ostrzeżenie, ponieważ jest raczej mało prawdopodobne. Zjawisko takie jednak mogłoby mieć wpływ na zachowanie rzeczywistego systemu, gdyby z bliżej nieznanych przyczyn linia *Trans* miała wyższy priorytet niż linia *Ack* i parametry licznika czasu były niewłaściwie dobrane w stosunku do czasów transmisji.

7.3.4 Wynik weryfikacji dla wersji III

Trzeci z testowanych modeli systemu różnił się od poprzedniego zastosowaniem kanałów przesyłowych $Trans_b$ oraz Ack_b . Modelują one bufory o nieograniczonej pojemności nie zapewniające porządku FIFO odbieranych komunikatów. Model ten może być właściwy dla sieci, które kierują pakiety różnymi drogami, nie zapewniając w ten sposób porządku otrzymywanych wiadomości.

W tym przypadku w systemie zaobserwowano wiele (ok. 700) przypadków niepoprawnych przejść. Ciągi przejść prowadzące do sytuacji niepoprawnych zawierały minimum kilkadziesiąt elementów, dlatego trudno je tu przedstawiać. Źródła błędów były wynikiem realizacji opisanego poniżej scenariusza:

1. Proces *Sender* wysyłał kilka komunikatów oznaczonych bitem 0 (ponieważ nie otrzymywał na czas potwierdzenia).
2. Proces *SimpleReceiver* odbierał pierwszy z nich i wysyłał potwierdzenie.
3. Proces *Sender* wysyłał kilka komunikatów oznaczonych bitem 1.
4. Proces *SimpleReceiver* ignorował część z komunikatów oznaczonych bitem 0 ale nie wszystkie i odbierał komunikat oznaczony bitem 1, następnie wysyłał potwierdzenie.
5. Następnie *SimpleReceiver* przechodził w stan *Waiting*(0) i wówczas odbierał jedną z poprzednio wysłanych wiadomości oznaczonych bitem 0, która „nie dotarła na czas" i dostarczał ją na zewnątrz.

7.3.5 Wynik weryfikacji dla wersji IV

Wersja ta różniła się od dwóch poprzednich użyciem kanałów komunikacji z możliwością straty informacji ($Trans_c$ oraz Ack_c na Rys. 7-9 c).

Wynikiem przebiegu programu było stwierdzenie częściowej poprawności powodowanej wyłącznie obecnością dywergencji, co ujmuje poniższa tabela.

Liczba wykonanych przejść	282
Liczba niepoprawnych przejść	0
Liczba stanów procesu sprzężonego	116
Liczba ciągów rozwiązań prowadzących do stanów pętlących	71
Liczba ciągów rozwiązań zawierających dywergencje	63
Liczba ciągów rozwiązań prowadzących do stanów blokujących	0
Liczba stanów końcowych	0

Oznacza to, że system zachowuje się poprawnie także dla przypadku, kiedy linie przesyłowe są zawodne. Jak wydaje się, ten ostatni model najwierniej oddaje praktycznie stosowane rozwiązania.

7.4 Bankomat

Trzeci z omawianych przykładów opisuje system sterujący bankomatem, którego ideę przedstawiono w przykładach w rozdziale 3. W stosunku do przedstawionej tam wersji wprowadzono kilka rozszerzeń, które bardziej zbliżają model do rzeczywistości. Przede wszystkim dodano operacje dotyczące karty: *insertCard* (klient wkłada kartę do szczeliny czytnika), *ejectCard* (bankomat wysuwa kartę) *eatCard* (bankomat zatrzymuje kartę w zasobniku, ponieważ klient nie zna kodu PIN). Dodatkowo założono, że klient może anulować daną operację.

Weryfikacja podzielona jest na dwa etapy przebiegające pomiędzy trzema warstwami opisu systemu. Pierwsza z nich jest najbardziej abstrakcyjna, druga bardziej szczegółowa (i spójna) stanowi specyfikację, która podlega weryfikacji w pierwszym etapie.

Do opisu ogólnego trzeciej warstwy posłużono się diagramami DFD; wynikiem analizy wymagań jest jak zwykle opis procesów w postaci maszyn skończenie-stanowych.

Przedstawiony przykład jest najbliższy tworzonym w rzeczywistości systemom. Pokazane na nim zostaną możliwości specyfikacji wymagań w typowych przykładach dekompozycji

funkcjonalnej, gdzie większość operacji rozdzielana jest pomiędzy drzewa funkcji, a ich wynik najczęściej utożsamiony jest ze zwróconą wartością.

7.4.1 Pierwszy poziom specyfikacji systemu

Na Rys. 7-11 pokazano kilka prostych rozłącznych procesów opisujących wymagania stawiane systemowi. Opiszmy poszczególne z nich:

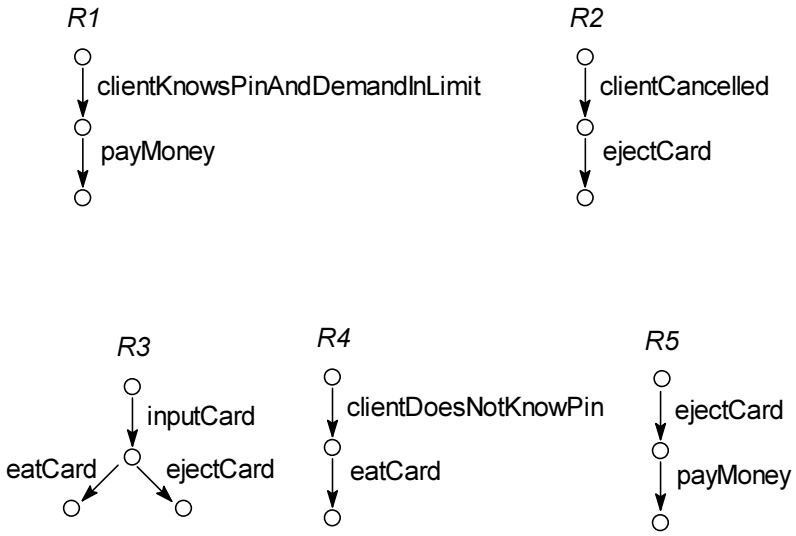
R1 Jeśli klient zna kod PIN i żądana kwota mieści się w limicie, bankomat wypłaci pieniądze.

R2 Jeśli klient odwoła operację, bankomat zwróci kartę.

R3 Jeśli karta zostanie wprowadzona, to albo zostanie wysunięta albo zatrzymana w zasobniku.

R4 Jeśli klient nie zna kodu PIN, bankomat zatrzyma kartę.

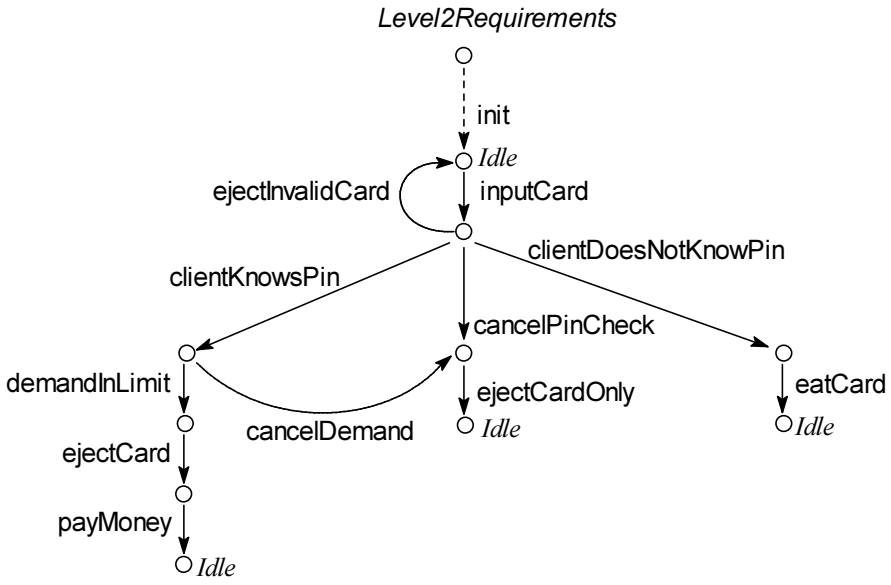
R5 Bankomat wpierw wysunie kartę, a potem wypłaci pieniądze, ponieważ zdarza się, że roztargnieni klienci biorą pieniądze i zapominają o karcie.



Rys. 7-11 Wymagania pierwszego poziomu R1-R5

7.4.2 Drugi poziom specyfikacji systemu

Drugi poziom specyfikacji systemu zaprezentowano na Rys. 7-12. Podobnie, jak w poprzednich przykładach przyjęto konwencję etykietowania węzłów dla poprawienia czytelności rysunku. Proces ten łączy kilka scenariuszy zachowania systemu. Poszczególne łuki są etykietowane symbolami operacji bardziej szczegółowych; na tym poziomie opisu przypominają one wciąż jeszcze nazwy predykatów (typu *clientKnowsPin*, *clientDoesNotKnowPin*). Nowymi operacjami są: *ejectInvalidCard* (wysunięcie nieczytelnej karty, np.: przeznaczonej dla innego systemu) oraz operacje anulujące usługę (*cancelPinCheck* oraz *cancelDemand*).



Rys. 7-12 Drugi poziom specyfikacji

Specyfikacja funkcji obserwacji dla tak opisanych specyfikacji ma postać:

R1.clientKnowsPinAndDemandInLimit = clientKnowsPin ° DemandInLimit

R1.payMoney = payMoney

R2.clentCancelled = cancelPinCheck ⊕cancelDemand

R2.ejectCard = ejectCardOnly

R3.inputCard = inputCard

R3.eatCard = eatCard

R3.ejectCard = ejectCard ⊕ ejectCardOnly ⊕ ejectInvalidCard

R4. clientDoesNotKnowPin = clientDoesNotKnowPin

R4. eatCard = eatCard

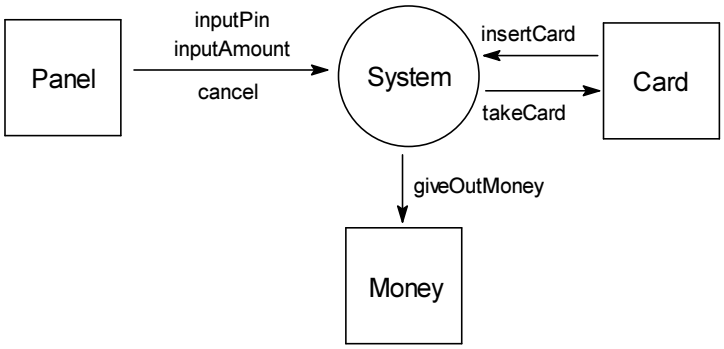
R5.ejectCard = ejectCard

R4.payMoney = payMoney

System jest na tyle prosty, że jak łatwo przypuszczać weryfikacja dała w pełni poprawne rezultaty. Przedstawiono go przede wszystkim, aby pokazać zależności, które mogą być specyfikowane przez funkcję obserwacji

7.4.3 Trzeci poziom specyfikacji systemu

Kolejny poziom specyfikacji przedstawiony zostanie w ujęciu typowym dla analizy strukturalnej. Na Rys. 7-13 przedstawiono diagram kontekstowy z zaznaczeniem sygnałów przesyłanych pomiędzy systemem a otoczeniem. Obiekty terminalne systemu to *Panel* (czyli klawiatura) skąd mogą pochodzić sygnały *inputPin* (wprowadzenie kodu), *inputAmount* (wprowadzenie żądanej kwoty), *cancel* (anulacja usługi). Obiekt *Card* symbolizujący mechanizm czytnika karty z operacjami *insertCard* (wykrycie wsuniecia karty) i *takeCard* (wysunięcie karty) oraz obiekt *Money* (mechanizm wydający pieniądze).



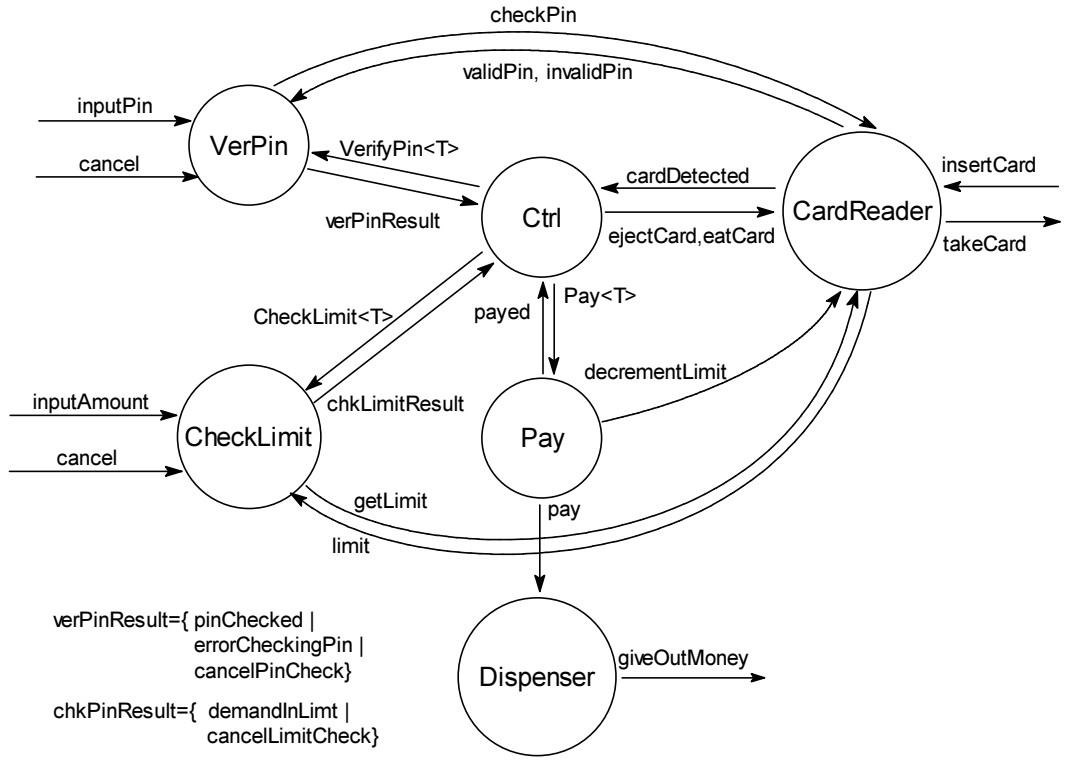
Rys. 7-13 Diagram kontekstowy systemu

Na Rys. 7-14 przedstawiono wynik dekompozycji systemu na podprocesy (z pominięciem jednego etapu). W systemie wyróżnione są trzy logiczne grupy procesów (które mogą zostać zaimplementowane jako trzy zadania): grupa złożona z procesów *Ctrl*, *VerPin*, *CheckLimit* i *Pay*, oraz grupy zawierające po jednym procesie: *CardReader* oraz *Dispenser*.

Zadaniem procesu *Ctrl* jest sterowanie przebiegiem aplikacji, proces *VerPin* jest wydzielonym procesem odpowiedzialnym za sprawdzanie kodu (funkcją), podobnie proces *CheckLimit* jest odpowiedzialny za sprawdzanie, czy żądana kwota mieście się w limicie wypłat, proces *Pay* obsługuje operacje związane z wypłatą pieniędzy. Wszystkie te procesy można traktować jako wynik dekompozycji procesu bardziej ogólnego składającego się na odrębny moduł. Poszczególne procesy podrzędne są wołane za pośrednictwem sygnałów <T> (*trigger*) podobnie, jak w metodzie SART [WM 85; Perez 90].

Proces *CardReader* reprezentuje moduł oprogramowania obsługujący czytnik kart. Jego zadaniem jest komunikacja z mechanizmem czytnika, a także odczyt i zapis karty. Dodatkowo powierzono mu zadanie sprawdzenia kodu, ponieważ uniezależnia to od przyszłych zmian sposobu kodowania.

Proces *Dispenser* obsługuje od strony oprogramowania mechanizm realizujący wypłatę pieniędzy. Jego jedynym zadaniem jest wydanie pieniędzy w odpowiedzi na sygnał *pay*.



Rys. 7-14 Dekompozycja systemu na procesy składowe

Scenariusz działania

Założony scenariusz działania systemu jest następujący:

1) Po wprowadzeniu karty (*insertCard*) i jej prawidłowym odczytaniu proces *CardReader* wysyła sygnał *cardDetected* .

2) Następnie proces *Ctrl* woła procedurę *VerPin*; dalej w zależności od zwróconej wartości :

   - dla *errorCheckingPin* wysyła sygnał *eatCard*
   - dla *cancelPinCheck* wysyła sygnał *ejectCard*
   - dla *pinChecked* woła procedurę *CheckLimit* (sygnał *CheckLimit <T>*)

3) Jeżeli proces *CheckLimit* zwróci wartość *cancelLimitCheck*, wówczas wysyła sygnał *ejectCard* .

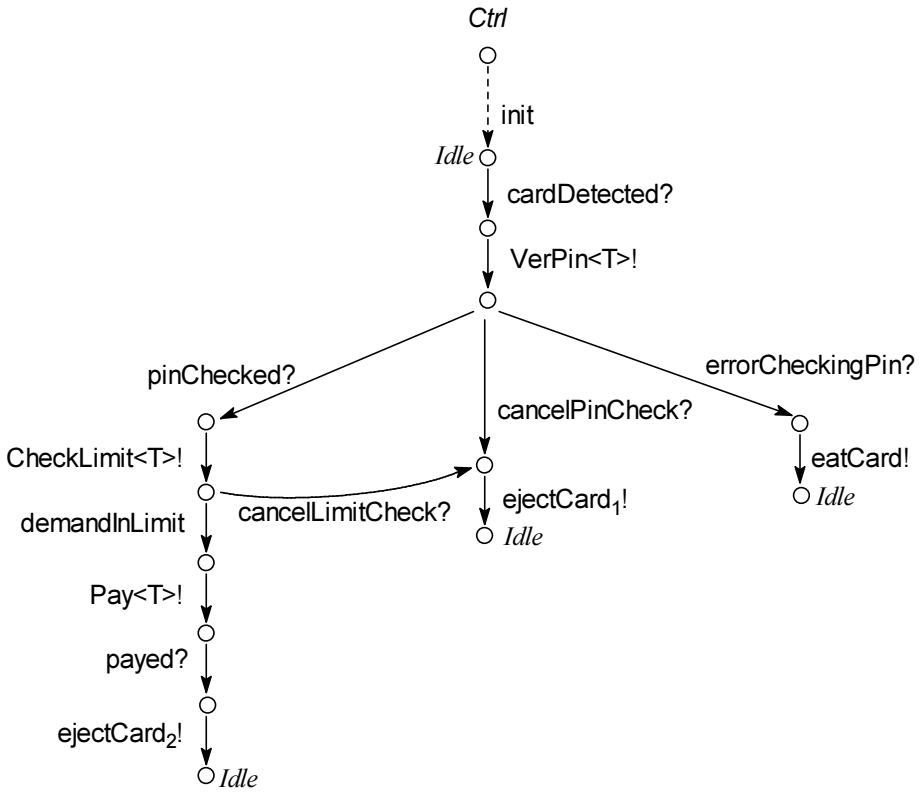
4) W przeciwnym wypadku *Ctrl* woła proces *Pay*, który z kolei wysyła sygnał *decrementLimit* (zmniejsz kwotę limitu) oraz sygnał *pay*, w odpowiedzi na który proces *Dispenser* wypłaca pieniądze (*giveOutMoney*)

5) Następnie proces *Pay* wysyła sygnał *payed*, w odpowiedzi, na który *Ctrl* wysyła sygnał *ejectCard* do procesu *CardReader*.

Powyższy scenariusz działania może również stanowić specyfikację kryterialną (i w typowych przypadkach właśnie takie będzie jej pochodzenie). Ważną zaletą tak postawionego problemu weryfikacji jest możliwość stworzenia homomorficznej funkcji obserwacji, która będzie odwzorowywała w siebie te same dziedziny znaczeń.

Niehomomorficzna funkcja obserwacji pozwoli jednak na sprawdzenie, czy rzeczywiście można postawić znak równoważności pomiędzy symbolami *clientDoesNotKnowPin* i *errorCheckingPin*, czy *payMoney* i *payed*, wchodząc głębiej w samo jądro operacji.

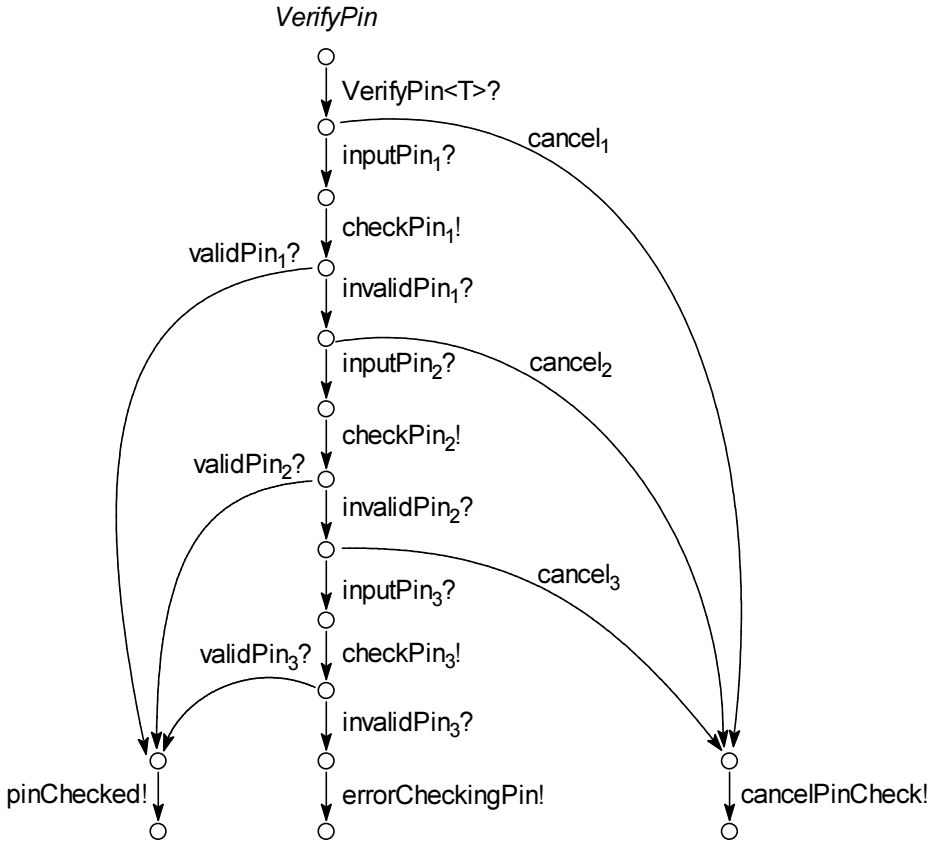
Na Rys. 7-15 przedstawiono specyfikację weryfikowanego procesu *Ctrl*. Jak można zauważyć, celowo został wprowadzony błąd w stosunku do specyfikacji kryterialnej *Level2* (Rys. 7-12), ponieważ wypłata pieniędzy następuje przed wysunięciem karty.



Rys. 7-15 Proces Ctrl

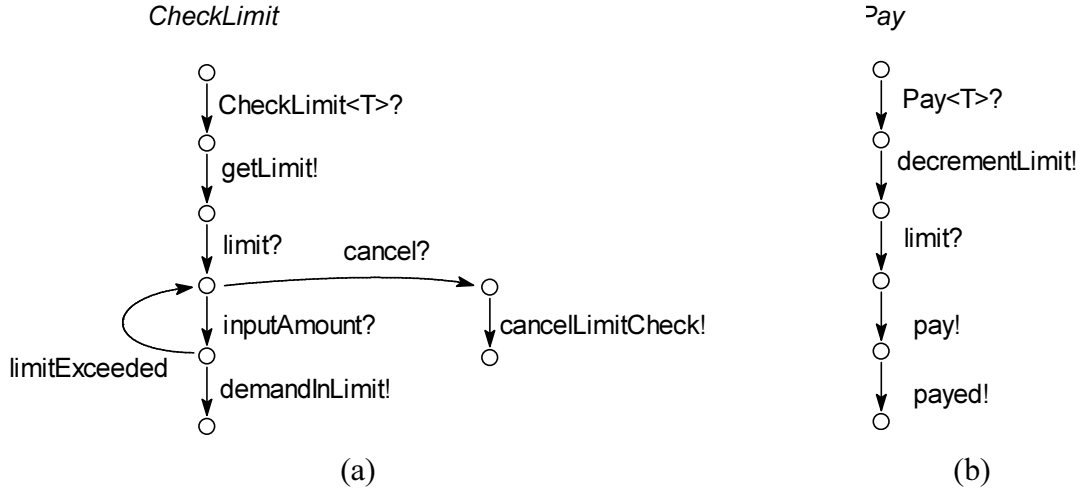
Definicja procesu *VerPin* jako maszyny skończenie stanowej pokazana została na Rys. 7-16. Dość złożona struktura specyfikacji odzwierciedla rzeczywiste działanie bankomatu, który zazwyczaj pozwala na trzykrotne wprowadzenie kodu i dopiero w przypadku, kiedy klient za każdym razem popełnił błąd zatrzymuje kartę.

Konstrukcja procesu odzwierciedla istnienie ukrytej zmiennej licznikowej, która przebiega wartości od 1 do 3, stąd powtarzające się operacje są indeksowane wartościami tej zmiennej.



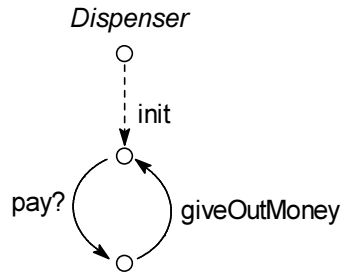
Rys. 7-16 Proces VerifyPin *(funkcja)*

Rysunki Rys. 7-17 a i b pokazują specyfikacje procesów *CheckLimit* oraz *Pay*. Proces *CheckLimit* woła usługę *getLimit* procesu *CardReader* i otrzymuje z powrotem wartość limitu wypłat zapisanego na karcie. Następnie sprawdza wielokrotnie kolejne żądane sumy, dopóki któraś z nich nie będzie mieściła się w zakresie lub nie zostanie użyty przycisk *cancel*.



Rys. 7-17 Procesy(a) CheckLimit *oraz (b)* Pay

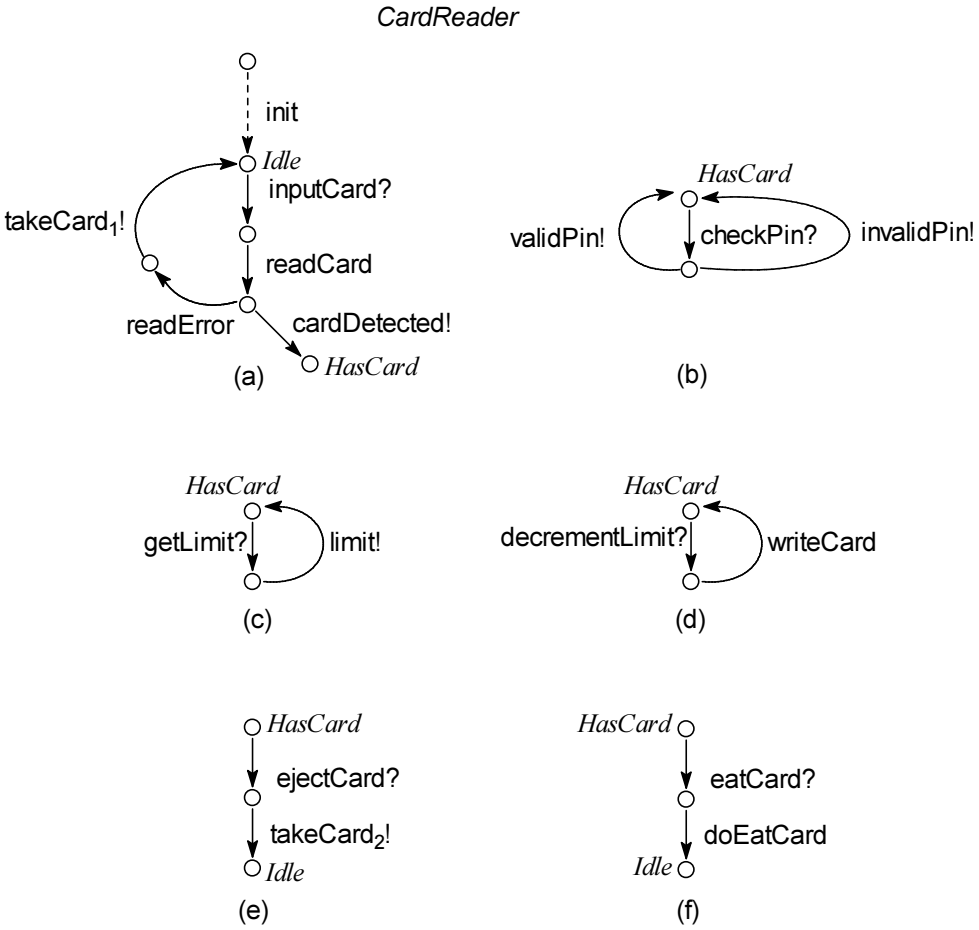
Przedstawiony na Rys. 7-18 proces *Dispenser* realizuje jedynie operację *giveOutMoney* w odpowiedzi na żądanie *pay*.



Rys. 7-18 Proces Dispenser

Proces *CardReader* (Rys. 7-19) przedstawiono dla zwiększenia czytelności w postaci kilku rozłącznych podprocesów; etykietami oznaczono powtarzające się stany. Rys. 7-19 a obejmuje sekwencję związaną z wprowadzeniem karty i odczytem zwartości. W przypadku błędu (*readError*) karta jest wysuwana, w przypadki poprawnego odczytu wysyłany jest sygnał *cardDetected* do procesu *Ctrl*. Pozostałe podprocesy opisują zachowanie przejawiane w odpowiedzi na zlecenie wykonania różnych usług: *checkPin* może zwracać *validPin* i *invalidPin*, *getLimit* zwraca zapisany na karcie limit wypłat. Sygnał *decrementLimit* powoduje zapisanie nowej wartości na karcie. Sama wartość jest ignorowana, ponieważ nie ma wpływu na przebieg sterowania.

Otrzymanie zlecenia *ejectCard* pociąga za sobą wykonanie *takeCard* (czyli wysunięcie karty), natomiast *eatCard* – umieszczenia karty w zasobniku.



Rys. 7-19 Proces CardReader

7.4.4 Funkcja obserwacji

Dla modelu specyfikacji złożonego z określonych wcześniej procesów zdefiniowano funkcję obserwacji następującej postaci:

$Level2.inputCard = CardReader.inputCard$

$Level2.ejectInvalidCard = CardReader.readError \circ CardReader.takeCard_1!$

$Level2.clientKnowsPin = VerPin.validPin_1 \oplus VerPin.validPin_2 \oplus VerPin.validPin_3$

$Level2.cancelPinCheck = VerPin.cancel_1 \oplus VerPin.\ cancel_2 \oplus VerPin.\ cancel_3$

$Level2.clientDoesNotKnowPin = VerPin.invalidPin_1 \circ VerPin.invalidPin_2 \circ VerPin.invalidPin_3$

$Level2.demandInLimit = CheckLimit.amountInLimit$

$Level2.cancelDemand = CheckLimit.cancel$

$Level2.ejectCard = CardReader.decrementLimit? \circ CardReader.writeCard \circ Ctrl.ejectCard_2 \circ CardReader.takeCard_2!$

$Level2.payMoney = Dispenser.giveOutMoney$

$Level2.ejectCardOnly = Ctrl.ejectCard_1 \circ CardReader.takeCard_2!$

$Level2.eatCard = Ctrl.eatCard_1 \circ CardReader.doEatCard_2!$

Interesująca z punktu widzenia siły wyrazu wydaje się odwzorowanie *clientDoesNotKnowPin* – do zaobserwowania takiej operacji wymagane jest wykonanie wszystkich operacji $VerPin.invalidPin_i$, dla $i \in \{1, 2, 3\}$. Jak widać, zrealizowanie operacji *Level2.payMoney* utożsamiane jest z operacją *Dispenser.giveOutMoney*, a nie jest utożsamione z sygnałem *payed* otrzymywanym z procesu *Pay*.

7.4.5 Rezultaty weryfikacji

Zgodnie z oczekiwaniami, rezultatem było stwierdzenie niepoprawności modelu. Jednakże specyfikacja okazała się potencjalnie poprawna, czyli istniały takie ciągi przejść elementarnych, które były zgodne ze specyfikacją i dla tych ciągów zaobserwowane zostały wszystkie operacje określone w specyfikacji kryterialnej.

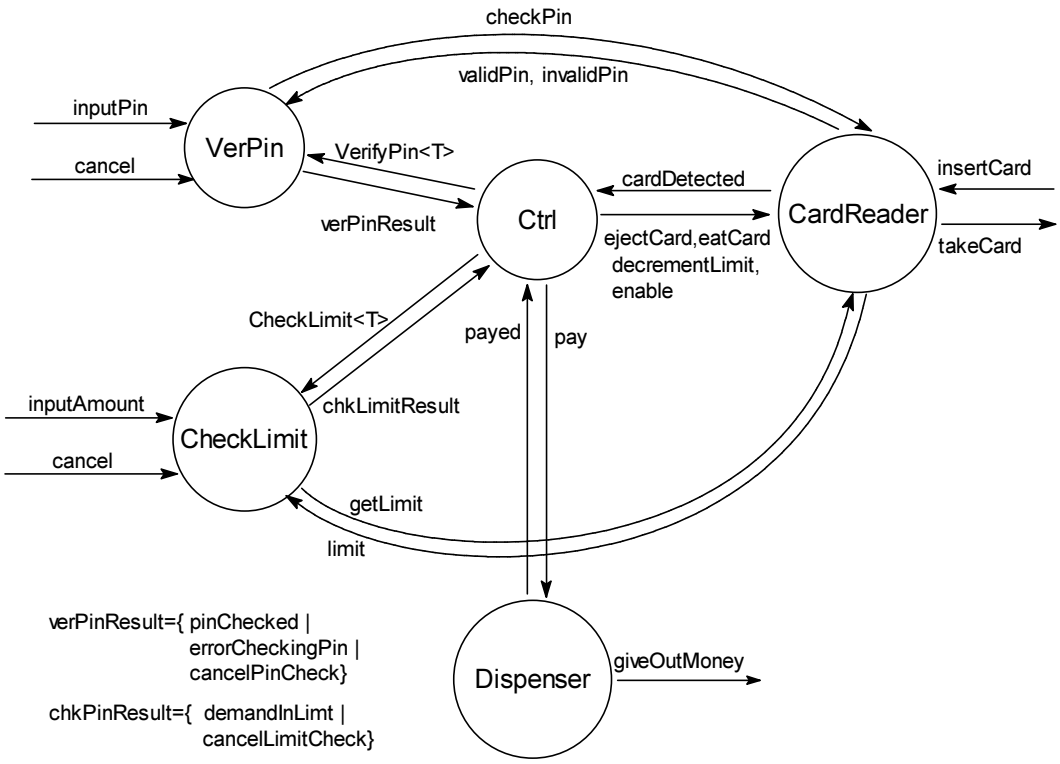
Proces *Dispenser* mimo otrzymania sygnału *pay* mógł przystąpić do realizacji wypłaty dopiero po tym, jak proces *CardReader* wysunął kartę. Taka sytuacja może mieć miejsce w praktycznie zrealizowanym oprogramowaniu, jeżeli na przykład priorytety zadań, którym przydzielono procesy *Ctrl*, *Pay* i *CardReader* są wyższe niż priorytet zadania obsługującego proces *Dispenser*. Błędnie skonstruowany system może poprawnie działać dla danego sprzętu i systemu operacyjnego, dopóki nie podjęta zostanie próba przeniesienia oprogramowania na inną platformę.

System wykazywał jednakże i inne błędy – możliwe było wysunięcie karty i przyjęcie nowej bez wcześniejszego wydania klientowi pieniędzy! Powodem był brak sygnału zwrotnego od procesu *Dispenser* pozwalającego na stwierdzenie, że została dokonana wypłata, oraz brak sygnału synchronizującego procesy *Ctrl* i *CardReader*.

7.4.6 Poprawiony system

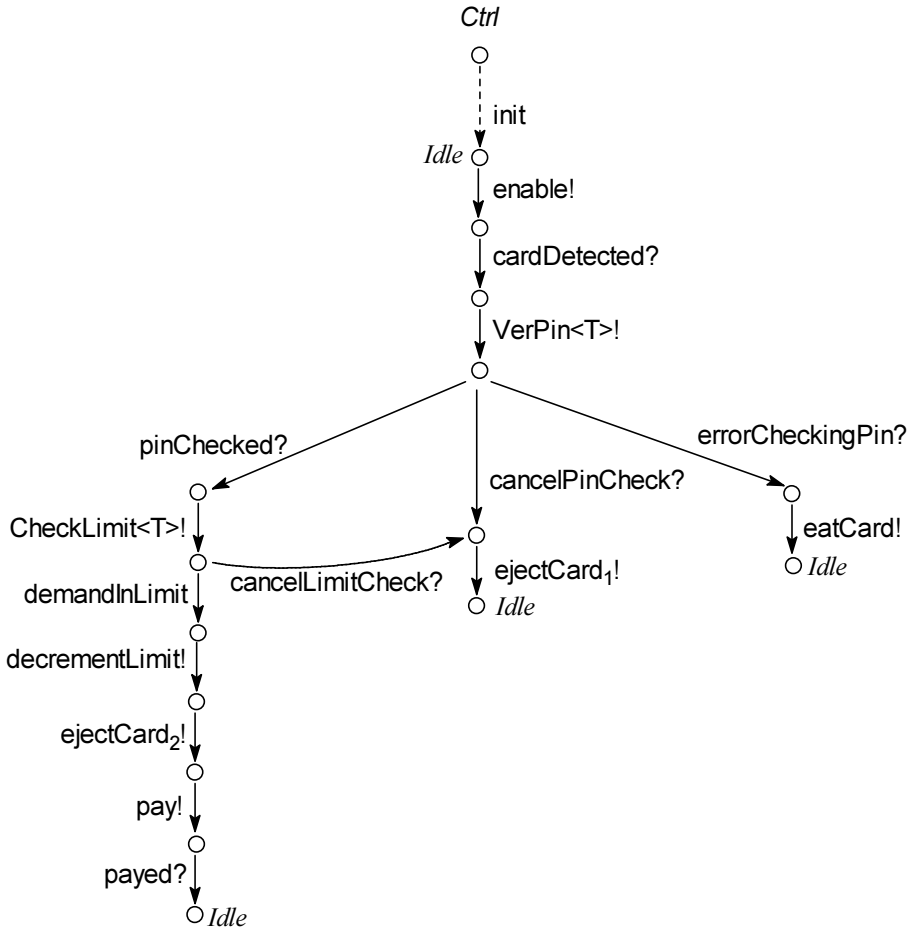
Na Rys. 7-20 pokazano komponenty poprawionego systemu. W stosunku do poprzedniej wersji dokonano następujących zmian:

1. Proces *Pay* został usunięty, jego operacje zostały przeniesione do procesu *Ctrl*
2. Proces *Dispenser* wysyła sygnał o dokonaniu wypłaty (*payed*)
3. Działanie procesu *Ctrl* i *CardReader* jest synchronizowane za pośrednictwem sygnału *enable*

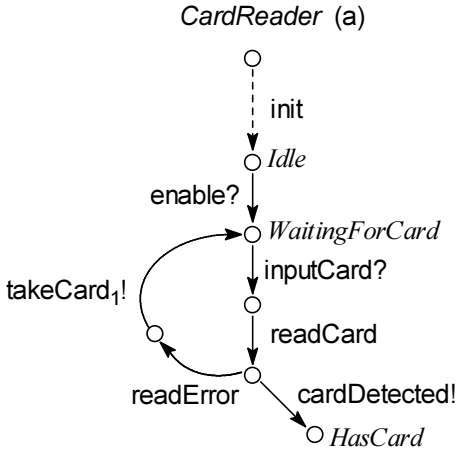


Rys. 7-20 Komponenty i komunikacje w poprawionym systemie

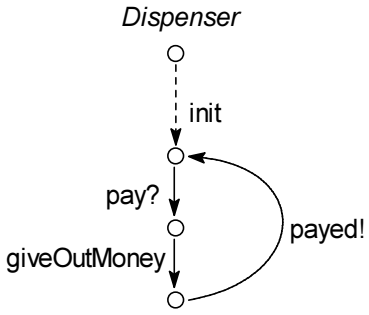
Rys. 7-21 pokazuje zmodyfikowaną wersję procesu *Ctrl*. Przed przejściem w stan oczekiwania na sygnał *cardDetected* proces ten wysyła sygnał *enable* do procesu *CardReader*, którego fragment pokazano na Rys. 7-22. Dopiero po otrzymaniu tego sygnału czytnik przyjmie nową kartę. Proces *Ctrl* wywołuje także usługę *decrementLimit* procesu *CardReader* oraz bezpośrednio komunikuje się z procesem *Dispenser* (Rys. 7-23). W odróżnieniu od poprzedniej wersji, proces ten wysyła sygnał zwrotny o dokonaniu operacji.



Rys. 7-21 Poprawiony proces Ctrl



Rys. 7-22 Poprawiony fragment procesu CardReader



Rys. 7-23 Poprawiona wersja procesu Dispenser

Testy zmodyfikowanego systemu przy tej samej funkcji obserwacji wykazały jego częściową poprawność spowodowaną wyłącznie obecnością dywergencji związanych z pętlą wewnątrz procesu *CheckLimit*. Ta pętla została jednak wprowadzona do specyfikacji celowo.

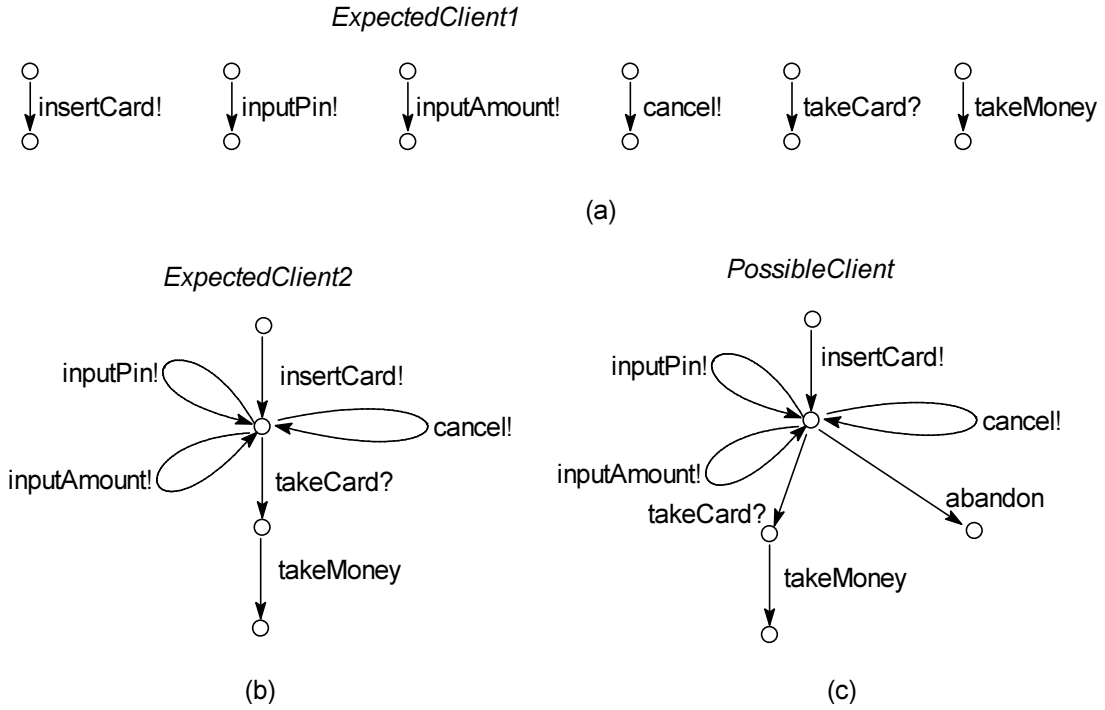
Zbiorcze zestawienie raportowane przez algorytm weryfikacji ujęto w poniższej tabeli.

Liczba wykonanych przejść	45
Liczba niepoprawnych przejść	0
Liczba stanów procesu sprzężonego	36
Liczba ciągów rozwiązań prowadzących do stanów pętlących	5
Liczba ciągów rozwiązań zawierających dywergencje	1
Liczba ciągów rozwiązań prowadzących do stanów blokujących	0
Liczba stanów końcowych	0

Jak można zauważyć, mimo obszernej specyfikacji wygenerowany proces jest nieduży. Dzieje się tak, ponieważ specyfikacja jest efektem dekompozycji funkcjonalnej, stąd podprocesy mają drzewiastą budowę i są mocno sprzężone z procesem nadrzędnym. Liczba niezależnych pętli, które są główną przyczyną złożoności obliczeniowej jest niewielka.

7.4.7 Otoczenie systemu

Powyższa specyfikacja nie obejmuje modelu otoczenia systemu. W konsekwencji otoczenie traktowane jest jako proces, który bez żadnych warunków gotów jest współpracować z systemem i we właściwej kolejności dokonywać odpowiednich akcji (o ile są dozwolone). Ten typ otoczenia jest pokazany na Rys. 7-24 a.



Rys. 7-24 Różne modele otoczenia systemu

Jak sprawdzono, system zachowuje się poprawnie również dla modelu otoczenia (klienta) przedstawionego na Rys. 7-24 b.

Jednakże konstruując tego typu oprogramowanie niezbędne jest uwzględnienie faktu (i wyspecyfikowanie tego w wymaganiach), że otoczenie może odmówić współpracy z systemem. Przykład takiego procesu jest pokazany na Rys. 7-24 c . W tym przypadku system osiągał stany blokujące za każdym razem, kiedy w procesie modelującym klienta wybrane zostanie przejście *abandon*.

W tym świetle przedstawiony przykład systemu daleki jest jeszcze od możliwych do zażyczenia własności funkcjonalnych, ponieważ każda operacja wymagająca współpracy otoczenia (np.: *inputPin*, *inputAmount*) powinna być chroniona przez licznik czasu, który w przypadku jej niewykonania powinien doprowadzić do zatrzymania karty i przejścia systemu w stan wyjściowy.

8. Poprawność dla specyfikacji rozszerzonej o model danych

8.1 Wprowadzenie

Tematem rozdziału jest opis specyfikacji weryfikowanej rozszerzonej o algebraiczny model danych. Przyjęto zasadę, że zdefiniowany zostanie pewien podstawowy zbiór typów danych, które wydają się najbardziej istotne z punktu widzenia modelowania. Celem wprowadzenia operacji na zmiennych jest wyłączenie uwzględnienie ich wpływu na przebieg sterowania; nie zakłada się modelowania związków pomiędzy danymi przetwarzanymi przez oprogramowanie.

Omawiane typy danych obejmują:

- zmienne wyliczeniowe
- zmienne licznikowe nieograniczone
- zmienne licznikowe ograniczone
- kolejki o nieograniczonej pojemności (typu FIFO)
- kolejki o ograniczonej pojemności (typu FIFO)

Opisane rozszerzenie modelu podstawowego polega na możliwości przypisania akcjom predykatów, których argumentami są wartości zmiennych oraz ciągów operacji wykonywanych na zmiennych. Każdemu z przejść może więc zostać przypisana komenda dozorowana podobnie, jak ma to ma miejsce w CRSM [Shaw 92] (Por. 2.2.4).

Powyższy zestaw typów danych został wybrany ze względu na możliwość:

1. uproszczenie struktury specyfikacji,
2. modelowania dynamicznej zmiany struktury systemu,
3. modelowania magazynowania sterowania i jego przepływu przez kanały asynchronicznej komunikacji (kolejki komunikatów).

Omawiane rozszerzenia stosują się w obecnej wersji wyłącznie do specyfikacji weryfikowanej. Główną przesłanką dla ich wprowadzenia jest uproszczenie specyfikacji modelu weryfikowanego i możliwość opisu szerszej klasy systemów. W modelu podstawowym (modelu liniowym) brak jest dozorowanych przejść. Wprowadzenie predykatów sterujących przejściami rozszerza model podstawowy o możliwość specyfikowania kolejności akcji w zależności od stanu danych. Model rozbudowany o dozorowane akcje może być porównywany z klasą sieci Petriego rozszerzonych o test zera realizowanych przez tzw. łuk zabraniający (ang.: *inhibitor arc*). [Peterson 81; Murata 89]

Zmiana modelu specyfikacji weryfikowanej pociąga za sobą propozycję zmian w definicji funkcji obserwacji. Zaproponowana postać będzie uzależniała swój wynik od wartości danych, przy których wykonywane były konkretne akcje.

8.1.1 Uproszczenie struktury systemu

Uproszczenie struktury specyfikacji jest najbardziej widoczną zaletą wprowadzenia danych do modelu, zwłaszcza, kiedy niezbędne jest modelowanie przesyłania rozróżnialnych wartości danych lub parametryzowanie nimi stanów.

Przedstawione w rozdziale 7 przykłady specyfikacji umożliwiały również modelowanie przesyłania danych, jako argumentów komunikacji. Sposób realizacji był podobny do rozwiązań proponowanych w CCS [Milner 89]. Każdej akcji parametryzowanej zmienną $v \in V$ odpowiadało $|V|$ różnych akcji, w wyniku których osiągane było $|V|$ stanów.

Stąd, dla agenta (procesu) zdefiniowanego jako:

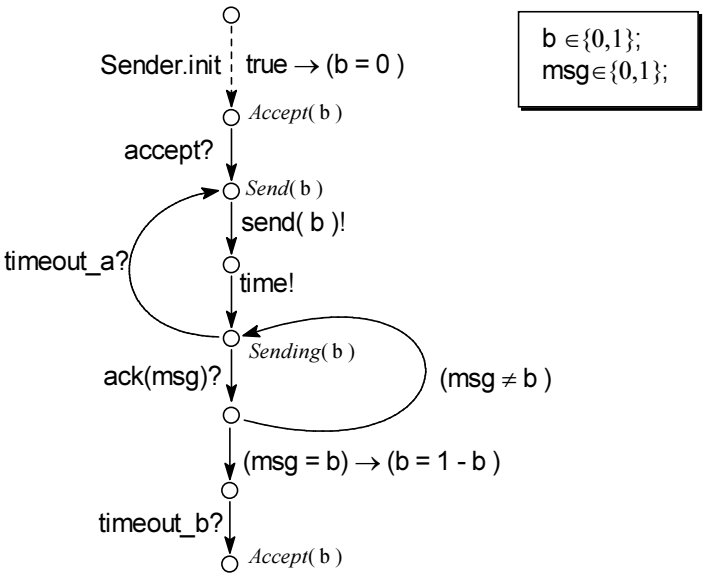
$$C \equiv in(v).C'(v),\ v \in V$$

konieczne było modelowanie rodziny przejść i stanów w postaci:

$$C \equiv \sum_{v \in V} in_v.C_v'$$

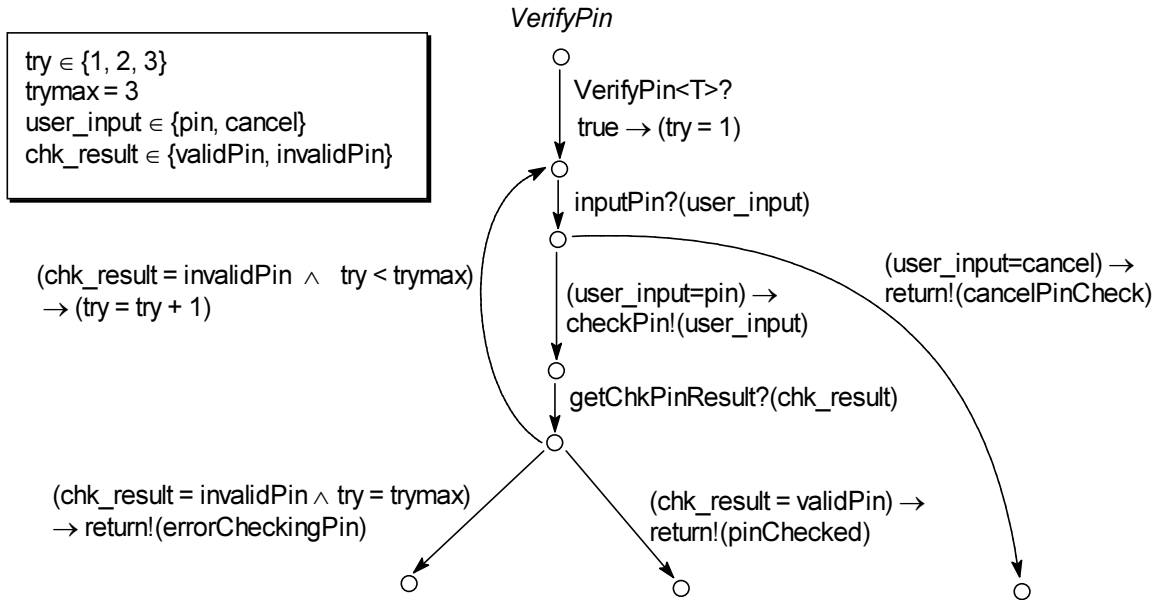
Rozważmy przykład specyfikacji procesu *Sender* dla protokołu z bitem potwierdzenia omawianego w rozdziale 7 (Rys. 7-7). Jak łatwo zauważyć, przedstawiony wcześniej przykład stanowił rozwinięcie oryginalnej specyfikacji CCS przez powtórzenie przejść odpowiadających różnym wartościom zmiennej binarnej *b* oraz odpowiednio podwojenie liczby stanów .

Ta sama specyfikacja w postaci bliższej oryginałowi pokazana jest na Rys. 8-1. Wprowadzono jawnie zmienną binarną *b* wchodzącą w skład stanu oraz zmienną *msg* odpowiadającą wartościom odczytanym z kanału *Ack*.



Rys. 8-1 Specyfikacja procesu Sender

Jeszcze większe uproszczenie można zaproponować dla procesu *VerPin* (przykład bankomatu, Rys. 7-16). Zdefiniowany wcześniej proces był bardzo rozbudowany, ponieważ procedura wprowadzania i sprawdzania kodu mogła być wykonywana trzykrotnie. Zmodyfikowana i bardzo uproszczona wersja przedstawiona została na Rys. 8-2. Zauważmy, że ostateczna „rozwinięta" postać procesu parametryzowana jest stałą *trymax*. W przypadku, kiedy przyjmuje ona wartość *trymax* = 3, jawne modelowanie rozwiniętego procesu jest jeszcze możliwe, jednakże dla większych wartości staje się niezwykle kłopotliwe.



Rys. 8-2 Specyfikacja VerPin

Na obu rysunkach występują komendy dozorowane postaci:

predykat → *operacja*

Brak takiego symbolu oznacza, że predykat przyjmuje domyślną wartość prawdy (*true*).

8.1.2 Dynamiczna zmiana struktury systemu

Przez dynamiczną zmianę struktury systemu rozumiana jest możliwość zmiany statusu pojedynczej akcji lub częściej grupy akcji składających się na proces. Zmiany te zachodzą w trakcie wykonania systemu i polegają na wyłączeniu (uniemożliwieniu aktywacji) lub włączeniu (zezwoleniu na aktywację) .

Typowym przykładem zastosowań są sygnały sterujące *Disable* i *Enable* w specyfikacji SART [WM 85; Perez 90]. Pierwszy z nich wstrzymuje wykonanie współbieżnego procesu, drugi z nich reaktywuje proces. Najprostszym rozwiązaniem jest związanie z procesem P_i zmiennej logicznej $enabled_{Pi}$ i użycie jej jako argumentu predykatu dozorującego wszystkie akcje procesu. Z akcjami procesu, który wysyła sygnały *Disable* lub *Enable* wiąże się wtedy operację przypisania $enabled_{Pi} = false$ lub $enabled_{Pi} = true$.

Alternatywą jest oczywiście modelowanie wartości zmiennej $enabled_{Pi}$ za pomocą dwóch stanów (czy miejsc w sieci Petriego). Aktywność stanu (lub obecność żetonu w pewnym miejscu) modeluje stan zawieszenia lub aktywności procesu. Podobne rozwiązania zastosowano w [ELP 93]. Jak można zauważyć, struktura otrzymanej w ten sposób specyfikacji jest sztucznie rozbudowana o dodatkowe łuki łączące każde przejście procesu z miejscem modelującym stan aktywności.

Rozwiązania oparte o jawnie zdefiniowane zmienne opisujące stan były użyte przy modelowaniu i weryfikacji aplikacji opisanych językiem LACATRE [SS 94]. Z każdym zadaniem we współbieżnym systemie związana była zmienna opisująca jego stan (*ready*, *delayed*, *pending*, itp.) Wszystkie przejścia w stworzonym modelu były dozorowane predykatem (*state* = *ready*).

8.1.3 Kanały asynchronicznej komunikacji

Omówiony w rozdziale 4 model specyfikacji zakładał wyłącznie synchroniczną komunikację pomiędzy procesami. W celu zamodelowania komunikacji asynchronicznej, konieczne było wprowadzenie pośredniczącego procesu implementującego bufor komunikatów. Łatwość modelowania bufora komunikatów zależy przede wszystkim od jego wielkości. W przypadku, kiedy jego pojemność wynosi 1 (jak np.: dla kanałów *Trans* i *Ack* w przykładzie z bitem potwierdzenia) modelowanie jest bardzo proste. Podobnie, łatwe jest modelowanie bufora o ograniczonej lub nieograniczonej pojemności, który zawiera nierozróżnialne komunikaty.

Jednakże zbudowanie opisu bufora o pojemności *n*, który przechowuje rozróżnialne komunikaty i działa jak kolejka FIFO jest już bardzo kłopotliwe. Konieczne jest wówczas wprowadzenie do opisu systemu *n* buforów o pojemności 1 sprzężonych ze sobą dodatkowymi ukrytymi komunikacjami. (Dyskusję równoważności znaleźć można w [Milner 89].)

Wprowadzenie zmiennych zachowujących się jak proces i pełniących rolę pasywnych buforów bardzo upraszcza specyfikację. Podobne podejście stosowane było przy modelowaniu kolejek komunikatów i semaforów licznikowych – w pracach nad weryfikacją aplikacji opisanych językiem LACATRE. Jak opisano w podrozdziale 2.2.2, kolejki komunikatów mogą być jawnym elementem systemu definiowanego w postaci zbioru komunikujących się asynchronicznie maszyn skończenie stanowych.

8.2 Typy danych

W obecnej wersji prototypowego oprogramowania obsługującego rozszerzony model opisane dalej typy danych stanowią klasy C++, natomiast predykaty i operacje dokonywane na zmiennych są zapisywane jako funkcje w tym języku. Nie jest to wygodnym rozwiązaniem, ponieważ wymaga generacji kodu funkcji i ich kompilacji i przed uruchomieniem zadania weryfikacji.

Dla docelowej wersji oprogramowania planuje się raczej użycie interpretera wyrażeń, dlatego też taka postać zostanie przyjęta przy opisie operacji. Alternatywą mógłby być opis klas, które w rzeczywistości zostały zaimplementowane, np.: z użyciem notacji Z++, podobnie jak prowadzone są specyfikacje w [Lano 95].

Przedstawiona składnia języka zapisu predykatów i operacji bazuje na ONP (odwrotnej notacji polskiej) i podobna jest np.: do języka PostScript [PostScript 85]. Zakładane są jednakże znaczne uproszczenia – np.: brak tu definicji funkcji. Ogólne zasady interpretacji wyrażeń dla tego typu języka są ogólnie znane, dlatego też nie będą tu dokładniej opisywane.

Znanym językiem interpretacji wyrażeń jest Lisp [Graham 96]. Różnicą składniową w stosunku do przedstawionych tu definicji jest umieszczanie na stosie interpretera nazwy operacji przed argumentami zamiast na ich końcu oraz użycie nawiasów zamykających listę argumentów.

8.2.1 Zmienne i zapis operacji na zmiennych

Def. 8-1 Zmienna

Zmienna zdefiniowana jest proces $PV = (V, B, T, A, name, eval)$ gdzie:

- V – jest zbiorem wartości (stanów),
- $B \subset V$ – jest dokładnie jednoelementowym zbiorem zawierającym wartość początkową;
- $T \subset V \times V$ – jest niejawnie podanym zbiorem przejść związanych z wykonaniem operacji; zbiór stanów (wartości) końcowych został pominięty jako zbiór pusty. Zakłada się również, że $B \subset \text{Ran } T$
- A – jest zbiorem symboli
- $name \in A$ – jest nazwą zmiennej
- *eval* – jest funkcją ewaluacji wyrażeń

■

Zbiór zmiennych występujących w specyfikacji modelowany będzie jako zbiór procesów.

Niech *expr* oznacza ciąg symboli ze zbioru A. Ciąg ten zapisywany będzie w postaci: $(\alpha_1\ \alpha_2\ \alpha_3\ ....\ \alpha_n)$, gdzie $\alpha_i \in$ A. Dodatkowo dopuszczamy pojawienie się wewnątrz nawiasów symbolizujących początek i koniec wyróżnionych podciągów. Nawiasy te są raczej elementem syntaktycznym, dlatego też nie będą szczegółowo omawiane. Zakładamy, że zbiór A zawiera pewne predefiniowane symbole; są to:

true – wynik obliczenia wartości predykatu;

false – wynik obliczenia wartości predykatu;

error – błąd interpretacji wyrażenia;

Omawiając poszczególne operacje na zmiennych posługiwali będziemy się dodatkowym wewnętrznym identyfikatorem *self*; należy go traktować jako symbol identyfikujący daną zmienną.

Oznaczmy przez $S(A)$ zbiór wszystkich ciągów symboli alfabetu A.

Dla danej zmiennej PV operacje na niej zostaną opisane w kategoriach funkcji częściowej *eval* , która odwzorowuje $V \times S(A) \rightarrow V \times S(A)$. Przyjęto standardową formę opisu par postaci $((v,expr) , (v',expr')) \in eval$.

Szczególnym typem operacji są predykaty, które nie zmieniając wartości zmiennych przekształcają wyrażenie *expr* w $expr' \in \{(true),(false),(error)\}$. Mogą być one zbudowane z użyciem operatorów *and*, *or*, *not* , które jako oczywiste zostaną w opisie pominięte w opisie.

Ważną rolę w algorytmie generacji procesu sekwencyjnego i weryfikacji odgrywa predykat *covers*, który używany jest dla wykrywania stanów pokrywających. Nie należy on bezpośrednio do języka operacji, ale będzie on opisany w ten sam sposób. Jak można będzie zauważyć, predykat ten przyjmuje zawsze wartość *false* dla zmiennych, których zbiór wartości jest ograniczony.

8.2.2 Zmienne wyliczeniowe

Zmienna wyliczeniowa przyjmuje wartości ze zbioru symboli $V \subset A$. Zdefiniowane są dla niej jedynie operacje *set* oraz *get* . Zadaniem zmiennych wyliczeniowych jest przede wszystkim modelowanie wartości komunikatów oraz zapis stanu obiektu. W typowych specyfikacjach, jak CRSM [Shaw 92], SART, [WM 85], SDL [BH 93] proces otrzymuje komunikat z asynchronicznego kanału i dalej w zależności od jego wartości podejmuje dalsze działanie.

Operacja *set*

Dla $v = v_0$ i *expr* = (*self val set*)

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = val; expr' = () \; if \; val \in V \\ v' = v_0; expr' = (error) \; if \; val \notin V \end{cases}$$

Operacja *get*

Dla $v = v_0$ i *expr* = (*self get*)

$$eval(v,expr) = (v',expr'), \text{ gdzie } v' = v_0; expr' = (v_0)$$

Predykat *covers*

Dla $v = v_0$ i *expr* = (*self val covers*)

$$eval(v,expr) = (v',expr'), \text{ gdzie } v' = v_0; expr' = (false)$$

8.2.3 Zmienne licznikowe nieograniczone

Zmienne licznikowe przyjmują wartości ze zbioru $V = \{0\} \cup N$. Operacje *set* i *get* zdefiniowane są analogicznie jak dla zmiennych wyliczeniowych. Dodatkową operacją jest *inc*.

Operacja *inc*

Dla $v = v_0$ i *expr* = *(self inc)*

$$eval(v,expr) = (v',expr'), \text{ gdzie } v' = v_0 + 1; expr' = ()$$

Predykat *covers*

Dla $v = v_0$ i *expr* = *(self val covers)*

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = v_0; expr' = (true) \; if \; val \le v_0 \\ v' = v_0; expr' = (false) \; if \; val > v_0 \\ v' = v_0; expr' = (error) \; if \; val \notin V \end{cases}$$

8.2.4 Zmienne licznikowe ograniczone

Zmienne licznikowe ograniczone liczbą całkowitą *range* przyjmują wartości ze zbioru $V = \{0 \ldots range\}$. Operacje *set* i *get* zdefiniowane są analogicznie jak dla zmiennych wyliczeniowych.

Operacja *inc*

Dla $v = v_0$ i *expr = (self inc)*

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = v_0 + 1; expr' = () \; if \; v_0 < range \\ v' = v_0; expr' = (error) \; if \; v_0 = range \end{cases}$$

Predykat *inRange*

Dla $v = v_0$ i *expr = (self inRange)*

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = v_0; expr' = (true) \; if \; v_0 < range \\ v' = v_0; expr' = (false) \; if \; v_0 = range \end{cases}$$

Predykat *covers*

Dla $v = v_0$ i *expr = (self val covers)*

$$eval(v,expr) = (v',expr'), \text{ gdzie } v' = v_0; expr' = (false)$$

8.2.5 Kolejki o nieograniczonej pojemności

Kolejki nieograniczone zawierają ciągi symboli należące do pewnego zbioru wyliczeniowego *C* (alfabetu); stąd *V*=*S*(*C*). Zdefiniowane są dla nich operacje *get* (analogicznie jak dla wszystkich zmiennych), *send*, *retrieve* oraz predykat *isEmpty*.

Zadaniem kolejki jest modelowanie asynchronicznego kanału przesyłania wiadomości.

Operacja *send*

Dla $v = v_0$ i *expr* = (*self* α *send*)

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = v_0 \bullet \langle \alpha \rangle; expr' = () \; if \; \alpha \in C \\ v' = v_0; expr' = (error) \; if \; \alpha \notin C \end{cases}$$

Operacja *retrieve*

Dla $v = v_0$ i *expr* = (*self retrieve*)

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = tail(v_0); expr' = (\alpha) : head(v_0) = \langle \alpha \rangle if \; v_0 \neq \langle \rangle \\ v' = v_0; expr' = (error) \; if \; v_0 = \langle \rangle \end{cases}$$

Predykat *isEmpty*

Dla $v = v_0$ i *expr* = (*self isEmpty*)

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = v_0; expr' = (true) \; if \; \#v_0 = 0 \\ v' = v_0; expr' = (false) \; if \; \#v_0 > 0 \end{cases}$$

Zapis $\# v$ oznacza liczbę elementów ciągu *v*.

Predykat *covers*

Dla $v_1, v_2 \in S(C)$ zdefiniujmy relację $\prec$ jako:

$$v_1 \prec v_2 \Leftrightarrow \forall \alpha \in C(v_1 \downarrow \alpha \leq v_2 \downarrow \alpha),$$

gdzie $v \downarrow \alpha$ oznacza liczbę wystąpień symbolu α w ciągu v.

Dla $v = v_0$ i $expr = (self\ val\ covers)$

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = v_0; expr' = (true)\ if\ val \prec v_0 \\ v' = v_0; expr' = (false)\ if\ \neg(val \prec v_0) \\ v' = v_0; expr' = (error)\ if\ val \notin V \end{cases}$$

Definicja predykatu *covers* dla kolejki o nieograniczonej pojemności jest dyskusyjna. Przyjęta jest ona przez analogię do znakowania pokrywającego dla kolorowych sieci Petriego. Zmienna typu kolejka istotnie różni się jednak od miejsca dla kolorowych sieci Petriego, ponieważ z kolejki „żetony" mogą być z niej usuwane tylko w takiej kolejności w jakiej zostały wstawione.

Jako alternatywę można rozważać definicję relacji $\prec$ jako:

$$v_1 \prec v_2 \Leftrightarrow \exists v_{12} \in S(C): v_2 = v_1 \bullet v_{12}$$

Przy takiej definicji ciąg v_2 będzie „pokrywał" ciąg v_1 jeżeli będzie on jego podciągiem początkowym. Jak się wydaje, można mnożyć różne definicje, których celowość użycia zależy od struktury konkretnych specyfikacji.

8.2.6 Kolejki o ograniczonej pojemności

Kolejki o ograniczonej pojemności zawierają ciągi symboli należące do pewnego zbioru C (alfabetu) o długości nie przekraczającej stałej *range*,; stąd $V=\{s \in S(C) \mid \# s \leq range\}$. Dla kolejek o ograniczonej pojemności zdefiniowany jest predykat *isFull*; zmienia się także semantyka operacji *send*.

Operacja *send*

Dla $v = v_0$ i $expr = (self\ \alpha\ send)$

$$eval(v,expr) = (v',expr'), \text{ gdzie } \begin{cases} v' = v_0 \bullet \langle \alpha \rangle; expr' = ()\ if\ (\alpha \in C \wedge \# v_0 < range) \\ v' = v_0; expr' = (error)\ if\ (\alpha \notin C \vee \# v_0 = range) \end{cases}$$

Predykat *covers*

Dla $v = v_0$ i $expr = (self\ val\ covers)$

$$eval(v,expr) = (v',expr'), \text{ gdzie } v' = v_0; expr' = (false)$$

8.3 Model danych

Def. 8-2 Model danych

Model danych zdefiniowany jest jako krotka:

$$MD = (Vars, Alpha, Values, B, eval, Pred, Operations, T), \text{ gdzie:}$$

- *Vars* –to zbiór zmiennych; zakładamy, że nazwa każdej zmiennej jest unikalna, czyli *Vars*(*name*) identyfikuje zmienną o nazwie *name*

- A – jest alfabetem
- $Values = \prod_{var \in Vars} V_{var}$ – zbiór wartości, przez V_{var} oznaczono zbiór wartości zmiennej *var*; przez Π iloczyn kartezjański;
- $B = \prod_{var \in Vars} B_{var}$ – jest jednoelementowym zbiorem zawierającym wartość początkową (iloczyn kartezjański zbiorów wartości początkowych zmiennych)
- *eval*–funkcja ewaluacji
- *Pred* –zbiór predykatów zapisanych jako ciągi symboli $S(A)$

  Zak³adamy, ¿e

  $\forall\ val \in Values\ .\forall\ pred \in Pred\ .\ eval(\ (val\ ,\ pred\ )\ )=(\ val'\ ,\ result\ ) \wedge val = val'.$

  Zakładamy, że predykat $(true) \in Pred.$
- *Operations* – jest zbiorem operacji zapisanych jako ciągi symboli $S(A)$.

  Zakładamy, że do zbioru należy operacja pusta : $(\ ) \in Operations.$
- $T \subset Values \times Values$ – jest relacją przejścia

  $(val_1, val_2) \in T \Leftrightarrow$

  $\exists\ op \in Operations\ .\ eval((val_1,op))=(\ val_2,result) \wedge result \neq (error)$

■

8.3.1 Funkcja ewaluacji dla modelu danych

Działanie funkcji ewaluacji dla modelu danych nie będzie dokładnie omawiane, ponieważ wymagałoby to zdefiniowanie gramatyki pewnego języka wyrażeń i podania dokładnego modelu semantycznego – podobnie jak dla zmiennych składowych. Taki opis wydaje się dość typowy – i mało interesujący.

Przykładem zasad interpretacji może być zapis przypisania $y = x$, gdzie y, x są zmiennymi wyliczeniowymi, jako (y (x *get*) *set*). Wyrażenie to jest interpretowane w kolejnych krokach:

Wyrażenie	Wartości zmiennych
(y (x get) set)	x = a ; y = b
(y a set)	x = a ; y = b
( )	x = a ; y = a

Z punktu widzenia zastosowań, najistotniejsze jest założenie, że *eval* jest funkcją, czyli że dla dwóch dowolnych stanów modelu danych $val_1, val_2 \in Values$, $val_1 = val_2$ i wyrażeń $expr_1, expr_2 \in S(A)$, $expr_1 = expr_2$ zachodzi:

$$eval(\ (val_1, expr_1)\ ) = eval(\ (val_2, expr_2)\ ) = (val', result)$$

Dzięki temu założeniu model danych MD zachowuje się deterministycznie: dla danego stanu *val* i zdarzenia wejściowego *expr* przechodzi zawsze do stanu *val'* i zwraca zdarzenie wyjściowe *result* .

8.3.2 Predykaty equal i covers dla modelu danych

Dla dowolnej pary wartości val_1 i val_2 zmiennej $PV = = (V, B, T, A, name, eval)$ zdefiniujmy

$$equal(val_1, val_2) \Leftrightarrow \quad \forall\, expr \in S(A).$$
$$eval_{var1}\,((var_1\,,\, expr)) = ((var_1\text{'}\,,\, expr\text{'})) \wedge$$
$$eval_{var1}\,((var_2\,,\, expr)) = ((var_2\text{'}\,,\, expr\text{'}))$$
$$\wedge\, equal(var_1\text{'}, var_2\text{'})$$

W szczególności $val_1 = val_2$ spełnia $equal(val_1, val_2)$.

Dla dowolnej wartości $val_1 \in Values$ oznaczmy przez $val_1(name)$ wartość zmiennej o nazwie *name* .

Dla dowolnych par wartości modelu danych $MD : val_1, val_2 \in Values$ zdefiniujmy

$$equal(val_1, val_2) \Leftrightarrow \quad \forall\, var = (V, B, T, A, name, eval) \in Vars.$$
$$equal(val_1(name)\,,\, val_2(name)\,)$$

Dla dowolnych par wartości modelu danych $MD : val_1, val_2 \in Values$ zdefiniujmy

$$covers(val_1, val_2) \Leftrightarrow \quad \forall\, var = (V, B, T, A, name, eval) \in Vars.$$
$$equal(val_1(name)\,,\, val_2(name)\,) \vee covers(val_1(name)\,,\, val_2(name)\,)$$
$$\exists\, var = (V, B, T, A, name, eval) \in Vars.$$
$$covers(val_1(name)\,,\, val_2(name)\,)$$

8.4 Model specyfikacji rozszerzony o MD

8.4.1 Definicja modelu

Def. 8-3 Model specyfikacji rozszerzony o model danych

Model specyfikacji rozszerzony o model danych jest czwórką $MLD = (ML, MD, \eta, \varphi)$, gdzie

1. *ML* – jest modelem liniowym $ML = (X\,, x_0\,, A_I\, x \le b, A_E\, x = 0)$
2. *MD* – jest modelem danych $MD = (Vars, Alpha, Values, B, eval, Pred, Operations, T)$
3. Niech $E \subset X = \{e_i \mid i = 1,\ldots,n\}$. Przez e_i oznaczamy wersor, który ma element równy 1 na *i*-tej pozycji, pozostałe elementy zerowe. η jest funkcją: $E \rightarrow Pred$
4. Niech *V* będzie zbiorem przejść elementarnych modelu ML. φ jest funkcją: $V \rightarrow Operations$

■

Składnik *ML* rozszerzonego modelu specyfikacji opisuje przepływ sterowania (akcje). Każdej akcji przypisany jest pewien predykat (w szczególności może on zawsze przyjmować wartość prawdy). Wykonaniu przejścia elementarnego towarzyszy wykonanie operacji modyfikującej aktualną wartość modelu danych *MD*.

Zakładamy, że predykaty przydzielone są akcjom, natomiast operacje na danych przejściom elementarnym – w tym komunikacjom.

Podczas wykonania modelu *MLD* kolejno wybierane są dopuszczalne przejścia elementarne $\langle v_1, v_2,\ldots\rangle$ i generowany jest zarówno ciąg rozwiązań $s(x_0 , \bullet) = \langle x_0, x_1,\ldots\rangle$ modelu *ML*, jak i ciąg wartości danych $\langle val_0, val_1,\ldots\rangle$.

Rozwiązaniem modelu *MLD* będziemy dalej nazywali parę $(x_i , expr_i)$, gdzie $x_i \in X$, natomiast $expr_i \in S(A)$ spełnia warunek $expr_i = expr_{i-1} \bullet \varphi(v_i)$.

Stanem modelu *MLD* będziemy nazywali parę (y_i , val_i), gdzie y_i jest stanem modelu liniowego wchodzącego w skład *ML*, natomiast val_i wartością modelu danych.

Dla osiągalnego stanu (y_i , val_i) zachodzi $y_i = y(x_i)$ oraz $eval(\ (val_0, expr_i)) = (val_i, ())$, gdzie val_0 jest wartością początkową dla MD.

Def. 8-4 Zbiór przejść dopuszczalnych w modelu MLD

Zbiorem przejść dopuszczalnych $V_F(y_i , val_i)$ dla stanu modelu MLD (y_i , val_i) nazywany będzie podzbiór $V_F(y_i)$ spełniający:

$\forall\ v \in V_F(\ y_i , val_i\ )\ .\ \forall\ j\ .\ (\ v(\ j\ ) \neq 0 \Rightarrow eval(val_i, \eta(e_j)) = (val_i,(true))\ )$

$\forall\ v \in V_F(\ y_i\ ) \setminus V_F(\ y_i , val_i\ )\ .\ \exists\ j\ .\ v(\ j\ ) \neq 0 \wedge eval(\ (val_i\ ,\ \eta(e_j)\ )\ ) = (val_i,(false))$,

gdzie e_j oznacza wersor z elementem 1 na *j*–tej pozycji.

■

Analogicznie będziemy mówić o zbiorze przejść dopuszczalnych $V_F(x_i , val_i\ )$ dla rozwiązania x_i i stanu val_i.

Def. 8-5 Ciąg rozwiązań osiągalnych modelu MLD

Ciągiem rozwiązań osiągalnych modelu $MLD = (ML, MD, \eta, \varphi)$ dla rozwiązania początkowego x_0 i początkowej wartości danych $val_0 \in Values$ będziemy nazywali ciąg postaci

$$s(\ (x_0 , expr),\ \bullet\ ) = \langle\ (x_0, ()), (x_1, expr_1),\ldots, (x_i, expr_i),\ldots\rangle$$

spe³niaj¹cy:

$$eval(\ (\ val_0 , expr_i\ )\ ) = (\ val_i , result\ ) \wedge result \neq (error)$$

$$v_i = x_i - x_{i-1} \in V_F(x_{i-1} , val_{i-1}\ )$$

Będziemy mówić, że ciąg $s(\ x_0\ ,\ \bullet\ ) = \langle\ x_0, x_1,\ldots, x_i,\ldots\rangle$ jest ciągiem rozwiązań składowej *ML* modelu *MLD* jeśli istnieje ciąg rozwiązań osiągalnych $s(\ (x_0 , expr_0),\ \bullet\ )$ taki, że kolejne elementy x_i ciągu $s(\ x_0 , \bullet\ )$ są składnikami kolejnych par wchodzących w skład ciągu $s(\ (x_0 , expr_0)\ , \bullet\ )$.

Bedziemy mówić, że ciąg wartości $s(\ val_0 , \bullet\ ) = \langle\ val_0, val_1,\ldots, val_i,\ldots\rangle$ odpowiada danemu ciągowi rozwiązań składowej *ML* $s(\ x_0\ ,\ \bullet\ ) = \langle\ x_0, x_1,\ldots, x_i,\ldots\rangle$ jeśli istnieje ciąg rozwiązań osiągalnych $s(\ (x_0 , expr_0)\ , \bullet\ )$ taki, że dla każdego *i* zachodzi:

$$eval(\ (\ val_0 , expr_i\ )\ ) = (\ val_i , result\ ) \wedge result \neq (error)$$

■

Ze względu na to, że wraz z wykonywaniem kolejnych przejść ciągi wyrażeń wchodzące w skład rozwiązań modelu *MLD* mogą potencjalnie rosnąć w nieskończoność, rola rozwiązań w bezpośredniej analizie będzie mniejsza. Bardziej interesująca będzie reprezentacja mieszana zbioru rozwiązań w postaci par $(x,val) \in X \times Values$, reprezentujących zbiór rozwiązań postaci $\{(x_i,expr_i) \mid x_i = x \wedge eval(val_0, expr_i) = (val, ())\}$.

Def. 8-6 Mieszany ciąg rozwiązań

Mieszanym ciągiem rozwiązań osiągalnych modelu $MLD = (ML, MD, \eta, \varphi)$ dla rozwiązania początkowego x_0 i początkowej wartości danych $val_0 \in Values$ będziemy nazywali ciąg postaci

$$s((x_0 , val_0), \bullet) = \langle (x_0, val_0), (x_1, val_1),\ldots, (x_i, val_i),\ldots\rangle$$

spełniający: $\forall\, j \in \{1,\ldots,i\}$.

$$x_j - x_{j-1} \in V_F(x_{j-1} , val_{j-1})$$

$$eval((val_{j-1} , \varphi(v_j))) = (val_j , result) \wedge result \neq (error)$$

■

Każdemu ciągowi rozwiązań można przypisać mieszany ciąg rozwiązań przez obliczenie wartości stanu $eval(val_0, expr_i)$.

8.4.2 Wykonanie rozszerzonego modelu specyfikacji

Poniżej przedstawiony jest algorytm wykonania modelu MLD, czyli generacji sekwencji rozwiązań i stanów.

1. Wyznacz zbiór przejść elementarnych V .
2. Podstaw $x_i = x_0$, $val_i = val_0$, $i = 0$.
3. Wyznacz $V_F(x_i , val_i)$ zbiór przejść elementarnych dopuszczalnych w rozwiązaniu x_i i stanie val_i.
4. Jeśli zbiór $V_F(x_i , val_i) = \varnothing$ przejdź do 9.
5. Wybierz dowolne przejście $v_i \in V_F(x_i)$.
6. Podstaw $x_{i+1} = x_i + v_i$;
7. Oblicz $(val_{i+1} , result) = eval((val_i,\varphi(v_i)))$;
   jeœli $result \neq () \rightarrow$ STOP
8. Podstaw $i := i+1$; przejdź do 3.
9. Dokonaj analizy rozwiązania x_i i stanu val_i kończącego sekwencję.

W modelu liniowym zdefiniowano predykat *final*(*x*) (por. 4.7.4), który służył do rozróżniania pomiędzy osiągnięciem stanu końcowego i blokowania dla pustego zbioru przejść dopuszczalnych w rozwiązaniu *x*. Analogiczne warunki będą stosowane dla sprawdzania, czy wyznaczono stan końcowy w modelu *MLD*.

Można jednakże rozważyć dodatkowe wymagania dotyczące stanu danych. Model danych nie ma jawnie określonego stanu końcowego i dla wielu typów danych trudno jest podać uzasadnienie dla wyboru określonej wartości, jako poprawnej w stanie końcowym. Typowe asercje dotyczące wartości danych w stanie końcowym dotyczą kolejek komunikatów – wymaga się, aby były one puste. Takie założenie można spotkać w [Holzmann 91, 97],

podobne wymagania były stosowane przy weryfikacji aplikacji opisanych językiem LACATRE [SS 94; SSSS 94; SS 95a, 95b].

Zdefiniujmy przez analogię predykat *final*(*val*), gdzie *val* ∈ *Values* , który w zależności od konkretnych wymagań może przyjmować zawsze wartość prawdy, lub tylko wtedy, kiedy zmienne typu kolejkowego spełniają predykat *isEmpty*().

8.4.3 Proces sekwencyjny opisujący zachowanie MLD

Specyfikacja modelu *MLD* może być bardzo zwięzła, ale w wielu przypadkach zbiór stanów procesu sekwencyjnego opisującego jego zachowanie może być nieskończony. Tak zazwyczaj będzie się działo, jeśli komponenty modelu danych będą mogły przyjmować nieograniczone wartości. Zwłaszcza dotyczy to kolejek komunikatów o nieskończonej pojemności. Nawet dla skończonych kolejek komunikatów zbiór osiągalnych stanów modelu może być bardzo duży i przekraczać możliwości obliczeniowe maszyny .

Ze względów formalnych podamy konstrukcję procesu $\widetilde{P} = (\widetilde{S}, \widetilde{B}, \widetilde{F}, \widetilde{T})$ opisującego zachowanie modelu *MLD* – konstrukcja ta jest modyfikacją procesu opisującego zachowanie modelu liniowego.

Proces $\widetilde{P}$ jest skonstruowany jako granica ciągu procesów $\Theta = \langle P^{(0)},\ldots,P^{(i)},\ldots,P^{(n)}, \ldots\rangle$ postaci $P^{(i)} = (S^{(i)}, B^{(i)}, F^{(i)}, T^{(i)})$, gdzie $S^{(i)} = \mathrm{R}^n \times Values$.

Stany procesów zapisywali będziemy jako pary (*x* , *val*), gdzie *x* jest rozwiązaniem modelu *ML*, natomiast *val* jest wartością modelu danych *MD*. Oczywiście każdej parze (*x* , *val*) można przypisać stan modelu *MLD* postaci (y(*x*) , *val*).

Zdefiniujmy relacjê *Similar* $\subset (\mathrm{R}^n \times Values) \times (\mathrm{R}^n \times Values)$ jako:

$$Similar = \{((x_1,val_1) , (x_2, val_2)) \mid y_1 = y(x_1), y_2 = y(x_2) \wedge (y_1 = y_2) \wedge (equal(val_1, val_2) \}$$

Proces rozpoczynający ciąg opisany jest następującymi zależnościami

1. $S^{(0)} = B^{(0)} = \{ (x_0, val_0)\}$, gdzie x_0 jest rozwiązaniem początkowym modelu *ML*, $val_0 \in Values$ jest wartością początkową modelu danych MD.
2. $F^{(0)} = \begin{cases} B^{(0)} & \text{jeżeli } V_F(y(x_0), val_0) = \varnothing \wedge final(x_0) \wedge final(val_0) \\ \varnothing & \text{w p. p.} \end{cases}$
3. $T^{(0)} = \varnothing$

Przejście od $P^{(i-1)}$ do $P^{(i)}$ następuje jeśli

$$\exists\, s_k \in S^{(i-1)} \setminus F^{(i-1)}\ s_k = (x_k, val_k) \wedge \exists\, v \in V_F(y_k , val_k) .\ s_i = (x_k + v, val_i) \notin S^{(i-1)},$$

gdzie $eval((val_k, \varphi(v))) = (val_i , result)$ i $result \neq (error)$

Wówczas konstruowany jest nowy proces $P^{(i)}$ postaci:

1. $S^{(i)} = S^{(i-1)} \cup \{ s_i \}$, $s_i = (x_k + v, val_i)$
2. $B^{(i)} = B^{(i-1)}$
3. $T^{(i)} = T^{(i-1)} \cup \{(s_k, s_i)\}$
4. Zbiór $F^{(i)}$ modyfikowany jest następująco:
   a) Jeśli $V_F(y_i) = \varnothing \wedge \forall\, i \in N .\ A_I\, x_i = 0$ to $F^{(i)} = F^{(i-1)} \cup \{ s_i \}$

b) Jeśli $V_F(y_i) \neq \varnothing$ oraz $\exists\ s_r \in S^{(i-1)}$, $(s_i, s_r) \in Similar$, to $F^{(i)} = F^{(i)} \cup \{ s_i \}$

c) Jeżeli nie zachodzi żaden z powyższych przypadków $F^{(i)} = F^{(i-1)}$

■

Dla ciągu Θ skonstruowanego w wyżej opisany sposób brak jest gwarancji jego skończoności. Powodem może być nieskończony zbiór osiągalnych wartości modelu danych i pokrywanie stanów.

Skończoność procesu ciągu gwarantować może poniższa definicja relacji *Similar* :

$$Similar = \{((x_1,val_1) , (x_2, val_2)) \mid y_1 = y(x_1), y_2 = y(x_2) \wedge (y_1 = y_2 \vee y_1 \prec y_2) \wedge (equal(val_1, val_2) \vee covers(val_1, val_2))\}$$

Otrzymany w ten sposób proces będzie podprocesem procesu $\widetilde{P}$ (procesem częściowym).

Zmienne występujące w składowym modelu danych są traktowane jak zmienne globalne. Podobnie, jak model liniowy nie pozwala na dynamiczne tworzenie instancji procesów, model danych nie pozwala na tworzenie instancji zmiennych.

Analizując przykłady generacji procesu sekwencyjnego dla kilku uruchomionych specyfikacji, zauważono, że nie rozróżnianie zmiennych lokalnych prowadzi do nadmiernego rozrostu automatycznie wyznaczonych procesów w stosunku do postaci rozwiniętej, którą można jawnie wyspecyfikować – podobnie jak było to przeprowadzone dla procesu *Sender*.

Powodem niepożądanego rozrostu jest to, że zmienne, które mają charakter lokalny wchodzą w skład stanu modelu *MLD*. Prowadzi to do niepotrzebnego rozróżniania stanów i przejść dla różnych wartości zmiennych lokalnych, nawet jeśli wiemy, że nie mają one żadnego znaczenia dla dalszego wykonania modelu.

Typowym przykładem powstawania nadmiarowości może być rozwijanie procesu opisującego zachowanie bankomatu dla różnych wartości zmiennej lokalnej *try* występującej w specyfikacji procesu *VerPin*. Po zakończeniu działania podprocesu *VerPin* i zwróceniu wartości *validPin* wygenerowany proces będzie zawierał gałęzie dla wartości *try* = 1, *try* = 2 i *try* = 3. Aby zaradzić temu zjawisku, wydaje się celowe wprowadzenie do modelu znacznika pełniącego rolę wartości nieokreślonej i konsekwentne przypisywanie go zmiennym lokalnym w stanie początkowym i po zakończeniu użycia.

Opisany sposób przeprowadzania operacji na danych w modelu *MLD* wymaga nadawania wartości zmiennym przed użyciem. Jest przez to potencjalnie znacznie mniej wydajny niż modele dowodzenia pewnych własności, które nie przydzielają zmiennym konkretnych wartości lecz używają asercji o zbiorach wartości. Przykładem takiego działania mogą być mechanizmy wnioskowania stosowane w Prologu [SB 87].

Tego typu podejście mogłoby być interesujące w przypadku zmiennych, którym w sposób niedeterministyczny są nadawane wartości poza systemem (poprzez komunikacje zewnętrzne). Obecna postać modelu wymaga w takich przypadkach dodania procesów symulujących otoczenie, których zadaniem jest inicjowanie przesyłanych danych konkretnymi wartościami.

Alternatywna postać modelu może zakładać, że zmienna, której w momencie komunikacji zewnętrznej przypisywana jest zdefiniowana poza systemem wartość x przybiera dowolną wartość ze swojej dziedziny $Dom(x)$. W miarę poruszania się w głąb procesu dozory przejść (predykaty) powodują ograniczanie zbioru możliwych wartości, aż do ustalenia

wartości konkretnej lub osiągnięcia zbioru pustego (co jest równoznaczne z zaobserwowaniem blokowania). Jest to interesujące podejście, które być może zostanie rozważone w przyszłości, szczególnie, że mogłoby ułatwić analizę własności w przypadku kolejek komunikatów

Warto zwrócić uwagę, że podobna strategia dowodzenia zwykle jest stosowana w formalnych dowodach dla systemów opisanych z użyciem pewnej algebry. Przykładem może być dowód bisymulacji dla protokołu ze zmianą bitów przedstawiony w [Milner 89]

W podrozdziale 4.8 podano przekształcenie modelu liniowego do macierzowego zapisu równań stanu sieci Petriego. W wyniku przekształcenia powstaje sieć, której tranzycje odpowiadają przejściom elementarnym, natomiast pary łuków prowadzącym od miejsc do tranzycji i od tranzycji do miejsc mogą być przypisane akcjom.

W takim ujęciu model *MLD* może być traktowany jako rozszerzona sieć Petriego, dla której łukom wejściowym tranzycji przypisane są predykaty zezwalające, natomiast tranzycjom operacje na danych. Można znaleźć pewne analogie pomiędzy definicją modelu *MLD* i numerycznych sieci Petriego oraz modelem przetwarzania danych w sieciach TRANSNET [Sacha 95]. Podobnie, jak dla modelu *ML* proces sekwencyjny opisujący zachowanie modelu *MLD* odpowiada grafowi stanów osiągalnych zmodyfikowanej sieci.

8.5 Warunkowa funkcja obserwacji

8.5.1 Wprowadzenie

W podrozdziale omówiony zostanie model poprawności zakładający, że przedmiotem weryfikacji jest model rozszerzony *MLD* , natomiast specyfikacja kryterialna jest wyrażona w postaci modelu liniowego *ML*'. Postać problemu jest więc asymetryczna, co może utrudniać zastosowanie w przypadku wieloetapowej analizy kolejnych warstw specyfikacji.

Zgodnie z uwagami zamieszczonymi w podrozdziale 8.1.1, model liniowy rozszerzony o dane pozwala na bardziej zwięzły zapis specyfikacji, bez konieczności rozwijania systemu i etykietowania przejść oraz stanów sterowania wartościami danych. Z punktu widzenia specyfikacji wymagań, wybrane przejścia dla różnych wartości danych powinny być rozróżnialne.

Przykładem tego może być specyfikacja odwzorowania dla przykładu z bankomatem. Fragment definicji funkcji obserwacji (por. 7.4.4) opisywał odwzorowanie postaci:

$Level2.cancelPinCheck = VerPin.cancel_1 \oplus VerPin.cancel_2 \oplus VerPin.cancel_3$

Akcje $cancel_1$, $cancel_2$, $cancel_3$ jawnie występowały w specyfikacji procesu *VerPin* (Rys. 7-16). Nie występują one jednak specyfikacji na Rys. 8-2. Zaobserwowanie akcji $cancel_1$ możemy powiązać z wykonaniem akcji *MLD* *inputPin*?(*user_input*) przy wartościach danych: *user_input* = *cancel* i *try* = 1.

Uogólniając, akcji $cancel_k$ odpowiada:

inputPin?(*user_input*) dla wartości danych *user_input* = *cancel* i *try* = *k*.

Podobnie akcja $validPin_k$ zostanie zaobserwowana dla akcji:

getChkPinResult?(*chk_result*), gdy *chk_result* = *validPin* i *try* = *k* ,

natomiast $invalidPin_k$ dla:

getChkPinResult?(*chk_result*), gdy *chk_result* = *invalidPin* i *try* = *k* .

Jak widać z powyższych przykładów, celem funkcji obserwacji będzie rozwinięcie specyfikacji opisanej jako model liniowy rozszerzony o dane *MLD* do postaci procesu opisanego bezpośrednio modelem liniowym *ML*. Rozwiniecie to nie będzie jednak pełne; z góry zakładamy, że pożądana jest postać kongruencyjna pomijająca wartości pewnych zmiennych. Na przykład, do stwierdzenia, że zaobserwowano akcję $cancel_k$ wartość zmiennej *chk_result* nie musi być brana pod uwagę.

Opisując problem poprawności, który za przedmiot weryfikacji bierze model *MLD* zakładamy, że obserwacja akcji procesu rozwiniętego może być utożsamiona z wykonaniem akcji w modelu *MLD* przy spełnieniu pewnych warunków zdefiniowanych dla wartości danych. Z tego względu zaproponowana funkcja obserwacji będzie nazywana *warunkową funkcją obserwacji*. Jej ogólna konstrukcja polegała na przypisaniu wybranym akcjom modelu *MLD* zbioru predykatów, których argumentami są wartości danych składowej *MD* i na tej podstawie dokonanie odwzorowania w zbiór akcji specyfikacji kryterialnej.

Pierwszym pytaniem jakie się nasuwa jest, czy brane są pod uwagę prewarunki (czyli czy argumentami dla predykatów są wartości przed wykonaniem akcji), czy też postwarunki, czyli brane są pod uwagę obliczone wartości danych.

Zakładamy, że funkcja obserwacji będzie funkcją postwarunkową. Dla uzasadnienia tego wyboru przeanalizujmy podaną wcześniej specyfikację akcji $invalidPin_k$. Do jej obliczenia konieczna jest wartość zmiennej *chk_result* ustalona w wyniku wykonania akcji, a nie wartość poprzednia, która w szczególności może być wartością nieokreśloną. Możliwa jest oczywiście konstrukcja mieszana dopuszczająca zarówno prewarunki, jak i postwarunki. Na razie tego typu konstrukcje nie były rozważane.

8.5.2 Definicja warunkowej funkcji obserwacji

Warunkowa funkcja obserwacji traktowana będzie jako złożenie dwóch przekształceń. Pierwsze z nich nazywane jest *funkcją ekspansji* γ. Jego zadaniem jest odwzorowanie wykonanych akcji w modelu weryfikowanym *MLD* dla różnych wartości danych w akcje pewnej (zazwyczaj szerszej) przestrzeni X_γ. Drugie przekształcenie jest zdefiniowane jako niehomomorficzna funkcja obserwacji π, która odwzorowuje $X_\gamma \rightarrow X'$, gdzie X' jest przestrzenią rozwiązań modelu służącego jako specyfikacja kryterialna.

Standardowo, specyfikując odwzorowanie γ będziemy je zapisywali jako:

$$akcja_{X\gamma} = akcja_{ML} \textbf{ at } predykat_{MD}$$

Poniższy przykład odzwierciedla ten sposób zapisu.

γ: $cancel_1 = inputPin?(user\_input)$ **at** $(user\_input = cancel \wedge try = 1)$

γ: $cancel_2 = inputPin?(user\_input)$ **at** $(user\_input = cancel \wedge try = 2)$

γ: $cancel_3 = inputPin?(user\_input)$ **at** $(user\_input = cancel \wedge try = 3)$

π: $Level2.cancelPinCheck = VerPin.cancel_1 \oplus VerPin.cancel_2 \oplus VerPin.cancel_3$

Kluczowym elementem funkcji obserwacji warunkowej jest funkcja ekspansji. Funkcja ta musi zostać zdefiniowana rekurencyjnie, ponieważ operacji na danych nie można w ogólnym przypadku cofnąć i ich wynik jest uzależniony od kolejności operacji.

Nie zmienia to jednak ogólnej struktury algorytmów weryfikacji, ponieważ rozważanym modelem jest ciąg rozwiązań i warunki poprawności dotyczą akcji zaobserwowanych przy wykonaniu przejścia pomiędzy kolejnymi rozwiązaniami.

Przed zdefiniowaniem odwzorowania γ zdefiniujemy przekształcenie pomocnicze *cond*, którego zadaniem jest obliczenie zmiany w rozszerzonej przestrzeni X_γ dla przejścia elementarnego *v* modelu *MLD* i obliczonych wartości danych *val*.

Def. 8-7 Odwzorowanie cond

Niech $MLD = (ML, MD, \eta, \varphi)$ będzie modelem liniowym rozszerzonym o model danych, gdzie

$ML = (X, x_0, A_I\, x \le b, A_E\, x = 0)$

$MD = (\, Vars, Alpha, Values, B, eval, Pred, Operations, T\,)$

Załóżmy, że rozmiarem przestrzeni X jest n , $B = \{val_0\}$, rozmiarem przestrzeni X_γ jest k.

Niech *pred* będzie odwzorowaniem: $\{1,\ldots,k\} \rightarrow Pred$.

Bêdziemy pisali

$pred(\, val\, ,\, i\,) = true$ jeœli $eval(val\, ,\, pred(\, i\,)) = (val, (true))$ oraz

$pred(\, val\, ,\, i\,) = false$ jeœli $eval(val\, ,\, pred(\, i\,)) = (val, (false))$, $val \in Values$.

Niech C będzie macierzą o wymiarach $k \times n$, której elementy przyjmują wartości ze zbioru {0,1}. Odwzorowanie $cond : X \times Values \rightarrow X_\gamma$ jest zdefiniowane jako:

$$cond(v,val) = v_\gamma \Leftrightarrow v_\gamma(\, i\,) = \begin{cases} 1 & \text{if} \quad \min\{v(\, j)\,|\, C(i,j) \neq 0\} = 1 \wedge pred(val,i) = true \\ 0 & \text{in other case} \end{cases}$$

Przez $C(\, i\, ,\, j\,)$ oznaczono element macierzy w *i*–tym wierszu i *j*–tej kolumnie.

■

Macierz *C* została wprowadzona dla przypisania zbiorom akcji elementarnych predykatów. W szczególności zbiory te mogą być jednoelementowe lub zawierać akcje składające się na przejście elementarne (komunikację). Pojawienie się wszystkich akcji z danego zbioru opisanego przez wiersz macierzy C i spełnienie predykatu dla aktualnych wartości danych prowadzi do zaobserwowania przejścia w przestrzeni X_γ . Oczywiście danemu zbiorowi akcji może odpowiadać kilka predykatów związanych z różnymi obserwowalnymi przejściami. W zależności od wartości poszczególnych predykatów mogą być obserwowane różne przejścia.

Jeśli dla danej akcji $act \in \{1,\ldots,k\}$ przestrzeni X_γ nie jest wymagana obserwacja warunkowa, a więc powinna być obserwowana niezależnie od stanu danych, funkcja *pred*(*act*) przyjmuje wartość (*true*). Jeśli dana akcja modelu weryfikowanego nie jest obserwowalna – opisana jest przez kolumnę macierzy C wypełnioną zerami.

Def. 8-8 Funkcja ekspansji

Dla danego modelu $MLD = (ML, MD, \eta, \varphi)$ oraz przestrzeni X_γ funkcja ekspansji γ zdefiniowana jest jako $\gamma : X_\gamma \times X \times Values \rightarrow X_\gamma$

$$\gamma(\, x_{\gamma\, i-1}\, ,\, v,\, val) = x_{\gamma\, i}$$

gdzie zachodzi $x_{\gamma\, i} = x_{\gamma\, i-1} + cond(\, v\, ,\, val\,)$

■

Typowym zastosowaniem funkcji ekspansji będzie odwzorowanie ciągu rozwiązań składowej *ML* i odpowiadającego mu ciągu wartości danych. Dla ciągów $s(x_0 , \bullet) = \langle x_0, x_1,\ldots, x_i,\ldots\rangle$ oraz $s(val_0 , \bullet) = \langle val_0, val_1,\ldots, val_i,\ldots\rangle$ konstruowany będzie ciag postaci $s(x_{\gamma 0} , \bullet) = \langle x_{\gamma 0}, x_{\gamma 1},\ldots, x_{\gamma i},\ldots\rangle$, gdzie $x_{\gamma i} \in X_\gamma$, którego kolejne wyrazy będą spełniały:

$$x_{\gamma 0} = \gamma(0 , x_0, val_0)$$

$$x_{\gamma i} = \gamma(x_{\gamma i-1} , \; x_i - x_{i-1} , val_i)$$

Dla danego zbioru *Z* niech *S*(*Z*) oznacza zbiór wszystkich ciągów jego elementów.

Def. 8-9 Warunkowa funkcja obserwacji

Niech $MLD = (ML, MD, \eta, \varphi)$ będzie modelem specyfikacji weryfikowanej, X_γ i X' przestrzeniami.

Niech γ będzie funkcją ekspansji : $X_\gamma \times X \times Values \rightarrow X_\gamma$,

Niech π będzie funkcją obserwacji (liniową lub niehomomorficzną) : $X_\gamma \rightarrow X'$.

Zdefiniujmy funkcjê $\pi_\gamma : S(X \times Values) \rightarrow S(X_\gamma)$:

$$\pi_\gamma (\langle \rangle) = 0;$$

$$\pi_\gamma (\langle (x_0,val_0) \rangle) = \gamma(0 , x_0, val_0)$$

$$\pi_\gamma (s_i) = \gamma(\pi_\gamma (s_{i-1}), \; x_i - x_{i-1} , val_i),$$

gdzie $s_i = s_{i-1} \bullet \langle (x_i,val_i) \rangle = s_{i-2} \bullet \langle (x_{i-1},val_{i-1}) \rangle \bullet \langle (x_i,val_i) \rangle$, $s_i, s_{i-1}, s_{i-2} \in S(X \times Values)$

Zdefiniujmy $\pi_s : S(X_\gamma) \rightarrow S(X')$ jako:

$$\pi_s(\langle x_{\gamma 0}, x_{\gamma 1},\ldots, x_{\gamma i},\ldots\rangle) = \langle \pi(x_{\gamma 0}), \pi(x_{\gamma 1}),\ldots, \pi(x_{\gamma i}),\ldots\rangle$$

Funkcja obserwacji warunkowej π_{cond} odwzorowuje $S(X \times Values) \rightarrow S(X')$ i zdefiniowana jest jako złożenie funkcji π_s i π_γ

$$\pi_{cond} = \pi_s \cdot \pi_\gamma$$

■

Wniosek 8-1

Niech (y_i,val_i) będzie stanem modelu *MLD*. Niech $v \in V_F(y_i,val_i)$.

Załóżmy, że w wyniku wykonania przejścia *v* wyznaczony zostanie następny stan (y_{i+1},val_{i+1}) spełniający:

$$y_{i+1} = y_i - A \; v \wedge eval((val_i , \varphi(v))) = (val_{i+1} , ())$$

Przejściu $(y_i,val_i) \rightarrow (y_{i+1},val_{i+1})$ odpowiada przejście Δx_γ w przestrzeni X_γ opisane zależnością: $\Delta x_\gamma = \gamma(0, v, val_{i+1})$

■

8.6 Poprawność dla warunkowej funkcji obserwacji

Podobnie, jak dla modelu weryfikowanego *ML* zdefiniujemy poprawność względną ciągu rozwiązań modelu *MLD*. Definicja będzie zakładała niehomomorficzną postać składnika π_s funkcji obserwacji, co jest przypadkiem bardziej ogólnym.

Przedstawiona dalej definicja wykorzystuje pojęcia odwrotnej dopuszczalności i lokalnej osiągalności, które były zdefiniowane dla wektorów przestrzeni X i ich ciągów weryfikowanego modelu *ML*. Definicje tych pojęć nie zależą bezpośrednio od własności modelu postaci ML, ale od funkcji obserwacji π – stąd przenoszą się bez zmian dla przestrzeni X_γ .

Def. 8-10 Poprawność względna ciągu rozwiązań

Dane są dwa modele specyfikacji

MLD = (*ML*, *MD*, η, φ) – model specyfikacji weryfikowanej) oraz

ML' = (X', x_0', A_I' x' ≤ b', A_E' x' = 0) (model specyfikacji kryterialnej)

Dana jest funkcja obserwacji warunkowej $\pi_{cond} : S(X) \rightarrow S(X')$ zdefiniowana jak w Def. 8-9, a więc określona przez funkcję ekspansji γ i funkcję obserwacji π.

Zakładamy, że $\pi_{cond}(\langle (x_0, val_0) \rangle) = \langle x_0' \rangle$, czyli spełnione jest

$$x_{\gamma 0} = \gamma(0 , x_0, val_0)$$

$$\pi(x_{\gamma 0}) = x_0'.$$

Niech $s((x_0 , val_0), \bullet)$ będzie mieszanym ciągiem rozwiązań postaci:

$\langle (x_0 , val_0), (x_1 , val_1), (x_2 , val_2),\ldots, (x_i , val_i), (x_{i+1} , val_{i+1}), \ldots\rangle$.

Niech $s(x_{\gamma 0} , \bullet) = \langle x_{\gamma 0}, x_{\gamma 1},\ldots, x_{\gamma i},\ldots\rangle$, będzie ciągiem obliczonych wartości w przestrzeni X_γ, czyli $x_{\gamma 0} = \gamma(0 , x_0, val_0)$; $x_{\gamma i} = \gamma(x_{\gamma i-1} , x_i - x_{i-1} , val_i)$.

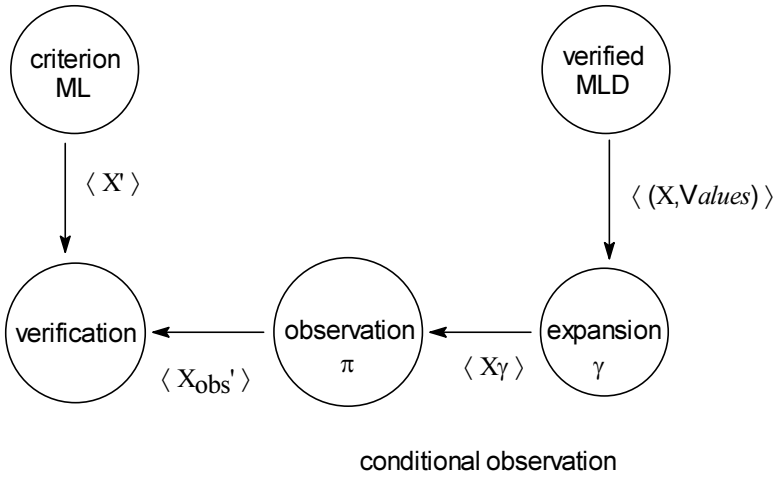
Ciąg $s((x_0 , val_0), \bullet)$ jest poprawny, jeżeli:

1. Dla każdej pary rozwiązań $(x_{\gamma i}, x_{\gamma i+1})$ spełnione są następujące warunki:
   a) Oznaczmy przez $\Delta x_\gamma = x_{\gamma i+1} - x_{\gamma i}$. Warunkiem *odwrotnej dopuszczalności* jest: $\forall i.(\Delta x_\gamma(i) \neq 0 \wedge e_{\gamma i} \in O_\pi \Rightarrow e_{\gamma i} \in RE_{X\gamma}(x_{\gamma i}))$
   b) Przejście w specyfikacji kryterialnej $v_c' = \pi(x_{\gamma i+1}) - \pi(x_{\gamma i})$ spełnia:
      $v_c' = 0$ lub
      $v_c' \in V_F(\pi(x_{\gamma i}))$ i v_c' jest dopuszczalnym przejściem złożonym w $\pi(x_{\gamma i})$.
   c) Przejście v_c' jest *lokalnie osiągalne* w ciągu rozwiązań $s(x_{\gamma 0} , x_{\gamma i+1})$
2. Jeżeli ciąg $s((x_0 , val_0), \bullet)$ jest skończony, wówczas jego ostatnie rozwiązanie (x_n , val_n) jest rozwiązaniem końcowym, czyli $final(x_n) \wedge final(val_n)$.
3. Jeśli (x_n , val_n) rozwiązaniem końcowym to $x_n' = \pi(x_{\gamma n})$ spełnia $final(x_n')$.

■

Zaproponowana definicja poprawności jest modyfikacją definicji wprowadzonej w podrozdziale 6.2.3. Odzwierciedla ona omówione wcześniej stwierdzenie, że zachowanie weryfikowanego modelu liniowego może być traktowane jako rozwinięcie zachowania modelu rozszerzonego o dane. Stąd główną różnicą jest przedmiot weryfikacji – jest nim ciąg rozwiązań modelu *MLD*, który poprzez funkcję ekspansji jest zamieniany na ciąg rozwiązań $s(x_{\gamma 0} , \bullet)$ przestrzeni obserwacji X_γ. Do tego ostatniego stosowane standardowe warunki opisujące poprawność.

Rys. 8-3 ilustruje konstrukcję definicji poprawności .



Rys. 8-3 Schemat poglądowy modelu poprawności

Analogicznie, jak w przypadku definicji poprawności ciągów rozwiązań modelu *ML* zdefiniujmy predykat *correct* ($s((x_0, val_0), \bullet), \pi, ML'$), który przyjmuje wartość prawdy, jeśli ciąg $s((x_0, val_0), \bullet)$ jest poprawny względem modelu *ML*' dla warunkowej funkcji obserwacji.

Def. 8-11 Dywergencja ciągu rozwiązań

Niech $s((x_0, val_0), \bullet) = \langle (x_0, val_0), (x_1, val_1), \ldots, (x_i, val_i), (x_{i+1}, val_{i+1}), \ldots \rangle$ będzie mieszanym ciągiem rozwiązań.

Niech $s(x_{\gamma 0}, \bullet) = \langle x_{\gamma 0}, x_{\gamma 1}, \ldots, x_{\gamma i}, \ldots \rangle$, będzie odpowiadającym mu ciągiem wartości w przestrzeni X_γ wyznaczonym jak w Def. 8-10.

Niech π będzie dowolną funkcją obserwacji.

Mówimy, że ciąg $s((x_0, val_0), \bullet)$ zawiera dywergencję, jeśli istnieje jego podciąg $s_{div}((x_{div,1}, val_{div,1}), (x_{div,n}, val_{div,n}))$ ciągu $s(x_0, \bullet)$ taki, że:

1. $y(x_{div\,1}) = y(x_{div\,n}) \vee y(x_{div\,1}') \prec y(x_{div\,n}')$
2. $val_{div,1} = val_{div,n}$
3. $\forall\, i : div_1 \le i \le div_n \,.\, \pi(x_{\gamma i}) = \pi(x_{\gamma\, div\,1})$

■

Definicje poprawności częściowej i całkowitej modelu weryfikowanego *MLD* względem modelu *ML* (specyfikacji kryterialnej) przenoszą się bez zmian. Tak więc dla poprawności częściowej będziemy żądali, by wszystkie mieszane ciągi rozwiązań modelu *MLD* były poprawne, natomiast dla poprawności całkowitej będziemy stawiali wymaganie braku dywergencji i pokrycia wszystkich akcji przestrzeni X'.

8.6.1 Proces sprzężony

Badanie poprawności dla modelu *MLD* i warunkowej funkcji obserwacji odbywa się z wykorzystaniem procesu sprzężonego. Jego konstrukcja jest podobna do zdefiniowanych wcześniej postaci dla liniowej i niehomomorficznej funkcji obserwacji.

Różnicą jest zwiększona liczba składowych stanu. Stan procesu sprzężonego dla niehomomorficznej funkcji obserwacji, będzie zapisywany jako krotka $(x , x_\gamma , (y , val , y', R))$, gdzie x jest rozwiązaniem składowej *ML* modelu weryfikowanego, x_γ odpowiadającym mu wektorem w przestrzeni X_γ, y i *val* są składowymi stanu modelu *MLD*, y' stanem modelu kryterialnego, natomiast R jest macierzą lokalnej osiągalności opisującej osiągalność w przestrzeni X_γ .

Niech *MLD* = (*ML*, *MD*, η, φ) będzie modelem liniowym rozszerzonym o model danych . Zakładamy, że zbiór stanów osiągalnych modelu *ML* jest skończony. Zakładamy także, że zbiór osiągalnych stanów (wartości zmiennych) modelu danych *MD* jest skończony.

Niech $ML' = (X', x_0', A_I' x' \le b', A_E' x' = 0)$ będzie modelem specyfikacji kryterialnej , Y' jego przestrzenią stanów.

Niech γ będzie funkcją ekspansji : $X_\gamma \times X \times Values \to X_\gamma$. Załóżmy, że *s* jest wymiarem przestrzeni Xγ .Niech π będzie funkcję obserwacji : $X_\gamma \to X'$

Zakładamy, że $\pi(\gamma (0, x_0, val_0)) = x_0'$.

Proces sprzężony postaci $\hat{P} = (\hat{S} , \hat{B} , \hat{F} , \hat{T})$ jest skonstruowany jako granica ciągu procesów $\Theta = \langle P^{(0)},\ldots, P^{(i)},\ldots, P^{(n)},\ldots\rangle$ postaci $P^{(i)} = (S^{(i)}, B^{(i)}, F^{(i)}, T^{(i)})$, gdzie $S^{(i)} \subset X \times X_\gamma \times \mathbf{Y}$; $\mathbf{Y} = Y \times Values \times Y' \times (\mathbf{R}^r)^s$.

Podobnie, jak dla wszystkich procesów sprzężonych spełnione są zależności:

$$\hat{S} = \bigcup_{i=0}^{\infty} S^{(i)}$$

$$\hat{B} = \bigcup_{i=0}^{\infty} B^{(i)}$$

$$\hat{F} = \bigcup_{i=0}^{\infty} F^{(i)}$$

$$\hat{T} = \bigcup_{i=0}^{\infty} T^{(i)}$$

Proces rozpoczynający ciąg opisany jest następującymi równaniami:

1. $S^{(0)} = B^{(0)} = \{ (x_0 , x_{\gamma 0} , (y_0 , val_0 , y_0', R^{(0)}))\}$,
   gdzie $x_{\gamma 0} = \gamma(0, x_0, val_0)$ $y_0 = y(x_0)$, $y_0' = y(\pi(x_{\gamma 0}))$, macierz początkowa określona jest wzorem Def. 6-5 (1)
2. $F^{(0)} = \begin{cases} B^{(0)} & \text{jeżeli } final(x_0) \wedge final(val_0) \\ \varnothing & \text{w p.p.} \end{cases}$
3. $T^{(0)} = \varnothing$

Przejście od $P^{(i-1)}$ do $P^{(i)}$ następuje jeśli:

$\exists\, s_k \in S^{(i-1)} \setminus F^{(i-1)} . s_k = (x_k , x_{\gamma k} , (y_k, val_k, y_k', R^{(k)})) \wedge$

$\wedge \exists\, v \in V_F(y_k) . s_i = (x_i , x_{\gamma i} , (y_i, val_i, y_i', R^{(i)})) \notin S^{(i-1)}$, gdzie:

$x_i = x_k + v$

$x_{\gamma i} = \gamma(x_{\gamma k}, v, val_i)$

$y_i = y(x_k + v)$,

$eval(val_k, \varphi(v)) = (val_i, result) \wedge result \neq (error)$

$y_i' = y(\pi(x_{\gamma i}))$,

$R_i = lrm(R^{(k)}, \Delta x_{\gamma i})$, natomiast $\Delta x_{\gamma i} = \gamma(0, v, val_i)$

Wówczas konstruowany jest nowy proces $P^{(i)}$ postaci:

1. $S^{(i)} = S^{(i-1)} \cup \{ s_i \}$, $s_i = (x_i , x_{\gamma i} , (y_i, val_i, y_i', R^{(i)}))$
2. $B^{(i)} = B^{(i-1)}$
3. $T^{(i)} = T^{(i-1)} \cup \{(s_k, s_i)\}$
4. Zbiór $F^{(i)}$ modyfikowany jest następująco:
   a) Jeśli $V_F(y_i) = \varnothing \wedge \forall\, i \in N\,.\, A_I\, x_i = 0$ to $F^{(i)} = F^{(i-1)} \cup \{ s_i \}$
   b) Jeśli $V_F(y_i) \neq \varnothing$ oraz istnieje stan $s_t \in S^{(i-1)}$, $s_t = (x_t , x_{\gamma i} , (y_t , val_t , y_t' , R^{(t)}))$ spełniający:

      $(y_t = y_i) \wedge equal(val_t, val_i) \wedge (y_t' \leq y_i') \wedge (R^{(t)} = R^{(i)} \vee R^{(t)} \prec R^{(i)})$

      to $F^{(i)} = F^{(i)} \cup \{ s_i \}$
   c) Jeśli $V_F(y_i) \neq \varnothing$ oraz istnieje $s_t \in S^{(i-1)}$, $s_t = (x_t, (y_t , y_t' , R^{(t)}))$ spełniający:

      $(y_t = y_i \vee y_t \prec y_i) \wedge (equal(val_t, val_i) \vee covers(val_i, val_t)) \wedge \neg\, (y_t = y_i \wedge equal(val_t, val_i)) \wedge (y_t' \leq y_i') \wedge (R^{(t)} = R^{(i)} \vee R^{(t)} \prec R^{(i)})$

      to $F^{(i)} = F^{(i)} \cup \{ s_i \}$
   d) Jeżeli nie zachodzi żaden z powyższych przypadków $F^{(i)} = F^{(i-1)}$

■

Warunek 4.c związany jest z wykrywaniem stanów pokrywających w specyfikacji weryfikowanej, a więc nieskończonego zbioru stanów osiągalnych modelu *MLD*. W przypadku, kiedy zostanie on spełniony – weryfikację traktujemy jako częściową.

Zastosowanie procesu sprzężonego w weryfikacji poprawności jest podobne jak w przypadku pozostałych procesów sprzężonych (dla liniowej i niehomomorficznej funkcji obserwacji i weryfikowanego modelu *ML*).

W każdym z przypadków wyznaczenie procesu sprzężonego pociąga za sobą pełny przegląd jego przestrzeni stanów. W stosunku do procesu sprzężonego dla niehomomorficznej funkcji obserwacji zmianie uległa postać składowej stanu modelu weryfikowanego – wektor y opisujący stan modelu *ML* został zastąpiony przez stan modelu *MLD* – parę (y, val). Bezpośrednio obserwowane przejścia w przestrzeni X zostały zastąpione odtwarzanymi przejściami w przestrzeni X_γ (patrz Wniosek 8-1) .

Zauważmy jeszcze, że w przypadku, kiedy funkcja obserwacji jest funkcją homomorficzną postać procesu sprzężonego może ulec znacznemu uproszczeniu. Nie jest konieczne przechowywanie składowej x_γ ponieważ każde przejście w tej przestrzeni jest bezpośrednio odwzorowywane przez funkcję liniową na przejście w X', nie jest konieczne obliczanie macierzy lokalnej osiągalności .

8.7 Implementacja

Implementacja algorytmów do generacji procesu opisującego zachowanie modelu *MLD* i algorytmów weryfikacji jest w stadium prototypowym. Jak już wcześniej wspomniano, nie został zaimplementowany interpreter wyrażeń, lecz do modelowania zmiennych używane są klasy C++ , natomiast wyrażenia (predykaty i operacje) opisywane są za pomocą funkcji.

Na Rys. 8-4 przedstawiono architekturę oprogramowania. Jej struktura została zaprojektowana w celu oddzielenia funkcji modułu weryfikacji od funkcji interpretera.

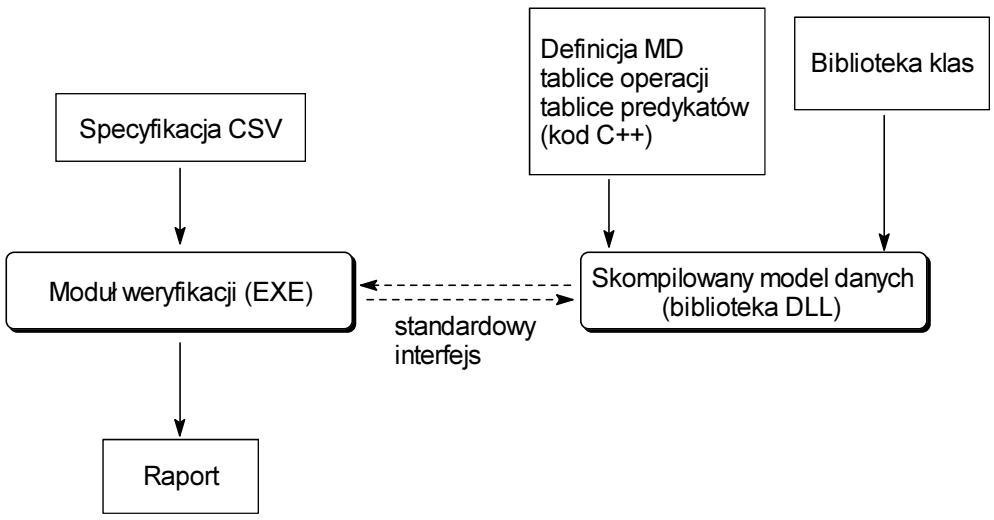
Podobnie, jak w przypadku badania spójności i weryfikacji specyfikacji w postaci modelu liniowego *ML* specyfikacje sporządzane są w formacie CSV programu Excel. Są one wejściem dla modułu weryfikacji.

Definicja modelu danych *MD* jest przygotowywana jako klasa C++ złożona ze standardowych komponentów odpowiadającym wprowadzonym typom danych. Operacje na danych i predykaty są zapisywane jako funkcje określonych typów. Dostęp do nich realizowany jest poprzez tablice wskaźników. Powyższe elementy zapisywane są przez użytkownika w postaci plików C++, następnie po skompilowaniu są łączone z biblioteką standardowych klas w bibliotekę DLL.

Zadaniem biblioteki DLL jest przechowywanie zbioru wartości modelu danych (obiektów klasy), tworzenie kopii obiektów, wykonywanie na nich operacji, testowanie predykatów, porównywanie obiektów, itd. Biblioteka DLL definiuje w tym celu wąski interfejs funkcjonalny.

Każdy obiekt opisujący model danych posiada unikalny identyfikator (ang. *handle*); jedynie on jest używany w module weryfikacji. Wszelkie operacje na danych dokonywane są poprzez wywołanie funkcji biblioteki DLL.

W wyniku wykonania programu sporządzany jest raport z przebiegu weryfikacji.



Rys. 8-4 Architektura oprogramowania

Przygotowanie wszystkich danych niezbędnych do analizy oprogramowania jest niestety czynnością pracochłonną, jak dotąd stworzono jedynie kilka specyfikacji testowych służących do uruchamiania oprogramowania. Sama specyfikacja modelu *MLD* wymaga

podania dwóch macierzy: opisującej składowy model liniowy i macierzy służącej do zdefiniowania odwzorowania φ. Kłopotliwe jest też zapewnienie „spójności syntaktycznej" specyfikacji CSV i kodu funkcji w C++ (zwłaszcza dotyczy to sporządzanych tablic operacji i predykatów). Jest to jednak etap niezbędny przed ewentualnym przystąpieniem do konstrukcji narzędzia specyfikacji graficznej, które generowałoby odpowiednie dane wejściowe.

W przypadku formalizmów pozwalających na zapis operacji na danych, a zwłaszcza używających modelu komunikacji asynchronicznej (kolejek) łatwo jest stworzyć specyfikacje, dla których zbiór stanów osiągalnych jest nieskończony lub tak duży, że jego przetwarzanie przekraczało będzie zasoby sprzętowe.

Typowe praktycznie stosowane algorytmy badania własności modeli obejmujących dane zadowalają się zazwyczaj częściowym przeszukaniem przestrzeni stanów i są optymalizowane ze względu na zużycie pamięci roboczej. Do takich należy opisany w podrozdziale 2.7 algorytm supertrace [Holzmann 88, 91, 95, 97; HP 89].

Wydaje się, że jednym z kierunków dalszego rozwoju tworzonego oprogramowania powinno być zaimplementowanie strategii częściowego przeszukiwania przestrzeni stanów procesu sprzężonego, w tym zastosowanie algorytmu bazującego na konstrukcji supertrace.

8.8 Podsumowanie

W rozdziale:

- zdefiniowano model danych *MD*,
- zdefiniowano model model liniowy rozszerzony o model danych *MLD*,
- zaproponowano funkcję obserwacji zachowania modelu *MLD* (funkcję obserwacji warunkowej),
- wprowadzono definicję poprawności względnej modelu *MLD* dla warunkowej funkcji obserwacji i modelu *ML* służącego jako specyfikacja kryterialna,
- podano konstrukcję procesu sprzężonego jako narzędzia weryfikacji

Niewątpliwą wadą opisanego w tym rozdziale modelu poprawności jest brak symetrii zagadnienia. Weryfikowany proces pozwala na modelowanie danych a specyfikacja kryterialna jest opisana modelem liniowym. Uniemożliwia to zbudowanie zależności pomiędzy wieloma warstwami specyfikacji. Będzie to jednym z kierunków prac w przyszłości. Wydaje się, że jeśli model danych miałby się pojawić w specyfikacji kryterialnej – wówczas powinien być on traktowany jako element wymagań. Konieczne byłoby więc dodanie odwzorowania pomiędzy stanami specyfikacji weryfikowanej i stanami (wartościami danych) specyfikacji kryterialnej. Można się zastanawiać, czy wszystkie wartości danych specyfikacji kryterialnej powinny należeć do przeciwdziedziny tego odwzorowania, czy też jedynie wybrane spośród nich. Tego typu odwzorowania uszczegółowiające (ang. *refinement mapping*) pomiędzy różnymi warstwami specyfikacji opisano w podrozdziale 2.5.1.

9. Zakończenie

Centralnym motywem pracy jest konstrukcja zagadnień poprawności wykorzystujących różne typy funkcji obserwacji do zapisu odwzorowania pomiędzy zachowaniem dwóch modeli specyfikacji: weryfikowanej i kryterialnej. Tak zdefiniowana poprawność ma względny charakter, podobnie jak ma to miejsce dla algebraicznych metod poprawności względnej [Szmuc 89a, 89b, 91, 93, 97a, 97b] czy poprawności wyrażonej jako istnienie relacji symulacji lub uściślania [KMP 94]. Różnicą w stosunku do cytowanych podejść jest wybór akcji, a nie stanu jako pojęcia pierwotnego stosowanego przy specyfikacji wymagań i opisie weryfikowanego systemu.

Zachowanie modelowanych specyfikacji reprezentowane jest przez ciągi wektorów wykonanych akcji. Zadaniem funkcji obserwacji jest przekształcenie ciągu wektorów (wielozbiorów) akcji specyfikacji weryfikowanej w zbiór ciągów wektorów akcji modelu odgrywającego rolę kryterium.

9.1 Model specyfikacji

W pracy posłużono się modelem specyfikacji w postaci zbioru nierówności i równań liniowych (por. rozdział 4.). Jest on wystarczająco ekspresywny, by pozwolić na specyfikację zachowania procesu sekwencyjnego, systemu synchronicznie komunikujących się procesów, a także specyficznych rozszerzeń dla procesu kryterialnego zaproponowanych w pracach poświęconych algebraicznym metodom poprawności względnej. Jak pokazano, model liniowy może zostać przekształcony do macierzowej reprezentacji sieci Petriego.

Bardziej złożony model, wzbogacony o predefiniowane typy danych, zaprezentowany został w rozdziale 8. Pozwala on na włączenie zmiennych do specyfikacji i zawiera ułatwienia dla modelowania mechanizmów komunikacji asynchronicznej (kolejki komunikatów). Może on znaleźć zastosowanie jako model obliczeniowy dla rozszerzonych specyfikacji maszyn skończenie stanowych (por. rozdział 2.2.4).

Przyjęte modele pozwalają na opis rzeczywistych systemów współbieżnych w sposób możliwie abstrakcyjny. Takie podejście bliskie jest metodologii stosowanej w algebraicznych metodach poprawności względnej. W pracy pominięto zagadnienie ich zastosowania jako modeli semantycznych dla konkretnych języków specyfikacji. Pewne możliwości wskazują zamieszczone w pracy przykłady (zwłaszcza specyfikacje systemów w postaci maszyn skończenie stanowych).

Zaletą wprowadzonych modeli jest swoboda wyboru elementu opisu: akcji i stanów. Podobna swoboda cechuje też sieci Petriego. Rozważane modele traktują jednak akcje bardziej atomowo i pozwalają na rozróżnienie sterowania i synchronizacji procesów. Jak można zauważyć, wektory akcji wykorzystywane są przy specyfikacji funkcji obserwacji, natomiast stany przy dowodzeniu poprawności.

Warto zwrócić uwagę, że w odróżnieniu od algebraicznych metod poprawności względnej, gdzie przedmiotem weryfikacji jest proces sekwencyjny opisujący działanie systemu współbieżnych procesów, w prezentowanych rozważaniach przedmiotem weryfikacji może być bezpośredni model systemu zachowujący podział na procesy i rozróżniający ich akcje składowe. (Por. przykłady w rozdziale 7.)

9.2 Funkcje obserwacji

Traktując model kryterialny i weryfikowany jako specyfikacje dwóch kolejnych warstw abstrakcji, definicja funkcji obserwacji może być uznana za formalną realizację możliwości prześledzenia odwzorowania wymagań (ang. *requirements traceability*).

W pracy zdefiniowano relacje poprawności dla trzech typów funkcji obserwacji:

- homomorficznej (liniowej),
- niehomomorficznej,
- funkcji obserwacji warunkowej.

Zadaniem funkcji homomorficznej jest odwzorowanie wybranych (obserwowalnych akcji) specyfikacji weryfikowanej w zbiory akcji specyfikacji kryterialnej. Odwzorowanie to odgrywa podobną rolę, jak operatory ukrywania i zmiany etykiet rozważane w modelu poprawności CSP [Hoare 85] (Por. podrozdział 2.3.1).

Zaproponowana relacja poprawności częściowej dla homomorficznej funkcji obserwacji opiera się na inkluzji dwóch zbiorów ciągów reprezentujących zachowanie modeli: weryfikowanego i kryterialnego. Przy sprawdzaniu inkluzji ciągi reprezentujące działanie specyfikacji weryfikowanej przekształcane są przez funkcje obserwacji.

W przypadku poprawności całkowitej wymagane jest zaobserwowanie wszystkich akcji specyfikacji kryterialnej i brak dywergencji. Ten ostatni warunek zapewnia zgodność pomiędzy nieskończonymi pętlami obu specyfikacji.

Niehomomorficzna funkcja obserwacji pozwala na wyrażenie wymagań odnośnie dekompozycji akcji. Podczas dekompozycji dana akcja specyfikacji kryterialnej może zostać zastąpiona przez proces. W tym przypadku przyjęto odmienne założenia niż w pracy [GG 98] omawiającej uściślanie akcji. (Por. rozdział 2.5.2). Funkcja niehomomorficzna może być traktowana jako specyfikacja rodziny procesów, którymi mogą być zastąpione poszczególne akcje kryterialne. Konkretny sposób realizacji wymagań określony jest w wyniku decyzji projektanta. (Idea ta widoczna jest w rozdziale 7 omawiającym przykłady.)

Niehomomorficzna funkcja obserwacji specyfikuje zbiory akcji, których wykonanie może zostać utożsamione z zaobserwowaniem danej akcji kryterialnej. Przy konstrukcji pojęcia poprawności przyjęto założenie, że atomowość i relacje następstwa akcji specyfikacji kryterialnej muszą znaleźć odwzorowanie w rozłączności czasowej odpowiadających im zbiorów akcji specyfikacji weryfikowanej. Formalnie, opisują to wprowadzone w rozdziale 6 pojęcia odwrotnej dopuszczalności i lokalnej osiągalności. Warto zwrócić uwagę na dodatkowe założenia dotyczące klasy rozważanych modeli: nie jest możliwe etykietowanie tym samym symbolem różnych przejść (lub zdarzeń). Stąd przytoczone za [GG 98] przykłady specyfikacji procesów Q i Q' (Rys 2-3) są niedopuszczalne. Zgodnie z przyjętymi założeniami specyfikacje P i Q oraz ich zmodyfikowane wersje nie mogą zostać uznane za równoważne, ponieważ nie istnieje funkcja obserwacji przeprowadzająca P (P') w Q (Q'). Dla przeciwnego kierunku funkcja taka istnieje (po nadaniu tranzycjom w Q lub Q' unikalnych etykiet).

Funkcja obserwacji warunkowej stosuje się do modelu rozszerzonego o definicję danych. Funkcja ta jest niesymetryczna (nie pozwala na zapis wymagań dotyczących danych). Uzależnia ona zaobserwowane akcje od wartości predykatów określonych na zmiennych. Mówiąc ogólnie, funkcja ta jest złożeniem dwóch odwzorowań: funkcji ekspansji i niehomomorficznej (lub liniowej) funkcji obserwacji. Zadaniem funkcji ekspansji jest

rozszerzenie obserwowanej przestrzeni akcji przez rozróżnienie przejść wykonanych dla różnych wartości danych. Przestrzeń rozszerzona poddawana jest dalszemu przekształceniu przez „zwykłą" funkcję obserwacji.

Wydaje się, że zaproponowane metody specyfikacji poprawności są ekspresywne i wygodne w zastosowaniu, jednakże ostateczną ocenę należy pozostawić czytelnikowi. Autor preferuje je w stosunku do relacji lub odwzorowań w dziedzinie stanów. W pracy przyjęto założenie, że pojecie stanu jest wtórne w stosunku do akcji (zdarzenia, operacji na danych). Jawna reprezentacja stanu, jako wartościowania zmiennych występuje jedynie w modelu rozszerzonym o dane (zdefiniowanym w rozdziale 8). Podobna tendencja wydaje się dominować w najbardziej popularnych obecnie obiektowych metodach specyfikacji, gdzie jawnym elementem interfejsu obiektów są wejściowe zdarzenia (wywołania metod) i atrybuty.

Zaproponowane metody specyfikacji wymagań zdaniem autora są najbliższe praktyce uruchamiania oprogramowania. Polega ono zazwyczaj na śledzeniu wykonania instrukcji, okresowym sprawdzaniu wartości wybranych zmiennych i ustalaniu na tej podstawie, czy badany system zachowuje się zgodnie z wymaganiami. Zasadniczym celem pracy była automatyzacja tego procesu, co pociąga za sobą konieczność formalizacji zapisu wymagań.

Przy określaniu wymagań zawsze istnieje alternatywa pomiędzy specyfikacjami negatywnymi i pozytywnymi (wyrażającymi zachowania niemożliwe albo pożądane). Systemy weryfikacji własności opisanych logiką temporalną często specyfikują własności negatywne [VW 86; CVWY 92; CGL 94; Holzmann 91, 97]. Ich zastosowanie w wieloetapowym cyklu projektowania uzależnione jest od możliwości automatyzacji negacji formuł definiujących system. Jawne określanie wymagań negatywnych (zachowań niemożliwych) wydaje się bardzo kłopotliwe i prowadzi często do przeoczeń. Opis wymagań negatywnych jest też bardziej naturalny dla stanów niż akcji (np.: stosunkowo łatwo jest scharakteryzować stany wzbronione). W pracy, podobnie jak dla algebraicznych metod poprawności względnej, specyfikacja określa pożądane zachowania.

9.3 Weryfikacja poprawności

Formalną konstrukcją służącą weryfikacji poprawności jest proces sprzężony. Opisuje on wspólne wykonanie modelu weryfikowanego i kryterialnego. W zależności od badanego zagadnienia (dla liniowej, niehomomorficznej i warunkowej funkcji obserwacji) podano trzy definicje procesów sprzężonych.

Ich złożoność uzależniona jest od typu funkcji obserwacji. Rosnący stopień komplikacji najłatwiej jest uwidocznić porównując składowe stanu procesów sprzężonych. Do weryfikacji zagadnienia, w którym używana jest liniowa funkcja obserwacji niezbędne jest składowanie informacji o stanach modelu weryfikowanego i kryterialnego. W przypadku funkcji niehomomorficznej składowa stanu rozszerzana jest o tzw. macierz lokalnej osiągalności pełniącą rolę reprezentacji historii dojścia do danego stanu. Dla warunkowej funkcji obserwacji dodatkowym elementem stanu jest wartościowanie danych.

Dla homomorficznej i niehomomorficznej funkcji obserwacji wykazano, że konstrukcja odpowiednich procesów sprzężonych pozwala na weryfikację poprawności przejść ciągów wektorów akcji reprezentujących zachowanie modelu weryfikowanego. Wykazano także skończoność procesów sprzężonych, co jest niezbędne do konstrukcji algorytmów automatycznej weryfikacji wykorzystujących proces sprzężony. Dla funkcji obserwacji warunkowej dowody te zostały pominięte. (Uwzględniając postać funkcji ekspansji i

jednoznaczność interpretacji wyrażeń opisujących predykaty i operacje na danych, byłyby one niewielką modyfikacją poprzednich.)

9.4 Oprogramowanie służące weryfikacji

W dodatkach A, B oraz C podano algorytmy automatycznej weryfikacji poprawności wykorzystujące konstrukcję procesu sprzężonego. W dużym uproszczeniu, algorytmy te przeszukują w głąb przestrzeń stanów procesu sprzężonego wykonując z przeplotem dopuszczalne przejścia modelu weryfikowanego i obliczając przejścia modelu kryterialnego odpowiadające osiągniętym rozwiązaniom.

Oprogramowanie implementujące opisane algorytmy ma charakter prototypowy, niemniej potwierdzona została jego przydatność do analizy różnych specyfikacji. (Wybrane przykłady znaleźć można w rozdziale 7.) W najbardziej zaawansowanym stadium rozwoju jest oprogramowanie służące weryfikacji dla liniowej i niehomomorficznej funkcji obserwacji. W przypadku obserwacji warunkowej jest ono w fazie pierwszych testów.

Użyteczność opracowanego oprogramowania jest obecnie limitowana niewygodną formą wprowadzania specyfikacji (jawna reprezentacja macierzowa przygotowywana jest z wykorzystaniem arkusza kalkulacyjnego). Autor testując różne specyfikacje uzyskał pewną wprawę w manualnym definiowaniu problemów; jednakże zdaje sobie sprawę, że taki sposób wprowadzania danych jest nieakceptowalny dla typowego użytkownika (projektanta, inżyniera oprogramowania). Opracowane i zaimplementowane algorytmy weryfikacji wydają się na tyle efektywne, by celowym było skonstruowanie narzędzia ułatwiającego graficzną specyfikację problemów.

Jak wspomniano w rozdziale 2.7, możliwości zastosowania algorytmów automatycznej analizy ograniczone są zjawiskiem eksplozji stanów. Podane metody przeszukiwania częściowego, w tym algorytm supertrace mogą również zostać wykorzystane w odniesieniu do zaproponowanych modeli.

Warto zwrócić uwagę, że przyjęte w oprogramowaniu reprezentacje danych nie były mocno optymalizowane (poza przyjętą z góry rzadką strukturą wektorów). Nawet dla algorytmów przeszukujących całą przestrzeń stanów możliwa jest dalsza optymalizacja pozwalająca na redukcję zużycia pamięci (a tym samym pokrycie większej liczby stanów). Dla niektórych typów specyfikacji możliwe jest, na przykład, reprezentowanie składowanych stanów procesu sprzężonego w postaci pól bitowych.

9.5 Przyszłe badania

W pracy pominięto problem wstępnej redukcji weryfikowanej specyfikacji, który może istotnie ograniczyć złożoność obliczeniową algorytmów. Idea redukcji może być przedstawiona jako ciąg przekształceń wyjściowego modelu ML_0 i funkcji obserwacji π_0:

$$(ML_0, \pi_0) \rightarrow (ML_1, \pi_1) \rightarrow \ldots \rightarrow (ML_n, \pi_n)$$

Zapisując poprawność modelu ML_i jako $ML_i \models (ML', \pi_i)$, pragnęlibyśmy zapewnić możliwość wnioskowania, że:

$$ML_n \models (ML', \pi_n) \Rightarrow ML_{n-1} \models (ML', \pi_{n-1}) \ \ldots \Rightarrow ML_0 \models (ML', \pi_0) .$$

Najprostszą z możliwych jest redukcja nie zmieniająca definicji funkcji obserwacji. Polega ona na usunięciu z modelu nieobserwowalnych przejść. Ograniczeniem dla tego typu redukcji jest konieczności uwzględnienia przypadków synchronizacji przejść związanych z

komunikacjami. Autor podjął próby opracowania algorytmów redukcji obejmujących także zmianę funkcji obserwacji. Wymagają one jednak dość ograniczających założeń i nie są dostatecznie dobrze opracowane.

Alternatywą dla redukcji wstępnej jest redukcja wykonywana w trakcie weryfikacji. Jest ona przedmiotem zainteresowania wielu ostatnio powstających prac dotyczących algorytmów automatycznej weryfikacji [Peled 94; Schoot 97]. Jak zauważono, algorytmy wyczerpującego przeszukiwania przestrzeni stanów często wykonują zbędne kroki związane z analizą wielu możliwych permutacji równoległych akcji. Opisane w cytowanych pracach modyfikacje zmierzają do ograniczenia zjawiska przeplotu przez unikanie powtórzeń analizowanych permutacji lub wykonywanie akcji równoległych w jednym kroku. Kryterium dopuszczalności takich redukcji jest niezmienniczość weryfikowanych własności.

Dla modeli liniowych nasuwającym się kierunkiem jest badanie reprezentacji zachowania, w którym kolejne wektory akcji różnią się o wartości będące wektorami równolegle wykonanych przejść (semantyka kroków [GG 98]).

Pewnych uzupełnień wymaga analiza poprawności dla niehomomorficznej funkcji obserwacji. W pracy nie zajmowano się złożeniami funkcji obserwacji. Uniemożliwia to stwierdzenie, że zdefiniowana relacja poprawności jest relacją częściowego uporządkowania.

W ogólnym przypadku, nie dowiedziono, że zakładając częściową lub całkowitą poprawność:

$$ML_3 \models (ML_2, \pi_{32}) \text{ oraz } ML_2 \models (ML_1, \pi_{21})$$

można wnioskować, że

$$ML_3 \models (ML_1, \pi_{32} \cdot \pi_{21}).$$

Wnioskowanie takie wydaje się oczywiste dla homomorficznej funkcji obserwacji (ponieważ złożenie funkcji liniowych jest funkcją liniową), jednakże dla funkcji niehomomorficznej wymaga odrębnej analizy. Złożenie niehomomorficznych funkcji obserwacji $\pi_{32} \cdot \pi_{21}$ nie jest funkcją obserwacji. Wydaje się jednak, że na podstawie ich definicji można zaproponować postać funkcji π_{31} zachowującą co najmniej własność częściowej poprawności:

$$ML_3 \models (ML_2, \pi_{32}) \wedge ML_2 \models (ML_1, \pi_{21}) \Rightarrow ML_3 \models (ML_1, \pi_{31}).$$

Interesującym zagadnieniem jest także głębsza analiza problemów częściowego uporządkowania dla macierzy lokalnej osiągalności.

Dalszych prac wymaga zaproponowanie symetrycznej postaci problemu poprawności dla modeli liniowych rozszerzonych o dane i opracowanie dla nich metod weryfikacji. Tak stawiany problem byłby rozszerzeniem klasy specyfikacji kryterialnych korzystnym z punktu widzenia zastosowań w cyklu życia oprogramowania.

Problem czasu i zagadnienia weryfikacji własności czasu rzeczywistego nie były w pracy rozważane. Rozszerzenie analizy poprawności o zagadnienia weryfikacji uwarunkowań czasowych może przebiegać w jednym z następujących kierunków:

- analizy średnich czasów wykonania akcji (i ewentualnie odchyleń tych wartości)
- częściowej weryfikacji modeli, w których czas wykonania akcji podany jest w postaci przedziałów.

Ten ostatni kierunek badany był w pracach [SS 94; SSSS 94; SS 95a, 95b, 97a, 97b].

Możliwość uwzględnienia wątków sterowania i dynamicznej kreacji procesów pozwoli zwiększyć obszar zastosowań zaproponowanych metod. Jak już wspomniano, reprezentacją odrębnych wątków sterowania może być zbiór wektorów rozwiązań. Modelowanie dynamicznej kreacji procesów wydaje się istotne ze względu na wzrastającą popularność obiektowych metod specyfikacji.

Literatura

[Barnes 84] Barnes, J.: *Programming in Ada*, Addison-Wesley, 1984

[Ben-Ari 82] Ben-Ari, M.: *Principles of Concurrent Programming*, Prentice Hall, 1982; tłum. na jęz. polski: *Podstawy programowania współbieżnego*, WNT 1989

[BF 85] Burden, R.L., Faires, J.D. *Numerical Analysis, third edition*, PWS-KENT, Boston 1985

[BH 93] Braek, R., Haugen, O.: *Engineering Real Times Systems*, Prentice Hall, 1993

[BHR 84] Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes, *Journal of the ACM* 31(3), 1984, 560–599

[BK 86] Bergstra, J.A., Klop, J.W.: Algebra of communicating processes, w de Bakker, J.W., Hazewinkel, M., Lenstra, J.K. (red.): *Mathematics and Computer Science*, CWI Monograph 1, North-Holland, Amsterdam, 1986, 89–138

[Boehm 84] Boehm, B.W.: Verifying and Validating Software Requirements and Design Specifications, *IEEE Software* Vol. 1, No. 1, 1984, 75–88

[Booch 94] Booch, G.: *Object-Oriented Analysis and Design with Applications*, Benjamin/Cummings, 1994

[BR 85] Brookes, S.D., Roscoe, A.W.: An improved failures model for communicating processes, w Brookes, S.D., Roscoe, A.W. Winskel, G.: (red.): *Seminar on Concurrency, Lecture Notes in Computer Science* 197, Springer-Verlag, 1985, 281–305

[BRR 90] de Bakker, J.W., de Roever, W.P., Rozenberg, G.: *Proceedings REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*, Mook, The Netherlands, May/June 1989, LNCS 430. Springer Verlag 1990

[BRS 77] Bartol W., Raś Z., Skowron A.: Theory of computing systems, w Mazurkiewicz A., Pawlak Z. (red.): *Mathematical Foundations of Computer Science*, Banach Center Publications, 2, 1977, 101–165

[Calvez 93] Calvez, J-P.: *Embedded Real-Time Systems*, J.Wiley & Sons, 1993

[CGL 94] Clarke, E., Grumberg, O., Long, D.: Verification Tools for Finite-State Concurrent Systems, w de Baker, J.W., de Roever, W.-P., Rozenberg, G., (red.) *A Decade of Concurrency. Reflections and Perspectives*, Springer Verlag 1994

[CVWY 92] Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M., Memory efficient algorithms for the verification of temporal properties, *Formal Methods in System Design*, Vol. I, 1992, 275–288

[CY 1990] Coad, P., Yourdon, E.: *Object-oriented analysis*, Prentice Hall, Inc. 1990

[Deo 74] Deo, N.: *Graph teory with applications to engineering and computer science*, Prentice Hall, Englewood Clifs, 1974; tłum. na jęz. polski: *Teoria grafów i jej zastosowania w technice i informatyce*, PWN 1980

[Dijkstra 68] Dijkstra, E.W., Cooperating Sequential Processes w Genuys, F.: (red.) *Programming Languages*, Academic Press, 1968, 43–112

[Dijkstra 76] Dijkstra, E.W., *A Discipline of Programming*, Prentice Hall, 1976; tłum. na jęz. polski: *Umiejętność Programowania*, PWN 1978

[ELP 93] Elmstrom, R., Lintulampi, R., Pezze, M.: Giving Semantics to SA/RT by means of High-Level Timed Petri Nets, *Real Time Systems*, Vol. 5, Kluwer Academic Publishers, 1993, 249–271

[Floyd 67] Floyd, R.W.: Assigning meanings to programs, *Proc. Symposium on Applied Mathematics*, American Mathematical Society, Vol. 19, 1967, 19–32

[FS 97] Fowler, M., Scott, K., *UML Distilled, Applying Standard Obejct Modelling Language*, Addison-Wesley 1997

[GG 98] van Glabbeek, R.J., Goltz, U.: Refinement of Actions and Equivalence Notions for Concurrent Systems, *Hildesheimer Informatik Bericht 6/98*, Institut für Informatik, Universität Hildesheim, Germany 1998

[GH 93] Godefroid, P., Holzmann, G.J.: On the Verification of Temporal Properties, *Proc. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV93)*, Liege, Belgium, North-Holland, June 1993, 109–124

[Glabbeek 90] van Glabbeek, R.J.: *Comparative Concurrency Semantics and Refinement of Actions*, PhD thesis, Free University, Amsterdam, 1990 Second edition available as CWI tract 109, CWI, Amsterdam 1996

[GMMP 91] Ghezzi, C., Mandrioli, D., Morasca, S., Pezze, M.: A unified high-level Petri net model for tome critical systems, *IEEE Transactionns on Software Engineering*, Vol. 17 (2), 1991, 160–172

[Graham 96] Graham, P.: *The ANSI Common Lisp*, Prentice Hall 1996

[Hack 76] Hack, M.: The equality problem for vector addition system is undecidable, *Theoretical Computer Science*, Vol. 2, 77-95, 1976

[Harel 88] Harel, D.: STATEMATE1: a Working Environment for Development of Complex Reactive Systems, *Proceedings of the Tenth International Conference on Software Engineering*, *IEEE Press*, 1988

[HD 85] Harel D., Pnueli, A., On the development of reactive systems, w Apt, K.R. (red.) *Logics and models of concurrent systems*, Vol. F13 NATO ASI Series, Springer Verlag, 1985, 477–498

[Hoare 85] Hoare, C.A.R.: *Communicating Sequential Processes*, Prentice-Hall International, Englewood Cliffs, 1985

[Hohn 58] Hohn, F.E.: *Elementary matrix algebra*, Macmilain, New York, 1958

[Holzmann 85] Holzmann, G.J.: Tracing protocols, *AT&T Technical Journal*, Vol 64, December 1985, 2413–2434

[Holzmann 88] Holzmann, G.J., An improved reachability analysis technique, *Software Practice and Experience*, Vol. 18, No. 2, 1988, 137–161

[Holzmann 91] Holzmann, G.J.: *Design and Validation of Computer Protocols*, Prentice Hall, 1991

[Holzmann 95] Holzmann, G.J.: An Analysis of Bit-State Hashing, Proc*. IFIP/WG6.1 Symp. Protocol Specification, Testing, and Verification (PSTV95)*, Warsaw, Poland, Chapman & Hall, June 1995, 301–314

[Holzmann 97] Holzmann, G.J.: The Model Checker Spin, *IEEE Trans. on Software Engineering*,Vol. 23, No. 5, May 1997, 279–295.

[HP 89] Holzmann, G.J., Patti, J.: Validating SDL specifications: an experiment, *Proc. 9th Int. Conf on Protocol Specification, Testing, and Verification, INWG/IFIP*, (red.). Vissers, C., Brinksma, E. Twente, Netherlands, June, 1989.

[HP 94] Holzmann, G.J., Peled, D.: An Improvement in Formal Verification, *Proc. Seventh FORTE Conf. Formal Description Techniques*, Bern, Switzerland, Oct. 1994, 177–194

[HS 92] Halang, A.W., Sacha, K.M, *Real-Time Systems – Implementation of Industraial Computerised Process Automation*, World Scientific 1992

[IEEE 83] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 729-1983, Los Alamos, California 1983

[IEEE 86] IEEE/ANSI Std.: *Guide for Software Quality Assurance Planning*, IEEE 1986

[Jensen 91] Jensen, K.: Coloured Petri Nets, A High-level Language for System Design and Analysis. w Rozenberg, G.: (red) *Advances in Petri Nets 1990, Lecture Notes in Computer Scienece*, Vol. 483, Springer Verlag, 1991, 342-416

[Jensen 95-96] Jensen K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Vol. I-III, Springer Verlag, 1995/96

[Klimek 98] Klimek, R.: *Warstwowa weryfikacja systemów współbieżnych z wykorzystaniem logiki temporalnej*, Katedra Automatyki AGH, Kraków 1998, rozprawa doktorska

[Klimek 99] Klimek, R.: *Wprowadzenie do logiki temporalnej*, Wydawnictwa Naukowo-Dydaktyczne AGH, Kraków 1999, (w druku)

[KMP 94] Kesten, Y., Manna, Z. Pnueli, A.: Temporal Verification of Simualation and Refinement w de Baker, J.W., de Roever, W.-P., Rozenberg, G.: (red.) *A Decade of Concurrency. Reflections and Perspectives*, Springer Verlag 1994

[KS 90] Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence, *Information and Computation*, Vol 86, No. 1, 1990, 43–68

[Lakos 94] Lakos, C.A., Object Petri Nets – Definitions and Relationship to Coloured Nets, *Technical Report TR94-3*, Computer Science Department, University of Tasmania, 1994

[Lakos 95] Lakos, C.A., From Coloured Petri Nets to Object Petri Nets, *Proceedings of 15th International Conference on the Application and Theory of Petri Nets*, Torino, LectureNotes In Computer Science, Springer Verlag 1995

[Lamport 77] Lamport, L.: Proving the correctness of multiprocess programs, *IEEE Trans. on Software Engineering*, Vol SE-3, No. 2, 1977, 125–143

[Lamport 83] Lamport, L.: What good is temporal logic?, *Information Processing 83: Proc. of the 9th IFIP World Computer Congress*. Ed. R.E.A. Mason, Elsevier Publ., September 1983, 657–668

[Lamport 94] Lamport, L.: Verification and Specification of Concurrent Programs, w de Baker, J.W., de Roever, W.-P., Rozenberg, G., (red) A *Decade of Concurrency. Reflections and Perspectives*, Springer Verlag 1994

[Lano 95] Lano, K.: *Formal Obejct Oriented Development*, Springer Verlag, London, 1995

[LB-GG 94] Lee, I., Brémond-Grégoire, P., Gerber, R. A Process Algebraic Approcach to the Specification and Analysis of Resource-Bound Real-Time Systems, *Proceedings of the IEEE*, Vol. 82, No. 1, January 1994

[LCL 87] Lin, F.J., Chu, P.M., Liu, M.T.: Protocol verification using reachability analysis, *Computer Communication Review*, Vol. 17, No. 5, 1987, 126–135

[Milner 89] Milner, R.: Communication and Concurrency. Prentice Hall, Englewood Cliffs, 1989

[MMNPNMV 92] Miller, K.W., Morell, L.J., Noonan, R.E., Park, S.K., Nicol, D.M., Murrill, B.W., Voas, J.M.: Estimating the Probability of Failure When Testing Reveals No Failures. *Transactions On Software Engineering* 18(1), 1992, 33–43

[MP 81] Manna, Z., Pnueli, A.: Verification of concurrent programs: The temproral framework – w Bayer. R.S., Moore J.S. (red) *The Correctness Problem in Computer Science – International Lecture Series in Computer Science*, Academic Press, 1981, 215–272

[MP 91] Manna, Z., Pnueli, A.: *The Temporal Logic of Reactive and Concurrent Systems*, Springer Verlag, 1991

[MSSS 94] Maranzana M., Schwarz, J-J, Skubich, J., Szwed, P.: Application objects and modularity in real time graphical modelling, *6th Euromicro Workshop on Real Time Systems*, Vaesteraas. Sweden June 1994

[Murata 89] Murata T.: Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE*, Vol.77, No 4, April 1989

[OH 86] Olderog, E.-R., Hoare, C.A.R.: Specification-oriented semantics for communicating processes, *Acta Informatica*, 23, 1986, 9–66

[Pawlak 69] Pawlak, Z.: Maszyny programowalne, *Algorytmy*, 10, 1969, 5–19

[Peled 94] Peled, D.: Combining Partial Order Reductions with On-the-fly Model Checking, *6th Int. Conf. on Computer Aided Verification*, Stanford, Ca., June 1994

[Perez 90] Perez, J.P.: *Systèmes temps réel – méthodes de spécification et de conception*, Dunod 1990

[Peterson 81] Peterson, J.L.: *Petri Net Theory and the Modelling of Systems*, Englewood Clifs, Prentice Hall, 1981

[Pnueli 77] Pnueli, A.: The temporal logic of programs, *Proc. 18th IEEE Symposium on Foundations of Computer Science*, Providence, R.I., 1977, 46–57

[PostScript 85] Adobe Systems Inc.: *PostScript Language Reference Maual*, Adison Wesley, 1985

[Reisg 85] Reisig W.: *Petri Nets – An Introduction*, EATCS Monographs on Theoretical Computer Science, Volume 4. Springer. 1985, tłum. na jęz. polski: *Sieci Petriego – Wprowadzenie*, WNT 1988

[RS 82] Rockstrom, A., and Saracco, R.: SDL – CCITT Specification and Description Language, *IEEE Trans. on Communications*, Vol. COM-30, No. 6, 1982, 1310–1318

[RS 94] Raju, S.C.V, Shaw A.C. A prototyping Environment for Specifying Executing And Checking Communicating Real-Time State Machines, *Software Practice and Experience*, February 1994, 175–195

[Sacha 95] Sacha K.: Projektowanie oprogramowania systemów wbudowanych, *Prace Naukowe Politechniki Warszawskiej*, Elektronika z.107, Warszawa 1995

[SB 87] Schupp, P., Bernhard, L.w.: *Productive PROLOG Programming*, Prentice Hall, Englewood Clifs, 1987

[Schoot 98] van der Schoot, H.: Partial-order verification in SPIN can be more efficient, *Proc. Third SPIN Workshop – SPIN97*, Twente University, Enschede, The Netherlands, 5 April 1997, http://netlib.bell-labs.com/netlib/spin/ws98/rdevries.ps.gz

[Schwarz 92] Schwarz, J.-J.: Language d'Aide a la Conception d'Applications Multitaches Temps Reel, *APII-AFCET*, Vol 26, no 5, Dunod 1992

[Shaw 92] Shaw, A.C.: Communicating Real-Time State Machines, *IEEE Transactions on Software Engineering*, 1992 Vol. 18, No 9, 805–816

[Shaw 94] Shaw, A.C.: On Scalable State-Based Specifications for Real-Time Systems, Department of Computer Science and Engineering, University of Washington, TR#94-02-03

[SS 91] Szmuc, T., Szwed, P., System weryfikacji poprawności względnej, *Zeszyty naukowe AGH*, Vol. 59, 1991, 35–42

[SS 94] Szmuc, T., Szwed, P.: Towards Automatic Correctness Verification of Concurrent Systems, *Applied Mathematics and Computer Science*, Vol. 4. No. 4, 1994, 643–660

[SS 95a] Szmuc, T., Szwed, P.: Hierarchiczne tworzenie aplikacji czasu rzeczywistego, *Materiały II Ogólnopolskiej Konferencji Systemów Czasu Rzeczywistego*, Szklarska Poręba 1995, Oficyna Wydawnicza Politechniki Wrocławskiej, 178–188,

[SS 95b] Szmuc, T., Szwed, P.: Hierarchical Specification And Verification of Real-Time Applications, *Elektrotechnika*, Vol. 14, No. 4, 1995, 403–415

[SS 96] Szmuc, T., Szwed, P.: Systematyczne rozwijanie programów czasu rzeczywistego, *Materiały III Konferencji Systemów Czasu Rzeczywistego*, Szklarska Poręba 1996, Oficyna Wydawnicza Politechniki Wrocławskiej, 71–80

[SS 97a] Szmuc, T., Szwed, P.: Towards Systematic Development of Real-Time Programs, *Ogólnopolska Konferencja Zastosowań Teorii Systemów*, Zakopane, 1997, Zeszyty naukowe AGH, Automatyka, Tom 1, Z. 1, 1997, 383–393

[SS 97b] Szmuc, T., Szwed, P.: Weryfikacja poprawności aplikacji czasu rzeczywistego opisywanych językiem LACATRE, *Materiały 1 Krajowej Konferencji Metody i Systemy Komputerowe w Badaniach Naukowych i Projektowaniu Inżynierskim*, 25-26 listopad, 1997, 77–84

[SS 97c] Szmuc, T., Szwed, P.: Wspomaganie konstrukcji oprogramowania czasu rzeczywistego z zastosowaniem języka LACATRE, *Materiały 1 Krajowej Konferencji Metody i Systemy Komputerowe w Badaniach Naukowych i Projektowaniu Inżynierskim*, 25-26 listopad, 1997, 515–522

[SSR 89] Saracco, R., Smith, J.R.W., Reed, R.: *Telecommunications Systems Engineering using SDL*, North-Holland Publ., Amsterdam 1989

[SSSM 93] Schwarz, J.-J., Skubich, J.-J., Szwed, P., Maranzana, M.: Real Time Multitasking Design with a Graphical Tool, *First IEEE Workshop on Real Time Applications*, New York May 1993

[SSSS 94] Szmuc, T., Szwed, P., Schwarz, J-J., Skubich, J.: Hierarchical Correctness Verification in Multiphase Real-Time Software Design, *Proceedings of the 19th IFAC/IFIP/IEEE Workshop on Real-Time Programming*, Reichenau, Germany, 22-24 June 1994, Pergamon Press, 87–94; *also in* Control Eng. Practice. Vol. 3, No. 6, 1995, 829-836

[SSSS 96] Szmuc, T., Szwed, P., Schwarz, J-J., Skubich, J.: Hierarchical Development of Reactive Real-Time Programs, *Proceedings of the 21st IFAC/IFIP/IEEE Workshop on Real-Time Programming*, WRTP'96, Brasil, 129–134

[Szmuc 88] Szmuc, T.: Correctness verification of real-time programs, w de la Puente, J.A., Crespo, A.: (red.): *Proceedings of 15th IFAC/IFIP Workshop on Real-Time Programming*, Valencia, 1988, 3–8

[Szmuc 89a] Szmuc, T.: Correctness verification of concurrent systems, w Shriver, B.D.: (red.): *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, vol. II, Software Track, *IEEE Computer Society Press*, 1989, 295–304

[Szmuc 89b] Szmuc, T.: Poprawność współbieżnych systemów oprogramowania, *Zeszyty Naukowe AGH*, *Automatyka*, vol. 46, 1989

[Szmuc 91] Szmuc, T.: Partial and total relative correctness for analysis of real-time systems, *Applied Mathematics and Computer Science*, 1991, Vol. 1, 31–42

[Szmuc 92] Szmuc, T.: Relative Correctness of Real-Time Systems, w Boullart, L., de la Puente, J.A.: (red.): *Proceedings of the IFAC/IFIP International Workshop on Real-Time Programming*, Bruges 23-26 June 1992, Pergamon Press, 173–177.

[Szmuc 93] Szmuc, T.: Relative Correctness Verification of Concurrent Systems, *Scientific Papers of the University of Mining and Metallurgy*, *Automatics*, vol. 64, 1993, 91–106

[Szmuc 97a] Szmuc, T.: Formalna Weryfikacja Poprawności Oprogramowania, *Prace z Automatyki Dedykowane Profesorowi Henrykowi Góreckiemu z Okazji 70-lecia Urodzin*, Wydawnictwa AGH, 1997, 41–48

[Szmuc 97b] Szmuc, T.: Analiza poprawności systemów, *Materiały 1 Krajowej Konferencji Metody i Systemy Komputerowe w Badaniach Naukowych i Projektowaniu Inżynierskim*, 25-26 listopad, 1997, 69-76

[Szwed 94] Szwed, P.: *LACATRE Reference Guide*, Opracowanie Instytutu Automatyki, Akademii Górniczo-Hutniczej, Kraków, 1994

[Telelogic 97] Telelogic AB., *Telelogic Tau 3.2 User's Manual*, Sweden, Spetember 97

[US 94] *MIL-STD-498 – Software Development and Documentation*, Military Standard, U.S Department of Defense, 5 December 1994

[VB 98] Visser, W., Barringer, H.: CTL* Model Checking for SPIN, *Proc. 4th International SPIN Workshop – SPIN98*, Paris 2 November 1998, http://netlib.bell-labs.com/netlib/spin/ws98/p6.ps.gz

[VT 98] de Vries, R., Tretmans, J.: On-the-Fly Conformance Testing Using Spin, *Proc. 4th International SPIN Workshop – SPIN98*, Paris 2 November 1998,
http://netlib.bell-labs.com/netlib/spin/ws97/schoot.ps.Z

[VW 86] Vardi, M.Y., Wolper, P.: An Automata Theoretic Approach to Automatic Program Verification*, Proc. First Symposium on Logic in Computer Science*, June 1986, 322–331

[VW 94] Vardi, M.Y., Wolper, P.: Reasoning about Infinite Computations, *Information and Computation*, Vol. 115 No. 1, 1994, 1–37

[West 86] West, C.H.: Protocol validation by random state exploration, *Proc. 6th IFIP WG 6.1 Int. Workshop on Protocol Specification, Testing, and Verification*, North-Holland Publ., Amsterdam 1986, 233–242.

[West 89] West, C.H.: Protocol validation in complex systems, *Proc. 8th ACM Symposium on Principles of Distributed Computing*, Austin, Texas, August 1989

[Wirth 71] Wirth, N.: Program development by stepwise refinement, *Communications of the ACM*, Vol. 14, No. 4, 1971, 221–227

[WM 85] Ward, P., Mellor, S.: *Structured Development for Real-Time Systems*, Prentice Hall, 1985.

[Yourdon 88] Yourdon, E.: *Modern Structured Analysis*, Prentice Hall, 1988, tłum. na jęz. polski, Współczesna Analiza Strukturalna, WNT 1996

A. Algorytm badania spójności wymagań

Celem przedstawionego tu algorytmu jest badanie spójności modelu poprzez generację procesu sekwencyjnego opisującego jego zachowanie. Omówmy podstawowe komponenty danych występujące w algorytmie, czyli:

- reprezentację analizowanego rozwiązania
- stos
- zbiór osiągniętych stanów modelu
- zbiór akcji modelu osiągalnych w poprawnych ciągach rozwiązań.

A.1 Reprezentacja aktualnie analizowanego rozwiązanie

Zdefiniujmy obiekt opisujący analizowane rozwiązanie jako:

$es = (x, V_{\text{Next}}, y,)$, gdzie

- x – jest pewnym rozwiązaniem,
- $V_{\text{Next}} \subset V_{\text{F}}(x)$ – jest pewnym podzbiorem zbioru rozwiązań dopuszczalnych w rozwiązaniu x ,
- $y = \text{y}(x)$ – jest stanem modelu *ML* odpowiadającym rozwiązaniu x .

A.2 Stos

W algorytmie do reprezentacji analizowanego ciągu rozwiązań używa się stosu zawierającego obiekty typu *es* . Zakładamy, że zdefiniowane dwie operacje: *push* umieszczająca obiekt *es* na stosie i *pop* usuwająca obiekt ze stosu. Stąd pisali będziemy: *push*(*Stack*, *es*) oraz *es* = *pop*(*Stack*).

Oznaczmy przez *Z*(*Stack*) zbiór elementów umieszczonych na stosie (należących do ciągu) oraz przez *top*(*Stack*) ostatni element.

Dalej będziemy się posługiwać następującą notacją: jeśli obiekt *comp* zdefiniowany jest jako obiekt złożony (krotka) postaci $comp = (t_1,\ldots, t_{\text{i}}, t_{\text{k}})$, wówczas symbol $comp.t_{\text{i}}$ oznaczał będzie jego składową t_{i} .

Zdefiniujmy predykat *isLastStateBranching*, który przyjmuje wartość prawdy, jeśli ciąg kolejnych rozwiązań reprezentowanych przez stos zawiera pętlę:

$$\textit{isLastStateBranching}(\textit{Stack}) \Leftrightarrow es_{\text{top}} = \textit{top}(\textit{Stack}) \wedge \exists\, es \in Z(\textit{Stack}) \setminus \{ es_{\text{top}} \} \,.\, es.y = es_{\text{top}}.y$$

Podobnie, zdefiniujmy predykat *isLastStateCovering* , który przyjmuje wartość prawdy, jeśli ciąg kolejnych rozwiązań prowadzi do stanu pokrywającego:

$$\textit{isLastStateCovering}(\textit{Stack}) \Leftrightarrow es_{\text{top}} = \textit{top}(\textit{Stack}) \wedge \exists\, es \in Z(\textit{Stack}) \setminus \{ es_{\text{top}} \} \,.\, es.y \prec es_{\text{top}}.y$$

A.3 Zbiór osiągniętych stanów modelu

Trzecim obiektem występującym w algorytmie jest zbiór osiągniętych (przeanalizowanych) stanów S modelu ML. Początkowo jest on zbiorem pustym; wyznaczane w kolejnych iteracjach stany y będą do niego dodawane. W przypadku, kiedy zbiór stanów osiągalnych modelu Y_R jest skończony, wówczas po zakończeniu działania algorytmu zachodzi $S = Y_R$; w przeciwnym razie zbiór S będzie jedynie pewnym podzbiorem Y_R.

A.4 Zbiory wykonanych akcji (przejść) modelu

Celem algorytmu jest badanie spójności modelu. Dla stwierdzenia częściowej spójności, konieczne jest przechowywanie informacji o akcjach wykonanych w poprawnych ciągach rozwiązań, czyli ciągach zawierających pętle lub skończonych ciągach, dla których nie wystąpiło blokowanie. Zbiór akcji osiągalnych w poprawnych ciągach rozwiązań przechowywany jest w postaci wektora $x_{corr} \in X$ przyjmującego wartość początkową x_0.

Poza akcjami osiągalnymi w poprawnych rozwiązaniach zbierane są informacje o wykonanych akcjach (niezależnie od poprawności ciągów, w skład których wchodzą). Akumulowane są one w postaci wektora $x_{reach} \in X$.

A.5 Opis algorytmu

Dany jest model liniowy $ML = (X, A_I\, x \le b, A_E\, x = 0)$ oraz rozwiązanie początkowe x_0 .

Faza I (Przygotowawcza)

Wyznaczany jest zbiór przejść elementarnych V zgodnie z procedurą opisaną w podrozdziale 4.7.2.

Faza II (Właściwa wykonania)

1. Podstaw $es_0.x = x_0$,
2. Oblicz wartości $es_0.y$ oraz $es_0.V_{Next} = V_F(\, es_0.x\,)$
3. Jeśli $V_F(\, es_0.x\,) = \varnothing$ sprawdź, czy spełnione są predykaty $final(\, es_0.x\,)$ albo $deadlock(\, es_0.x\,)$; zakończ: *STOP*.
4. Umieść es_0 na stosie: $push(Stack, es_0)$ i dodaj $es_0.y$ do zbioru S.
5. Jeśli stos jest pusty – *STOP*; w przeciwnym przypadku podstaw $es_{top} := top(\, Stack\,)$
6. Jeśli $es_{top}.V_{Next} = \varnothing$, wykonaj $pop(Stack)$ i przejdź do 5.
7. Wybierz dowolne przejście elementarne $v \in es_{top}.V_{Next}$;

   podstaw $es_{top}.V_{Next} := es_{top}.V_{Next} \setminus \{\, v\,\}$
8. Utwórz nową krotkę es i podstaw $es.x = es_{top}.x + v$; Oblicz wartości $es.y$ oraz $es.V_{Next} = V_F(\, es.x\,)$ oraz postaw $x_{reach} := x_{reach} + es.x$.
9. Jeżeli $V_F(\, es.x\,) = \varnothing$
   a) sprawdź, czy spełniony jest predykat $final(\, es.x\,)$.
      Jeśli tak, podstaw: $x_{corr} := x_{corr} + es.x$. (Ścieżka jest poprawna i jej akcje są dodawane do zbioru reprezentowanego przez x_{corr})
      W przeciwnym przypadku spełnione jest $deadlock(\, es.x\,)$;
   b) jeżeli $es.y \notin S$, dodaj $es.y$ do zbioru S ;

c) przejdź do (5).

10. Umieść *es* na stosie: *push*(*Stack*, *es*)
11. Jeśli spełniony jest predykat *isLastStateBranching*(*Stack*) lub *isLastStateCovering*(*Stack*)
    a) podstaw: $x_{corr} := x_{corr} + es.x$;
    b) usuń element ze stosu *pop*(*Stack*) ;
    c) przejdź do (5).
12. Jeśli stan *es.y* był już wcześniej analizowany, czyli $es.y \in S$, usuń element ze stosu: *pop*(*Stack*).
    W przeciwnym przypadku dodaj *es.y* do zbioru *S*.
13. Przejdź do (5)

Faza III (Analiza rezultatów)

1. Sprawdź, czy x_{corr} zawiera elementy zerowe; jeśli tak, raportuj brak spójności.
2. Jeżeli wystąpiły rozwiązania prowadzące do stanów blokowania i wszystkie elementy x_{corr} są niezerowe raportuj częściową spójność.
3. W przypadku kiedy wszystkie elementy x_{corr} są niezerowe i nie wystąpiło blokowanie – raportuj całkowitą spójność.

Przedstawiony algorytm jest porównywalny z algorytmami badania własności sieci Petriego opartymi na konstrukcji drzewa stanów osiągalnych [Reisg 85; Murata 89].

Pozwala on na stwierdzenie:

- braku blokowania
- bezpieczeństwa (brak stanów pokrywających)
- żywotności (wszystkie elementy x_{reach} powinny być niezerowe)

Został on rozwinięty jako algorytm pomocniczy w stosunku algorytmów badania poprawności względnej.

B. Algorytm weryfikacji dla homomorficznej funkcji obserwacji

Algorytm weryfikacji dla homomorficznej funkcji obserwacji jest w konstrukcji podobny do algorytmu badania spójności. Jego podstawowe elementy danych to:

- reprezentacja aktualnie analizowanego rozwiązania,
- stos,
- zbiór osiągniętych stanów procesu sprzężonego,
- zbiór stanów modelu weryfikowanego, z których osiągalne są stany końcowe,
- informacje o zbiorach osiągalnych akcji w obu modelach.

B.1 Reprezentacja aktualnie analizowanego rozwiązania

Zdefiniujmy obiekt opisujący analizowane rozwiązanie jako:

$vsh = (x, V_{\text{Next}} , x' , y , y')$, gdzie

x jest pewnym rozwiązaniem, $x \in$ X,

$V_{\text{Next}} \subset V_{\text{F}}(x)$ jest pewnym podzbiorem zbioru rozwiązań dopuszczalnych w stanie x ,

x' jest obrazem rozwiązania x poprzez funkcję obserwacji $x' = \pi(x)$

$y =$ y(x) jest stanem modelu *ML* odpowiadającym rozwiązaniu x ,

$y' =$ y(x') jest stanem modelu *ML*' odpowiadającym rozwiązaniu x' .

B.2 Stos

W algorytmie do reprezentacji analizowanego ciągu rozwiązań używa się stosu zawierającego obiekty typu *vsh*. Podobnie, jak w przypadku algorytmu analizy spójności wymagań zdefiniowane są operacje *pop*, *push* i *top*.

Definicje predykatów *isLastStateBranching* oraz *isLastStateCovering* odzwierciedlają zmienioną postać składowych stanu.

isLastStateBranching(*Stack*) $\Leftrightarrow$

$vsh_{\text{top}} = top(Stack) \wedge$

$\exists\ es \in Z(Stack) \setminus \{ vsh_{\text{top}} \} .\ vsh.y = vsh_{\text{top}}.y \wedge vsh.y' \le vsh_{\text{top}}.y')$

isLastStateCovering(*Stack*) $\Leftrightarrow$

$vsh_{\text{top}} = top(Stack) \wedge$

$\exists\ vsh \in Z(Stack) \setminus \{ vsh_{\text{top}} \} .\ vsh.y \prec vsh_{\text{top}}.y \wedge vsh.y' \le vsh_{\text{top}}.y'$

Stos w omawianym algorytmie reprezentuje zarówno ciąg rozwiązań modelu weryfikowanego, jak i kryterialnego. Zdefiniujmy predykat *isLastStateDivergent*(*Stack*), który przyjmuje wartość prawdy, jeśli ciąg rozwiązań zawiera dywergencję. W tym celu załóżmy, że stos zawiera *count* elementów i jego poszczególne elementy są ponumerowane od

1 do *count*. Oznaczmy przez *vsh*(*i*) jego *i*–ty element. Zakładamy, że *vsh*(*count*) = *top*(*Stack*).

isLastStateDivergent(*Stack*) $\Leftrightarrow$

$vsh_{top} = top(\ Stack\ ) \wedge$

$\exists\ i < count\ .\ (\ vsh(\ i\ ).y = es_{top}.y \vee vsh(\ i\ ).y \prec es_{top}.y\ )$

$\forall\ j \geq i\ .\ vsh(\ i\ ).x' = vsh_{top}.x'$

B.3 Zbiór osiągniętych stanów procesu sprzężonego

Zbiór osiągniętych stanów procesu sprzężonego *S* zawiera pary postaci (*y* , *y*'), gdzie *y* jest stanem modelu *ML*, natomiast *y* stanem modelu *ML'*. Zapis elementu *y*' jest konieczny, ponieważ w zależności od ścieżki dojścia do stanu modelu weryfikowanego *y* , możemy obliczyć różne wartości rozwiązań modelu kryterialnego, a zatem otrzymać wartości *y*' , różniące się zbiorem dopuszczalnych przejść.

B.4 Zbiór stanów modelu weryfikowanego, z których osiągalne są stany końcowe

Ze względu na konieczność badania sprowadzalności do stanu końcowego, stany modelu weryfikowanego, z których możliwe jest osiągniecie stanu końcowego gromadzone są w zbiorze Y_{fr} (początkowo pustym). W rzeczywistej implementacji zbiory *S* i Y_{fr} obsługiwane są przez jedną strukturę.

B.5 Zbiory osiągalnych akcji

Podobnie, jak dla analizy spójności przechowywane są informacje o wykonanych akcjach modelu weryfikowanego *ML* w postaci wektorów $x_{corr} \in X$ oraz $x_{reach} \in X$. Pierwszy z nich akumuluje informacje o akcjach, które były wykonane w poprawnych ciągach rozwiązań, drugi o osiągalnych akcjach.

Dodatkowo (ze względu na warunki poprawności całkowitej) zbierane są informacje o obserwowalnych akcjach modelu kryterialnego *ML*' w postaci wektora $x_{obs}' \in X'$ oraz o akcjach modelu kryterialnego zaobserwowanych w poprawnych ciągach rozwiązań $x_{corr}' \in X'$.

B.6 Opis algorytmu

Dane są modele specyfikacji weryfikowanej i kryterialnej:

$ML = (X, x_0, A_I\ x \leq b, A_E\ x = 0)$

$ML' = (X', x_0', A_I'\ x' \leq b', A_E'\ x' = 0)$

Dana jest liniowa funkcja obserwacji $\pi : X \rightarrow X'$ określona przez macierz L ($\pi = \pi_{Lin}$)

Zakładamy, że $\pi(x_0) = x_0'$

Faza I (Przygotowawcza)

Wyznaczany jest zbiór przejść elementarnych V modelu ML, zgodnie z procedurą opisaną w podrozdziale 4.7.2.

Analogicznie, wyznaczany jest zbiór przejść elementarnych V' modelu ML'.

Faza II (Właściwa wykonania)

1. Podstaw $vsh_0.x = x_0$,
2. Oblicz wartości $vsh_0.V_{Next} = V_F(vsh_0.x)$, $vsh_0.y$, $vsh_0.x' = \pi(vsh_0.x)$ oraz $vsh_0.y'$. Sprawdź, czy $vsh_0.x'$ jest dopuszczalnym rozwiązaniem modelu ML', czyli czy $vsh_0.y' \geq 0$.
3. Jeśli $V_F(vsh_0.x) = \varnothing$:
   a) sprawdź czy spełniony jest predykaty *final*($vsh_0.x$) ;
      jeśli tak, sprawdź czy zachodzi także *final*($vsh_0.x'$) ;
      w przeciwnym przypadku napotkano blokowanie w pierwszym rozwiązaniu *deadlock*($vsh_0.x$);
   b) zakończ: *STOP*.
4. Umieść vsh_0 na stosie: *push*(*Stack*, vsh_0), dodaj parę ($vsh_0.y$, $vsh_0.y'$) do zbioru S.
5. Jeśli stos jest pusty – *STOP*;
   w przeciwnym przypadku podstaw $vsh_{top} := top(Stack)$.
6. Jeśli $vsh_{top}.V_{Next} = \varnothing$, wykonaj *pop*(*Stack*) i przejdź do 5.
7. Wybierz dowolne przejście elementarne $v \in vsh_{top}.V_{Next}$;
   podstaw $vsh_{top}.V_{Next} := vsh_{top}.V_{Next} \setminus \{ v \}$
8. Utwórz nową krotkę vsh i podstaw $vsh.x = vsh_{top}.x + v$;
   a) oblicz wartości: $vsh.V_{Next} = V_F(vsh.x)$, $vsh.x' = \pi(vsh.x)$, $vsh.y$ oraz $vsh.y'$
   b) podstaw $x_{reach} := x_{reach} + vsh.x$ oraz $x_{obs}' := x_{obs}' + vsh.x'$
9. Jeżeli zaobserwowano zmianę wartości funkcji obserwacji, czyli $v' = vsh.x' - vsh_{top}.x \neq 0$, sprawdź poprawność przejścia, czyli czy v' jest dopuszczalnym przejściem równoległym w stanie $vsh.y'$ (Patrz Def. 4-9).
   W przypadku błędu przejdź do (5)
10. Jeżeli $V_F(vsh.x) = \varnothing$
    a) sprawdź, czy spełniony jest predykat *final*($vsh.x$) .
       Jeśli tak, to:
       i) podstaw: $x_{corr} := x_{corr} + vsh.x$ oraz $x_{corr}' := x_{corr}' + vsh.x'$
       ii) dla wszystkich elementów $vsh \in Z(Stack)$ dodaj $vsh.y$ do Y_{fr}
       W przeciwnym przypadku spełnione jest *deadlock*($vsh.x$);
    b) jeżeli ($vsh.y$, $vsh.y'$) $\notin S$, dodaj parę ($vsh.y$, $vsh.y'$) do zbioru S ;
    c) przejdź do (5).
11. Umieść vsh na stosie: *push*(*Stack*, vsh)

12. Jeśli spełniony jest predykat *isLastStateBranching*(*Stack*) lub *isLastStateCovering*(*Stack*) :
    a) sprawdź, czy zachodzi przypadek dywergencji, czyli czy spełnione jest *isLastStateDivergent*(*Stack*)
    b) podstaw: $x_{corr} := x_{corr} + vsh.x$ oraz $x_{corr}' := x_{corr}' + vsh.x'$;
    c) usuń element ze stosu *pop*(*Stack*)
    d) przejdź do (5)
13. Jeśli para ($vsh.y$, $vsh.y'$) była już wcześniej analizowana, czyli ($vsh.y$, $vsh.y'$) $\in S$ lub $\exists\, s \in S\,.\, (\, s.\, y = vsh.y \wedge s.\, y' \leq vsh.y'\,)$, usuń element ze stosu *pop*(*Stack*). W przeciwnym przypadku dodaj parę ($vsh.y$, $vsh.y'$) do zbioru *S*.
14. Przejdź do (5)

Faza III (Analiza rezultatów)

1. Poprawność całkowita jest raportowana w przypadku, kiedy:
    a) specyfikacja weryfikowana jest wolna od blokowania,
    b) wszystkie przejścia są poprawne
    c) $\forall\, y \in Y_{fr}\ \forall\, s_1\,,\, s_2 \in S\ .y = s_1.y = s_2.y\ .\neg\ (s_1.y' < s_2.y' \vee s_2.y' < s_1.y')$
    d) nie stwierdzono dywergencji
    e) wszystkie akcje modelu kryterialnego zostały zaobserwowane w poprawnych ciągach rozwiązań, czyli wektor x_{corr}' ma wszystkie składowe niezerowe.
2. Poprawność częściowa jest raportowana, kiedy spełnione są warunki 1.a,b,c.
3. Poprawność potencjalna jest raportowana w przypadku, kiedy wszystkie akcje modelu kryterialnego zostały zaobserwowane w poprawnych ciągach rozwiązań
4. Podobnie, jak dla algorytmu badania spójności, raportowane są akcje modelu ML osiągalne w poprawnych ciągach rozwiązań; jest to informacja dodatkowa
5. Pozytywny werdykt algorytmu weryfikacji uznawany jest za rozstrzygający, jeśli nie napotkano stanów pokrywających dla modelu weryfikowanego oraz nie napotkano stanów modelu weryfikowanego, z których osiągalne są stany końcowe i którym odpowiadają pokrywające stany modelu kryterialnego.

C. Algorytm weryfikacji dla niehomomorficznej funkcji obserwacji

Algorytm weryfikacji dla niehomomorficznej funkcji obserwacji jest w konstrukcji podobny do algorytmu badania opracowanego dla funkcji liniowej. Podstawowe przetwarzane dane to:

- reprezentacja aktualnie analizowanego rozwiązania,
- stos,
- zbiór osiągniętych stanów procesu sprzężonego,
- zbiór stanów modelu weryfikowanego, z których osiągalne są stany końcowe,
- informacje o zbiorach osiągalnych akcji w obu modelach.

C.1 Reprezentacja aktualnie analizowanego rozwiązania

Zdefiniujmy obiekt opisujący analizowane rozwiązanie jako:

$vsnh = (x, V_{\mathrm{Next}} , x' , x_{\mathrm{re}} , y , y' , R)$, gdzie

x – jest pewnym rozwiązaniem, $x \in \mathrm{X}$,

$V_{\mathrm{Next}} \subset V_{\mathrm{F}}(x)$ – jest pewnym podzbiorem zbioru rozwiązań dopuszczalnych dla rozwiązania x ;

x' – jest obrazem rozwiązania x poprzez funkcję obserwacji $x' = \pi(x)$;

x_{reven} – jest wektorem reprezentującym zbiór akcji odwrotnie dopuszczalnych; zbiór ten jest funkcją stanu y' modelu ML' ;

$y = \mathrm{y}(x)$ – jest stanem modelu ML odpowiadającym rozwiązaniu x ;

$y' = \mathrm{y}(x')$ – jest stanem modelu ML' odpowiadającym rozwiązaniu x' ;

R – jest macierzą lokalnej osiągalności.

C.2 Stos

W algorytmie do reprezentacji analizowanego ciągu rozwiązań używa się stosu zawierającego obiekty typu *vsnh* . Podobnie, jak w przypadku algorytmów analizy spójności wymagań i badania poprawności dla homomorficznej funkcji obserwacji zdefiniowane są operacje *pop*, *push* i *top*.

Definicje predykatów *isLastStateBranching* oraz *isLastStateCovering* odzwierciedlają zmienioną postać składowych stanu, natomiast predykat *isLastStateDivergent* pozostaje praktycznie bez zmian. (Zakładamy, że stos zawiera *count* elementów i jego poszczególne elementy są ponumerowane od 1 do *count*. Przez $vsnh(i)$ oznaczmy jego i–ty element; spełnione jest $vsnh(count) = top(Stack)$.)

$isLastStateBranching(Stack) \Leftrightarrow$

$$vsnh_{\mathrm{top}} = top(Stack) \wedge$$

$$\exists\, es \in Z(Stack) \setminus \{ vsnh_{\mathrm{top}} \} \,.\, vsnh.y = vsnh_{\mathrm{top}}.y \wedge vsnh.y' \le vsnh_{\mathrm{top}}.y' \wedge$$
$$\wedge (vsnh.R = vsnh_{\mathrm{top}}.R \vee vsnh.R \prec vsnh_{\mathrm{top}}.R)$$

$isLastStateCovering(\ Stack\ ) \Leftrightarrow$

$$vsnh_{top} = top(\ Stack\ ) \wedge$$

$$\exists\ vsnh \in Z(\ Stack\ ) \setminus \{\ vsnh_{top}\ \}\ .\ vsnh.y \prec vsnh_{top}.y \wedge vsnh.y' \leq vsnh_{top}.y'$$
$$\wedge\ (\ vsnh.R = vsnh_{top}.R \vee vsnh.R \prec vsnh_{top}.R\ )$$

$isLastStateDivergent(\ Stack\ ) \Leftrightarrow$

$$vsnh_{top} = top(\ Stack\ ) \wedge$$

$$\exists\ i < count\ .\ (\ vsnh(\ i\ ).y = vsnh_{top}.y\ )$$

$$\forall\ j \geq i\ .\ vsnh(\ i\ ).x' = vsnh_{top}.x'$$

Predykat *isDeterministicTrace* służy do sprawdzenia, czy ścieżka reprezentowana przez stos jest ścieżką deterministyczną. Używany jest on tylko w przypadku, kiedy algorytm służy do „zgrubnego" sprawdzenia specyfikacji w celu znalezienia akcji modelu kryterialnego, które nie są osiągalne w deterministycznych ciągach rozwiązań. W takim przypadku ścieżki niedeterministyczne są odcinane.

$isDeterministicTrace(\ Stack\ ) \Leftrightarrow$

$$\forall\ i < count\ .\ \neg\ \exists\ j < count\ .$$

$$vsnh(\ i\ ).y = vsnh(\ j\ ).y \wedge vsnh(\ i{+}1\ ).x - vsnh(\ i\ ).x\ \neq\ vsnh(\ j{+}1\ ).x - vsnh(\ j\ ).x$$

C.3 Zbiór osiągniętych stanów procesu sprzężonego

Zbiór osiągniętych stanów procesu sprzężonego S zawiera trójki postaci (y , y', R), gdzie y jest stanem modelu ML, y' stanem modelu ML', natomiast R macierzą lokalnej osiągalności.

W stosunku do algorytmu weryfikacji dla homomorficznej funkcji obserwacji, postać przechowywanych stanów uległa rozszerzeniu o składnik R .

C.4 Zbiór stanów modelu weryfikowanego, z których osiągalne są stany końcowe

Ze względu na konieczność badania sprowadzalności do stanu końcowego, stany modelu weryfikowanego, z których możliwe jest osiągniecie stanu końcowego gromadzone są w zbiorze Y_{fr} (początkowo pustym). W rzeczywistej implementacji zbiory S i Y_{fr} obsługiwane są przez jedną strukturę.

C.5 Zbiory osiągalnych akcji

Analogicznie, jak dla algorytmu badania poprawności dla homomorficznej funkcji obserwacji, zbierane są informacje o zbiorach osiągalnych i obserwowalnych akcji w obu modelach ML i ML'. Zbiory te dla uproszczenia gromadzone są w postaci wektorów.

$x_{corr} \in X$ zbiór akcji modelu weryfikowanego osiągalnych w poprawnych ciągach rozwiązań

$x_{reach} \in X$ zbiór wszystkich osiągalnych akcji modelu weryfikowanego

$x_{corr}' \in X'$ zbiór akcji modelu kryterialnego obserwowalnych w poprawnych ciągach rozwiązań

$x_{obs}' \in X'$ zbiór wszystkich akcji obserwowalnych modelu kryterialnego

$p_{obs} \in P$ zbiór obserwowalnych akcji w przestrzeni liniowej obserwacji

C.6 Opis algorytmu

Dane są modele specyfikacji weryfikowanej i kryterialnej:

$$ML = (X, x_0, A_I\, x \le b, A_E\, x = 0)$$

$$ML' = (X', x_0', A_I'\, x' \le b', A_E'\, x' = 0)$$

Dana jest niehomomorficzna funkcja obserwacji

$$\pi : X \to X',$$

której składnik liniowy π_{Lin} określony jest przez macierz L, natomiast składnik nieliniowy π_{Min} określony jest przez macierz M.

Zakładamy, że $\pi(x_0) = x_0'$

Faza I (Przygotowawcza)

Wyznaczany jest zbiór przejść elementarnych *V* modelu *ML*, zgodnie z procedurą opisaną w (4.7.2)

Wyznaczany jest zbiór przejść elementarnych *V'* modelu *ML'*.

Faza II (Właściwa wykonania)

1. Podstaw $vsnh_0.x = x_0$,
2. Oblicz wartości $vsnh_0.V_{Next} = V_F(\, vsnh_0.x\,)$, $vsnh_0.y$, $vsnh_0.x' = \pi(\, vsnh_0.x\,)$, $vsnh_0.y'$, $vsnh_0.x_{reven}$ oraz $vsnh_0.R$. (Początkowa macierz lokalnej osiągalności obliczana jest zgodnie z Def. 6-5).
3. Sprawdź, czy $vsnh_0.x'$ jest dopuszczalnym rozwiązaniem modelu *ML'*, czyli czy $vsnh_0.y' \ge 0$.
4. Jeśli $V_F(\, vsnh_0.x\,) = \varnothing$:
   a) sprawdź czy spełniony jest predykaty *final*($vsnh_0.x$) ;
   jeśli tak, sprawdź czy zachodzi także *final*($vsnh_0.x'$) ;
   w przeciwnym przypadku napotkano blokowanie w pierwszym rozwiązaniu *deadlock*($vsnh_0.x$);
   b) zakończ: *STOP*.
5. Umieść $vsnh_0$ na stosie: *push*(*Stack*, $vsnh_0$), dodaj trójkę ($vsnh_0.y$, $vsnh_0.y'$, $vsnh_0.R$) do zbioru *S*.
6. Jeśli stos jest pusty – *STOP*;
   w przeciwnym przypadku podstaw $vsnh_{top}$:= *top*(*Stack*) .

7. Jeśli $vsnh_{top}.V_{Next} = \varnothing$, wykonaj *pop*(*Stack*) i przejdź do 5.
8. Wybierz dowolne przejście elementarne $v \in vsnh_{top}.V_{Next}$;
   podstaw $vsnh_{top}.V_{Next} := vsnh_{top}.V_{Next} \setminus \{ v \}$
9. Utwórz nową krotkę *vsnh* i podstaw $vsnh.x = vsnh_{top}.x + v$;
   a) oblicz wartości: $vsnh.V_{Next} = V_F(vsnh.x)$, $vsnh.x' = \pi(vsnh.x)$, $vsnh.y$, $vsnh.y'$
   b) wyznacz nową wartość macierzy lokalnej obserwacji:
      $vsnh.R = lrm(vsnh_{top}.R , v)$
   c) jeżeli $v' = vsnh.x' - vsnh_{top}.x \neq 0$ wyznacz nowy zbiór $vsnh.x_{reven}$ na podstawie zbioru przejść dopuszczalnych modelu $ML' V_F'(vsnh.y')$;
      w przeciwnym przypadku podstaw $vsnh.x_{reven} := vsnh_{top}.x_{reven}$
   d) podstaw $x_{reach} := x_{reach} + vsnh.x$ oraz $x_{obs}' := x_{obs}' + vsnh.x'$
   e) podstaw $p_{obs} := p_{obs} + \pi_{Min}(vsnh.x)$
10. Sprawdź poprawność przejścia:
    a) sprawdź, czy wykonane przejście jest odwrotnie dopuszczalne, czyli czy zachodzi $v \leq vsnh_{top}.x_{reven}$. W przypadku błędu przejdź do (5)
    b) Jeżeli zaobserwowano zmianę wartości funkcji obserwacji, czyli $v' = vsnh.x' - vsnh_{top}.x \neq 0$:
       i) sprawdź poprawność przejścia dla modelu kryterialnego, czyli czy v' jest dopuszczalnym przejściem równoległym w stanie $vsnh.y'$
       ii) sprawdź warunek lokalnej osiągalności: $v' = ltrans(vsnh_{top}.R , v)$
       iii) w przypadku błędu przejdź do (5)
11. Jeżeli $V_F(vsnh.x) = \varnothing$
    a) sprawdź, czy spełniony jest predykat *final*($vsnh.x$) . Jeśli tak, to:
       i) podstaw: $x_{corr} := x_{corr} + vsnh.x$ oraz $x_{corr}' := x_{corr}' + vsnh.x'$
       ii) dla wszystkich elementów $vsnh \in Z(Stack)$ dodaj $vsnh.y$ do Y_{fr}

       W przeciwnym przypadku spełnione jest *deadlock*($vsnh.x$);
    b) jeżeli trójka $s =(vsnh.y , vsnh.y , vsnh.R) \notin S$, dodaj s do zbioru S ;
    c) przejdź do (5).
12. Umieść *vsnh* na stosie: *push*(*Stack*, *vsnh*)
13. Opcjonalnie: (tylko w przypadku badania deterministycznej zgodności)
    jeśli nie jest spełniony predykat *isDeterministicTrace*(*Stack*), usuń element ze stosu *pop*(*Stack*), przejdź do (5)
14. Jeśli spełniony jest predykat *isLastStateBranching*(*Stack*) lub *isLastStateCovering*(*Stack*) :
    a) sprawdź, czy zachodzi przypadek dywergencji, czyli czy spełnione jest *isLastStateDivergent*(*Stack*)
    b) podstaw: $x_{corr} := x_{corr} + vsnh.x$ oraz $x_{corr}' := x_{corr}' + vsnh.x'$;

c) usuń element ze stosu *pop*(*Stack*)

d) przejdź do (5)

15. Jeśli trójka (*vsnh.y* , *vsnh.y*' , *vsnh.R*) była już wcześniej analizowana, czyli (*vsnh.y* , *vsnh.y*' , *vsnh.R*) $\in S$ lub $\exists\, s \in S$. ($s.\, y = vsnh.y \wedge s.\, y' \leq vsnh.y' \wedge s.\, R \approx vsnh.R$) , usuń element ze stosu *pop*(*Stack*).

W przeciwnym przypadku dodaj trójkę (*vsnh.y* , *vsnh.y*' , *vsnh.R*) do zbioru *S*.

16. Przejdź do (5)

Faza III (Analiza rezultatów)

Analiza rezultatów podobna jest do algorytmu dla badania poprawności dla homomorficznej funkcji obserwacji. (Patrz podrozdział B.C.6).

Różnice dotyczą jedynie zaostrzonych warunków określających rozstrzygalność w przypadku pozytywnego werdyktu dotyczącego całkowitej lub częściowej poprawności.

Dodatkowe wymaganie wynika z założenia (2) twierdzenia 6-4.

Niech $RSUM = \sum_{(y,y',R)\in\hat{S}} R$.

Pozytywny werdykt jest uważany za nierozstrzygalny jeśli nie jest spełniony następujący warunek:

$$\forall k\,.\, p_{obs}(k) \neq 0.\exists i \in \{1,\ldots,n\}\,.\, RSUM_k(i) = 0 \wedge M_k(i) \neq 0$$