CZASOPISMO TECHNICZNE
WYDAWNICTWO POLITECHNIKI KRAKOWSKIEJ

z. 1-I/2008
ISSN 0011-4561
ISSN 1897-631X

ARTUR KRYSTOSIK*

# EMLAN: JĘZYK MODELOWANIA I FORMALNEJ WERYFIKACJI OPROGRAMOWANIA SYSTEMÓW WBUDOWANYCH

# EMLAN: LANGUAGE FOR MODELING AND FORMAL VERIFICATION OF EMBEDDED SYSTEMS

Streszczenie

*Embedded System Modeling Language* (EMLAN) jest wysokiego poziomu językiem modelowania i formalnej weryfikacji oprogramowania systemów wbudowanych. Mechanizmy języka pozwalają na modelowanie rozmaitych aspektów systemu wbudowanego, takich jak: współbieżność, przerwania, mechanizmy synchronizacji, czas. Formalna weryfikacja polega na translacji modelu systemu wyrażonego w języku EMLAN do systemu automatów DT-CSM (*Discrete Time Concurrent State Machines*), generacji grafu stanów osiągalnych systemu i badaniu jego własności temporalnych z wykorzystaniem logiki CTL. Przykładem zastosowania jest weryfikacja oprogramowania systemu alarmu samochodowego.

*Słowa kluczowe*: *formalna weryfikacja*, *modelowanie*, *systemy wbudowane*

Abstract

Embedded System Modeling Language (EMLAN) is high-level language for modeling and model checking the embedded systems software. The language addresses a number of topics such as: partitioning of the system, concurrency, interrupts, synchronization mechanisms, time, data transformations, hardware interactions. Model checking of the EMLAN specification is based on translations into DT-CSM (Discrete Time Concurrent State Machines), generation of a reachability graph (represented in BDD) and checking temporal formulas (CTL) representing requirements. As an example a verification of car alarm system is given.

*Keywords*: *formal verification*, *modeling*, *embedded systems*

* Dr inż. Artur Krystosik, Instytut Informatyki, Wydział Elektroniki i Technik Informacyjnych, Politechnika Warszawska.

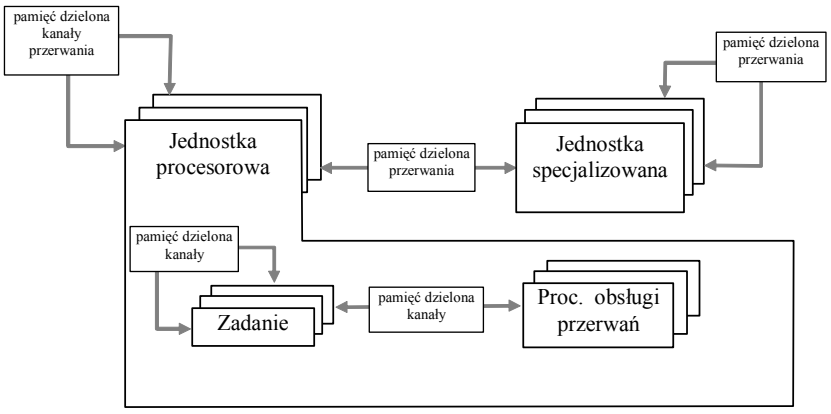
## 1. Wstęp

Zakres zastosowań systemów wbudowanych jest bardzo szeroki. Od systemów sterownia w przemyśle lotniczym, medycznym, motoryzacyjnym do popularnych urządzeń gospodarstwa domowego oraz komputerów przenośnych. W każdym z obszarów ich zastosowań problem niezawodności jest jednym z kluczowych aspektów procesu projektowania i produkcji. Ze względu na cechy systemów wbudowanych, takie jak: współbieżność, reaktywność oraz interakcja ze sprzętem, weryfikacja ich poprawności jest szczególnie trudna z użyciem tradycyjnych technik testowania oprogramowania. Jedną z obiecujących dróg efektywnego zapewnienia wysokiej jakości oprogramowania systemów wbudowanych jest zastosowanie weryfikacji modelowej (ang. *model checking*). Współcześnie rozróżnia się dwa główne podejścia do weryfikacji modelowej oprogramowania. Pierwsze z nich polega na weryfikacji specyfikacji systemu za pomocą wybranego formalizmu, weryfikacji modelu za pomocą danej metodyki, a następnie ręcznej lub automatycznej syntezy oprogramowania (lub sprzętu). Drugim podejściem jest weryfikacja istniejącego kodu oprogramowania. Jednym z najważniejszych problemów weryfikacji specyfikacji jest przejście pomiędzy modelem a późniejszą implementacją. Dystans pomiędzy modelem systemu, a jego implementacją zależy od wielu czynników, ale najważniejszymi są: poziom modelowania, zrozumienie przez inżynierów metodyki danego narzędzia modelowania, a także zastosowane abstrakcje. Narzędzia weryfikacji modelowej ogólnego przeznaczenia, takie jak SPIN [4], SMV [3] są często stosowane do weryfikacji systemów wbudowanych. Niestety, ich zastosowanie nie jest łatwe, w szczególności dla ludzi bez odpowiedniego przygotowania. Modelowanie systemów wbudowanych pociąga za sobą konieczność wykonania modeli rozmaitych operacji podstawowych (mechanizmów synchronizacji, przerwań, mechanizmów interakcji ze sprzętem, modeli szeregowania zadań), co wymaga głębokiego zrozumienia stosowanej metodyki oraz jest zadaniem o dużym stopniu złożoności. Dopiero w następnym kroku, posługując się nimi, można wykonać model danego systemu. Wszystko to powoduje, że zastosowanie narzędzi modelowania ogólnego przeznaczenia jest trudne. Propozycją rozwiązania powyższego problemu jest język EMLAN (o składni zbliżonej do języka C), który został wyposażony w większość mechanizmów podstawowych niezbędnych do modelowania i weryfikacji oprogramowania systemów wbudowanych. Głównym celem było stworzenie języka tak bliskiego, jak to tylko możliwe, do języka implementacji systemu wbudowanego, przy zachowaniu pełnych możliwości weryfikacji modelowej. EMLAN umożliwia również wyrażanie aspektów często pomijanych podczas weryfikacji modelowej, takich jak: różne rodzaje współbieżności (podział czasu, współprogramy, równoległość) czy abstrakcje czasu. Język został oparty na konstrukcjach i pojęciach powszechnie stosowanych w środowisku systemów wbudowanych, takich jak: zadanie, instrukcja, zmienna, przerwanie, semafor, monitor, kanał itd. Wszystkie te cechy pozwalają na redukcję dystansu pomiędzy modelem a późniejszą implementacją systemu, a także na bardzo łatwą automatyczną generację kodu programu. Zbliżone podejście oferuje System C [8], język oparty na C++, wyposażony w dedykowane dla systemów wbudowanych mechanizmy. Jednak złożoność C++ umożliwia w praktyce jedynie symulacyjne badanie poprawności modelu. Badanie modelowe staje się możliwe [9] dopiero po silnych ograniczeniach nałożonych na język i zasadniczo sprowadza się do problemu weryfikacji gotowego oprogramowania.

Dalsza część artykułu została pomyślana w następujący sposób: rozdz. 2 zawiera wprowadzenie do automatów DT-CSM, formalizmu, w którym wyrażony jest język EMLAN; rozdz. 3 zawiera skrócony opis języka EMLAN; w rozdz. 4 znajduje się przykład weryfikacji oprogramowania alarmu samochodowego.

## 2. Język EMLAN

EMLAN jest strukturalnym, imperatywnym językiem modelowania oprogramowania dla reaktywnych systemów wbudowanych. Jego podstawowymi cechami są:
- operowanie zestawem typowych instrukcji sterujących, takich jak: *if*, *while*, *switch*,
- arytmetyka stałoprzecinkowa na zmiennych,
- operowanie zestawem konstrukcji językowych i mechanizmów stosowanych w oprogramowaniu systemów wbudowanych, takich jak: zadania, przerwania, semafory, kanały komunikacyjne itp.,
- instrukcje modelujące upływ czasu,
- możliwość zastosowania różnych modeli współbieżności, np. modelu współprogramów i modelu z podziałem czasu,
- semantyka wyrażona formalnie za pomocą translacji do automatów DT-CSM.




Rys. 1. Elementy modelu systemu wbudowanego języka EMLAN

Fig. 1. Components of the EMLAN embedded system model

Na potrzeby języka przyjęto pewien ogólny model systemu wbudowanego (zob. rys. 1) pokrywający możliwie szeroki zakres reaktywnych systemów wbudowanych. Zgodnie z nim system wbudowany składa się z niezależnych *jednostek wykonawczych* reprezentujących komponenty odpowiedzialne za sterowanie w systemie, komunikujących się ze sobą za pomocą: *pamięci dzielonej*, *kanałów komunikacyjnych* i *przerwań*. Jednostki wykonawcze podzielone są na dwa rodzaje:
- Jednostki procesorowe (odpowiadające procesorom, komputerom itp.), na których uruchamiane są *zadania* w postaci sekwencyjnych programów EMLAN oraz znajdują się procedury obsługi przerwań.

– Jednostki specjalizowane, które odpowiadają dedykowanym komponentom sprzętowym występującym w danym systemie wbudowanym.

### 2.1. Typy danych

EMLAN umożliwia definiowanie zmiennych należących do tzw. typów prostych obejmujących: *int*<*n*> (*n*-bitowy integer), *enum* (typ wyliczeniowy) oraz typów obiektowych reprezentujących mechanizmy synchronizacji i komunikacji, np. *Semaphore*, *Channel*. Zmienne typów prostych mogą być deklarowane jako zmienne lokalne i globalne, natomiast zmienne typów obiektowych mogą być wyłącznie zmiennymi globalnymi.

### 2.2. Zadania i procedury

Zadaniem jest mająca nazwę sekwencja instrukcji, która jest statycznie przypisana do jednostki wykonawczej. Nie jest możliwe dynamiczne tworzenie nowych zadań. Wynika to z oparcia języka EMLAN na automatach DT-CSM, które są modelem skończeniestanowym. Ciałem zadania jest sekwencja deklaracji zmiennych oraz instrukcji. Semantyka procedur jest identyczna jak w klasycznych językach imperatywnych. Celem ich zastosowania jest zwiększenie modularności i czytelności programu.

Własności procedur:

– Parametry procedur są zawsze przekazywane przez referencję, przy czym referencja może odnosić się do wartości stałej.
– Zabronione jest rekurencyjne wywołanie procedur (bezpośrednie i pośrednie).
– Typy parametrów formalnych muszą być identyczne jak typy parametrów aktualnych.
– Powrót z procedury następuje po wykonaniu ostatniej instrukcji procedury lub napotkaniu instrukcji *return*.

### 2.3. Instrukcje

Instrukcje dzielą się na instrukcje proste oraz instrukcje złożone. Instrukcją złożoną jest sekwencja instrukcji objęta nawiasami klamrowymi { }. Instrukcje proste należą do jednej z następujących klas: instrukcje sterujące, instrukcja przypisania, instrukcje operujące na obiektach synchronizacji i komunikacji, instrukcje modelowania upływu czasu i instrukcje systemowe.

Zbiór instrukcji sterujących składa się z instrukcji: *if*, *while*, *switch*, *select*, które są dobrze znane z języka C. W skład wyrażeń logicznych mogą wchodzić zmienne lokalne i globalne oraz operatory logiczne i arytmetyczne. Instrukcje systemowe pozwalają na zgłaszanie i maskowanie przerwań oraz przełączanie zadań (dla współprogramów).

### 2.4. Mechanizmy synchronizacji i komunikacji

Język EMLAN udostępnia kilka podstawowych klas obiektów służących do synchronizacji i komunikacji między zadaniami. Zostały one dobrane tak, aby pokrywały możliwie szerokie spektrum mechanizmów spotykanych we współczesnych systemach operacyjnych. Zdefiniowano następujące klasy obiektów:

– *Semaphore* – semafor binarny lub wielowartościowy,
– *Mutex* – sekcja krytyczna,
– *MonitorHansen* – monitor Hansena,

– *MonitorHoare* – monitor Hoare'a,
– *Channel* – kanał do komunikacji za pomocą komunikatów w trybie *rendez-vous*.

## 2.5. Przerwania

Przerwania są często wykorzystywanym mechanizmem w oprogramowaniu systemów wbudowanych i nie mogą być pomijane w procesie jego modelowania.

Język EMLAN modeluje zarówno przerwania maskowalne, jak i niemaskowalne, a modele jednostek procesorowych mogą zawierać układ kontrolera przerwań lub mogą go nie zawierać. Zastosowanie kontrolera przerwań pozwala na zamodelowanie hierarchicznego układu przerwań o architekturze zaczerpniętej z rodziny układów Intel 8259. W przypadku braku kontrolera układ przerwań jest płaski o architekturze zaczerpniętej z rodziny procesorów Intel 8251. Celem udostępnienia dwóch modeli układów obsługi przerwań jest pokrycie jak najszerszego spektrum zastosowań.

## 2.6. Czas i abstrakcje czasu

Modele w języku EMLAN mogą wyrażać upływ czasu za pomocą trzech abstrakcji czasu: abstrakcji następstwa, abstrakcji uniezależniającej, abstrakcji dyskretnego czasu.

Abstrakcja następstwa przyjmuje, że czas jest dyskretny (podzielony na chwile czasowe) i globalny (dla każdego zadania upływa w tym samym tempie). Upływ czasu jest modelowany jako następstwo kolejnych stanów w automatach DT-CSM, które reprezentują instrukcje języka EMLAN. W przypadku, gdy automatowe modele poszczególnych instrukcji EMLAN zawierają różne liczby stanów, oznacza to różny czas ich wykonania.

Abstrakcja uniezależniająca polega na zastosowaniu takiej konstrukcji modeli poszczególnych instrukcji języka, aby czas ich wykonania mógł przyjmować dowolne wartości. Jeżeli dla tak skonstruowanego modelu systemu wynik jego weryfikacji jest pozytywny, oznacza to, że weryfikowane własności nie zależą od czasu. Przekładając wynik weryfikacji modelowej na rzeczywisty system, można wnioskować, że jego zachowanie jest niezależne np. od prędkości pracy jego komponentów składowych czy interwałów czasu pomiędzy sygnałami zewnętrznymi.

Abstrakcja dyskretnego czasu pozwala na bezpośrednie wyrażanie własności czasowych modelowanego systemu. Jej semantyka jest następująca:
– Czas jest dyskretny, wyrażany w umownych jednostkach.
– Czas musi zostać wprowadzony do modelu wprost, za pomocą instrukcji modelowania upływu czasu *wait*.
– Upływ czasu odmierzany jest przez lokalne zegary związane z instrukcjami *wait*. Wszystkie zegary lokalne odmierzają upływ czasu w tym samym tempie.
– Czas upływa w momencie, gdy wszystkie zadania znajdują się w jednym ze stanów:
  – wykonują operację *wait*,
  – są zawieszone w oczekiwaniu na zakończenie operacji synchronizacyjnej lub komunikacyjnej.
– Instrukcja niepowodująca zawieszenia zadania i niebędąca instrukcją *wait* nie powoduje upływu czasu, jej czas trwania wynosi zero.

### 2.7. Modele współbieżności

Język EMLAN jest w założeniach przeznaczony dla systemów wbudowanych, w których występują zarówno zależności czasowe, jak i różne rodzaje współbieżności. Czynniki te powodują, że są one wrażliwe na niewłaściwe zastosowanie abstrakcji współbieżności czy abstrakcji szeregowania. W związku z tym efektywne modelowanie tego rodzaju systemów wymaga odpowiedniego wsparcia ze strony narzędzia.

W języku EMLAN wsparcie to polega na zdefiniowaniu czterech modeli współbieżności, wg których mogą działać maszyny wirtualne jednostek wykonawczych. Modele te odpowiadają zdefiniowanym wcześniej klasom współbieżności, z wyjątkiem modelu równoległego z przerwaniami, który reprezentuje system mieszany.

- Model równoległy, w którym zarówno zadania, jak i procedury obsługi przerwań wykonywane są równolegle. Model ten charakteryzuje się najmniejszymi rozmiarami automatów składających się na poszczególne komponenty systemu.
- Model równoległy z przerwaniami, w którym zadania wykonywane są równolegle, natomiast obsługa przerwania powoduje zatrzymanie wykonywania zadań na czas obsługi przerwania.
- Model współprogramów, w którym w danej chwili działa dokładnie jedno zadanie. Sterowanie pomiędzy zadaniami przekazywane jest jawnie przez programistę za pomocą instrukcji *switchTo*. Obsługa przerwania blokuje wykonanie aktywnego zadania na czas obsługi przerwania.
- Model z wywłaszczaniem, w którym zadania wykonywane są z podziałem czasu określonym przez algorytm szeregowania zamodelowany za pomocą automatu DT-CSM.

Model współbieżności określa się niezależnie dla każdej jednostki wykonawczej, przy czym zadania wykonywane na różnych jednostkach wykonawczych pracują względem siebie zawsze równolegle.

## 3. Automaty DT-CSM

Automaty DT-CSM (*Concurrent State Machines*) są etykietowanymi, skierowanymi grafami połączonymi ze zmiennymi całkowitoliczbowymi i zegarami. Model DT-CSM jest rozszerzeniem automatów CSM opracowanych przez dr. J. Mieścickiego [5]. Automat DT-CSM (podobnie do CSM) składa się z: zbioru stanów, zbioru krawędzi etykietowanych formułami logicznymi, zbioru symboli alfabetu wyjściowego i wejściowego. Dodatkowo w jego skład wchodzą: zbiór zmiennych lokalnych, zbiór zmiennych globalnych, zbiór zegarów, zbiór predykatów, zbiór akcji krawędziowych.

W stanach automatu mogą być generowane podzbiory dowolnych symboli należących do alfabetu wyjściowego automatu. Jeden ze stanów automatu jest stanem początkowym. Ze stanem związany jest również predykat upływu czasu (zależny od zmiennych), którego wartość *true* oznacza, że podczas pozostawania w danym stanie czas może płynąć.

Zbiory zmiennych zawierają zmienne całkowitoliczbowe o ograniczonym zakresie.

- Zmienne lokalne, mogą być wykorzystywane wyłącznie w predykatach i akcjach krawędziowych danego automatu.
- Zmienne globalne, są współdzielone pomiędzy innymi automatami.

Zbiór zegarów zawiera lokalne dla automatu, ograniczone liczniki całkowitoliczbowe, pozwalające na odmierzanie kwantów czasu. Wartości początkowe zegarów wynoszą zero. Na zegarach mogą być wykonywane operacje inkrementacji z nasyceniem (po osiągnięciu wartości maksymalnej wartość zegara nie ulega dalszemu zwiększaniu) oraz operacje zerowania.

Zbiór predykatów zawiera predykaty zależne od zmiennych lokalnych, zmiennych globalnych i zegarów automatu. Zdefiniowane predykaty mogą być wykorzystywane w formułach logicznych określających warunki przejść pomiędzy stanami automatu.

Przejścia pomiędzy stanami automatu etykietowane są formułami logicznymi i akcjami krawędziowymi. Formuły logiczne uzależniają przejście od wystąpienia symboli alfabetu wejściowego oraz wartościowania predykatów zależnych od wartości zmiennych i zegarów automatu. Dla przykładu, formuła *a * ! b* p()* oznacza wystąpienie symbolu *a* i niewystąpienie symbolu *b* i prawdziwość predykatu *p*.

Podobnie jak w CSM krawędź pomiędzy stanami $s$ i $s'$, gdy $s \neq s'$, etykietowana formułą *f* oznacza, że automat może wykonać przejście ze stanu $s$ do $s'$ wtedy i tylko wtedy, gdy formuła *f* jest prawdziwa. Formuła taka nazywana jest warunkiem przejścia. W przypadku gdy $s = s'$, formuła *f* jest warunkiem pozostania automatu w stanie $s$. W przypadku gdy w danym stanie jednocześnie prawdziwych jest więcej niż jedna formuła, oznacza to niedeterministyczny wybór krawędzi, po której zostanie wykonane przejście. Jeżeli krawędź jest etykietowana wartością *true*, która reprezentuje logiczną wartość *prawda*, oznacza to przejście bezwarunkowe, czyli przejście, które może być wykonane spontanicznie, niezależnie od dostępnych symboli alfabetu wejściowego. Automaty DT-CSM są automatami zupełnymi. Oznacza to, że suma formuł etykietujących krawędzie wychodzące z danego stanu musi być równa *true*.

Akcje krawędziowe określają nowe wartości zmiennych lokalnych, zmiennych globalnych i zegarów automatu. Mogą być przypisane wyłącznie do krawędzi pomiędzy różnymi stanami automatu.

W opisie automatu DT-CSM nie zdefiniowano semantyki upływu czasu, co jest związane z założeniem, że dla pojedynczego automatu nie definiuje się pojęcia zachowania. Pojęcie zachowania oraz semantyka upływu czasu są zdefiniowane dla systemu automatów DT-CSM składającego się z niepustego zbioru automatów DT-CSM oraz zbioru wartości początkowych zmiennych globalnych automatów.

Zachowanie systemu automatów opisywane jest przez graf stanów osiągalnych GSO, który opisuje wszystkie możliwe przejścia między stanami automatów składowych przy następujących założeniach:

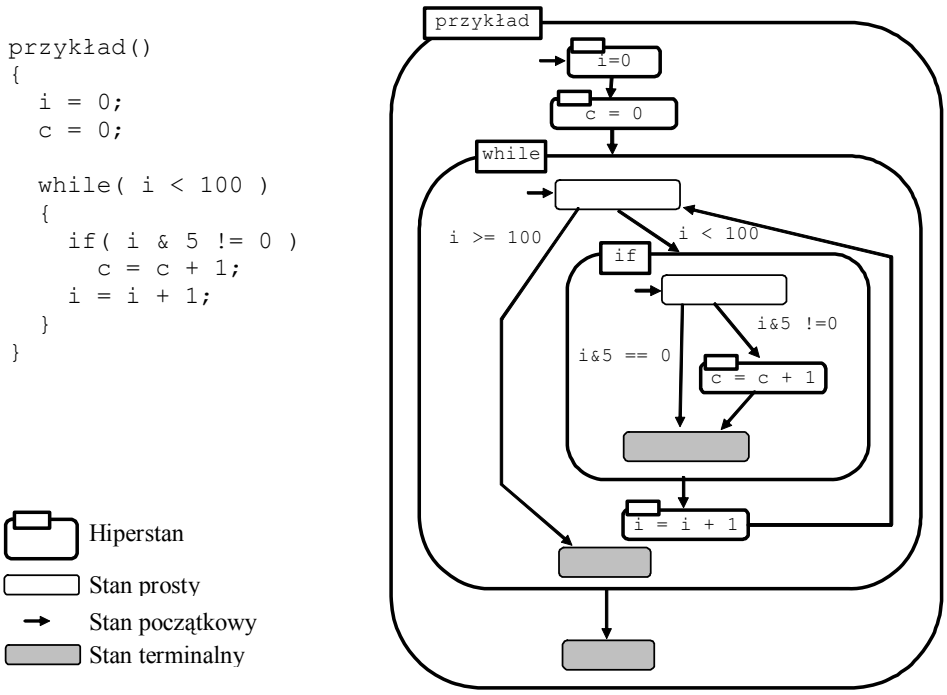
– Automaty systemu startują w swoich stanach początkowych z początkowymi wartościami zmiennych i zegarów.
– W kolejnych krokach każdy z automatów równocześnie z innymi wykonuje przejście do nowego stanu (lub pozostaje w stanie bieżącym).
– Podczas przejść wykonywane są akcje krawędziowe, przez co modyfikowane są wartości określonych dla danego przejścia zmiennych i zegarów automatów (zegary mogą być wyłącznie zerowane).

Semantyka upływu czasu dla systemu automatów jest następująca:

1. W stanie początkowym systemu automatów wszystkie zegary mają wartość 0.
2. Upływ czasu odbywa się przez zwiększanie wartości zegarów.

3. Wartości zegarów mogą być zwiększane wtedy i tylko wtedy, gdy wszystkie automaty wykonują przejście niewyprowadzające ich z aktualnego stanu (przejścia po „uszach"). W takim przypadku:
   a) zegary każdego automatu, dla którego predykat upływu czasu jest prawdziwy, są inkrementowane modulo zakres zegara,
   b) zegary każdego automatu, dla którego predykat upływu czasu jest fałszywy, zachowują swoją wartość.
4. Zerowanie wartości zegarów możliwe jest w ramach akcji krawędziowych.

Dla automatów DT-CSM, na podstawie BDD (ang. *Binary Decisions Diagrams*) [1], zostały opracowane symboliczne algorytmy generowania GSO, detekcji blokad oraz detekcji współbieżnych dostępów do zmiennych dzielonych. Automaty DT-CSM są formalizmem, do którego translowane są konstrukcje języka EMLAN opisanego w rozdz. 3.




Rys. 2. Przykład zadania EMLAN i jego reprezentacja w postaci hipergrafu

Fig. 2. EMLAN task and it's representation as a hypergraph

## 4. Translacja EMLAN na automaty DT-CSM

EMLAN jest strukturalnym językiem imperatywnym, co pozwala na reprezentację zadań i procedur obsługi przerwań w postaci grafu sterowania (hipergrafu), w którym stany odpowiadają poszczególnym instrukcjom, a przejścia reprezentują przepływ sterowania. Krawędzie mogą być etykietowane warunkami determinującymi wykonanie przejścia.

Krawędź niemająca etykiety oznacza przejście bezwarunkowe. Hipergraf składa się z dwóch typów stanów: stanów prostych oraz hiperstanów. Każdy hiperstan ma wyróżnione stan początkowy oraz stan końcowy. Hiperstany reprezentują konstrukcje językowe, takie jak: zadanie, procedura, instrukcja. Stany proste reprezentują niepodzielne konstrukcje języka (niemające struktury wewnętrznej), takie jak rozejścia warunkowe oraz przypisania wartości zmiennych. Hiperstany zawierają w sobie hipergrafy z własnymi stanami początkowymi i terminalnymi. Każdy z hiperstanów grafu sterowania ma dokładnie jedną krawędź wejściową i jedną krawędź wyjściową (co jest efektem strukturalności języka). Przykładowy program w języku EMLAN i jego hipergraf przedstawiono na rys. 2. Dla uproszczenia rysunku hiperstany reprezentujące instrukcje przypisania nie zawierają podgrafów (byłyby one podgrafami jednoelementowymi).

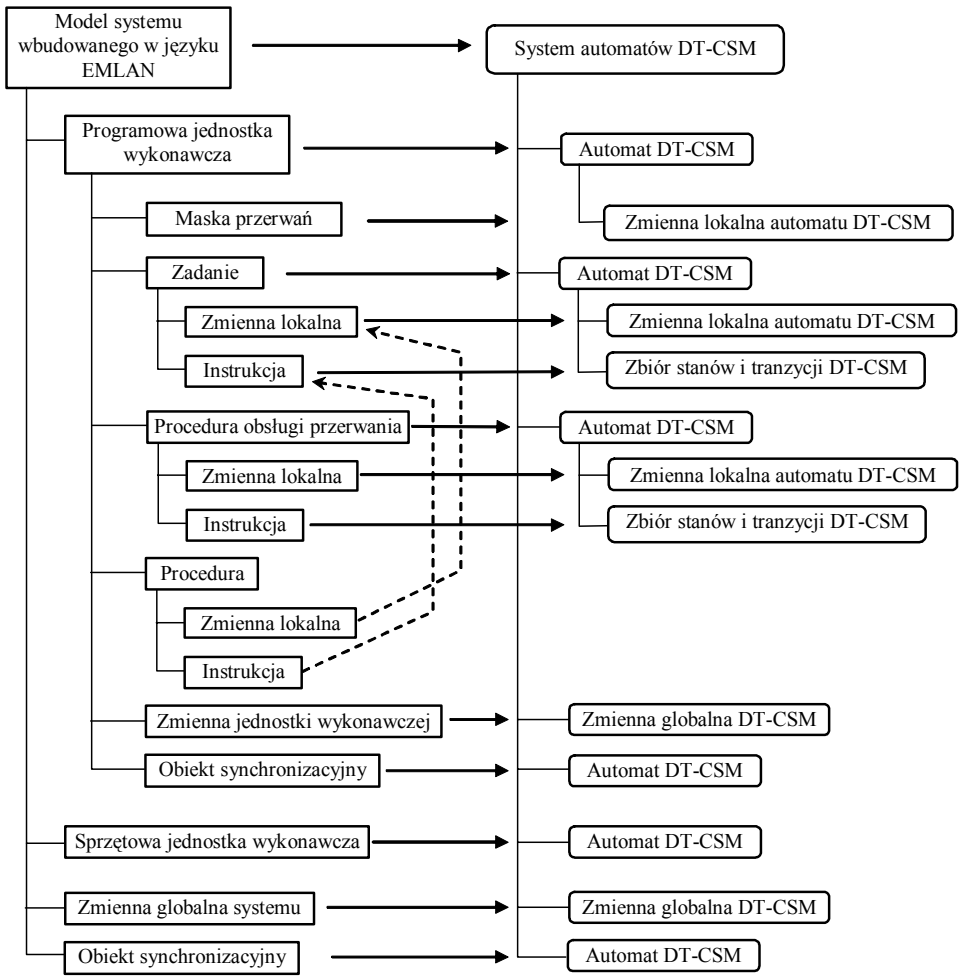
Semantyka EMLAN wyrażona jest przez translację poszczególnych elementów języka na odpowiadające im konstrukcje automatów DT-CSM. Jasne i czytelne przedstawienie reguł translacji wymaga zastosowania podejścia hierarchicznego, polegającego na definiowaniu reguł translacji dla kolejnych poziomów szczegółowości: od poziomu modelu systemu wbudowanego do modelu poszczególnych instrukcji języka EMLAN. Ogólne reguły translacji modelu EMLAN na model DT-CSM przedstawiono na rys. 3.

Strzałki ciągłe wskazują elementy modelu DT-CSM, na które translowane są poszczególne elementy EMLAN. Strzałki przerywane reprezentują wewnętrzną translację, w której dany element specyfikacji EMLAN jest translowany na inny element tej samej specyfikacji. Translacji tej podlegają np. wywołania procedur, dla których polega ona na wstawieniu kodu procedury w miejsce jej wywołania.

Metodyka translacji EMLAN na DT-CSM jest następująca:

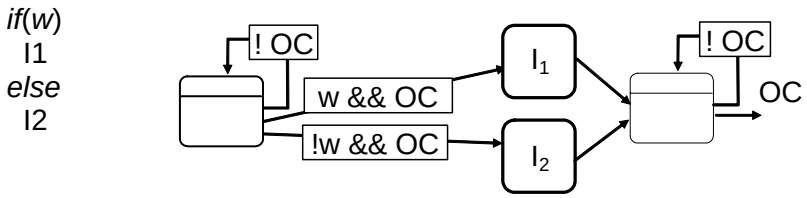
– Zadania i procedury obsługi przerwań są reprezentowane jako hipergrafy, w których każda instrukcja języka jest reprezentowana jako hiperstan.
– Część konstrukcji języka (wywołanie procedury, typy wyliczeniowe) podlega translacji wewnętrznej, która upraszcza specyfikację.
– Dla każdej instrukcji języka został opracowany jej model DT-CSM w postaci hipergrafu. Postać modeli jest zależna od rodzaju współbieżności, jaki realizuje dana jednostka procesorowa.
– Hipergrafy są rozwijane w grafy płaskie, które po zapisaniu w odpowiedniej notacji stają się automatami DT-CSM.

Dla obiektów synchronizacji i komunikacji, a także dla mechanizmów zgłaszania i obsługi przerwań opracowano ich modele w postaci automatów DT-CSM. Przykładowy model dla instrukcji rozejścia warunkowego *if* przedstawiono na rys. 5.




Rys. 3. Zasady translacji EMLAN na DT-CSM

Fig. 3. EMLAN to DT-CSM translation rules




Rys. 4. Translacja instrukcji warunkowej *if*

Fig. 4. IF statement translation

Warunek OC nazywany jest warunkiem kontynuacji. Służy do zatrzymywania wykonywania zadania w przypadku np. obsługi przerwania. Postać tego warunku zależy, oczywiście, od modelu współbieżności, który został przyjęty dla danej specyfikacji. W przypadku modelu równoległego warunek OC ma postać *true*, gdyż zadanie nigdy nie jest przerywane. Dla modelu z przerwaniami przybiera on postać *! <u>.stop*, gdzie *<u>.stop* jest sygnałem automatu DT-CSM zatrzymującym pracę zadań jednostki procesorowej *<u>*, do której zgłoszono przerwanie.

## 5. Weryfikacja autoalarmu samochodowego

Podstawową funkcją urządzenia jest uniemożliwienie uruchomienia samochodu osobie niepowołanej, czyli osobie nieznającej kodu dezaktywującego alarm. Blokada realizowana jest przez odcięcie zasilania układu zapłonowego oraz pompy paliwowej. W przypadku wykrycia prób zgadnięcia kodu dezaktywującego uruchomiana jest syrena alarmowa. Do celów komunikacji z użytkownikiem autoalarm wyposażony jest w klawiaturę numeryczną, diodę świecącą oraz głośniczek. Po włączeniu zasilania urządzenie przechodzi do stanu czuwania, w którym wyłączony jest zapłon, a urządzenie oczekuje na wprowadzenie 6-cyfrowego kodu dezaktywującego. Po wprowadzeniu pierwszych 4 cyfr alarm ulega częściowej dezaktywacji (włączane jest zasilanie zapłonu i pompy paliwa), a urządzenie wchodzi w stan oczekiwania na wprowadzenie 2 pozostałych cyfr. Po ich wprowadzeniu alarm jest całkowicie zdezaktywowany. Natomiast jeżeli całkowita dezaktywacja nie nastąpi w określonym czasie (1,5 min) od wprowadzenia pierwszych 4 cyfr, urządzenie przechodzi do stanu alarmu, w którym wyłączane jest zasilanie zapłonu i pompy paliwa oraz uruchamiana jest syrena alarmowa.

Oprogramowanie systemu liczy ok. 800 linii kodu w języku C, przy czym ponad 90% kodu w języku C zostało przeniesione wprost do specyfikacji EMLAN. Sformułowano 12 warunków poprawności systemu, które zostały wyrażone za pomocą 22 formuł logiki QsCTL [2]. Czas pracy nad weryfikacją ww. systemu nie przekroczył kilkunastu godzin.

Weryfikację przeprowadzono, modelując zależności czasowe (timeouty) występujące w systemie oraz stosując abstrakcję czasu polegającą na dopuszczeniu wszystkich możliwych koincydencji czasowych. Rezultaty generowania grafu stanów osiągalnych (GSO) przedstawiono w tab. 1 (Athlon 64 1,8 GHz).

Tabela 1

**Rezultaty generowania GSO**

| Parametr | Z zależnościami czasowymi | Bez zależności czasowych |
|---|---|---|
| Liczba stanów | 18 609 | 3 722 281 |
| Liczba tranzycji | 19 162 | 10 810 912 |
| Czas wyznaczenia i zapisania GSO | 53,82 s | 7 969,74 s |
| Liczba węzłów w drzewie BDD reprezentującym GSO | 55 737 | 1 148 219 |
| Czas detekcji jednoczesnych dostępów typu do zmiennych dzielonych | 1,87 s | 98,54 s |
| Czas detekcji blokad (algorytm symboliczny) | 0,03 s | 1,66 s |
| Liczba zmiennych BDD | 168 | 164 |
| Wykorzystana pamięć operacyjna | 30 MB | 120 MB |

Wnioski z przeprowadzonej weryfikacji są następujące:

1. Stopień złożoności weryfikowanego urządzenia jest duży. Rozmiar oprogramowania sterującego w języku C przekracza 800 linii.
2. Badania modelu urządzenia nie wykazały istnienia błędów. Urządzenie zostało wszechstronnie przetestowane w ramach prac nad systemami tolerującymi błędy [6, 7] prowadzonych w Instytucie Informatyki.
3. Wykryto drobną usterkę polegającą na przechodzeniu do stanu alarmu mimo podania poprawnego kodu rozbrojenia. Zjawisko to ujawnia się, gdy ostatni klawisz kodu rozbrajającego zostanie wprowadzony bezpośrednio przed przerwaniem powodującym upłynięcie timeotu oczekiwania na rozbrojenie. Powyższe zachowanie jest praktycznie niemożliwe do zaobserwowania podczas testowania urządzenia.

## 6. Zakończenie

W artykule przedstawiono koncepcję języka modelowania i formalnej weryfikacji EMLAN, a także opis automatów DT-CSM będących formalizmem, w którym EMLAN jest wyrażony. Opracowany język został zaimplementowany w narzędziu weryfikacji EMLAN 1.0 oraz zastosowany z dobrym rezultatem do weryfikacji oprogramowania autoalarmu samochodowego.

*Praca została wykonana w ramach grantu KBN nr NN516429733.*

## Literatura

[1] Bryant R.E., *Binary Decision Diagrams: Enabling Technologies for Formal Verification*, Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, 1995, 236-243.

[2] Daszczuk W.B., *Verification of temporal properties in concurrent systems*, Ph.D. thesis, ICS WUT, 2003.

[3] Halbwachs N., Peled D., *NuSMV: a new symbolic model verifier. Proceeding of International Conference on Computer-Aided Verification (CAV '99)*, In Lecture Notes in Computer Science, No. 1633, Springer, Trento, Italy, July 1999, 495-499.

[4] Holzmann G.J., *The Model Checker Spin. IEEE Transactions On Software Engineering*, Vol. 23, No. 5, May 1997.

[5] Mieścicki J., *Concurrent system of communicating machines*, Institute of Computer Science, WUT, Research Report 35/92.

[6] Sosnowski J., Gawkowski P., *Embedded System dependability evaluation with simulation tools*, Proc. of IFAC Workshop on Programmable Devices and Embedded Systems, 2006, 180-185.

[7] Wilczyński A., Sosnowski J., Gawkowski P., *Flexible microcontroller Simulator for testing purposes*, Proc. of IFAC Workshop on Programmable Devices and Embedded Systems, 2004, 310-315.

[8] Groetker T., Liao S., Martin G., Swan S., *System Design with System C*, Springer, 2002.

[9] Grobe D., Drechsler R., *Formal verification of LTL formulas for System C designs*, In Proc. Int'l Symposium on Circuits and Systems, 2003, 245-248.