PODSTAWY STEROWANIA
Tom 10 (1980), z. 1
pp. 57-68

LISTY DO REDAKCJI

Implementacja języka do symulacji układów ciągłych ze zdarzeniami dyskretnymi

JERZY KRÓL, JACEK KURAŚ, JACEK LEMBAS, MACIEJ ŚLUSAREK

(*Maszynopis wpłynął 15 maja 1979*)

1. Wprowadzenie

Celem pracy jest prezentacja metody implementacji procesora języka symulacyjnego, na przykładzie języka GODYS-5 [6]. Język GODYS-5 przeznaczony jest do symulacji komputerowej systemów dynamicznych ciągłych, w których występują zdarzenia powodujące skokowe zmiany wartości zmiennych stanu. Język jest, w szczególności, dostosowany do symulacji systemów o złożonym torze pomiarowym, zawierającym takie elementy, jak przerzutniki, liczniki, komutatory, detektory przekroczeń alarmowych i inne. Dostosowanie to uzyskano przez odpowiedni dobór operacji języka i przez zastosowanie zmiennokrokowej metody całkowania numerycznego z lokalizacją punktów nieciągłości [4].

2. Struktury programowe procesorów symulacyjnych

Procesorem symulacyjnym jest specjalny program tłumaczący program źródłowy w języku symulacyjnym do postaci docelowej oraz organizujący proces symulacyjny. Do podstawowych zadań procesora zaliczyć należy:

1. translację programu źródłowego do postaci pośredniej,
2. porządkowanie wyrażeń programu źródłowego, lub w języku pośrednim, w celu wyznaczenia sekwencji obliczalnej operacji,
3. realizację procesu symulacyjnego,
4. obsługę wejścia i wyjścia,
5. zarządzanie biblioteką podprogramów specjalnych, takich jak linia opóźniająca, generatory funkcji jednej i dwóch zmiennych.

Wspomnianym powyżej językiem pośrednim może być zarówno język zorientowany blokowo, jak też proceduralny, na przykład FORTRAN. Podkreślić należy, że zastosowanie pośredniego języka zorientowanego blokowo zasadniczo upraszcza strukturę procesora i minimalizuje czas translacji. Jeśli chodzi o obsługę wejścia i wyjścia, to przed zapoczątkowaniem eksperymentu symulacyjnego procesor wczytuje

niezbędne wartości parametrów i warunków początkowych, a po jego zakończeniu opracowuje wyniki i redaguje wydawnictwo.

System programowy procesora symulacyjnego zrealizowany może być jako: 1. interpreter, 2. bezpośredni program maszynowy, 3. kompilator, 4. prekompilator. W rozwiązaniu typu interpreter, na etapie tłumaczenia programu źródłowego, tworzona jest macierz strukturalna symulowanego systemu oraz baza danych zawierająca charakterystyki i parametry poszczególnych jego elementów. Zawartość macierzy strukturalnej interpretowana jest na każdym kroku całkowania, przy czym każdej operacji języka odpowiada podprogram, wywoływany wielokrotnie z różnymi parametrami.

Bezpośredni program maszynowy jest programem całkowania numerycznego równań różniczkowych zwyczajnych, zawierającym zestaw podprogramów sterowanych parametrycznie, dla którego program w języku symulacyjnym stanowi jedynie dane organizujące proces obliczeniowy. Zaletą tego rozwiązania jest pomijalny czas translacji, do wad należy mała elastyczność oraz konieczność rezerwacji dużej i stałej ilości pamięci. Tego rodzaju rozwiązanie stosowane było między innymi w procesorach języków MIMIC i ASIM [2, 5].

W systemie typu kompilator, w wyniku translacji programu źródłowego, tworzony jest program półskompilowany, który, po dołączeniu niezbędnych podprogramów bibliotecznych, przekształcany jest w binarny program wynikowy. Do zalet tego rozwiązania należy dobre wykorzystanie pamięci i duża efektywność procesu symulacji. Do wad zaliczyć trzeba złożoność kompilatora i brak możliwości wprowadzenia modyfikacji do programu wynikowego.

Procesor typu prekompilator przekształca program źródłowy w program w języku algorytmicznym (jest nim zwykle FORTRAN), który z kolei przetwarzany jest przez odpowiedni kompilator. Pozwala to na łatwe wprowadzenie, do programu w języku symulacyjnym, podprogramów i sekcji w języku proceduralnym oraz na prostą i efektywną organizację optymalizacji parametrycznej i obliczeń postsymulacyjnych. Do wad tego rozwiązania należy względnie długi czas trwania translacji i konieczność stosowania, jako narzędzia pomocniczego, języka proceduralnego.

3. Procesor języka GODYS-5

W przypadku języka GODYS-5 krótki przewidywany okres realizacji projektu i jego eksperymentalny charakter skłonił autorów do zastosowania, w realizacji procesora, rozwiązania mieszanego. Procesor ten składa się z dwóch współdziałających programów:

1. translatora języka opisu modelu,
2. systemu wykonawczego.

Struktura ta odzwierciedla funkcjonalny podział języka i tym samym pozwala na ograniczenie zapotrzebowania na pamięć operacyjną.

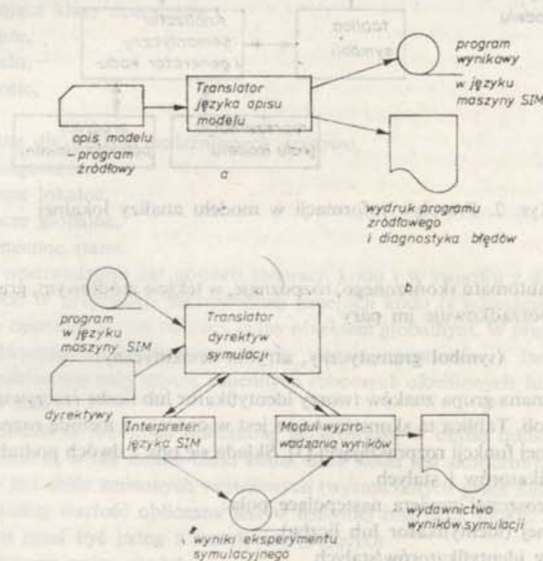
Translator języka opisu modelu analizuje program źródłowy od deklaracji MODEL do END. W trakcie tej analizy tworzona jest tablica identyfikatorów, przechowująca podstawowe ich atrybuty, oraz generowany jest program półskompilowany (przemieszczalny) w języku, zdefiniowanej dla potrzeb projektu, maszyny abstrakcyjnej SIM. Rozkazy tej maszyny odpowiadają elementarnym operacjom języka GODYS-5. Równocześnie tworzona jest lista prosta reprezentująca graf strukturalny modelu. W przypadku błędów syntaktycznych i semantycznych wydawana jest diagnostyka.

Po rozpoznaniu ogranicznika opisu modelu (deklaracja END) translator dokonuje globalnej analizy semantycznej, w szczególności wykrywane są odwołania do niezdefiniowanych zmiennych oraz wyznaczana jest sekwencja obliczalna operacji.

Po wyznaczeniu tej sekwencji następuje podział pamięci maszyny SIM pomiędzy poszczególne klasy operandów i generacja programu wynikowego, który zapisywany jest na pliku w pamięci masowej. System wykonawczy, po napotkaniu dyrektywy LOAD, odszukuje plik zawierający odpowiedni program wynikowy, wczytuje go i przechodzi do stanu, w którym wykonywane mogą być dyrektywy symulacji. Wykonanie dyrektyw EXECUTE/CONTINUE poprzedzone jest kontrolą poprawności parametrów modelu i symulacji, przykładowo sprawdzana jest sensowność tablic używanych przez funkcje FUNCT1, FUNCT2, SPFUN1, wykrywana jest próba użycia linii opóźniających przy zmiennokrokowej metodzie całkowania itp. Dodatkowo, przy wykonaniu dyrektywy EXECUTE, ustawiane są warunki początkowe.

W trakcie właściwego procesu symulacyjnego dokonywana jest interpretacja programu w języku maszyny abstrakcyjnej SIM, z cykliczną kontrolą warunków zakończenia symulacji. Proces symulacji kończony jest zamknięciem pliku z wynikami danego eksperymentu, co umożliwia ich ewentualne wykorzystanie do obliczeń postsymulacyjnych.

Ogólną strukturę programową procesora języka GODYS-5 przedstawia rys. 1.



Rys. 1. Struktura programowa procesora języka GODYS-5: a. translator języka opisu modelu, b. system wykonawczy

W dalszym ciągu przedstawimy organizację translatora języka opisu modelu oraz metodę wyznaczania sekwencji obliczalnej.

3.1. Struktura translatora języka opisu modelu

Jak wspomniano, zadaniem translatora języka opisu modelu jest analiza deklaracji i właściwego opisu modelu oraz synteza programu wynikowego w języku maszyny abstrakcyjnej SIM.

W celu dobrego wykorzystania pamięci i dostosowania się do standardów programów nakładanych dla komputerów serii ODRA-1300, translator podzielono na trzy współdziałające moduły analizy lokalnej, rozpoznające odpowiednio

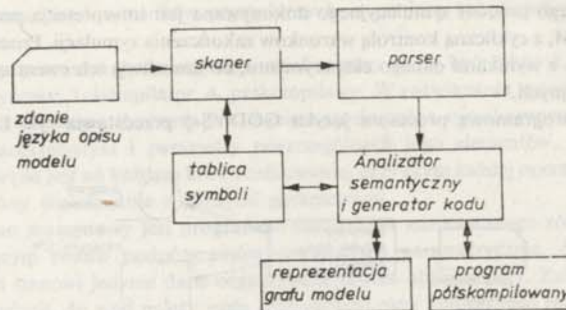
a. nagłówek programu (deklarację MODEL),
b. pozostałe deklaracje,
c. właściwy opis modelu.

oraz moduł analizy globalnej i generacji programu wynikowego.

Strukturę modułu analizy lokalnej przedstawia rys. 2.

Każdy z modułów analizy lokalnej jest przetwornikiem sterowanym składnią [1], którego jądrem jest parser oparty o gramatykę precedensyjną prostą [7]. Pracą procesora sterują tablice z zakodowanymi produkcjami i macierzą precedensyjną. Przedmiotem analizy syntaktycznej są pojedyncze deklaracje i instrukcje. Analiza ta kończona jest akceptacją lub wykryciem błędu. Parser wywołuje dwie podstawowe procedury

a. analizator leksykalny (skaner),
b. analizator semantyczny.



Rys. 2. Przepływ informacji w modelu analizy lokalnej

Skaner, będący realizacją automatu skończonego, rozpoznaje, w tekście źródłowym, grupy znaków tworzące atomy leksykalne i przyporządkowuje im pary

(symbol gramatyczny, atrybut semantyczny).

W przypadku, gdy rozpoznana grupa znaków tworzy identyfikator lub liczbę rzeczywistą, skaner przegląda i aktualizuje tablicę symboli. Tablica ta skonstruowana jest w oparciu o metodę rozpraszania, z wykorzystaniem pierwotnej i wtórnej funkcji rozpraszającej [3]. Składa się ona z dwóch podtablic: 1) rozproszonej, 2) kombinowanej identyfikatorów i stałych.

Element tablicy rozproszonej zawiera następujące pola:
— wskaźnik typu danej (identyfikator lub liczba),
— odsyłacz do tablicy identyfikatorów/stałych,
— wskaźnik typu identyfikatora (zastrzeżony, zmiennej, stałej, parametru itp.),
— atrybuty poszczególnych typów identyfikatorów (wskaźniki zdefiniowania i użycia, adres względny itp.).

Tablica symboli inicjowana jest przez wprowadzenie identyfikatorów zastrzeżonych, identyfikatora zmiennej niezależnej *T* i stałej rzeczywistej 1. Pozostałe identyfikatory definiowane są w programie źródłowym, przez wystąpienie na liście deklaracji lub użycie w opisie modelu.

Podkreślić tu należy, że w trakcie analizy właściwego opisu modelu, wszystkim niezadeklarowanym identyfikatorom przypisywany jest atrybut typu — zmienna modelu.

Zgodnie z ogólną procedurą analizy składni [7] symbole gramatyczne gromadzone są na stosie syntaktycznym. Niepuste atrybuty semantyczne umieszczone są na stosie semantycznym, na którym operują procedury semantyczne. Procedury te przyporządkowane są, jednoznacznie, produkcjom gramatyki języka wejściowego, zgodnie z techniką translacji sterowanej składnią. Do zadań ich należy generacja i lokalna optymalizacja kodu półskompilowanego oraz realizacja pewnych operacji na globalnych strukturach danych translatora. Najistotniejsze z tych operacji to
— aktualizacja atrybutów w tablicy symboli,
— konstrukcja listy prostej reprezentującej graf strukturalny symulowanego systemu,
— wykrywanie i neutralizacja błędów semantycznych.

3.2. Generacja programu półskompilowanego

Jak wspomniano, w trakcie analizy semantycznej generowany jest program półskompilowany w języku maszyny SIM. Maszyna ta ma organizację i listę operacji dostosowaną do potrzeb symulacji. Operacje te odpowiadają elementarnym funkcjom i operatorom języka GODYS-5 i są określone w zbiorze liczb rzeczywistych. Operandy i wyniki operacji umieszczone są w pamięci maszyny SIM, a rozkazy mają długość uzależnioną od typu operacji, na przykład

$$Y \leftarrow \text{INTEG}, X, Y\emptyset, P$$

gdzie: INTEG — kod operacji całkowania, Y — adres wyniku, X — adres argumentu, YØ — adres warunku początkowego, P — adres początku obszaru roboczego danego rozkazu, czy też

Y ← +, X1, X2

gdzie: + — kod operacji dodawania, Y — adres wyniku, X1, X2 — adresy operandów.

W programie półskompilowanym operandy są adresowane względnie w poszczególnych klasach. Rozróżniono następujące klasy operandów:

1. stałe rzeczywiste,
2. zmienne modelu,
3. parametry proste,
4. tablice,
5. obszary robocze dla funkcji uzależnionych czasowo,
6. parametry zanegowane,
7. zmienne robocze lokalne,
8. zmienne robocze globalne,
9. generowane zmienne stanu.

Operandy klas 7—9 wprowadzono dla potrzeb generacji kodu i w związku z dążeniem do optymalnego wykorzystania pamięci. W przypadku bezpośredniej generacji kodu (z adresacją bezwzględną w pamięci maszyny SIM), każdy operand i wynik operacji byłby obiektem globalnym. W praktyce znaczna część opisu modelu zawiera podwyrażenia skonstruowane przy użyciu operatorów i funkcji natychmiastowych, które mogą być wartościowane przy użyciu zmiennych roboczych określonych lokalnie (w bloku kodu) i zorganizowanych w stos.

Dla potrzeb wyznaczania sekwencji obliczalnej generowane, w czasie translacji, instrukcje maszyny abstrakcyjnej grupowane są w tak zwane bloki kodu. Blok kodu jest skończonym ciągiem instrukcji, dla którego zdefiniowany jest zbiór zmiennych wejściowych (wyznaczanych przez inne bloki kodu) oraz jedna zmienna wyjściowa, której wartość obliczana jest w ostatniej instrukcji bloku. Zmienna wyjściowa (jak i wszystkie wejściowe) musi być jedną z trzech następujących

a. zmienną źródłowego opisu modelu,
b. zmienną roboczą globalną (jest to dodana, przez translator, zmienna dla bloku wartościującego argument funkcji z pamięcią),
c. generowaną zmienną stanu (jest to dodana, przez translator, zmienna wyjściowa dla bloku realizującego funkcję z pamięcią).

Jedna instrukcja opisu modelu może być skompilowana na więcej niż jeden blok kodu, ale każdy blok jest tłumaczeniem co najwyżej jednej instrukcji modelu. Przykładowo, dla instrukcji opisu modelu

Y = INTEG(A * X+B * SIN(T)); Ø)

wygenerowany zostanie następujący kod

blok 1: $\xi_1 \leftarrow *, A, X$
$\xi_2 \leftarrow SIN, T$
$\xi_2 \leftarrow *, B, \xi_2$
$\$_1 \leftarrow +, \xi_1, \xi_2$
blok 2: $\pounds_1 \leftarrow$ INTEG, $\$_1$, = Ø, P_1
blok 3: $Y \leftarrow \pounds_1$

gdzie: $\xi_{1,2}$ — zmienne robocze lokalne, $\$_1$ — zmienna robocza globalna, $\pounds_1$ — generowana zmienna stanu. W trakcie generacji kodu dla pojedynczej instrukcji opisu modelu dokonywane jest jego ulepszenie. Polega ono na eliminacji nadmiarowych przesłań, usuwaniu zbędnych rozkazów (na przykład negacji stałej lub parametru) oraz na scalaniu bloków kodu, które w sekwencji obliczalnej będą musiały następować jeden po drugim.

W naszym przykładzie możliwe jest scalenie bloków 2 i 3 w blok 2′: Y ← INTEG, $\$_1$, = Ø, P_1.

3.3. Wyznaczanie sekwencji obliczalnej

Generację kodu wynikowego istotnie komplikuje algorytm numerycznego rozwiązywania układów równań różnicowych, do jakiego sprowadza się model cyfrowy symulowanego systemu.

Niech wektor $x(t)$ określa wartości zmiennych opisujących stan modelu (to jest będących wyjściami dla funkcji z pamięcią). Wówczas układ równań różnicowych (implikowany, na przykład, przez metodę całkowania numerycznego równań różniczkowych zwyczajnych) ma postać

$$x(t_{k+1}) = \varphi(f(x(t_k))), \quad k = \varnothing, 1, 2, \ldots \tag{1}$$

gdzie: f — reprezentuje wartościowanie argumentów funkcji z pamięcią, φ — jest operacją determinowaną przez wartościowanie funkcji z pamięcią.

Rozwiązanie równania (1) wymaga wartościowania f i φ w następującej kolejności

1. $s \leftarrow f(x)$
2. $x \leftarrow \varphi(s)$

gdzie s jest wektorem zmiennych przechowujących wartości argumentów funkcji z pamięcią (w procesorze języka GODYS-5 — zmiennych roboczych globalnych).

Zauważmy, że nieproceduralny opis modelu i jego dekompozycja na ciąg bloków kodu powoduje, że operacja f nie jest wprost określona, lecz musi być syntetyzowana w formie sekwencji obliczalnej.

Opis modelu, podany przez użytkownika w programie w języku GODYS-5, implikuje pewien graf, zwany grafem strukturalnym modelu. Węzłami tego grafu są zmienne występujące w opisie modelu, natomiast łuki odpowiadają zależnościom między zmiennymi. Ściślej, jeśli w wyrażeniu definiującym zmienną y występuje zmienna x, to węzeł x jest poprzednikiem węzła y w grafie.

W trakcie dekompozycji opisu modelu na bloki kodu (w fazie generacji kodu półskompilowanego) do grafu dodawane są nowe węzły, odpowiadające generowanym zmiennym stanu oraz globalnym zmiennym roboczym. Jeśli w opisie modelu pojawiła się instrukcja

$$y = F(x_1, \ldots, x_k)$$

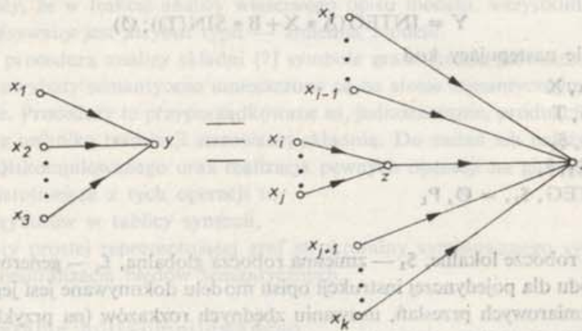
dodano węzeł z przy jej analizie, to

$$z = G(x_i, x_{i+1}, \ldots, x_j), \quad 1 \leqslant i \leqslant j \leqslant k$$

gdzie G reprezentuje podwyrażenie F, oraz

$$y = F_1(x_1, \ldots, x_{i-1}, z, x_{j+1}, \ldots, x_k)$$

gdzie F_1 powstaje przez zastąpienie podwyrażenia G zmienną z w F. Powyższe ilustruje rys. 3. Oczywiste jest, że przy zachowaniu podanej reguły dekompozycja opisu modelu nie zwiększa liczby cykli w grafie strukturalnym, określonym dla zmiennych modelu, a jedynie wydłuża niektóre cykle.



Rys. 3. Graf z wprowadzonym węzłem reprezentującym wartość podwyrażenia

Algorytm rozwiązywania układu równań różnicowych wylicza zależność, która, w grafie strukturalnym, przedstawiana jest łukami dochodzącymi do węzłów — odpowiedników bloków dla funkcji z pamięcią. Zatem dla wyznaczenia sekwencji obliczalnej łuki te można pominąć. Jeśli rozważany model jest sprowadzalny do układu równań różnicowych, to wspomniane usunięcie łuków powoduje przecięcie wszystkich

cykli grafu. W grafie acyklicznym można podać takie uporządkowanie węzłów, by każdy z nich występował po swoich poprzednikach. Co więcej, można z góry założyć, że w tym uporządkowaniu węzły zmiennych stanu są na początku, a węzły zmiennych roboczych globalnych jako ostatnie i że w każdym z tych dwóch typów kolejność jest dowolna. Istotne zatem jest tylko uporządkowanie pozostałych wierzchołków grafu. Ponieważ każdy węzeł reprezentuje zmienną wyjściową bloku, więc przedstawione uporządkowanie wierzchołków determinuje sekwencję obliczalną bloków kodu. Wyznaczenie jej realizowane jest zgodnie z przedstawionym w Dodatku algorytmem, tu naszkicowane są jedynie niektóre problemy jego implementacji.

Wygenerowane, w fazie analizy semantycznej, niepuste bloki kodu zapisywane są na pliku w pamięci masowej. W takiej samej kolejności zapisywane są, w pamięci operacyjnej, w postaci listy prostej wiązanej, odpowiadające im bloki charakterystyk węzłów generowanego grafu strukturalnego. Każdy z nich zawiera
— odsyłacz do następnego bloku w liście (w Dodatku — wartość funkcji NEXT dla węzła),
— numer bloku w kolejności generowania,
— numer linii programu źródłowego,
— długość kodu,
— typ bloku,
— adres względny zmiennej wyjściowej bloku (zero — dla zmiennych generowanych),
oraz, w zależności od typu bloku,
— ilość węzłów w otoczeniu danego węzła,
— adresy względne zmiennych modelu należących do otoczenia.

Rozróżnia się trzy typy bloków:

typ 1: bloki, których zmienna wyjściowa określona jest *a priori* (na przykład przez stałe lub parametry) lub jest zmienną stanu definiowaną przez funkcję z pamięcią,

typ 3: bloki obliczające wartości globalnych zmiennych roboczych — argumentów funkcji z pamięcią,

typ 2: pozostałe bloki (tylko one zawierają informację o otoczeniu, niezbędną przy wyznaczaniu sekwencji obliczalnej).

W konstruowanym grafie blokom typu 1 odpowiadają węzły źródłowe, blokom typu 3 — węzły ściśle spływowe.

Algorytm wyznaczania sekwencji obliczalnej realizowany jest w formie operacji na zdefiniowanej tu liście prostej charakterystyk.

W przypadku modeli niesprowadzalnych do układu równań różnicowych, w grafie strukturalnym pojawiają się cykle nie zawierające węzłów reprezentujących zmienne definiowane przez funkcje z pamięcią, tak zwane pętle uwikłane. Pętle te związane są z występowaniem w modelu matematycznym systemu równań lub układów równań algebraicznych. Ogólnie możliwe jest wtedy zastosowanie jednej z następujących metod:

1. taka modyfikacja modelu, by wspomniane pętle nie wystąpiły,

2. iteracyjne rozwiązywanie, na każdym kroku całkowania, pojedynczych równań algebraicznych, implikowanych przez niespójne pętle,

3. iteracyjne rozwiązywanie, na każdym kroku całkowania, układu równań algebraicznych implikowanego przez pętle.

W implementacji procesorów symulacyjnych najczęściej stosowane jest rozwiązanie (1). Przyczyną jest fakt, że pojawienie się pętli uwikłanych jest, w znacznej części przypadków, wynikiem błędów programowania, a w większości pozostałych przypadków istnieje równoważny model nie zawierający wspomnianych pętli lub, w celu ich likwidacji, wystarczy wprowadzenie do modelu operacji z pamięcią, zwanej izolacyjną [8], i realizującej ekstrapolację.

Rozwiązania (2) i (3) są znacznie ogólniejsze, charakteryzują się jednak niską efektywnością.

W języku GODYS-5 przyjęto rozwiązanie (1), a do zestawu funkcji wprowadzono operację izolacyjną — BLOCK.

3.4. Generacja programu wynikowego

Po wyznaczeniu sekwencji obliczalnej możliwe jest utworzenie programu wynikowego w pamięci maszyny SIM (patrz rys. 4). W trakcie generacji tego programu dokonywany jest

1. podział pamięci pomiędzy poszczególne klasy operandów, z wyznaczeniem odpowiednich adresów bazowych (relatywizatorów),

id zastrzeżone
id zmiennych obserwowanych
id parametrów
id tablic
nagłówki tablic
adresy względne parametrów zanegowanych
informacje o strukturze programu źródłowego
program dla operacji bez pamięci
program dla operacji z pamięcią
obszar niewykorzystany
generowane zmienne stanu
zmienne robocze globalne
zmienne robocze lokalne
parametry zanegowane
obszary robocze
tablice
parametry proste
zmienne modelu
stałe typu rzeczywistego

Rys. 4. Struktura programu wynikowego w pamięci maszyny abstrakcyjnej SIM

2. wypełnienie obszaru identyfikatorów,

3. wypełnienie obszaru nagłówków zadeklarowanych tablic (postaci: adres względny początku tablicy, długość, wypełnienie),

4. wypełnienie obszaru adresów względnych parametrów zanegowanych,

5. wypełnienie obszaru informacji o strukturze programu źródłowego, w którym dla każdego bloku, w sekwencji obliczalnej, przechowywane są: długość bloku i numer linii programu źródłowego, z której ten blok powstał,

6. utworzenie kodu wynikowego.

Kod wynikowy tworzony jest z kodu półskompilowanego poprzez uporządkowanie bloków, zgodnie z sekwencją obliczalną, i zastąpienie adresów względnych operandów adresami bezwzględnymi w pamięci maszyny SIM.

4. Struktura systemu wykonawczego

System wykonawczy języka GODYS-5 składa się z następujących modułów:

1. translator dyrektyw symulacji (skonstruowany w oparciu o parser precedensyjny prosty),
2. interpreter realizujący maszynę SIM,
3. moduł wyprowadzania wyników.

Kluczowy wpływ na efektywność symulacji ma realizacja maszyny SIM. W prezentowanym rozwiązaniu maszynę tę implementowano przez interpreter. Dało to duże możliwości diagnostyczne, w niewielkim tylko stopniu obniżając efektywność (w stosunku do bezpośredniego wykonywania rozkazów maszyny rzeczywistej). Przyczyną jest duża złożoność operacji maszyny SIM, a w konsekwencji mała wartość współczynnika T_0/T_v, gdzie: T_0 — czas wykonania czynności organizacyjnych związanych z interpretacją rozkazu maszyny abstrakcyjnej, T_v — średni czas wykonania rozkazu tej maszyny. W przypadku bezpośredniego wykonywania rozkazów maszyny rzeczywistej współczynnik ten jest równy zeru.

W projekcie interpretera niską wartość współczynnika T_0/T_v starano się dodatkowo zapewnić przez

1. przyspieszenie rozpakowywania rozkazów maszyny SIM poprzez umieszczenie każdego ich elementu (kodu operacji, adresu operanda) w jednym słowie maszyny rzeczywistej,
2. zminimalizowanie kontroli na etapie wykonania.

Wymagało to podziału interpretera na trzy podmoduły:

a. kontroli poprawności parametrów operacji,
b. ustawiania warunków początkowych,
c. właściwej symulacji.

W trakcie właściwej symulacji kontrolowane są jedynie

— wskaźnik przepełnienia zmiennopozycyjnego,
— poprawność argumentów funkcji SQRT, LOG i innych.

W przypadku wykrycia błędu wykonania następuje przerwanie cyklu interpretacyjnego i wydanie diagnozy, wskazującej przyczynę tego błędu i lokalizującej jego położenie w programie źródłowym. Jest to możliwe dzięki zachowaniu informacji o strukturze programu źródłowego, pozwalającej odwzorować adresy w programie maszyny SIM w numery linii programu źródłowego.

5. Podsumowanie

Przedstawiona w niniejszej pracy realizacja języka symulacyjnego charakteryzuje się przejrzystością i efektywnością. Modularność procesora i oparcie go o schemat translacji sterowanej składnią umożliwia łatwe modyfikowanie języka i samego procesora, a tym samym dalsze dostosowanie do szczególnych potrzeb użytkownika.

Ponad półroczne doświadczenia w użytkowaniu języka i jego procesora (dla celów badawczych i dydaktycznych) pokazały, że jest on dobrym narzędziem symulacyjnym, o zakresie zastosowań istotnie większym od modelowania systemów pomiarowych.

Implementation of the Language for the Simulation of Continuous Systems with Discontinuities

The paper presents the implementation of the GODYS-5 simulation language. This language enables the simulation of continuous systems with discontinuities and is adjusted to the modeling of complex measurement systems.

The processor of the language consists of two modules. One of them is the syntax-directed translator, which generates the object program in the language of some abstract machine. This machine is implemented by the effective interpreter, which is the main part of the second module.

All of the essential aspects of the processor realization are discussed, in particular the problem of the determination of computing sequence of operations.

Реализация языка для моделирования непрерывных систем с прерываниями

Статья представляет метод реализации процессора языка GODYS-5. Этот язык предназначен для моделирования непрерывных динамических систем с прерываниями, особенно для моделирования сложных измерительных систем. В состав процессора языка GODYS-5 входят две

программы; первая из них это синтаксически управляемый транслятор, создающий исходную программу на языке некоторой абстрактной машины. Эта машина реализирована эффективным интерпретером, который является основной частью второй программы. В статье обсуждены все существенные проблемы реализации процессора, а особенно проблема определения правильного порядка операции в исходной программе.

Literatura

[1] A. V. Aho, J. D. Ullman, *The Theory of Parsing, Translation and Compiling*, vol. I: *Parsing*, vol. II: *Compiling*, Prentice-Hall, Englewood Cliffs, New Jersey 1973.

[2] Y. Chu, *Digital Simulation of Continuous Systems*, Mc Graw-Hill, New York 1969.

[3] D. Gries, *Compiler Construction for Digital Computers*, Wiley, New York 1971.

[4] J. L. Hay, R. E. Crosbie, R. I. Chapin, *Integration Routines for System with Discontinuities*, The Computer Journal, vol. 17, 1974, 3, pp. 275-278.

[5] W. Jentsch, *Digitale simulation kontinuierlicher systeme*, R. Oldenbourg Verlag, München—Wien 1969.

[6] J. Król, *GODYS-5 — język do symulacji układów ciągłych ze zdarzeniami dyskretnymi*, Podstawy Sterowania, t. 9, z. 3, 1979.

[7] J. Król, J. Lembas, J. Rosek, *Analiza składni oparta o gramatykę precedensyjną prostą*, Podstawy Sterowania, 7, 1977, z. 2, pp. 149-164.

[8] G. K. Steel, *Programming of Digital Computers for Transient Studies in Control Systems*, Int. J. Electr. Eng. Educ., vol. 3, 1965, pp. 261-278.

DR INŻ. J. KRÓL, MGR J. KURAŚ, MGR J. LEMBAS, MGR M. ŚLUSAREK
INSTYTUT INFORMATYKI, UNIWERSYTET JAGIELLOŃSKI
UL. KOPERNIKA 27, 31-501 KRAKÓW, POLAND

DODATEK

Algorytm wyznaczania sekwencji obliczalnej

Dla ścisłości wywodu przytoczymy kilka podstawowych definicji teorii grafów skierowanych.

DEFINICJA 1

Skończonym grafem skierowanym (krótko: grafem) nazywamy parę $G = (X, R)$, gdzie X — jest niepustym skończonym zbiorem węzłów, R — jest relacją binarną w zbiorze X.

Niech $G = (X, R)$ będzie grafem.

DEFINICJA 2

Listę sąsiedztwa (listą poprzedników, otoczeniem) węzła x nazywamy zbiór

$$\mathrm{LP}(x) = \{y\colon (y, x) \in R\}.$$

DEFINICJA 3

Drogą w grafie G nazywamy ciąg węzłów $(x_0, x_1, \ldots, x_n)$, gdzie $n \geqslant 1$ oraz $(x_i, x_{i+1}) \in R$, $i = 0, \ldots, n-1$. Drogę $(x_0, x_1, \ldots, x_n)$ nazywamy zamkniętą (cyklem), gdy $x_0 = x_n$.

DEFINICJA 4

Graf nazywamy acyklicznym, jeżeli nie istnieje w nim droga zamknięta.

DEFINICJA 5

Cięciem zerowego rzędu w grafie G nazywamy dowolny podzbiór $\mathcal{F}_0$ zbioru X, taki, że graf $G_1 = (X \setminus \mathcal{F}_0, R|_{X \setminus \mathcal{F}_0})$ jest acykliczny.

Załóżmy, że węzły grafu G (utożsamiane z liczbami $1, \ldots, n$; $n = \# X$) uporządkowane są liniowo przez pewną permutację cykliczną

$$\mathrm{NEXT}: \{\varnothing, 1, \ldots, n\} \to \{\varnothing, 1, \ldots, n\}$$

rzędu $n+1$, przy czym

1. $\mathrm{NEXT}(\varnothing) = k$ oznacza, że węzeł k jest pierwszy w tym uporządkowaniu,
2. jeśli węzeł k jest i-ty w tym uporządkowaniu oraz $\mathrm{NEXT}(k) = j$, to węzeł j jest $i+1$-szy w tym uporządkowaniu,
3. $\mathrm{NEXT}(k) = \varnothing$ oznacza, że k jest n-ty w tym uporządkowaniu.

Oznaczmy przez NEXTØ zacieśnienie funkcji NEXT, traktowanej jako relacja, do zbioru $\{1, \ldots, n\}$, a przez $\mathrm{NEXT}\varnothing^{+}$ — przechodnie domknięcie NEXTØ.

W fazie projektowania translatora języka GODYS-5 pojawił się następujący problem. Przy pewnym wyborze podzbioru węzłów T ustalić nowe uporządkowanie węzłów przez podanie funkcji ORD zdefiniowanej w podobny sposób jak NEXT, której odpowiadająca relacja $\mathrm{ORD}\varnothing^{+}$ miałaby następujące własności

i) $\qquad \forall x_i \in T, \quad x_j \in X \setminus T: (x_i, x_j) \in \mathrm{ORD}\varnothing^{+}$

(*ii*) $\qquad \forall x_i \in X \setminus T: \mathrm{LP}(x_i) \subseteq \{x: (x, x_i) \in \mathrm{ORD}\varnothing^{+}\}$.

Powstaje pytanie, jakie warunki musi spełniać zbiór T, by taka relacja istniała. Odpowiedzią na nie jest następujące twierdzenie.

TWIERDZENIE

Aby dla danego podzbioru T zbioru węzłów grafu istniała funkcja ORD o własnościach (*i*), (*ii*) potrzeba i wystarcza, by T był cięciem zerowego rzędu w grafie G.

Dowód

Oznaczmy relację $\mathrm{ORD}\varnothing^{+}$ symbolem $\lessdot$.

1. Jeśli T nie jest cięciem zerowego rzędu, to istnieje cykl $(x_0, x_1, \ldots, x_n = x_0)$ taki, że $x_i \notin T$, $i = 1, \ldots, n$. Załóżmy dla dowodu nie wprost, że istnieje funkcja ORD o żądanych własnościach. Ponieważ $x_{n-1} \in \mathrm{LP}(x_0)$, więc zachodzi $x_{n-1} \lessdot x_0$. Z analogicznego powodu $x_{n-2} \lessdot x_{n-1}$ itd., w końcu $x_0 \lessdot x_1 \lessdot \ldots \lessdot x_0$, co jest sprzeczne z własnością liniowego porządku relacji $\mathrm{ORD}\varnothing^{+}$. Zatem dla istnienia funkcji ORD warunek, że T jest cięciem zerowego rzędu, jest konieczny.

2. Załóżmy, że $T = \{x_1, \ldots, x_k\}$ jest cięciem zerowego rzędu w grafie. Kładziemy, z definicji, $\mathrm{ORD}(\varnothing) = x_1$, $\mathrm{ORD}(x_i) = x_{i+1}$, $i = 1, \ldots, k-1$. Następnie wybieramy dowolny węzeł $x_{k+1} \in X \setminus T$. Jeśli $\mathrm{LP}(x_{k+1}) \subseteq T$, to kładziemy

$$T \leftarrow T \cup \{x_{k+1}\}, \quad \mathrm{ORD}(x_k) = x_{k+1}, \quad k \leftarrow k+1 \tag{1}$$

Jeżeli inkluzja nie zachodzi, to dokonujemy podstawienia za x_{k+1} dowolnego węzła z jego otoczenia i jeśli inkluzja $\mathrm{LP}(x_{k+1}) \subseteq T$ ponownie nie zachodzi, powtarzamy tę czynność aż zawieranie zajdzie i wówczas wykonujemy operację (1). Proces ten kończy się, bo graf $(X \setminus T, R|_{X \setminus T})$ nie ma cykli i jest skończony. Następnie powracamy do wyboru nowego węzła i powtarzamy całą procedurę. W ten sposób przy każdym jej wykonaniu definiujemy funkcję ORD dla kolejnego węzła. Ilość wierzchołków jest skończona, więc procedura ta porządkuje cały zbiór $X \setminus T$, co dowodzi dostateczność warunku twierdzenia.

Podany dalej algorytm A konstruuje funkcję ORD przy nieco rozszerzonych założeniach. Zakłada on istnienie dwóch predykatów p1 i p3, takich że

$$\mathrm{p}\,1(x) \Leftrightarrow x \in T,$$
$$\mathrm{p}\,3(x) \Rightarrow \forall y \in X: \big(x \in \mathrm{LP}(y) \Rightarrow \mathrm{p}\,1(y)\big)$$

czyli predykat p 3 oznacza możliwość wstawienia węzła na końcu w tworzonym uporządkowaniu ORD. Algorytm niszczy funkcję NEXT, a w razie niespełnienia warunku twierdzenia wykonuje procedurę ERROR. Zbiór SET jest zbiorem pomocniczym. Część A1 algorytmu wykrywa, w zbiorze węzłów, wierzchołki, które spełniają predykat p 1 lub p 3 i osobno je porządkuje, używając, w celu uniknięcia kolizji, zmiennej first1 dla przechowania pierwszego z węzłów spełniających p 1 i odpowiednio first3 dla spełniających p 3.

Algorytm A (wyznaczenia sekwencji obliczalnej)

```
(* A1 *) SET := Φ;
         last1 := last2 := last 3:= first1 = first3 := Ø;
         ia := NEXT[Ø];
         while ia ≠ Ø do
         begin
         nxt := NEXT[ia];
         if not (p1(ia)) or (p3(ia)) then last2 := ia else
            begin
            NEXT[last2] := nxt; ORD[ia] := Ø;
            if p 1(ia) then
               begin
               if first1 = Ø then first1 := ia else ORD[last1] := ia;
               last1 := ia; SET := SET ∪ {ia}
               end else
               begin
               if first3 = Ø then first3 := ia else ORD[last3] := ia;
               last3 := ia
               end
            end;
            ia := nxt
            end;
(* A2 *) if NEXT[Ø] ≠ Ø then
         repeat found := false; ia := NEXT[Ø]; last2 := Ø;
         while ia ≠ Ø do
            begin
            nxt := NEXT[ia];
            if LP(ia) ⊆ SET then
               begin
               NEXT[last2] := nxt; found := true;
               ORD[ia] := Ø; ORD[last1] := ia;
               last1 := ia; SET := SET ∪ {ia}
               end else last2 := ia;
               ia := nxt
               end
         until NEXT[Ø] = Ø or not found;
(* A3 *) if NEXT[Ø] ≠ Ø then ERROR else ORD[last1] := first3.
```

W stosunku do realizacji algorytmu A w translatorze języka GOD YS-5 sformułować można następujące uwagi:

1. Postać algorytmu A jest pewną formalizacją operacji na listach prostych. W terminologii struktur danych algorytm ten dzieli listę wejściową na trzy podlisty, według predykatów spełnianych przez poszczególne elementy, a następnie sortuje listę 2, dołączając jej elementy do listy 1 w takiej kolejności, by spełniony był warunek *(ii)*; końcową operacją jest konkatenacja list 1 i 3. Funkcje NEXT i ORD podają wartości odsyłaczy do kolejnych elementów w listach.

2. Predykat p 1 spełniają węzły reprezentujące bloki typu 1 (realizujące operacje z pamięcią) natomiast p 3 spełniają bloki typu 3 (definiujące globalne zmienne robocze).

3. Akcja ERROR wykonywana jest tylko wtedy, gdy w grafie modelu istnieje cykl nie zawierający funkcji z pamięcią. Najpierw usuwane są z listy 2 wszystkie węzły nie leżące na cyklach, a następnie wydawana jest diagnostyka podająca numery linii programu źródłowego, powodujących ten błąd.

4. Wartości funkcji NEXT i ORD przechowywane są w tych samych działkach elementów list.