Charakterystyka języka OCCAM

Wojciech Mikanik*

23 stycznia 1997 r.

1 Wstęp

Język OCCAM[1] jest proceduralnym językiem programowania współbieżnego opracowanym przez brytyjską firmę Inmos. Jego twórcy nazwali go tak na cześć średniowiecznego filozofa Wilhelma Ockhama, autora zasady zwanej „brzytwą Ockhama”, stwierdzającej, iż nie należy mnożyć bytów bez istotnej potrzeby. Zasada ta znalazła swe odzwierciedlenie w strukturze języka, którego prostota i elegancja pozwalają na tworzenie zwięzłych i efektywnych aplikacji, zarówno dla pojedynczych procesorów jak i systemów wieloprocesorowych. OCCAM z powodzeniem zastosowano m.in. do złożonych obliczeń numerycznych, przetwarzania i rozpoznawania obrazów, sterowania robotami, symulacji sieci neuronowych [6, 7].

OCCAM opracowywano równolegle z transputerami i oba produkty były od początku swego istnienia silnie ze sobą związane. Przez stosunkowo długi czas transputery były jedynymi procesorami, dla których była dostępna implementacja tego języka. W chwili obecnej sytuacja zmieniła się i dostępny jest on również na innych platformach.

Najnowszą wersją języka jest wersja 3.0 [1], która według informacji dostępnych dla autora w momencie pisania artykułu nie została jeszcze zaimplementowana. Dostępne są jedynie implementacje obejmujące podzbiór OCCAMu 3.0, określany jako OCCAM 2.1. W artykule przedstawiono charakterystykę języka OCCAM 2.0[2, 4, 5], ponieważ ta wersja jest szerzej stosowana w praktyce. W punkcie 5 omówiono niektóre nowe elementy języka wprowadzone w wersji 3.0.

2 Model programowania

Na projekt języka silny wpływ wywarła notacja CSP (ang. communicating sequential processes) [3]. Przyjętym modelem obliczeń jest zbiór asynchronicznie wykonujących się procesów współbieżnych, które komunikują się między sobą przy pomocy synchronicznie przesyłanych wiadomości (ang. message). Procesy mogą także pobierać wartość współdzielonych danych, co jest rozrzeszeniem w stosunku do notacji CSP.

3 Podstawy składni

3.1 Identyfikatory

Identyfikatory w języku OCCAM mają dowolną długość, rozpoczynają się od litery i zawierają litery, cyfry oraz znak kropki. Wielkie i małe litery są rozróżniane. Słowa kluczowe (zastrzeżone) pisane są zawsze wielkimi literami.

3.2 Typy

Każdy obiekt występujący w programie napisanym w języku OCCAM musi być określonego typu. Typem zmiennej lub stałej może być jeden z kilku typów podstawowych bądź typ tablicowy, który jest jedynym dostępnym rodzajem typów złożonych. Odrębnymi rodzajami obiektów są kanały, porty i zegary (ang. timer). Podstawowymi typami danych w języku OCCAM są:

`BOOL` typ logiczny;

`BYTE` typ bajtowy o wartościach z przedziału 0...255;

`INT` typ całkowity ze znakiem, którego wartości reprezentowane są w formacie umożliwiającym najbardziej efektywne dokonywanie operacji na tych wartościach w konkretnym systemie;

`INT16`, `INT32`, `INT64` typy całkowite ze znakiem, odpowiednio 16-, 32- i 64-bitowe, reprezentowane w formacie uzupełnienia do dwóch;

`REAL32`, `REAL64` 32- i 64-bitowe typy zmiennopozycyjne reprezentowane zgodnie ze standardem ANSI/IEEE 754–1985.

Precyzyjna definicja typów całkowitych i rzeczywistych — określająca sposób ich reprezentacji w pamięci oraz wykonywania operacji arytmetycznych — zapewnia, że obliczenia wykonywane na danych tych typów przebiegają w sposób identyczny (choć może z różną szybkością) we wszystkich systemach. Ułatwia ona także

*Instytut Informatyki Politechniki Śląskiej

[1] OCCAM jest znakiem towarowym firmy Inmos Ltd.

komunikację między procesami wykonywanymi w procesorach różniących się długością słowa czy sposobem reprezentacji danych.

Tablicę definiuje się podając liczbę jej elementów oraz ich typ, jak w poniższym przykładzie (wszystkie deklaracje w języku OCCAM zakończone są znakiem dwukropka; znaki „--" rozpoczynają komentarz, który rozciąga się do końca wiersza):

```
[10] INT a: -- tablica 10-ciu liczb typu INT
```

Tablice indeksowane są od 0. Korzystając z procesu przypisania możliwe jest przypisywanie wartości całym tablicom (np. `a := b`), o ile ich rozmiary oraz typy bazowe tablic są identyczne. Możliwe jest także odwoływanie się do fragmentów tablic, co ilustruje poniższy przykład, w którym wartości z dolnej połowy zadeklarowanej powyżej tablicy `a` kopiowane są do jej górnej części:

```
[a FROM 5 FOR 5] := [a FROM 0 FOR 5]
```

Kanał jest jednokierunkowym połączeniem między dwoma procesami, o zdefiniowanym protokole określającym ilość i typ przesyłanych danych. Kanały nie mogą gromadzić danych, służą jedynie do ich przesyłania. Podczas kompilacji sprawdzana jest zgodność danych przesyłanych kanałem z jego protokołem. W trakcie kompilacji można również wykryć próbę przesyłania jednym kanałem danych w obu kierunkach lub też przydzielenie jednego kanału więcej niż dwóm procesom. Ogólna postać deklaracji kanału jest następująca:

`CHAN OF` *protokół nazwa_kanału*`:`

Poniżej przedstawiono deklarację kanału o nazwie `Numer`, którym są przesyłane pojedyncze liczby całkowite:

```
CHAN OF INT Numer:
```

Porty i zegary omówione są w pkt. 4, poświęconym cechom języka istotnym przy programowaniu systemów czasu rzeczywistego.

W języku OCCAM nie ma odrębnego typu znakowego. Znaki ASCII reprezentowane są przez dane typu `BYTE`, a napisy — przez tablice bajtowe. Literały znakowe zapisywane są z użyciem apostrofu, np: `'A'`, `'1'`, zaś literały napisowe z pomocą cudzysłowu, np: `"OCCAM"`.

Zakresem widoczności deklaracji jest proces bezpośrednio po niej następujący.

3.3 Wyrażenia

W języku OCCAM istnieje możliwość tworzenia wyrażeń o dowolnym z istniejących typów danych. Oprócz standardowych operatorów arytmetycznych, logicznych, bitowych i relacyjnych, w języku zdefiniowane są operatory arytmetyki modulo. Istnieją także dwa operatory pozwalające dla dowolnego typu całkowitego określić wartość największej dodatniej (operator `MOSTPOS`) i najmniejszej ujemnej (operator `MOSTNEG`) liczby należącej do tego typu.

Argumenty wyrażeń muszą być tego samego typu, który jest również typem całego wyrażenia. Konwersje typów należy specyfikować jawnie, np. przy pomocy jednego z dwóch przeznaczonych do tego celu operatorów.

Wszystkie operatory mają ten sam priorytet. Programista musi wskazać jawnie przy pomocy nawiasów okrągłych kolejność wartościowania wyrażeń złożonych. Brak nawiasów w wyrażeniach zawierających więcej niż jeden operator jest błędem składniowym.

3.4 Procesy elementarne

Programy w języku OCCAM zbudowane są z trzech rodzajów procesów elementarnych: przypisania, wejścia i wyjścia. Każdy proces musi być zapisany w osobnym wierszu w tekście programu.

Proces przypisania powoduje przypisanie nowej wartości wybranej zmiennej, np.

```
x := 10
```

Możliwe jest dokonanie przypisania wielokrotnego (ang. multiple assignment), jak w poniższym przykładzie

```
x, y := 1, 2
```

W procesie przypisania wielokrotnego najpierw określane są wartości wszystkich wyrażeń znajdujących się po prawej stronie operatora przypisania, a następnie wartości te przypisywane są odpowiadającym im zmiennym, tak jak to ma miejsce w zamieszczonym poniżej przykładzie, w którym dokonywana jest wymiana wartości między zmiennymi `a` i `b`:

```
a, b := b, a
```

Proces wejściowy powoduje nadanie zmiennej wartości pobranej z kanału. Na przykład proces:

```
k ? z
```

powoduje przypisanie zmiennej `z` wartości pobranej z kanału `k`.

Proces wyjściowy powoduje wysłanie wartości do kanału. Na przykład proces:

```
k ! w
```

powoduje wysłanie wartości wyrażenia `w` do kanału `k`.

Komunikacja w języku OCCAM jest synchroniczna, tzn. przesłanie wiadomości przez kanał następuje dopiero wtedy, gdy zarówno proces nadawca jak i proces odbiorca są gotowe do wykonania operacji. Wykonywanie procesu, który jako pierwszy osiągnie stan gotowości do przesłania wiadomości jest wstrzymywane do chwili, gdy drugi proces będzie gotowy do jej odebrania.

3.5 Procesy specjalne

Oprócz procesów elemetarnych zdefiniowano także dwa procesy specjalne. Proces `SKIP` odpowiada funkcjonalnie instrukcji pustej. Jego działanie polega na rozpoczęciu się, nie wykonaniu żadnych obliczeń i zakończeniu. Proces `STOP` rozpoczyna się, lecz nigdy się nie kończy.

3.6 Konstrukcje

Możliwe jest grupowanie procesów prostych (elementarnych i specjalnych) w procesy złożone nazywane konstrukcjami. Składowymi takich procesów mogą być również procesy złożone. Konstrukcje rozpoczynają się od jednego z następujących słów kluczowych: `SEQ`, `PAR`, `WHILE`, `IF`, `CASE` oraz `ALT`, po którym w kolejnych wierszach występują procesy składowe, wcięte o dwie spacje w stosunku do słowa kluczowego rozpoczynającego konstrukcję. Wcięcie określa, jakie procesy są składowymi konstrukcji i jest obowiązkowe. W ten sposób zapewnia się zgodność struktury logicznej programu z układem jego tekstu, co zwiększa czytelność programu.

Konstrukcja `SEQ` przeznaczona jest do specyfikowania ciągu procesów, które mają być wykonane sekwencyjnie, tak jak w konwencjonalnych językach programowania, np. w Pascalu.

Konstrukcja `PAR` służy do specyfikowania ciągu procesów, które mogą być wykonane równolegle (współbieżnie). Wykonanie procesów składowych rozpoczyna się w tej samej chwili. Wykonanie zamieszczonego poniżej fragmentu programu powoduje równoległe wprowadzenie liczb z kanałów `we1` oraz `we2`, a następnie wysłanie sumy wczytanych liczb do kanału `wy`. Kolejność wczytywania danych wejściowych nie jest określona.

```
SEQ
  PAR -- pierwszy proces skladowy konstrukcji SEQ
    we1 ? liczba1 -- procesy skladowe
    we2 ? liczba2 -- konstrukcji PAR
  wy ! liczba1 + liczba2 -- drugi proces skladowy
                         -- konstrukcji SEQ
```

Wykonanie konstrukcji `PAR` kończy się z chwilą zakończenia wykonywania ostatniego z procesów składowych.

Należy podkreślić, że w odróżnieniu od konwencjonalnych języków programowania, sekwencyjność wykonania fragmentów programu w języku OCCAM nie jest domyślna. Programista musi jawnie wskazać, czy procesy mają wykonać się sekwencyjnie czy równolegle.

Konstrukcja `IF` umożliwia warunkowe wykonanie jednego z procesów składowych. Każdy proces składowy poprzedzony jest wyrażeniem logicznym. W trakcie wykonania konstrukcji `IF` wyrażenia te są wartościowane w kolejności ich występowania w tekście programu, aż do znalezienia pierwszego, które ma wartość `TRUE`. Wykonywany jest wówczas proces składowy odpowiadający temu wyrażeniu. W poniższym przykładzie zmienna `sgn` typu `INT` przyjmuje wartość znaku zmiennej `x`, tj. `sign(x)`.

```
IF
  x < 0 -- warunek sprawdzany jako pierwszy
    sgn := -1
  x > 0 -- warunek sprawdzany jako drugi
    sgn := 1
  TRUE -- jesli poprzednie nie sa spelnione ...
    sgn := 0
```

Konstrukcja `CASE` grupuje dowolną liczbę procesów składowych, z których każdy poprzedzony jest listą złożoną z jednego lub kilku wyrażeń. Wykonywany jest proces poprzedzony wyrażeniem, które ma taką samą wartość jak zmienna wyboru. Jeżeli zgodność taka nie występuje, to wykonywany jest proces po słowie kluczowym `ELSE`. Wartości wyrażeń sprawdzane są w kolejności ich występowania w tekście programu. W poniższym przykładzie w zależności od wartości zmiennej `wariant`, wartość zmiennej `liczba` wyprowadzana jest do odpowiedniego kanału:

```
CASE wariant
  1
    kanal1 ! liczba
  2, 3
    kanal2 ! liczba
  -- jesli zmienna wariant
  -- ma wartosc rozna od 1, 2 i 3
  ELSE
    kanal3 ! liczba
```

Należy zwrócić uwagę na fakt, iż jeśli podczas wykonywania konstrukcji `IF` i `CASE` nie jest możliwe wybranie do wykonania któregoś z procesów składowych, to wówczas konstrukcje te są równoważne procesowi `STOP`.

Konstrukcja `WHILE` jest jedną z dwóch — obok omawianej w pkt. 3.9 wersji konstrukcji `SEQ` z replikatorem — instrukcji iteracyjnych w języku OCCAM. Jej semantyka jest taka sama jak np. instrukcji **while** w Pascalu. Przykład zastosowania pętli `WHILE` przedstawiono na rys. 2.

Konstrukcja `ALT` grupuje dowolną liczbę procesów składowych, z których każdy poprzedzony jest dozorem (ang. guard). Dozór zbudowany jest z wyrażenia logicznego i procesu wejściowego, oddzielonych znakiem '&', jak to ma miejsce w dozorze trzeciego procesu z rys. 1. Wyrażenie logiczne może być pominięte (por. dozór pierwszego procesu) i wówczas przyjmuje się, że ma ono wartość `TRUE`. Proces wejściowy w dozorze może zostać zastąpiony procesem specjalnym `SKIP`, jak to ma miejsce w przypadku dozoru drugiego procesu. Dozór jest prawdziwy w przypadku, gdy wyrażenie logiczne jest prawdziwe i możliwe jest pobranie danych przez proces wejściowy. W trakcie wykonania konstrukcji `ALT` sprawdzane są wszystkie dozory, a następnie spośród dozorów prawdziwych arbitralnie wybierany jest jeden z nich i wykonywany jest odpowiadający mu proces. Jeśli wyrażenia logiczne we wszystkich dozorach są fałszywe, to wykonanie konstrukcji `ALT` jest równoważne wykonaniu procesu specjalnego `STOP`. W przypadku gdy żaden z procesów wejściowych występujących w dozorach, których wyrażenia logiczne są prawdziwe, nie jest gotowy do wykonania, wykonywanie konstrukcji `ALT` zostaje wstrzymane do czasu osiągnięcia gotowości przez którykolwiek z nich. Konstrukcja `ALT` kończy

```
ALT
  -- dozor bez wyrazenia logicznego
  x2 ? x
    ... proces pierwszy
  -- dozor bez procesu wejsciowego
  (y > 10) & SKIP
    ... proces drugi
  -- dozor zawierajacy wyrażenie logiczne
  -- oraz proces wejsciowy
  (y = 0) & (x1 ? x)
    ... proces trzeci
```

Rys. 1: Przykład użycia konstrukcji `ALT`

się w momencie, gdy zakończy się proces wybrany do wykonania.

3.7 Podprogramy

W języku OCCAM są dwa rodzaje podprogramów: procedury i funkcje. Procedura jest nazwanym procesem i podobnie jak inne procesy może być wykonywana równolegle z innymi procesami, o ile nie modyfikuje zmiennych współdzielonych. Na rys. 2 przedstawiono procedurę, w której pobiera się liczby całkowite z kanału `we` do momentu, gdy napotkana zostanie pewna wartość `x`, jednocześnie zliczając otrzymane wiadomości w parametrze `ile`. Wartości parametrów podprogramów przekazywane są zawsze przez zmienną, tak jak parametr `ile`. W przypadku gdy istotna jest tylko początkowa wartość parametru i nie będzie ona modyfikowana — jak to ma miejsce w przypadku parametru `x` — można jego deklarację poprzedzić słowem kluczowym `VAL`. Taki parametr staje się wówczas lokalną stałą danego podprogramu.

```
PROC szukaj.x (VAL INT x, CHAN OF INT we, INT ile)
  BOOL wykonuj:
  SEQ
    ile, wykonuj := 0, TRUE
    INT nowy: -- zmienna lokalna konstrukcji WHILE
    WHILE wykonuj
      SEQ
        we ? nowy
        wykonuj, ile := nowy <> x, ile + 1
: -- w tej samej kolumnie, co slowo 'PROC'
```

Rys. 2: Procedura pomijania liczb przesyłanych kanałem `we` aż do napotkania wartości równej parametrowi `x` oraz zliczania liczby wykonanych pobrań z kanału

Funkcje w języku OCCAM są nazwanymi procesami, które pobierają wartości przekazane im jako parametry, wykonują na nich obliczenia i zwracają uzyskany rezultat lub rezultaty (dozwolone jest by funkcja zwracała więcej niż jeden wynik). Funkcje nie mogą mieć efektów ubocznych, tj. nie mogą modyfikować wartości swych parametrów, które muszą być deklarowane jako lokalne stałe, ani wartości zmiennych zewnętrznych widocznych w jej wnętrzu, choć mogą odczytywać ich wartość. Ciało funkcji nie może zawierać konstrukcji `PAR` oraz `ALT` ani procesów wyjściowych i wejściowych. Słowa kluczowe `VALOF` oraz `RESULT` stanowią nawiasy składniowe procesu tworzącego ciało funkcji. Funkcja `MaxMin` z rys. 3 wyznacza wartości minimalnego i maksymalnego elementu w nieposortowanej tablicy. Sposób deklaracji parametru `tab` ilustruje ważną cechę języka OCCAM. Jeśli parametrem podprogramu jest tablica, to jego deklaracja nie musi określać jej rozmiaru, tak jak to ma miejsce w przykładzie z rys. 3. Dzięki temu zabiegowi funkcję `MaxMin` można stosować do wszystkich jednowymiarowych tablic liczb typu `INT`, niezależnie od rozmiaru tych tablic. Liczbę elementów w tablicy będącej parametrem aktualnym funkcji wyznacza się przy pomocy operatora `SIZE`.

Rekurencyjne wywoływanie podprogramów nie jest dozwolone. Deklaracje procedur i funkcji mogą być zagnieżdżone.

3.8 Protokoły kanałów

W pkt. 3.2 przedstawiono już sposób deklarowania kanału o nieskomplikowanym protokole. Poniżej przedstawiono kilka możliwych definicji protokołów i deklaracji kanałów.

```
INT, INT FUNCTION MaxMin (VAL []INT tab)
  INT max, min:
  VALOF
    INT i:
    SEQ
      max, min := tab[0], tab[0]
      i := 1
      WHILE i < (SIZE tab)
        SEQ
          IF
            tab[i] > max
              max := tab[i]
            tab[i] < min
              min := tab[i]
            TRUE
              SKIP
          i := i + 1
    RESULT max, min
:
```

Rys. 3: Funkcja wyznaczania wartości minimalnego i maksymalnego elementu w nieposortowanej tablicy

```
PROTOCOL Wspolrzedne IS BYTE; INT32:
PROTOCOL ZmiennaPorcja IS INT::[]REAL32:
CHAN OF Wspolrzedne Dane:
```

Dwa pierwsze wiersze zawierają definicje nowych protokołów. Pojedyncza wiadomość zgodna z protokołem `Wspolrzedne` składa się z dwóch części, z których pierwsza jest bajtem, a druga 32-bitową liczbą całkowitą ze znakiem. Protokół `ZmiennaPorcja` umożliwia przesyłanie wiadomości o różnych długościach. Pierwsza część wiadomości jest liczbą całkowitą równą rozmiarowi tablicy przesyłanej jako jej druga część. Deklaracja w trzecim wierszu określa, że w kanale `Dane` obowiązuje protokół `Wspolrzedne`. W przykładzie przedstawionym na rys. 4 określenie dla kanału `Beztypowy` protokołu `ANY` umożliwia przesyłanie tym kanałem danych dowolnych typów. W trakcie kompilacji nie sprawdza się, czy dane wysyłane i odbierane kanałem o takim protokole są identycznego typu; pełną odpowiedzialność za sensowność przesłania ponosi programista. Protokół został wprowadzony do języka po to, aby ułatwić programowanie operacji wejścia/wyjścia i raczej należy używać go tak rzadko, jak to tylko możliwe.

Ostatnim rodzajem protokołów, które omówimy, są protokoły wariantowe, czyli zbiory różnych protokołów. Każdy protokół składowy jest identyfikowany przez jego pierwszy element nazywanym znacznikiem (ang. tag). Poniżej zamieszczono definicję protokołu `Zlecenie` obejmującego cztery warianty:

```
CHAN OF ANY Beztypowy:
PAR
  SEQ
    Beztypowy ! 'a'
    Beztypowy ! 'b'
  INT16 x:
  Beztypowy ? x
```

Rys. 4: Ilustracja własności kanału z protokołem `ANY`

```
PROTOCOL Zlecenie
  CASE
    czysc
    kolor; INT16; BOOL
    punkt; INT; INT
    wypelnij; INT; INT; INT; INT
:
```

Użycie zdefiniowanego powyżej protokołu zademonstrowano na przykładzie procedury sterującej monitorem graficznym o rozdzielczości 1024×1024 i o 65536 kolorach (rys. 5). W procedurze pobiera się wiadomości z kanału `DoMonitora` i wykonuje jedną z czterech operacji w zależności od wariantu otrzymanej wiadomości:

`czysc` — wypełnienie całego ekranu bieżącym kolorem tła;

`kolor` — określenie bieżącego koloru tła lub „pióra", zależnie od wartości drugiej części wiadomości;

`punkt` — zamalowanie punktu bieżącym kolorem „pióra";

`wypelnij` — wypełnienie prostokątnego fragmentu ekranu bieżącym kolorem „pióra".

3.9 Replikatory

Konstrukcje `ALT`, `IF`, `PAR` oraz `SEQ` mogą występować w wariantach z replikatorami, które umożliwiają tworzenie tablicy procesów. Rodzaj konstrukcji determinuje sposób wykonania procesów należących do tablicy. Umożliwiając skrócenie tekstu programu, replikatory zwiększają jego przejrzystość i ułatwiają jego pielęgnację. Konstrukcja `SEQ` z replikatorem odpowiada funkcjonalnie instrukcji iteracyjnej **for** w języku Pascal. Przykład jej użycia przedstawiono na rys. 5, gdzie została zastosowana do nadania wartości wszystkim elementom tablicy `Ekran`. Współbieżna wersja tego fragmentu programu wygląda następująco:

```
PAR i = 0 FOR X
  PAR j = 0 FOR X
    Ekran[i][j] := KolorTla
```

```
PROC Monitor (CHAN OF Zlecenie DoMonitora)
  VAL X IS 1024: -- deklaracja stalej calkowitej X
  [X][X] INT16 Ekran:
  INT16 KolorTla, KolorPiora:
  INT x, y, dx, dy:
  SEQ
    -- 0 jest typu INT, a 0(INT16) jest typu INT16
    KolorTla, KolorPiora := 0(INT16), 0(INT16)
    WHILE TRUE -- proces nieskonczony
      -- wariantowy proces wejściowy
      DoMonitora ? CASE
        czysc -- wariant pierwszy
          SEQ i = 0 FOR X -- konstrukcja SEQ
            SEQ j = 0 FOR X -- z replikatorem
              Ekran[i][j] := KolorTla
        BOOL tlo: -- zmienne lokalne dla kodu
        INT16 pom: -- obslugi wariantu kolor
        kolor; pom; tlo -- wariant drugi
          IF
            tlo
              KolorTla := pom
            TRUE
              KolorPiora := pom
        punkt; x; y -- wariant trzeci
          Ekran[x][y] := KolorPiora
        wypelnij; x; y; dx; dy -- wariant czwarty
          SEQ i = x FOR dx
            SEQ j = y FOR dy
              Ekran[i][j] := KolorPiora
:
```

Rys. 5: Sterownik monitora graficznego

W powyższym przykładzie każdy element tablicy „obsługiwany" jest przez osobny proces i nowa wartość jest nadawana wszystkim elementom jednocześnie.

4 Udogodnienia dla programów czasu rzeczywistego

4.1 Priorytety

Konstrukcje `ALT` oraz `PAR` mogą występować w wersjach priorytetowych, oznaczanych odpowiednio `PRI ALT` oraz `PRI PAR`. Przykładowy proces z rys. 6 wykonuje pewne obliczenia co jakiś czas sprawdzając, czy nie nadszedł sygnał do zakończenia pracy. Ponieważ dozór `TRUE & SKIP` jest zawsze spełniony, nie jest więc wykluczone, że ten właśnie wariant byłby wybierany przy każdym wykonaniu pętli, gdyby zastosowano „zwykłą" konstrukcję `ALT`. W konstrukcji `PRI ALT` wybór procesu do wykonania nie jest jednak arbitralny, jak to ma miejsce w przypadku konstrukcji `ALT`. Spośród wszystkich wariantów, których dozory są spełnione, wybierany jest ten, który występuje tekstowo najwcześniej, podobnie jak to ma miejsce z wyborem procesu w konstrukcji `IF`. W przypadku konstrukcji

```
SEQ
  wykonuj := TRUE
  WHILE wykonuj
    PRI ALT
      koniec ? x
        wykonuj := FALSE
      TRUE & SKIP
        ... obliczenia
```

Rys. 6: Przykład użycia priorytetowej konstrukcji `ALT`

`PRI PAR` procesom współbieżnym nadawane są różne priorytety według kolejności umieszczenia ich w tekście programu. Najwyższy priorytet ma proces umieszczony jako pierwszy. W każdej chwili spośród wszystkich procesów gotowych do wykonania wykonywany jest ten, który ma najwyższy priorytet.

4.2 Zegary

Zegary (ang. timer) są obiektami, które zachowują się podobnie jak kanały, z tym że jedyną operacją jaką można wykonać z udziałem zegara jest pobranie z niego wartości typu `INT` określającej bieżący czas. Rozdzielczość zegara zależy od systemu i np. w transputerze firmy Inmos istnieją dwa rodzaje zegarów, umożliwiające pomiar czasu z dokładnością do 1 i 64 μs. Mierzenie czasu rozpoczyna się w chwili startu systemu, chyba że umożliwia on pomiar czasu rzeczywistego. Z upływem czasu wartości pobierane z zegara zmieniają się od 0 do `MOSTPOS INT`, po przekroczeniu której wskazanie zegara staje się równe `MOSTNEG INT` i rośnie do 0. Z tego względu operując na wartościach odczytanych z zegara należy stosować operatory arytmetyki modulo. Procedura zamieszczona na rys. 7 powoduje wstrzymanie wykonania programu na czas określony wartością jej parametru. Przykład przedstawiony na rys. 8

```
PROC czekaj(VAL INT przerwa)
  TIMER zegar:
  INT czas:
  SEQ
    zegar ? czas
    zegar ? AFTER czas PLUS przerwa
:
```

Rys. 7: Tekst procedury powodującej zawieszenie wykonywania programu na określony czas demonstruje użycie zegara do wykrywania opóźnienia w otrzymaniu wiadomości. Jeśli przez czas równy 100 „tiknięciom” zegara nie nadejdzie kanałem `dane` oczekiwana wartość, to proces wyśle kanałem `alarm` informację o braku danych.

```
VAL oczekiwanie IS 100:
TIMER zegar:
INT czas, x:
SEQ
  zegar ? czas
  ALT
    dane ? x
      ... przetworzenie danych
    zegar ? AFTER czas PLUS oczekiwanie
      alarm ! "Brak danych"
```

Rys. 8: Przykład zastosowania zegara do wykrywania opóźnienia w otrzymaniu wiadomości

4.3 Porty

```
-- rej.ster - rejestr sterujacy
-- rej.danych - rejestr danych
PROC czytaj (PORT OF BYTE rej.ster,
             PORT OF INT16 rej.danych,
             INT16 wart)
  VAL Gotow IS #01:
  BYTE s:
  SEQ
    rej.ster ? s
    WHILE ((INT s) /\ Gotow) = 0
      SEQ
        czekaj(1000)
        rej.ster ? s
    rej.danych ? wart
:
```

Rys. 9: Procedura pobierania danych z urządzenia, którego dwa rejestry widoczne są w przestrzeni adresowej programu (tekst procedury `czekaj` zamieszczono na rys. 7)

Porty są obiektami podobnymi do kanałów, z tą jednak różnicą, że komunikacja z ich pomocą nie jest synchroniczna. Porty z reguły nie są używane do komunikacji między procesami języka lecz do obsługi urządzeń odwzorowywanych w pamięci. Na rys. 9 przedstawiono tekst procedury, w której pobierane są dane z urządzenia. Dwa jego rejestry udostępnione są dla programu jako porty. W procedurze co jakiś czas (konkretnie co 1000 „tiknięć” zegara) odczytuje się stan rejestru sterującego tego urządzenia i testuje jego zawartość. Jeśli odpowiedni bit ma wartość 1, to wówczas odczytywana jest wartość z rejestru danych. W warunku pętli `WHILE` zmienna `s` podlega konwersji do typu `INT`, gdyż operator iloczynu bitowego „/\” stosuje się tylko do argumentów tego typu.

4.4 Konfiguracja

Celem konfiguracji jest dostosowanie programu do konkretnego sprzętu i nie wpływa ona na jego strukturę logiczną. OCCAM umożliwia

- przyporządkowanie kanałów logicznych występujących w programie kanałom fizycznym istniejącym w systemie, co pozwala np. na realizację synchronicznych operacji wejścia/wyjścia;
- przydzielenie procesorom — w przypadku systemu wieloprocesorowego — poszczególnych procesów, które będą się w nich wykonywać;
- umiejscowienie kanału, portu, zmiennej lub tablicy pod konkretnym adresem w pamięci operacyjnej, co ułatwia programowanie obsługi urządzeń odwzorowywanych w pamięci; przykład z rys. 5 można w następujący sposób uzupełnić o informację, że pamięć ekranu rozpoczyna się od adresu #F000:

```
[X][X] INT16 Ekran:
PLACE Ekran AT #F000:
```

Do konfiguracyjnych elementów języka należy zaliczyć także omawiane wcześniej priorytety.

5 OCCAM 3.0

Nowa wersja OCCAMu jest znacznie bogatsza od poprzedniej. Programista może deklarować własne typy, powiększono liczbę rodzajów typów złożonych o rekordy i unie, a także wprowadzono nowe rodzaje kanałów oraz nowe mechanizmy strukturalizacji programów, omówione poniżej.

Programowanie komunikacji między procesami ułatwiono przez wprowadzenie kanałów dzielonych (ang. shared channels) oraz kanałów zdalnego wywołania procedury (ang. remote call channels). Pierwsze z nich umożliwiają komunikację między jednym nadawcą (odbiorcą) a dowolną liczbą odbiorców (nadawców). Dzięki kanałom zdalnego wywołania możliwe jest przekazywanie parametrów do procedury wykonywanej przez inny proces współbieżny, tak jak to ma miejsce w przypadku zwykłego wywołania procedury. Kanały zdalnego wywołania mogą być deklarowane jako kanały dzielone.

Nowymi elementami języka pozwalającymi na znaczną strukturalizację programów są moduły i biblioteki. Wnętrze modułu stanowią procesy. Dla zewnętrznych procesów moduł jest „czarną skrzynką", a komunikacja między jego wnętrzem a nimi dokonuje się przy pomocy kanałów określonych w deklaracji modułu. Możliwe jest deklarowanie typów modułowych, a następnie kolejnych obiektów tych typów. Biblioteka jest zbiorem definicji typów, stałych, zmiennych, funkcji, procesów i modułów. Każdy program, który korzysta z danej biblioteki, ma dostęp do wszystkich elementów przez nią eksportowanych.

6 Podsumowanie

W niniejszym opracowaniu omówiono język programowania OCCAM w wersji 2.0. Podstawowymi pojęciami w tym języku są proces i kanał. Procesy są podstawowymi elementami konstrukcyjnymi programu i mogą wykonywać się sekwencyjnie lub równolegle. Jednokierunkowe kanały umożliwiają synchroniczną komunikację między asynchronicznie wykonywanymi procesami współbieżnymi. OCCAM bardzo dobrze nadaje się do tworzenia oprogramowania dla systemów czasu rzeczywistego ze względu m.in. na precyzyjną definicję operacji arytmetycznych, możliwość tworzenia programów współbieżnych i nadawania priorytetów procesom, a także dzięki uwzględnieniu zagadnień konfiguracji oprogramowania czy pomiaru czasu.

Istotną cechą języka jest „statyczność" programów w nim napisanych. Połączenia między procesami nie mogą być w żaden sposób modyfikowane w trakcie wykonania programu. W języku nie istnieją typy wskaźnikowe, nie jest więc możliwe korzystanie z dynamicznych struktur danych. Liczba procesów, które mają być wykonywane współbieżnie, musi być znana na etapie kompilacji. OCCAM nie pozwala na rekonfigurację programu w trakcie jego wykonania. Te i inne cechy języka umożliwiły zwiększenie zakresu kontroli wykonywanej w trakcie kompilacji, co korzystnie wpływa na poprawność tworzonego oprogramowania. Z drugiej strony własności te są źródłem pewnych ograniczeń co do obszaru zastosowań OCCAMu, a także mogą mieć negatywny wpływ na efektywność czasową i pamięciową programów pisanych w tym języku.

7 Podziekowania

Chciałym bardzo podziękować osobom, które przeczytały wstępną wersję niniejszego artykułu i przekazały wiele cennych uwag: Bożenie Bartoszek, Zbigniewowi Czechowi oraz Jarosławowi Francikowi.

Cytowana literatura

[1] Barrett, G.: occam 3 reference manual, Inmos Ltd., 1992.

[2] Burns, A.: Programming in occam 2, Addison–Wesley Publishing Company, 1988.

[3] Hoare, C.A.R.: Communicating sequential processes, w: Communications of the ACM, Vol. 21, No. 8, pp. 666–677.

[4] Pountain, D., May D.: A tutorial introduction to occam programmming, BSP Professional Books, 1988.

[5] Wojtulewicz M.: Język occam II, w: Czech, Z. (red.): Programowanie współbieżne. Wybrane zagadnienia, Skrypt nr 1931 Politechniki Śląskiej, wyd. III, 1995.

[6] Edwards, J.: Occam and the transputer — current developments, IOS Press, 1991.

[7] Allen, A.: Transputer systems — ongoing research, IOS Press, 1992.