sprawdzania, czy z kanału korzystają dokładnie dwa procesy i czy jeden z nich jest nadawcą a drugi odbiorcą. Upraszcza to zapis procesów, a tym samym zrozumienie, jak one działają. Wadą jest to, że procesy muszą znać nawzajem swoje nazwy, co utrudnia tworzenie bibliotek. Dlatego języki programowania oparte na CSP, takie jak occam, Parallel C czy Edip, wymagają komunikacji przez kanały. Języki te omówiamy dokładniej na końcu rozdziału.

Nie będziemy przedstawiać całego języka CSP. Pominiemy szczegóły, które nie są konieczne do zrozumienia przykładów i rozwiązania podanych zadań, a zwłaszcza nie będziemy omawiać sposobu komunikacji programu ze światem zewnętrznym.

5.1.2 Struktura programu

Program w CSP składa się z ujętego w nawiasy kwadratowe ciągu treści procesów oddzielonych od siebie znakiem „||", oznaczającym, że procesy będą wykonywane równolegle. Treść każdego procesu jest poprzedzona etykietą, będącą jednocześnie nazwą procesu, po której następują dwa dwukropki. Ogólny schemat programu wygląda więc następująco:

```
P1::<treść P1> || P2::<treść P2> ||...|| Pn::<treść Pn>
```

Procesy możemy parametryzować tworząc tablicę procesów. Tak więc

```
P(k:1..N):: <treść zależna od zmiennej związanej k>
```

jest skrótowym zapisem `N` procesów o nazwach odpowiednio `P(1),...,P(N)`, przy czym treść procesu `P(i)` powstaje przez zastąpienie zmiennej związanej `k` stałą `i`. Możliwe jest tworzenie wielowymiarowych tablic procesów w naturalny sposób.

5.1.3 Instrukcje

Charakterystyczną cechą języka CSP jest prostota i zwięzłość. Wyróżnia się w nim cztery rodzaje instrukcji prostych: przypisania, pustą, wejścia i wyjścia oraz tylko dwa rodzaje instrukcji strukturalnych: alternatywy i pętli. (Instrukcje wejścia i wyjścia pełnią w CSP szczególną rolę i dlatego omówimy je oddzielnie w punkcie 5.1.4) Oprócz sekwencyjnego wykonania instrukcji jest możliwe także wykonanie równoległe w obrębie jednego procesu.

Przypisanie

Oprócz zwykłej instrukcji przypisania, w CSP jest możliwe także przypisanie jednoczesne. Wówczas po lewej stronie znaku „:=" zamiast nazwy zmiennej znajduje się ujęty w nawiasy ciąg nazw zmiennych oddzielonych przecinkami, a po prawej stronie znajduje się odpowiadający mu ciąg wyrażeń, także oddzielonych przecinkami i ujętych w nawiasy. Wykonanie takiej instrukcji

polega na równoległym obliczeniu wartości wyrażeń, a następnie jednoczesnym przypisaniu ich odpowiadającym im zmiennym. Przykładowo,

```
(x,y) := (y,x)
```

jest jednoczesną zamianą wartości `x` i `y`. Dla uproszczenia przyjmujemy, że wyrażenia można tworzyć za pomocą takich samych operatorów jak w Pascalu.

Instrukcja pusta

Instrukcję pustą oznacza się słowem kluczowym `skip`. Potrzebę jej wprowadzenia wyjaśniamy w następnym punkcie.

Instrukcja alternatywy

W CSP, zamiast zwykłej instrukcji warunkowej, jest dostępna ogólniejsza instrukcja *alternatywy* wzorowana na instrukcjach dozorowanych Dijkstry (por. [Dijk78]). Instrukcję alternatywy zapisuje się według schematu

```
[D1 -> I1 [] D2 -> I2 [] ... [] Dn -> In]
```

Dozór `Di` jest niepustym ciągiem warunków logicznych oddzielonych średnikami. Średnik w tym przypadku oznacza operator logiczny `and`. Warunki w dozorze są obliczane po kolei (począwszy od lewej strony), a po napotkaniu pierwszego fałszywego warunku obliczanie jest przerywane. Symbol `Ii` oznacza tu niepusty ciąg instrukcji oddzielonych średnikami. Średnik w tym przypadku jest operatorem następstwa. (Między warunkami w dozorze i między instrukcjami mogą się pojawiać także deklaracje zmiennych, por. 5.1.5.)

Instrukcje dozorowane w obrębie instrukcji alternatywy można parametryzować w następujący sposób:

```
(k:1..N) dozór zależny od k -> instrukcje zależne od k
```

Taki zapis jest równoważny wypisaniu `N` odpowiednich dozorów i `N` następujących po nich ciągów instrukcji, w których w miejsce zmiennej związanej `k` jest wstawiony numer kolejny dozoru.

Wykonanie instrukcji alternatywy polega na równoczesnym obliczeniu wszystkich jej dozorów, niedeterministycznym wybraniu jednego dozoru spośród tych, które są spełnione i wykonaniu następującego po tym dozorze ciągu instrukcji. Jeśli w instrukcji alternatywy żaden dozór nie jest spełniony, uważa się to za błąd powodujący przerwanie wykonania programu. Tak więc instrukcja `[x>0 -> x:=1]` spowoduje błąd, gdy $x \leq 0$ (jeśli zawsze $x > 0$, nie ma sensu używania instrukcji alternatywy). Efekt instrukcji pascalowej postaci `if w then I` można uzyskać w CSP pisząc

```
[ w -> I [] not w -> skip]
```

Pętla

Instrukcję pętli zapisuje się dostawiając gwiazdkę przed instrukcją alterna tywy. Pętla w CSP ma więc następującą postać:

```
*[D1 -> I1 [] D2 -> I2 [] ...  [] Dn -> In]
```

Wykonuje się ją tak długo, aż nie będzie spełniony żaden z jej dozorów Jeśli w danej chwili jest spełniony więcej niż jeden dozór, podobnie jak w instrukcji alternatywy, niedeterministycznie jest wybierany jeden z nich i wykonuje się następującą po nim instrukcję dozorowaną.

Wykonanie równoległe

Równoległe wykonanie kilku ciągów instrukcji zapisuje się w CSP następująco

```
I1 || I2 || ...  || In
```

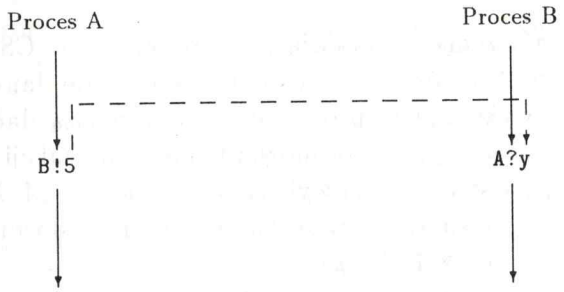
Komentarze

Komentarze w CSP poprzedzamy słowem kluczowym `comment` i kończymy średnikiem. (Warto w tym miejscu zauważyć, że w CSP występują tylko dwa słowa kluczowe: `comment` i `skip`.)

5.1.4 Spotkania

W CSP procesy komunikują się ze sobą w sposób synchroniczny za pomocą instrukcji wejścia-wyjścia. Instrukcja wyjścia postaci `B!X(y1,...,yn)` w procesie `A` oznacza, że proces `A` chce wysłać do procesu `B` ciąg wartości wyrażeń `y1,...,yn`, identyfikowany nazwą `X`. Aby było to możliwe, w procesie `B` musi znajdować się dualna instrukcja wejścia `A?X(z1,...,zn)`, przy czym `z1,...,zn` są zmiennymi odpowiednio tego samego typu co wyrażenia `y1,...,yn`. Instrukcja wejścia-wyjścia jest wykonywana wtedy, gdy sterowanie w procesie `A` osiągnie instrukcję wyjścia, a sterowanie w procesie `B` odpowiadającą jej instrukcję wejścia (zwykle więc jeden z procesów musi czekać na drugi). Wówczas odbywa się *spotkanie*, które polega na jednoczesnym wykonaniu ciągu przypisań `zi := yi`, dla `i = 1,...,n`. Parę instrukcji wejścia i wyjścia można więc uważać za dwie części tej samej instrukcji wejścia-wyjścia, a jej wykonanie za „rozproszone przypisanie". Ideę spotkania ilustruje rys. 5.1.

Ciąg wyrażeń w instrukcji wyjścia (i odpowiednio zmiennych w instrukcji wejścia) może być pusty. Taką instrukcję wejścia-wyjścia nazywamy *sygnałem*. Jest to mechanizm służący jedynie do synchronizacji procesów. Jeżeli nie prowadzi to do niejednoznaczności, można pominąć nazwę identyfikującą ciąg wyrażeń lub zmiennych, a ponadto, gdy ciągi wyrażeń i zmiennych są jednoelementowe, można również pominąć nawiasy.



Rys. 5.1. Symetryczne spotkanie w CSP

Instrukcje wejścia mogą występować w dozorach. Instrukcja wejścia postaci `Q?...` występująca w dozorze procesu `P` ma wartość `false`, jeśli proces `Q` już nie istnieje, oraz wartość `true`, jeśli proces `Q` czeka na wykonanie odpowiedniej instrukcji wyjścia `P!...` — wyliczenie tego dozoru wiąże się wtedy z jednoczesnym wykonaniem instrukcji wejścia w procesie `P` i instrukcji wyjścia w procesie `Q`. Jeśli proces `Q` nie czeka na wykonanie swojej instrukcji wyjścia, obliczenie dozoru w procesie `P` zawiesza się do chwili, gdy proces `Q` dojdzie do tej instrukcji. W jednym dozorze może wystąpić tylko jedna instrukcja wejścia i musi być ona ostatnią częścią dozoru. To ograniczenie wynika z faktu, że nie ma możliwości cofnięcia operacji wejścia-wyjścia, gdy następujący po instrukcji wejścia warunek jest fałszywy. Jeżeli wszystkie dozory w instrukcji alternatywy lub instrukcji pętli zawierają instrukcję wejścia i w danej chwili żadna z nich nie może być wykonana, to wykonanie całej instrukcji alternatywy lub pętli zawiesza się do czasu, gdy którąś z instrukcji wejścia będzie można wykonać. Jeśli w danej chwili można wykonać więcej niż jedną instrukcję wejścia w dozorach, to do wykonania jest wybierana niedeterministycznie jedna z nich. Zakładamy przy tym, że niedeterminizm ten jest realizowany w sposób, który zapewnia własność *żywotności*. Oznacza to, że jeśli instrukcja alternatywy lub pętli jest wykonywana dostatecznie wiele razy i za każdym razem można wykonać daną instrukcję wejścia, to w końcu instrukcja wejścia zostanie wykonana.

5.1.5 Deklaracje

Wyróżniamy cztery standardowe typy: `integer`, `real`, `boolean` i `char` oraz typ tablicowy. Przykładowo, `i:integer` jest deklaracją zmiennej całkowitej `i`, a `x:(1..N)char` jest deklaracją jednowymiarowej tablicy typu znakowego. Deklaracja może wystąpić w dowolnym miejscu treści procesu. Obowiązuje ona od miejsca wystąpienia aż do końca tej instrukcji złożonej, w której wystąpiła (jeśli występuje na najwyższym poziomie, to obowiązuje do końca treści procesu).

5.1.6 Ograniczenia

Dwa istotne ograniczenia utrudniają programowanie w CSP. Po pierwsze, nazwy wszystkich procesów, z którymi komunikuje się dany proces, muszą być znane przed wykonaniem programu, a więc muszą dać się wyznaczyć statycznie. Oznacza to np., że nie można napisać instrukcji wyjścia postaci `P(i)!x`, w której `i` jest wartością wyliczaną w procesie. Można ten problem ominąć korzystając ze sparametryzowanych instrukcji dozorowanych

```
[(j:1..N) i = j -> P(j)!x]
```

przy czym `j` jest zmienną związaną. W instrukcji alternatywy tylko ten dozór jest spełniony, w którym `j = i`.

Drugim utrudnieniem jest brak możliwości umieszczenia w dozorze instrukcji wyjścia. To ograniczenie ma zapobiegać nadmiernemu niedeterminizmowi programów mogącemu utrudniać ich zrozumienie i dowodzenie poprawności (por. [Bern80]). Gdyby instrukcja wyjścia mogła wystąpić w dozorze, wówczas niedeterministyczny wybór mógłby objąć nie jedną, ale jednocześnie wiele instrukcji dozorowanych w wielu różnych procesach. Na przykład w programie postaci

```
[ A::  ...  [B!x -> ...  [] C?y -> ...  ]
||B::  ...  [C!z -> ...  [] A?u -> ...  ]
||C::  ...  [A!v -> ...  [] B?w -> ...  ] ]
```

decyzja, które z przypisań `u := x`, `w := z` czy `y := v` ma być wykonane, obejmuje aż trzy procesy.

Są jednak przypadki, w których możliwość wykonania instrukcji wyjścia powinna decydować o wyborze pewnego ciągu instrukcji. Ponieważ instrukcji wyjścia nie można umieścić w dozorze, należy odpowiadającą jej instrukcję wejścia w procesie-odbiorcy poprzedzić wysłaniem sygnału informującego nadawcę o gotowości odbioru. Odbiór takiego sygnału można wówczas umieścić w dozorze w procesie-nadawcy. Tę technikę stosujemy np. w przykładzie 5.2.2.

5.2 Przykłady

5.2.1 Wzajemne wykluczanie

Jest kilka sposobów rozwiązania problemu wzajemnego wykluczania. Najprostszy polega na zrealizowaniu w CSP semafora. Podamy tu realizację semafora ogólnego i binarnego. W przykładzie 5.2.3 pokazujemy, jak w CSP można zrealizować monitor.

W systemach rozproszonych nie stosuje się zwykle globalnych semaforów czy monitorów do zapewnienia wzajemnego wykluczania. Rozproszony sprzęt wymaga rozproszonego oprogramowania, co oznacza rozproszenie zarówno danych, jak i sterowania. Przy takim podejściu decyzja o