Увод в езика occam

Бойко Банчев

Сред езиците, предоставящи средства за паралелно програмиране, occam заема особено място. Важни негови отличителни черти са това, че практически всеки оператор в програмата може да се изпълнява паралелно с другите (налице е паралелност на операторно, а не на подпрограмно, както в Ada, Erlang и др. равнище) и това, че езикът е разработен с оглед на реализирането му на специфична компютърна архитектура с паралелно действие.

Теоретична основа на паралелността в occam е езикът CSP, задаващ модел за описване на взаимодействащи, паралелни помежду си процеси. Заедно с това езикът е специално разработен като инструмент за програмиране на транспютърните системи на фирмата INMOS: онези негови свойства, които имат отношение към паралелността, се реализират непосредствено чрез архитектурата на системата.

Транспютърът е блок за построяване на изчислителни системи с паралелно действие. Всеки транспютър има собствена памет, процесор и четири линии за връзка с други транспютри, всяка от които съдържа по един входен и един изходен канал. Транспютърната изчислителна система е мрежа от определен брой транспютри, всеки от които може да изпълнява отделен процес паралелно с останалите. Обменът на данни между транспютрите в мрежата се осъществява само чрез свързване на техните входни и изходни канали един с друг, но не и чрез обща памет.

В една транспютърна мрежа могат да участват различни видове транспютри, а съединенията между тях могат да бъдат фиксирани или превключваеми. Настройването на съединенията в мрежа с променлива топология става по програмен път, но преди зареждане на основната програма за изпълнение в мрежата. В езика occam са предвидени средства за конфигуриране и настройване на системата и всяка програма може сама да задава както апаратната конфигурация, необходима за нейното изпълнение, така и назначаването на процесори за съставящите я процеси и на линии за връзка за осъществяване на обмен между процесите.

Езикът occam не е стандартизиран и се използват различни негови диалекти. За съвременно определение се приема даденото във фирмения документ [Occ88], където за различаване се използва името occam2. Това определение обогатява езика с множество съществено важни за широкото му прилагане конструкции и именно него следваме тук.

Обща структура на програмата

Програмирането на occam предполага построяване на програмата като множество от паралелни един на друг процеси. Всеки процес има свои локални променливи и може да взаимодейства с другите процеси посредством канали. Каналът за обмен на данни свързва *еднопосочно и синхронно* два процеса, за единия от които той е *изходен*, а за другия – *входен*: първият процес предава данни към втория.

Предаването на стойности чрез канал е единственото средство за взаимодействие между процеси. Не се предвижда обобществяване или друг вид съвместен достъп до данни за два или повече процеса. Това определя модела на паралелност в occam като строго *съобщителен* и следователно – подчертано *децентрализиран*. Такъв подход има важни достойнства: (а) премахва се възможността за възникване на грешки,

дължащи се на неправилно използване на общи променливи и (б) улеснява се разпределянето и преразпределянето на програмните действия по процесори.

Основните действия в програмата се задават чрез оператори за:

- присвояване
- задаване на условно разклоняване на управлението в две или повече посоки (`IF`) и на циклично изпълнение, управлявано от предусловие (`WHILE`)
- задаване на структури от процеси
- обръщение към процедури
- обмен на данни между процеси

Всеки оператор се изписва на отделен ред и по принцип представлява самостоятелен процес. Влагането на група от оператори в друг оператор се изразява чрез разполагане на вложения фрагмент с отстъп надясно в текста спрямо обхващащия го оператор.

Даннови типове. Присвоявания. Синоними

Основните даннови типове са булев, байтов, няколко вида цели и числа с плаваща точка. Към целите числа, освен аритметичните операции, са приложими и поразредни двоични: булеви операции и поразредни измествания.

Може да се определят едномерни или многомерни масиви с елементи от основните типове, както и масиви от канали. Например:

```
[10] INT a:                 -- едномерен масив от цели
[10][10] CHAN OF INT b:     -- двумерен масив от канали за цели
```

Масивите са единствения вид агрегатни типове в езика. Други агрегатни типове могат да се имитират посредством *канални протоколи*, за което ще стане дума по-нататък.

Операторът за присвояване допуска присвояване на повече от една стойност и това става по два различни начина. При *съвместното присвояване* на две или повече променливи могат да бъдат зададени съответния брой стойности, както в оператора `a,b:=3,8`. При *сегментното присвояване* отрязък от масив се присвоява на друг отрязък със същата дължина. Например

```
[a FROM i FOR n] := [b FROM j FOR n]
```

присвоява отрязък от масива `b` на отрязък от масива `a`.

occam е език със строго типизиране на стойностите и имената. За всяка променлива и канал, използвани в програмата, чрез задължително поименно описване еднозначно се задава тип на стойностите, които могат да бъдат присвоявани на променливата или предавани по канала. Описанията могат да се разполагат на произволно място в програмата преди действията, използващи съответните имена.

При пресмятането на изрази не се извършват преобразования на стойности от един тип в друг, освен ако такива преобразования не са посочени явно.

Описателят

име `IS` *израз*

определя „*име*“ като *синоним* на данновия обект, зададен чрез „*израз*“. Използването на синоними служи за преименуване на променливи, когато това е желателно, а

също, когато се прилага за именуване на елемент на масив или на отрязък от масив, улеснява записването на различни действия с такива обекти в програмата.

Ако за даден обект е определен синоним, в областта на действие на синонима този обект не може да бъде именуван другояче, освен чрез него. В частност, за един обект не може да има повече от един синоним. Ако се именува отрязък от масив, никой негов елемент не може да бъде цитиран чрез името на масива, докато синонимът е валиден. За да се осъществи достъп до елементите на масива, които не са в отрязъка, за тях трябва да се зададат един или повече други синоними. Всички синоними за части от един и същ масив трябва да отговарят на неприпокриващи се отрязъци. Тези правила осигуряват свойството всеки обект в програмата да бъде именуван във всяка точка на нейното изпълнение чрез не повече от едно име, т.е. отстранява се явлението синонимия. (За отсъствието на синонимия има значение и това, че в OCCAM няма указателни стойности.)

Освен изрично, механизмът на синонимите се прилага по подразбиране и при определяне на съответствието между имената на формалните параметри на процедура и аргументите на обръщенията към нея. Именно, тялото на процедурата се разглежда като *текстово вложено* в програмата в мястото на обръщението, а формалните параметри се тълкуват като синоними на съответните им аргументи, все едно, че за всяка двойка аргумент-параметър се изпълнява описател `IS`. От това следва, че два различни формални параметъра не могат да съответстват на един и същ обект-аргумент. Така чрез общото правило за недопустимост на синонимията това явление се отстранява и в характерния частен случай на обръщение към процедура с еднакви параметри.

Доколкото проверката за отсъствие на синонимия не винаги може да се извърши от компилатора, в редица случаи тя се прави в хода на изпълнението на програмата.

Структури от процеси

OCCAM предвижда средства за задаване на последователно или паралелно изпълнение на фрагменти от програмата с произволна сложност и равнище на вложеност в друг програмен фрагмент. Изпълняваната програма представлява динамично изменящо се множество от паралелни процеси, взаимодействащи по назначени за целта канали.

Операторът `SEQ` задава няколко последователно изпълнявани процеса:

```
SEQ
  процес1
  процес2
  ...
```

Всеки от процесите е отделен оператор: за присвояване, за обмен, за обръщение към процедура и т.н. Аналогично операторът `PAR` задава паралелно изпълнявани процеси:

```
PAR
  процес1
  процес2
  ...
```

Паралелно изпълняване на процеси, съдържащи еднообразни действия, се задава чрез *повторителна форма* на същия оператор. Ако `channels` е масив от 5 канала и `pipe` е процедура, която приема данни по един канал и ги извежда по друг, следният фрагмент образува конвейер от 5 паралелно изпълнявани процеса, всеки от които е екземпляр на процедурата `pipe` с различни аргументи-канали:

```
PAR i=0 FOR 4
  pipe(channels[i],channels[i+1])
```

Двумерна мрежа от паралелно изпълнявани подобни един на друг процеси може да се получи чрез влагане на повторителни оператори `PAR` и използване на двумерен масив от канали.

За оператора `SEQ` също е допустима повторителна форма: чрез нея се задава последователно изпълняване на програмен фрагмент определен брой пъти, което всъщност е обичайният за „последователните“ езици цикъл с брояч на итерациите.

Операторът `PAR` има и *приоритетна форма*, която се записва като `PRI PAR`. Ако два или повече от процесите, включени в конструкцията `PRI PAR`, са едновременно готови за изпълнение (не са в очакване на канален обмен), задейства се само онзи от тях, който е най-напред в списъка на блока `PRI PAR`. Тази форма не задава действително паралелно изпълнение, а *избор* на процес според готовността му и приоритета му спрямо другите.

Предвидени са правила за избягване на потенциално многозначно, т.е. водещо до неопределеност, съвместно използване на променливи и канали от два или повече паралелни процеса. Например, ако дадена променлива е достъпна в известен брой паралелни един на друг процеси, нейната стойност може да бъде само ползвана, но не променяна. Промяна на променлива е възможна само в процес, който има индивидуален достъп до нея, а ползването на стойността на такава променлива и от други процеси става само чрез предаване към тях (и съответно приемане от тяхна страна) на стойността посредством оператори за канален обмен. Съвместен достъп до една и съща променлива от няколко паралелни процеса е възможен само ако всички те ползват стойността ѝ, но не я променят.

Аналогично, даден канал може да се използва от най-много два паралелни процеса, като винаги единият предава, а другият приема данни по него. Така всеки канал е средство за обмен между точно два процеса.

Изпълнението на горните правила се проверява *статично* (от компилатора), като се предполага отсъствието на синонимия. Последното свойство на свой ред се осигурява, както бе вече споменато: отчасти статично и отчасти чрез проверки по време на изпълнението на програмата.

Процедури и функции

В occam процедури и функции се описват и използват, както и в непаралелните езици, преди всичко като метод за получаване на параметризирана абстракция за действие, обикновено извършвано повече от един път. Заедно с това процедурите на occam, отговарящи на определени условия, могат да бъдат самостоятелно компилирани, което улеснява разработването на програмата като съставена от отделни дялове.

За да може да се компилира отделно, дадена процедура трябва да не съдържа

обръщения към външни за нея променливи и да не извършва канален обмен на данни. Тя трябва да взаимодейства с останалите съставки на програмата само посредством параметрите си.

Определенията на процедури и функции, които не се компилират отделно, се разполагат на произволно място в програмата преди използването им, подобно на описанията на променливи и канали.

Тъй като обръщението към процедура е самостоятелен оператор в програмата, то може да бъде (и в много случаи е) процес, изпълняван паралелно с други процеси. В частност, паралелно могат да се изпълняват няколко екземпляра на една и съща процедура.

Както за процедури, така и за функции не се допуска рекурсивно определяне. Това ограничение, заедно с други свойства на езика, дава възможност количеството памет, необходимо за изпълнение на програмата, да бъде нейна статична характеристика и да се създават високоефективни реализации на езика.

Предаването на аргументи към процедури може да става по стойност и като променливи. Аргументите могат да бъдат данни от всякакъв тип, включително например канали и отрязъци от масиви. Процедурите не могат да бъдат предавани като аргументи на други процедури, тъй като в езика те не се разглеждат като стойности.

Функциите се описват подобно на процедури, но обръщението към функция става в израз, а не в самостоятелен оператор. Освен това, аргументите на обръщение не могат да бъдат предавани като променливи, функциите не могат да извършват канален обмен на данни и изобщо по какъвто и да било начин да предизвикват промяна на стойност на външна спрямо тях променлива. Така се избягва получаването на странични ефекти при пресмятане на функция: единственото последствие от пресмятането е получаването на резултат.

Предаване на данни по канал

Предаването и получаването на стойности по канал стават съответно с операторите

```
c ! v
```

и

```
c ? v
```

където `c` е името на канала, а `v` е в единия случай стойност, а в другия – име на променлива, получаваща стойността. Предаваната стойност трябва да има тип, съвпадащ с този на променливата, което се осигурява посредством строгото типизиране на стойностите и задължителното описване на променливата и на канала по тип.

Когато даден процес изпълнява оператор за предаване или получаване по канал, той преминава в състояние на *очакване*, докато друг процес достигне до изпълняване на оператор съответно за получаване/предаване по същия канал. По този начин каналното съобщаване между процеси, освен че служи за предаване на данни, има ролята и на *взаимно синхронизиращо действие* за двата участващи в него процеса. Всъщност, ако предаваната стойност е без значение за програмата, изпълняваното при канален обмен действие е именно и *единствено синхронизиране*.

Синхронното взаимодействие позволява да се предават данни с произволен обем без използване на междинна памет: в occam се реализира *безбуферен* канален

обмен.

Може да бъде желателно предаващият процес да продължи изпълнението си непосредствено след достигане на оператора за предаване, без да чака готовността на приемащия процес. Това може да се постигне чрез използване на допълнителен канал и трети процес, служещ като буфер между двата зададени. Вместо пряко към получателя, изпращачът предава данните в процеса-буфер, който ги съхранява в локалните си променливи, а след това (синхронизирано) ги изпраща по друг канал към получателя.

При осъществяване на обмен по канал в даден процес се посочва името на канала, но не и името на процеса, който е другият участник в обмена. Вторият може да бъде произволен друг процес, за който програмата предвижда обмен по същия канал; действителният участник във всеки отделен случай се определя от това, кой от тях пръв достига до изпълняване на оператор за обмен. Еднозначност на взаимодействието може да се постигне, като се изберат различни канали за всяка взаимодействаща двойка процеси.

Операторът `ALT` дава възможност в един процес да се приемат данни алтернативно по един от няколко канала. За всеки вход могат да се задават изпълнявани при активирането му действия. Например

```
WHILE TRUE
  ALT
    input.1 ? x
      output ! x
    input.2 ? x
      output ! x
```

многократно въвежда стойност за `x` или по `input.1`, или по `input.2` и я предава по канала `output`. Изборът на входа става според това за кой от двата канала в дадения момент има готов за предаване процес.

За всеки вход освен името на канала може да се зададе булев израз, така че приемането чрез него да бъде възможно само когато изразът има стойност `TRUE` (истина). Такива входове се наричат *охраняеми*.

Във всеки клон на оператора `ALT` вместо вход от канал може да се зададе активатор по време. Съответното действие се изпълнява при изтичане на посочения времеви интервал.

За оператора `ALT` се допуска и форма с *повторител*, аналогично на оператора `PAR` с повторител. Следният пример показва цикличен процес, който при всяка итерация въвежда данни по един (неопределено кой) от каналите от масива `inp.ch`. На всеки от входовете се предават данни от няколко сходни в някакъв смисъл процеса.

```
WHILE ...
  ALT i=0 FOR n
    inp.ch[i] ? ... -- приемане на данни от канала i
      ...           -- действия за вход i
```

Приоритетната форма на `ALT` (записва се `PRI ALT`) се използва в случай, че се смята за възможно да постъпи вход по два или повече канала едновременно. Тогава се задейства този от потенциално активните входове, който стои най-напред в списъка.

Канални протоколи

Даден канал може да бъде предвиден за предаване както на прости, така и на сложни стойности. В първия случай каналът се описва чрез единствения тип на предаваните стойности. В по-общ вид описването на стойностите, предавани по канал, се нарича *канален протокол*. Каналният протокол бива няколко вида.

Един от видовете канален протокол е *масив с брояч*, при който по канала се предава отрязък от масив, предхождан от число, задаващо дължината на отрязъка. За описване на канала се задава типът на брояча и типът на елементите на масива. В следния пример се описва канал за предаване на масиви с елементи от тип `BYTE` с целочислен брояч и се определя двойка паралелни процеси, единият от които извършва предаване, а другият – приемане по канала:

```
PROTOCOL COUNTED.BYTES IS INT::[]BYTE:
CHAN OF COUNTED.BYTES c:
[20] BYTE str:
INT n:
PAR
  c ! 12::"страсти и неволи"
  c ? n::str
```

Променливата `n` получава стойността 12, а след това на `str` се присвоява низът `"страсти и не"`.

Последователният протокол определя предаването на известен брой стойности със зададени типове. Например описанието

```
PROTOCOL B.I.R IS BOOL;INT;REAL:
```

определя канал, по който се предават последователно булева, цяла и реална стойности. Предаване и приемане по такъв канал могат да се зададат например с:

```
CHAN OF B.I.R c:
PAR
  c ! TRUE;7;1.6
  c ? t;j;z
```

където `t`, `j` и `z` са променливи от съответните типове.

Вариантният протокол дава възможност по канала да се предава стойност, имаща един от няколко посочени типа. За всеки от вариантите се задава етикет и тип на предаваната стойност. Например:

```
PROTOCOL VARYING.B.I.C
  CASE
    flag; BOOL
    num; INT
    chars; [20]BYTE
:
```

При приемане по канал с вариантен протокол се посочва за всеки вариант етикетът му, име на променлива, получаваща приеманата стойност и действия, които да се извършат при наличие на дадения вариант. Предаващият процес изпраща най-напред етикета на варианта, а след него и съответната стойност. За горния пример приемането може да стане с:

```
CHAN OF VARYING.B.I.C c:
c ? CASE
      flag; f
        ...
      num; n
        ...
      chars; s
        ...
```

Използването на последователния и вариантния вид протоколи в програми на occam има роля, аналогична на формирането на данни съответно от хетерогенен агрегатен и от вариантен тип, т.е. използването на невариантни и вариантни съчетания в други езици за програмиране.

В общия случай за всеки вариант може да има по няколко предавани стойности, както при последователен протокол. Последният може да се съчетава с протокол за масив с брояч, а такова съчетание може също да бъде вариант във вариантен протокол. Допускат се и варианти, за които са зададени само етикети. Предаването на такива варианти служи като изпращане на *сигнал* към процеса получател, който според сигнала избира едно от няколко действия.

Комбинирайки описания механизъм за изпращане на сигнал със споменатото по-горе буфериране посредством междинен процес, можем да построим например процес, който асинхронно, без да задържа действието си, изпраща сигнал на един или повече процеси, или дори различни сигнали на различни процеси.

При предаване на данни по канал с вариантен протокол от гледна точка на смисъла на програмата обикновено е съществен не само протоколът сам по себе си, но и редът, в който могат да се предават различните варианти. Например, даден процес може да обслужва работата с файлове, като в отговор на сигнали-заявки, получавани по входен за него канал, извършва действията „отваряне на файл“, „четене на блок“, „затваряне на файл“ и „прекратяване на обслужването“. Едновременно може да бъде отворен само един файл. Сигналите, съответни на четирите действия, се описват като варианти на протокол и значи могат да бъдат предавани в по начало произволен ред. От друга страна, правилността на работата на обслужващия процес изисква заявките да бъдат изпълнявани само ако следват една друга в подходящ ред: отварянето на файл трябва да предхожда четенето от него и не се извършва, ако преди това е бил отворен и не е бил затворен друг файл; между отварянето и затварянето може да има произволен брой четения и т. н.

Строежът на всеки нетривиален процес – такъв, който се състои от взаимодействащи един с друг процеси – често се определя почти непосредствено от реда на извършване на обменните действия с другите процеси. При подходящ избор на процесова структура за програмата, решаваща определена задача, текстът на програмата лесно се получава от описанията на допустимите редици от предавани данни за всеки канал и връзките по ред на следване на взаимодействията по различните канали.

(Вж. реализацията на описания процес за работа с файлове в края на тази глава.)

Реагиране на външни събития и обмен на данни с външни устройства

Програмата може да реагира на *асинхронно възникващо събитие*, ако апаратната конфигурация предвижда възможността за настъпването му. Такива събития могат да бъдат предизвикани от системния часовник или от различни външни устройства. Взаимодействието с външни устройства може освен приемане на сигнали за събития да включва и обмен на данни. Всяко от тези действия се записва в програмата на occam аналогично на канален обмен между процеси и също като него се осъществява по някоя от линиите за връзка на някой от транспютрите.

При съставяне на програми за взаимодействие с външни устройства е удобно да се използват операторите `PRI PAR` и `PRI ALT` за задаване на различни приоритети за процесите, реагиращи на различните събития, както и за останалите процеси, пряко или косвено свързани с взаимодействието. Например на процес, който приема сигнал от входно устройство, съобщаващ готовността на устройството да предава данни и който изпълнява собствено въвеждането на данни, обикновено се приписва по-висок приоритет от този на процес, който обработва въведените данни.

Описателят `TIMER` служи за определяне на „канал“ за връзка със системния часовник. Чрез такъв „канал“ програмата получава информация за текущото време или преустановява действието на текущия процес за посочен интервал от време. Всъщност става дума за специфична конструкция в езика, която само синтактично наподобява употребата на входен канал, включително и в оператори `ALT` и `PRI ALT`. Следващата процедура отмерва пауза с големина `dt` от текущия момент, получаван в променливата `t0`:

```
PROC wait(VAL INT dt)
  TIMER tm:
  INT t0:
  SEQ
    tm ? t0
    tm ? AFTER t0 PLUS dt
:
```

Чрез описателя `PORT` се назначава канал за връзка с външно устройство, по който с устройството могат да се обменят данни от посочен тип. С помощта на описателя `PLACE ... AT ...` този канал може да бъде свързан с определен адрес в паметта на транспютъра, апаратно предназначен за обмен на данни между транспютъра и устройството. Следният фрагмент определя канал за връзка с устройство и въвежда байт от него:

```
VAL loctn IS #10000:
PORT OF BYTE usart:
PLACE usart AT loctn:
BYTE b:
SEQ
  usart ? b
     ...
```

Реагиране на грешки

Грешките при изпълнение на програма могат да бъдат аритметични, нарушаване на границите на масив и др. При възникване на грешка се реагира или с прекратяване на изпълнението на програмата, или с прекратяване само на предизвикалия грешката процес, или изобщо не се реагира. Избраната една от трите възможности се прилага за цялата програма и за всяка грешка, която може да възникне при изпълнението ѝ.

При втория вид реакция се предизвиква прекратяване впоследствие и на всички процеси, които правят опит да взаимодействат чрез канал с прекратения поради грешка процес. Това на свой ред води до прекратяване и на процесите, които взаимодействат с вторично прекратените и т.н.

Пример

Даденият тук програмен фрагмент реализира споменатия вече процес, обслужващ работа с входни файлове чрез необходимите четири операции. Взаимодействието с възможните потребители на процеса става посредством канала `client`, описан с вариантния протокол `READER`. По канала се приемат сигналите `open.file` (отваряне на файл), `read.block` (четене), `close.file` (затваряне на файл) и `terminate` (прекратяване на работата).

Вариантът `open.file` на протокола `READER` съдържа освен етикета си и описание на масив, в който да се помести името на отваряния файл. Процесът прочита това име в локалния си масив `filename` непосредствено след получаване на сигнала `open.file`. Останалите варианти съдържат само етикети.

По същество процесът наподобява действието на много прост краен автомат с три състояния (едното от тях – завършващо), имащ азбука от четири команди. Автоматът преминава от едно състояние в друго според постъпващите команди (сигнали), като за управление на работата му се използват няколко булеви променливи.

Променливата `going` има стойност `TRUE` от стартирането на процеса до приемането от него на сигнала `terminate`, което води и до завършването му. Променливата `open.ok` се установява в `TRUE` при успешно отваряне на файл, след приемане на сигнал `open.file`. След като бъде успешно отворен даден файл е възможно приемане на сигналите `read.block` и `close.file` (операторът `IF`). Оставането в цикъла `WHILE`, където е възможно приемането само на тези два сигнала се осигурява чрез променливата `reading`, локална за `IF`, която има стойност `TRUE` от отварянето на файл до затварянето му. При наличие на отворен файл е невъзможно да се отвори друг, нито да се прекрати процеса чрез `terminate`. След затваряне на файл отново става възможно приемането на `open.file` и `terminate` и невъзможно приемането на `read.block` и `close.file`.

```
PROTOCOL READER
  CASE
    open.file; [100]BYTE
    read.block
    close.file
    terminate
:
CHAN OF READER client:
```

```
[100]BYTE filename:
BOOL going:
BOOL open.ok:

SEQ
  going := TRUE
  WHILE going
    SEQ
      client ? CASE
         open.file; filename
            ... -- отваряне на файл, записване на
                -- резултата в open.ok
         terminate
            going := FALSE
      IF
        going AND open.ok
          BOOL reading:
          SEQ
          reading := TRUE
          WHILE reading
            client ? CASE
               read.block
                  ...   -- предаване на прочетения блок
                        -- или на знак за край на файл
               close.file
                  reading := FALSE
        TRUE
          SKIP
```