オペレーティング・システム 32－4
（１９８６．９．１２）

Ｍｏｄｕｌａ－２のＯＳ記述能力

東京大学理学部情報科学科
木　下　佳　樹

プログラミング言語Ｍｏｄｕｌａ－２の同期プリミティブ、データ抽象機能について紹介する。言語組込みの同期プリミティブとしてコルーチンを選んだことの妥当性、Ｍｏｄｕｌａ－２のモジュールに備わるデータ抽象機能の限界についても論ずる。ハードウェアが割込みを使って同期をとるようなものならばコルーチンは、安価に実現できて、しかもかなり読みやすいプリミティブである。真の並行プログラムを書けなくなるのがその代償である。モジュール概念は情報隠蔽の機能を提供しており、しかもコンパイル単位間の直接の依存関係を明白にしている。けれども、引数を持つモジュール、型の引数などの概念がなく、また、複雑な依存関係を記述することもできないので、プログラムの部品化のためには、なお能力の不足が感じられる。

LANGUAGE FEATURES OF MODULA-2 FOR OPERATING SYSTEMS DESCRIPTION

We introduce the synchronization primitive and data encapsulation facility of the programming language Modula-2. When the underlying hardware uses the interrupt mechanism for synchronization, coroutine can be cheaply implemented and is a fairly comprehensible primitive. Proper concurrency must, however,be abandoned. Module concept provides information hiding facility and make explicit the dependency between compilation units. However,there is no concept of parameterized module and type parameters and complicated dependency can not be described. In this point, Modula-2 does not have enough power for writing program parts.

◎Ｍｏｄｕｌａ－２の概観

Ｍｏｄｕｌａ－２は、スイスのチューリッヒ工科大学のＮ．Ｗｉｒｔｈが汎用のシステム記述言語として設計、1980年に発表した言語である。制御構造、データ構造はＰａｓｃａｌから継承したものが多く、型の概念もＰａｓｃａｌと同様である。

○モジュール

可視性制御のための言語構造として、入れ子式の可視性規則を持つ手続きの他に可視性を輸出（ＥＸＰＯＲＴ）、輸入（ＩＭＰＯＲＴ）によって指示するモジュールを設けている。手続きでは、親手続きで見える算体は原則として全て子手続きからも見え、逆に子手続きで宣言された算体は親手続きからは見えない（可視性）。また、手続きで宣言された算体はその手続きが呼ばれた時に生成され、その手続きからの復帰とともに消滅する（存在範囲）。これに対し、モジュールの外で見えている算体は、輸入によって指示されない限り、モジュール内では原則として見えない。また、モジュール内で宣言された算体も輸出で指示すると、モジュールの外でも見える（可視性）。モジュールで宣言される算体は、そのモジュールが属している手続きの内で、最も内側のものの呼出しから復帰までの間、存在している（存在範囲）。このように、モジュールを使うと、算体の可視性と存在範囲の制御を別々に行うことができる。

名前のＱＵＡＬＩＦＩＥＤ　ＥＸＰＯＲＴというものがあって、これを使うと輸入する側では、名前をどのモジュールから輸入するのかを常に指定しなければならない。これは、宙に浮いた名前を無くし、コンパイル単位間の依存関係を明白にするのに役立つ。

コンパイル単位は特別な形をしたモジュールで、プログラム、定義、実現モジュールの三種がある。プログラム・モジュールは実行の本体である。他の二つは組になってライブラリを形成し、外部へのインターフェイス（主に型の情報）を定義モジュールに、手続きの本体、陰の変数や手続き、初期化などの外部から隠すべき事項を実現モジュールにそれぞれ記述する。定義モジュールに記される事項を参照することによって、型の整合性を複数のコンパイル単位間にまたがって検査できる。

コンパイル単位から型名を輸出する場合、その型の実現を外部から隠すことが

できる。これは、不透明な輸出（ｏｐａｑｕｅ　ｅｘｐｏｒｔ）と呼ばれる。

○コルーチン

並行処理のための同期プリミティブを言語機構の中には持たせず、その代わりにコルーチンの制御プリミティブ、及び優先順位（割込み禁止レベル）を、モジュール単位で設定する機能を用意している。

コルーチンのプリミティブは、

```
NEWPROCESS(p:PROC;  a:ADDRESS;  n:CARDINAL;  VAR new:ADDRESS)
TRANSFER(VAR p1,p2:ADDRESS)
```

の二つである。ＮＥＷＰＲＯＣＥＳＳによって、アドレスａで始まるｎ語の作業領域を持ち、実行するコードがｐである（ＰＲＯＣ型は引数を持たず、値も返さないような手続きの型）ようなコルーチンのディスクリプタを作成して、そのアドレスをｎｅｗに返す。ＴＲＡＮＳＦＥＲは、現在実行中のコルーチンを中断してそのディスクリプタのアドレスをｐ１に代入し、ｐ２の指すディスクリプタが表すコルーチンを再開する。この二つの他に、割込み制御のための

```
IOTRANSFER(VAR p1,p2:ADDRESS;  va:CARDINAL)
```

がある。これは、ＴＲＡＮＳＦＥＲと同様の効果を持つが、
「ＩＯＴＲＡＮＳＦＥＲ実行の後にベクトルｖａの割込みが起こった時に、その時実行中のコルーチンをｐ２に入れてｐ１を再開させる」という副作用を合わせ持つ。

モジュールの頭部に優先順位を指定できる。そのモジュール内で宣言された手続きの実行中は指定された優先順位及びそれより低い順位の割込みが禁止される。

○その他

型の概念は基本的にＰａｓｃａｌと同じものであるが、低レベルのプログラミングのため、型検査を意図的に破る機能を設けている。一つは、型変換関数と呼ばれるもので、ある値の機械語表現を別の型として解釈させるものである。言語Ｃのｃａｓｔに相当する。もう一つは、ＷＯＲＤ型及びＡＤＤＲＥＳＳ型で、ＷＯＲＤ型の仮引数には任意の基本型（機械語一語で表現される型）の実引数を与えることができ、ＡＤＤＲＥＳＳ型は任意のポインタ型に同等なものである。この二つの型は、「型検査をしない型」だといえる。

入出力や動的記憶管理、数学関数等は定義・実現モジュールの組によるライブラリとして提供されるが、低レベルの算体（例えばＮＥＷＰＲＯＣＥＳＳ、ＴＲＡＮＳＦＥＲなどのコルーチンプリミティブ）は、処理系に組み込まれた

擬似モジュールＳＹＳＴＥＭから輸出されている。

デバイス・ハンドラなどを記述するには、いわゆるｍｅｍｏｒｙ
ｍａｐｐｅｄ　Ｉ／Ｏのための言語機能が必要だが、Ｍｏｄｕｌａ－２では、変数の絶対番地を指定できるようにして、これを実現している。

言語の設計にあたっては、文面の量と、それに対応するコードの量とが、かけはなれたものにならぬよう配慮されている。この制限のために採用しなかった言語機能が随分あるように思われる。また、Ｐａｓｃａｌのプログラマが自然にＭｏｄｕｌａ－２に移ることへの配慮も払われたという。

言語の定義書は［１］だが、この本のＲＥＰＯＲＴに明記されておらずにＵＳＥＲ　ＭＡＮＵＡＬには記されているようなことがいくつかあり、記述はあまり厳密なものではない。なお、現在、ＡＮＳＩによってＭｏｄｕｌａ－２の標準化作業が進められている。

◎言語機能の評価

○データ抽象

型の不透明な輸出と、モジュールの可視性規則を併せると、ある程度のデータ抽象が可能になる。ただし実現上の問題から、不透明に輸出できるのはポインタ型のみに制限されている。

型の引数、モジュールへの引数など、実行時に型情報が必要になる機能は一切設けられていない。そのため、いわゆるジェネリック・モジュールを自由に定義するには言語機能の不足が感じられる。ＷＯＲＤ型、ＡＤＤＲＥＳＳ型を使ってＭｏｄｕｌａ－２のジェネリック・モジュールを定義する試みもなされてはいる［２］。

○型検査とライブラリ

複数のコンパイル単位間にまたがった型検査がコンパイル時に行われるため、ライブラリの側では与えられたパラメータ値の範囲検査を省くこともできる。しかし、定義モジュールによって可能になるのは型の整合検査だけで、プログラムの意味の整合検査までは行えない。第一、言語の意味が形式的に定義されていない。

○モジュールの依存関係

コンパイル単位間の直接の依存関係を輸出部、輸入部を参照することにより、プログラムの文面から判定できるため、ＵＮＩＸにあるｍａｋｅコマンドに相当することを、依存関係を記述する別の言語を設けることなしに行える。けれども、あまり複雑な依存関係は記述できない。例えば、モジュールＡがＢ、Ｃに依存している場合、Ｂ、Ｃがさらに別の同じモジュールＤに依存していることをＡで指定するといったことが、Ｍｏｄｕｌａ－２ではできない。

プログラムの文面からコンパイル単位が格納されているファイル名を推定することはできない。そのため、一つの定義モジュールに、複数の実現モジュールを作成したような場合、クライアントから、特定の実現モジュールをプログラムの文面上で指定することができない。これは、Ｍｏｄｕｌａ－２がプログラムを保存するファイルシステムをはじめとする操作環境までは定義していないためである。

○同期プリミティブの実現

コルーチンとモジュールの優先順位指定機能を組み合わせると、必要に応じて様々な同期プリミティブをＭｏｄｕｌａ－２だけを使って記述できる。そのようにして記述した同期プリミティブを一つのコンパイル単位に集めてライブラリを構成し、適当なデータ抽象を施すと、それを使用する側（クライアント）からは、並行プログラミングのための言語構造が用意されているのと変わりなく見える。

一般に、使用するハードウェアに用意された同期命令と言語機構に用意された同期プリミティブがかけはなれたものになると、プリミティブの実現コストが高くなり、場面によっては、不必要な無駄が多くなってしまう。Ｍｏｄｕｌａ－２では、使用するハードウェアを、割込みを使って同期するものと想定し、そのようなハードウェアの上で安価に実現できるコルーチンを組込みの同期プリミティブとしている。高レベルの同期命令（例えばホアのモニタやメッセージ交換など）を言語に組み込まなかったのは、想定されたハードウェアとの相違が大きすぎるためであろう。けれども、そのような高レベル同期命令がハードウェアに組み込まれている場合は、言語組込みの同期プリミティブとして、コルーチンが最良の選択だとは思われない。コルーチンでは、真の並行処理が記述できないからである。

なお、Ｍｏｄｕｌａ－２では、実時間プログラミングを行えない。

Ｍｏｄｕｌａ－２の各文に対する「実行時間」の概念が定義されていないからである。

◎参考文献

[１] N.Wirth, "Programming in Modula-2," 3rd ed., Springer-Verlag, Berlin, 1985.

[２] R.F.Sincovec and R.S.Wiener, "Data Structures using Modula-2," John Wiley & Sons, New York, 1986.