SR並行プログラムにおける依存関係についての考察

1J−6

蒲池正幸, 乃村能成, 笠原義晃, 程京徳, 牛島和夫
九州大学 工学部

1. はじめに

SRはアリゾナ大学で開発された新しい並行処理プログラミング言語である[1]。特徴としては次のようなものが挙げられる。

- ハードウェアやOSにおける並列処理・並行処理の実現方法の違いを意識することなく並行プログラムを記述することができる。
- プロセス間の同期・通信のための機構の種類が豊富である。

本論文ではSRで記述されたプログラムにおける依存関係を考えるときに問題となる点について述べ、それに対する考察を行う。

2. スワップ演算子

SRは2つの変数の間で値の交換を行うスワップ演算子を持っている。変数aの値と変数bの値の交換を行う場合には次のように記述する。

```
a :=: b
```

次のプログラムについてデータ依存を考えてみる。(左端の数字は行番号)

```
1: a := 1
2: b := 2
3: a :=: b
4: if a > 0 -> ... fi
5: if b > 0 -> ... fi
```

このプログラムのデータ依存関係をいままでのやり方で抽出し、グラフにすると図1のようになる。

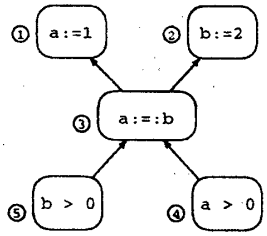

図1: データ依存グラフ

図中の各ノードはプログラムの各文(または制御述語)に対応している。また有向枝はノード間のデータ依存関係を表す。

5行目は3行目に依存している。3行目は1行目と2行目に依存しているから、5行目も1行目と2行目に依存していると考えられる。しかし実際には5行目におけるbの値は1行目でのaの値によってのみ決定し、2行目には依存しない。このように実際には依存関係のない箇所にまで依存関係があることになってしまう。

スワップ演算について依存関係を抽出すると、このように不正確な依存関係まで抽出することになる。このような依存関係を排除するの手段をいくつか検討する。

i) スワップ演算に対応するノードを2つに分け、aの値の定義を表すノードと、bの値の定義を表すノードとする。図2(1)に例を示す。これにより2つの交差しない依存関係の組が存在することになり、不必要な依存関係が抽出されることはない。しかし、この場合は各ノードは文(または制御述語)に対応するという原則が破られる。各ノードが何を表すのかについて考えなおす必要がある。

ii) 各データ依存枝に印を付けて区別する。あるノードで複数の変数の値が定義されている場合、そのノードからのデータ依存はどの変数について依存しているかを明確にする。そのために各枝にどの変数についてデータ依存するかを記述する。図2(2)に例を示す。これにより、変数aについてのデータ依存だけを辿るといったことが可能になる。

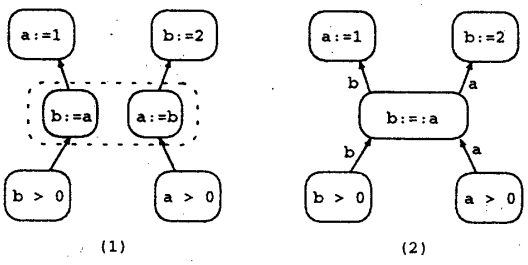

図2: データ依存グラフの例

3. if文

SRのif文の例を示す。

Some Considerations on Dependences in SR Concurrent Programs
M.Kamachi,Y.Nomura,Y.Kasahara,J.Cheng,K.Ushijima
Kyushu University

```
if 条件式 A -> ブロック A'
[] 条件式 B -> ブロック B'
[] 条件式 C -> ブロック C'
[] else -> ブロック D'
fi
```

いずれかの式が真になるか、すべての式が偽になるまで評価が行われる。真となった式に対応するブロック(すべて偽の場合は else に対応するブロック)が実行され、if 文は終了する。

複数の条件式がある場合、式を評価する順序は非決定的である。複数の式が同時に真になる場合には、そのうちの先に評価された式に対応するブロックが実行される。

ある条件式 A の評価した結果は必ず真になるものとする。他に真になる条件式 B が先に評価されれば、A が真でも A に対応するブロックの実行はない。

この if 文における制御依存について検討する。

一般のプログラミング言語では 1 つの制御述語により複数のブロックへの分岐が基本的な制御構造であり、複雑な構造についてもこれを組み合わせることで等価的に実現することができる。しかし、SR の if 文では非決定的順序に何らかの対応をしなければ解析することができない。

ここでは if 文中の制御依存を表現する手段についていくつか検討する。

i) 各制御述語は他の全ての制御述語に依存しているものと考える。例を図 3(1) に示す。
しかし、この場合あるノードは他のすべてのノードに依存することになる。これは正しい関係とは言い難い。

ii) if 文の各条件式の評価をまとめて 1 つのノードとして扱う。
全ての条件式を用いていずれのブロックを実行するかを決定するものと考え、これを 1 つのノードとして記述する。例を図 3(2) に示す。
非決定的な順序はノードの中に隠蔽されて、条件式同士での依存関係を考える必要がなくなる。
しかし、複雑な評価を行う箇所が 1 つのノードになることで、2 節と同様に、依存していない箇所まで依存関係として抽出してしまうような問題が起きるおそれがある。

また SR の if 文では変数への代入など副作用のある式を条件式として記述できることも問題になる。

各条件式は非決定的順序で評価されることから、式が評価される場合には副作用が生じるが、式が評価されない場合には副作用は生じない。

次のプログラムについて考えてみる。

```
if (a +:= 10) > b -> ...
[] (b +:= 20) > c -> ...
```

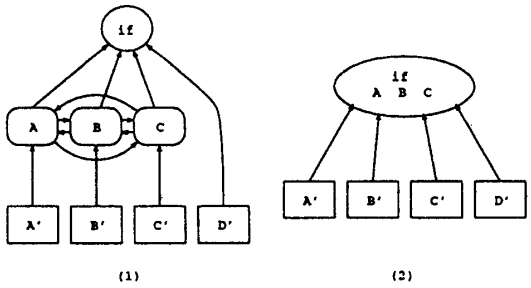

図 3: if 文の制御依存

```
[] (c +:= 30) > a -> ...
fi
```

それぞれの条件式の中で変数の値が定義されていて、その変数が他の条件式で参照されているためにどういう順序で評価するかによって、実行結果は全く異なる。このように決定的順序では起こり得ない不都合が、非決定的であるために起こる可能性がある。

非決定的であることは、条件式の評価の並列処理が可能になるなどの利点は考えられる。しかし、デバッグの難しさなどその短所の深刻さをつぐなえるほどの利点はないと考える。

4. 並列処理

SR はプロセス間の同期・通信のための機構として共有変数、非同期メッセージパス、セマフォ、リモートプロシージャコール、ランデヴをサポートしている。これらはすべてランデヴを用いて等価に記述できる。そのためランデヴについての依存関係を一般に表現できるならば、SR のプロセス間の同期・通信については問題なく扱える。このことから通信依存、同期依存については SR 特有の問題は生じないものと考える。

5. 最後に

スワップ演算子の問題は SR 以外の言語でも複数の変数に対する副作用を伴う演算子、関数、プロシージャなどで同様な問題について議論する必要があると考えられる。

SR ではプロセスの生成やプロセス間の通信や同期を簡潔に記述できるなどの新しい試みがなされている。しかし、SR は非決定的順序で評価を行う if 文などを見ると、解析やデバッグのやり易さについてはあまり考慮されていないように思われる。SR は実用的なプログラミング言語となるためにはこの点を改善することが不可欠であると考える。

参考文献

[1] G. R. Andrews and R .A. Olsson,"The SR Programming Language: Concurrency in Practice", Benjamin/Cummings, 1993