5C-10

DataFlowMachine用高級言語への指針(その2)

- 衝突検出同期の提案 -

竹岡尚三

(京都工芸繊維大学)

1. はじめに

データフロー・アーキテクチャはトークンがやって来るまでアクターが止まっているという性格から、イベント・ドリブンなシステムと整合性が良い。また、自然なプログラミングにとっては、I/O装置などからの緊急のアテンション（割込み）や同期の失敗等の例外処理は、それらのイベントで、例外処理部を起動するのがよい。しかしながら、既存の多くの言語処理系では、イベントをトリガーとして例外処理部の起動を記述することは、非常に困難である。

そこで本論では、データフロー・アーキテクチャ向きの全く新しい同期プリミティブとして、同期の失敗をイベント・ドリブンな例外動作として記述できる「衝突検出同期」を提案する。

次に「衝突検出同期」を使ったプログラミングを有効に実現するための新しい概念として「副作用の大きさ」を提案し導入する。

2. 衝突検出同期

ここで提案する新しい同期プリミティブとは

「クリティカルの侵入において2つ以上の競合が起こった場合、衝突というイベントによって、それまでWait状態にあった衝突処理部を起動する。」

というものである。

そして、このプリミティブは副作用の扱いに大きな特徴がある。

以下に、交差点をモデルに取って説明する。[図1]

i) プロセスP_1,P_2は信号を見ずに交差点に入る。

ii) たまに衝突することもある。

iii) P_1,P_2共、副作用S_1,S_2を出している。放っておくと副作用を出し続ける。

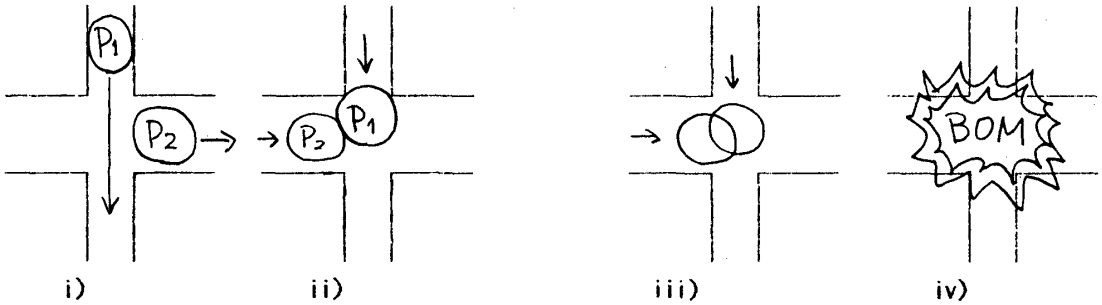
iv) 衝突を検出したらP_1,P_2の副作用の発生を止め、衝突処理部P_{EX}を起動する。ここで、P_1,P_2はクリティカルに入って副作用をまき散らしながら動作を続けていたのであるから、そのまま放って置いては不都合が生じる。そのため、P_{EX}では、P_1,P_2の副作用S_1,S_2を取り消すか、または覆い隠す様な副作用S_{EX}を発生させなければならない。

このように

「クリティカルに他のプロセスが入っていても、同期待ちをせずに構わず入って副作用を出すが、後でそれを覆い隠す様なより大きな副作用で辻妻を合わせる」

というのが衝突の起きた場合の副作用の取り扱いである。

ここに「より大きな副作用で辻妻を合わせる」と、書いたがより厳密な議論の為に「副作用の大きさ」という概念が必要になる。



[図1] 交差点のモデル

A Guide Line of High Level Language for Data Flow Machine (2)
- A New Primitive of Synchronization:Collision Detecting Synchronization -
TAKEOKA,Shozo
Kyoto Institute of Technology

3．副作用の大きさ

上記2．節におけるS_1,S_2の発生をどの程度まで許すのか、という点と、S_{EX}でS_1,S_2を覆い隠すことができるかを、明確に示すために「副作用の大きさ」を導入する。

S_1,S_2,S_{EX}のそれぞれに「副作用の大きさ」を定義する事によってS_1,S_2はどれだけ実行を進めても良いか、またS_{EX}はどれだけの副作用をもっていればS_1,S_2のリカバリーが可能かが決定できる。つまり「衝突検出同期」が正しく同期の機能を果たすためには、$S_{EX}>=S_1+S_2$ となる関係が満たせていなければならない。

4．データフロー・マシンでのインプリメント

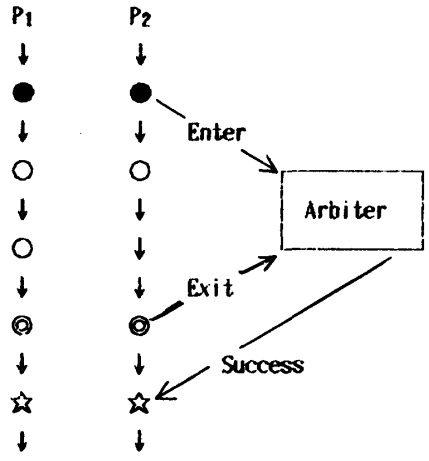
［図2］にフローグラフでの実現を示す。

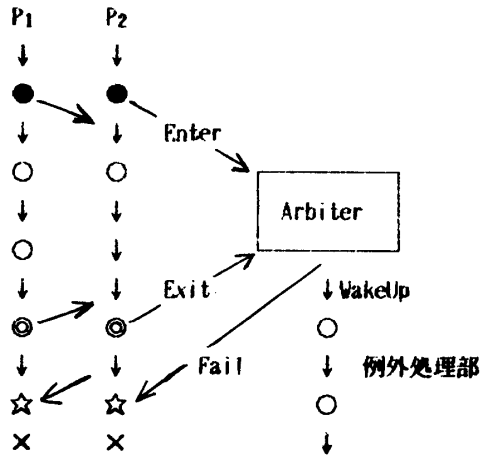
●は「クリティカルへの侵入(Enter)」をアービタ(Arbiter)へ知らせる。◎は「クリティカルからの退出(Exit)」をアービタへ知らせる。アービタは◎からExitを受け取ると☆へ、「衝突が起こらなかった(Success)」か、または「衝突が起こった(Fail)」かを送りつける。

☆は衝突が起こらなかった時(Success)はそのままプロセスの実行を続けるが、もし衝突が起きた場合(Fail)は衝突を起こしたプロセスを殺す(Kill)かまたは、休止(Suspend)させる。

そして、アービタは衝突が起きた場合には例外処理部を起動する(WakeUpを送る)。またクリティカル中で出る副作用とは、●から◎の間で発生する物である。よって副作用の大きさはかなり簡単に決定することができる。

☆はアービタからの結果を必ず待つ。これは本質的に同期待ちと同じ事である。つまり、不条理な動きをしないためには同期が必要だが、本論の方式では,同期待ちをなるべく遅らせて同期に失敗しなかった場合の実行を進めるということである。これを「同期の遅延評価」と呼ぶ。





［図2］フローグラフ　i) Success　　ii) Fail

5．衝突検出同期の特徴と考察

1)ハードウェアが同期を取る時間の間に実行が進む。
2)同期に失敗した時に何をするかが、うまく書ける。
3)自然現象のシミュレーションや　実時間制御において、衝突という現象がうまく記述できる。
4)同期に失敗した時と、割込み等のイベントからの例外処理部起動が同じ形でかける。
5)同期の遅延評価(Lazy Evaluation)、または緩い同期(Lenient Synchronization)である。
6)衝突を起こしたプロセスは基本的には殺されてしまうが、例外処理部によって元のプロセスを途中から再起動してリトライさせることができる。
7)アービタは　ラッチとオアだけで実現できる、非常に簡素な構造である。
8)この同期プリミティブはデータフロー・アーキテクチャに限らず有効である。
9)セマフォを実現するには、新しく◇(Protect)を導入する。◇はアービタへProtectを送り、これ以降　他人と競合があった時は自分のみが生き残り、他人は殺す事をアービタに知らせる。

◇と 6)のリトライ機能によって　セマフォを実現できるので、過去にあった同期プリミティブと同じ記述力がある。

6．おわりに

ここで提案した「衝突検出同期」は非常に有効であると考える。今後は、高級言語でのインプリメントを検討したい。