3R-6

並列オブジェクト指向言語DinnerBell
――非決定的な部分のデバッギング――

渡部眞幸，河野真治，田中英彦

東京大学　工学部

1. はじめに

DinnerBellは並列言語であるが、単一代入則に基づくpureな言語であるという仮定のもとでは（事実、旧版DinnerBellがそうであったように）各オブジェクトは副作用をもたず、並列性は大幅に制限されてしまう。しかしメッセージのjoin機構の導入［1］により、状態をもつオブジェクトが表現でき、より柔軟な並列プログラミングができるようになった。と同時に、プログラムの非決定性に起因するデバッグの困難もクローズアップされることとなる。非決定的な部分をデバッグする1つの方法としてペトリネットを利用することを考えてみた。

2. DinnerBellのメッセージjoin

メッセージjoinは、複数のプロセスからある1つのオブジェクトに非同期に送られてくるメッセージを待ち合せるための機構である。これにより送信側プロセスは全く独立に実行を進めることができ、並列度が向上する。また、メッセージをjoinするオブジェクトは状態をもつので、これを利用して排他制御も可能になる。プログラム1は、2つのプロセスを同期をとりつつ繰り返し実行するプログラムである。joinを用いないで同期をとるためには、一方が他方のプロセス終了を待たねばならず並列性が制約を受けざるを得ないが、joinによってごく自然な記述ができる。また、プログラム2は、joinを利用したレジスタである。クラスRegisterは、他プロセスからのaccessメッセージを再帰的に返されるstate メッセージとjoinすることによって複数アクセスの排他制御を実現している。

```
class ProsessJoin [
    process1!· process2! □ Process1 eval!.
                           Process2 eval! ]
```

プログラム1　プロセスの同期

```
class Register [
    new: X □ state: X;
    access: A· state:X □
        ((A inst!) =: 'read') yes:(□ ⇑X. state:X).
        ((A inst!) =: 'write') yes:(□ state:(A arg!)) ]
```

プログラム2　Register

3. DinnerBellのペトリネット表現

プログラム3に、DinnerBellによるtalkプログラムを示す。このプログラムでは、2台のKeyBoardが、key-inされたデータの出力メッセージを2台のDisplay に非同期に送り続けるが、Display に送られた複数のoutputメッセージが実行される順番は非決定的なのでKeyBoardで打った順にDisplay で表示してくれるという保証はない。'tea' と打ったのに'ate' と表示されてしまうかもしれないのである。このことは、ペトリネット表現にすると一層はっきりする。図1にプログラム3のペトリネット表現を示す。DinnerBellプログラムをペトリネットで表現する場合、原則として、

```
class Talk [
    between:User1 and:User2 □
        DA ⇐ Display newFor:User1 out:StdErr.
        DB ⇐ Display newFor:User2 out:StdErr.
        KeyBoard from:StdIn to:DA and:DB.
        KeyBoard from:StdIn to:DB and:DA          ]
class Display [
    newFor:~User out:Out □ to:Out;
    output:X· to:Out1 □
        Out2 ⇐ ((Out1 write:~User) write:X) nl!.
        Out2 wait: (□ to: Out2)                   ]
class KeyBoard [
    from:~In to:~A and:~B □ ack!;
    ack! □ split: (~In getC!).
           ack! ;
    split: Data □ ~A output: Data.
                  ~B output: Data ]
```

プログラム3　talk 1

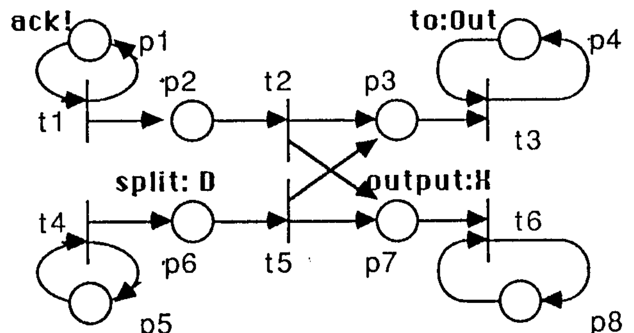

図1　talk 1のPetriNet表現

Concurrent Object Oriented Language 'DinnerBell'
Debugging on Nondeterministic Parts
Masaki WATANABE, Shinji KONO and Hidehiko TANAKA
The University of TOKYO

(1) メソッドはトランジションで表す。入力多重度はヘッド部でjoinされるメッセージの数に一致し、出力多重度はボディ部で並列に送信されるメッセージの数に一致する。

(2) メッセージはトークンで表す。したがってメッセージのたまり場であるプレースは1つのメソッドに特徴づけられたオブジェクトの閉包と考えられる。

を採用する。これによってsequencialな実行部分は1つのプレースにまとめられ、プログラムの非決定的部分のみが抽出される。

図1を見ると、t1では次々にsplit メッセージを送信しt3ではoutputメッセージを1つづつ受け取って処理しているが、両者が非同期に発火されるためp2やp3に複数のトークンがたまってしまう可能性があり、t2やt3がこれを処理する順序は必ずしもt1で生成された順ではない。これを修正するには、p4やp8からのacknowledge トークンをt1に返してp2やp3にたまるトークンが高々1個になるようにしなければならない。またp3はt2とt5の出力トークンを区別しなければならず、さらに複雑な制御が必要となる。

プログラム4に改良したtalkプログラムを示す。Display からのacknowledge をKeyBoardに返すことだけを念頭において人間の手で書き直したもので、何かのアルゴリズムにしたがってペトリネット表現を自動修正したりしたわけではない。このプログラムを逆にペトリネットで表現しようとすると、前述の(1)、(2)だけではできないことがわかる。すなわち、Display がTRUEを返す先はoutputメッセージを送信してきたオブジェクトであり、DinnerBellプログラム上ではそれを指示するsyntaxがあるのにペトリネット上ではトークンの送り先を動的に変化させられないからである。結局図1においてt2とt5両者からのoutputメッセージを同一プレースp3に送り込んだことが誤りであると考えられる。図2はこの点を考慮して書いたプログラム4のペトリネット表現である。以上の考察から、次の原則(3)を得る。

```
class Display [
    newFor:~User out:Out □ to:Out;
    output:X· to:Out1 □
        Out2 ⇐ ((Out1 write:~User) write:X) nl!.
        Out2 wait: (□ to: Out2. ⇑TRUE)              ]
class KeyBoard [
    from:~In to:~A and:~B □ ackA!.ackB!;
    ackA!·ackB! □ split: (~In getC!);
    split: Data □ (~A output: Data) yes:(□ ackA!).
                  (~B output: Data) yes:(□ ackB!) ]
```

プログラム4　talk 2

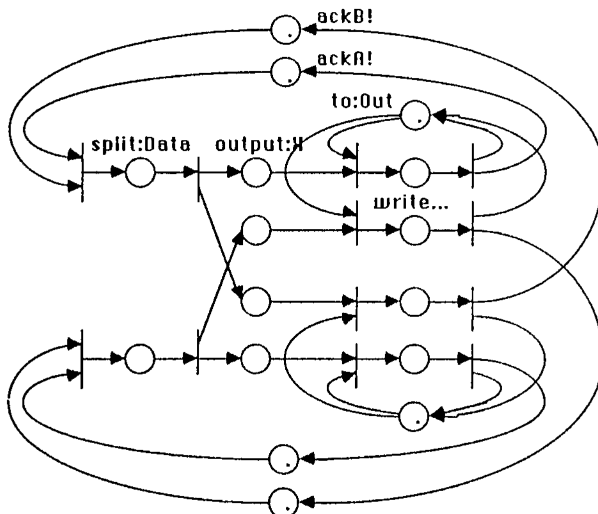

図2　talk 2 のPetriNet表現

(3) メッセージの送信先が動的に変化するようなプログラムをペトリネットで表現する時は、可能な全ての場合について、ペトリネット上でこれを複製しなければならない。

原則の(1)〜(3)をより厳密に定義することによって、プログラムからペトリネット表現を構成するアルゴリズムを導きだすことができるだろう。これに関連する研究として興味深いのは、μ式とペトリネットとの対応づけについて述べた文献［2］である。［2］のJコンビネータはまさにメッセージjoinに対応するし、Iはsequencialなメッセージ送信に、Fは複数メッセージの並列送信に対応し、Bは再帰メッセージとのjoinによる排他制御と関連する。しかし、joinにおいてreadとwrite を区別しない点、3つ以上のメッセージをjoin、forkするsyntaxを持っている点でDinnerBellプログラムはより自然なペトリネット表現を得られると思われる。いずれにしても今後の研究に何らかの形で参考になるだろう。

4. まとめ

並列に実行の進むMessage Passing の様子を把握するには、状態遷移図を書いたり、時相論理で順序関係を解析したりする方法も考えられるが、ペトリネット表現はDinnerBellプログラム中の非決定的部分を抽出してくれるので都合がよい。現段階ではアイデアのみであるが、プログラムとペトリネット相互の書替え、さらにペトリネット表現を用いたプログラムの仕様記述ができるようになれば、DinnerBellのデバッグに大いに役立つだろう。

参考文献

［1］　三尾，河野，田中，情報処理学会第34回全国大会，6U-7，(1987)

［2］　橘爪，神保，猪股，西村，情報処理学会第34回全国大会，6U-2，(1987)

［3］　Peterson，J.L.，"Petri Nets，" Comp.Surv. vol.9，no.3，pp.223-252，(1977)