4S-5 時相論理型言語 Tokio における非決定的実行

河野 真治, 中村 宏, 田中 英彦
(東京大学 工学部)

1 はじめに

時相論理による高いレベル Hardware 記述の利点は、単に細かいレベルの記述を抽象的に表現するだけでなく、実際の timing 情報の記述についても抽象的なレベルからより詳細なレベルまで記述を使用できる点にある。時相論理型言語 Tokio では、Interval Temporal Logic で記述した様々なレベルの Hardware 記述を直接実行することができ、Hardware 記述における rapid prototyping を実現している。この実行可能な記述の範囲を拡大する方法について考察する。

2 Tokio による Hardware 記述

Tokio のプログラムは、Prolog の Horn clause を ITL の時相演算により拡張したものになっている。ここで、ITL は、時相論理演算子として、@(先頭が、一つ短くなった時間区間), empty (空の時間区間)、&& (chop これは、一つの時間区間を二つに分けるという意味を持つ) を持つ。ITL では、変数の値、述語の真偽値ともに、各時間区間毎に定まる。よく使われる時相論理演算子の□ (いつも)、◇ (いつか) などもこれらを用いて表される。時区間の長さは、length(N) 特に、長さ 1 の場合は、skip という述語で指定する。

Tokio での Hardware 記述の基本は、&&と Horn clause を組み合わせた状態遷移記述と、and 論理演算子による並列実行記述であり、&&を使った逐次型の記述も使うことができる。時区間の長さや、逐次実行の指定、時相論理変数などにより、同期関係を記述する。この時に、状態遷移図などにより、正確な同期関係を記述しなくても、逐次性や、◇による時区間の同期などにより、必要な同期関係を最小限記述することができる。例えば、pipeline synthesis のような場合は、資源に対する待ち合わせがこのような同期の例となる。(Hardware 記述の例にはそぐわないが)Dining Philosopher では、以下のように、資源に対する id の割り振りというような形で記述される。(図 1)

もう一つの例は、Hardware 記述で良く出てくる特定の事象の間の時間関係である。例えば、send signal の後 3clock 以内に ack signal を出さなければならないという場合である。前の例に、時間制限をつけてみよう。(図 2)

これらの正確な実装のない同期記述は、すべて、Tokio による記述の非決定性を表している。つまり、その時点で同期を行なうかどうかの、非決定的な選択である。Tokio に導入される非決定性は、Horn clause の or から導入される Prolog と

```
ph_think(L,R,I) :- @free(L,R,I)
        && ph_left(L,R,I).
ph_left(L,R,I) :- @left(L,R,I), skip
        && ph_eat(L,R,I).
ph_eat(L,R,I) :- @both(L,R,I), length(3)
        && ph_left_off(L,R,I).
ph_left_off(L,R,I) :- @right(L,R,I),skip
        && ph_think(L,R,I).

left( I,_,(I,left)).
both( I,I,(I,eat)).
right(_,I,(I,right)).
free( _,_,(_,free)).
```

図 1: dining philosopher の例

```
less(N) :- M=N,less1(M).
less1(N) :- N =< 1,!, empty.
less1(N) :- @N = N-1,next(less1(N)).
ph_think(L,R,I) :- @free(L,R,I),less(8).
```

図 2: 時間制限

同じ性質のものと、時相論理演算子から来るものがある。このうち、時相論理演算子による非決定的な記述は、いわば時間を先読みした実行となる。Tokio での非決定的な部分の実行は、Prolog と同じく backtrack(後戻り) によって行なわれる。この backtrack は、同じ時間内での通常の backtrack と、時間を越えた過去への backtrack の 2 種類がある。時相論理演算子による backtrack でも通常の backtrack を生じる場合もある。

3 Tokio での非決定的な部分の実行

実際の backtrack は、記述と実行の差が明らかになった時、

- Unification の失敗
- empty と notEmpty の衝突
- 時区間の長さの衝突

この 3 つの場合に起きる。ここから、通常の Prolog と同じく、一番最近の非決定のあった部分 (choice point) までの実行を取り消す (undo)、そこで、別な選択肢を選ぶことになる。この場合の選択肢には、Horn clause の or の他に、時相論理演算子から生じる以下のような選択肢がある。

- 現在の時区間を終了するか否か (empty or notEmpty)
- &&の分割の仕方

実際の Tokio の実行で重要なのは後者である。例えば、

---

[0] Non-deterministic Execution in Temporal Logic Progarmming Language Tokio
Shinji KONO, Hiroshi NAKAMURA, Hidehiko TANAKA
University of Tokyo

```
test1 :-
        p(3,I),p(4,J),p(2,K),fin((I=J,J=K)),#write((I,J,K)).

p(J,K) :- K=J,K gets K+1 && stable(K).
```

図 3: 複雑な同期の人工的な例

この例では、最終的に三つの数字が等しくなるように、三つの時区間の長さを調整しなければならない。この選択は、Tokio での実行時刻が短くなるように、まずその場で時区間を可能ならば終らせるように選択される。

4 Tokio での実行可能な範囲

この backtrack は、FILO 形式で行なわれるために、すべての選択を均等に探すわけではないので、Tokio で記述できる範囲をすべて実行できるわけではない。したがって、Tokio の記述は、その実行の可能性によって以下のように分類することができる。RTL Tokio は、Tokio の時間を越えた非決定性を排除したものとなっている。Fair な backtrack の制御をした場合は、FILO よりも実行できる範囲が広がる。無限の backtrack を許せば、あらゆる場合が実行できるが、Tokio の記述の中には、実行不可能な記述も存在する。

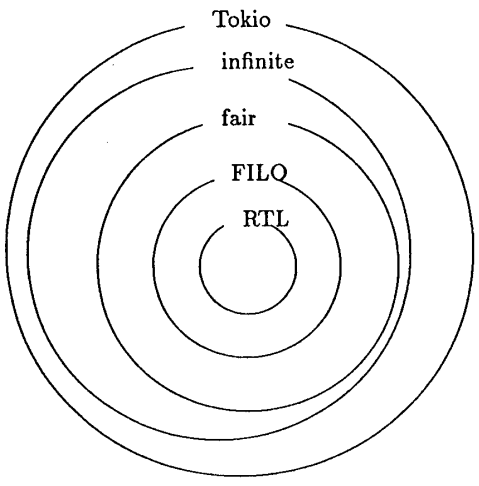

図 4: RTL/FILO backtrack/fair backtrack/infinite backtrack

5 Backtrack の制御

FILO 形式の backtrack で特に問題になる場合は、同期関係で fail した時に無制限に時区間を延長してしまう場合である。例えば、(図 3) の例では、このまま実行すると、その様になってしまう。

この時に、choice point の順序を制御する方法として、以下のような minimum 述語を使う方法がある。この方法では、上のように FILO でうまくいかない場合でも実行することができる。

```
minimum :- minimum(0).

minimum(I) :- length(I).
minimum(I) :- K = I+1,minimum(K).

test :- minimum,
        p(3,I),p(4,J),p(2,K),fin((I=J,J=K)),#write((I,J,K)).
```

図 5: minimum 述語

しかし、この方法では、必ず minimum のところまで backtrack が起こるのであまり効率が良くない。したがって、一旦この選択肢の移動が必要なくなった時点で、新しく choice point を移動する必要がある。このためには、Tokio の記述をこの時点で、前半と後半の 2 つの時区間に分割する必要があり、これは、逐次型の記述を用いている時には、非常に不便である。

6 Constraint の利用

このような場合に対しては、[1] のような Constraint 計算を行なうことも考えられる。Tokio の時区間の長さの関係は、length 述語と&&を使うことにより、かけ算と足し算の不等式で表せるので、基本的には、Presburger Logic であり、実際に充足可能性を計算することができる。ただし、Tokio での実行時の同期関係を、この方法で解くには、時間がかかりすぎると考えられる。

7 Choice Point の fair 選択

この問題は、基本的には、一つの時区間を複数の仕方で分割する時に、その分割の仕方の fairness を保証することにある。例えば、dining philosopher では、eat, think などの各 phase に対する長さの配分を fair にする必要がある。しかし、すべての分割でこれを保証することは必要ないし、手間もかかるので、特定の述語に対して、これを保証することになる。このための道具としては、Tokio の Program 自体で行なう minimum のような方法や、Tokio compiler に特定の述語として内蔵させる方法がある。

実際の Hardware では、backtrack による同期は、具体的な機構としてよういしなければならない。これは、例えば、pipe line synthesis の場合のように、同期に反する遷移を禁止するという方法や、round robin schedular などを実装する方法が考えられる。

参考文献

[1] 大木優, 沢本潤, 藤井裕一. Sup-inf 法に基づいた制約論理型プログラミング言語. 日本ソフトウェア学会 5 回大会, Vol. C1, No. 1,, 1988.