2J-8

時相論理型言語Tokio のコンパイラ

青柳龍也　河野真治　藤田昌宏＊田中英彦　元岡達
東大　工学部　＊現在　富士通研

0) Tokio コンパイラ

ここでは，Tokio のコンパイラについて考察する。現在のTokio はスケジューラを持つ擬似並列言語であるが，基本的な実行は逐次型Prologと同じである。各縮約毎に，スケジューリングするConcurrent Prolog などと異なり，論理的なクロック毎に実行する為に，スケジューラーへの負荷は軽く，より効率的にコンパイルすることができる。また，Tokio は，幾つかの重要な二階の述語と，関数の展開も行う。

1) Abstract Prolog Instruction set の拡張

Tokio のコンパイラコードは，D.H.D.Warrenの Abstract Prolog Instruction set 〔1〕の拡張となっている。ここではその拡張部分について述べる。Tokio の基本的な実行は，時相論理変数の単一化・未来への縮退・時区間の分割の三つの部分よりなる。拡張コードもこれに対応して分類することができる。

1－1) 単一化の拡張コード

時相論理変数は，$t をノードとして持つリストにより表現される。全時刻単一化の時は，このリスト全てが単一化される。単一時間の単一化の時には，新しくノードをつくる必要がある。

get-value,unify-value

これらの，単一化を必要とするコードは，時間論理変数の単一化をサポートする為に拡張される。この拡張は，通常の単一化と，構造体を必要に応じて，現在の部分と未来の部分にコピーする部分がらなる。

get-tempvar

$t ノードを新しく生成する命令である。このコードは，'=' により，一つの時刻に対する値を取り出す時に用いる。動作は，get-structure とほとんど同じである。$t の構造体は通常処理系のそとからは見えない。

1－2) 縮約の為の拡張コード

Tokio の擬似並列実行は，他の言語と同様にキューイングすることによって行われる。Tokio のキューは，三種存在し，それぞれ，next，fin，keep に対応する。これらのキューへの挿入を行う命令が存在する。これらの呼出しは，通常のcallと同じ形式であり，pseudo call と呼ぶ。fin，keep は，# (always) を含んでいるので，以下で述べるalwaysの展開も同時に行う。

next queueに挿入されたリテラルは，次の時刻で実行される。fin，keep queueに挿入されたリテラルは，そのリテラルの評価されている時区間の長さが確定した時点で実行される。時区間の長さの指定には，次の評価型述語を用いる。

```
pseudo call

        next    label
        wnext   label
        keep    label,nextLabel
        fin     label,nextLabel

temporal variable

        get_tempvar     Rx
        unify_value     Rx
        get_value       Rx,Rx

subintervals

        sub_begin
        subend
        chop    label
```

Fig.1　Tokio の拡張コード

empty　現在時刻で，この時区間が終了する。

length N　あと，N クロックで終了する。

nextには，現在時点で時区間が終了すると，真となるweek nextと，偽となるstrong next の二種がある。それぞれ，wnext,nextのコードが対応する。

1－3) 時区間の分割の為の拡張コード

'&&' (chop) 演算子は，ITL の中心的な演算子あり，新しい時区間を生成する。時区間は，その時区間の終りの時間と，その時区間で実行すべきリテラル，その時区間が終了してから実行すべきリテラルの三つの部分よりなる。時区間に切替えは，次の三つの命令により行う。

sub-begin　前半のリテラルの始まりを示す。

sub-end　前半のリテラルの終りを示す。

chop　後半のリテラルのpseudo call

sub-begin ／sub-end は，時区間の終りを表す変数を，スタックに積んで，前半の区間を評価する。前半の区間での次の時間への縮退は，次の時刻で時区間の切替えをするsub-begin ／sub-end と共に，next queueに積まれる。chopは，後半の区間を評価するかしないかを決定する。後半の区間は，実行される時は外の時区間と一致しているので，sub-begin ／sub-end で時区間を切替える必要はない。

2) Second Orderの展開

Tokio の様相演算子は，二階の述語とみることもできる。# (always) は，○をweek next として，次の様に定義することができる。

#P :- P, ○#P.

コンパイラでは，alwaysはこのように展開される。fin, keepも，次の様に展開される。

fin P :-if empty then P else @fin P.

keep P :-if not empty then P, @keep P.

Tokio の特徴の一つは，ループの抽象である。例えば，while do ループは，次の様に定義される。

```
while A do B  :-
   if A then   (B && while A do B )
       else   empty.
```

インタプリタは，実際に，この定義により動作する。しかし，コンパイラでは，次のように展開して実行する。

```
while I<5 do  @I = I+1
```

↓

```
while00 (I)   :-
   if I <5 then
      (@I =I+1 &&  while00 (I) )
   else empty.
```

これは，一種のマクロ展開である。

3) 関数

Tokio では，'=' の両辺で評価される関数をもっている。もっとも基本的な関数は，@である。通常の述語の中に，関数としての@が引数として現われることは許していない。これは，次のような場合に，時間的なループが形成される為である。

～, eq (A, @A), ～

eq (X, X).

この様なループの単一化には，少し手間がかかる。

Tokio の関数には，引数が時間的に変化すると，値も変化するという特徴がある。たとえば，A＋1は，Aの値により，各時刻ごとに値がことなる。関数で問題になるのは，いつ評価されるかという点である。Temporal assignmentを次のような定義で与える。

B<－A :－

A＝C, stable (C), fin (C＝B).

ここで，A<－A＋1を考える。A＋1は，Temporal assignmentの定義のなかで評価してもよい。インタプリタではそうなっている。コンパイラでは，関数の呼出しは，二階の述語に近いので，実行前に展開することがやはり望ましい。

4) まとめ

Tokio は，D.H.D.Warrenのコードを拡張することにより，容易にコンパイルすることができる。しかし，効率的にコンパイルするためには，適切な二階の述語の展開が必要である。

参考文献

D.H.D.Warren, "AN ABSTRACT PROLOG INSTRUCTION SET ", Technical Note 309, SRI International, (1983)

```
%
%       A <- B  :-      C<--B,fin(C=A).
%
%       p(A) :- A <- A+1.
%       p(A) :- C is A+1,fin(A=C).

p/1:
        get_tempvar     A1              %       p(A) :-
        unify_variable  X2              %       A = $t(
                                        %        nowA,
        unfiy_void      1               %        @A),
        calc            X2+1,X3         %       +1 --> C,
        put_variable    X3,A2           %       fin_0(A,C
        execute         fin_1/2         %       ).

fin_1/2:                                %       fin_1(A,C) :-
        fin             fin_2/2,fin_1/2 %       fin(fin_2(A,C)),@fin_1(A,C
        proceed                         %       ).

fin_2/2:                                %       fin_2(A,C) :-
        get_tempvar     A1              %       A = $t(
        unify_variable  X3              %        NowA,
        unify_void      1               %        @A),
        get_tempvar     A2              %       C = $t(
        unify_variable  X3              %        NowC = NowA,
        unify_void      1               %        @C)
        proceed                         %
```

Fig. 2 サンプルコンパイル