4R−8

プログラム実行監視用コードの生成について

伊藤英樹　宇都宮公訓　藤原譲
筑波大学　電子・情報工学系

1　はじめに

主記憶を共有する多重プロセッサ系を活用して効率的にプログラムの実行を監視するシステムを提案した[1]。今回は、そのような監視システム上で実際に監視を行うためのコードを生成する具体的手法について報告する。

著者等の実行監視システムの概念図を図1に示す。実行コードは応用そのもののためのコードであり、監視コードは実行コードを監視するためのコードである。両者は並列に動くが、実行コードは監視コードの監視下にある。

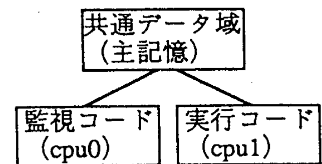

図1　監視システム

実行コードは通常のコンパイラが生成するが、そのコンパイラに手を加えることによって、監視コードも容易に生成できることを示す。話を具体的にするために、プログラム言語としてPascalを選び、Pysterのコンパイラ[2]を用いることにする。

2　監視内容とコードの枠組

本報告では次の5つの監視だけを考える。

(1)　特定の文の実行前の中断(bef_brk)と実行後の中断(aft_brk)

(2)　特定の変数の参照による中断(ref_brk)

(3)　特定の変数の更新による中断(upd_brk)

(4)　変数の値に対する特定の条件の成立による中断(cnd_brk)

(5)　値が未定義の変数を参照することによる中断(und_brk)

代入文に対して生成されるコードを図2に示す。この場合が最も複雑なコードになる。指定されていれば、監視コードの先頭、棹尾でそれぞれbef_brk、aft_brkを発生させる。ref_brk、und_brkが指定されていれば、record_ref_brk等の部分にそのためのコードが生成される。upd_brk、cnd_brkが指定されているときは、record_upd_brk等の部分にそのためのコードが生成される。実際の中断はcheck_ref_etc、check_upd_etcの部分で発生させる。

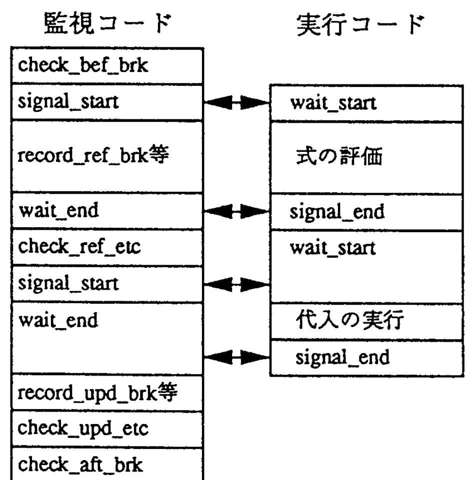

図2　代入文のコードの枠組

3　コンパイラの改造

PysterのPascalコンパイラは基本的にLL(1)翻訳文法になっている。パーサの適当な個所に新しいアクション記号を追加し、それに対するアクションルーチン

On a Generation of Code for Monitoring the Execution of Program
Hideki ITOH, Kiminori UTSUNOMIYA, Yuzuru FUJIWARA
University of TSUKUBA

を追加したり、既存のアクションルーチンを変更することによって、コンパイラを改造する。変更を施したいくつかの部分を以下に示す。

代入文:

```
ASSIGN_STMT → NON_KEY_IDx push_operand(x) ':=' push_operator(':=')
    m_code(check_bef_break) m_code(signal_start)e_code(wait_start)
    EXPRESS ';' e_code(wait_start) m_code(wait_end) m_code(check_ref_etc)
    move(pop_operand,work1) move(pop_operand,work2) e_code(pop_operator,work1,work2)
    e_code(signal_end) m_code(wait_end) m_code(record_upd_brk,work2)
    m_code(record_cnd_brk,work2) def(work2) m_code(check_upd_etc) m_code(check_aft_brk)
```

※move(i,j)は、iの値をjに移す手続きの呼出し

m_code(check_ref_etc)は、ref_brk成立、未定義変数参照の発生による中断を生じる監視コードを生成する手続きの呼出し

m_code(check_upd_etc)は、upd_brk成立、cnd_brk成立による中断を生じる監視コードを生成する手続きの呼出し

def(work2)は、work2中のidentifierに値が代入されたことを示す印をつける手続きの呼出し

式(EXPRESS)の一部分:

```
PART →'not' push_operator('not') PART e_code(pop_operator,pop_operand)
     →( NON_KEY_IDx m_code(record_ref_brk,x) m_code(record_und_brk,x) |
        INTEGERx | BOOLEANx ) push_operand(x)
     →'(' EXPRESS ')'
     →'+' ( '(' EXPRESS ')' | ( INTEGERx | NON_KEY_IDx m_code(record_ref_brk,x)
        m_code(record_und_brk,x)) push_operand(x) )
     →'-' ( '(' EXPRESS ')' e_code('neg',pop_operand) |
        INTEGERx push_operand('-'||x) |
        NON_KEY_IDx e_code('neg',x) m_code(record_ref_brk,x)
        m_code(record_und_brk,x))
```

IF文:

```
IF_STMT → 'if' m_code(check_bef_brk) m_code(signal_start) e_code(wait_start)
    EXPRESS e_code(signal_end) m_code(wait_end) m_code(check_ref_etc)
    m_code(signal_start) e_code(wait_start) 'then' move(pop_operand,work)
    m_code(then,work) e_code(then,work) m_code(signal_start) e_code(wait_start)
    EXEC_STMT ELSE_PT
ELSE_PT → 'else' move(pop_operand,work1) m_code(else,work1) e_code(else,work1)
    m_code(signal_start) e_code(wait_start) EXEC_STMT move(pop_operand,work1)
    m_code(post_if,work1) e_code(post_if,work1) e_code(signal_end) m_code(wait_end)
    m_code(check_aft_brk)
```

WHILE文:

```
WHILE_STMT → 'while' m_code(check_bef_brk) m_code(signal_start)
    e_code(wait_start) m_code(while) e_code(while)EXPRESS e_code(signal_end)
    m_code(wait_end) m_code(check_ref_etc) 'do' m_code(signal_start)
    e_code(wait_start) move(pop_operand,work1) m_code(do,work1) e_code(do,work1)
    m_code(signal_start) e_code(wait_start) EXEC_STMT m_code(signal_start)
    e_code(wait_start) move(pop_operand,work1) move(pop_operand,work2)
    m_code(post_while,work1,work2) e_code(post_while,work1,work2)
    e_code(signal_end) m_code(wait_end) m_code(check_aft_brk)
```

4 おわりに

今回発表した内容は第0版とでもいうべきもので、骨格部分に過ぎない。手続き、関数、ブロック構造、エラー処理等のための拡張を行ない、本システム上で動的監視を基盤とするツール、例えば、シンボリックデバッガをインプリメントすることにしている。

参考文献

[1] 宇都宮他: 多重プロセッサ系によるプログラム実行監視システムについて, 情報処理学会第34回全国大会予稿集, 1987.

[2] Arthur B. Pyster:COMPILER DESIGN AND CONSTRUCTION,Van Nostrand Reinhold Co., 1980.