Common ESP処理系における最適化の一考察(2)

5J－4

松浦 聡　田中 吉廣　実近 憲昭
㈱ AI言語研究所

1．はじめに

Common ESP(CESP)はICOTで開発されたESPを基に、汎用ワークステーション上で稼働するシステムを構築し、さらに機能を高めた言語である。言語体系としては、論理型とオブジェクト指向型の両パラダイムを融合した言語であり、AIプログラムの開発および、システム記述に適している。現在CESPは暫定版の開発を終了し、これに基づき、最終版の基礎となる基本仕様版の開発途中である。本論文はCESP基本仕様版における論理型言語部分の実行方式についての最適化とその性能評価について述べる。

2．CESP基本仕様版で採用した方式

前年度に開発したCESP暫定版では、WAMのエミュレータ方式で処理系を開発した[1]。今年度開発中の基本仕様版ではWAMから機械語を生成する方式を採用している。我々は基本仕様版にすでに以下のWAMレベルの最適化を施した[2]。

(1) allocate,deallocate の省略
(2) if_then_else命令の強化
　(jump_unless_constant_less_than など)
(3) 内部データベース用命令の追加
　(get_arguments)

3．CESP基本仕様版の最適化方式

今回は特にPrologの基本実行制御であるバックトラックについてのWAMレベルの最適化を検討した。CESP基本仕様版では、コンパイラによりソースプログラムをWAMの中間言語まで翻訳し、その後、機械語を生成する方式で開発を進めている。我々は、このWAMレベルの実証評価を行うため、最適化を組み込んだ処理系を作成し、最適化に関する性能評価を行った。次の項目について最適化を検討した。

(1) 高速化try_me_else,try 命令の追加
　(fast_try_me_else,fast_try)
(2) allocate命令の遅延実行

以下、これらの最適化の内容について述べる。

3.1 高速化try_me_else,try 命令の追加

CESPでは、プログラムに選択肢が存在すると、choice point stackに選択肢フレームが作成される。この選択肢フレームには引数レジスタがつまれるが、引数レジスタが破壊されることが無い場合には引数レジスタの保存は無駄である。そこで我々は引数レジスタの保存を行わないtry_me_else 命令として、fast_try_me_else命令を導入した。そしてfast_try_me_else命令によって作成される選択肢フレームをfast_choice_point_frame と命名した。図1にfast_choice_point_frame の構造を示す。fast_choice_point_frame に引数レジスタの領域を確保してある理由は、例外発生時およびbind_hook-ハンドラ起動時には引数レジスタを保存する必要があるためである。

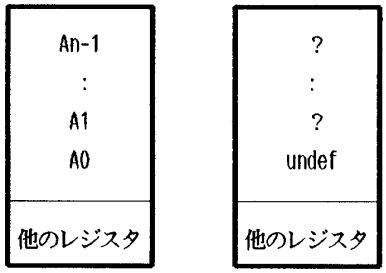

図1 fast_frameの構造

fast_try_me_else命令を出すのは、次の時である。

・最後の節以外のすべての節で引数レジスタの内容を破壊する前にカットが出現する。

以前、我々は最適化のためにif_then_else命令を導入したが、これらの命令は引数レジスタを破壊してはならないだけでなく変数に代入を行ってもいけなかった。今回は引数レジスタを破壊しなければよいのでかなり適用範囲が広がる。fast_try_me_elseが出る例を示す。

An optimizu method for Common ESP System(2)
Satoshi Matsuura,Yoshihiro Tanaka,Noriaki Sanechika
AI Language Research Institute,Ltd.

【例1】

```
append([],X,X) :- !;
append([A|X],Y,[A|Z]) :-
    append(X,Y,Z);
```

```
$append     jump_on_non_list    $nil,$var,$else
$list       get_vart_vart_list  A0,A3,A0
            get_valt_vart_list  A2,A3,A2
            execute             $append
$nil        get_nil             A0
            get_value_t         A2,A1
            cut_me_and_proceed
$var        fast_try            3,$nil
            trust               $list
$else       fail
```

図2　fast_tryの適用例（プログラム1　append）

【例2】

```
max(A,B,A) :-
    A >= B,
    !;
max(A,B,B);
```

```
$max        fast_try_me_else    3,$max_2
            get_value_t         A0,A2
            not_less_than       A0,A1
            cut_me_and_proceed
$max_2      trust_me_else_fail
            get_value_t         A1,A2
            proceed
```

図3　fast_try_me_elseの適用例(プログラム2　max)

【例3】

```
delete([A|X],A,X) :- !;
delete([A|X],B,[A|Y]) :-
    delete(x,B,Y);
```

```
$delete     jump_on_non_list    A0,$nil,$var,$else
$var        fast_try_me_else    3,$delete2
            get_list            A0
            unify_local_value_t A1
            unify_local_value_t A2
            cut_me_and_proceed
$delete2    trust_me_else_fail
            get_vart_vart_list  A0,A3,A0
            get_valt_vart_list  A2,A3,A2
            execute             $delete
$nil
$else       fail
```

図4　fast_try_me_elseの適用例(プログラム3　delete)

3.2 allocate命令の遅延実行

allocate命令は環境フレームを作成するために節の初めに出すのが普通であるが、ヘッドユニフケーションでバックトラックが発生する場合には環境フレームを取り除かなければならない。我々はallocate命令を遅延させることによりバックトラックですぐ取り除かれる可能性のある環境フレームをなるべく作成しないようにした。allocate命令の遅延実行の例を示す。

【例4】

```
p(0,X) :- q(X),r(X);
```

〖従来〗

```
$p          allocate            1
            get_integer         A0,int!0
            get_variable_p      A1,Y0
                  :
```

〖現在〗

```
$p          get_integer         A0,int!0
            allocate            1
            get_variable_p      A1,Y0
                  :
```

図5　allocate命令の遅延実行の適用例(プログラム4)

4．性能評価

上記のプログラム2～4に対して速度性能をＣＥＳＰ基本仕様版で測定した。測定したマシンはＳＵＮ－３／６０（３ＭＩＰＳ）である。それぞれのプログラムに対してバックトラックのある場合とない場合について速度を測定した。

表1　速度性能評価結果（高速化try-me-elsef命令）

	従来(LIPS)	改良(LIPS)	性能比(%)
max (*1)	6316	8219	130
max (*2)	11429	12632	111
delete(*1)	4166	4839	116
delete(*2)	9524	9918	104

表2　速度性能評価結果（allocate命令の遅延実行）

	従来(LIPS)	改良(LIPS)	性能比(%)
p (*1)	7595	8392	111
p (*2)	16217	16439	101

(*1)　バックトラックあり。

(*2)　バックトラックなし。

5．まとめと今後の課題

ＣＥＳＰ基本仕様版の処理系におけるバックトラック時の最適化技術について述べた。fast_tryによりバックトラックありの場合で３０％，なしの場合で１１％の速度向上がみられた。

allocateの遅延はあまり効果がなかったが、簡単に実現できたので採用した。今後はＣＥＳＰシステムでの性能評価を行っていく予定である。

また、研究課題としては、出現頻度の多いＷＡＭの組み合わせの検出による最適化の適用範囲拡大やモード宣言等のプラグマ導入時による最適化がある。

これらの研究により、ＣＥＳＰの高速実行を推進していきたい。

6．参考文献

[1] 田中ほか :暫定版Common ESPシステムにおける実行方式
第３８回情報処理学会全国大会(1989.3)

[2] 松浦ほか :Common ESP処理系における最適化の一考察(1)
第３９回情報処理学会全国大会(1989.10)