1K−10

Common ESP プログラム解析ツールの開発

松浦 聡[1]、佐々木 玲子[2]、実近 憲昭[1]

[1](株)AI言語研究所、[2]富士ソフトウエア(株)

1 はじめに

論理型言語はプログラムが論理的に記述できるため、既存の手続き型言語よりプログラムの生産性がよいことで知られている。その反面、CやLispに比べてデバッグツールの不足やプログラムテクニックの未熟さから、効率の悪いプログラムが多く書かれている。我々はこの問題を解決するためにオブジェクト指向論理型言語Common ESP(CESP[1])用にプログラム静的解析ツールを開発した。CESPは論理型言語とオブジェクト指向言語の両方の特徴を備えているが今回開発したプログラム解析ツールはCESPの論理型言語として記述された部分に作用する。このシステムは次の機能を含んでいる。

1. CESPの文法チェック機能
2. 不要な述語、不要な引数の発見
3. 実行効率の悪いプログラムへのアドバイス
4. 引数のモードの自動判定

さらに我々はこのシステムでの解析結果をCESP機械語生成部[2]で利用し、効率のよいコードを生成する研究をすすめている。

2 プログラム解析ツール

プログラムの開発において強力なデバッグ環境は必要不可欠である。これまでに我々は、CESP処理系にインスペクタ、ソースレベルデバッガなどのデバッグツールを導入してきたが、これらのツールは動的デバッグツールであり、バグのある箇所まで実行しないかぎりデバッグができない。そのため、早期にバグを発見する方法として静的なデバッグツールの開発が望まれていた。また、プログラムのデバッグが終了しても、プログラムの中にはデバッグ中にいれた不要なコードが残っていることが多い。そのため、不要なコードを検出してくれる機能があればユーザにとってはありがたい。

さらに、静的解析によって不必要な例外の判断などが削除できればプログラムの高速化、メモリの削減に役に立つ。

以上のような理由から、我々はプログラム解析ツールとしてCESP文法解析ツールとCESPプログラム構造解析ツールを開発中である。図1にこれらのツールの構成図を示す。

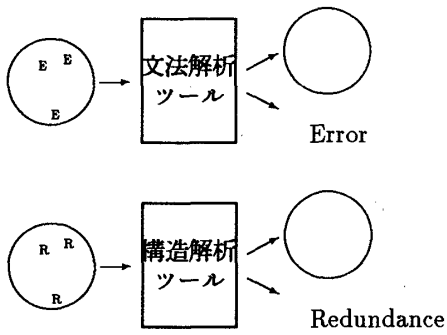


図1: CESPプログラム解析ツール構成図

2.1 CESP文法解析ツール

プログラムの開発において文法エラーは必ず発生するものである。これらのバグはコンパイル時に判明するがコンパイラはコード生成も同時に行なうので文法エラーを発見するツールとしては問題がある。そこで、我々はCESPの文法を理解し、構文をチェックしてくれるツールとして、CESP文法解析ツールを開発した。このツールの機能としては

- CESPの文法解析
- 組み込み述語の引数チェック
- 未定義述語の検出、不要な述語の検出
- アトム、変数のクロスリファレンス

がある。文法解析ツールはCESP処理系とは独立して実行できるのでCESPが動かない計算機の上でもCESPの文法チェックができる。図2に文法解析ツールを使用した時のデバッグ時間の比較を示す。

2.2 CESPプログラム構造解析ツール

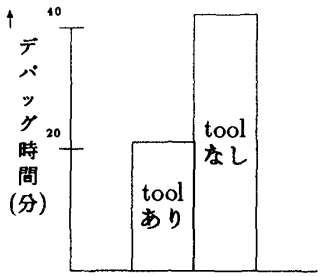
論理型言語において不必要な選択肢を残すことは、実行効率、メモリ効率、デバッグの効率からみて好ましいことではない。また、インデキシングのかからないプログラムは著しく実行効率を下げる。そこで、我々はプログラムの構造を理解し、欠点を指摘するツールとしてCESPプログラム構造解析ツールを開発中である。

その機能としては

- カットが無い述語の検出

---

[0]Development of Program Analysis Tools on Common ESP
Satoshi Matsuura[1] Reiko Sasaki[2] Noriaki Sanechika[1]
[1]AI Language Research Institute, Ltd. [2] Fuji Software Inc.



文法エラーが１０箇所あるプログラムのデバッグに要した時間

図 2: デバッグ時間の比較

```
?- cesp_analyzer.
Filename>max.esp
Reading max.esp
Non-det Checking...
Indexing Checking...
You writes:
-----------
        max(A,B,A):-A>B,!;
-----------
But I think You have to
-----------
        max(A,B,A___1):-A>B,!,A=A___1;
-----------
```

図 3: プログラム解析ツールの実行例

- インデキシングがかからない述語の検出およびアドバイス
- 述語の引数のモードの検出

がある。図 3 に CESP プログラム構造解析ツールの実行例を示す。

3 機械語生成への応用

現在、我々は機械語生成方式で CESP 処理系を開発しているが、プログラム構造解析ツールの解析結果をコード生成に応用する研究をすすめている。以下にコード生成の応用例を示す。

3.1 例外チェックの省略

CESP では動的デバッグの方法として例外処理機能を備えている。例えば、組み込み述語 subtract では、Illegal Input,Type mismatch, Integer Overflow,Flowting-point Overflow の例外が発生する。しかし、組み込み述語に渡ってくる引数が事前に判明すればチェックを省略することが可能である。例えば図 4 のプログラムでは subtract に渡ってくる引数は正の整数なので subtract での例外チェックは必要ない。

ループのプログラム

```
      :
   loop(100),
      :

loop(0) :- !;
loop(N) :-
    subtract(N,1,N1),
    loop(N1);
```

従来のコード生成

```
        movl    d2,a0
        movl    #1,a1
        jbsr    _subtract_constant2
        movl    d1,d2
        tstl    d0
        jeq     L_next
        jbsr    L_exception
```

例外チェックを省略したコード生成

```
        subql #1,d2
```

図 4: 例外チェックの省略

3.2 ユニフィケーションの最適化

ヘッドユニフィケーションにおいて渡ってくる引数の型が判明していればさまざまな最適化が可能であることは周知の事実である。例えば CESP では append の第１引数に対して、未定義変数用、Hook 変数用、リスト用、nil（[]）用の処理が用意されているが、必ずリストか nil がくることが判明すればコードが小さくすることが可能である。

4 まとめと今後の課題

我々はプログラムのデバッグ用に文法解析ツールおよびプログラム構造解析ツールを開発した。文法解析ツールにより、プログラムの開発時間の短縮が可能となった。また、プログラム構造解析ツールにより、効率の悪いプログラムを自動的に発見できるようになった。今後の課題としては機械語生成部への応用と、計算効率の悪いプログラムを改善するツールの開発を行なっていきたい。

参考文献

[1] 中澤 修、実近 憲昭 「CESPer への道」 1990 bit ９０年１１月号

[2] 佐藤 良治、高橋 文男、田中 吉廣、実近 憲昭 「Common ESP 機械語ジェネレータの概要」 1990.3 第４０回情報処理学会全国大会