6Q-3

FLENGコンパイラとその抽象化コード

下山 健, 島田 健太郎, 小池 汎平, 田中英彦
(東京大学 工学部)

1 はじめに

我々は、大規模な知識処理を行なうためにcommitted-choice型言語FLENG[3]を提案している。FLENGには、上位言語としてFLENG++、低水準言語に、FLENG--がある。FLENG++は、FLENGのオブジェクト指向プログラミングを支援する上位言語でありFLENGにコンパイルされる。一方FLENG--は、FLENGに、処理効率向上のために様々なアノテーションを追加して、低水準な動作の記述を可能にしている言語である。FLENG--は、FLENGのスーパーセット言語であり、FLENGで書かれたプログラムもそのまま実行できる。我々は、FLENGおよびFLENG--を汎用型計算機で高速に実行させるために、FLENGコンパイラを製作している。以下にそのFLENGコンパイラの概要を説明する。

2 タグの構成

現在、FLENGコンパイラはSUN-4上で製作されており、最終的にはSPARCのコードに落とす予定である。したがって、SPARCのアーキテクチャにあったタグアーキテクチャを考慮しなくてはいけない。論理型言語の専用ハードウェアを持ったマシンでは、1ワードを32～40ビットとしているものが多いが、既存のCPUでシステムを構成する場合、1ワードは32ビットとなる。また、SPARCは、32ビット(1ワード)の下位2ビットをタグとして使用できるように考慮されており、Tagged Add, Tagged Subtractなどの命令や、Tag Overflow Trapも用意され、これらの命令はコンパイラの開発時のエラー検出に有効に利用できる。そこで、セルをワード境界におくことにより、下位2ビットをタグとして利用する。しかし、タグは4種類では足りないので、下位2ビットが00のときは、タグを拡張して上位のビットもタグとして利用することにする。図1に、タグの構成を示す。図中で、mark bitの部分はGCや、アノテーションのマークに利用する。

3 命令体系

PROLOGなどの論理型言語の抽象化命令セットとしては、WAMコード[1]が最も標準的であり、広く普及している。また、GHCなどのcommitted-choice言語に対応する、抽象化命令セットとしてはKL1B[2]がある。FLENGは、GHCと比べてGuard部がないなど言語要素を簡潔にして並列計算機への実装を容易にしているが、抽象化命令セットとしては基本的にKL1Bを基にできる。ただし、read命令、unify命令、

FLENG Compiler and its Abstruct Code
Takeshi SHIMOYAMA, Kentarou SHIMADA,
Hanpei KOIKE, Hidehiko TANAKA
the University of Tokyo

型	31 0
VAR	mmxxxxxxpppppppppppppppppppppp01
LIST	mmxxxxxxpppppppppppppppppppppp10
VECTOR	mmxxxxxxpppppppppppppppppppppp11
UNDEF	mm111111pppppppppppppppppppppp00
SYMBOL	mm111110ssssssssssssssssssssss00
FLOAT	mm0ddddddddddddddddddddddddddd00
INT	mm10ddddddddddddddddddddddddddd00

m : Mark bit, p : Pointer, x : no matter,
s : Symbol ID, d : Data

図 1: タグ構成

	Register間	Register,SR+n間
Passive Unification	wait命令	read命令
Active Unification	get命令	unify命令
Goal Reduction	put命令	write命令

これらの命令に対象として、Variable, Value, Void, Constant, Nil, List, Structureが追加される。また、その他にcreateGoal, enqueue, proceed, executeなどの命令群や、system predicate関連の命令がある。

図 2: 抽象化コード

write命令などは、構造体を指すポインタのポストインクリメントアドレッシングモードでなく、オフセットを明示して、後述に述べるように最適化を行なう。図2に、その基本となる命令群を示す。

4 各種アノテーション

FLENG--は、FLENGにアノテーションを付け加えて拡張している。以下にこれらのアノテーションの意味とこれらをコンパイルする技法について説明する。

4.1 Active Unify Annotation

Active Unify Annotationは、'!'で表される。FLENGにおけるヘッドのマッチングの規則はGHCと同じで呼びだし側を具体化するマッチングはサスペンドされるがこのアノテーションが前置きされた変数はPROLOGと同様にActive Uni-

fication が行なわれる。たとえば、

```
append([H|T],X,![H|Y]):-append(T,X,Y).
append([H|T],X,R):-append(T,X,Y),unify(_,R,[H|Y]).
```

この二つのクローズは等価である。このアノテーションをコンパイルする場合、実際にコミットされるまでActive Unification が行なえないことに注意する必要がある。特に、構造体の内部にこのアノテーションがあった場合、一度そのセルを readVariable 命令で Temporary Register に退避させてコミット後に Active Unification を行なう必要がある。コンパイル例を以下にあげる。

```
a(A, ![B | C], b(![D | E])).

waitStructure 'b'/1, A3
readVariable X1, 0
getList A2
unifyVariable X2, 0
unifyVariable X3, 1
getList X1
unifyVariable X4, 0
unifyVariable X5, 1
proceed
```

4.2 Bind to Non Variable Annotation

Bind to Non Varable Annotation は、'#' で表される。このアノテーションが前置きされた変数は、変数以外のものとだけマッチングする。このアノテーションは、`waitBindVariable`, `waitBindValue`, `readBindVariable`, `readBindValue` 命令にコンパイルされる。

4.3 Single Reference Annotation

Single Reference Annotation は、'' で表される。このアノテーションが前置きされた引数、および単一化されるゴール側変数に対して他からの参照がないことを表す。このアノテーションに関しては、[4] を参照されたい。このアノテーションは、領域の再利用の可能性を示すだけなので、コンパイラはどのように利用するかを判断しなければいけない。判断する要素としては、以下のものが考えられる。

- 再利用できる領域を最大限に利用できるもの
- 既に書かれているデータをそのまま利用できるもの

追加される命令は、`waitListReuse`, `waitStructureReuse`, `getListReuse`, `getStructureReuse`, `getSameStructure`, `putListReuse`, `putStructureReuse`, `putSameStructure` 以上である。いずれも Argument Register と Reuse Area Register を指定する。このアノテーションがついているセルに対応するレジスタは、Reuse Area Register に保持され、read, unify, write 命令 は、このレジスタを使う reuse モードで実行される。既に書かれているデータを再利用するため、これらの命令は、従来の SR のポストインクリメントアドレッシングモードでなく、オフセットをオペランドに持ったアドレッシングモードで行なわれる。SPARC では、13bit のイミディエイト値をオフセットに取るアドレッシングモードが存在し、むしろ従来の方法より高速に実行できる。コンパイル例を示す。

```
append('[H|#T], X, ![H|Y]):-append(T,X,Y).

waitListReuse A1, X1
readBindVariable A1, 1
getListReuse A3, X1
unifyVariable A3, 1
execute append/3
```

5 おわりに

今後の課題として以下のものがあげられる。

- より低レベルで SPARC のアーキテクチャに近い中間言語による最適化、高速化。
- System Predicate の充実
- プログラムの大局的な最適化

参考文献

[1] Warren, D.H.D., "An Abstruct Prolog Instruction Set", Tech. Note 309, SRI International, 1983.

[2] Y.Kimura and T.Chikayama, "An Abstruct KL1 Machine and its Instruction Set", Proc. of SLP'87

[3] Nilsson M. and Tanaka H., "FLENG Prolog - The Language which turns supercomputers into Prolog machines Logic Programing Conference. 1986, pp.209-216.

[4] 小池, 田中, "単一参照アノテーションを用いた論理型言語プログラムの最適化プログラム", 本大会