ライブラリ型Cインタプリタの開発

4D－7

斉藤　隆　増石哲也　中重　亮*　坂元和彦
（株）日立製作所システム開発研究所　*日立ソフトウェアエンジニアリング（株）

1．はじめに

ユーザインタフェース構築ツールなどのビジュアルプログラミング型の開発環境において、Cのような汎用言語のインタプリタに対するニーズが高まっている。それは、(1)ビジュアルプログラミングで実現できない部分は汎用言語で記述したい、(2)その言語で作成したプログラムとビジュアルプログラミングで実現した部分とを連動させて即座に実行させたいというユーザ要求があるからである。

本稿では、さまざまな対話型プログラム開発環境において利用できるように、ライブラリとして開発したCインタプリタについて述べる。

2．Cインタプリタの特徴

従来型のCインタプリタは図１のような構造をしている。Lispインタプリタも同様の構造である。この形態のインタプリタでは、サポートされる組み込み関数の集合は予め決められており[1]、ユーザインタフェースはインタプリタにくくりつけられている。

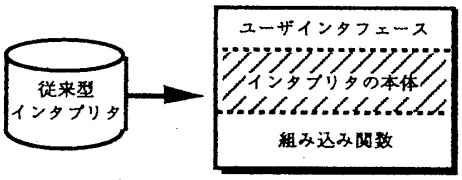

図１．従来型のインタプリタの構造

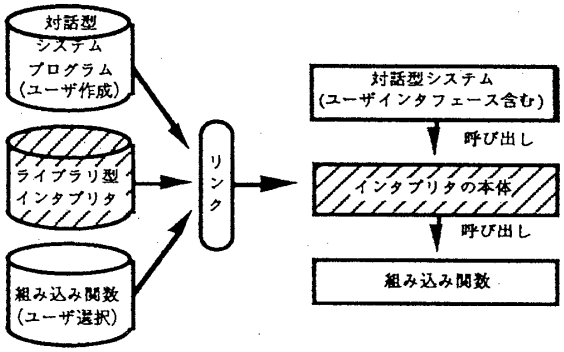

図２　ライブラリ型のインタプリタの構造

しかしCの場合、サポートすべき組み込み関数の集合を予め決めることができない。また、ウィンドウシステムの関数、通信系のライブラリなどすべての関数を組み込むことは不可能である。そこで、図２に示すようなライブラリ型のCインタプリタを開発することにした。

このライブラリ型インタプリタは、次の特徴をもつ。

(1)望みの組み込み関数の集合をサポートできる。

(2)望みのユーザインタフェースを付けることができる。

3．Cインタプリタの内部構造

利用形態を柔軟にするために、図３のようにデバッガとCインタプリタを分離して開発することにした。そして、デバッガとCインタプリタとが、以下に示すように連係して、デバッグ処理を行なうようにした。

(1)デバッガはCインタプリタの関数インタフェースを介して、Cインタプリタにプログラムを実行させる。

(2)Cインタプリタは、文の先頭などの予め決められた位置でプログラムの実行を中断し、制御をデバッガに戻す。

(3)デバッガは、プログラム実行の中断位置において、ブレイクポイント管理などのデバッグ処理を行なう。

(4)デバッガは、デバッグ処理を終えた後に、再度Cインタプリタを呼び出す。

(5)Cインタプリタは、中断位置よりプログラムを継続実行する。

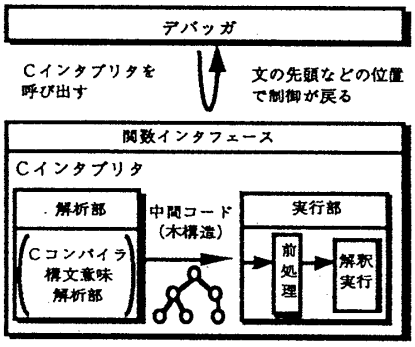

図３．Cインタプリタの内部構造

また、Cインタプリタの解析部には、以下の理由からCコンパイラの構文意味解析部を利用することにした。

(a)Cコンパイラと同様にANSI[2]およびK&R[3]の両方の言語仕様のCプログラムを解析できる。

(b)エラー処理などの構文意味解析処理がCコンパイラと共通化できる。

A C interpreter as a general purpose library

Takashi SAITO[1], Tetsuya MASUISHI[1], Ryo NAKASHIGE[2], Kazuhiko SAKAMOTO[1]
1 Systems Development Laboratory, Hitachi, Ltd.
2 Hitachi Software Engineering Co., Ltd.

(c)Cコンパイラが生成する実行系とプログラム動作を共通化しやすい。

利用したCコンパイラの構文意味解析部は、木構造の中間コードを生成する。そのためCインタプリタの実行部は、「木構造の中間コードをたどりながらプログラムを実行できること」および「プログラムの実行中断／継続実行ができること」の両方が必要となった。

4．Cインタプリタの実行部

4.1 木構造中間コードの解釈実行方式

木構造のデータ構造を扱うためには、通常再帰アルゴリズムが採用される。しかし再帰アルゴリズムは、木のノードをたどっている途中で制御を戻すことが必要な処理には利用できない。木のノードをどこまでたどったかという情報（以降、ノードトレース情報と呼ぶ）は、システムスタック中に蓄えられるために、呼び出し元に制御を戻したときに失われるからである。

木のノードをたどりながら、プログラムの実行中断／継続実行ができるように、以下の方式を採用した。

(1)繰返しアルゴリズムを用いて、木のノードをたどって中間コードを解釈実行する。

(2)ノードトレース情報をCインタプリタのスタックで管理する。

これは、「最適化コンパイラには木構造の中間コードを生成するものが多いこと」、「Cインタプリタをデバッガから独立に構成することが利用形態を柔軟にすることにつながること」から、一般的に重要な技術であると考える。

4.2 スタックで管理する情報

Cインタプリタの実行部では、ノードトレース情報として、「ノードの位置」と「ノードのたどり方（行きがけ／通りがけ／帰りがけの区別）」との組み合わせを用いることにした。Cインタプリタでは、評価対象であるノード種類とそのたどり方との組み合わせと対応付けてノード評価処理を定義している。そのため、Cインタプリタの実行部は、スタックトップにあるノードの位置とたどり方から、次に行なうノード評価処理と評価対象ノードを特定することができる。

4.3 中断位置からの継続実行処理

Cインタプリタ実行部は、以下の処理を行なうことにより、中断位置からプログラムを継続して実行できる。

(a)スタックからノードの位置とたどり方をポップする。

(b)ノード位置で示されるノードの種類を調べる。

(c)ノードの種類とノードのたどり方より、ノード評価処理を特定する。

(d)(c)で特定できたノード評価処理をノード位置で示されるノードに対して行なう。

たとえば、解析部は式"a*(b+c)"に対して図4のように表現される中間コードを生成する。実行部は、その木のノードを(1)から(9)の順番にたどって、ノード評価を行なう。図4の(4)の位置で、スタック状態は図5の左のようになっている。ノード評価処理は、次に行なうノード評価処理および評価対象ノードを特定するために、スタックから"node(＋,行き)"をポップする。そして"＋"ノードの行きがけの処理として、以下の処理を行なう。「"＋"ノードの第一子である"b"ノードを行きがけで評価する」、「"＋"ノードを通りがけで評価する」ことを以降この順番で行なうために、スタックに"node(＋,通り)"と"node(b,行き)"をプッシュする。

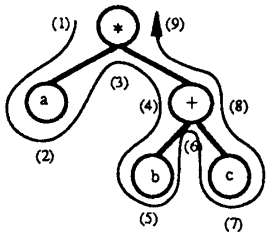

図4．木のノードのたどり方

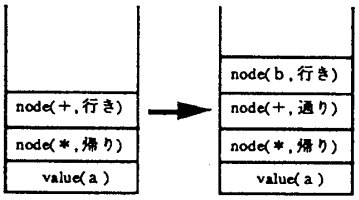

図5．図4の(4)の処理前後のスタック状態の変化

5．おわりに

コンパイラと言語仕様が共通でかつ利用形態が柔軟なCインタプリタを開発するために、「プログラム実行を中断し制御をデバッガに戻すことができること」および「木構造の中間コードを解釈実行できること」の両方を可能とする以下の方式を採用した。

(1)繰返しアルゴリズムで木構造の中間コードをたどる。

(2)スタックで木のノードトレース情報を管理する。

最適化コンパイラには、構文意味解析の結果を木構造で記憶するものが多い。このため、コンパイラと構文意味解析部を共用するタイプのインタプリタを、デバッガと独立させた形で構成するためには、本方式が有効であると考える。

参考文献

[1] J.W.Davidson, "Cint: A RISC Interpreter for the C Programming Language", SIGPLAN NOTICES, Vol.22, No.7, pp.189-198, Jul., 1987

[2] B.W.Kernighan, D.M.Ritchie, 石田晴久訳, "プログラミング言語C第2版", 共立出版, 1992

[3] B.W.Kernighan, D.M.Ritchie, 石田晴久訳, "プログラミング言語C", 共立出版, 1981