Ｃ言語の解析木インタプリタのアーキテクチャの設計

9Ｌ－4

闞　暁薇　、板野肯三
筑波大学

１．はじめに

対話性の高いプログラミングシステムを実現する一手法として、構造エディタと解析木インタプリタをベースにした統合的なプログラミング環境の設計と実現に関する研究を行ってきた[1,2]。この研究の一貫として、ハードウェア化によって解析木インタプリタの実行速度を向上させることを目指して本研究を開始した。プロトタイプとしてのＰＬ／０用のハードウェア解析木インタプリタＰＡＴＩＥ_０[3]を設計する過程で、プログラムの内部表現が実行速度やハードウェアの性能に最も大きな影響を与えることが判明した。そこで、Ｃ言語用のハードウェア解析木インタプリタＰＡＴＩＥ_Ｃの設計においては、ハードウェアの設計を最適化すると共に、解析木を内包している”実行木”と呼ぶ新しいプログラムの内部表現を考案した[4]。

２．実行木の概念

実行木は、プログラムの内部表現として、解析木を内包しており、実行の前に意味解析によって実行時に必要な情報を数値や木構造などの様々な形式で表現し（解析木と対称的に”意味木”と呼ぶ）、解析木に埋め込んで解析木と一体化するデータ構造である（実行木＝解析木＋意味木）。図１には実行木の例を示す。図中で、実線は解析木部分を点線は意味木部分を示している（厳密に言えば、実行木は木構造ではなく、有向グラフである）。

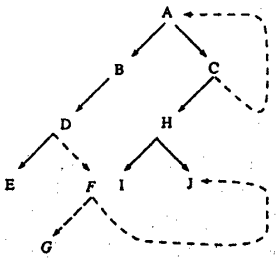

図１．　実行木の例

解析木はインクリメンタルパーサによって作成されるが、意味木は意味解析器によって作成される。解析木インタプリタは、両方を区別せずに同じメカニズムで解釈実行する。

２．１　意味解析の方針

意味解析は構文規則ごとに定義されている意味解析のアルゴリズムに従って行う。主に解析する意味としては、データ構造に関するものと制御構造に関するものがある。

（１）データ構造に関する意味

名前の型、大きさ、オフセットなどの情報を名前の宣言に対応するノードに埋め込み、名前の参照ノードから宣言ノードへリンクを作成する（リンクは意味木の一部）。このような意味解析によって、実行時の記号表の参照が不必要となり、ブレイクポイントを入れるなどのデバック機能にも対応しやすくなる。

（２）制御構造に関する意味

制御構造の意味的一貫性を検査し、意味木を生成したりすることによって、実行木の上で効率のよい走査のためのバイパスを作り出す。この制御構造の中で最も問題となるのはgotoなどのジャンプである。

実行木を走査しながら実行するプロセスは、再帰的実行であるため、実行木のあるノードから、別のノードに直接ジャンプすると実行の内部状態の一貫性が失われてしまう。そこで、意味解析の段階で、走査パスを検査し、上向きのジャンプに対しては、ジャンプ先ノードまでの間で捨てる状態の数をポップカウントとして計算し、ジャンプの始点ノードに格納する。下向きの走査に対して、ジャンプ先ノードまで木を順次辿ったときに得られる状態と同じ状態が得られるようなバイパスを作り出し、このバイパスは意味木としてジャンプの始点ノードに付け加える。

２．２　実行木の例

breakやcontinueの場合は上向きのみのジャンプなので、ポップカウントを利用すれば簡単に高速ジャンプできるが、図２に示すようなgoto の場合は上向きのジャンプと下向きのジャンプ両方あるので、ポップカウントとバイパスを組み合わせて高速ジャンプを実現させる必要がある。この例では、ポップカウントとして１を、gotoノードに格納し、さらに、callを行うノードdowntrvを２個生成してバイパスを作り出し、このバイパス（点線）を意味木としてgotoの下に付加している。

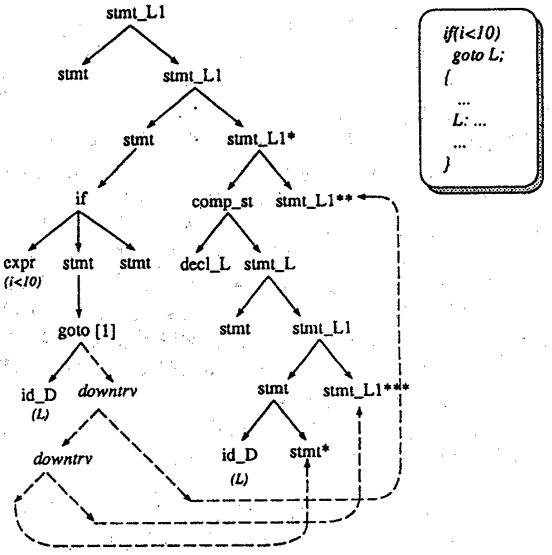

図２．gotoの実行木の例

Design of the Architecture of a Parse Tree Interpreter for C Language
Xiaowei Kan, Itano Kozo
University of Tsukuba

３．インタプリタの設計

ＡＮＳＩ＿Ｃを対象言語としてハードウェアの解析木インタプリタＰＡＴＩＥ＿Ｃ（Parse Tree Interpreter for C）を設計した。

３．１　インタプリタのアルゴリズム設計

解析木インタプリタのアルゴリズムは実行木のノードの種類ごとに定義する。実行時に解析木インタプリタは実行木ノードの種類を確認し、その種類に対応するアルゴリズムに従ってそのノードを解釈実行する。一般に、子ノードの実行は親ノードの実行中に呼び出されるので、インタプリタの実行は、再帰的な実行になる。従って、実行の状態をスタックに保存していく必要がある。各ノードの実行中での子ノードの呼び出しと意味の実行の関係は

S1; call1; S2; call2; S3;call3;S4;....

のような形で示される。ここで、calliは子ノードの実行の呼び出しで、Siは親ノード中での意味の実行である。

（１）データ構造に関する実行アルゴリズムの設計

インタプリタはハードウェアで実現するので、実行中の内部状態の保存は最終的にはレジスタで行う必要がある。

メモリ領域のモデルとしては、フレームスタックを採用した。フレームスタックの一番底の部分にはグローバル変数の領域をとり、関数呼び出しのたびに新しいフレームを生成して割り当てる。各フレームの一番下はパラメタの領域で、その上に局所的変数の領域がとられる。定数はフレームに入れずに意味解析の時にノードに直接書き込むことにした。関数からリータンを行う時に現在のフレームを削去し、フレームの下に格納してある隣接フレームのベースポインタを利用して、そのフレームを回復する。

名前の参照は、実行時の記号表の参照を行わずに、意味解析で作られている参照ノードから宣言ノードへのリンクを辿ることによって宣言ノードに参照を行う。アドレッシングは、宣言ノードに保持しているフレーム中のオフセットにベースポインタを足すことによって行う。型の検査は意味解析で宣言ノードに集まられた型に関する情報を用いて実行時に動的に検査を行う。

（２）制御構造に関する実行アルゴリズムの設計

ループに関する制御は、ループ本体の終わりに当たるノードからループの先頭に当たるノードへのリンクを辿ることによって実現する。

関数呼び出しに関する制御は、呼び出し側の関数名に当たるノードから関数の定義ノードへのリンクを辿るだけでよいので、特別な処理は必要でない。関数からのリータンは関数呼び出しの次に実行するべきノードをスタックからポップすることによって制御を呼び出し側に戻す。

条件分岐に関する制御は、実行すべき子ノードの選択によって自然に実現される。但し、case文のような構造化されていない条件分岐に関しては、意味解析の時意味木を用いて構造化させる必要がある。実行時に意味木を解釈実行することって条件分岐を実現する。

ジャンプに関する制御は、前述のポップカウントとバイパスをそのまま実行するだけでよい。

（３）ハードウェア化に関するアルゴリズム設計上の制限

パイプライン型のアーキテクチャを採用するために、実行アルゴリズムの設計に”子ノードは一回しか実行せず、子ノードの実行呼び出し順番は親ノードの実行状態に依存せず、親ノードの種類のみで決定できること”という制限を加えた。

一方、ハードウェアの実行効率を向上させるために、実行木を二進木に制限し、子ノードの数を２個以下にした。

３．２　パイプライン・アーキテクチャ

実行木の各ノードの実行は、実行木の走査、ノードの再帰的な実行の制御、データ演算・転送という３つの処理に分けられる。ＰＡＴＩＥ＿Ｃのアーキテクチャは、これらのそれぞれの処理に対応したトラバースコントロール・ユニット（Traverse Control Unit、TCU）、リカーシブコントロール・ユニット（Recursive Control Unit、RCU）、データユニット（Data Unit、DU）の３つの独立したユニットをパイプライン型に接合して構成する（図３）。TCUは、メモリに格納されている実行木のノードを一つづフェッチしてくる。フェッチしてきたノードの種類によって、実行木の走査順番を決定する。RCUは、TCUからノードのcodeを受け取って、そのノードの再帰的な実行を制御するマイクロ命令列を実行する。その実行に於て、実際にデータ演算や転送などの必要があれば、次のユニットDUへその実行を指示するコードを送る。DUは、RCUから渡されたコードをデコードし、データの演算や転送を行う。また、条件分岐の結果をTCUに送り返す。

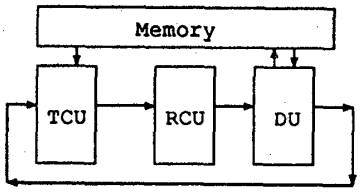

図３．　パイプラインのアーキテクチャ

４．おわりに

実行木の概念は、特にＣ言語の構文やセマンティクスに依存しているわけではないので一般性があると考えている。また、今回の設計では言語依存性の高い処理はなくべく意味解析の段階に持込み、実行木に埋め込んだので、解析木インタプリタのアーキテクチャからも言語依存性はかなり減少したと考えている。

参考文献：

[1] 佐藤豊、板野肯三：COSMOSにおける構造エディタおよびソースコード・インタプリタの実現法、情報処理学会第31回全国大会,1f-7,1985,pp.447-448.

[2] 佐藤豊、板野肯三：動的複合実行方式一直接実行系と翻訳実行系を統合した対話型実行方式、コンピュ ータソフトウェア、2巻、4号、1985、pp.19-29.

[3] 闕　暁薇、板野肯三：解析木インタプリタPATIEOのアーキテクチャ、情報処理学会論文誌、32巻、9号、1991、pp.1113-1121.

[4] 闕　暁薇、板野肯三：ハードウェアインタプリタ向きの解析木の形式と実行アルゴリズムの設計、情報処理学会研究報告、92-PRG-6、1992、pp.61-70.