1M－2

プログラミング言語において動的データと宣言的データとを統一的に扱う方式

阿部 雅彦　高田 義広　鳥居 宏次

奈良先端科学技術大学院大学 情報科学研究科

1 はじめに

C, Pascal などの手続き型のプログラミング言語においては, **動的データ**と呼ばれる変数が存在する[1]. それは, 手続き呼び出し (あるいは, 命令の実行, 演算) により生成され, 別の手続き呼び出しにより消去されるデータである. 例えば, Pascal において, 手続き New と Dispose とにより生成・消去されるデータである. 動的データ以外の大多数のデータは, 特定の手続き, あるいは, プログラムの開始時に生成され, 終了時に暗黙的に消去される. 例えば, 手続きに局所的な変数, 手続きに値渡しされる引数, 大域変数などである. プログラム中で宣言されることより, それらを**宣言的データ**と呼ぶことにする.

複雑なデータ構造を処理する場合には, 動的データを扱うことが必須である. ところが, 従来の言語においては, 宣言的データと動的データとの扱いが大きく異なっており, しかも, 動的データの扱いが難しい. 例えば, 動的データは, 名前で直接的に参照できないし, 特定の手続きを呼び出さない限り消去されない.

そこで, 本発表では, 手続き型言語における動的データの問題を詳述し, その問題が解決するように言語を改良する方式を提案する. 改良後の言語では, 全てのデータが, 統一的に, 手続き呼び出しにより生成され, 名前により直接的に参照され, 暗黙的に消去されることになる.

2 従来の言語の問題

図1の例を用いて問題を説明する. これは, スタックに項目を追加・削除する手続きであり, Pascal で書かれている. スタックを構成するノードは, 手続き createNode と disposeNode とにより生成・消去される動的データとして扱われている.

2.1 動的データの参照の問題. 手続き push において

```
type node = record Next : ↑node; ... end;
type stack = record Top : ↑node; ... end;

procedure push (var Stack : stack; A : integer);
var p : ↑node;
begin  p := createNode(A);
    setNext(p ↑, getTop(Stack)); setTop(Stack, p);
end;

procedure pop (var Stack : stack);
var p : ↑node;
begin
    p := getTop(Stack); setTop(Stack, getNext(p ↑));
    disposeNode(p);
end;
```

図 1: 従来の言語のコード

てのように, 一般に, 動的データ (ノード) は, 名前で直接的に参照されず, ポインタ変数 (p) により間接的に参照される. 宣言的データが簡単に名前で参照できることに比べると, 複雑なコードを書くことがプログラマに強いられている. その原因は, 手続きからの結果の戻され方にあると考えられる. 従来の言語における手続きの結果は, 手続き終了の直後にだけ存在し, すぐに暗黙的に消去される. そこで, プログラマは, 通常, 生成したデータ自身を戻す代わりに, そのデータへのポインタを戻すようにコードを書く. しかも, そのポインタもすぐに消去されるので, ポインタ変数を用意しておき, そこにポインタを代入するようなコードを書く. もし, 手続きの結果がすぐに消去されず, しかも, 名前を付けて直接的に参照できれば, そのような複雑なコードを書く必要がない.

2.2 動的データの消去の問題. 手続き pop 中の disposeNode のように, 動的データの消去には, 手続き呼び出しが必要である. 但し, 消去の手続きを忘れていないことは, 容易に確認できない. また, もし忘れた場合には, プログラムの実行中に, 不要な動的データでメモリが溢れると言う原因を見つけにくいエラーを起こすことがある. 宣言的データに消去の手続きを書く必要がないことに比べると, やはり, プ

Unification of the Dynamic and Declarative Data in a Programming Language

Masahiko Abe, Yoshihiro Takada, and Koji Torii
Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama-cho, Ikoma-shi, Nara 630-01, Japan

ログラマに多大な負担がかかっている.

処理系によっては, ゴミ集め (garbage collection) を採り入れて, 実行中に不要な動的データを自動的に探して消している場合もある [1]. その場合, 消去の手続きを忘れても (あるいは, 始めから書かなくても) 安全であるが, 代わりに深刻な問題がある. ゴミ集めは時間のかかる処理であり, その度にプログラムの実行が中断することである. そのような重い処理を伴わずに, 消去の手続きの必要をなくす工夫が望まれる.

3 言語の改良

提案する改良は次の手順に従う.

3.1 手続きの結果に命名する構文の導入. 例えば, Pascal では,

$NewNode \leftarrow$ createNode(A);

と書くと, 手続き createNode から戻されるデータを, *NewNode* と言う名前で直接的に参照できることにする. この拡張によって, 手続きにより生成されるデータの参照が簡単になる.

3.2 宣言によるデータ生成の廃止. つまり, 局所変数の宣言や, 引数の値渡しをなくす. この縮退によって, コードのスタイルは若干変わるが, 記述できる処理のクラスに変わりはない. 従来の言語において宣言により生成されるデータは, 一般に, まず代入により式の評価値が上書きされ, その後に参照される. 改良後の言語においてそれに対応するデータは, 式の評価値に直接的に名前がつけられたものになる. ここまでの変更により, 全てのデータが, 統一的に, 手続きにより生成され, 名前により参照されることになる.

3.3 データの所有権の導入. 所有権と言う新しい概念を導入し, それに合わせて手続きの宣言の部分の構文を拡張する. 詳しくは次章に述べるが, この拡張によって, 各データを消去してよい時点が, 計算機に判断できるようになる. 従って, データ消去の手続きが全く必要なくなる. また, その時点は, コンパイラが判断できるので, 実行時の効率の心配もなく, ゴミ集めの必要もない.

4 データの所有権

直観的に言うと, 各データを消去させない権利を意味する. そのデータを処理している手続き (厳密には, 手続きの実行環境), または, そのデータを部品とする複合型データに属する. データの生成と同時に発生し, 移動したり消滅したりするが, 複製はされない. 所有権の消滅は, そのデータの存続が保証されなくなったことを意味する. あるデータの所有権がある手続きに属することは, そのデータの処分がその手続きに委ねられていることを意味する.

所有権の移動は, 手続きに引数が渡される際や, 手続きの結果が戻される際に起こる. 複合型データへの移動は, 複合型データの部品を設定する手続きの呼び出しの際に起こる. 手続き A の中の手続き B の呼び出しでデータ D が生成されるとしよう. D の所有権は, まず B に発生し, 次に A に移動する. その後, D の所有権は, もし他へ移動しなかった場合, A の終了と同時に消滅する. もし, A の終了の前に手続き C へ D が引数として渡され, そして, C が例えば

procedure setNext (*Node* : **trans** node);

のように宣言されている場合, D の所有権は C に移動する. 予約語 "trans" は, node 型の引数が渡される時に所有権も移動することを表している. "trans" がなければ, 所有権は移動せず, 単に D の参照渡しが行なわれる. また, もし, D の所有権が A の終了まで残っていて, A の結果として A の親手続きに D が戻され, しかも, A が 例えば

function getNext (*Node* : node) : **trans** node;

のように宣言されている場合, D の 所有権は A の親手続きに移動する.

所有権の導入に伴う構文の変更は, 上述のように, 主に手続きの宣言の拡張 ("trans") だけである. ところが, 前述のように, それだけの拡張により, データ消去の時点の自動的な判断が可能になっている. それは, 所有権が消滅する時点であり, コードの静的解析により容易に特定できる.

5 おわりに

動的データの問題が解決するように, 手続き型言語を改良する方式を提案した. この方式は, 特定の言語に依存しておらず, C, C++, Pascal, Ada のような手続き型の言語に広く適用できる. 現在, 我々は, 提案した方式を C 言語に適用して得られた言語のコンパイラを実装している.

参考文献

[1] Aho, A. V., Ullman, J. D. 著, 土居範久 訳: コンパイラ, 培風館 (1986).