C-032

耐故障関数型言語処理系の実現
Implementation of a Dependable Interpreter for a Functional Language

瀧閨 洋介 †　太田 康章 †　金子敬一 †
Yousuke Takigiku　Yasuaki Oota　Keiichi Kaneko

1. はじめに

今日，コンピュータの偏在化に伴い，耐故障性と信頼性の保証はますます重要となってきている．しかしながら，その対策には依然として特別に設計されたハードウェアを用いることが多く，必然的に高価である．本研究ではハードウェアのうちメモリ故障に焦点を当て，ソフトウェア的に耐故障性を実現する関数型言語インタプリタを提案する．メモリ故障を許すことができれば，汎用メモリが使用可能となり，価格的なコスト削減を達成することができる．

2. 耐故障インタプリタの設計

2.1 故障とメモリモデル

本研究では故障を「プログラム実行中に生じるメモリ内容の誤謬」と想定する．処理系が使用するメモリ領域のうち，本体常駐部分や，レジスタ，実行時スタック領域の故障は考慮しない．また入力されたプログラム（式），実行前に静的に定義されるオブジェクト（大域環境）については，メモリ全体に占める割合が小さいため，再構築のコストを考慮し，故障を想定しない．これらはハードウェア的に故障の回復が可能な領域(Fault-Free Area，以下 FF と略)に配置することにし，FF に存在する情報から再計算によって回復が可能なものを汎用メモリ領域(Fault-Sensitive Area，以下 FS と略)に配置するものとする．

2.2 一時バッファ領域

FS 領域にあるオブジェクトの内容には誤りの可能性がある．よって FS 領域を参照する際には誤りの検出を行い，内容の正当性を確かめる必要がある．この時に，FS 領域に対してそのまま誤り検出を行ってしまうと，検出と内容参照の間にも誤りが発生する可能性がある．よって，FS 領域のメモリ内容参照の際は，あらかじめ FF 領域に用意された専用の一時バッファ領域(Temporary-Buffer Area，以下 TB と略)上に内容をコピーした上で，間接的に参照する．TB 領域に関しては 1 度に参照されるオブジェクトの最大サイズのみ領域確保すればよい．

2.3 閉包

関数型言語において評価を遅延するために，閉包と呼ばれるデータ構造が使われる．閉包は評価を再開するための情報である「評価式へのポインタ」と「評価時の環境」を持ち，評価後は結果の値で更新される（図 1 参照）．更新は，いったん評価した値に対して不必要な再計算を防ぐという目的による．しかし，この構造のままでは評価結果を再構築するための情報が失われてしまう．そこで閉包構造に評価結果のために 1 フィールド追加し，評価のために必要な情報を常に保持しておくことにする（図 2 参照）．この変更により，タグを判別することで不必要な再評価を避け，かつ評価結果の誤りが検出された場合は再評価によって値を求めなおすことが可能になる．

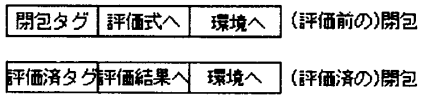

図 1　閉包構造（変更前）

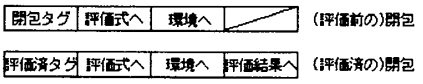

図 2　閉包構造（変更後）

2.4 環境

環境は，変数と値を結び付けておくためのデータ構造であり，一般に，変数と閉包からなる対のリスト構造として表現される．本研究では，図 3 のように実現する．

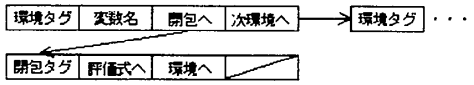

図 3　環境の内部表現

3. 再計算の可能性と手段

3.1 動的生成オブジェクト

関数型言語の処理系では，プログラムの実行は，式を評価することに等しい．本研究のインタプリタがプログラム実行中に動的に生成するオブジェクトは以下の 2 種に分類される．

(1)評価結果オブジェクト

処理系は式の評価時にメモリ上のヒープ領域に評価結果のオブジェクト（整数値，コンスセルまたは空リスト，関数値）を生成する．

(2)一時環境オブジェクト

関数の適用時には一時環境オブジェクトを生成し，引数部分を関数本体部の仮引数に束縛し，その環境下で手続き本体を評価する．ここで生成されるオブジェクトは，組込み関数など，評価前に静的に定義される大域環境オブジェクトとは区別される．

† 東京農工大学工学部

3.2 評価結果オブジェクトの再構築

評価結果オブジェクトが参照されるタイミングは，その生成直後，または再参照時の 2 つに大別できる．このうち生成直後の参照については，FS 領域への生成時に同内容を TB 領域にも生成し，参照は TB 領域のオブジェクトに対して行えばよい．また，再参照時の場合は必ず再構築のための情報を保持している閉包を通してアクセスされるため，誤りが検出された場合は再構築が可能である．これらのことから評価結果オブジェクトは FS 領域に配置することができる．

3.3 一時環境オブジェクトの再構築

本処理系では遅延評価を採用しており，関数適用のたびに一時環境オブジェクトが生成される．このため動的生成オブジェクト全体に占める割合は平均約 70%にも及ぶ．しかしながら，図 3 の構造のままでは再構築に必要な情報が保持されていないため，FS 領域に配置することができない．一時環境オブジェクトを生成する際に必要な情報は，関数適用式(FF 領域上にある)へのポインタとその評価時の環境へのポインタであるから，これら再構築のために必要な 2 つの情報を保持している補環境オブジェクトを新たに定義し，FF 領域に配置する．そして一時環境にはこの構造を通してアクセスするように変更する（図 4 参照）．この変更によって一時環境オブジェクトを FS 領域に配置することが可能になり，従来 FF 領域に要求していたフィールド数を 8 から 3 へ減らすことができる．

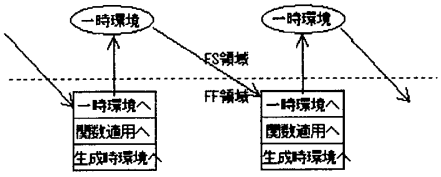

図 4　補環境の導入と環境アクセス法の変更

4. ゴミ集め(Garbage-Collection)

本研究ではメモリの故障によるポインタ情報の誤謬を考慮し，オブジェクトの移動を伴わないマーク方式によるゴミ集めを行う．また，FS 領域のオブジェクト本体にマーク情報を与えると内容に誤謬が発生する可能性があるため，FF 領域上にヒープの空き領域情報を保持する領域を設け，メモリ管理はマーキングと合わせてそこで一括して行う．

5. 評価実験

前章までの設計方針に基づき，プログラミング言語 Delphi を用いて処理系を実現した．この処理系に対し故障率を変化させ，フィボナッチ数列第 n 項を求めるプログラムを 1000 回実行し，実行時間と要求フィールド数の平均を測定した．また，耐故障性を備えていない処理系との比較を行った．実行環境には，Pentium III 933MHz を CPU とし，384MB の主記憶を備え，Microsoft Windows 2000 を操作系とする目的機械を利用した．図 5 および図 6 に平均実行時間比と平均要求フィールド数比を示す．これらの図より，耐故障性を付加したことによる時間的オーバヘッドは最大約 1.5 倍であることが分かる．

6. 結論

本研究では，プログラム実行時に動的に生成されるオブジェクトについて再構築するための手法を提案した．処理系を実現して実験した結果，捕環境以外のオブジェクトについて，故障を許容することに成功した．また提案手法を組込んだことにより生じる時間的オーバヘッドは最大約 1.5 倍程度であることを確認した．今後は捕環境オブジェクトも含め，動的生成オブジェクトをすべて FS 領域に配置する手法の開発と，その際に生じると考えられるオーバヘッドの軽減方法の検討が課題となる．

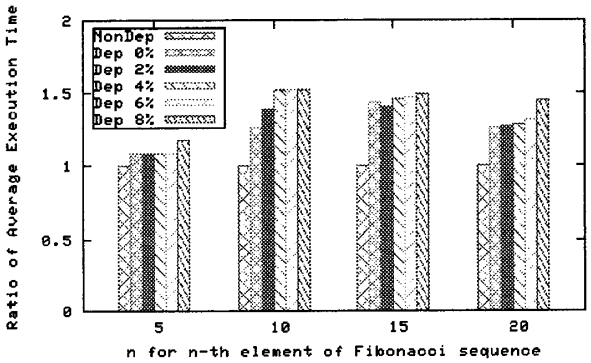

図 5　平均実行時間比

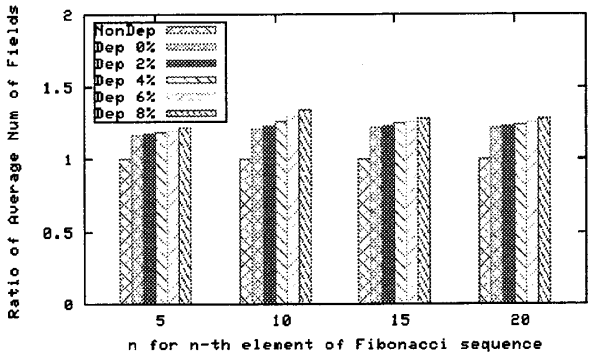

図 6　平均要求フィールド数比

参考文献

[1]Simon L. Peyton Jones: The Implementation of Functional Programming Languages, Prentice/Hall International, 1987.

[2]Richard Bird and Philip Wadler: Introduction to Functional Programming, Prentice Hall, 1988.

[3]Jacques Cohen: Garbage Collection of Linked Data Structures, Computing Surveys, Vol. 13, No. 3, pp. 341–367, Sept. 1981.

[4]ジェラルド・ジェイ・サスマン，ハロルド・エイブルソン，ジェリー・サスマン共著，和田英一訳：計算機プログラムの構造と解釈　第二版，ピアソン，2000.