6U-6

並列オブジェクト指向言語DinnerBell
—並列計算機上のDebuggerの考察—

渡部眞幸，河野真治，田中英彦

東京大学　工学部

1．はじめに

計算機の処理能力の飛躍的向上を求めて、あるいは並列処理を必要とする様々な問題解決のために、並列計算機は今日その重要性をますます高めている。しかし、こうした並列環境でのプログラムのデバッグはそう単純ではない。その理由は主に、プログラムが非決定的に振舞うために、再現性に乏しい点にある。一方、我々が研究を進めている並列オブジェクト指向言語DinnerBellは、単一代入則を守っている限り基本的に副作用をもたない言語であるため、プログラムの非決定性は実行結果に影響を与えない（現在副作用をもつオブジェクトの実装も進行中であるが、とりあえずここでは考えないものとする）。この条件下においてDinnerBellのデバッグは比較的容易であるといえる。

本稿ではDinnerBellのプログラミングを支援するデバッガシステムの構想を述べる。特に手続きの実行順序という点に着目し、結果が非決定的にならないようにプログラムに要求される逐次性が、プログラム上で保証されているかどうかを確かめる機能を備えることで、並列環境におけるデバッガとしての特色を出している。本システムの一部は、既にObjectPeeperとして発表され、試作されているものである。［1］

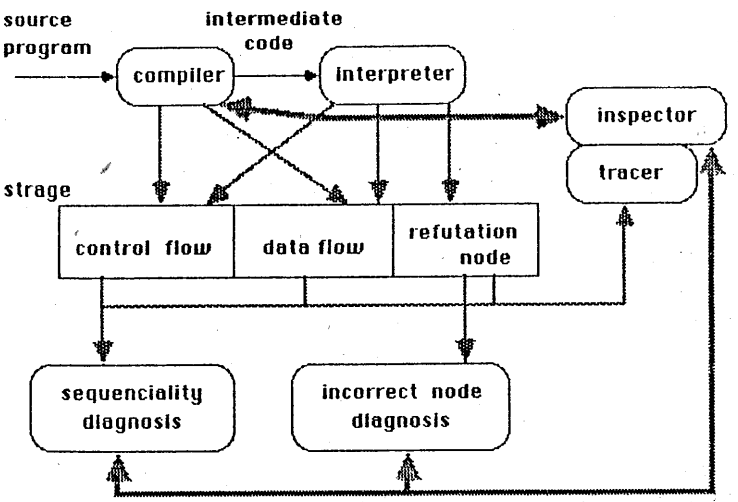

図1　デバッガシステムの構成

2．デバッグに必要な蓄積情報

図1にデバッガシステム全体の構成を示す。

DinnerBellの実行は、Actor モデル［2］と同様にオブジェクト同士が互いにメッセージをやりとりしながら処理を進めるという図式をとる。DinnerBellの実行系は、このメッセージ通信をコンテクストという単位で考える。コンテクストは、図2に示すようにメッセージの送信先であるオブジェクト（Destination）、キーワード（Key）と引数オブジェクト（Argument）から成るメッセージ（Message）、および答を返す返信先オブジェクトを示すポインタ（Reply）から構成される。一般に1つのコンテクストの実行はいくつかのコンテクストを生成する。両者の間には実行順序に関して明らかに1つの逐次性が存在する。この依存関係をここではコントロールフローと称する。またあるコンテクストのDestination が別のコンテクストのReply を参照している場合、後者の実行が終わってReply が確定しないうちに前者の実行を開始することはできない。このように、コンテクスト間の変数による参照関係はコンテクストの実行順序に直接関与するものであり、この依存関係をデータフローと称する。プログラムの逐次性が保証されていることを確めるには、この2種類の情報を蓄積する必要がある。さらにプログラムの実行履歴として、各ノード（コンテクスト）をその変数が全て確定した形で蓄積しておく。これはShapiro のAlgorithmic Debugging ［3］に基づく誤りノードの検出にも用いる。

3．システム各部

3．1　インスペクタ

インスペクタの主な働きは、プログラマと計算機とのコミュニケーションを支援することにある。具体的には、ソースプログラムの呈示とエディット環境の提供、トレース情報の表示とコマンド入力、診断プログラムとプログラマとの情報交換やquery支援等々が挙げられる。現在、オブジェク

Concurrent Object Oriented Language 'DinnerBell'　Some Notes on a Debugger for a Parallel Computer

Masaki WATANABE, Shinji KONO and Hidehiko TANAKA

The University of TOKYO

Destination	Message	Reply
Generator	QS: 5	sender

図2　コンテクスト

ト内部情報の表示機能と簡単なメッセージ問合せ機能を備えたObjectPeeperが動いている。

3．2　トレーサ

並列プログラムはその非決定性のためプログラム全体のトレースはほとんど意味がない。ここでは局所的なトレースを計算木のノード展開という形で行う。計算木はコンテクストをノードとし、その生成関係（コントロールフロー）を枝として定められる（図3）。ノードを展開するとは、あるノードをそれが生成するいくつかの子ノードで置き換えることであり、論理型言語のリダクションと同じである。また不必要なノードはカットして、これ以降展開される子ノードを表示の対象からはずすこともできる。このような展開・カットをルートに有限回施して得られるノード（コンテクスト）のサブセットがトレーサによって表示される（図4）。プログラマはコントロールフローによるプログラムの逐次性を明確に把握することができ、また変数名を対照することによってデータフローに起因する逐次性もある程度把握できる。このトレーサに関してインスペクタは、表示されたトレース情報とソースプログラムの対応や、展開・カットのコマンド入力を視覚的にサポートする。また、トレースの各段階におけるオブジェクト内部情報の表示機能も備える。

3．3　誤りノード診断

誤りノードの診断は、Shapiro のAlgorithmic Debuggingの手法による。refutation node として蓄積されるノードは、対応するコンテクストの変数オブジェクトによる参照部分に最終的な確定値を代入したもので、ある手続きと、与えられた入力、及び得られた出力の三つ組を示している。Devide-and-Queryの方法により、誤りノードの探索を行う。

3．4　逐次性診断

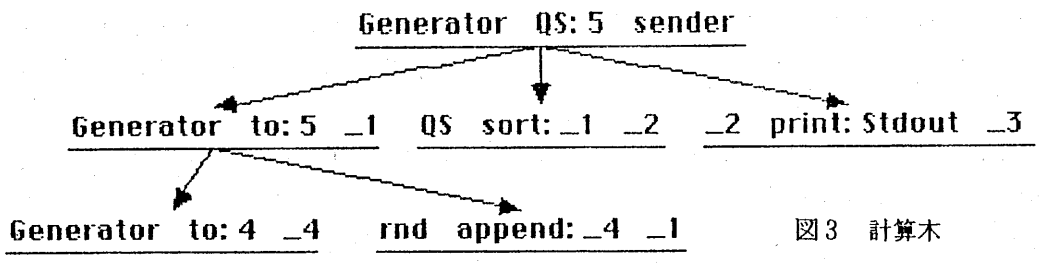

図3　計算木

プログラマが要求する逐次性がプログラム上で保証されているかどうかを確かめるためにコントロールフローとデータフローの情報を用いる。すなわち、ある2コンテクスト間の実行順序は、(1)一方により他方が生成される場合、生成される方が後、(2)一方が他方を参照する場合、参照する方が後、という2つの条件だけで定められ、これ以外の場合は不定とする。これらの情報を用いて、プログラムの振る舞いから逐次性の異常な箇所をAlgorithmic に同定する方法を検討中である。

4．現状と今後

デバッガシステムの構成の概略を示したが、これが決定的というわけではなく、研究を進めるにしたがって適宜変容することが予想される。現在は蓄積情報のフォーマットについて検討中であり、コンパイラとインタプリタに手を加えて必要な情報を抽出することを試みている。将来は、逐次性診断の部分について、Algorithmic にバグを見つけ出す方法をまず理論的に検討し、実際に動いてくれるツールを何か作ろうと思っている。

参考文献

[1]　阿部他，情報処理学会第32回全国大会，6F-2，(1986)
[2]　C.Hewitt, A Universal Modular ACTOR Formalism for Artificial Intelligence,　(1973)
[3]　E.Shapiro, Algorythmic Program Debugging, MIT Press,　(1982)

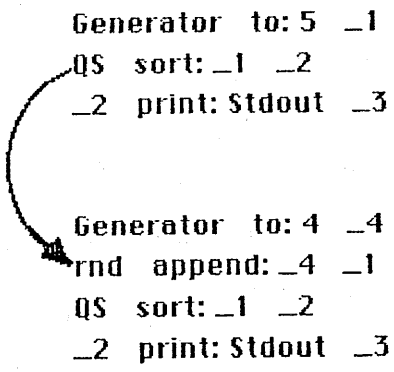

図4　ノードの展開