コンテキスト指向例外処理方式における例外条件検出 †

山下 俊之 [1)] 中西恒夫 [2,3,4)] 福田晃 [2,4)]

[1)] 九州大学工学部電気情報工学科 [2)] 九州大学大学院システム情報科学研究院
[3)] 九州大学システム LSI 研究センター [4)] 九州大学情報基盤センター

1 はじめに

C++や Java の try-catch 構文を用いる例外処理方式では，例外条件の検出はプログラマが明示的に行い，適切な例外を throw しなければならない．この方式では，プログラマによる例外条件検出コード記述が煩雑になり，例外条件検出に見落としが発生する可能性がある．例外条件検出の負荷を軽減するべく，我々はコンテキスト指向例外処理方式 [2] を提案する．

本稿では，コンテキスト指向例外処理方式において必要となる，例外条件検出の際のオーバーヘッドを軽減する方法について述べる．

2 コンテキスト指向例外処理方式

本章では，コンテキスト指向例外処理方式の概要を述べる．コンテキスト指向例外処理方式では，常に満たしていなければならない条件式とその条件を逸脱したときの例外処理を，プログラマがプログラムとは別に記述する．そして，コンパイル時に，記述された条件式に基づいてコンパイラが適切な例外条件判定コードをプログラム中に埋め込むことによって，より安全なプログラムの構築を可能とする．

条件式は，プログラム中のクラスまたは関数に持たせ，その適用範囲は条件を持ったクラスまたは関数のスコープと同じである．また，記述可能な条件式は，プログラム中の複数の変数と数値による大小関係，同値・非同値関係，またはそれらの論理積・論理和によって構成される．

複数の満たすべき条件の記述が可能であり，それぞれの条件に対して，それを満たさなかった場合には任意の例外処理を行わせることができる．

また，プログラムが条件を満たしているかの検証は常に行うのではなく，条件式に含まれている変数が更新されたとき，すなわち，変数への代入操作を契機として例外条件の判定を行う．このとき，変数の更新直後に判定を行う場合と，基本ブロックなどあるまとまったプログラムの処理終了後に判定を行う場合を提案する．

3 例外条件検出のオーバーヘッド軽減

本章では，プログラム実行時における例外条件判定オーバーヘッド軽減のための効率的な判定方法について考察する．

3.1 例外条件判定の省略可能な箇所

例外条件判定の対象となっている変数への初期化および定数の代入操作では，コンパイル時に条件を満たしているかどうかのチェックを行うことで，プログラム実行時に条件判定を行う必要はない．

また，重い例外条件判定処理を行わなければならない場合，関係するすべての対象変数の値が前回の条件判定時と同じであるならば，判定結果の履歴を利用することで，不必要な判定処理を省略可能である．

3.2 例外条件判定リストの作成と利用

ここでは, 例外条件判定リストの導入によって，実行時のオーバーヘッドを軽減する方法，また動的な条件式の評価順変更について述べる．

3.2.1 更新された変数に関係ない条件判定の省略

条件式 f が，ある変数 x をその内部に含むとき，条件式 f は変数 x の更新が行われたときの評価対象の条件式となる．そこで，コンパイル時に，条件式に出現する各々の変数 u において，その変数が更新されたときチェックすべき条件式の情報をリスト l_u に保持しておく．この操作によって，プログラム

---

*† Toshiyuki Yamashita, Tsuneo Nakanishi, and Akira Fukuda, "Exception Detecting in Context Oriented Exception Handling."

*1) Department of Electrical Engineering and Computer Science, Faculty of Engineering, Kyushu University.

*2) Graduate School of Information Science and Electrical Engineering, Kyushu University.

*3) System LSI Research Center, Kyushu University.

*4) Computing and Communications Center, Kyushu University.

実行時に，u が更新されると，u に対応するリスト l_u が保持している条件式の判定を順に実行することで，不必要な例外条件判定を省くことができる．

また，基本ブロックの処理終了後に例外条件判定を行う場合，一連の処理の中で複数の対象変数 u, v が更新されるとき，l_u,l_v の重複を許さずに，条件をマージすることによりコンパイル時に新たなリスト l_{uv} を作成する (図 1)．このとき，プログラム実行時の例外条件判定は新たなリスト l_{uv} にある条件式を順に判定することで，変数 u, v に関係ない条件式の判定を省くとともに重複した条件判定を避けることができる．

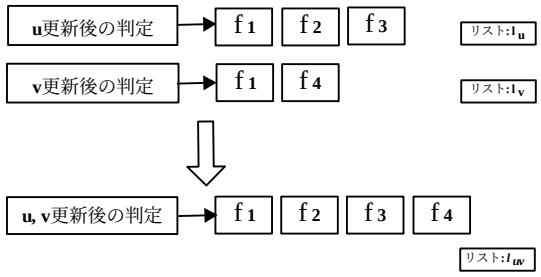

図 1: リスト l_{uv} の作成

3.2.2 例外条件評価順の変更

例外条件判定リストの利用によって，あらかじめプログラマが各条件式に優先度を与えて判定順序を一意に定めたり，プログラム実行時の状況により判定順序の動的な変更が可能となる．

3.3 BDD による論理条件式簡単化

コンテキスト指向例外処理方式では，複雑な条件式の記述を可能とし，プログラム実行時における例外条件式の判定オーバーヘッド軽減のために条件式を BDD(二分決定グラフ) によって表現する．プログラム実行時には条件式と同義な BDD を辿ることで，プログラムが条件を逸脱しているかどうかの判定を行う．BDD では，変数評価順によって，そのサイズや構造が一意に定まることが知られており，よい変数評価順を見つけることが問題となる．

そこで，プログラムに対して最適な BDD であるかどうかの判断基準は BDD のサイズにより決定する場合と，プログラム仮実行時の条件式入力値分布を考慮した目的関数 E により決定する場合を考える．

前者は，単純に BDD のサイズが最小となる変数評価順を最適解とする．

後者は，BDD の根から葉に至る各経路を p_i とするとき，p_i を辿る割合を $d(p_i)$，経路 p_i における階層の深さを h_i，BDD の総経路数を M と定義して，目的関数 E を以下の式により算出する．

$$E = \sum_{i=1}^{M} h_i * d(p_i)$$

E が最小となるような BDD 構造，つまり変数評価順を最適解とする．

4 評価

今回は，条件式を BDD に変換する際の簡単化アルゴリズムにシフティングアルゴリズム [1] を採用し，BDD のサイズが最も小さくなる変数評価順を最適解とする．サンプルプログラムとしてビデオ予約システムを作成し，日付や時刻などに境界条件，整合性条件を与えたコンテキスト指向例外処理方式を導入，実装する．実行の結果，プログラムが条件を逸脱した場合には，指定された適切な例外処理が行われることを確認した．

5 まとめ

本稿では，コンテキスト指向例外処理方式の提案とコンテキスト指向例外処理方式において必要となる，実行時の例外条件検出オーバーヘッドを軽減する方法について述べた．今後は，コンテキスト指向例外処理方式において，プログラムに対する最適 BDD 評価法の考察やコンパイラによる条件式解析および例外条件検出コード埋め込みの自動化の実現を試みる．

謝辞

本研究は，平成 14 年度科学研究費若手研究 (B) の助成を受けている．

参考文献

[1] R.Rudell. ”Dynamic variable ordering for orderd binary decision diagrams.” In International Conference of Computer-Aided Design, 1993.

[2] 中西恒夫，山下俊之，福田晃，「コンテキスト指向例外処理方式とそのコンパイル技術」，情処研報 (OS)，2003 年 2 月 (発表予定)