平成11年度冬学期 第6回情報工学輪講

# デバッグのためのプログラム逆実行

寺田研究室 M1 丸山 一貴
kazutaka@sanpo.t.u-tokyo.ac.jp

# Contents

1. はじめに

2. プログラム実行中の状態一部保存

3. プログラム実行中の状態完全保存

4. Inverse Statement の利用

5. 関数単位の疑似逆実行

6. まとめ

# はじめに

デバッグにおける主な作業

□誤りが発覚する場所にブレークポイント設定

□変数の値などを調べる

□プログラムを一行ずつ実行

□さらに、誤りが発覚する場所にブレークポイント

# 現在のデバッガ

□任意の場所で実行を停止させる

□実行状態 (変数の値) の表示・変更

□順方向にしか実行できない

○思い違いや操作ミスで、バグの現場を通り過ぎてしまう

↓

通り過ぎてしまった場所に戻りたい

↓

プログラムの逆実行

# プログラム逆実行

□以下のような場合に有効

○以前のプログラムの実行状態の取得

○本来見るべき実行点へ、容易に戻る

○通り過ぎた分岐点に戻り、条件を変えてそこから再実行

□従来のデバッガでは手動で先頭から再実行せざるを得ない

↓

処理系やデバッガによる逆実行支援

# 逆実行の手法

- □プログラム実行中の状態一部保存
  - ○DDD (文献[3])
- □プログラム実行中の状態完全保存
  - ○ZStep 95 (文献[4])
- □Inverse Statement の利用
  - ○BDb (文献[8])
- □関数単位の疑似逆実行
  - ○プロファイル用関数の利用

# Contents

1. はじめに

2. プログラム実行中の状態一部保存

3. プログラム実行中の状態完全保存

4. Inverse Statement の利用

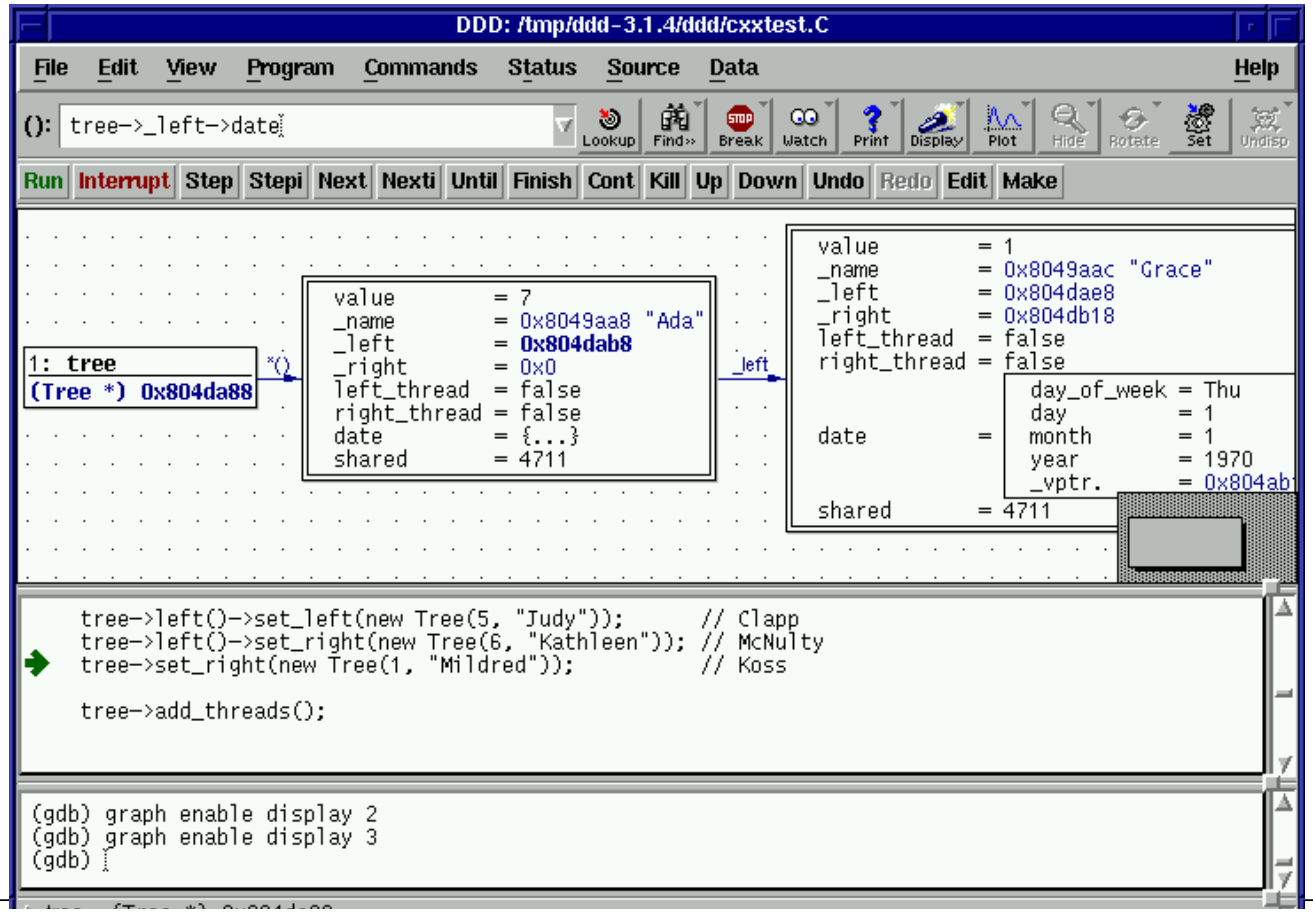
5. 関数単位の疑似逆実行

6. まとめ

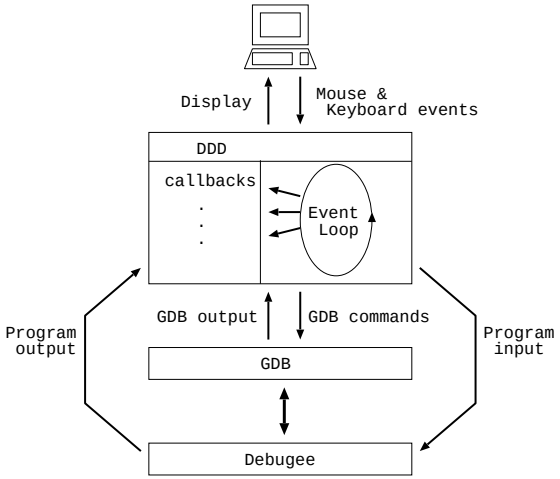
# 状態の一部保存：DDD

- The Data Display Debugger
- UNIX 上の各種デバッガの GUI として実装されるフロントエンド
  - GDB, JDB, Perl debugger, ...
- データ構造のグラフィカル表示が主眼
  - "display"：停止するたびに変数の値を表示する
  - これをグラフィカルに行う
  - サイズ可変のバッファを持ち、"undo" を実現

# DDD の UI


DDD: /tmp/ddd-3.1.4/ddd/cxxtest.C
File Edit View Program Commands Status Source Data Help
(): tree->_left->date
Lookup Find»» Break Watch Print Display Plot Hide Rotate Set Undisp
Run Interrupt Step Stepi Next Nexti Until Finish Cont Kill Up Down Undo Redo Edit Make
1: tree
(Tree *) 0x804da88
*()
value = 7
_name = 0x8049aa8 "Ada"
_left = 0x804dab8
_right = 0x0
left_thread = false
right_thread = false
date = {...}
shared = 4711
_left
value = 1
_name = 0x8049aac "Grace"
_left = 0x804dae8
_right = 0x804db18
left_thread = false
right_thread = false
day_of_week = Thu
day = 1
month = 1
year = 1970
shared = 4711
tree->left()->set_left(new Tree(5, "Judy")); // Clapp
tree->left()->set_right(new Tree(6, "Kathleen")); // McNulty
tree->set_right(new Tree(1, "Mildred")); // Koss
tree->add_threads();
(gdb) graph enable display 2
(gdb) graph enable display 3
(gdb)

# DDD の概要


Display
Mouse &
Keyboard events
DDD
callbacks
Event
Loop
GDB output
GDB commands
Program
output
Program
input
GDB
Debugee

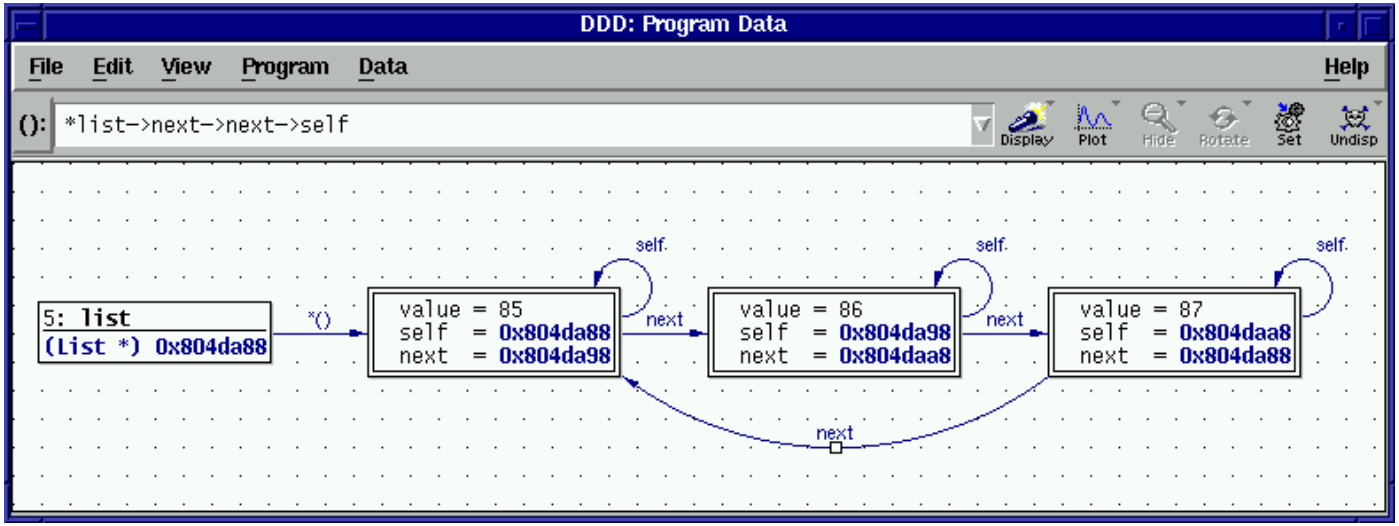
# DDD の "display"

□ポインタでセルを自動的に結ぶ

○セルの表示は手動

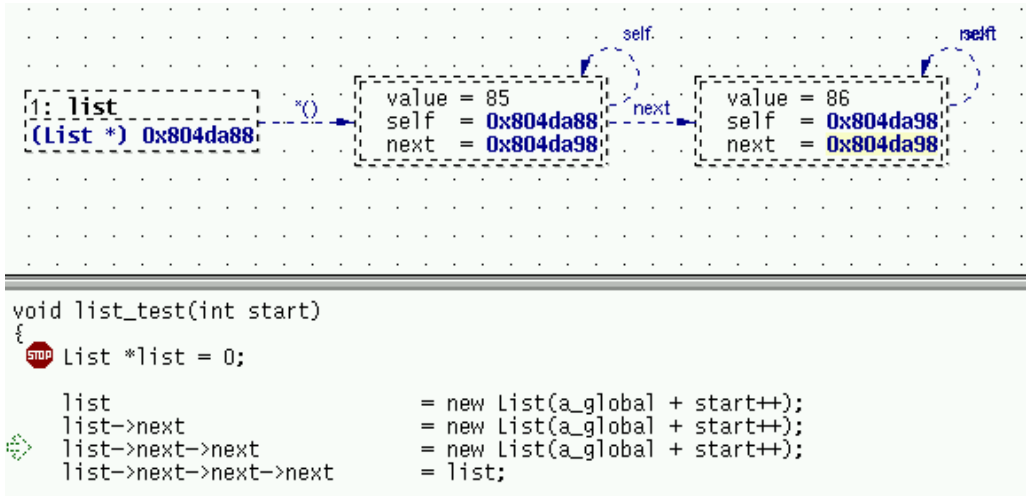
□配置はユーザが自由にできる

○逆に言えば、ユーザがやらされる



# DDD の "undo"

- □"display" ウィンドウの表示が undo される
  - ○"undo" 中はセルの枠と実行点が点線表示



# DDD の問題点

- 所詮は "undo"
  - 「見たいと思った場所」で過去に止まっていないといけない
- 再現の対象は DDD に対する操作だけ
  - "display" していなかった変数は再現できない
- 公開されて使える点は評価できる

# Contents

1. はじめに

2. プログラム実行中の状態一部保存

3. プログラム実行中の状態完全保存

4. Inverse Statement の利用

5. 関数単位の疑似逆実行

6. まとめ

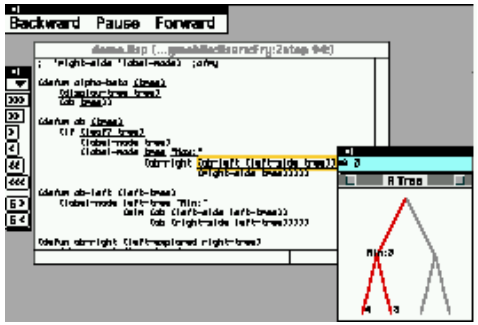
# 状態の完全保存 ： ZStep 95

□プログラマにとっての困難

○「静的な」ソースコードと「動的な」実行状態との対応づけ

□この溝を埋めるためのデバッグ環境

○Macintosh Common Lisp 2.0 上に構築



# ZStep 95 の特徴

- 差分生成による完全なプログラム実行履歴と、出力の履歴の保持

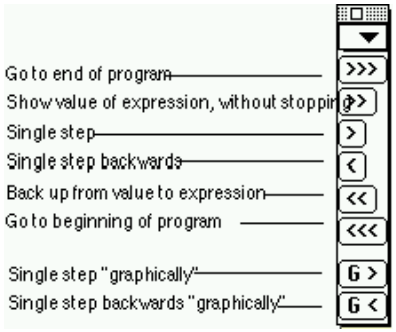
- "Video recorder" 形式ボタンによる実行方向と表示の深さの制御

- その他 (ユーザインタフェースの視点から)
  - プログラム実行をソーステキストの形でアニメーション
  - 現在の実行点に表示される、変数表示用のウィンドウ
  - グラフィカルオブジェクトとそれに対応する式を1クリックでアクセス

# ZStep ： 履歴の保持と実行方向の指示

□専用のインタプリタにより、Lisp の式の評価順を記憶

○">" や "<<" ボタンで、順方向や逆方向の "step", "next"



# ZStep 95 の問題点と主張

□専用インタプリタを用いており、非常に遅い
- ○バグで動かないよりはマシ、という立場

□履歴が大きくなり過ぎる
- ○履歴保存を選択的にオン・オフする必要

□目標は、動的なプログラム視覚化に対する新しいインタフェースの提唱
- ○だから、実用的でなくても ok
- ○初心者教育用には使える

# Contents

1. はじめに

2. プログラム実行中の状態一部保存

3. プログラム実行中の状態完全保存

4. Inverse Statement の利用

5. 関数単位の疑似逆実行

6. まとめ

# Inverse Statement の利用 ： BDb

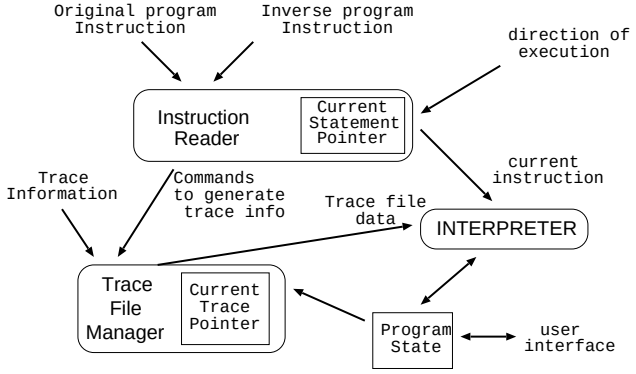
- Inverse Statement の導入
  - プログラムの statement の逆変換
- 計算で分かるものは、履歴として保持しない
  - 保持する履歴の量が減る
- 逆変換できない分は Trace File に保存
- 専用インタプリタと3つのファイル
  - Original Program
  - Trace File
  - Inverse Program

# Inverse Program の例 ： 条件分岐

| | Original program | Trace file | | Inverse program |
|---|---|---|---|---|
| 8 | a = 20; | a = 0; | 8 | /* Read history */ |
| 9 | if(a < 10){ | | 9 | |
| 10 | a = 1; | | 10 | /* Read history */ |
| 11 | b *= a; | | 11 | b /= a; |
| 12 | } | Goto_line 9; | 12 | /* Read history */ |
| 13 | else if(a > 10){ | | 13 | |
| 14 | a = 100; | a = 20 | 14 | /* Read history */ |
| 15 | c += a; | | 15 | c -= a; |
| 16 | } | Goto_line 15; | 16 | /* Read history */ |
| 17 | else{ | | 17 | |
| 18 | a = 10; | | 18 | /* Read history */ |

# BDb の構成

□専用インタプリタを使っている



# BDb の特徴と問題点

□履歴は小さくなっていると思われる
  ○が、計測などの情報はない

□専用インタプリタによる実行
  ○遅くなると思われるが、実行速度の情報もない

□ZStep と同じく、再生・表示するだけ
  ○変数の値の変更などはできないようである

↓

デバッガを標榜するからには、
遡ったところでも変数の値の変更などをしたい

# Contents

1. はじめに

2. プログラム実行中の状態一部保存

3. プログラム実行中の状態完全保存

4. Inverse Statement の利用

5. 関数単位の疑似逆実行

6. まとめ

# 関数単位の疑似逆実行

□これまでの手法の問題点

○実行速度の低下

○専用の処理系 → 記述量が大きい

○プログラミング言語のサブセットしか使えない

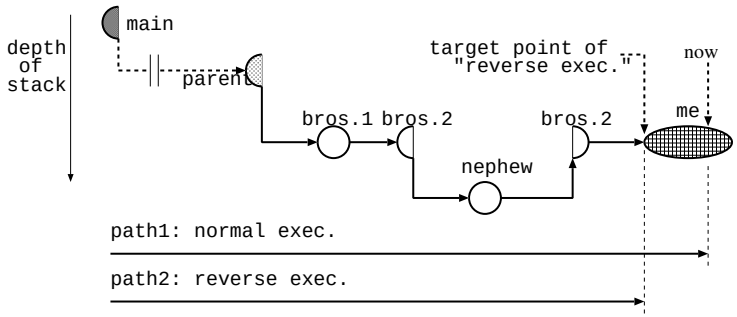
○デバッガとしては、従来型のものに及ばない

↓

疑似的な逆実行を行う

# 疑似逆実行

□人間が既存のデバッガで行う手法を自動化



# プロファイル用関数の利用

□プロファイル情報取得関数 mcount

□全ての関数の入口で呼ばれる関数

□コンパイラが自動的に挿入
- ○GCC の -pg オプション
- ○ユーザはオプションを指定するだけで良い

□標準の mcount は使わず、自分で同名の関数を定義

↓

専用のコンパイラは必要ない

# mcount 呼び出しの挿入

source code

```
f(){
  a();
  b();
}
a(){
 [body]
}
b(){
 [body]
}
```

with "-pg" opt.

```
f(){
  mcount();
  a();
  b();
}
a(){
  mcount();
 [body]
}
b(){
  mcount();
 [body]
}
```

# タイムスタンプの付与

□自作の mcount で、呼ばれるごとに変数を1増やす
- ○これを「タイムスタンプ」と呼ぶ

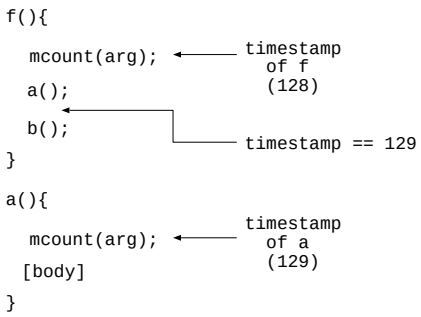
□全ての関数呼び出しにタイムスタンプがつく
- ○タイムスタンプで、関数呼び出しが unique に決まる
- ○条件付ブレークポイントで、任意の関数呼び出しの入口へ制御を移動できる

↓

問題は「戻り場所のタイムスタンプ」の取得

# タイムスタンプの更新

□現在実行中の関数の入口すら、簡単には分からない

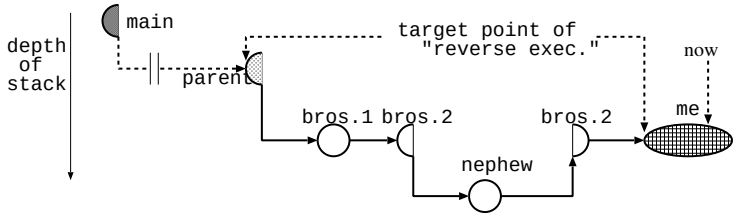


# タイムスタンプの取得

- タイムスタンプは関数呼び出しに固有の情報
  - スタックフレームに持てれば良い
  - コンパイラ間のポータビリティに問題がある
- mcount 内でシミュレートする
  - アーキテクチャやコンパイラに非依存

# mcount でのシミュレート

- スタックの各レベルで最新のタイムスタンプを保持
- mcount だけでは、入口しか分からない (近似的なスタック)
  - スタックを正確に調べる [scan pass]
  - 実際に制御を移す [execute pass]
  - 逆実行に2パス必要



# 特徴

- □専用の処理系が不要
  - ○mcount を C で記述するだけで良い
- □ポータビリティが高い
  - ○コンパイラさえ対応していれば良い
- □逆実行に2パスを要する
- □対象プログラムに再現性が必要

# 1パス実行

- 関数の出口が分からないのが問題
- mcount の「出口版」があれば良い
- 関数の返り値があるので、簡単にはいかない
  - ソースコードを変更するだけではダメ
  - アセンブリ言語レベルでやるとポータビリティが犠牲に
    - Intel Architecture で、コンパイラの出すアセンブリソースへのフィルタとして実装した

↓

ポータブルな形でコンパイラに実装したい (研究中)

# Contents

1. はじめに

2. プログラム実行中の状態一部保存

3. プログラム実行中の状態完全保存

4. Inverse Statement の利用

5. 関数単位の疑似逆実行

6. まとめ

# まとめと今後の展望

□実行履歴再生や逆実行に関する手法を紹介

□最近は落ち目の分野であった
- ○正攻法とも言うべき手法の限界

□mcount を利用するような「曲芸的」手法の登場
- ○全自動とはいかない
- ○実用的な速度で動作

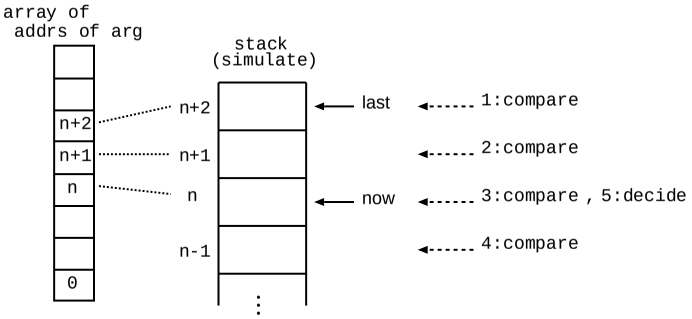
↓

コンパイラへの工夫次第で、研究すべきところはある

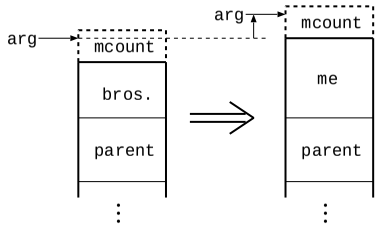
以上

# mcount での処理

- □シミュレートしているスタックフレームのアドレスと比較
  - ○前回より今回の方がアドレスが深い → スタックが1深くなった
  - ○前回より今回の方がアドレスが浅い → スタックが浅くなった
    - ▹いくつ浅くなったか (リターンしたか) 分からない



# シミュレートが失敗する場合

□「兄」のフレームより「自分」のフレームの方が大きいとき

- ○誤って、実際より深いと思ってしまう
- ○余分なフレームを含んだスタックができる
- ○[scan pass] でデバッガを使えば修正可