Secure Compiler Seminar (2006/9/19)

米澤研 M2 佐藤秀明

今回の内容

- Static Typing for a Faulty Lambda Calculus
  - David Walker, Lester Mackey, Jay Ligatti, George A. Reis, and David I. August
  - ICFP 2006
- ハードウェアのエラーを多重化で回避
  - 多数決により計算結果の信頼性を向上
- システムの正しさを型で保証
  - 型の整合性
  - 各系の独立性

Transient Faults とは

- 外部要因によりトランジスタの状態が変化
  - エネルギー粒子の衝突
- プログラムの正しい実行を妨害
  - レジスタやメモリのビット値を反転
- 実際に被害が発生
  - AOL, eBay, Los Alamos Neutron Science Center, ...

型システムでなんとかしよう！

λ_{zap}: Faulty Lambda Calculus

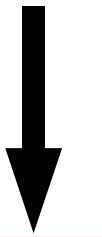
- 評価中に Transient Faults が起こり得る
  - 操作的意味論に定義
- 高々 1 者が誤っても検知かつ修復可能
  - 同じ計算を 3 回分実行
- キャッシュやメモリの値は保護対象外
  - 既存のハードウェア的手法で十分
  - CPU 内の値のみを対象とする

λ_{zap} の例 (1)

- out: 多数決の結果を出力
- 各複製の value を色で区別
  - Red/Green/Blue
  - 評価とは関係なし
- [・] は tuple ではない
  - 各要素はすべてレジスタ上にある

```
let x = 2 in
let y = x + x in
out y
```

$\lambda_{\rightarrow}$



```
let [x1,x2,x3] = [R 2, G 2, B 2] in
let [y1,y2,y3] = [x1 + x1, x2 + x2, x3 + x3] in
out [y1,y2,y3]
```

λ_{zap} の例 (2)

- 分岐条件も 3 回複製

```
if [eb1,eb2,eb3] then e1 else e2
```

λ_{zap} の例 (3)

- 関数適用も 3 回実行
- 実行コード自体は複製しない
  - 関数ポインタ / クロージャへのポインタを複製

```
let [f1,f2,f3] =
  λ[x1,x2,x3].
    [x1 + R 1,x2 + G 1,x3 + B 1]
in
[f1,f2,f3] [R 7,G 7,B 7]
```

λ_{zap} Syntax

Colors	C	$::=$	$R \mid G \mid B$
Uncolored Types	I	$::=$	$\texttt{int} \mid \texttt{bool} \mid T_1 \to T_2$
Colored Types	T	$::=$	$C\ I \mid [R\ I, G\ I, B\ I]$
Code Locations	l		
Uncolored Vals	w	$::=$	$l \mid n \mid \texttt{true} \mid \texttt{false}$
Colored Vals	v	$::=$	$C\ w$
Expressions	e	$::=$	$x \mid v \mid e + e \mid e \leq e$
		$\mid$	$\texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$
		$\mid$	$\lambda[x_1{:}R\ I, x_2{:}G\ I, x_3{:}B\ I].e \mid e\ e$
		$\mid$	$[e_1, e_2, e_3]$
		$\mid$	$\texttt{let}\ [x_1, x_2, x_3] = e\ \texttt{in}\ e$
		$\mid$	$\texttt{let}\ x = e\ \texttt{in}\ e \mid \texttt{out}\ e;\ e$

λ_{zap} の Operational Semantics(1)

$$(M; e) \longrightarrow_k^s (M'; e')$$

- M: code location から関数本体への map
- e: expression
- s: out コマンドにより出力された値の列
- k: 起こった fault の数

λ_{zap} の Operational Semantics(2)

- 評価戦略は Call-by-value かつ left-to-right
- 詳細は Figure 2 を参照
  - E[e]: 次に評価される部分式が e であるような式
    - (Octxt)
  - F[v]: fault が起こり得る部分式が v であるような式
    - (zap)
  - addtot(): 加算 (+)
  - lesstot(): 比較演算 (<)
  - vote(): 多数決により正しい値を選択

プログラム変換による信頼性の低下

- 複製された計算結果が相互に依存してしまう
  - 例 1: $\lambda_{\rightarrow}$ から λ_{zap} への変換器にバグが存在
  - 例 2: 最適化ルーチンがコードの冗長性を解消

```
let [x1,x2,x3] = [2, 2, 2] in
let [y1,y2,y3] = [x1 + x1, x2 + x2, x3 + x3] in
out [y1,y2,y3]
```

↓ 共通部分式の削除

```
let x1 = 2 in
let y1 = x1 + x1 in
out [y1,y1,y1]
```

多数決の意味がない！

λ_{zap} の Type System

- 複製された各計算結果の相互独立性を保証
  - Colors の混合を制限
- Fault の起きた値は任意の型をとれる
  - (Tany)
- 詳細は Figure 4 を参照
  - Z: fault が起きたかもしれない値の色
    - 証明に用いる
- Figure 5 は機械の状態についての typing
  - M は location からクロージャへの map
  - L は location から型への map

型システムの性質 :Safety(1)

- Definition[Safe States]

  状態 (M;e) が安全であるとは、以下のいずれかをいう。

  (1) e が value であるか、

  (2) 次の状態 (M';e') が存在するか、または

  (3) 多数決が必要な場面で正しい値を導けなくなった。

- (3) の場合は処理系が exit() または例外 throw
  - vote() は 3 つの値が全て異なるとき undefined

型システムの性質 :Safety(2)

- Lemma[Progress]
  - 型付けできる状態は安全である。
- Lemma[Preservation]
  - fault が 1 色以下ならば、状態遷移の前後で
    型は保存される。
- Theorem[Safety]
  - 型付け可能な状態 S から到達可能な状態 F について、
    S から F までに起きた fault が 1 回以下なら、
    F は安全である。

型システムの性質 :Simulation(1)

- Definition[C-simuate]

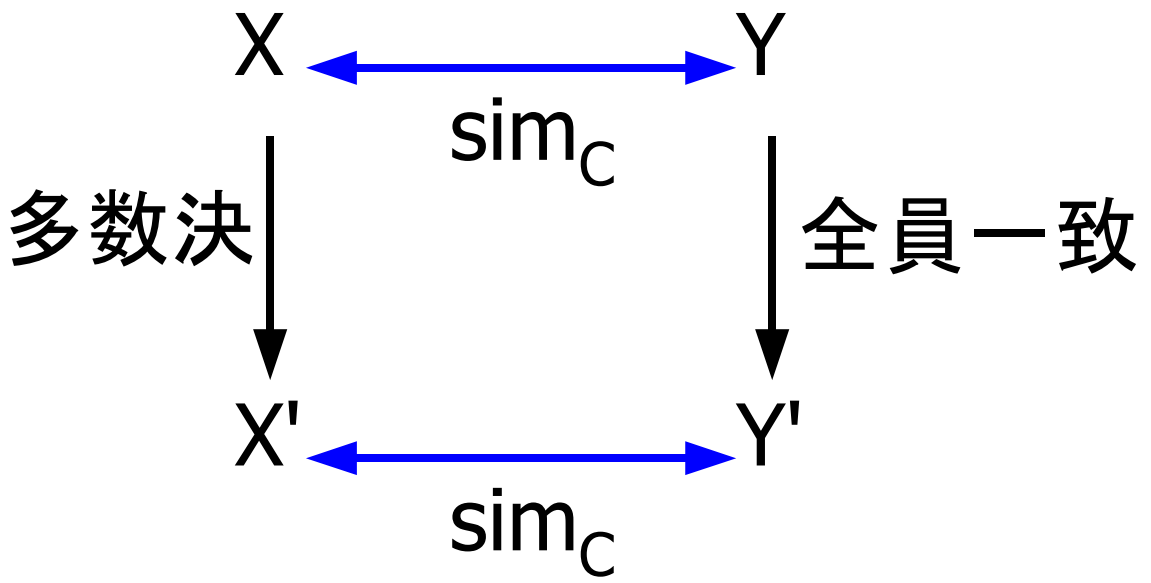
  ある色 C があって、
  2 つの状態 X 、Y の中の”C w”の形の出現が
  wの部分のみ異なっている場合、
  X と Y は C-simulate しているといい、
  以下のように表記する。

$$X \; sim_C \; Y$$

  - C は fault が起きているかもしれない色を指す
    → C に色づけされた値 w は異なっている可能性がある

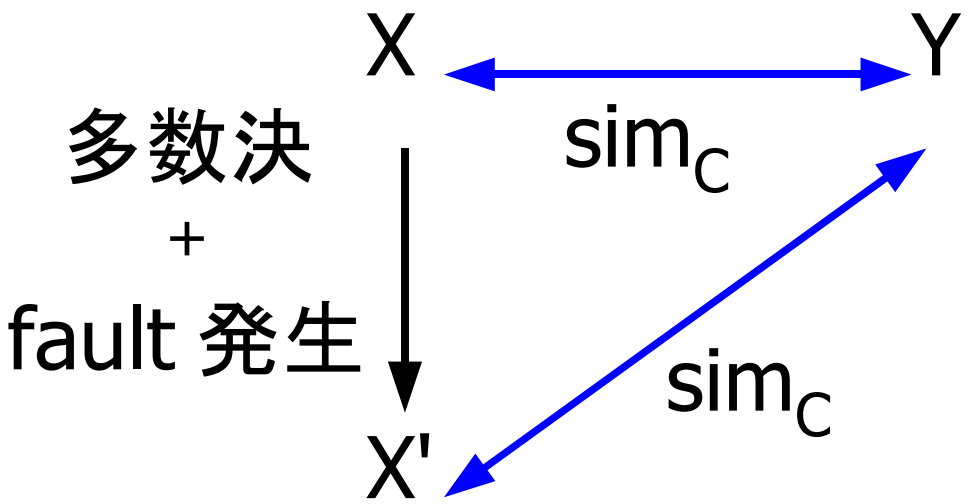
型システムの性質 :Simulation(2)

- Lemma[Fault-Free Simulation]
  - fault が起こらないなら、
    一方を多数決制で、
    他方を全員一致制でそれぞれ遷移させても、
    C-simulate の関係は保存する。
    - 全員一致 : vote() の引数は全て同じ値でなければならない



型システムの性質 :Simulation(3)

- Lemma[Faulty Simulation]
  - fault が起こる場合、
    一方を多数決制で遷移させても、
    C-simulate の関係は保存する。

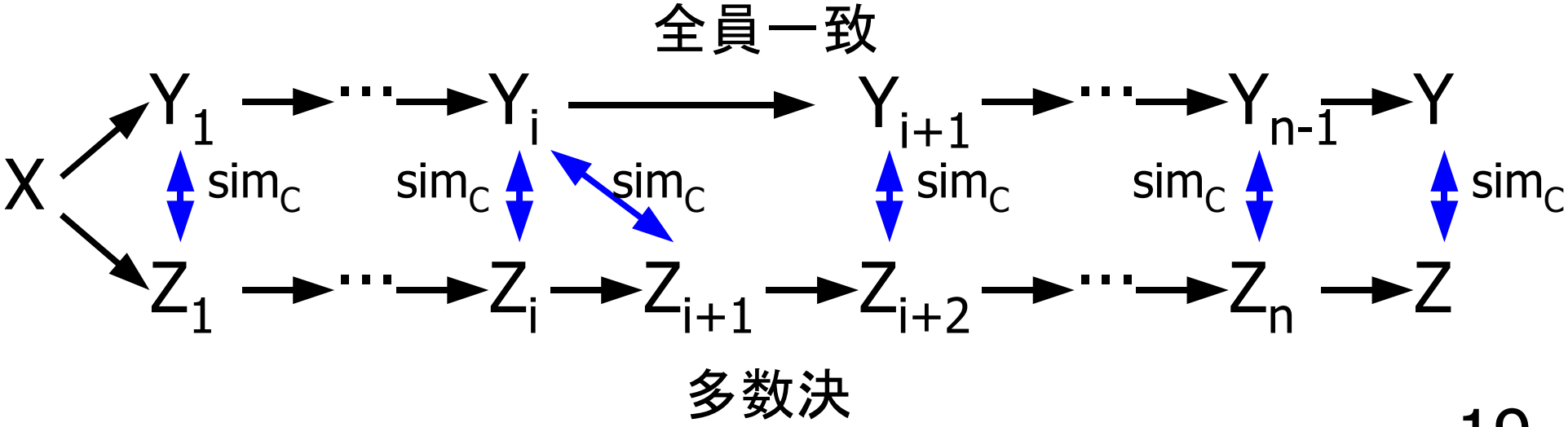


型システムの性質 :Simulation(4)

- Theorem[Fault Tolerance 1]
  - 全員一致制による状態遷移が stuck しないなら、
    fault が 1 回以下である限り、
    多数決制による状態遷移も stuck しない。

型システムの性質 :Simulation(5)

- Theorem[Fault Tolerance 2]
  - 状態 X から状態 Y へ全員一致制で n ステップの遷移が可能で、かつ
    状態 X から状態 Z へ多数決制で 1 回だけ fault の起こる (n+1) ステップの遷移が可能ならば、
    ある色 C について Y と Z は C-simulate している。

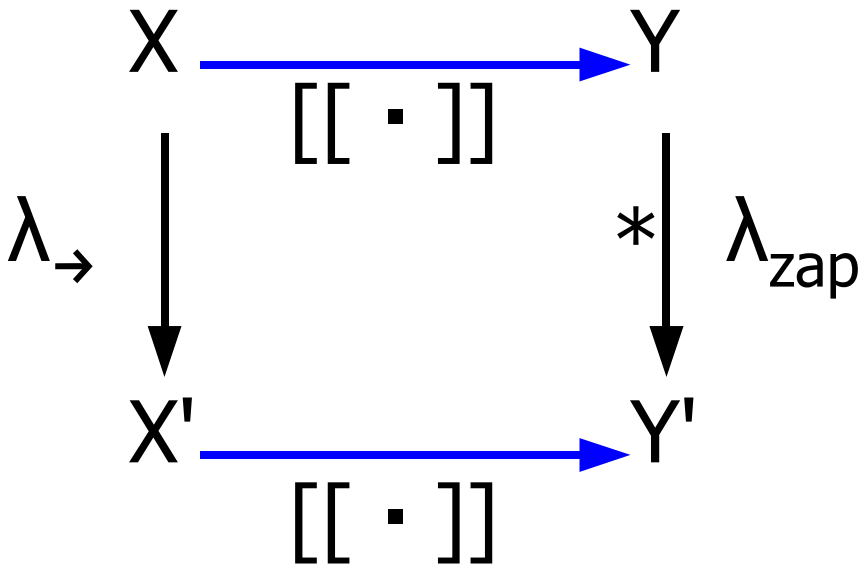


$\lambda_{\rightarrow}$ から λ_{zap} への変換

- $[[\cdot]]$: すべての計算を 3 度複製
- 詳細は Figure 6, 7 を参照

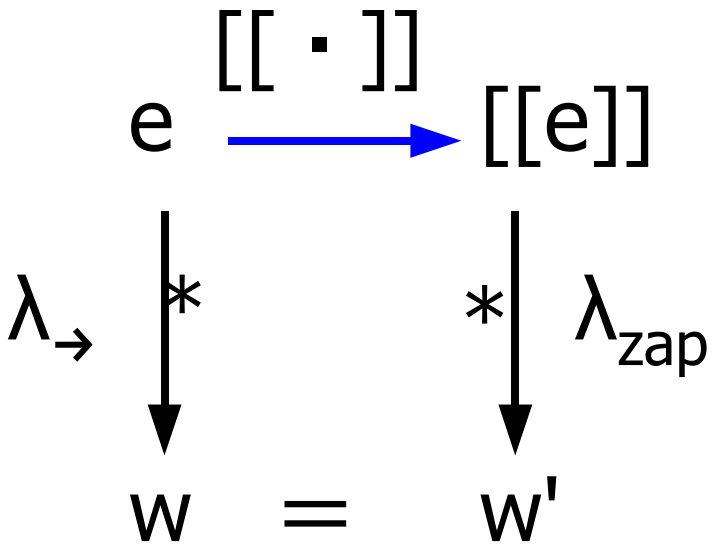
$\lambda_{\rightarrow}$ から λ_{zap} への変換の性質 (1)

- Lemma
  - 変換操作[[・]]は型を保存する。
  - 変換操作[[・]]と代入操作 [w/x] は可換である。
  - 変換操作[[・]]と状態遷移操作→は可換である。



$\lambda_{\rightarrow}$ から λ_{zap} への変換の性質 (2)

- Theorem[End-to-end Reliability]
  - $\lambda_{\rightarrow}$ で初期状態から式 e を評価して得た値 w と、
    λ_{zap} で初期状態から式[[e]]を評価して得た値 w' は、
    λ_{zap} 評価中の fault が 1 回以下である限り、等しい。



まとめ

- λ_{zap}: faulty lambda calculus の提案
  - 計算の 3 倍多重化による信頼性の向上
  - 型の整合性と各複製の独立性を型システムで保証
  - $\lambda_{\rightarrow}$ から λ_{zap} への変換を定義