Synthese von Kontrollfluss für eine synchrone Datenflusssprache

Verteidigungsvortrag zur Masterarbeit

Bearbeiter: Andreas Loth
Betreuer: Prof. Dr. Baltasar Trancón Widemann
Dr. Carl Friedrich Bolz

Datum: 11.06.2015

TECHNISCHE UNIVERSITÄT
ILMENAU

Gliederung

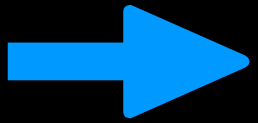
- Einleitung, Aufgabenstellung
- Sig
- Synthese von Kontrollfluss
- Vorbereitung der nächsten Compilerphase
- Statistiken
- Evaluation
- Zusammenfassung
- Ausblick

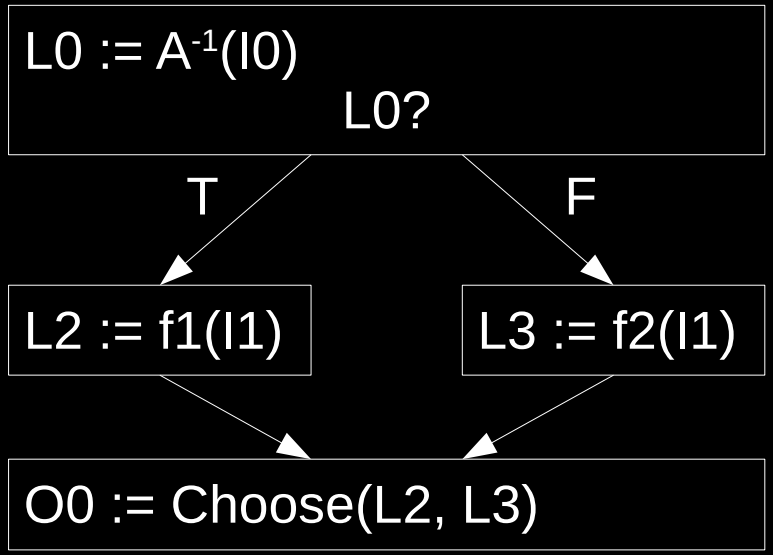
Einleitung

- Programmierer: System.out.println("EICAR-STANDARD-ANTIVIRUS-TEST-FILE!");
- CPU (ASCII, 16 bit x86 COM): X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
- Compiler
  - Übersetzung
  - Optimierung

Einleitung
Aufgabenstellung

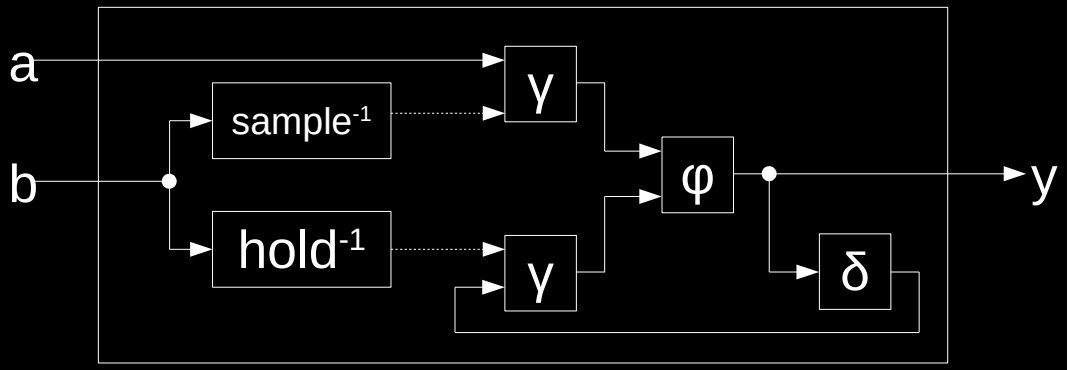
- erst berechnen, dann selektieren
- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  L2 := f1(I1)
  L3 := f2(I1)
  L4 := Guard(L2, L0)
  L5 := Guard(L3, L1)
  O0 := Choose(L4, L5)



- im konkreten Fall nicht benötigtes überspringen
- 

Sig

- Datenflusssprache, getaktet synchron
- Kontrollvariablen erzeugt durch Destruktoren
- Kontrollfluss
  - Guard (γ): bedingte Weiterleitung eines Datums, Konjunktion
  - Choose (φ): Auswahl, Disjunktion
- Verzögerung (δ)
  → Zustand (Vor-, Nach-)
- Wert ⊥ : undefiniert,
  Kontrollvariablen: false
- Init- / Loop-Teil



Synthese von Kontrollfluss

- mehrere voneinander abhängige Phasen

Synthese von Kontrollfluss
Das Zwischenformat dieser Arbeit

- doppelt verkettete Liste
  → Einfügen / Löschen effizient möglich
- Elemente:
  - Statement / Anweisung
  - Sprung
  - Label
  - (GoTo)
- Überführung Eingabeformat → Zwischenformat:
  nur benötigte Statements

Synthese von Kontrollfluss
Vereinfachung von Bedingungen

- keine Negation möglich (Guard: UND, Choose: ODER)
- Analyse
  - konstante boolesche Variablen (jeweils eine Menge)
  - zurückrechnen boolescher Verknüpfungen (Variable → DNF)
  - exklusive Variablenmengen („Negation“)
    - algebraische Datentypen (Beispiel: farben = (rot, grün, blau, gelb))
    - Variablen paarweise UND-Verknüpft: falsch
    - vollständig ⇔ alle Variablen ODER-Verknüpft: wahr

Synthese von Kontrollfluss
Vereinfachung von Bedingungen

- Vereinfachungsregeln
  - Idempotenz
  - neutrales Element
  - wahre und falsche Variablen
  - Variable und ihre „Negation“ innerhalb einer Konjunktion
  - Negation einer einzigen Variablen
  - Absorptionsgesetz

Synthese von Kontrollfluss
Def-Use Kette

- Datenstruktur
  - (Variable, Definition, {Uses})
- SSA: jede Variable nur einmal geschrieben
  → leicht auslesbar
- Guard
  - nicht Use für Kontrollvariablen
  - Use für Variablen in DNF, die Kontrollvariablen repräsentieren
    - a = (b UND c) ODER (d UND e UND f)
    - … = Guard(…, a)

Synthese von Kontrollfluss
Notwendigkeit von Variablen und Statements

- Guards können berechnete Werte verwerfen
- Statements nur notwendig, wenn berechnete Werte benutzt werden
- Berechnung
  - Bedingung
  - direkte Guards

Synthese von Kontrollfluss
Synthese von Sprüngen aus Guards

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  L2 := f1(I1)
  L3 := f2(I1)
  L4 := Guard(L2, L0)
  L5 := Guard(L3, L1)
  O0 := Choose(L4, L5)

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  { ((L0))
  L2 := f1(I1)
  L4 := Move(L2)
  } goto Label_1
  label: Label_1
  { ((L1))
  L3 := f2(I1)
  L5 := Move(L3)
  } goto Label_2
  label: Label_2
  O0 := Choose(L4, L5)

Synthese von Kontrollfluss
Synthese von Sprüngen aus Guards

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  L2 := f1(I1)
  L3 := f2(I1)
  L4 := Guard(L2, L0)
  L5 := Guard(L3, L1)
  O0 := Choose(L4, L5)

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  L3 := f2(I1)
  { ((L0))
      L2 := f1(I1)
      L4 := Move(L2)
  } goto Label_1
  label: Label_1
  L5 := Guard(L3, L1)
  O0 := Choose(L4, L5)

Synthese von Kontrollfluss
Synthese von Sprüngen aus Guards

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  L3 := f2(I1)
  { ((L0))
  L2 := f1(I1)
  L4 := Move(L2)
  } goto Label_1
  label: Label_1
  L5 := Guard(L3, L1)
  O0 := Choose(L4, L5)

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  { ((L0))
  L2 := f1(I1)
  L4 := Move(L2)
  } goto Label_1
  label: Label_1
  { ((L1))
  L3 := f2(I1)
  L5 := Move(L3)
  } goto Label_2
  label: Label_2
  O0 := Choose(L4, L5)

Synthese von Kontrollfluss
Synthese von Sprüngen aus Guards

1) …
2) L10 := f1(L1)
3) L11 := f2(L10)
4) L12 := f3(L10)
5) L13 := Guard(L11, L2)
6) L14 := Guard(L12, L3)
7) L15 := Choose(L13, L14)
8) L16 := Guard(L15, …)
9) …

{G5, G6} ; {G6, G8} ; {G8}

Synthese von Kontrollfluss
Synthese von Sprüngen aus Guards

1) …
2) L10 := f1(L1)
3) L11 := f2(L10)
4) L12 := f3(L10)
5) L13 := Guard(L11, L2)
6) L14 := Guard(L12, L3)
7) L15 := Choose(L13, L14)
8) L16 := Guard(L15, …)
9) …

{G5, G6} ; {G6, G8} ; {G8}

Synthese von Kontrollfluss
Synthese von Sprüngen aus Guards

1) …
2) L10 := f1(L1) {G5, G6} ; {G6, G8} ; {G8}
3) L11 := f2(L10)
4) L12 := f3(L10)
5) L13 := Guard(L11, L2)
6) L14 := Guard(L12, L3)
7) L15 := Choose(L13, L14)
8) L16 := Guard(L15, …)
9) …

Synthese von Kontrollfluss
Synthese von Sprüngen aus Guards

1) …
2) L10 := f1(L1) {G5, G6} ; {G6, G8} ; {G8}
3) L11 := f2(L10)
4) L12 := f3(L10)
5) L13 := Guard(L11, L2)
6) L14 := Guard(L12, L3)
7) L15 := Choose(L13, L14)
8) L16 := Guard(L15, …)
9) …

Synthese von Kontrollfluss
Statements mit mehreren direkten Guards

- …
  L10 := f1(L1)
  { ((L2))
      L11 := f2(L10)
      L13 := Move(L11)
  } goto Label_10
  label: Label_10
  { ((L3))
      L12 := f3(L10)
      L14 := Guard(L12, L3)
  } goto Label_11
  label: Label_11
  L15 := Choose(L13, L14)

- …
  { ((L2) OR (L3))
      L10 := f1(L1)
  } goto Label_12
  label: Label_12
  { ((L2))
      L11 := f2(L10)
  …

Synthese von Kontrollfluss
Versch. der Berechnungen von Kontrollvariablen

- …
  { ((L2) OR (L3))
  L10 := f1(L1)
  } goto Label_12
  label: Label_12
  L2 := A^{-1}(I0)
  L3 := B^{-1}(I0)
  …

- …
  L2 := A^{-1}(I0)
  L3 := B^{-1}(I0)
  { ((L2) OR (L3))
  L10 := f1(L1)
  } goto Label_12
  label: Label_12
  …

- Sicherheit der Codeverschiebung: Def-Use Ketten

Synthese von Kontrollfluss
Schachtelung der Sprünge

- { ((L2) OR (L3))
  ...
  } goto Label_12
  label: Label_12
  { ((L2))
  ...
  } goto Label_10
  label: Label_10
  { ((L3))
  ...
  } goto Label_11
  label: Label_11

- { ((L2) OR (L3))
  ...
  { ((L2))
  ...
  } goto Label_10
  label: Label_10
  { ((L3))
  ...
  } goto Label_11
  label: Label_11
  } goto Label_12
  label: Label_12

- Sicherheit der Codeverschiebung: Def-Use Ketten

Synthese von Kontrollfluss
Sprungbedingungen vereinfachen

- definierte Variablen
- Kontext
  (bereits genommene Sprünge)
- Vereinfachungsregeln
  (mit exklusiven Variablenmengen)

- { ((L0))
  { ((L4 AND L0 AND L1 AND L2)
  OR (L4 AND L0 AND L1 AND L3))
  } goto …
  L4 := …
  } goto ...

- { ((L0))
  { ((L1)) [L2 OR L3 = true]
  } goto …
  L4 := …
  } goto ...

Synthese von Kontrollfluss
Sprungbedingungen vereinfachen

- definierte Variablen
- Kontext (bereits genommene Sprünge)
- Vereinfachungsregeln (mit exklusiven Variablenmengen)

- { ((L0))
  { ((L4 AND L0 AND L1 AND L2)
  OR (L4 AND L0 AND L1 AND L3))
  } goto …
  L4 := …
  } goto ...

- { ((L0))
  { ((L1))
  } goto …
  L4 := …
  } goto ...

[L2 OR L3 = true]

Synthese von Kontrollfluss
Sprungbedingungen vereinfachen

- definierte Variablen
- Kontext
  (bereits genommene Sprünge)
- Vereinfachungsregeln
  (mit exklusiven Variablenmengen)

- { ((L0))
    { ((L4 AND L0 AND L1 AND L2)
      OR (L4 AND L0 AND L1 AND L3))
    } goto …
    L4 := …
  } goto ...

- { ((L0))
    { ((L1))
    } goto …
    L4 := …
  } goto ...

[L2 OR L3 = true]

Synthese von Kontrollfluss
Sprünge mit der gleichen Bedingung

- { ((L0))
  L1 := …
  …
  } goto Label_0
  label: Label_0
  …
  { ((L0))
  L2 := …
  …
  } goto Label_1
  label: Label_1

- …
  { ((L0))
  L1 := …
  …
  L2 := …
  …
  } goto Label_1
  label: Label_1

Synthese von Kontrollfluss
Copy Propagation

- …
  { ((L0))
  L2 := Move(I1)
  } goto Label_3
  label: Label_3
  { ((L1))
  L3 := f2(I1)
  L4 := Move(L3)
  } goto Label_4
  label: Label_4
  O0 := Choose(L2, L4)
  O1 := Move(I2)

- …
  { ((L0))
  L2 := Move(I1)
  } goto Label_3
  label: Label_3
  { ((L1))
  L3 := f2(I1)
  } goto Label_4
  label: Label_4
  O0 := Choose(L2, L3)
  O1 := Move(I2)

Synthese von Kontrollfluss
Sprünge organisieren

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  { ((L0))
  L2 := f1(I1)
  } goto Label_1
  label: Label_1
  { ((L1))
  L3 := f2(I1)
  } goto Label_2
  label: Label_2
  O0 := Choose(L2, L3)

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  label: Label_1
  label: Label_2
  { ((L0))
  L2 := f1(I1)
  } goto Label_3
  { ((true))
  L3 := f2(I1)
  } goto Label_3
  label: Label_3
  O0 := Choose(L2, L3)

Synthese von Kontrollfluss
Dead Code Elimination (DCE)

- Organisation der Sprünge: Sprungbedingung immer wahr
  → toter Code kann entstehen

- L0 := A^{-1}(I0)
  L1 := B^{-1}(I0)
  label: Label_1
  label: Label_2
  { ((L0))
  L2 := f1(I1)
  } goto Label_3
  { ((true))
  L3 := f2(I1)
  } goto Label_3
  label: Label_3
  O0 := Choose(L2, L3)

Synthese von Kontrollfluss
Generierung des Codes zur Bedingungsprüfung

- { ((L0))
- { ((L10 AND L11 AND L12))
- { ((L20) OR (L21) OR (L22))
- { ((L30) OR (L31 AND L32)
  OR (L33 AND L34))

→ { ((L0)

→ L13 := Guard(L10, L11, L12)
{ ((L13))

→ L23 := Choose(L20, L21, L22)
{ ((L23))

→ L35 := Guard(L31, 32)
L36 := Guard(L33, L34)
L37 := Choose(L30, L35, L36)
{ ((L37))

Synthese von Kontrollfluss
Common Subexpression Elimination (CSE)

- redundante Berechnungen entfernen
- kein 3-Adress-Code:
  keine Auflösung bei mehr als 2 Operatoren notwendig
- nur Ausdrücke (Guard, Choose) zur Berechnung von Kontrollvariablen
- nicht nur lexikalisch gleiche Ausdrücke, sondern Wert (DNF) berücksichtigt
- redundante Berechnungen → φ-Knoten
- kann toten Code und Kopieranweisungen erzeugen

Synthese von Kontrollfluss
Überführung Zwischenformat → Ausgabeformat

- keine leeren Basic Blocks (damit keine unnötigen Sprünge)
  Ausnahme: End-Basic Block
- Sprungbedingung
  - Tautologie: Zielliste in aktuelle Programmliste einfügen (+ GoTo)
  - Kontradiktion: Sprung übergehen
  - sonst: Branch erstellen
- Basic Blocks erst erstellen, wenn benötigt (Anweisung / Branch)
- Referenzen setzen, wenn Basic Block erstellt wurde

Synthese von Kontrollfluss
Aufräumen der lokalen Variablen

- Init- und Loop-Teil getrennt behandelt
  → neu erstellte local-Variablen nur in einem Teil genutzt
- Copy Propagation, DCE: Variablen gelöscht
- Konsolidierung der Variablendeklaration,
  keine Änderung der lebenden Variablen
- erster Durchlauf: Anzahl der verwendeten Variablen pro Typ
- zweiter Durchlauf: Variablenersetzungen

Vorbereitung der nächsten Compilerphase

- nicht alle Operationen unterstützen $\perp$ als Eingabe
- bisher: Laufzeitprüfung
- Prüfung zur Compilezeit:
  Ausführungsbedingung des Basic Blocks impliziert Definiertheitsbedingung der Operanden
  → keine Laufzeitprüfung erforderlich

Statistiken

Produktivcode	alles	ohne Javadoc	ohne Kommentare
LOC	7 902	5 303	5 189
KB	251	169	162

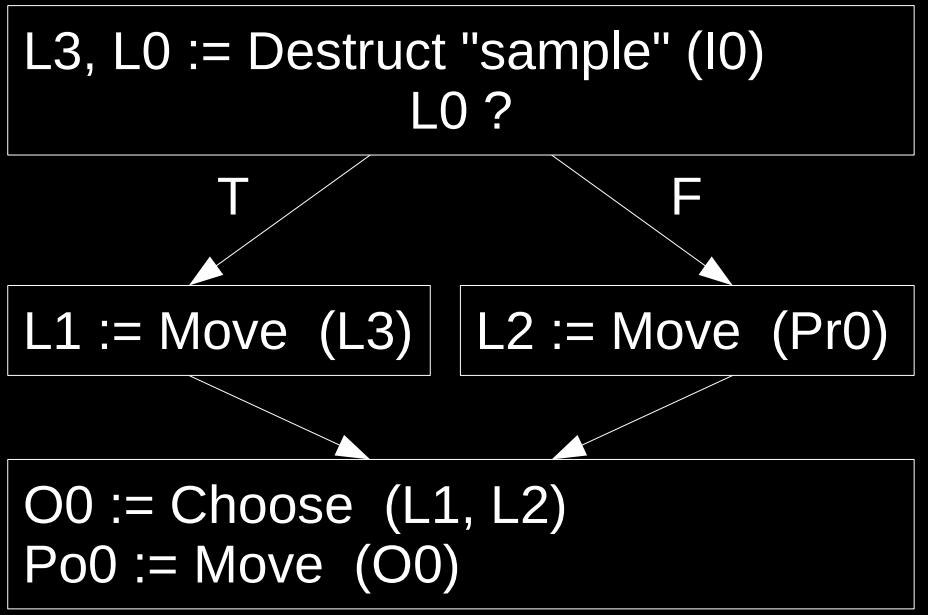
Testcode	alles	ohne Kommentare
LOC	3 591	3 189
KB	148	137

Evaluation

- Struktur des Kontrollflusses
- Definiertheitsanalyse

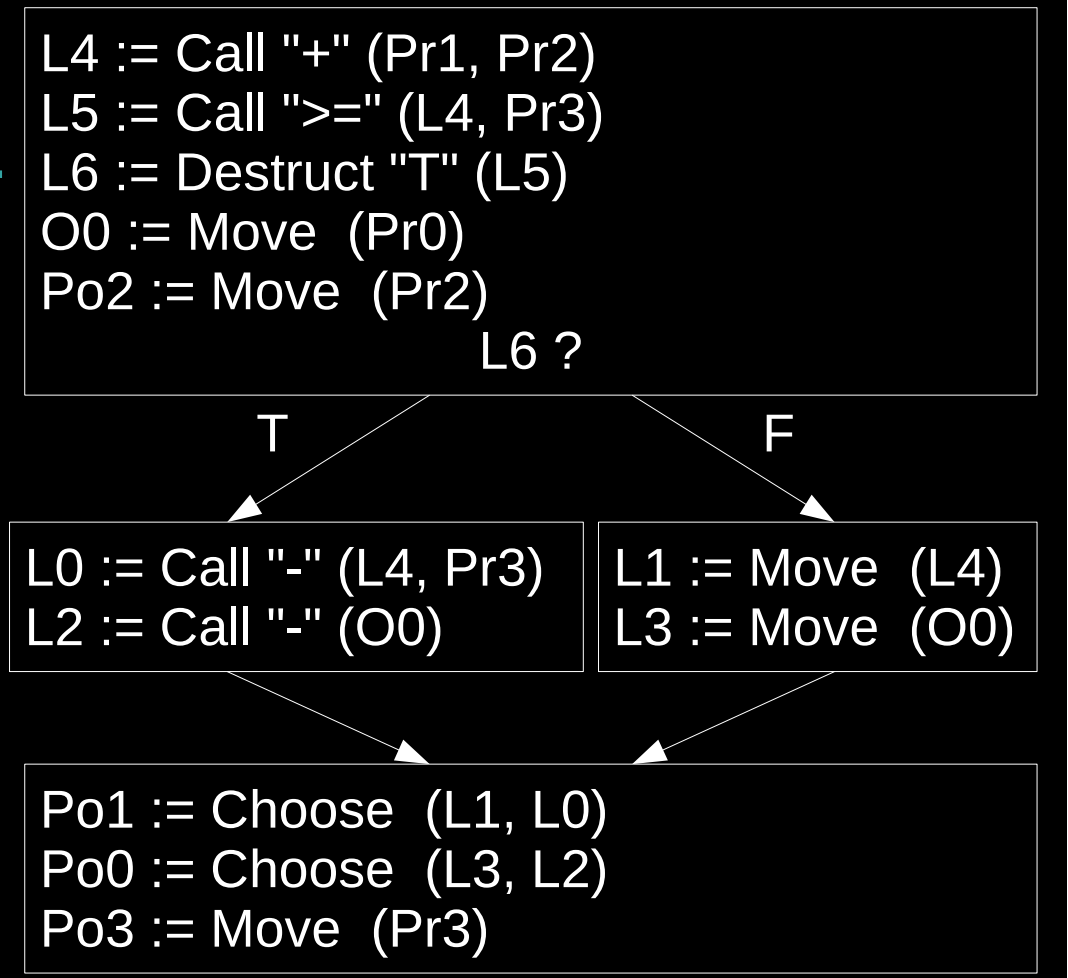
Evaluation
sah2-1

- sah2 := [
  x : { sample(real) | hold } ->
  y : real
  Where
  y := case x of {
  .sample(z) -> z
  .hold -> 0.0 ; y
  }
  ]



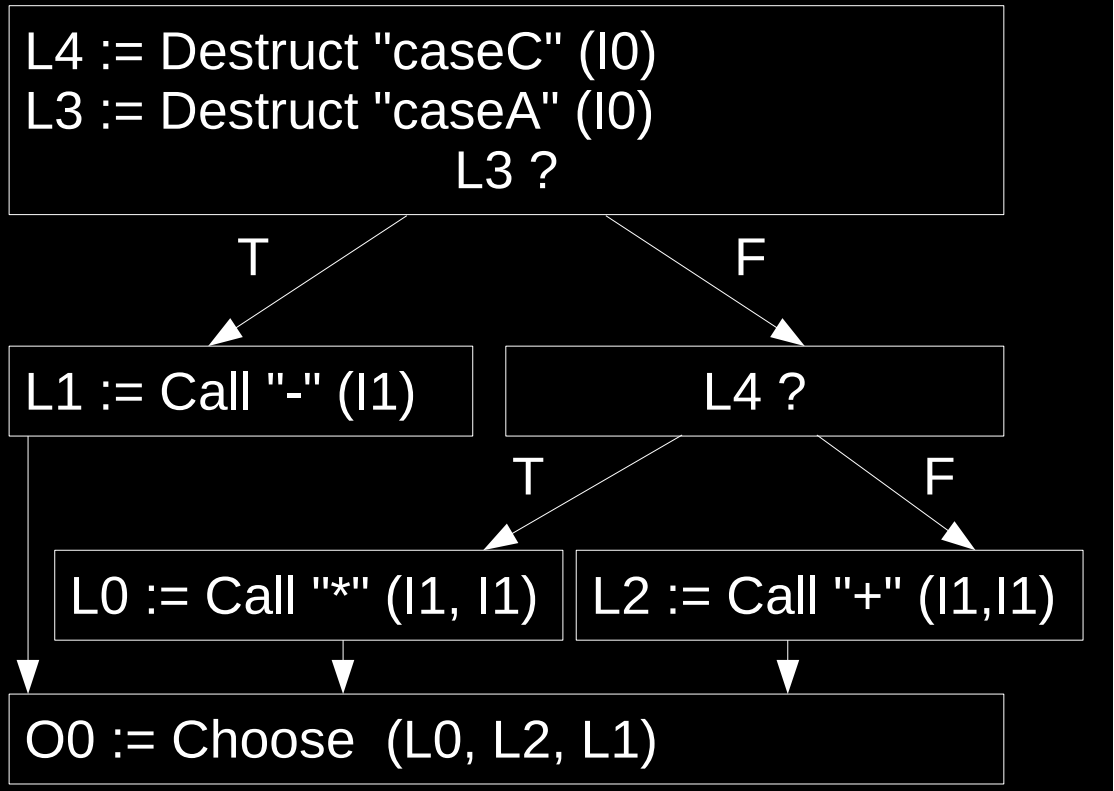
Evaluation rectangle

- rectangle := { eps : real ->
  #[-> out : real
  Where
  out', phase, phase', tmp : real
  over : bool
  out := 1.0 ; out'
  phase := 0.0 ; phase'
  tmp := phase + eps / 1.570796
  over := tmp >= 1.0
  out' := if over then -out else out
  phase' := if over then tmp - 1.0
  else tmp
  ] }



Evaluation
choose

- Choose := [
  t : { caseA | caseB | caseC }, x : real -> y : real
  Where
  y := case t of {
  .caseA -> -x
  .caseB -> x + x
  .caseC -> x * x
  }
  ]



Evaluation
Definiertheitsanalyse

- Verwendung Definiertheitsanalyse Pre-Variablen aus Vereinfachung von Bedingungen
- Bedingungen Ausführung Basic Blocks, Notwendigkeit Variablen / Statements richtig berechnet
- Beziehungen von Variablen nicht berücksichtigt
  → manche Tautologien nicht als Tautologie erkannt

Zusammenfassung

- aufeinander aufbauende Phasen – kein monolithischer Algorithmus
- Überspringen fallweise nicht benötigter Anweisungen (synthese Kontrollfluss)
- Vereinfachung der Sprungbedingungen
- Organisation der Sprünge
- Entfernung unerwünschter Effekte (DCE, CSE, Copy Propagation)
- Vorbereitung der nächsten Compilerphase
- Evaluation

Ausblick

- exklusive Variablenmengen Analyse Pre-Variablen $\perp$
- Einfluss Bedingungsvereinfachung (Entfernen von Variablen) auf frühere Phasen
- Organisation der Sprünge: gezielte Auswahl Bedingung Tautologie
- Regeln Vereinfachung boolescher Ausdrücke
  - Bedingungsvereinfachung Kontext nicht nur wahre Variablen
- statt CSE mächtigere Verfahren (z. B. PRE)
- Sprungbedingungen zerlegen, Sprünge tiefere Strukturen
- Zerlegung von Ausdrücken (CSE)
- Partial Dead Code Elimination

42

Weitere Literatur / Bilder

- EICAR-AV-Test: http://www.eicar.org/86-0-Intended-use.html , Stand: 14.11.2014
- Bild Folie 44: Werner Seiler
  https://commons.wikimedia.org/wiki/User:Centine
  gemeinfrei / public domain
  https://commons.wikimedia.org/wiki/File:AnthocharisCardamines_1606Z1.jpg?uselang=de

Danke für Ihre Aufmerksamkeit

Fragen? → Q&A

Werner Seiler, gemeinfrei / public domain