Assignment 2 Tutorial

Alessa Angerschmid

27. März 2020

Kommende Termine

- Finale Abgabe: 20.04., 10:00
- Abgabegespräche: 23.04. und 24.04.

→ Newsgroup verfolgen: `tu-graz.lv.soma`

Organisation

Gruppenanmeldung

- Die Gruppe von Assignment 1 bleibt bestehen
- Gitlab Projekt wird weiterverwendet

Organisation

- 20 Punkte pro Assignment
- Aufteilung in Theorie & Praxis
  $\rightarrow$ je 50%
- Unterlagen im **TeachCenter**
  - Aufgabenstellung
  - Framework (Praxis)
  - Templates für die Theorie (**müssen verwendet werden**)

Theoriebeispiele

Topics:

- Backward Slicing
- Forward Slicing
- Delta Debugging
  - Minimal Delta Debugging
  - Isolation Difference Delta Debugging

Theoriebeispiele

Aufgaben:

- Tasks zu jedem Topic
- Struktur lt. Template/Vorlesung/Skriptum
  - Templates (handschriftlich und LaTeX) befinden sich im TeachCenter
- CFG ist keine Pflicht (aber empfohlen)
- Falsche Struktur oder Unleserlichkeit führen zu Punkteabzügen!

Abgabe Theorieteil am 20.04.

- **Coversheet** verwenden
- Digitale Abgabe (submission2/group-XX-ass2.pdf)

Backward Slicing

- *def(n)* = alle Variablen, die definiert werden (z.B.: a = 3 + b;)
- *ref(n)* = alle Variablen, die referenziert werden

- *INFL(n)* = wenn Zeile n ein *if/while/for* enthält, werden alle eingeschlossenen Zeilen eingetragen
- *kill(n)* = $\{v|v \in def(n)\}$
- *gen(n)* = $\{v|v \in ref(n) \wedge ((def(n) \cap in(n) \neq \emptyset) \vee (inSlice(n) = T))\}$
  - Die Definitionsmenge und die in-Menge haben mind. ein gemeinsames Element
  - Die Zeile befindet sich bereits im Slice

- *gen(n)* = $\{v|v \in V \wedge n = i\}$ *für das Slicing Criterion C = (i, V)*
  - Die gen-Menge des Slicing Criterions enthält ebenfalls deren Variablen

Backward Slicing

- $in(n) = U_{p \in succ(c)}\ out(p)$
  - Vereinigungsmenge der out-Mengen aller Nachfolger von n
- $out(n) = gen(n) \cup (in(n) \setminus kill(n))$
  - Vereinigungsmenge aus allen generierten Variablen und der in-Menge ohne die Elemente der kill-Menge
- $inSlice(n) = T$
  - Statement, für welches der Slice berechnet werden soll
  - $def(n) \cup in(n) \neq \emptyset \vee (\exists m \in INFL(n) : inSlice(m) = T)$
    - Definitions-Menge und Input-Menge haben min. ein gemeinsames Element
    - Zeile n ist ein Kontrollfluss-Statement und ein Statement, welches im Influenzbereich von n, ist bereits im Slice
- $inSlice(n) = F$
  - In allen anderen Fällen

Forward Slicing

- *def(n)* = alle Variablen, die definiert werden
- *ref(n)* = alle Variablen, die referenziert werden

- *INFL(n)* = wenn Zeile n ein *if/while/for* enthält, werden alle eingeschlossenen Zeilen eingetragen
- $kill(n) = \{v | v \in def(n)\}$
- $gen(n) = \{v | v \in def(n) \wedge ((ref(n) \cap in(n) \neq \emptyset) \vee (inSlice(n) = T))\}$
  - Die Referenzmenge und die in-Menge haben mind. ein gemeinsames Element
  - Die Zeile befindet sich bereits im Slice

Forward Slicing

- $in(n) = U_{p \in pre(n)}\ out(p)$
  - Vereinigungsmenge der out-Mengen aller Vorgänger von n
- $out(n) = gen(n) \cup (in(n) \setminus kill(n))$
  - Vereinigungsmenge aus allen generierten Variablen und der in-Menge ohne die Elemente der kill-Menge
- $inSlice(n) = T$
  - Statement, für welches der Slice berechnet werden soll
  - $ref(n) \cap out(n) \neq \emptyset \vee (n \in INFL(m) : inSlice(m) = T)$
    - Referenz-Menge und Output-Menge haben min. ein gemeinsames Element
    - Zeile n befindet sich in der INFL-Menge einer anderen Zeile m, welche sich bereits im Slice befindet
- $inSlice(n) = F$
  - In allen anderen Fällen

Delta Debugging - Minimizing

- Initialisierung: $ddmin(c_f) = ddmin2(c_f, 2)$
- Algorithmus:

$$ddmin2(c_f, n) = \begin{cases} ddmin2(\Delta_i, 2) & \text{if } test(\Delta_i) = FAIL \\ ddmin2(\nabla_i, max(n-1, 2)) & \text{else if } test(\nabla_i) = FAIL \\ ddmin2(c_f, min(|c_f|, 2n)) & \text{else if } n < |c_f| \\ c_f & \text{otherwise} \end{cases}$$

- Abbruch: $|\Delta| = 1$

Delta Debugging - Isolation Difference

- Initialisierung: $dd(c_s, c_f) = dd2(c_s, c_f, 2)$
- Algorithmus:

$$dd2(c'_s, c'_f, n) = \begin{cases} dd2(c'_s, c'_s \cup \Delta_i, 2) & \text{if } \exists\{i | test(c'_s \cup \Delta_i) = FAIL\} \\ dd2(c'_f \setminus \Delta_i, c'_f, 2) & \text{else if } \exists\{i | test(c'_f \setminus \Delta_i) = PASS\} \\ dd2(c'_s \cup \Delta_i, c'_f, max(n-1, 2)) & \text{else if } \exists\{i | test(c'_s \cup \Delta_i) = PASS\} \\ dd2(c'_s, c'_f \setminus \Delta_i, max(n-1, 2)) & \text{else if } \exists\{i | test(c'_f \setminus \Delta_i) = FAIL\} \\ dd2(c'_s, c'_f, min(2n, |\Delta|)) & \text{else if } n < |\Delta| \\ (c'_s, c'_f) & \text{otherwise} \end{cases}$$

- Abbruch: $|\Delta| = 1$ bzw. $|c_f|$ - $|c_s| = 1$

Delta Debugging - String Splitting

Aufgaben:

- Zeichen so ausgeglichen wie möglich verteilen
- Delta mit niedrigerem Index zuerst ’füllen’

Beispiel:

- Input: `012345, n = 4` `0123456, n = 3`
- Output:

$\Delta_1 = 01$	$\Delta_1 = 012$
$\Delta_2 = 23$	$\Delta_2 = 34$
$\Delta_3 = 4$	$\Delta_3 = 56$
$\Delta_4 = 5$	

Programmierbeispiel

Flow Propagation Algorithm: Backward and Forward Slicing

Framework:

- Java Application $\rightarrow$ `Java 8`
- Input: Java Program inkl. Slicing Kriterium
- Framework parsed den Input und erstellt einen CFG (Assignment 1)
- `TODO`s implementieren

Programmierbeispiel

GitLab:

- Gruppen bleiben gleich $\rightarrow$ das bestehende Projekt kann weiter verwendet werden
- Neuer Unterordner `submission2`
- GitLab User `@soma_tutor` als Reporter hinzufügen
- 2 **sinnvolle** GitLab Issues erstellen (Label: BUG2)

Programmierbeispiel

TODOs: Backward Slicing

- Gegeben: Slicing Kriterium und CFG
  - CFG wie in Assignment1 inkl. neuer Funktionen:
    - `ArrayList<CFGNode> getPredecessors()`
    - `ArrayList<CFGNode> getSuccessors()`
- Task: den `bwdSlicingTable` füllen
  - Für jede Zeile/Node ein `FPANode` einfügen
  - Def-, Ref-, Gen-, Kill-, In- und Out-Menge berechnen
  - inSlice setzen

Programmierbeispiel

TODOs: Forward Slicing Kriterium

- Gegeben: die bereits berechneten Backward Slice Tables
- Task: aus den Backward Slices das Forward Slice Kriterium berechnen
  - Die Zeile, welche in den meisten Backward Slices vorkommt.
  - Wenn zwei Zeilen gleich oft vorkommen, soll die niedrigere verwendet werden.

1/31

Programmierbeispiel

TODOs: Forward Slicing

- Gegeben: CFG und das berechnete Kriterium
- Task: den `fwdSlicingTable` füllen
  - Für jede Zeile/Node ein `FPANode` einfügen
  - Def-, Ref-, Gen-, Kill-, In- und Out-Menge berechnen
  - inSlice setzen

Programmierbeispiel

Issues

- Neues Label: BUG2
- Bis zu **-10%** auf den Programmierteil, wenn nicht vorhanden/unvollständig
- Schritte:

1. Wie kann ich den Bug reproduzieren?
2. Was ist wirklich passiert?
3. Was hätte passieren sollen?

Programmierbeispiel

Testsystem:

- Wird kurz vor der Deadline gestartet (siehe `Newsgroup`)
- Testet in Intervallen
- Prozentuelle Auswertung der bestandenen Testcases
- Gruppenranking: **beste Gruppe gibt 100% vor**
  $\rightarrow$ Ersetzt eigenes Testen nicht!
- Bewertungsgrundlage:
  $\rightarrow$ 0/10 Punkten wenn die Public Testcases failen

Abgabe Programmierbeispiel am 20.04.

- Abgabe via GitLab Repository (Master branch)
- **Ordnerstruktur** und **Namenskonvention** einhalten
- Submission-Checkout um 10:00
  $\rightarrow$ Letzter Commit davor am Master Branch zählt
- Plagiate erhalten automatisch 0 Punkte!

Fragen?

- Jetzt ;)
- Discord Channel
- Newsgroup: `tu-graz.lv.soma`
- Fragestunde
- Email an `soma@ist.tugraz.at`