Visualisieren von Algorithmen: Dynamic/Relevant Slicing

Wie wird der eingelesene Code aussehen?

- Variablenzuweisungen
- Optionale begin/end-Blöcke
- If-Blöcke
- While-Blöcke

Beispielcode:	**2. Variante (nur Einrückungen):**	**3. Variante (geschwungene Klammern):**

1. begin
2. z = 0;
3. t = b;
4. **if** (x > b) **then**
5. **begin**
6. x = x - b;
7. **end**
8. **fi**
9. **while** (x <= t) **do**
10. **begin**
11. t = t - x;
12. z = t + 1;
13. **end**
14. **od**
15. write(z);
16. end
	1. z = 0;
	2. t = b;
	3. **if** (x > b)
	4. x = x - b;
	5. **while** (x <= t)
	6. t = t - x;
	7. z = t + 1;
	8. write(z);
		1. z = 0;
		2. t = b;
		3. **if** (x > b)
		4. {
		5. x = x - b;
		6. }
		7. **while** (x <= t)
		8. {
		9. t = t - x;
		10. z = t + 1;
		11. }
		12. write(z);

Wie sieht der Algorithmus für Dynamic Slicing aus?

Slicing Criterion hat folgende Informationen: C = (x, I^n, V)

- **x** stellt den Programminput (test case) dar
- **I** ist ein Statement im Execution Trace, **n** steht für den Index des Statements
- **V** enthält die Menge der relevanten Variablen

1. Berechne Execution trace (ET) für Testcases x
2. Zeichne gerichteten Graphen (Execution trace graph) mit
    a. Statements als Knoten
    b. Kanten sind die Abhängigkeiten
        i. Datenabhängigkeiten
        ii. Kontrollabhängigkeiten
        iii. Symmetrische Abhängigkeiten (nur bei Schleifen und Pfeile gehen in beide Richtungen)
3. Markiere Knoten m (m <= n) im Graphen
    a. Eine Variable v1 Element von V wird neu definiert
    b. Es gibt keinen Knoten i (m < i <= n), der v1 neu definiert
4. Rückwärts alle Kanten (Abhängigkeiten), die eingezeichnet wurden und entsprechende Knoten kennzeichnen
    a. Diese Knoten sind dann im Slice

Dynamic Slicing an einem Beispiel

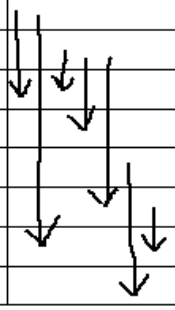
Slicing Criterion: ({a=1}. 14^8, {b})

```
1. begin
2.      read (a) ;
3.      i = 1 ;
4.      while (i <= a) do
5.              begin
6.              if (i mod 2 == 0) then
7.                      b = 17 ;
8.              else
9.                      b = 18 ;
10.             fi;
11.             i = i + 1 ;
12.             end
13.     od
14.     write ( b ) ;
15. end
```

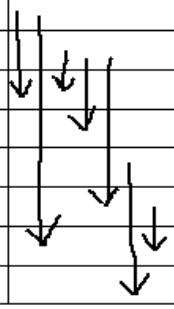
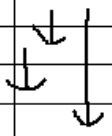
1) Berechne Execution Trace

ET	Data Dep	Con Dep	Symm Dep	In Slice
2^1 read(a) *a= 1*				
3^2 i = 1; *i = 1*				
4^3 while (i<=a) *1<= 1 → True*				
6^4 if (i mod 2 == 0) *i% 2 == 1 → False*				
9^5 b = 18;				
11^6 i = i+1; *i = 2*				
4^7 while (i<=a) *2<=1 → False*				
14^8 write(b);				

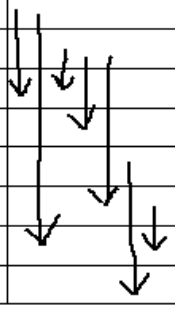
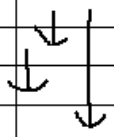
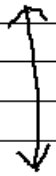
2) Zeichne gerichteten Graphen Datenabhängigkeiten
   a. Statements als Knoten
   b. Kanten sind die Abhängigkeiten
      i. **Datenabhängigkeiten**

ET	Data Dep	Con Dep	Symm Dep	In Slice
2^1 read(a) *a= 1*				
3^2 i = 1; *i = 1*				
4^3 while (i<=a) *1<= 1 → True*				
6^4 if (i mod 2 == 0) *i% 2 == 1 → False*				
9^5 b = 18;				
11^6 i = i+1; *i = 2*				
4^7 while (i<=a) *2<=1 → False*				
14^8 write(b);				

ii. Kontrollabhängigkeiten

ET	Data Dep	Con Dep	Symm Dep	In Slice
2^1 read(a) *a=1*				
3^2 i = 1; *i = 1*				
4^3 while (i<=a) *1<= 1 → True*				
6^4 if (i mod 2 == 0) *i% 2 == 1 → False*				
9^5 b = 18;				
11^6 i = i+1; *i = 2*				
4^7 while (i<=a) *2<=1 → False*				
14^8 write(b);				

iii. Symmetrische Abhängigkeiten

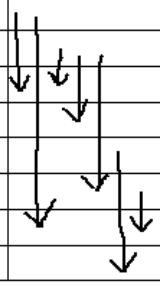
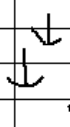
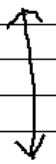
ET	Data Dep	Con Dep	Symm Dep	In Slice
2^1 read(a) *a=1*				
3^2 i = 1; *i = 1*				
4^3 while (i<=a) *1<= 1 → True*				
6^4 if (i mod 2 == 0) *i% 2 == 1 → False*				
9^5 b = 18;				
11^6 i = i+1; *i = 2*				
4^7 while (i<=a) *2<=1 → False*				
14^8 write(b);				

3) Markiere Knoten m (m <= n) im Graphen

Unser n ist 8. Wir suchen also das Statement, in dem die Variable b zuletzt geändert wurde. Das gesuchte m ist demnach 5.

4) Rückwärts alle Kanten (Abhängigkeiten), die eingezeichnet wurden und entsprechende Knoten kennzeichnen

   a. **Diese Knoten sind dann im Slice**

ET	Data Dep	Con Dep	Symm Dep	In Slice
2^1 read(a) *a=1*				X
3^2 i = 1; *i = 1*				X
4^3 while (i<=a) *1<= 1 → True*				X
6^4 if (i mod 2 == 0) *i% 2 == 1 → False*				X
9^5 b = 18;				X
11^6 i = i+1; *i = 2*				X
4^7 while (i<=a) *2<=1 → False*				
14^8 write(b);				

Slice = {2,3,4,6,9,11}

Bisher nur dynamic Slicing!

Wie funktioniert das Programm?

Das Programm wird eine Windows Forms Anwendung sein. Die Anwendung verfügt über einen Button, um ein Programm als Textdatei einzulesen. Die Textdatei kann so aussehen wie in Abschnitt „**Wie wird der Code aussehen?**“ beschrieben ist. Das Programm kommt mit den drei verschiedenen Formaten klar. Nun wird die Textdatei eingelesen. Falls die Datei ein fehlerhaftes Format aufweist, wird der Nutzer darauf aufmerksam gemacht. Das Programm teilt mit, dass nun nach dem Algorithmus gearbeitet wird.

Schritt 1 wird es sein, den Execution Trace (ET) zu zeichnen. Das Programm benötigt dafür vom Nutzer folgende Eingaben: C = (x, I^n, V). **x** stellt den Programminput dar, **I** ist ein Statement im Execution Trace, **n** ist der Index des Statements, **V** enthält die Menge der relevanten Variablen. Sobald der Nutzer die Eingaben getätigt hat, wird der ET berechnet. Das Programm findet selbstständig heraus, welche Statements nacheinander durchlaufen werden. Im Anschluss wird der ET in der linken Spalte einer Tabelle angezeigt. Neben den Programmcodezeilen enthält jede Zelle auch den aktuellen Wert der Variable, damit nachvollziehbar ist, wie sich der ET zusammengestellt hat. Optional können die ersten drei Zellen schrittweise gefüllt werden. Der Nutzer hat die Möglichkeit, den ersten Zellen dabei zuzusehen, wie sie gefüllt werden. Währenddessen werden hilfreiche Informationen in einer Textbox angezeigt (z.b. „i bekommt den Wert 1 zugewiesen.“ oder „Das if Statement if (i mod 2 == 0) evaluiert zu True, da i eine gerade Zahl ist.“). Die restlichen Zellen werden dann vom Programm gefüllt. Somit ist der Execution Trace angelegt.

Schritt 2 wird es sein, dass die Abhängigkeiten grafisch dargestellt werden. Zuerst werden die Datenabhängigkeiten eingezeichnet. Ein Pfeil zeigt dabei von einem Statement zum nächsten, in welchem die Variable aus ersterem Statement verwendet wird. Diese Regel wird dem Nutzer in einer Textbox angezeigt und der Pfeil wird eingeblendet. Auch hier bestimmt der Nutzer selbst, wann der nächste Pfeil eingezeichnet werden soll. Ab dem viertem Pfeil (falls das Programm mehr als drei Pfeile einzeichnen möchte) werden alle restlichen Pfeile hinzugefügt. Jetzt enthält die zweite Spalte der Tabelle alle Datenabhängigkeitspfeile. Das Gleiche passiert auch für die Kontrollabhängigkeiten und die Symmetrischen Abhängigkeiten. Am Ende von Schritt 2 sind die Spalten zwei bis vier mit Pfeilen ausgestattet, sofern es der Algorithmus für den gegebenen ET vorsieht. (Falls der Code keine while-Schleifen enthält, können auch keine Symmetrischen Abhängigkeiten erwartet werden.)

Schritt 3 des Algorithmus ist es, die Zelle zu finden, in welcher die relevanten Variablen zuletzt geändert wurden. Die Zelle(n) wird/werden dem Nutzer farblich hervorgehoben. Dem Nutzer wird mitgeteilt, dass laut Definition ein m gesucht wird, das kleiner/gleich n ist. Das n hat der Nutzer am Anfang selbst angegeben. Das m wird die Stelle sein, in welchem die Variable aus Statement n zuletzt geändert wurde. Falls es mehrere relevante Statements gibt, werden dem Nutzer nacheinander alle Statements hervorgehoben, die zuletzt den Wert einer relevanten Variablen geändert haben.

Schritt 4 ist der letzte Schritt für das dynamic Slicing. Die Statements mit Index m aus Schritt 3 haben nun Relevanz. Denn nun wird von ihnen ausgehend rückwärts die Abhängigkeitspfeile entlang gewandert. Dem Nutzer wird nacheinander jeder Abhängigkeitspfeil farblich hervorgehoben, damit er den Weg nachvollziehen kann. Schließlich setzt sich aus dem Weg der Slice zusammen. Sobald ein Pfeil farblich markiert wurde, wird die entsprechende Zelle in der fünften Spalte mit einem X gekennzeichnet. Auch hier empfiehlt es sich, die ersten Pfeile langsam abzugehen (den Nutzer klicken zu lassen) und erst ab dem vierten Pfeil wieder alle restlichen Pfeile direkt abzugehen. Schritt 4 endet damit, dass in der letzten Spalte alle Zellen mit einem X markiert wurden, deren Statement im Slice stehen. In einer Textbox werden außerdem alle Statements aufgeführt (beispielsweise Slice = {2,3,4,6,9,11}).