Wie ist das Visualisierungstool zu verwenden?

1) Einführende Worte:

Willkommen zum Visualisierungstool *Dynamic Slicing* der Lehrveranstaltung *Software Maintenance*.

Dieses Tool soll Ihnen dabei helfen, den Algorithmus des Dynamic Slice zu verstehen und anwenden zu können. Das Programm funktioniert auf den Plattformen Windows und Linux (Ubuntu). Unter Windows können Sie das Programm ohne weitere Vorbereitungen starten und zum Punkt 3) vorspringen. Für Linux bitte ich Sie, zuerst Punkt 2) zu behandeln. MacOS funktioniert derzeit leider nicht, da Mono auf Carbon läuft und keine 64bit unterstützt (https://www.mono-project.com/docs/about-mono/supported-platforms/macos/#32-and-64-bit-support).

2) Voreinstellungen unter Linux

Da unter Linux per Default keine .Net Umgebung besteht, ist es von Nöten, eine solche Umgebung mit Hilfe von Mono zu erstellen.

Orientiert wurde sich an folgendem Link:

https://www.mono-project.com/download/stable/#download-lin-ubuntu

Schritt 1)

Unter Ubuntu 20.04:

```
sudo apt install gnupg ca-certificates
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
echo "deb https://download.mono-project.com/repo/ubuntu stable-focal main"
| sudo tee /etc/apt/sources.list.d/mono-official-stable.list
sudo apt update
```

Unter Ubuntu 18.04:

```
sudo apt install gnupg ca-certificates
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
echo "deb https://download.mono-project.com/repo/ubuntu stable-bionic main"
| sudo tee /etc/apt/sources.list.d/mono-official-stable.list
sudo apt update
```

Schritt 2)

Installiere Mono mit:

```
sudo apt install mono-devel
```

Schritt 3)

Installiere Toolkit für Grafikoberfläche:

```
sudo apt install libcanberra-gtk-module libcanberra-gtk3-module
```

Nun lässt sich das Programm mit dem Kommando "mono DynamicSlicing.exe" ausführen.

3) Programmaufbau

Das Programm teilt sich in drei Bereiche ein:

- Code auswählen

- Eingabeparameter setzen

- Algorithmus ausführen

Zunächst erwartet das Programm einen Eingabecode. Diesen wählen Sie mit Klick auf den Button *Pick Source Code* aus. Sie können den Code händisch eingeben, einen der Codebeispiele auswählen oder eine Source-Datei einlesen. Beachten Sie dabei die Syntax, die in den Beispielcodes verwendet wurde. Mit Klick auf den Button *Formate Code* wird überprüft, ob der ausgewählte Code valide ist. Wenn er es ist, können Sie mit Klick auf *Choose Code* das Fenster schließen.

Für den Fall, dass der Code Werte für Variablen erwartet, die schon vorher gesetzt werden müssen, erscheint das Textfeld für den *test case* rot hinterlegt. Sie müssen nun für die jeweiligen Variablen ganzzahlige Werte einsetzen. Achten Sie auf das Format z.B.: "x=3,b=2". Falls Sie nicht wissen, welche Variablen Werte erwarten, entfernen Sie einfach den Inhalt im Textfeld und die benötigten Variablen werden wieder dargestellt. Wenn Sie schon jetzt wissen, für welche Variablen Sie sich am Ende interessieren, können Sie diese im Textfeld zu *Variables of interest* eintragen.

Als nächstes können Sie den Dynamic Slice Algorithmus auf zwei Weisen ausführen: Entweder mit oder ohne Zwischenschritte. Wenn Sie Zwischenschritte wünschen, aktivieren Sie einfach die CheckBox *intermediate steps*, bevor Sie den Button *Dynamic Slicing* betätigen. Ohne Zwischenschritte wird der Algorithmus direkt bis zum Ende ausgeführt. Mit Zwischenschritten können Sie schrittweise mit Klick auf den Button *Next Step* durch den Algorithmus wandern und erhalten zu jedem Schritt eine Erklärung (siehe Punkt 4). Mit Klick auf den Button *Skip to end* können Sie direkt zum Ende springen und erhalten dabei auch den Verlauf, den der Algorithmus genommen hat.

Sobald der Algorithmus terminiert, werden die Pfeile aller Abhängigkeiten und der berechnete *Slice* angezeigt. Per Default interessiert sich das Programm für die letzte Zeile im Code (der Wert im Textfeld *n^th element of trajectory* stellt die letzte Zeile dar). Man sieht nun, welche Zeilen im Originalcode für das Endergebnis in der letzten Zeile

verantwortlich sind. Falls das Ergebnis bei den *Variables of interest* nicht stimmt, muss man lediglich die Zeilen im Slice betrachten, um den Fehler zu korrigieren.

Nachträglich kann man den Wert des Textfeldes *n^th element of trajectory* ändern, wenn man sich doch für den *Slice* einer anderen Zeile in dem Algorithmus interessiert.

4) Erläuterung der Begriffe in der *History*:

Begriff	Erläuterung
x gets value n	Variable x erhält einen neuen Wert n.
evaluates to	Ein Vergleich testet, ob ein *If-Statment* oder ein *While-Statment* zu wahr oder falsch evaluiert.
write	Ruft lediglich die Methode *write()* auf.
ET finished	Der *Execution Trace* wurde fertig erstellt, es kann mit den Abhängigkeiten fortgesetzt werden.
x datadependency to y	Zeile x bekommt einen Wert, der in Zeile y verwendet wird.
datadependencies finished	Die Datenabhängigkeiten wurden berechnet. Die Kontrollabhängigkeiten können folgen.
x control dependency to y	In Zeile x ist ein Schleifenaufruf oder ein If-Statement. Die Zeile y liegt innerhalb und wird nur ausgeführt, wenn Zeile x zu true wird.
Control finished	Die Kontrollabhängigkeiten wurden berechnet. Die symmetrischen Abhängigkeiten können folgen.
x symmetric dependency to y	Die Schleife aus Zeile x wird in Zeile y erneut aufgerufen.
Symmetric finshed	Die symmetrischen Abhängigkeiten wurden berechnet. Der Slice kann berechnet werden.
slice from y to x	Es wird von unten nach oben berechnet. Zeile y hat eine Verbindung/Abhängigkeit zu Zeile x.
Slicing finished	Der Slice wurde fertig berechnet.
Dynamic Slicing finished	Der Algorithmus ist terminiert.

5) Erklärung des Algorithmus

Der Sinn des Algorithmus ist es, für einen gegebenen Code all die Zeilen herauszufinden, die dazu führen, dass in einer bestimmten Zeile X ein Ergebnis zustande kommt. Falls das Ergebnis nämlich nicht das gewünschte ist, muss man lediglich die relevanten Zeilen in Betracht ziehen.

Zuerst wird der Code herangezogen und von oben bis unten durchiteriert. Falls nötig, wird ein *Test case* mitgegeben, der jeder Variable einen Wert zuweist, welche verwendet werden, ohne dass sie im Code einen zugewiesen bekommen haben. Der *Execution Trace* beschreibt das Durchiterieren, wobei bei der Notation *n^m* das n für den Zeilenindex im Code und das m für den Iterationsindex steht. Sobald der *Execution Trace* geendet hat, weil der Code terminiert ist, werden die jeweiligen Abhängigkeiten bestimmt. Man arbeitet dabei immer von oben nach unten. Zuerst wird bei den Datenabhängigkeiten geschaut, in welcher Zeile eine Variable definiert und wann sie referenziert (verwendet) wird. Diese Verbindung/Abhängigkeit verdeutlicht man durch einen gerichteten Pfeil. Wenn alle Datenabhängigkeiten bestimmt wurden, sind als nächstes die Kontrollabhängkeiten dran. Wenn eine Zeile ein *If-Statement*, *For-Statement* oder *While-Statement* beschreibt, dann hat es zu jeder Zeile, die unmittelbar eine Ebene tiefer liegt, eine Kontrollabhängigkeit. Die tiefere Ebene wird nur dann ausgeführt, wenn das jeweilige Kontrollstatement mit *true* evaluiert. Zuletzt sind mit symmetrischen Abhängigkeiten alle Verbindungen von *for*- und *while-Schleifen* gemeint, die mehrmals aufgerufen werden. Auch sie werden miteinander verbunden. Abschließend wird für die Berechnung des Slices von unten nach oben gearbeitet. Ausgehend von der interessanten Zeile X werden die Pfeile entgegengesetzt verfolgt. Alle Zeilen, die damit erreicht werden, liegen nachher im Slice und sind an dem Ergebnis in Zeile X beteiligt.