

Von der Realität zur Virtualität und zurück am Beispiel der Programmanalyse

Andrea Flexeder

Technische Universität München, Boltzmannstrasse 3, 85748 Garching, Germany,
flexeder@cs.tum.edu,
WWW home page: <http://www2.cs.tum.edu/~flexeder>

Zusammenfassung In diesem Dokument zum Themenbereich **Modell und Wirklichkeit** wird das Vorgehen **Von der Realität zur Virtualität und zurück** am Beispiel der Programmanalyse dargestellt. Mit einem Einführungsbeispiel wird die Notwendigkeit der Programmanalyse illustriert. Wie das Beispiel erkennen lässt, sind Fehler in komplexen Computersystemen unvermeidlich, die es herauszufinden und zu beseitigen gilt. Da die nicht trivialen Programmeigenschaften nicht auf Anhieb erkennbar und in der **Realität** nicht beweisbar sind, muss eine Überführung der Problemstellung in die **Virtualität** vorgenommen werden. Durch diese Abstraktion kann die Frage nach bestimmten Programmeigenschaften in der Virtualität gelöst werden. Eine Auswertung des berechneten abstrakten Ergebnisses ermöglicht eine Aussage zu den Programmeigenschaften in der Realität. Das schrittweise Vorgehen und mögliche Anwendungsgebiete werden nachfolgend beschrieben.

1 Beispiel: Patriot

Die **Patriot Rakete** versagte 1991 in ihrer Aufgabe eine irakische Scud-Rakete abzufangen. Die Scud-Rakete traf eine amerikanische Kaserne in Saudi-Arabien und forderte 28 Menschenleben. Dieses Versagen war durch einen **Software-Fehler** im Waffenkontrollcomputer bedingt.

Die Zeitangabe der internen Uhr des Steuercomputers erfolgte in **Zehntel-Sekunden**, wohingegen das Steuerprogramm in **Sekunden** rechnete. Um mit diesen unterschiedlichen Zeitangabeformaten umzugehen, war eine Umrechnung notwendig, was durch eine Multiplikation mit 0.1 erreicht werden sollte. Die Zahl 0.1 ist im Rechner nicht exakt darstellbar. Es ergibt sich ein Rundungsfehler, so dass die Zeit seit Systemstart nur ungenau und somit fehlerhaft berechnet werden konnte. Je länger die Patriot Rakete in Betrieb war, um so mehr summierte sich dieser Fehler auf und führte zu falschen Berechnungen.

Zu dem Einsatzzeitpunkt der Rakete belief sich die Betriebsdauer des Patriot Systems bereits auf 100 Betriebsstunden, was zu einem Fehler von 0.34 Sekunden zwischen vergangener und berechneter Zeit führte. Die Scud Rakete fliegt in 0.34 Sekunden etwa 0.57 km weiter. Die Folge war, dass sich die Scud Rakete schon ausserhalb der Reichweite des Aufspürsystems der Patriot Rakete befand und deshalb nicht von ihr erfasst werden konnte.¹

¹ The Patriot Missile Failure, <http://www.ima.umn.edu/~arnold/disasters/patriot.html>

Weitere Informationen zu Software Bugs sind im Vortrag zu Software Bugs zu finden.
2

Dieses Beispiel soll die verheerenden Folgen verdeutlichen, die Fehler in Softwaresystemen mit sich bringen. Unsere Zielsetzung ist es, möglichst alle Fehler, die ein Programm enthält, zu erkennen und zu beseitigen.

2 Ziel: Fehlererkennung

Um die Fehlerfreiheit von Programmen zu gewährleisten, sollen kurz zwei Methoden für die Erkennung von Fehlern angesprochen werden.

– Testen:

Beim Testen wird das Programm nacheinander mit verschiedenen Eingabewerten ausgeführt und die daraus resultierenden Ergebniswerte werden beobachtet.

Das Problem hierbei besteht darin, dass es unendlich viele Eingabewerte gibt, dieses Verfahren sehr zeitaufwendig ist und per Hand durchgeführt werden muss.

– Verifikation:

Aufgrund der Komplexität moderner Computersysteme ist es oftmals nicht mehr möglich einen formalen Beweis für die Korrektheit eines Programms zu führen.

Um dennoch bestimmte Programmeigenschaften vollautomatisch beweisen zu können, verwendet man das nachfolgend beschriebene Mittel der **abstrakten Interpretation**.

3 Abstrakte Interpretation

Der Begriff der **abstrakten Interpretation** wurde von **Patrick Cousot** geprägt [?], der Vorreiter auf dem Gebiet der Programmanalyse, der die Theorie zum Nachweis der Korrektheit der Programmanalysen lieferte [?].

Mit Hilfe der **abstrakten Interpretation** versucht man durch die Betrachtung von Teilaspekten eines Programms Aussagen über das Programm zu treffen, so dass man z.B. bestimmte Fehler von vornherein ausschliessen kann.

Im folgenden werden die einzelnen Schritte beschrieben, die für die Bildung eines abstrakten Modells und die Auswertung des abstrakten Ergebnisses durchzuführen sind.

1. Konkrete Fragestellung:

Durch die Formulierung einer konkreten Fragestellung werden nur Teilaspekte des Programmes betrachtet und alle für diese spezielle Fragestellung unwichtigen Aspekte weggelassen.

2. Erstellung eines abstrakten Modells:

Abhängig von der Fragestellung ist ein abstraktes Modell des zu analysierenden Programmes zu erstellen, um mit Hilfe dieses Modells eine Lösung auf die Frage zu erhalten, die in der Realität nicht beantwortbar ist.

² A Collection of Software Bugs, <http://www5.in.tum.de/~huckle/bugse.html>

3. **Abstrakte Interpretation der Programmanweisung:**

Jede Programmanweisung benötigt eine abstrakte Interpretation, so dass sie im abstrakten Modell ausgeführt werden kann. Jede abstrakte Betrachtung und Interpretation von Programmanweisungen führt zu einem Effekt auf das abstrakte Modell.

4. **Rückschluss auf die konkrete Frage:**

Schliesslich muss das abstrakte Modell noch interpretiert werden, um Rückschlüsse auf die konkrete Fragestellung zu erhalten. Unter Verwendung der abstrakten Interpretation ist es möglich Aussagen über bestimmte Programmeigenschaften zu machen.

Die Schwierigkeit bei der **abstrakten Interpretation** besteht in der Modellwahl, so dass so viele Aspekte in das abstrahierte Modell zu übernehmen sind, um einen genügend *präzisen* abstrakten Wertebereich zu erhalten, und so viele Teilaspekte wegzulassen, so dass das Modell *berechenbar* wird.

4 **Beziehung Modell und Wirklichkeit**

Die nachfolgende Abbildung soll die Beziehung zwischen Modell und Wirklichkeit anschaulich illustrieren.

– von der Realität zur Virtualität:

Das Beispielprogramm verdeutlicht das konkrete Modell in der **Realität**, wo eine konkrete Programmanweisung z.B. durch Speicherzugriffe beschrieben wird. Wie bereits erwähnt ist es unmöglich eine **Berechnungsvorschrift** δ anzugeben, die es gestattet bereits auf der Ebene der Realität eine bestimmte Programmeigenschaft zu beweisen.

Deshalb ist ein Umweg über die Ebene der **Abstraktion** unumgänglich. Dazu ist eine **Abstraktionsfunktion** α zu bilden, die jeden Wert auf den zugehörigen abstrakten Wert abbildet. In der nachfolgend betrachteten Programmanalyse wird α mit Hilfe von affinen Abbildungen modelliert. In der Virtualität muss ebenso eine **Berechnungsvorschrift** $\delta^\#$ angegeben werden, um eine abstrakte Interpretation der einzelnen Programmanweisungen zu handhaben. Das abstrakte Modell wird hierbei durch geometrische Objekte (*konvexe Mengen*) beschrieben. $\delta^\#$ führt dazu, dass durch die Auswertung von Programmcode ein geometrisches Gebilde in ein anderes geometrisches Objekt überführt wird.

– von der Virtualität zurück zur Realität:

Schliesslich muss das abstrakte Modell wieder auf die Realität abgebildet werden, was die **Konkretisierungsfunktion** α' übernimmt, welche dem abstrakten Wert alle entsprechenden konkreten Werte zuordnet.

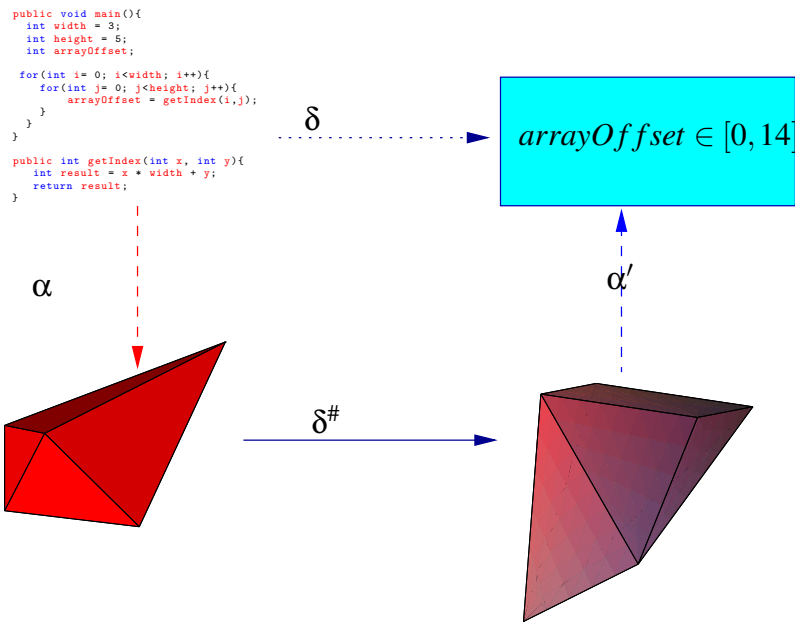


Abbildung 1. Modell und Wirklichkeit

Durch dieses eben beschriebene Vorgehen - Abstraktion des konkreten Modells - Berechnung in der Virtualität - Deutung des Ergebnisses für die Realität - können Aussagen zu Programmeigenschaften gemacht werden.

5 Programmanalyse

In diesem Abschnitt wird die Programmanalyse anhand der **Interprozeduralen Analyse linearer Ungleichungen**[?] genauer beschrieben . Eine Durchführung dieser Analyse resultiert in Aussagen über die linearen Beziehungen zwischen den Programmvariablen [?].

5.1 Konkrete Fragestellung

An jedem Programmpunkt möchte man wissen, welche linearen Ungleichheitsbeziehungen zwischen den Programmvariablen gelten.

```

1   public void main() {
      int width = 3;
      int height = 5;
      int arrayOffset;

```

```

6      for(int i= 0; i<width; i++){
          for(int j= 0; j<height; j++){
              arrayOffset = getIndex(i, j);
          }
11     }

      public int getIndex(int x, int y){
          int result = x * width + y;
          return result;
16     }

```

Abbildung 2. Beispielprogramm

Die Programmvariable *arrayOffset* in diesem Beispielprogramm stellt eine Array-Indizierungsvariable dar. Es ist interessant zu wissen, welche Werte diese Programmvariable *arrayOffset* annehmen kann, um eine falsche Arrayindizierung ausschliessen zu können.

5.2 Abstraktion der Programmezustände

Jedem Programmpunkt wird eine **konvexe Menge** zugeordnet, die sich als eine endliche Menge von Punkten, Strahlen und Linien repräsentieren lässt. Die konvexen Mengen beschreiben einen Teilbereich eines *n-dimensionalen Vektorraumes*, wobei jede Programmvariable einer Dimension des Vektorraumes entspricht. Lineare Abhängigkeiten zwischen den Programmvariablen können mit Hilfe von konvexen Mengen ausgedrückt werden. Eine konvexe Menge repräsentiert dabei alle möglichen Variablenbelegungen. Es kann nur der Wertebereich in dem Programm erreicht werden, den die konvexe Menge für das Programm beschreibt. Alle anderen Werte können in dem zu analysierenden Programm nicht auftreten.

5.3 Abstrakte Interpretation der Programmanweisungen

Jede Programmanweisung ist abstrakt zu interpretieren. Dabei muss für jeden konkreten Wert eine Abstraktion gefunden werden, die diesen konkreten Wert beschreibt und Rückschlüsse auf das Programmverhalten ermöglicht. Jede Interpretation einer Programmanweisung verändert die einen Programmezustand beschreibende konvexe Menge, so dass ein neues konvexes Objekt entsteht, das den nach Ausführung der Programmanweisung gültigen Programmezustand beschreibt [?].

5.4 Berechnungsverfahren

Es ist das kleinste abstrakte Modell zu berechnen, welches eine allgemeine Lösung für alle möglichen Programmabläufe darstellt. Die Berechnung dieses kleinsten Modells kann mit Hilfe eines **Fixpunktalgorithmus** durchgeführt werden. Damit das Verfahren des Fixpunktalgorithmus auch nach endlich vielen Schritten terminiert, kommt die

Methode des **Widenings** zum Einsatz. Widening gewährleistet zwar die Terminierung des Berechnungsverfahrens, führt aber durch eine Überapproximation des kleinsten abstrakten Modells zu einem Genauigkeitsverlust.

Eine detailliertere Beschreibung der Berechnungsvorschrift des Fixpunktalgorithmus und der Technik des Widenings, kann unter [?] nachgelesen werden.

5.5 Auswertung des Modells

Nach Anwendung des Fixpunktalgorithmus zusammen mit der Widening-Technik erhält man für das zuvor aufgeführte Beispielprogramm eine konvexe Menge, bestehend aus 4 Punkten, welche das in der folgenden Abbildung dargelegte geometrische Objekt beschreibt.

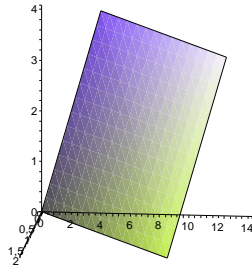


Abbildung 3. Konvexe Menge zum Beispielprogramm

Die Interpretation dieses konvexen Objektes führt zu dem Ergebnis, dass für die Programmvariable *arrayOffset* die folgende Ungleichungsbeziehung gilt:

$$0 \leq \text{arrayOffset} \leq 14;$$

6 Anwendungsbereiche

Viele Bereiche im **Compilerbau** stützen sich auf Programmanalysen ab, um Informationen über die Wertebereiche der Variablen an den jeweiligen Programmpunkten auszunutzen. Ein möglicher Einsatzbereich wäre **Array Bound Checking**. Bei der Programmiersprache Java zum Beispiel wird beim Compilervorgang bei jedem Arrayzugriff überprüft ob sich der Wert der Indexvariable im Intervall $[0, \text{arrayLength} - 1]$ befindet. Liefert eine vorgeschaltete Programmanalyse, dass sich die Arrayindizierungsvariable immer nur innerhalb des geforderten Intervalles befindet, so kann diese Abprüfung bei jedem Arrayzugriff ganz ausgespart werden. Weitere Anwendungen sind

die Codeerreichbarkeit, das Erkennen uninitialized Variablen, die Terminierung von Schleifen, usw.³

Die Programmanalyse findet auch Anwendung auf dem Gebiet der **Programmverifikation**, da mit ihrer Hilfe gültige Invarianten [?] angegeben werden können, die aus dem Programm nicht sofort erkennbar sind. Der Beweis der Korrektheit eines Programmes kann somit durch die aus der Programmanalyse gewonnenen Relationen erheblich vereinfacht werden.

7 Zusammenfassung

Da es unmöglich ist ab einem bestimmten Komplexitätsgrad korrekte Programme zu schreiben, kommt die Programmanalyse zum Einsatz. Der konkrete Datenraum muss abstrahiert werden, so dass in der Virtualität auf den abstrakten Werten die Berechnungen durchgeführt werden. Durch die Berechnung eines kleinsten Modells auf der Abstraktionsebene erfolgt eine oberste Approximation des konkreten Wertebereiches. Dieser sich ergebende abstrakte Wertebereich muss nun wieder interpretiert werden, so dass in der Realität Aussagen über die Programmeigenschaften möglich sind. Durch die Kenntnis der so berechneten Programmeigenschaften können viele Fehler erkannt und reduziert werden, so dass z.B. eine Referenzierung eines Arraybereiches ausserhalb der Arraygrenze ausgeschlossen werden kann.

Literatur

- CC77. Patrick Cousot and Radhia Cousot. Acm symposium on principles of programming languages. pages 238–252. ACM Press, 1977.
- CH78. Patrick Cousot and Nicholas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th Annual ACM Symposium on Principles of Programming Languages (POPL) 1978*, pages 84–97, 1978.
- Fle05. Andrea Flexeder. Interprozedurale Analyse linearer Ungleichungen. Master's thesis, Technische Universität München, München, 2005.
- MOS04. Markus Müller-Olm and Helmut Seidl. Program analysis through linear algebra. In *31th annual ACM Symposium on Principles of Programming Languages (POPL) 2004*, pages 330–341, 2004.
- Pet04. Michael Petter. Berechnung von polynomiellen Invarianten. Master's thesis, Technische Universität München, München, 2004.

³ Programoptimierung