22 Reverse-Engineering durch Identifikation von Eingabedaten-Äquivalenzklassen aus Programmabläufen

Rainer Schmidberger
Abteilung Software-Engineering, Institut für Softwaretechnologie, Universität Stuttgart
www.iste.uni-stuttgart.de/se

Kurzfassung

Die Programmlogik betriebswirtschaftlicher Software-Systeme implementiert häufig Entscheidungstabellen, die durch eine sehr direkte Verwendung der Eingabedaten als Bedingungskriterien geprägt sind. Speziell mit COBOL implementierte Systeme, die häufig seit vielen Jahren im Einsatz sind, zählen zu dieser Gruppe. Die ursprünglich zugrunde liegenden Entscheidungstabellen sind aber oft weder erhalten noch gepflegt worden. Zur wirtschaftlichen Wartung oder Neu-Implementierung sind diese Tabellen jedoch eine wichtige Voraussetzung.

Dieser Artikel beschreibt ein Verfahren, wie bei solchen Systemen, für die Programmcode sowie große Mengen von Eingabedaten verfügbar sind, die Entscheidungstabellen über Eingabedaten-Äquivalenz-klassen ermittelt werden können.

22.1 Einführung

Die funktionale Beschreibung eines Programms ist die Zuordnung von Ausgabedaten zu allen möglichen Eingabedaten. Sie kann daher prinzipiell aus einer Implementierung ermittelt werden, indem alle möglichen Eingaben bearbeitet und die Ausgaben gesammelt werden. Die Ausgabedaten definieren Äquivalenzklassen der Eingabedaten; diese können in Form einer Tabelle – der Entscheidungstabelle – dargestellt werden.

Praktisch ist dieser Weg in der Regel nicht gangbar, weil die Menge der Eingabedaten weit größer ist, als über beherrschbare Tabellen darzustellen wäre. Selbst bei einem kleinen Programm mit nur wenigen Integer-Eingaben gibt es 10^{20} Einträge und mehr.

Bei dem in der Folge beschriebenen Verfahren zur Ableitung der Entscheidungstabelle für Systeme mit vielen Eingabedaten werden anstelle aller möglichen Eingabedaten die folgenden zwei Informationsquellen verwendet:

1. der Programmcode der untersuchten Software und hier speziell die Verzweigungen und deren Bedingungsterme
2. die existierenden Eingabedaten des regulären Betriebs

Offensichtlich stellen die existierenden Eingabedaten des regulären Betriebs nur einen Bruchteil der theoretisch vollständigen Eingabedatenmenge dar; innerhalb der Menge der fachlich sinnvollen Eingabedaten ist ihr Anteil aber typischerweise relativ hoch.

22.2 Definitionen

Sei d_i ein vollständiger Eingabedatensatz für das Programm P. Mit dem Datensatz d_i durchläuft P deterministisch einen ganz bestimmten Programmpfad, den Pfad p_i. Der Pfad ist die vollständige Sequenz der ausgeführten Anweisungen inklusive Wiederholungen usw. Wir bilden nun Äquivalenzklassen von Eingabedatensätzen. Zwei Eingabedatensätze d_i und d_j sind schwach äquivalent, wenn die beiden zugeordneten Pfade p_i und p_j gleich sind.

Jeder Pfad ist bestimmt durch den Eintrittspunkt in das Programm und durch die Sequenz von Entscheidungen, die an den Verzweigungen getroffen werden. Eine Entscheidung an einer Programmverzweigung besteht, wie von Frühauf, Ludewig und Sandmayr beschrieben, aus einem oder mehreren Termen [1].

Wir gehen hier davon aus, dass jedes Programm genau einen Eintrittspunkt hat und dass Verzweigungen die Form von if- oder case-Anweisungen haben. Schleifen werden noch gesondert betrachtet.

Sei beispielsweise ein kurzes Programm (COBOL-Syntax)

```
0010 IF A >= 5 AND <=9
0020   STATEMENT1
0030 ELSE
0040   STATEMENT2
0050 END-IF
```

Wenn A den Wert 2 hat, liefert uns die Ausführung des Programms einen Pfad (`0010`, `0040`), die Sequenz von Termen, die den Pfad bestimmen und außerdem den Hinweis, dass es zu diesem Term weitere Pfade, mindestens einen weiteren Pfad, für den Fall gibt, dass die auf A angewendeten Terme ein anderes Resultat liefern. Da wir zunächst nicht wissen, wie viele Pfade sich hier verbergen, sprechen wir unbestimmt von einem „dunklen Pfad“.

Zwei schwach äquivalente Eingabedatensätze d_i und d_j sind stark äquivalent, wenn alle angewendeten Terme das gleiche Resultat ergeben, wenn also die annotierten, d.h. um die ausgewerteten Terme erweiterten Pfade, gleich sind. Im Beispiel oben wäre ein Datensatz dem ersten stark äquivalent, wenn $A = 4$ enthielte, denn der annotierte Pfad ist in beiden Fällen `0010` ($A \geq 5 \rightarrow$ `FALSE`, $A \leq 9 \rightarrow$ `TRUE`), `0040`. Dagegen wäre $A = 10$ nur schwach äquivalent, der annotierte Pfad ist `0010` ($A \geq 5 \rightarrow$ `TRUE`, $A \leq 9 \rightarrow$ `FALSE`), `0040`.

22.3 Verfahren zur Ermittlung von Eingabedaten-Äquivalenzklassen

Die These dieser Arbeit ist, dass stark äquivalente Eingabeklassen jeweils einem Fall in einer Entscheidungstabelle entsprechen. Darum werden viele, womöglich alle verfügbaren Eingabedatensätze verarbeitet und die annotierten Pfade verglichen. Wenn ein neuer annotierter Pfad entdeckt wird, kann er mit den Variablenwerten, die in den Termen verwendet wurden, als neuer Fall gespeichert werden.

Wenn noch kein einziger Datensatz angewendet wurde, stellt das gesamte Programm einen dunklen Pfad dar. Jede Ausführung (d.h. jeder Programmlauf mit einem weiteren Datensatz) erweist sich entweder als stark äquivalent zu einer früheren Ausführung oder beseitigt einen dunklen Pfad, indem sie einen neuen annotierten Pfad und im der Regel auch neue dunkle Pfade liefert.

In der Praxis ist natürlich nicht damit zu rechnen, dass am Ende alle dunklen Pfade beseitigt sind. Für die verbliebenen dunklen Pfade können unterschiedliche Ursachen vorliegen:

1. Der gefundene dunkle Pfad ist fachlich durchaus sinnvoll und gewollt. Es war nur kein entsprechender Eingabedatensatz angewendet worden.
2. Der gefundene dunkle Pfad ist bedeutungslos geworden, weil er eine Ausnahmebehandlung für Eingabedatensätze darstellt, die so nicht mehr existieren (weil es diese Eingabedatensätze z.B. nur zeitlich befristet gab)
3. Es handelt sich um eine Fehlerbehandlung für einen nicht aufgetretenen Fehler
4. Es handelt sich beim dunklen Pfad um Programmcode, der für sinnvolle Eingabedaten nicht erreichbar ist und damit unsinnig ist.

Welche dieser Ursachen vorliegt, kann automatisch nicht erkannt werden. Hierzu sind Programmdurchsichten erforderlich.

Schleifen werden derzeit noch nicht automatisch erkannt und behandelt. In der aktuellen Umsetzung wird speziell eine äußere Programmschleife, die über die Eingabedaten iteriert, so behandelt, dass die Programmzeilen für Schleifenein- und austritt von Hand festgelegt werden. Mit dem Eintritt in diese Schleife beginnt die Betrachtung des Pfades für den jeweiligen Datensatz, und mit dem Austritt endet sie wieder.

22.4 Technische Umsetzung

Die technische Umsetzung wurde für die Programmiersprache VS COBOL II vorgenommen. Der gesamte Ablauf gliedert sich in drei Abschnitte:

1. Parsen, Analysieren und Instrumentieren des COBOL-Programmcodes. Der Parser wurde über eine verfügbare COBOL-Grammatik und den Java-Compiler-Compiler erstellt.
2. Durchführen der Programmabläufe im instrumentierten Programmcode mit den Eingabedaten. Protokollieren der Pfade und Bedingungsterme in einer relationalen Datenbank.
3. Ermitteln der Entscheidungstabellen durch Datenbank-Auswertung: Stark äquivalente Eingabedaten werden in der Entscheidungstabelle zu einer Zeile zusammengefasst. Alle auf die Eingabedaten angewendeten Bedingungsterme bilden den Eintrag der linken Spalte. Der Pfad die rechte Spalte.

22.5 Zusammenfassung der Ergebnisse

Eingabedaten-Äquivalenzklassen können über das beschriebene Verfahren aus einem bestehenden Programmcode mit verfügbarer Grammatik sowie regulären Produktionsdaten gewonnen werden. Für COBOL-Programme wurde eine exemplarische Umsetzung erfolgreich implementiert. Der Einsatz in einem konkreten Projekt mit etwa 40 KLOC läuft derzeit.

Die automatische Behandlung von Schleifen, „goto“-Verzweigungen und Exceptions hinsichtlich der Auswirkung auf die Entscheidungstabelle wird noch weiter untersucht.

Literatur

[1] Frühauf K., Ludewig J., Sandmayr H.,: Software-Prüfung. vdf Hochschulverlag AG, Zürich, 1997