Übungen zur Vorlesung
Testautomatisierung
Sommersemester 2004

AG BS
Jan Peleska, Stefan Bisanz,
Aliki Tsiolakis

Blatt 3

Revision: 1.5

Test-Suite zum Testen eines kombinatorischen Systems Teil 2 – Übergangsüberdeckung

In diesem Aufgabenblatt wird die Test-Suite zur Komponente PLS aus Aufgabenserie 2 um die Überdeckung aller *Übergänge zwischen Eingabevektoren* erweitert.

Die Spezifikation der Komponente PLS ist gegenüber Aufgabenserie 2 unverändert.

Zur Erstellung der Test-Suite wird erneut ein Projektrahmen für den RT-Tester als Archiv zur Verfügung gestellt (`ta-project2.tgz`).

Aufgabe 1: Vorbereitung der Übergangsüberdeckung 40%

Teilaufgabe 1: Realisierung einer effizienten Mengenimplementierung

Eine der Grundvoraussetzungen des automatisierten Testens ist eine effiziente Implementierung der Testprozedur – damit die Tests mit möglichst geringem Zeitaufwand ausgeführt werden können, bzw. damit möglichst viele Tests in der verfügbaren Zeit durchgeführt werden können. Aus diesem Grund soll eine effiziente Implementierung von Mengen boolescher Eingabevektoren bereitgestellt werden:

Ganzzahlen eignen sich, um wiederum (beschränkte) Mengen von Ganzzahlen zu repräsentieren, indem jeweils ein Bit verwendet wird, um zu markieren, ob eine Ganzzahl in der Menge enthalten ist oder nicht. Dabei hängt es offensichtlich von der Anzahl der enthaltenen Bits ab, welche Zahlen als Elemente der jeweiligen Menge fungieren können – verwendet man z.B. den Typ `uint64_t` als Mengentyp, dann können darin die Zahlen `0, ..., 63` enthalten sein.

Weil die enthaltenen Ganzzahlen wiederum als boolesche Eingabevektoren interpretiert werden sollen, reicht diese Kodierung nur für $n = 6$ boolesche Eingaben aus, da $2^6 = 64$.

Für größere Eingabevektoren ist die Verwendung eines *Felds* von Ganzzahlen (statt nur einer Variablen) nötig: Wenn z.B. ein Feld `f[2]` zum Speichern einer Menge verwendet wird, so lassen sich die Zahlen `0, ..., 63` in `f[0]` und `64, ..., 127` in `f[1]` speichern, etc.

Verwenden Sie eine geeignete Struktur `setArray_t`, in welcher Sie sowohl das Feld als auch dessen tatsächliche Größe speichern. Somit können beliebig große Mengen verwaltet werden. Berücksichtigen Sie dabei geeignet, dass die Struktur ggf. zu viele Ganzzahlen enthalten kann.

Implementieren Sie folgende Funktionen mit Hilfe der verfügbaren Bitoperatoren:

```
/* Erzeugt eine leere Menge, welche für elementCount Elemente ausreicht. */
setArray_t *setCreate (uint64_t elementCount);
/* Vernichtet die angegebene Menge. */
void setDestroy (setArray_t *pSet);
/* Füllt die angegebene Menge mit allen Elementen, welche enthalten sein
   können. */
void setFill (setArray_t *pSet);
/* Fügt ein einzelnes Element in die angegebene Menge ein. */
void setInsert (setArray_t *pSet, uint64_t newelement);
/* Entfernt ein einzelnes Element aus der angegebenen Menge. */
```

```
void setRemove (setArray_t *pSet, uint64_t element);
/* Prüft, ob ein Wert Element der angegebenen Menge ist. */
bool_t setContains (setArray_t *pSet, uint64_t value);
/* Prüft, ob die angegebene Menge leer ist. */
bool_t setIsEmpty (setArray_t *pSet);
/* Liefert ein zufällig gewähltes Element aus der angegebenen Menge,
   falls sie nicht leer ist. */
uint64_t setGetRandom (setArray_t *pSet);
/* Gibt die angegebene Menge menschenlesbar auf stdout aus. */
void setPrintf (setArray_t *pSet);
```

Testen Sie Ihre Mengenimplementierung (unformal), indem Sie eine geeignete Methode `int main (int, char**);` bereitstellen, die obige Funktionen prüft. Dokumentieren Sie deren Ausgaben.

Teilaufgabe 2: Vorbereiten der "todo"-Liste für die Übergangsüberdeckung

Für die Realisierung der Testdatengenerierungsstrategie zur Überdeckung von Übergängen zwischen Eingabevektoren wurde in der Vorlesung vorgeschlagen, eine "todo"-Liste der Form $u : IN \to \mathbb{P}(IN)$ zu verwenden, die für jeden Eingabevektor speichert, welche Übergänge noch getestet werden müssen.
Diese Abbildung kann als Feld realisiert werden, dessen Indizes genau alle Quelleingabevektoren $source \in IN$ sind. Der jeweilige Feldinhalt ist dann die Menge der verbleibenden Zieleingabevektoren $target \in \mathbb{P}(IN)$.
Implementieren Sie die entsprechenden Funktionen:

```
/* Erzeugt eine todo-Abbildung, welche für Eingabevektoren mit
   inputCount Bits geeignet ist.
   randomSeed definiert eine Pseudozufallszahlenfolge für die zufällige
   Auswahl von Eingabevektoren über alle Verwendungen der entsprechenden
   Funktionen (s.u.). */
todoMap_t *todoCreate (uint64_t inputCount, uint64_t randomSeed);
/* Vernichtet die angegebene todo-Abbildung. */
void todoDestroy (todoMap_t *pTodo);
/* Entfernt den angegebenen Zieleingabevektor target aus der Zielmenge des
   Quelleingabevektors source in der angegebenen todo-Abbildung. */
void todoRemove (todoMap_t *pTodo, uint64_t source, uint64_t target);
/* Prüft, ob der angegebene Quelleingabevektor source in der angegebenen
   todo-Abbildung noch zugeordnete Zieleingabevektoren hat. */
bool_t todoHasTargets (todoMap_t *pTodo, uint64_t source);
/* Prüft, ob die angegebene todo-Abbildung noch Quelleingabevektoren mit
   zugeordneten Zieleingabevektoren hat. */
bool_t todoHasNonemptySource (todoMap_t *pTodo);
/* Liefert einen zufällig gewählten Zieleingabevektor zum angegebenen
   Quelleingabevektor aus der angegebenen todo-Abbildung, falls es einen
   solchen gibt. */
uint64_t todoGetTargetRandom (todoMap_t *pTodo, uint64_t source);
/* Liefert einen zufällig gewählten Quelleingabevektor aus der angegebenen
   todo-Abbildung, welcher noch zugeordnete Zieleingabevektoren hat,
   falls es einen solchen gibt. */
uint64_t todoGetNonemptySourceRandom (todoMap_t *pTodo);
/* Gibt die todo-Abbildung menschenlesbar auf stdout aus. */
void todoPrintf(todoMap_t *pTodo);
```

Verwenden Sie auch dafür eine geeignete Struktur `todoMap_t`.

Testen Sie Ihre Implementierung der todo-Abbildung ebenfalls (unformal), indem Sie die Methode `int main (int, char**);` entsprechend erweitern. Dokumentieren Sie auch die weiteren Ausgaben.

Aufgabe 2: Erweiterung der Test-Suite für Übergangsüberdeckung 40%

Teilaufgabe 1: Erstellung zusätzlicher Testprozeduren für Übergangsüberdeckung

Erweitern Sie die Test-Suite aus Aufgabenserie 2 so, dass Sie volle Übergangsüberdeckung bei den Kontrollvariablen erreichen. Für die Parameter paramDecompActivation und paramAutoSolutionType ist die Kombinationsüberdeckung hinreichend. Die restlichen Parameter (paramFsbSignAssigned, paramExitSignAssigned und paramRtsSignAssigned) werden nicht variiert, ihre Belegung ist im Testrahmen vorgegeben. Erweitern Sie die Test-Suite, indem Sie neue Testprozeduren hinzufügen.

Verwenden Sie – wie in der Vorlesung vorgeschlagen – ein Feld `bool *h[]`, um auch die Abbildung der Inputvektoren auf die tatsächlichen Eingabevariablen effizient zu gestalten.

Dokumentieren Sie die neuen Testfälle in den Testprozeduren, so dass diese nach der Testdurchführung mit in den Log-Dateien erscheinen. Daraus soll insbesondere hervorgehen, welcher *Übergang* jeweils getestet wird.

Teilaufgabe 2: Durchführung der neuen Testprozeduren

Führen Sie die in Teilaufgabe 1 erstellten Testprozeduren mit dem RT-Tester 6.0 aus. Protokollieren Sie während der Testdurchführung die Code-Coverage des Testlings.

Aufgabe 3: Bewertung der Teststrategien 20%

Im Gegensatz zu Aufgabenserie 2 wurde bzgl. der Eingabevektoren keine Kombinationsüberdeckung, sondern eine Übergangsüberdeckung realisiert (Aufgabe 2). Aus der Vorlesung wissen Sie, dass letztere geeignet ist, eine größere Klasse von Fehlern aufzudecken.

Teilaufgabe 1: Vergleich der verwendeten Teststrategien

Erläutern Sie für die von Ihnen sowohl (a) in *Aufgabenserie 2* als auch in (b) Aufgabe 2 durchgeführten Tests, welche Fehler Sie gefunden haben.

Ordnen Sie zu, mit welcher Teststrategie die jeweiligen Fehler aufgedeckt wurden.

Erläutern Sie, ob es dabei Fehler gibt, die sie mit der jeweiligen Teststrategie *zwangsläufig* aufdecken mussten. Erläutern Sie ebenfalls, ob sie Fehler mit der jeweiligen Teststrategie *zufällig* gefunden haben.

Teilaufgabe 2: Vollständigkeitsabschätzung der Testprozeduren

Bewerten Sie informal, ob Sie *alle* Fehler im vorliegenden Testling[1] aufgedeckt haben.

Die Ausgabedatei `psl.c.gcov` enthält die protokollierte Code-Coverage des Testlings (gemäß Aufgabe 2). Im Gegensatz zu Aufgabenserie 2 beinhaltet sie nicht nur Angaben zur *Anweisungsüberdeckung*, sondern auch zur *Zweigüberdeckung*. Darüber hinaus ist angedeutet, welche Teile von booleschen Ausdrücken zu der jeweiligen Auswertung beigetragen haben. Verwenden Sie Code-Coverage als Anhaltspunkt für Ihre Bewertung.

Abgabe: Bis Montag, 21. Juni 2004, im Tutorium.

Geben Sie alle Aufgabenlösungen sowohl (1) *ausgedruckt* (in MZH8210, MZH8170, oder im Tutorium) als auch (2) *elektronisch* (http://www.informatik.uni-bremen.de/~alien/upload.html) ab.

[1] Beachten Sie: Der Testling ist im Vergleich zu Aufgabenserie 2 leicht modifiziert.

In allen Dokumenten und Dateien die Namen aller Gruppenmitglieder nicht vergessen!

Übersicht der Abgaben:

Aufg.	**Schriftliche Abgabe**	**Elektronische Abgabe**
1	alles	alles (eingebettet in das RTT-Projekt, s. 2)
2	pro Testprozedur: *Real-Time Test Specification*-Dateien (`specs/*.rts`), Konfigurationsdatei (`conf/*.conf`), alle evtl. geänderten Dateien mit neuen Stubs, Typen o.ä. (Ausdruck gerne auch mit a2ps)	Das gesamte Projekt inklusive der Log-Dateien in allen Testprozeduren sowie des `code-coverage`-Verzeichnisses
3	Teil des Gesamtdokumentes	Teil des Gesamtdokumentes (`ps` oder `pdf`)

Die elektronische Abgabe bitte als Archiv (.tar.gz oder .zip).