Übungen zur Vorlesung
Testautomatisierung 1
WS 2009/2010

AG BS
Jan Peleska
Elena Vorobev

Übungsblatt 4

Revision: 1.0

Einbinden der W-Methode in den RT-Tester

Spezifikation

Folgende Spezifikation in Form eines Mealy-Automaten beschreibt eine Anwendung, die aus einem C-Programmtext alle Kommentare erkennt und ausgibt:

- Die Eingabemenge ist $I = \{*, /, \phi\}$. Dabei abstrahiert ϕ ein beliebiges Zeichen, welches nicht in $\{*, /\}$ enthalten ist.
- Die Ausgabemenge besteht aus den Aktionen $O = \{\mathsf{ignore}, \mathsf{reset}, \mathsf{concat}, \mathsf{truncate}, \mathsf{print}\}$, die einen in der Implementierung zu manipulierenden Ausgabepuffer B betreffen. Diese Aktionen haben folgende Interpretation:
  - ignore: Das eingelesene Zeichen wird ignoriert – der Ausgabepuffer B bleibt unverändert.
  - reset: B wird auf den leeren Puffer zurückgesetzt.
  - concat: Das aktuell eingelesene Zeichen wird an B hinten angehängt.
  - truncate: Das letzte Zeichen in B wird gelöscht.
  - print: Der Pufferinhalt wird ausgegeben.

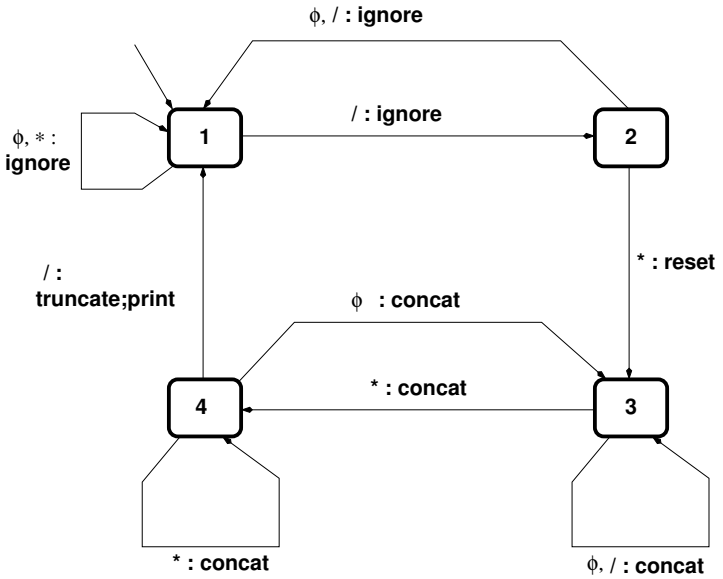

Abbildung 1: Mealy-Automat zur oben beschriebenen Anwendung

Der zu dieser Aufgabenstellung gehörige Mealy-Automat ist in Abb. 1 spezifiziert. Da "/" ein Eingabesymbol ist, wird als Trennungssymbol zwischen Eingabe und Aktion ":" gewählt.

Implementierung

Zur oben beschriebenen Spezifikation wurde ein C-Programm entwickelt[1]. Die zugehörigen Dateien `printcomments.hpp` und `printcomments.cpp` befinden sich im Verzeichnis `sut` des Testprojekts *TA1-Uebung-04*, welches auf der TA1-Webseite zum Download bereitsteht.

Aufgabe: Modultest im RT-Tester mit Hilfe der W-Methode

Als *System Under Test* (SUT) soll die Funktion

```
void mealyAutomaton( int c )
```

mit den Unterfunktionen

```
int handleState1/2/3/4( int c )
```

dienen. Da im Test `mealyAutomaton()` direkt aufgerufen werden soll, können die Funktionen `void main( void )` sowie `int input( void )` in `printcomments.cpp` vernachlässigt werden.

1. W-Test-Cases - abstrakt

   Erzeugen Sie mit Hilfe der von Ihnen im Rahmen des Übungsblatts 3 implementierten W-Methode alle W-Test-Cases unter der (falschen) Annahme, dass die Implementierung nicht mehr Zustände als die Spezifikation enthält.

   Um Ihre Implementierung nicht anpassen zu müssen, können Sie sowohl Inputs als auch Outputs (hier: Aktionen) des Mealy-Automaten als Strings darstellen. Zwei mögliche Input- und Outputfolgen wären dann beispielsweise

   $$\mathrm{i}_a = <\text{"slash"}, \text{"star"}, \text{"any"}, \text{"star"}, \text{"slash"}>$$
   $$\mathrm{o}_a = <\text{"ignore"}, \text{"reset"}, \text{"concat"}, \text{"concat"}, \text{"truncate;print"}>$$

2. W-Test-Cases - konkret

   Aus den abstrakten[2] sollen im nächsten Schritt konkrete Input- und Outputfolgen konstruiert werden.

   - Für konkrete Inputfolgen muss "slash" durch '/', "star" durch '*' und "any" durch ein beliebiges anderes (zufällig gewähltes) ASCII-Zeichen ersetzt werden. Zum obigen Beispiel passende konkrete Inputfolge wäre dann z.B. $\mathrm{i}_k = <$ '/', '*', 'd', '*', '/' $>$.

   - Für konkrete Outputfolgen soll für jede Aktion eine Kennung verwendet werden (z.B. der Anfangsbuchstabe). Da in der gegebenen Implementierung `truncate()` und `print()` zwei verschiedene Funktionen sind, müssen diese unterschiedliche Kennungen erhalten. Weil außerdem bei einem `print()`-Aufruf der Inhalt des Puffers zu prüfen ist, können auch die hier erwarteten Zeichen in die Outputfolge aufgenommen werden. Zum obigen Beispiel passende konkrete Outputfolge wäre dann $\mathrm{o}_k = <$ 'i', 'r', 'c', 'c', 't', 'p', 'd' $>$.

[1] Um Namenskollisionen zu vermeiden, wurden die Funktionen im Namespace `uebung04` definiert, so dass zur Übersetzung ein C++-Compiler verwendet werden muss.

[2] Durch die W-Methode konstruierte Input- und Outputfolgen sind insofern abstrakt, als dass statt konkreter Zeichen symbolische Bezeichner verwendet werden, z.B. "any" statt eines bestimmten ASCII-Zeichens.

- Konkrete Input- und Outputfolgen können bei Testinitialisierung erzeugt und in globalen Variablen abgelegt werden, damit diese zwecks Prüfung auch aus den Stubs heraus abgefragt werden können.

3. Instrumentierung des Testlings

   Um die Aufrufe der Aktionen `ignore()`, `reset()`, `concat()` und `truncate()` in der Testumgebung erkennen und überprüfen zu können, soll der Testling instrumentiert werden - in jeder der aufgeführten Aktionen soll ein zusätzlicher Stub-Aufruf `action( Kennung der Aktion )` (s.u.) erfolgen.

4. Definition von Stubs

   - Es ist ein RT-Tester-Stub `void action( Kennung der Aktion )` zu definieren, welcher überprüft, ob die ausgelöste Aktion mit der erwarteten übereinstimmt. Die Form der als Parameter übergebenen Kennung sollte der in den konkreten Outputfolgen entsprechen, damit ein Vergleich direkt möglich ist.
   - Die Funktion `void print( void )` ist durch einen RT-Tester-Stub zu ersetzen, welcher bei jedem Aufruf überprüft, ob der auszugebende Inhalt des Puffers mit dem erwarteten übereinstimmt.
   - Zur Definition der RT-Tester-Stubs sollen `@GLOBAL`, `@BODY` und `@ASSERT`-Sektionen verwendet werden.

Abgabe: 19.01.2010 bis 16:00 Uhr
Die Abgabe erfolgt per E-Mail an die Tutorin.