Übungen zur Vorlesung
Testautomatisierung
Wintersemester 2002/03

AG BS
Jan Peleska, Aliki Tsiolakis

Blatt 4

Checking-Algorithmen auf nicht-normalisierten Transitionsgraphen

Aufgabe 1: 100%

Transformieren Sie den in der Vorlesung am 13.1.2003 vorgestellten Algorithmus für on-the-fly Checking von nicht-normalisierten Transitionsgraphen in einen off-line anzuwendenden Algorithmus, der *timed traces* (Tupel aus Event und Time-Stamp) als Input bekommt und ohne das explizite Setzen und Ablaufen von Timern auskommt (also ohne physikalische Uhr).

Schreiben Sie den transformierten Algorithmus auf und zeigen Sie an (mindestens) einem geeigneten Beispiel, dass der Algorithmus korrekt funktioniert.

Abgabe: Bis Dienstag, 28.01.2003, vor dem Tutorium.

Geben Sie für die Aufgabe eine **schriftliche Lösung** ab. Bitte schicken Sie zusätlich das PDF oder Postscript Ihrer Abgabe an `tsio@informatik.uni-bremen.de`.

Sowohl bei der schriftlichen Lösung als auch in den CSP-Dateien die Namen aller Gruppenmitglieder nicht vergessen!

Algorithmus aus der Vorlesung am 13.1.2003:

Checking Algorithm on Un-normalised Transition Graphs

Let PS denote the set of potential states of the SUT. Encoding of PS:

$$\text{PS} = \mathcal{P}\ \{\ (s, T)\ \mid \text{s} \in \text{S},\ \text{T} \subseteq \underbrace{\{\ s \xrightarrow{e} s'\ \mid\ s \xrightarrow{e} s' \in TRANS\}}_{T(s)}\ \}$$

where

S : Set of all states in the transitions graph
TRANS : Set of all transitions in the transition graph
$T(s)$: Set of all possible transitions emanating from s

The checking algorithm starts with the initial state s_0 and repeats three phases:

```
PS = { (s0, T(s0)) };
PSOLD = ∅ ;
do {
  // phase 1: check for errors
  if ( PS == ∅) {
    error(); break;
  }
  // phase 2:
  pre_normalise_PS ();
  // phase 3:
  process_events ();
} while (1);
```

pre_normalise_PS ()

```
while ( PS ≠ PSOLD) {
  PSOLD = PS;
  PS = ∅;
  foreach ( (s, T) ∈ PSOLD) {
    // stop states are legal as long as SUT interface event occurs
    if ( T(s) == ∅) {
      PS = PS ∪ { (s, ∅) };
      continue;
    }
    TOLD = T;
    T = ∅;
    // add post-states of τ-transitions
    PS = PS ∪ { (s', T(s')) | s --τ--> s' ∈ TOLD };
    // add post states of set-timer events, set the associated timer
    E = { (s', T(s')) | ∃ x ∈ SETT. s --x--> s' ∈ TOLD };
    PS = PS ∪ E;
    foreach ( s --x--> s' ∈ TOLD )
      if (x ∈ SETT ) set_timer(x);
```

```
      // add post states of elapsed timers, if these exist, leave s
      E = { (s', T(s')) | ∃ e ∈ ELAT. elapsed(e) ∧ s -e-> s' ∈ TOLD };
      if (E ≠ ∅) {
        PS = PS ∪ E;
        continue;
      }
      // keep possible transitions related to SUT-interface events
      // or timers which have not yet elapsed
      T = T ∪ { s -e-> s' ∈ TOLD | e ∈ input(SUT) ∪ output(SUT) ∪ ELAT };
      if ( T ≠ ∅ )
        PS = PS ∪ { (s, T) };
    }
  }
```

process_events ()

```
  if ( event e ∈ input(SUT) ∪ output(SUT) has occured ) {
    PSOLD = PS;
    PS = ∅;
    foreach ( (s, T) ∈ PSOLD ) {
      E = { (s', T(s')) | s -e-> s' ∈ T };
      PS = PS ∪ E ;
    }
  }
  else {
    // remove SUT outputs which should have occured now
    PSOLD = PS;
    PS = ∅;
    foreach ( (s, T) ∈ PSOLD ) {
      if ( T ∩ { s -z-> s' | z ∈ output(SUT) } == ∅ )
        PS = PS ∪ { (s, T) };
    }
  }
```