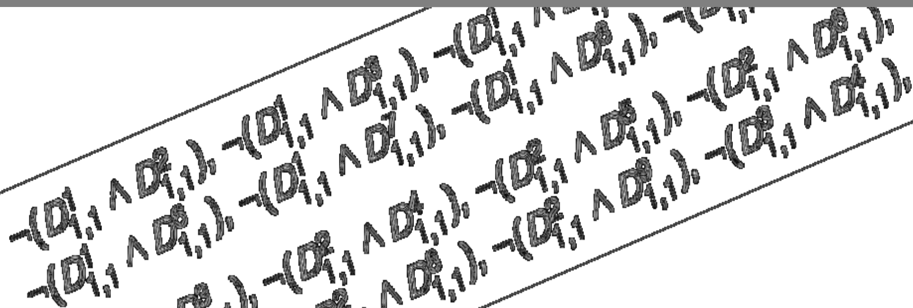
KIT
Karlsruhe Institute of Technology

Formale Systeme

Die Sprache PROMELA

Prof. Dr. Bernhard Beckert | WS 2009/2010

KIT – INSTITUT FÜR THEORETISCHE INFORMATIK



Beschreibung endlicher Automaten

Karlsruhe Institute of Technology

Die Darstellung konkreter endlicher Automaten in graphischer Form ist nur für kleine Automaten möglich, für größere, wie sie in realistischen Anwendungen auftreten, ist das nicht praktikabel.

Wir betrachten als eine Alternative die Modellierungssprache *Promela*.

Beschreibung endlicher Automaten

Karlsruhe Institute of Technology

Die Darstellung konkreter endlicher Automaten in graphischer Form ist nur für kleine Automaten möglich, für größere, wie sie in realistischen Anwendungen auftreten, ist das nicht praktikabel.

Wir betrachten als eine Alternative die Modellierungssprache *Promela*.

Promela

Karlsruhe Institute of Technology

- Process meta language
- Modellierungssprache für indeterministische gekoppelte erweiterte endliche Automaten.
- Entwickelt von Gerard Holzmann seit 1980.
- Weit verbreitetes Verifikationswerkzeug SPIN (Simple Promela INterpreter)
- Angabe der zu verifizierenden Eigenschaft als LTL Formel oder Büchi-Automat.

Promela

- Process meta language
- Modellierungssprache für indeterministische gekoppelte erweiterte endliche Automaten.
- Entwickelt von Gerard Holzmann seit 1980.
- Weit verbreitetes Verifikationswerkzeug SPIN (Simple Promela INterpreter)
- Angabe der zu verifizierenden Eigenschaft als LTL Formel oder Büchi-Automat.

Promela

Karlsruhe Institute of Technology

- Process meta language
- Modellierungssprache für indeterministische gekoppelte erweiterte endliche Automaten.
- Entwickelt von Gerard Holzmann seit 1980.
- Weit verbreitetes Verifikationswerkzeug SPIN (Simple Promela INterpreter)
- Angabe der zu verifizierenden Eigenschaft als LTL Formel oder Büchi-Automat.

Promela

- Process meta language
- Modellierungssprache für indeterministische gekoppelte erweiterte endliche Automaten.
- Entwickelt von Gerard Holzmann seit 1980.
- Weit verbreitetes Verifikationswerkzeug SPIN (Simple Promela INterpreter)
- Angabe der zu verifizierenden Eigenschaft als LTL Formel oder Büchi-Automat.

Promela

KIT Karlsruhe Institute of Technology

- Process meta language
- Modellierungssprache für indeterministische gekoppelte erweiterte endliche Automaten.
- Entwickelt von Gerard Holzmann seit 1980.
- Weit verbreitetes Verifikationswerkzeug SPIN (Simple Promela INterpreter)
- Angabe der zu verifizierenden Eigenschaft als LTL Formel oder Büchi-Automat.

Promela Beispiel

```
/* Peterson: mutual exclusion [1981] */
bool turn, flag[2];
byte ncrit;
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
 again: flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again}
```

Promela Beispiel

Karlsruhe Institute of Technology

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again: flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again   }
```

Promela Beispiel

KIT
Karlsruhe Institute of Technology

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again:  flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again    }
```

Promela beschreibt Zustandsübergangssysteme durch die Definition von Prozesstypen (process types).

Promela Beispiel

KIT Karlsruhe Institute of Technology

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again: flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again    }
```

In diesem Beispiel wird der Prozesstyp user() deklariert.

Promela Beispiel

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again:  flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again    }
```

Die Deklaration eines Prozesstyps bewirkt noch nichts. Erst mit der Erzeugung einer Instanz eines Prozesstypes sind Aktionen verbunden.

Promela Beispiel

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again:  flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again   }
```

Instanzen eines Prozesstyps werden normalerweise in einem Initialisierungprozess erzeugt, z.B.
`init { run users(); }`.

Promela Beispiel

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again:  flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again   }
```

`active [2]` ist eine Abkürzung für
`init { run users(); run users(); }`.

Promela Beispiel

KIT Karlsruhe Institute of Technology

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again: flag[_pid] = 1;
       turn = _pid;
       (flag[1 - _pid] == 0 || turn == 1 - _pid);
       ncrit++;
       assert(ncrit == 1); /* critical section */
       ncrit--;
       flag[_pid] = 0;
       goto again   }
```

Prozesse werden in der Reihenfolge ihrer Aktivierung durchnumeriert. Auf diese Nummer kann mit der lokalen Variablen `_pid` zugegriffen werden.

Promela Beispiel

```
bool turn, flag[2];
byte ncrit;
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again:  flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again   }
```

Globale Variablen, im Beispiel `turn`, `flag[2]`, `ncrit`, können von jedem Prozess gelesen und beschrieben werden.

Promela Beispiel

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again:  flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again    }
```

In Promela ist jeder Ausdruck auch ein Befehl. Ergibt die Auswertung den Wahrheitswert *wahr* oder einen Wert > 0 , dann geht die Ausführung mit der nächsten Anweisung weiter.

Promela Beispiel

Karlsruhe Institute of Technology

```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again: flag[_pid] = 1;
       turn = _pid;
       (flag[1 - _pid] == 0 || turn == 1 - _pid);
       ncrit++;
       assert(ncrit == 1); /* critical section */
       ncrit--;
       flag[_pid] = 0;
       goto again   }
```

Andernfalls bleibt der Prozess blockiert bis der Ausdruck durch Aktionen anderer Prozesse wahr wird.

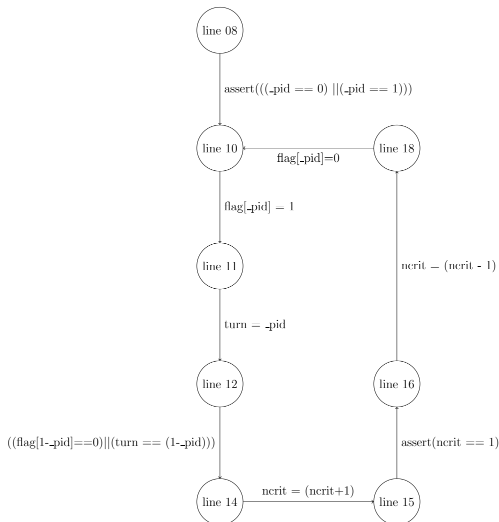
Promela Beispiel

KIT Karlsruhe Institute of Technology

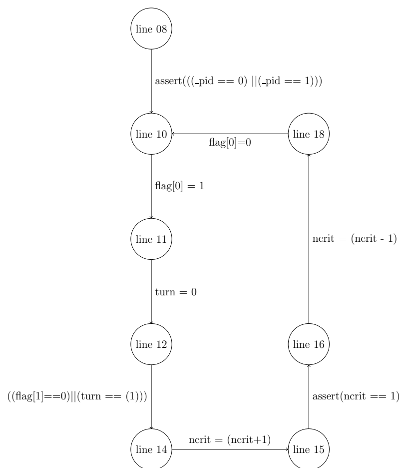
```
active [2] proctype user()
{       assert(_pid == 0 || _pid == 1);
again: flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1); /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again   }
```

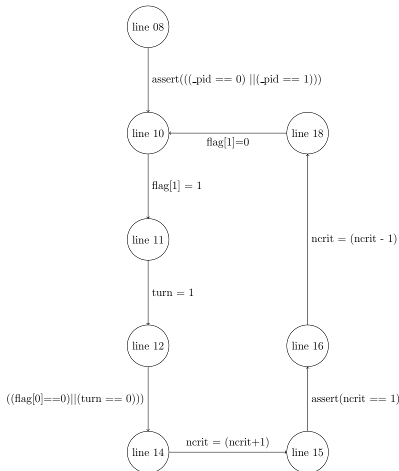
Falls das Argument des assert Befehls wahr ist geht die Ausführung mit der nächsten Zeile weiter, anderenfalls wird ein Fehler berichtet und die Simulation oder Verifikation abgebrochen.

Generischer Automat zum *user* Prozess



Instanzen des generischen Automaten





Approximation Produktautomat