Modelchecking

V. Ein Modelchecker: NuSMV

- Motivation und Hintergrund
- Modellierung
- Eigenschaften
- Anwendung

1. Motivation

- Wir kennen jetzt die Grundlagen des Modelcheckings, auch wenn uns noch ganz wesentliche Aspekte für die Effizienz fehlen.
  → Symbolisches Modelchecking etc.
- Die reine Theorie ist schön, macht aber nur im Hinblick auf ihre Anwendung Sinn; wir wollen aber auch keinen Modelchecker selber schreiben.
- Deshalb probieren wir in der Übung mit (kleinen) Beispielen einen Modelchecker aus: **NuSMV**.

NuSMV und SMV

- **NuSMV** ist eine Neuimplementierung und Erweiterung von **SMV**, dem ersten Symbolischen Modelchecker von der **CMU**.
  - **SMV** = **S**ymbolic **M**odel **V**erifier
  - **NuSMV** = **N**ew **S**ymbolic **M**odel Checker
- **NuSMV** unterstützt CTL und LTL.

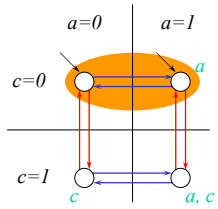
NuSMV

- Es folgt ein Überblick über die wesentlichen Features von **NuSMV**:
  - Notation zur Beschreibung von Kripkestrukturen (bzw. Systemen)
  - Notation für Eigenschaften (CTL, LTL)
  - Bedienung von **NuSMV**
- Für Details siehe
  `http://nusmv.irst.itc.it/`

2. Modellierung

- Übergangsbedingung
- Module
  - synchron
  - asynchron
- Felder
- Weitere Features

Erinnerung (Kapitel II)



- Boolesche Variablen:
  $V = \{ a, c \}$
- Anfangsbedingung:
  $S_0 \equiv \neg c$
- Übergangsbedingung:
  $\mathcal{R} \equiv$
  $(a' = \neg a \wedge c' = c) \vee$
  $(a' = a \wedge c' = \neg c)$
- Beschriftung implizit:
  $AP = V$

Übergangsbedingung

Beispiel:

```
MODULE main
VAR                    V
  a : boolean;
  c : boolean;
INIT                   S0
  !c
TRANS                  R     a'   ¬   ∧   c'        ∨
  ( next(a) = !a & next(c) =  c ) |
  ( next(a) =  a & next(c) = !c )
```


Modellierung

- Die Repräsentation einer Kripkestruktur durch boolesche Variablen (**VAR**), eine Anfangsbedingung (**INIT**) und eine Übergangsbedingung (**TRANS**) läßt sich 1:1 in die SMV-Notation übertragen.

$\neg$ → **!**	$\Rightarrow$ → **->**	v → **v**
$\vee$ → **|**	$\Leftrightarrow$ → **<->**	v' → **next(v)**
$\wedge$ → **&**	$\otimes$ → **xor**	

Einfachere Darstellung

- Die Repräsentation mit Hilfe der Übergangsbedingung ist bei großen Systemen oft unzweckmäßig!
- NuSMV bietet die Möglichkeit, die Übergänge „programmiersprachlicher“ darzustellen und ein Modell modular aufzubauen.

Bemerkung:
Die Notation ist etwas gewöhnungsbedürftig, da sie ursprünglich für den Schaltkreisentwurf entwickelt wurde.

Module: Beispiel

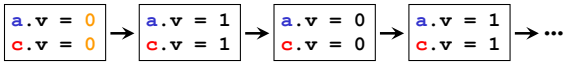

```
MODULE cycle
VAR
  v : boolean;
ASSIGN
  init(v):= 0;
  next(v):= !v;

MODULE main
VAR
  a : cycle();
  c : cycle();
```

Anfangsbedingung und Übergangsrelation werden jetzt durch Zuweisungen (an **init(v)** bzw. **next(v)** beschrieben).

Das Modul **cycle** wird zweimal instantiiert; es gibt also insgesamt zwei Variablen: **a.v** und **c.v**. Die Übergänge beider Module werden **synchron** ausgeführt!

Synchrone Ausführung

Die Module **a** und **c** des vorangegangenen Beispiels laufen synchron ab. Es gibt also nur einen Ablauf:



Diese synchrone Ausführung ist zur Modellierung von Schaltkreisen zweckmäßig; zur Modellierung unabhängiger Komponenten ist die synchrone Ausführung ungeeignet.

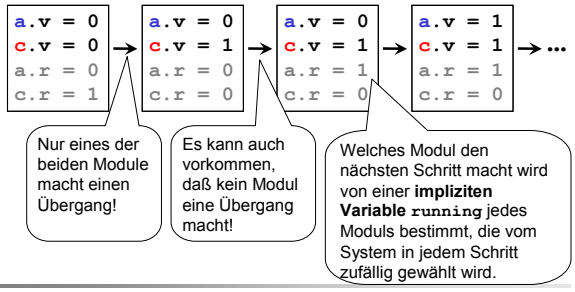
Beispiel: Asynchrone Ausführung


```
MODULE cycle
VAR
  v : boolean;
ASSIGN
  init(v):= 0;
  next(v):= !v;

MODULE main
VAR
  a : process cycle();
  c : process cycle();
```

Das Schlüsselwort **process** besagt, daß die entsprechenden Module **asynchron** ausgeführt werden; d.h. die Schritte der Module werden unabhängig voneinander ausgeführt.

Asynchrone Ausführung

Jetzt werden die Module **a** und **c** asynchron ausgeführt. Ein möglicher Ablauf:



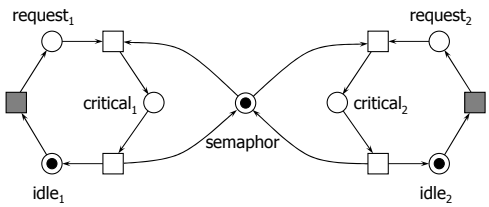
Asynchrone Ausführung

Achtung:

- Die asynchrone Ausführung funktioniert nur korrekt, wenn die Übergänge des Moduls mit Hilfe von Zuweisungen (**ASSIGN**) beschrieben werden!
- Wenn die Übergänge durch eine Übergangsbedingung (**TRANS**) beschrieben werden, läuft das Modul trotz asynchroner Deklaration (**process**) synchron!

  → Das hat etwas mit der Realisierung von **ASSIGN** und **process** zu tun!

Weiteres Beispiel



Weiteres Beispiel


```
MODULE agent(semaphor)

VAR
  state : { idle, request, critical };

ASSIGN
  init(state):= idle;

  next(state):=
    case
      state = idle              : { idle, request };
      state = request & semaphor : critical;
      state = critical          : idle;
      1                         : state;
    esac;

  next(semaphor):=
    case
      state = request & semaphor : 0;
      state = critical          : 1;
      1                         : semaphor;
    esac;
```

Explizite Definition des Wertebereiches

nichtdeterministische Auswahl

Innerhalb eines Moduls werden alle Zuweisungen synchron ausgeführt.

Weiteres Beispiel


```
...

MODULE main

VAR
  semaphor : boolean;
  agent1   : process agent(semaphor);
  agent2   : process agent(semaphor);

ASSIGN
  init(semaphor):= 1;
```

Definiert den Semaphor und zwei Instanzen des Moduls **agent**, die jeweils denselben Semaphor benutzen.

Initialisierung des Semaphors.

Beispiel: array


```
...

MODULE main

VAR
  semaphor : boolean;
  agent    : array 1..1000 of process agent(semaphor);

ASSIGN
  init(semaphor):= 1;
```

Definiert 1000 Agenten, die den Semaphor benutzen.

Allerdings sind selbst einfache Eigenschaften für dieses Modell nicht mehr in vertretbarer Zeit verifizierbar.

Weitere Features

- **DEFINE** erlaubt die Einführung von Abkürzungen für komplizierte Ausdrücke:

```
DEFINE
  violation:= agent1.state = critical  &
              agent2.state = critical
```

- **FAIRNESS** erlaubt die Formulierung von Fairnessannahmen

```
FAIRNESS
  running
```

  Sollte man für alle asynchronen Module (**process**) hinzufügen, damit garantiert ist, daß auch immer wieder mal ein Übergang dieses Moduls ausgewählt wird (→ Details in Kapitel VI. Fairness).

- u.v.a.m.

3. Eigenschaften

- Eigenschaften können in NuSMV als CTL-Formeln und LTL-Formeln formuliert werden.
- Es müssen lediglich die booleschen Operatoren umgewandelt werden:

¬ → **!**	⇒ → **->**
∨ → **|**	⇔ → **<->**
∧ → **&**	⊗ → **xor**

- Bei LTL-Formel entfällt das „führende“ **A**.

Eigenschaften

- Eigenschaften können entweder in derselben Datei wie das Modell formuliert werden
- oder nach dem Start von NuSMV interaktiv eingegeben werden.
  → Benutzung von NuSMV

Beispiel

```
...

MODULE main
VAR
  semaphor : boolean;
  agent1   : process agent(semaphor);
  agent2   : process agent(semaphor);
ASSIGN
  init(semaphor):= 1;

SPEC
  AG !( agent1.state=critical & agent2.state=critical )
SPEC
  EG agent1.state=request
LTLSPEC
  G ( agent1.state=request -> F agent1.state=critical )
```

Spezifiziert eine CTL-Eigenschaft

Spezifiziert eine LTL-Eigenschaft

4. Anwendung

- „Installation“ von NuSMV
- Start von NuSMV
- Einlesen und Initialisierung von Modellen (und Eigenschaften)
- Überprüfen von Eigenschaften
- Simulation des Modells

„Installation“ von NuSMV (Linux)

```
unix > setenv NuSMV_LIBRARY_PATH
/opt/nusmv-2.1.2-zchaff/share/nusmv
```

Pfad für die Hilfedateien von NuSMV.

Wenn Sie oft mit NuSMV arbeiten, lohnt es sich, diesen Pfad in der profile oder cshrc einzustellen.

Dort finden Sie auch einige Beispiele.

Start von NuSMV

```
unix > NuSMV -int  semaphor.smv
...
NuSMV > go
NuSMV > check_spec
NuSMV > check_ltlspec
```

Startet NuSMV im interaktiven Modus

Die Datei mit dem Modell (und evtl. den Eigenschaften)

Initialisiert das System für die Verifikation bzw. die Simulation

Überprüft alle CTL-Eigenschaften und gibt ggf. Abläufe aus, die als Gegenbeispiele dienen

Überprüft die LTL-Eigenschaften und gibt ggf. Abläufe aus, die als Gegenbeispiele dienen

Explizite Angabe von Formeln

```
NuSMV > check_spec -p "EF(!a & !c)"
```

Für CTL-Eigenschaften

Für LTL-Eigenschaften

```
NuSMV > check_ltlspec -p
        "G ( a -> F c )"
```


Erreichbare Zustände

```
NuSMV > compute_reachable
```

Berechnet die Menge der erreichbaren Zustände des Systems.

- Das kann zu erheblichen Verbesserungen beim nachfolgenden Überprüfen von Eigenschaften führen.
- Manchmal ist die Berechnung der erreichbaren Zustände jedoch sehr aufwendig (deshalb wird diese Berechnung vom System nicht automatisch durchgeführt).

Simulation

- Um zu überprüfen, ob das Modell sich so verhält, wie man es sich gedacht hat, stellt NuSMV einige Operationen zur Verfügung, um das Modell zu simulieren (nach `go`):

`pick_state [opt]`
wählt einen Anfangszustand
- `-r` : zufällig aus allen Anfangszuständen
- `-v` : gibt den gewählten Zustand aus
- `-i` : erlaubt die interaktive Auswahl eines Anfangszustandes

Simulation

`simulate [opt] n`
simuliert das Modell ausgehend vom zuvor gewählten Zustand für n Schritte
- `-r` : durch zufällig Auswahl des nächsten Übergangs
- `-p` : gibt den simulierten Pfad aus (wobei nur die geänderten Variablen ausgegeben werden)
- `-v` : gibt den simulierten Pfad aus (mit allen Variablen)
- `-i` : erlaubt die interaktive Auswahl des nächsten Übergangs

Simulation

`show_traces [n]`
gibt den simulierten Pfad aus
(man kann sich auch früher simulierte Pfade ansehen; siehe Dokumentation)

`print_current_state`
gibt den aktuellen Zustand aus

`goto_state n.m`
macht einen früheren Zustand zum aktuellen Zustand

- NuSMV bietet viele weitere Features, Kommandos und Optionen, die hier nicht im einzelnen behandelt werden können. Dies betrifft insbesondere Kommandos zur Optimierung des Modelcheckings.
- Der hier vorgestellte Ausschnitt reicht für den Einstieg.
- Einige weitere Features werden wir später noch kennenlernen.

- Der Rest findet sich in der Dokumentation:
  `http://nusmv.irst.itc.it/`