

SMV

21. Dezember 2004

SMV:

- ▶ kurz für : *Symbolic Model Verifier*

SMV:

- ▶ kurz für : *Symbolic Model Verifier*
- ▶ Teil der Doktorarbeit von *Ken McMillan* (1992)

SMV:

- ▶ kurz für : *Symbolic Model Verifier*
- ▶ Teil der Doktorarbeit von *Ken McMillan* (1992)
- ▶ Dient der Modellierung *endlicher* Systeme (FAs, Schaltkreise)

SMV:

- ▶ kurz für : *Symbolic Model Verifier*
- ▶ Teil der Doktorarbeit von *Ken McMillan* (1992)
- ▶ Dient der Modellierung *endlicher* Systeme (FAs, Schaltkreise)
- ▶ und dem Überprüfen von **CTL**-Eigenschaften.

SMV:

- ▶ kurz für : *Symbolic Model Verifier*
- ▶ Teil der Doktorarbeit von *Ken McMillan* (1992)
- ▶ Dient der Modellierung *endlicher* Systeme (FAs, Schaltkreise)
- ▶ und dem Überprüfen von **CTL**-Eigenschaften.
- ▶ nutzt *BDDs* zur kompakten Darstellung der Übergangsrelation.

SMV:

- ▶ kurz für : *Symbolic Model Verifier*
- ▶ Teil der Doktorarbeit von *Ken McMillan* (1992)
- ▶ Dient der Modellierung *endlicher* Systeme (FAs, Schaltkreise)
- ▶ und dem Überprüfen von **CTL**-Eigenschaften.
- ▶ nutzt *BDDs* zur kompakten Darstellung der Übergangsrelation.
 - ▶ BDDs im neuen Jahr

SMV:

- ▶ kurz für : *Symbolic Model Verifier*
- ▶ Teil der Doktorarbeit von *Ken McMillan* (1992)
- ▶ Dient der Modellierung *endlicher* Systeme (FAs, Schaltkreise)
- ▶ und dem Überprüfen von **CTL**-Eigenschaften.
- ▶ nutzt *BDDs* zur kompakten Darstellung der Übergangsrelation.
 - ▶ *BDDs* im neuen Jahr
- ▶ Einführung vom letzten Jahr auf Übungsseite verlinkt.

Themen heute:

- ▶ Einführung in SMV
- ▶ Entwurf einer Fahrstuhlsteuerung entsprechend einer Spezifikation
- ▶ Modellierung dieser in SMV
- ▶ Überprüfen, ob die Steuerung der Spezifikation entspricht mittels SMV.

Fahrstuhl - Problemstellung

Szenario:

- ▶ Haus mit drei Stockwerken ($\{S_1, S_2, S_3\}$) und einem Fahrstuhl.

Gesucht:

- ▶ Geeignete Fahrstuhlsteuerung. (Spezifikation später)

Bereitgestellte Informationen/Signale für Spezifikation:

- ▶ Aktuelle Position des Lifts; auf einem Stockwerk oder zwischen zwei: $\text{pos} \in \{S_1, S_2, S_3, S_{1,2}, S_{1,2}\}$

Bereitgestellte Informationen/Signale für Spezifikation:

- ▶ Aktuelle Position des Lifts; auf einem Stockwerk oder zwischen zwei: $\text{pos} \in \{S_1, S_2, S_3, S_{1,2}, S_{1,2}\}$
- ▶ Türstatus: $\text{door} \in \{\text{open}, \text{closed}\}$

Bereitgestellte Informationen/Signale für Spezifikation:

- ▶ Aktuelle Position des Lifts; auf einem Stockwerk oder zwischen zwei: $\text{pos} \in \{S_1, S_2, S_3, S_{1,2}, S_{1,2}\}$
- ▶ Türstatus: $\text{door} \in \{\text{open}, \text{closed}\}$
- ▶ Ob sich der Fahrstuhl bewegt: $\text{moving} \in \{0, 1\}$

Bereitgestellte Informationen/Signale für Spezifikation:

- ▶ Aktuelle Position des Lifts; auf einem Stockwerk oder zwischen zwei: $\text{pos} \in \{S_1, S_2, S_3, S_{1,2}, S_{1,2}\}$
- ▶ Türstatus: $\text{door} \in \{\text{open}, \text{closed}\}$
- ▶ Ob sich der Fahrstuhl bewegt: $\text{moving} \in \{0, 1\}$
- ▶ Richtung der letzten Bewegung: $\text{dir} \in \{\text{up}, \text{down}\}$

Bereitgestellte Informationen/Signale für Spezifikation:

- ▶ Aktuelle Position des Lifts; auf einem Stockwerk oder zwischen zwei: $\text{pos} \in \{S_1, S_2, S_3, S_{1,2}, S_{1,2}\}$
- ▶ Türstatus: $\text{door} \in \{\text{open}, \text{closed}\}$
- ▶ Ob sich der Fahrstuhl bewegt: $\text{moving} \in \{0, 1\}$
- ▶ Richtung der letzten Bewegung: $\text{dir} \in \{\text{up}, \text{down}\}$
- ▶ Türsensor. Meldet, ob Menschen gerade die Fahrstuhltüren durchschritten haben: $\text{sensor} \in \{0, 1\}$.

Bereitgestellte Informationen/Signale für Spezifikation:

- ▶ Aktuelle Position des Lifts; auf einem Stockwerk oder zwischen zwei: $pos \in \{S_1, S_2, S_3, S_{1,2}, S_{1,2}\}$
- ▶ Türstatus: $door \in \{\text{open}, \text{closed}\}$
- ▶ Ob sich der Fahrstuhl bewegt: $moving \in \{0, 1\}$
- ▶ Richtung der letzten Bewegung: $dir \in \{\text{up}, \text{down}\}$
- ▶ Türsensor. Meldet, ob Menschen gerade die Fahrstuhltüren durchschritten haben: $sensor \in \{0, 1\}$.
- ▶ Drei Knöpfe $ebut_i$; im Fahrstuhl, welche anzeigen, ob das jeweilige Stockwerk von innerhalb des Fahrstuhls ausgewählt wurde: $ebut_i \in \{0, 1\}$.

Bereitgestellte Informationen/Signale für Spezifikation:

- ▶ Aktuelle Position des Lifts; auf einem Stockwerk oder zwischen zwei: $pos \in \{S_1, S_2, S_3, S_{1,2}, S_{1,2}\}$
- ▶ Türstatus: $door \in \{\text{open}, \text{closed}\}$
- ▶ Ob sich der Fahrstuhl bewegt: $moving \in \{0, 1\}$
- ▶ Richtung der letzten Bewegung: $dir \in \{\text{up}, \text{down}\}$
- ▶ Türsensor. Meldet, ob Menschen gerade die Fahrstuhltüren durchschritten haben: $sensor \in \{0, 1\}$.
- ▶ Drei Knöpfe $ebut_i$ im Fahrstuhl, welche anzeigen, ob das jeweilige Stockwerk von innerhalb des Fahrstuhls ausgewählt wurde: $ebut_i \in \{0, 1\}$.
- ▶ Pro Stockwerk ein Knopf $fbut_i$; zum Rufen des Fahrstuhls. Jeder Knopf hat einen der Zustände $\{\text{on}, \text{blink}, \text{off}\}$

Bereitgestellte Informationen/Signale für Spezifikation:

- ▶ Aktuelle Position des Lifts; auf einem Stockwerk oder zwischen zwei: $\text{pos} \in \{S_1, S_2, S_3, S_{1,2}, S_{1,2}\}$
- ▶ Türstatus: $\text{door} \in \{\text{open}, \text{closed}\}$
- ▶ Ob sich der Fahrstuhl bewegt: $\text{moving} \in \{0, 1\}$
- ▶ Richtung der letzten Bewegung: $\text{dir} \in \{\text{up}, \text{down}\}$
- ▶ Türsensor. Meldet, ob Menschen gerade die Fahrstuhltüren durchschritten haben: $\text{sensor} \in \{0, 1\}$.
- ▶ Drei Knöpfe $e\text{but}_i$ im Fahrstuhl, welche anzeigen, ob das jeweilige Stockwerk von innerhalb des Fahrstuhls ausgewählt wurde: $e\text{but}_i \in \{0, 1\}$.
- ▶ Pro Stockwerk ein Knopf $f\text{but}_i$ zum Rufen des Fahrstuhls. Jeder Knopf hat einen der Zustände $\{\text{on}, \text{blink}, \text{off}\}$
- ▶ Sprechweise: Der Fahrstuhl wird auf Stockwerk i verlangt, falls $f\text{but}_i \neq \text{off} \vee e\text{but}_i = 1$.

Spezifikationen

- ▶ Ist `fbut`; nicht aus, dann blinkt er genau dann, wenn sich der Fahrstuhl bewegt.

Spezifikationen

- ▶ Ist f_{but_i} nicht aus, dann blinkt er genau dann, wenn sich der Fahrstuhl bewegt.
- ▶ Ist f_{but_i} nicht aus, dann erlischt er (höchstens) erst, wenn der Fahrstuhl auf Stockwerk (SW) i hält und die Türen offen sind.

Spezifikationen

- ▶ Ist f_{but_i} nicht aus, dann blinkt er genau dann, wenn sich der Fahrstuhl bewegt.
- ▶ Ist f_{but_i} nicht aus, dann erlischt er (höchstens) erst, wenn der Fahrstuhl auf Stockwerk (SW) i hält und die Türen offen sind.
- ▶ Wenn die Fahrstuhltüren offen sind, dann bleiben sie für mindestens 3 Zeiteinheiten (ZE) geöffnet.

Spezifikationen

- ▶ Ist f_{but_i} nicht aus, dann blinkt er genau dann, wenn sich der Fahrstuhl bewegt.
- ▶ Ist f_{but_i} nicht aus, dann erlischt er (höchstens) erst, wenn der Fahrstuhl auf Stockwerk (SW) i hält und die Türen offen sind.
- ▶ Wenn die Fahrstuhltüren offen sind, dann bleiben sie für mindestens 3 Zeiteinheiten (ZE) geöffnet.
- ▶ Ist die Tür offen, so darf sie erst geschlossen werden, wenn der Sensor für mind. zwei ZE keine Bewegung registriert hat.

Spezifikationen

- ▶ Ist f_{but_i} nicht aus, dann blinkt er genau dann, wenn sich der Fahrstuhl bewegt.
- ▶ Ist f_{but_i} nicht aus, dann erlischt er (höchstens) erst, wenn der Fahrstuhl auf Stockwerk (SW) i hält und die Türen offen sind.
- ▶ Wenn die Fahrstuhltüren offen sind, dann bleiben sie für mindestens 3 Zeiteinheiten (ZE) geöffnet.
- ▶ Ist die Tür offen, so darf sie erst geschlossen werden, wenn der Sensor für mind. zwei ZE keine Bewegung registriert hat.
- ▶ Der Fahrstuhl darf sich nicht bewegen, solange noch die Türen offen sind.

Spezifikationen

- ▶ Ist f_{but} ; nicht aus, dann blinkt er genau dann, wenn sich der Fahrstuhl bewegt.
- ▶ Ist f_{but} ; nicht aus, dann erlischt er (höchstens) erst, wenn der Fahrstuhl auf Stockwerk (SW) i hält und die Türen offen sind.
- ▶ Wenn die Fahrstuhltüren offen sind, dann bleiben sie für mindestens 3 Zeiteinheiten (ZE) geöffnet.
- ▶ Ist die Tür offen, so darf sie erst geschlossen werden, wenn der Sensor für mind. zwei ZE keine Bewegung registriert hat.
- ▶ Der Fahrstuhl darf sich nicht bewegen, solange noch die Türen offen sind.
- ▶ Der Fahrstuhl benötigt mind. zwei ZE, um das nächste SW zu erreichen. D.h. nach einer ZE ist er immer zwischen zwei SW. (Geschwindigkeitsbegrenzung)

Spezifikationen

- ▶ Ist f_{but} ; nicht aus, dann blinkt er genau dann, wenn sich der Fahrstuhl bewegt.
- ▶ Ist f_{but} ; nicht aus, dann erlischt er (höchstens) erst, wenn der Fahrstuhl auf Stockwerk (SW) i hält und die Türen offen sind.
- ▶ Wenn die Fahrstuhltüren offen sind, dann bleiben sie für mindestens 3 Zeiteinheiten (ZE) geöffnet.
- ▶ Ist die Tür offen, so darf sie erst geschlossen werden, wenn der Sensor für mind. zwei ZE keine Bewegung registriert hat.
- ▶ Der Fahrstuhl darf sich nicht bewegen, solange noch die Türen offen sind.
- ▶ Der Fahrstuhl benötigt mind. zwei ZE, um das nächste SW zu erreichen. D.h. nach einer ZE ist er immer zwischen zwei SW. (Geschwindigkeitsbegrenzung)
- ▶ Der Fahrstuhl darf sich erst nach zwei ZE bewegen, nachdem sich die Türen geschlossen haben.

noch mehr Spezifikationen

Bewegung des Fahrstuhls bei geschlossenen Türen

- ▶ Wird der Fahrstuhl auf keinem SW verlangt, so bewegt sich der Fahrstuhl nicht.

noch mehr Spezifikationen

Bewegung des Fahrstuhls bei geschlossenen Türen

- ▶ Wird der Fahrstuhl auf keinem SW verlangt, so bewegt sich der Fahrstuhl nicht.
- ▶ Befindet sich der Fahrstuhl zwischen zwei SW, so behält er seine Richtung bei.

noch mehr Spezifikationen

Bewegung des Fahrstuhls bei geschlossenen Türen

- ▶ Wird der Fahrstuhl auf keinem SW verlangt, so bewegt sich der Fahrstuhl nicht.
- ▶ Befindet sich der Fahrstuhl zwischen zwei SW, so behält er seine Richtung bei.
- ▶ Ist der Fahrstuhl auf dem zweiten SW und wird sowohl auf dem ersten als auch auf dem dritten verlangt, so bewegt er sich entsprechend seiner letzten Richtung weiter.

noch mehr Spezifikationen

Bewegung des Fahrstuhls bei geschlossenen Türen

- ▶ Wird der Fahrstuhl auf keinem SW verlangt, so bewegt sich der Fahrstuhl nicht.
- ▶ Befindet sich der Fahrstuhl zwischen zwei SW, so behält er seine Richtung bei.
- ▶ Ist der Fahrstuhl auf dem zweiten SW und wird sowohl auf dem ersten als auch auf dem dritten verlangt, so bewegt er sich entsprechend seiner letzten Richtung weiter.
- ▶ Ist der Fahrstuhl auf einem 'verlangten' SW, so verlässt er dieses erst, nachdem sich die Türen geöffnet haben.

Modellierung der Fahrstuhlsteuerung in SMV

- ▶ Als nächstes Modellierung der Steuerung in SMV.

Modellierung der Fahrstuhlsteuerung in SMV

- ▶ Als nächstes Modellierung der Steuerung in SMV.
- ▶ **Kein Anspruch auf Korrektheit.**

Modellierung der Fahrstuhlsteuerung in SMV

- ▶ Als nächstes Modellierung der Steuerung in SMV.
- ▶ Kein Anspruch auf Korrektheit.
- ▶ (Versuch) Modellierung der Umgebung, d.h. der Benutzer des Fahrstuhls durch Nicht-Determinismus:

Modellierung der Fahrstuhlsteuerung in SMV

- ▶ Als nächstes Modellierung der Steuerung in SMV.
- ▶ Kein Anspruch auf Korrektheit.
- ▶ (Versuch) Modellierung der Umgebung, d.h. der Benutzer des Fahrstuhls durch Nicht-Determinismus:

Nicht-aktive Knöpfe können spontan aktiv werden.

Modellierung der Fahrstuhlsteuerung in SMV

- ▶ Als nächstes Modellierung der Steuerung in SMV.
- ▶ Kein Anspruch auf Korrektheit.
- ▶ (Versuch) Modellierung der Umgebung, d.h. der Benutzer des Fahrstuhls durch Nicht-Determinismus:
Nicht-aktive Knöpfe können spontan aktiv werden.
Sensor kann spontan bei offenen Türen Bewegung melden.

Modellierung der Fahrstuhlsteuerung in SMV

- ▶ Als nächstes Modellierung der Steuerung in SMV.
- ▶ Kein Anspruch auf Korrektheit.
- ▶ (Versuch) Modellierung der Umgebung, d.h. der Benutzer des Fahrstuhls durch Nicht-Determinismus:
Nicht-aktive Knöpfe können spontan aktiv werden.
Sensor kann spontan bei offenen Türen Bewegung melden.
- ▶ Als erstes Beschreibung der Knöpfe:

Modellierung der Fahrstuhlsteuerung in SMV

- ▶ Als nächstes Modellierung der Steuerung in SMV.
- ▶ Kein Anspruch auf Korrektheit.
- ▶ (Versuch) Modellierung der Umgebung, d.h. der Benutzer des Fahrstuhls durch Nicht-Determinismus:
Nicht-aktive Knöpfe können spontan aktiv werden.
Sensor kann spontan bei offenen Türen Bewegung melden.
- ▶ Als erstes Beschreibung der Knöpfe:
 - ▶ Knöpfe für ein Stockwert ($ebut_i, fbut_i$) werden in Beschreibung des Stockwerks i zusammengefasst.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
    ebut : boolean;
    fbut : { on, blink , off };
ASSIGN
    init(ebut) := 0;
    init(fbut) := off;
    next(ebut) := case
        el.door = closed & next(el.door) = open & el.pos = lvl : 0;
        !ebut                                                    : { 0, 1 };
        1                                                         : ebut;
    esac;
    next(fbut) := case
        fbut = off & el.moving : { blink , off };
        fbut = off & !el.moving : { on, off };
        el.door = closed & next(el.door) = open & el.pos = lvl : off;
        1                                                         : fbut;
    esac;
DEFINE
    request := ebut | !( fbut = off );
```

SMV gliedert sich in MODULE.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : { 0, 1 };
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Hier wird das Verhalten der Stockwerksknöpfe durch Instanzen von 'floor' beschrieben.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                         : {0,1};
    1                                                             : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Jede Instanz bekommt als Parameter das Stockwert, welches sie repräsentieren soll (lvl), und die Fahrstuhlsteuerung el zum Informationsaustausch

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : { 0, 1 };
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Die lokalen Variablen werden in der VAR-Umgebung samt ihrem Wertebereich deklariert.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Jede Instanz speichert in lokalen Variablen die Zustände der Knöpfe.
SMV erlaubt, bel. *endliche* Wertebereiche zu verwenden.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : { 0, 1 };
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Die Initialisierung der Variablen geschieht über `init`. Hier sind also zu Beginn auf jedem Stockwerke keine Knöpfe gedrückt.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Die Übergangsrelation wird mit Hilfe von `next` definiert.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
    ebut : boolean;
    fbut : { on, blink , off };
ASSIGN
    init(ebut) := 0;
    init(fbut) := off;
    next(ebut) := case
        el.door = closed & next(el.door) = open & el.pos = lvl : 0;
        !ebut                                                    : {0,1};
        1                                                         : ebut;
    esac;
    next(fbut) := case
        fbut = off & el.moving : { blink , off };
        fbut = off & !el.moving : { on, off };
        el.door = closed & next(el.door) = open & el.pos = lvl : off;
        1                                                         : fbut;
    esac;
DEFINE
    request := ebut | !( fbut = off );
```

Entsprechend der Spezifikation: Öffnet sich die Türe bis zum nächsten Zustand auf dem Stockwerk lvl, so ist der Knopf deaktiviert im Folgezustand.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Ansonsten kann ebut nicht-deterministisch ($\{0,1\}$) gedrückt werden, falls er nicht schon gedrückt ist.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Ist er schon gedrückt, so behält er seinen Wert.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : { 0, 1 };
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

case wertet dabei die oberste Anweisung (nach ':') aus, deren Bedingung erfüllt ist.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

next(ebut) überschreibt dabei nicht sofort den Wert von ebut.
Dies geschieht erst nach Auswertung der ganzen, globalen Übergangsrelation.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : { 0, 1 };
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Auf die lokalen Variablen von `el`, der Fahrstuhlsteuerung, kann mittels `el.moving`, `el.door` zugegriffen werden.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Für `fbut` wird wieder erlaubt, dass der Knopf nicht-deterministisch gedrückt wird, falls nicht schon geschehen.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Ist er bereits gedrückt, so soll er deaktiviert werden, falls der Fahrstuhl im nächsten Schritt auf dem Stockwerk `lvl` ist und die Türen öffnet.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Ansonsten passt er seinen Zustand dem Fahrstuhl an.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : { 0, 1 };
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Im DEFINE-Bereich können *Macros* definiert werden.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )  
VAR  
    ebut : boolean;  
    fbut : { on, blink , off };  
ASSIGN  
    init(ebut) := 0;  
    init(fbut) := off;  
    next(ebut) := case  
        el.door = closed & next(el.door) = open & el.pos = lvl : 0;  
        !ebut                                                    : { 0 , 1 };  
        1                                                         : ebut;  
    esac;  
    next(fbut) := case  
        fbut = off & el.moving : { blink , off };  
        fbut = off & !el.moving : { on, off };  
        el.door = closed & next(el.door) = open & el.pos = lvl : off;  
        1                                                         : fbut;  
    esac;  
DEFINE  
    request := ebut | !( fbut = off );
```

Diesen könne von außen wie lokale Variablen angesprochen werden.

SMV - Beispiel: Modellierung der Fahrstuhlnutzer / Knöpfe

```
MODULE floor ( lvl , el )
VAR
  ebut : boolean;
  fbut : { on, blink , off };
ASSIGN
  init(ebut) := 0;
  init(fbut) := off;
  next(ebut) := case
    el.door = closed & next(el.door) = open & el.pos = lvl : 0;
    !ebut                                                    : {0,1};
    1                                                         : ebut;
  esac;
  next(fbut) := case
    fbut = off & el.moving : { blink , off };
    fbut = off & !el.moving : { on, off };
    el.door = closed & next(el.door) = open & el.pos = lvl : off;
    1                                                         : fbut;
  esac;
DEFINE
  request := ebut | !( fbut = off );
```

Ist `s1` eine Instanz von `floor`, so kann die Fahrstuhlsteuerung über `s1.request` prüfen, ob das entsprechende Stockwerk angefahren werden muss.

Spezifikation der Fahrstuhlsteuerung

```
MODULE elevator ( S1req , S2req , S3req )  
VAR  
  noSensorTime      : { 0, 1, 2 };  
  doorClosedTime    : { 0, 1 };  
  pos                : { S1, S2, S3, S12, S23 };  
  door               : { open, closed };  
  dir                : { up, down };  
  moving             : boolean;  
  sensor             : boolean;  
ASSIGN  
  init(pos)          := S1;  
  init(door)         := closed;  
  init(dir)          := up;  
  init(moving)       := 0;  
  init(sensor)       := 0;  
— Fortsetzung folgt ...
```

Die Fahrstuhlsteuerung bekommt von außen die Information, welche Stockwerke der Fahrstuhl anfahren soll: S1req,S2req,S3req.

Spezifikation der Fahrstuhlsteuerung

```
MODULE elevator ( S1req , S2req , S3req )  
VAR  
  noSensorTime      : { 0, 1, 2 };  
  doorClosedTime    : { 0, 1 };  
  pos                : { S1, S2, S3, S12, S23 };  
  door               : { open, closed };  
  dir                : { up, down };  
  moving             : boolean;  
  sensor             : boolean;
```

```
ASSIGN  
  init(pos)          := S1;  
  init(door)         := closed;  
  init(dir)          := up;  
  init(moving)       := 0;  
  init(sensor)       := 0;
```

— *Fortsetzung folgt ...*

Zusätzlich zu den spezifizierten Variablen `pos`, `door`, `dir`, `moving`, `sensor` noch zwei Zählervariablen, um zu messen, wie lange kein Mensch die Fahrstuhltüren durchschritten hat bzw. die Tür bereits geschlossen ist.

Spezifikation der Fahrstuhlsteuerung

```
MODULE elevator ( S1req , S2req , S3req )  
VAR  
  noSensorTime      : { 0, 1, 2 };  
  doorClosedTime    : { 0, 1 };  
  pos                : { S1, S2, S3, S12, S23 };  
  door                : { open, closed };  
  dir                 : { up, down };  
  moving              : boolean;  
  sensor              : boolean;
```

```
ASSIGN  
  init(pos)          := S1;  
  init(door)          := closed;  
  init(dir)            := up;  
  init(moving)         := 0;  
  init(sensor)         := 0;
```

— *Fortsetzung folgt ...*

Zu Beginn befindet sich der Fahrstuhl auf Stockwerk 1, die Türen sind geschlossen, der Fahrstuhl hat sich zuletzt nach oben bewegt, ...

Spezifikation der Fahrstuhlsteuerung

```
MODULE elevator ( S1req , S2req , S3req )  
VAR  
  noSensorTime      : { 0, 1, 2 };  
  doorClosedTime    : { 0, 1 };  
  pos                : { S1, S2, S3, S12, S23 };  
  door               : { open, closed };  
  dir                : { up, down };  
  moving             : boolean;  
  sensor             : boolean;
```

```
ASSIGN  
  init(pos)          := S1;  
  init(door)         := closed;  
  init(dir)          := up;  
  init(moving)       := 0;  
  init(sensor)       := 0;
```

— *Fortsetzung folgt ...*

Wären keine Anfangswerte spezifiziert, dann würde SMV die Wert nicht-deterministisch wählen, d.h. von allen möglichen Anfangswerten ausgehen.

Übergangsrelation - dir,moving,sensor

```
next(dir) := case
  pos = S1 & next(pos) = S12 : up;
  pos = S2 & next(pos) = S23 : up;
  pos = S2 & next(pos) = S12 : down;
  pos = S3 & next(pos) = S23 : down;
  1 : dir;
esac;
next(moving) := !( pos = next(pos) );
next(sensor) := case
  door = open & next(door) = open : {0,1};
  1 : 0;
esac;
```

Verändert der Fahrstuhl seine Position, so wird die Bewegungsrichtung entsprechend angepasst.

Übergangsrelation - dir,moving,sensor

```
next(dir) := case
  pos = S1 & next(pos) = S12 : up;
  pos = S2 & next(pos) = S23 : up;
  pos = S2 & next(pos) = S12 : down;
  pos = S3 & next(pos) = S23 : down;
  1 : dir;
esac;
next(moving) := !( pos = next(pos) );
next(sensor) := case
  door = open & next(door) = open : {0,1};
  1 : 0;
esac;
```

Der Fahrstuhl bewegt sich, falls sich seine Positionen im aktuellen und im folgenden Zustand unterscheiden.

Übergangsrelation - dir,moving,sensor

```
next(dir) := case
  pos = S1 & next(pos) = S12 : up;
  pos = S2 & next(pos) = S23 : up;
  pos = S2 & next(pos) = S12 : down;
  pos = S3 & next(pos) = S23 : down;
  1 : dir;
esac;
next(moving) := !( pos = next(pos) );
next(sensor) := case
  door = open & next(door) = open : {0,1};
  1 : 0;
esac;
```

Um das Betreten des Fahrstuhls (bei offener Tür) zu modellieren, wird wieder der Nicht-Determinismus benutzt.

Übergangsrelation - dir,moving,sensor

```
next(dir) := case
  pos = S1 & next(pos) = S12 : up;
  pos = S2 & next(pos) = S23 : up;
  pos = S2 & next(pos) = S12 : down;
  pos = S3 & next(pos) = S23 : down;
  1 : dir;
esac;
next(moving) := !( pos = next(pos) );
next(sensor) := case
  door = open & next(door) = open : {0,1};
  1 : 0;
esac;
```

Eine Person durchquert die Tür dabei nur, falls sie sich nicht schließt,
d.h. die Türe ist jetzt und im Folgezustand offen:

`door = open & next(door) = open`

Übergangsrelation - dir,moving,sensor

```
next(dir) := case
  pos = S1 & next(pos) = S12 : up;
  pos = S2 & next(pos) = S23 : up;
  pos = S2 & next(pos) = S12 : down;
  pos = S3 & next(pos) = S23 : down;
  1 : dir;
esac;
next(moving) := !( pos = next(pos) );
next(sensor) := case
  door = open & next(door) = open : {0,1};
  1 : 0;
esac;
```

Entsprechend könnte `door = closed & next(door) = open` als Öffnen der Tür interpretiert werden und `door = open & next(door) = closed` als das Schließen.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

Ist die Tür offen und im aktuellen Zeitschritt betritt niemand den Fahrstuhl, dann wird der Zähler noSensorTime erhöht.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

Ansonsten ist noSensorTime immer gleich 0.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

Ist die Tür geschlossen, der Fahrstuhl bewegt sich nicht und befindet sich auf einem angeforderten Stockwerk, so öffnen sich die Türen.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

Erinnerung: Ist moving wahr, so folgt, dass der Fahrstuhl bereits im letzten Zeitschritt auf dem aktuellen Stockwerk war.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

Sind die Türen offen, in den letzten beiden Zuständen hat niemand den Fahrstuhl betreten und durchschreitet auch jetzt niemand die Tür, so schließen sich die Türen.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

Ansonsten bleiben die Türen unverändert.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

doorClosedTime gibt nur an, ob im letzten Schritt die Tür geschlossen war.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

noSensorTime und doorClosed modellieren somit ein *Gedächtnis* für die Fahrstuhlsteuerung.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

Solche 'Gedächtnisse' oft wichtig, um sicherzustellen, dass jeder Prozess, der einen Bereich betreten will, dies auch irgendwann (in endl. Zeit) erreicht.

Übergangsrelation - noSensorTime,door,doorClosedTime

```
next(noSensorTime) := case
  door = open & !sensor & noSensorTime = 0 : 1;
  door = open & !sensor & noSensorTime = 1 : 2;
  1 : 0;
esac;
next(door) := case
  door = closed & !moving & ( pos = S1 & S1req
                              | pos = S2 & S2req
                              | pos = S3 & S3req ) : open;
  door = open & noSensorTime = 2 & !sensor : closed;
  1 : door;
esac;
next(doorClosedTime) := case
  door = closed : 1;
  1 : 0;
esac;
```

Z.B. sollte der Fahrstuhl irgendwann jedes Stockwerk erreichen, von welchem er angefordert wird, und dort die Türen öffnen.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down        : S1;
  pos = S23 & dir = up          : S3;
  pos = S23 & dir = down        : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req               : pos;
  pos = S2 & S2req               : pos;
  pos = S3 & S3req               : pos;
  pos = S2 & dir = up & S3req    : S23;
  pos = S2 & dir = down & S1req  : S12;
  pos = S2 & S1req               : S12;
  pos = S2 & S3req               : S23;
  pos = S1                       : S12;
  pos = S3                       : S23;
  1                               : pos;
esac;
```

Die ersten vier Fälle stellen sicher (?), dass der Fahrstuhl seine Bewegungsrichtung beibehält.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down         : S1;
  pos = S23 & dir = up           : S3;
  pos = S23 & dir = down         : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req                : pos;
  pos = S2 & S2req                : pos;
  pos = S3 & S3req                : pos;
  pos = S2 & dir = up & S3req     : S23;
  pos = S2 & dir = down & S1req   : S12;
  pos = S2 & S1req                : S12;
  pos = S2 & S3req                : S23;
  pos = S1                        : S12;
  pos = S3                        : S23;
  1                                : pos;
esac;
```

Ist die Tür jetzt offen, bzw. im letzten Zustand geöffnet gewesen, oder wird offen sein, so bewegt sich der Fahrstuhl nicht.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down        : S1;
  pos = S23 & dir = up          : S3;
  pos = S23 & dir = down        : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req               : pos;
  pos = S2 & S2req               : pos;
  pos = S3 & S3req               : pos;
  pos = S2 & dir = up & S3req    : S23;
  pos = S2 & dir = down & S1req  : S12;
  pos = S2 & S1req               : S12;
  pos = S2 & S3req               : S23;
  pos = S1                       : S12;
  pos = S3                       : S23;
  1                               : pos;
esac;
```

Erinnerung: Nach Semantik von case
gilt also für alle folgenden Bedingungen
door = closed & doorClosedTime = 1 & next(door) = 1.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down        : S1;
  pos = S23 & dir = up          : S3;
  pos = S23 & dir = down        : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req                : pos;
  pos = S2 & S2req                : pos;
  pos = S3 & S3req                : pos;
  pos = S2 & dir = up & S3req     : S23;
  pos = S2 & dir = down & S1req  : S12;
  pos = S2 & S1req                : S12;
  pos = S2 & S3req                : S23;
  pos = S1                        : S12;
  pos = S3                        : S23;
  1                               : pos;
esac;
```

Liegen keine Anforderungen vor oder wird der FS auf dem aktuellen SW noch benötigt, so behält er ebenfalls seine Position bei.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down        : S1;
  pos = S23 & dir = up          : S3;
  pos = S23 & dir = down        : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req               : pos;
  pos = S2 & S2req               : pos;
  pos = S3 & S3req               : pos;
  pos = S2 & dir = up & S3req    : S23;
  pos = S2 & dir = down & S1req  : S12;
  pos = S2 & S1req               : S12;
  pos = S2 & S3req               : S23;
  pos = S1                       : S12;
  pos = S3                       : S23;
  1                               : pos;
esac;
```

Ab hier gilt entsprechend für alle nachfolgenden Bedingungen, dass mind. eine Anforderung vorliegt.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down         : S1;
  pos = S23 & dir = up           : S3;
  pos = S23 & dir = down         : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req                : pos;
  pos = S2 & S2req                : pos;
  pos = S3 & S3req                : pos;
  pos = S2 & dir = up & S3req     : S23;
  pos = S2 & dir = down & S1req  : S12;
  pos = S2 & S1req                : S12;
  pos = S2 & S3req                : S23;
  pos = S1                        : S12;
  pos = S3                        : S23;
  1                                : pos;
esac;
```

Zunächst werden die Fälle behandelt, dass Fahrstuhl auf dem aktuellen Stockwerk bleibt, da er noch angefordert wird.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down        : S1;
  pos = S23 & dir = up          : S3;
  pos = S23 & dir = down        : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req                : pos;
  pos = S2 & S2req                : pos;
  pos = S3 & S3req                : pos;
  pos = S2 & dir = up & S3req     : S23;
  pos = S2 & dir = down & S1req   : S12;
  pos = S2 & S1req                : S12;
  pos = S2 & S3req                : S23;
  pos = S1                        : S12;
  pos = S3                        : S23;
  1                                : pos;
esac;
```

Für die folgenden Fälle wird also der Fahrstuhl nicht vom aktuellen Stockwerk angefordert, d.h. er muss sich bewegen.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down        : S1;
  pos = S23 & dir = up          : S3;
  pos = S23 & dir = down        : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req               : pos;
  pos = S2 & S2req               : pos;
  pos = S3 & S3req               : pos;
  pos = S2 & dir = up & S3req    : S23;
  pos = S2 & dir = down & S1req  : S12;
  pos = S2 & S1req               : S12;
  pos = S2 & S3req               : S23;
  pos = S1                       : S12;
  pos = S3                       : S23;
  1                               : pos;
esac;
```

Entsprechend der Spezifikation soll die letzte Bewegungsrichtung bevorzugt werden, falls der Fahrstuhl das mittlere Stockwerk verlässt und sowohl vom ersten als auch vom dritten Stockwerk angefordert wird.

Übergangsrelation - dir

```
next(pos) := case
  pos = S12 & dir = up           : S2;
  pos = S12 & dir = down        : S1;
  pos = S23 & dir = up          : S3;
  pos = S23 & dir = down        : S2;
  door = open | doorClosedTime = 0
                                | next(door) = open : pos;
  !S1req & !S2req & !S3req      : pos;
  pos = S1 & S1req                : pos;
  pos = S2 & S2req                : pos;
  pos = S3 & S3req                : pos;
  pos = S2 & dir = up & S3req     : S23;
  pos = S2 & dir = down & S1req   : S12;
  pos = S2 & S1req                : S12;
  pos = S2 & S3req                : S23;
  pos = S1                        : S12;
  pos = S3                        : S23;
  1                                : pos;
esac;
```

Ansonsten ist die nächste Position eindeutig vorbestimmt.

Instanzieren/Zusammenfügen

```
MODULE main
VAR
  s1 : floor( S1, el );
  s2 : floor( S2, el );
  s3 : floor( S3, el );
  el : elevator( s1.request , s2.request , s3.request );
```

Das Instanzieren und Verknüpfen der Prozesse zum endgültigen Transitionssystem geschieht in MODULE main.

Instanziieren/Zusammenfügen

```
MODULE main
```

```
VAR
```

```
  s1 : floor( S1, el );
```

```
  s2 : floor( S2, el );
```

```
  s3 : floor( S3, el );
```

```
  el : elevator( s1.request , s2.request , s3.request );
```

Jede Instanz von `floor` bekommt genau ein Stockwerk über den Parameter `lv1` und die Fahrstuhlsteuerung zu gewiesen.

Instanziieren/Zusammenfügen

```
MODULE main
```

```
VAR
```

```
  s1 : floor( S1, el );
```

```
  s2 : floor( S2, el );
```

```
  s3 : floor( S3, el );
```

```
  el : elevator( s1.request , s2.request , s3.request );
```

Entsprechend bekommt die Fahrstuhlsteuerung die request-Signale der drei Stockwerke.

Instanziieren/Zusammenfügen

```
MODULE main
```

```
VAR
```

```
  s1 : floor( S1, el );
```

```
  s2 : floor( S2, el );
```

```
  s3 : floor( S3, el );
```

```
  el : elevator( s1.request , s2.request , s3.request );
```

Hinweis: In jeder SMV-Beschreibung muss immer genau ein **MODULE main** existieren.

Instanziieren/Zusammenfügen

```
MODULE main
VAR
  s1 : floor( S1, el );
  s2 : floor( S2, el );
  s3 : floor( S3, el );
  el : elevator( s1.request , s2.request , s3.request );
```

Die Prozess können auch mit dem Schlüsselwort `process` instanziiert werden, z.B. `s1 : process floor(S1, el);`. Dann werden die Übergangsrelationen nicht *synchron* ($\bigwedge \mathcal{R}_i$), d.h. *asynchron* ($\bigvee \mathcal{R}_i$) ausgewertet.

Instanziieren/Zusammenfügen

```
MODULE main
```

```
VAR
```

```
  s1 : floor( S1, el );
```

```
  s2 : floor( S2, el );
```

```
  s3 : floor( S3, el );
```

```
  el : elevator( s1.request , s2.request , s3.request );
```

Vergleiche auch Mutex-Beispiel von Folien aus WS03/04.

Überprüfen des Modells

- ▶ Spezifikation verlangt:

Überprüfen des Modells

- ▶ Spezifikation verlangt:
- ▶ Ist f_{but} eines Stockwerks aktiviert, so bleibt dieser aktiviert, bis schließlich der Fahrstuhl auf dem Stockwerk hält und die Türen sich öffnen.

Überprüfen des Modells

- ▶ Spezifikation verlangt:
- ▶ Ist fbut eines Stockwerks aktiviert, so bleibt dieser aktiviert, bis schließlich der Fahrstuhl auf dem Stockwerk hält und die Türen sich öffnen.
- ▶ In CTL (SMV):

SPEC

```
AG(!(s1.fbut=off)->A[(!(s1.fbut=off))U((el.pos=S1 )
&(el.door=open))])
&AG(!(s2.fbut=off)->A[(!(s2.fbut=off))U((el.pos=S2 )
&(el.door=open))])
&AG(!(s3.fbut=off)->A[(!(s3.fbut=off))U((el.pos=S3 )
&(el.door=open))])
```

Überprüfen des Modells

- ▶ Spezifikation verlangt:
- ▶ Ist `fbut` eines Stockwerks aktiviert, so bleibt dieser aktiviert, bis schließlich der Fahrstuhl auf dem Stockwerk hält und die Türen sich öffnen.
- ▶ In CTL (SMV):

SPEC

```
AG(!(s1.fbut=off)->A[(!(s1.fbut=off))U((el.pos=S1 )
&(el.door=open))])
&AG(!(s2.fbut=off)->A[(!(s2.fbut=off))U((el.pos=S2 )
&(el.door=open))])
&AG(!(s3.fbut=off)->A[(!(s3.fbut=off))U((el.pos=S3 )
&(el.door=open))])
```

- ▶ Syntax von **AU** beachten: $A[\phi U \psi]$

Gegenbeispiele von SMV

- ▶ Ist eine Spezifikation nicht erfüllt, so liefert SMV ein Gegenbeispiel in Form einer Folge von Zuständen.

Gegenbeispiele von SMV

- ▶ Ist eine Spezifikation nicht erfüllt, so liefert SMV ein Gegenbeispiel in Form einer Folge von Zuständen.
- ▶ SMV gibt für Folgezustände dabei nur die veränderten Variablen an.

Gegenbeispiele von SMV

- ▶ Ist eine Spezifikation nicht erfüllt, so liefert SMV ein Gegenbeispiel in Form einer Folge von Zuständen.
- ▶ SMV gibt für Folgezustände dabei nur die veränderten Variablen an.
- ▶ **Problematisch: Aktuelle Modellierung erlaubt es, dass der Fahrstuhl auf einem Stockwerk stecken bleibt, weil:**

Gegenbeispiele von SMV

- ▶ Ist eine Spezifikation nicht erfüllt, so liefert SMV ein Gegenbeispiel in Form einer Folge von Zuständen.
- ▶ SMV gibt für Folgezustände dabei nur die veränderten Variablen an.
- ▶ Problematisch: Aktuelle Modellierung erlaubt es, dass der Fahrstuhl auf einem Stockwerk stecken bleibt, weil:
 - ▶ in zu kurzen Zeitintervallen der Rufknopf gedrückt wird

Gegenbeispiele von SMV

- ▶ Ist eine Spezifikation nicht erfüllt, so liefert SMV ein Gegenbeispiel in Form einer Folge von Zuständen.
- ▶ SMV gibt für Folgezustände dabei nur die veränderten Variablen an.
- ▶ Problematisch: Aktuelle Modellierung erlaubt es, dass der Fahrstuhl auf einem Stockwerk stecken bleibt, weil:
 - ▶ in zu kurzen Zeitintervallen der Rufknopf gedrückt wird
 - ▶ oder immer wieder der Sensor ausgelöst wird.

Gegenbeispiele von SMV

- ▶ Ist eine Spezifikation nicht erfüllt, so liefert SMV ein Gegenbeispiel in Form einer Folge von Zuständen.
- ▶ SMV gibt für Folgezustände dabei nur die veränderten Variablen an.
- ▶ Problematisch: Aktuelle Modellierung erlaubt es, dass der Fahrstuhl auf einem Stockwerk stecken bleibt, weil:
 - ▶ in zu kurzen Zeitintervallen der Rufknopf gedrückt wird
 - ▶ oder immer wieder der Sensor ausgelöst wird.
- ▶ 'Unfares' Verhalten der (Modell)-Fahrstuhlbenutzer.
(EF EG s1.request ist z.B. erfüllt.)

Gegenbeispiele von SMV

- ▶ Ist eine Spezifikation nicht erfüllt, so liefert SMV ein Gegenbeispiel in Form einer Folge von Zuständen.
- ▶ SMV gibt für Folgezustände dabei nur die veränderten Variablen an.
- ▶ Problematisch: Aktuelle Modellierung erlaubt es, dass der Fahrstuhl auf einem Stockwerk stecken bleibt, weil:
 - ▶ in zu kurzen Zeitintervallen der Rufknopf gedrückt wird
 - ▶ oder immer wieder der Sensor ausgelöst wird.
- ▶ 'Unfares' Verhalten der (Modell)-Fahrstuhlbenutzer. (EF EG s1.request ist z.B. erfüllt.)
- ▶ Mittels FAIRNESS 'CTL-Formel ϕ ' kann SMV veranlasst werden, die Spezifikationen nur auf Pfaden zu betrachten, auf welchen die angegebene *Fairness*-Bedingung ϕ immer wieder gilt. (AG AF ϕ)

Fairness

▶ Beispiel:

Fairness

- ▶ Beispiel:

Es sollen nur Pfade betrachtet werden, auf welchen nicht andauernd der Fahrstuhl von Stockwert 1 bzw. 2 bzw. 3 angefordert wird

Fairness

► Beispiel:

Es sollen nur Pfade betrachtet werden, auf welchen nicht andauernd der Fahrstuhl von Stockwert 1 bzw. 2 bzw. 3 angefordert wird

und auf welchen der Sensor auch nicht ständig aktiv ist.

FAIRNESS !s1.request

FAIRNESS !s2.request

FAIRNESS !s3.request

FAIRNESS (!el.sensor & **AX** !el.sensor)

Fairness

- ▶ Beispiel:

Es sollen nur Pfade betrachtet werden, auf welchen nicht andauernd der Fahrstuhl von Stockwert 1 bzw. 2 bzw. 3 angefordert wird
und auf welchen der Sensor auch nicht ständig aktiv ist.

FAIRNESS !s1.request

FAIRNESS !s2.request

FAIRNESS !s3.request

FAIRNESS (!el.sensor & **AX** !el.sensor)

- ▶ Jetzt ist EF EG s1.request nicht mehr erfüllt.

Fairness

- ▶ Beispiel:

Es sollen nur Pfade betrachtet werden, auf welchen nicht andauernd der Fahrstuhl von Stockwert 1 bzw. 2 bzw. 3 angefordert wird
und auf welchen der Sensor auch nicht ständig aktiv ist.

```
FAIRNESS !s1.request
```

```
FAIRNESS !s2.request
```

```
FAIRNESS !s3.request
```

```
FAIRNESS ( !el.sensor & AX !el.sensor )
```

- ▶ Jetzt ist EF EG s1.request nicht mehr erfüllt.

- ▶ **ABER:** Es werden auch die Pfade nicht mehr betrachtet, in welchen einmal S1 verlangt wird, der Fahrstuhl aber nie S1 erreicht und die Türen öffnet !!!

Fairness

- ▶ Beispiel:

Es sollen nur Pfade betrachtet werden, auf welchen nicht andauernd der Fahrstuhl von Stockwert 1 bzw. 2 bzw. 3 angefordert wird
und auf welchen der Sensor auch nicht ständig aktiv ist.

```
FAIRNESS !s1.request  
FAIRNESS !s2.request  
FAIRNESS !s3.request  
FAIRNESS ( !el.sensor & AX !el.sensor )
```

- ▶ Jetzt ist EF EG s1.request nicht mehr erfüllt.
- ▶ ABER: Es werden auch die Pfade nicht mehr betrachtet, in welchen einmal S1 verlangt wird, der Fahrstuhl aber nie S1 erreicht und die Türen öffnet !!!
- ▶ **Besitzt die Fahrstuhlsteuerung diese Eigenschaft also nicht, dann kann sie unter diesen Fairness-Anforderung dennoch verifiziert werden.**

Fairness

- ▶ Beispiel:

Es sollen nur Pfade betrachtet werden, auf welchen nicht andauernd der Fahrstuhl von Stockwert 1 bzw. 2 bzw. 3 angefordert wird
und auf welchen der Sensor auch nicht ständig aktiv ist.

```
FAIRNESS !s1.request  
FAIRNESS !s2.request  
FAIRNESS !s3.request  
FAIRNESS ( !el.sensor & AX !el.sensor )
```

- ▶ Jetzt ist $EF \ EG \ s1.request$ nicht mehr erfüllt.
- ▶ ABER: Es werden auch die Pfade nicht mehr betrachtet, in welchen einmal S1 verlangt wird, der Fahrstuhl aber nie S1 erreicht und die Türen öffnet !!!
- ▶ Besitzt die Fahrstuhlsteuerung diese Eigenschaft also nicht, dann kann sie unter diesen Fairness-Anforderung dennoch verifiziert werden.

Also besser Modellierung 'fairer' Fahrstuhlbenutzer abändern

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit.

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit. Z.B. sollte folgende Bedingung nicht erfüllt sein:

SPEC

```
AG( !( s1.fbut = off ) -> ( s1.fbut=blink <-> el.moving )
    & !( s2.fbut = off ) -> ( s2.fbut=blink <-> el.moving )
    & !( s3.fbut = off ) -> ( s3.fbut=blink <-> el.moving )
    )
```

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit. Z.B. sollte folgende Bedingung nicht erfüllt sein:

SPEC

```
AG( !( s1.fbut = off ) -> ( s1.fbut=blink <-> el.moving )
    & !( s2.fbut = off ) -> ( s2.fbut=blink <-> el.moving )
    & !( s3.fbut = off ) -> ( s3.fbut=blink <-> el.moving )
)
```

Zwei Fehler:

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit. Z.B. sollte folgende Bedingung nicht erfüllt sein:

SPEC

```
AG( !( s1.fbut = off ) -> ( s1.fbut=blink <-> el.moving )
    & !( s2.fbut = off ) -> ( s2.fbut=blink <-> el.moving )
    & !( s3.fbut = off ) -> ( s3.fbut=blink <-> el.moving )
)
```

Zwei Fehler:

- ▶ Knopf passt sich nicht nachträglich an Bewegung/Halten des Fahrstuhls an.

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit. Z.B. sollte folgende Bedingung nicht erfüllt sein:

SPEC

```
AG( !( s1.fbut = off ) -> ( s1.fbut=blink <-> e1.moving )  
    & !( s2.fbut = off ) -> ( s2.fbut=blink <-> e1.moving )  
    & !( s3.fbut = off ) -> ( s3.fbut=blink <-> e1.moving )  
    )
```

Zwei Fehler:

- ▶ Knopf passt sich nicht nachträglich an Bewegung/Halten des Fahrstuhls an.
- ▶ Und die ersten beiden Bedingungen beziehen sich auf gegenwärtiges `e1.moving`.

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit. Z.B. sollte folgende Bedingung nicht erfüllt sein:

SPEC

```
AG( !( s1.fbut = off ) -> ( s1.fbut=blink <-> e1.moving )  
    & !( s2.fbut = off ) -> ( s2.fbut=blink <-> e1.moving )  
    & !( s3.fbut = off ) -> ( s3.fbut=blink <-> e1.moving )  
    )
```

Zwei Fehler:

- ▶ Knopf passt sich nicht nachträglich an Bewegung/Halten des Fahrstuhls an.
- ▶ Und die ersten beiden Bedingungen beziehen sich auf gegenwärtiges `e1.moving`.
- ▶ Es wird jedoch der Verhalten von nächsten, zukünftigen Zustand beschrieben (\rightsquigarrow `next(e1.moving)`)

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit. Z.B. sollte folgende Bedingung nicht erfüllt sein:

SPEC

```
AG( !( s1.fbut = off ) -> ( s1.fbut=blink <-> e1.moving )
    & !( s2.fbut = off ) -> ( s2.fbut=blink <-> e1.moving )
    & !( s3.fbut = off ) -> ( s3.fbut=blink <-> e1.moving )
)
```

Zwei Fehler:

- ▶ Knopf passt sich nicht nachträglich an Bewegung/Halten des Fahrstuhls an.
 - ▶ Und die ersten beiden Bedingungen beziehen sich auf gegenwärtiges `e1.moving`.
 - ▶ Es wird jedoch der Verhalten von nächsten, zukünftigen Zustand beschrieben (\rightsquigarrow `next(e1.moving)`)
- ▶ Freiwillige Aufgabe:

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit. Z.B. sollte folgende Bedingung nicht erfüllt sein:

SPEC

```
AG( !( s1.fbut = off ) -> ( s1.fbut=blink <-> e1.moving )
    & !( s2.fbut = off ) -> ( s2.fbut=blink <-> e1.moving )
    & !( s3.fbut = off ) -> ( s3.fbut=blink <-> e1.moving )
)
```

Zwei Fehler:

- ▶ Knopf passt sich nicht nachträglich an Bewegung/Halten des Fahrstuhls an.
 - ▶ Und die ersten beiden Bedingungen beziehen sich auf gegenwärtiges `e1.moving`.
 - ▶ Es wird jedoch der Verhalten von nächsten, zukünftigen Zustand beschrieben (\rightsquigarrow `next(e1.moving)`)
- ▶ Freiwillige Aufgabe:
Restliche Spezifikationen in CTL übersetzen

Ende

- ▶ Folien und vollständige SMV-Beschreibung auf Übungsseite zum Runterladen.
- ▶ SMV-Beschreibung erhebt *NICHT* Anspruch auf Korrektheit. Z.B. sollte folgende Bedingung nicht erfüllt sein:

SPEC

```
AG( !( s1.fbut = off ) -> ( s1.fbut=blink <-> e1.moving )
    & !( s2.fbut = off ) -> ( s2.fbut=blink <-> e1.moving )
    & !( s3.fbut = off ) -> ( s3.fbut=blink <-> e1.moving )
)
```

Zwei Fehler:

- ▶ Knopf passt sich nicht nachträglich an Bewegung/Halten des Fahrstuhls an.
 - ▶ Und die ersten beiden Bedingungen beziehen sich auf gegenwärtiges `e1.moving`.
 - ▶ Es wird jedoch der Verhalten von nächsten, zukünftigen Zustand beschrieben (\rightsquigarrow `next(e1.moving)`)
- ▶ Freiwillige Aufgabe:
Restliche Spezifikationen in CTL übersetzen und Beschreibung damit testen und notfalls anpassen.