Spezifikation und Verfeinerung mit ASMs

Rüdiger Grammes

July 10, 2003

Gliederung

- Abstract State Machines
- Modell des Systems
- Präzisierung der Anforderungen
- Verfeinerung des Modells
- Verifikation der Verfeinerung

Statische Algebren (States)

Statische Algebren sind mehrsortige Strukturen mit partiellen Operationen. Eine statische Algebra S besteht aus:

- Vokabular Υ von S: Endliche Menge von Funktions- und Relationsnamen mit fester Stelligkeit
  - statische Funktionsnamen $true$, $false$, $undef$, $=$, $boolesche\ Operatoren$
- Superuniversum X von S
- Interpretation von Funktionsnamen von Υ in S
- Spezielle Domänen: $\overline{x} \in f$ gdw. $f(\overline{x}) = \mathrm{true}$ in S

Funktionen

Klassifikation der Funktionen:

- statisch/dynamisch: statische Funktionen können nicht verändert werden (z.B. boolesche Funktionen)
- normal/abgeleitet: abgeleitete Funktionen werden durch andere Funktionen definiert
- controlled/monitored/shared
  - controlled: können nur von der abstrakten Maschine verändert werden
  - monitored: können nur von der Umgebung geändert werden
  - shared: können sowohl von der abstrakten Maschine und der Umgebung verändert werden

Locations und Updates

Location: Paar $(f,\overline{x})$ mit

- f: nicht-statischer Funktionsname aus Υ
- $\overline{x}$: Tupel von Elementen aus S

Update: Paar $\alpha = (l,y)$ mit

- Location l von S $(\mathrm{Loc}(\alpha))$
- $y \in S$

Feuern von (l,y): $S'(l) = y$ (mit $S : \mathrm{Loc}(S) \to S$)

Update Sets

Update Set β: Menge von Updates

- $\mathrm{Loc}(\beta) = \{\mathrm{Loc}(\alpha) : \alpha \in \beta\}$
- Konsistenz: $\mathrm{Val}_\beta(l) = \{\mathrm{Val}(\alpha) : \alpha \in \beta \ \wedge \mathrm{Loc}(\alpha) = l\}$ ist einelementig für alle $l \in \mathrm{Loc}(\beta)$

Feuern eines Update Sets:

- Konsistentes Update Set: Feuern aller Elemente gleichzeitig
- Inkonsistentes Update Set: $S' = S$

Regeln

Elementare Transitionsregeln:

Regeln

Elementare Transitionsregeln:

- Update: $R = f(\bar{t}) := t_0$

  Updates(R,S) $= \{(l, \mathrm{val}_S(t_0))\}$ mit $l = (f, \mathrm{val}_S(\bar{t}))$

Regeln

Elementare Transitionsregeln:

- Update: $R = f(\bar{t}) := t_0$

  Updates(R,S) $= \{(l, \mathrm{val}_S(t_0))\}$ mit $l = (f, \mathrm{val}_S(\bar{t}))$

- Block: $R = R_1 \ldots R_k$

  Updates(R,S) $=$ Updates(R_1,S) $\cup \ldots \cup$ Updates(R_k,S)

Regeln

Elementare Transitionsregeln:

- Update: $R = f(\bar{t}) := t_0$

  Updates(R,S) $= \{(l, \mathrm{val}_S(t_0))\}$ mit $l = (f, \mathrm{val}_S(\bar{t}))$

- Block: $R = R_1 \ldots R_k$

  Updates(R,S) $=$ Updates(R_1,S) $\cup \ldots \cup$ Updates(R_k,S)

- Guard: $R =$ **if** g **then** R_0 **endif**

  Updates(R,S) $=$ Updates(R_0,S) wenn g in S gilt, $\emptyset$ sonst.

Import von Elementen

Spezielles Universum $Reserve$. Für jedes $\overline{x}$ dass ein Element der Reserve enthält:

- $f(\overline{x}) = \text{false}$ wenn f Relation
- $f(\overline{x}) = \text{undefined}$ wenn f Funktion
- Keine Funktion hat ein Element der Reserve im Wertebereich

Erweitern eines Universums U mit Elementen der Reserve:

extend U **with** $v_1, \ldots, v_n$
$R(v_1, \ldots, v_n)$
endextend

Nichtdeterminismus

R =

choose $v \in U$ **where** $g(v)$
  $R_0(v)$
endchoose

- Belege v mit einem beliebigen Element aus U
- Betrachte nur Elemente für die g(v) gilt
- Auswahl aus einer leeren Menge: Updates(R,S) $= \emptyset$

Läufe

Programm P: Regel ohne freie Variablen

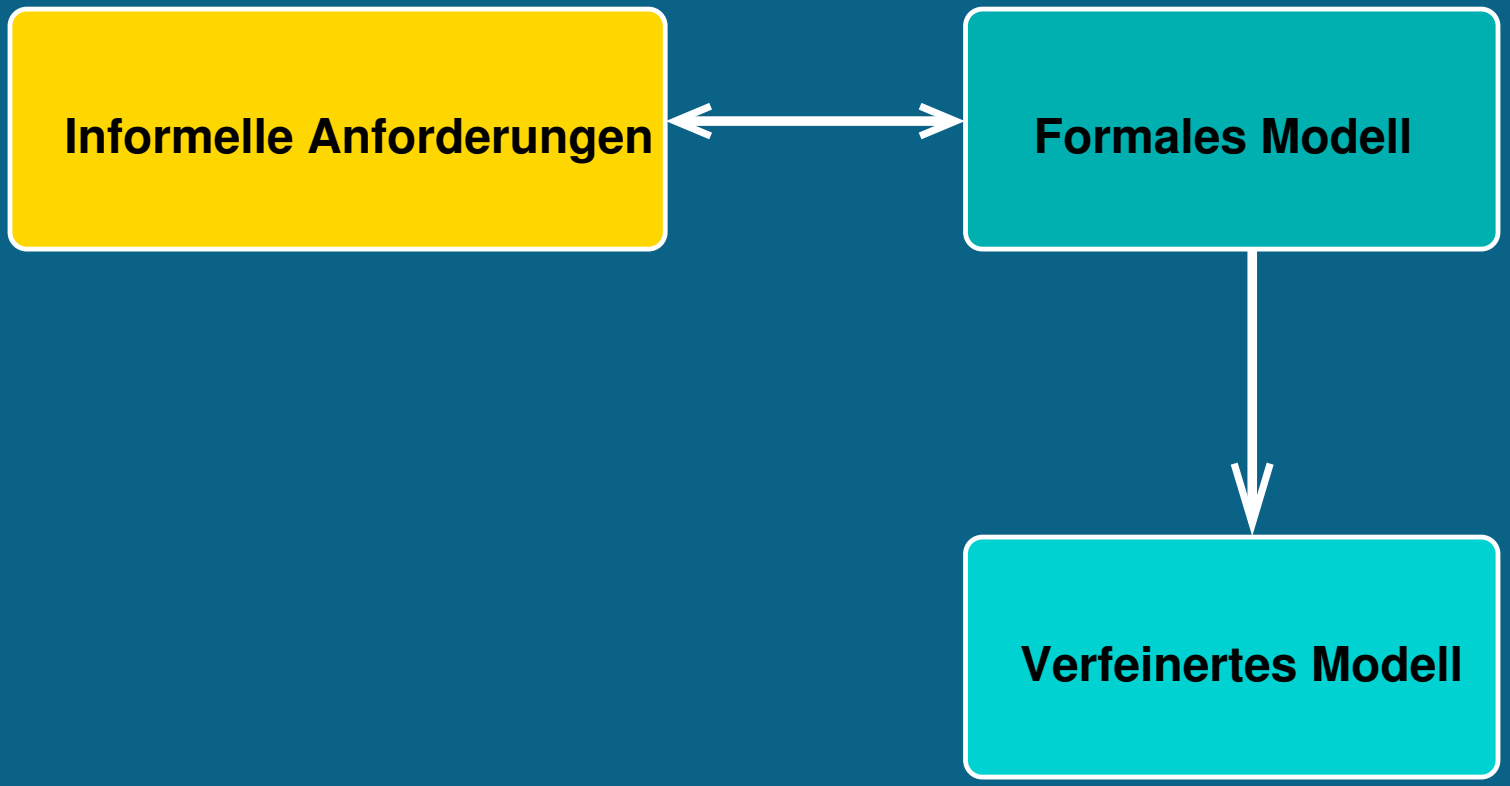
Lauf eines Programms P:

- Sequenz von States $< S_n : n \in \mathbb{N} >$
- S_{n+1} ergibt sich aus S_n durch Feuern von Updates(P,S_n)

Umgebung kann monitored- und shared-Funktionen verändern, aber nur von einem State auf den nächsten. Das Verhalten der Umgebung kann durch Constraints eingeschränkt werden.

$\Rightarrow$ Umgebung kann als zweiter Agent gesehen werden, der alternierend mit der abstrakten Maschine ausgeführt wird.

Modellierung



Modell - Vokabular

P.10 Das System verwaltet verschiedene Benutzer, die verschiedenen Gruppen zugehören. Benutzer können mehreren Gruppen angehören.

P.14 Für jeden Benutzer werden Login-Name, Name, eMail-Adresse und Passwort gespeichert.

Modell - Vokabular

P.10 Das System verwaltet verschiedene Benutzer, die verschiedenen Gruppen zugehören. Benutzer können mehreren Gruppen angehören.

P.14 Für jeden Benutzer werden Login-Name, Name, eMail-Adresse und Passwort gespeichert.

domain Benutzer
domain Gruppe

Modell - Vokabular

P.10 Das System verwaltet verschiedene Benutzer, die verschiedenen Gruppen zugehören. Benutzer können mehreren Gruppen angehören.

P.14 Für jeden Benutzer werden Login-Name, Name, eMail-Adresse und Passwort gespeichert.

domain Benutzer
domain Gruppe

shared function login: Benutzer $\longrightarrow$ String
controlled function name: Benutzer $\longrightarrow$ String
controlled function email: Benutzer $\longrightarrow$ String
shared function passwort: Benutzer $\longrightarrow$ String
controlled function gruppe: Benutzer $\longrightarrow \mathbb{P}$ Gruppe

Modell - System

monitored function mode: Benutzer $\rightarrow$ Aktion
monitored function finished: Benutzer $\rightarrow$ Bool
monitored function currentUser: $\rightarrow$ Benutzer
controlled function loggedin: Benutzer $\rightarrow$ Bool

constraint $\forall b \in$ Benutzer:

$$G(\neg \text{loggedin}(b) \rightarrow \text{mode}(b) = \text{idle} \vee \text{mode}(b) = \text{login})$$

$$G((\text{mode}(b) \neq \text{idle} \wedge \neg \text{finished}(b)) \rightarrow F(\text{currentUser} = b))$$

$$\forall m \in \text{Aktion} : G(\text{mode}(b) = m \rightarrow \text{mode}(b) = m \; U \; \text{finished}(b))$$

Modell - Programm

domain Step = {initialisierung, aktion}

controlled function pstep: → Step = initialisierung

Program =
    if pstep = initialisierung **then**
        Initialisierung
        pstep := aktion
    else
        Login
        Logout
        ChangePasswort
        ChangeEmail
    endif

Modell - Login und Passwort

Login =
if (Mode(currentUser,login) ∧ Berechtigt(currentUser,login)) **then**
if (input_login(currentUser) = login(currentUser) ∧
input_passwort(currentUser) = passwort(currentUser)) **then**
loggedin(currentUser) := true
endif
endif

ChangePasswort =
if (Mode(currentUser,chpwd) and Berechtigt(currentUser,chpwd)) **then**
if (input_passwort(currentUser) = input_cnfpasswort(currentUser)) **then**
passwort(currentUser) := input_passwort(currentUser)
endif
endif

Präzisierung der Anforderungen (1)

P.10 Das System verwaltet verschiedene Benutzer, die verschiedenen Gruppen zugehören. Benutzer können *keiner oder* mehreren Gruppen angehören.

Präzisierung der Anforderungen (1)

P.10 Das System verwaltet verschiedene Benutzer, die verschiedenen Gruppen zugehören. Benutzer können *keiner oder* mehreren Gruppen angehören.

P.11 Der Systemverwalter hat die Möglichkeit, zusätzliche Gruppen anzulegen, denen eine beliebige Kombination von Rechten zugeordnet werden kann. Er kann auch vorhandene Gruppen modifizieren oder löschen. *Wird eine Gruppe gelöscht, werden alle Benutzer, die dieser Gruppe angehören, aus dieser Gruppe entfernt*.

Präzisierung der Anforderungen (1)

P.10 Das System verwaltet verschiedene Benutzer, die verschiedenen Gruppen zugehören. Benutzer können *keiner oder* mehreren Gruppen angehören.

P.11 Der Systemverwalter hat die Möglichkeit, zusätzliche Gruppen anzulegen, denen eine beliebige Kombination von Rechten zugeordnet werden kann. Er kann auch vorhandene Gruppen modifizieren oder löschen. *Wird eine Gruppe gelöscht, werden alle Benutzer, die dieser Gruppe angehören, aus dieser Gruppe entfernt*.

P.15 *Jedes Mitglied der Gruppe Benutzer* hat die Möglichkeit, seine im System eingetragene eMail-Adresse und sein Passwort selbst zu ändern, *sofern ihm dieses Recht nicht vom Systemadministrator entzogen wird.*

Präzisierung der Anforderungen (2)

1. Ein- und Ausloggen am System. *Zum Einloggen am System gibt der Benutzer Login-Namen und Passwort ein. Stimmt das Passwort mit dem für den mittels Login-Namen identifizierten Benutzer gespeicherten Passwort überein, ist der Benutzer eingeloggt. Für das Ausloggen werden keine Eingaben benötigt.*

2. Ändern des Login-Passworts. *Zum Ändern des Passworts gibt der Benutzer es zweimal ein. Wenn die beiden Eingaben identisch sind, wird das eingegebene Passwort als neues Passwort gespeichert.*

3. *Ändern der Email-Adresse*

Verfeinertes Modell (1)

controlled function mode: Benutzer $\longrightarrow$ Aktion

controlled function finished: Benutzer $\longrightarrow$ Bool

controlled function currentUser: $\longrightarrow$ Benutzer

controlled function loggedin: Benutzer $\longrightarrow$ Bool

controlled function next: Benutzer $\longrightarrow$ Benutzer

Verfeinertes Modell (2)

```
NextUser =
    if currentUser = undefined then
      choose b ∈ Benutzer where mode(b) = login ∧ next(b) = undefined
        currentUser := b
        next(b) := b
      endchoose
    else
      choose b in Benutzer where mode(b) = login ∧ next(b) = undefined
        let bn = b ∈ Benutzer where next(b) = currentUser in
          next(b) := currentUser
          next(bn) := b
        endlet
      endchoose
      currentUser := next(currentUser)
    endif
```

Verfeinertes Modell (2)

```
NextUser =
    if currentUser = undefined then
      choose b ∈ Benutzer where mode(b) = login ∧ next(b) = undefined
        currentUser := b
        next(b) := b
      endchoose
    else
      choose b in Benutzer where mode(b) = login ∧ next(b) = undefined
        let bn = b ∈ Benutzer where next(b) = currentUser in
          next(b) := currentUser
          next(bn) := b
        endlet
      endchoose
      currentUser := next(currentUser)
    endif
```

Verfeinertes Modell (2)

```
NextUser =
    if currentUser = undefined then
      choose b ∈ Benutzer where mode(b) = login ∧ next(b) = undefined
        currentUser := b
        next(b) := b
      endchoose
    else
      choose b in Benutzer where mode(b) = login ∧ next(b) = undefined
        let bn = b ∈ Benutzer where next(b) = currentUser in
          next(b) := currentUser
          next(bn) := b
        endlet
      endchoose
      currentUser := next(currentUser)
    endif
```

Verfeinertes Modell (3)

```
Logout =
    if (Mode(currentUser,logout) ∧ Berechtigt(currentUser,logout)) then
      loggedin(currentUser) := false
      mode(currentUser) := idle
      next(currentUser) := undefined
      let bn = b ∈ Benutzer where next(b) = currentUser in
        if bn = currentUser then
          currentUser := undefined
        else
          currentUser := bn
          next(bn) := next(currentUser)
        endif
      endlet
      finished(currentUser) := true
    endif
```

Verfeinertes Modell (3)

```
Logout =
    if (Mode(currentUser,logout) ∧ Berechtigt(currentUser,logout)) then
      loggedin(currentUser) := false
      mode(currentUser) := idle
      next(currentUser) := undefined
      let bn = b ∈ Benutzer where next(b) = currentUser in
        if bn = currentUser then
          currentUser := undefined
        else
          currentUser := bn
          next(bn) := next(currentUser)
        endif
      endlet
      finished(currentUser) := true
    endif
```

Verfeinerung

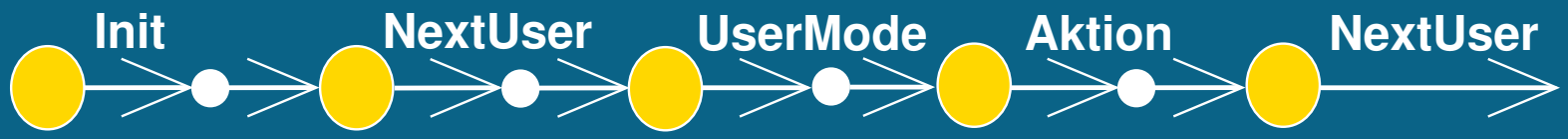
Definition:

M^* ist eine korrekte Verfeinerung von M gdw. für jeden Lauf $S_0^*, S_1^*, \ldots$ von M^* ein Lauf $S_0, S_1, \ldots$ von M und Sequenzen $i_0 < i_1 < \ldots$ und $j_0 < j_1 < \ldots$ existieren mit:

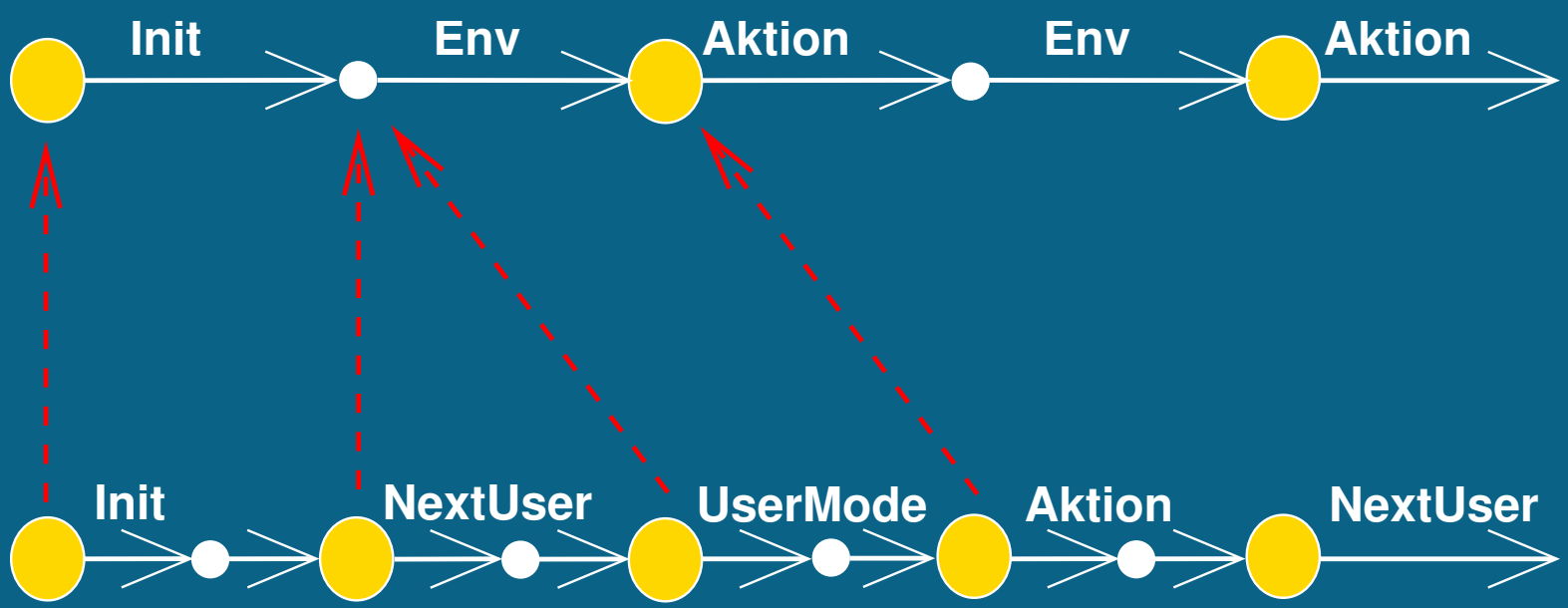
- Für alle k: $S_{i_k}^*|_{\Upsilon(M)} \equiv S_{j_k}$
- Beide Läufe und beide Sequenzen i, j sind unendlich

Verfeinerungsbeziehung

Verfeinerungsbeziehung



Verfeinerungsbeziehung



Verifikation (1)

Zeige: Das verfeinerte Modell erfüllt die Eigenschaft

$$\forall b \in \text{Benutzer}\; G((\text{mode}(b) \neq \text{idle}) \rightarrow F(\text{currentUser} = b))$$

Erster Schritt:

$$\begin{aligned} \text{Keine aktiven Benutzer im System} &\leftrightarrow \text{currentUser} = \text{undefined} \\ \text{n aktive Benutzer im System} &\leftrightarrow \forall b \in \text{Benutzer}\; \text{next}^n(b) = b \wedge \\ &\quad \forall i\; 0 < i < n : \text{next}^i(b) \neq b\; (n \geq 1) \end{aligned}$$

Beweis durch Induktion über einen beliebigen Lauf der Maschine

Verifikation (2)

I.A.: Initial keine aktiven Benutzer, und currentUser = undefined

I.S.: Die Behauptung gelte im aktuellen State, zeige:

- Kein Benutzer aktiv, dann gilt die Behauptung nach einem login (Regel NextUser)
- n Benutzer aktiv, dann gilt die Behauptung nach login (Regel NextUser) oder logout (Regel Logout)

Andere Regeln verändern die betroffenen Funktionen nicht.

Verifikation (3)

Zweiter Schritt: Bewertungsfunktion

$M(b) = i \in \mathbb{N}$ mit $\text{next}^i(\text{currentUser}) = b \wedge \forall j\ \text{next}^j(\text{currentUser}) = b : j \geq i$

Zeige: $M(b)$ monoton fallend bis $M(b) = 0$, oder mode(b) = idle

Beweis: Zeige für beliebigen Zustand S mit $M(b) > 0$ und einen beliebigen Benutzer b mit mode(b) $\neq$ idle: in S' ist M(b) kleiner oder gleich groß.

$\Rightarrow$ Regel NextUser und Logout reduzieren M(b), ansonsten M'(b) = M(b)

Dritter Schritt: NextUser wird unendlich oft gefeuert

Zusammenfassung

- Gute Modellierung der Daten möglich
- Relativ einfach Verständlich
- Erkennen von Mehrdeutigkeiten und Widersprüchen in den Anforderungen
- Verifikation von Hand

ASML (1)

```
public class Benutzer
  var login as String
  var name as String
  var email as String
  var passwort as String
  var gruppe as Set of Gruppe

var DBenutzer as Set of Benutzer = {}

extendBenutzer(l as String, n as String, e as String, p as String,
g as Set of Gruppe) as Benutzer
  let b = new Benutzer(l,n,e,p,g)
  add b to DBenutzer
  return b
```

ASML (2)

```
type Aktion = Recht or Idle

var mode as Map of Benutzer to Aktion = {->}
var finished as Map of Benutzer to Boolean = {->}
var currentUser as Benutzer
var loggedin as Map of Benutzer to Boolean = {->}

Berechtigt(b as Benutzer, r as Recht) as Boolean
  let gr = {g | g in DGruppe where r in g.recht}
  if (gr intersect b.gruppe = {}) then
    return false
  else
    return true
```

ASML (3)

```
Login()
  if (Mode(currentUser,login) and Berechtigt(currentUser,login))
    if (input_login(currentUser) = currentUser.login and
    input_passwort(currentUser) = currentUser.passwort)
      loggedin(currentUser) := true

GruppeLoeschen()
  if (Mode(currentUser,gruppeloeschen) and
  Berechtigt(currentUser,gruppeloeschen))
    let gr = the g | g in DGruppe where g.name =
    input_groupname(currentUser)
    forall b in Benutzer
      b.gruppe \ {gr}
    remove gr from DGruppe
```