

# Formalisierung von Sicherheitseigenschaften im $\mu$ -Kalkül

Michael Wahler (wahler@in.tum.de)

10. Dezember 2001

## Inhaltsverzeichnis

<b>1</b>	<b>Verifikation - harte Aussagen über weiche Ware</b>	<b>2</b>
1.1	Motivation . . . . .	2
1.2	Eine kleine Geschichte der Programmverifikation . . . . .	2
1.2.1	Hoare-Logik . . . . .	2
1.2.2	PDL, HML . . . . .	3
1.2.3	Temporallogik (CTL, CTL*, ...) . . . . .	3
1.3	Formulierung eines Programms als Transitionssystem . . . . .	3
1.3.1	Modellbegriff nach S. A. Kripke . . . . .	3
1.3.2	Umformung in Aktionen, Eigenschaften, Zustände . . . . .	4
1.4	Grundlagen der Fixpunkttheorie . . . . .	5
<b>2</b>	<b>Formulierung von Aussagen im <math>\mu</math>-Kalkül</b>	<b>6</b>
2.1	Aussagen über Aktionen, Eigenschaften, Zustände . . . . .	6
2.2	Aussagen über Lebendigkeit und Sicherheit . . . . .	6
2.2.1	Der kleinste Fixpunkt . . . . .	7
2.2.2	Der größte Fixpunkt . . . . .	8
2.3	Zusammenfassung der syntaktischen Elemente . . . . .	9
2.4	Semantik der Formeln . . . . .	9
2.5	Monotonie von $\Phi(Z)$ . . . . .	10
<b>3</b>	<b>Beispiele für Formeln im <math>\mu</math>-Kalkül</b>	<b>11</b>
3.1	Oktobertag, Lebendigkeit und kleinster Fixpunkt . . . . .	11
3.2	Sicherheit der elektronischen Geldbörse . . . . .	12
3.3	Nebenläufige Prozesse . . . . .	14
3.4	Fairness . . . . .	14

<http://www4.informatik.tu-muenchen.de/lehre/seminare/hs/WS0102/approx/>  
Betreuer: Dr. Martin Strecker (strecker@in.tum.de)

# 1 Verifikation - harte Aussagen über softe Ware

## 1.1 Motivation

In den 60er Jahren des vorigen Jahrhunderts wurde die "Softwarekrise" ausgerufen, da die Softwaresysteme immer komplexer wurden und Fragen der Wiederbenutzbarkeit, Kompatibilität etc. aufkamen, die mangels geeigneter Techniken schwer oder kaum zu beantworten waren. Deshalb suchte man nach Methoden, die "harte" Aussagen, also logische Formeln, über Software zuließen. Auf der einen Seite wurde daraufhin die Idee des Software Engineering geboren, das anschauliche Methoden zum Entwurf komplexer Software zur Verfügung stellt; auf der anderen Seite begann man, formale Logik zur Verifikation von bestimmten Programmeigenschaften einzusetzen.

In dieser Arbeit möchte ich den  $\mu$ -Kalkül vorstellen, der es erlaubt, Programme als Formeln zu notieren, die mit den Methoden der Logik auswertbar sind. Den  $\mu$ -Kalkül zeichnet gegenüber anderen Logiken aus, dass Prädikate über das System rekursiv definiert werden können. Das macht ihn so mächtig, dass er die meisten anderen Logiken wie z.B. CTL subsumiert [4]. Die zu verifizierenden Programme müssen als Zustandsautomaten vorliegen, damit Aussagen über die Aktionen und die erreichbaren Zustände gemacht werden können. Speziell sind folgende zwei Eigenschaften interessant:

- **Lebendigkeit:** Das System soll in keinen Deadlock-Zustand kommen. Das bedeutet, dass es möglich sein soll, bestimmte Zustände immer wieder zu erreichen. Aussage: "*Der Zustand Y kann immer wieder erreicht werden.*"
- **Sicherheit:** Bestimmte Eigenschaften müssen während des ganzen Programmablaufs vorhanden sein. So können Invarianten überprüft werden. Aussage: "*Der Zustand X ist niemals erreichbar.*"

Die Wahl der Worte wie *möglich*, *müssen* und *können* in den beiden Beispielen macht deutlich, dass es sich beim  $\mu$ -Kalkül um eine Modallogik<sup>1</sup> handelt.

Im Folgenden möchte ich – nachdem ich kurz auf die Geschichte der Programmverifikation eingegangen bin – anhand eines Beispiels die Ausdrucksstärke des  $\mu$ -Kalküls aufzeigen: Es soll gezeigt werden, dass auf einer JavaCard, auf der mit Hilfe von drei Applets eine elektronische Geldbörse realisiert ist, die Applets korrekt arbeiten. Das Beispielsystem, das in [5] vorgestellt wurde, muss dabei zunächst in einen Zustandsautomaten umgewandelt werden. Dann werden die zu verifizierenden Eigenschaften im  $\mu$ -Kalkül formuliert.

Für den  $\mu$ -Kalkül existieren effiziente Verfahren, die Gültigkeit von Formeln zu überprüfen. Beispielsweise wird in [7] ein Tableau-basiertes Model Checking vorgestellt, in [8] stellt A. Kick ein Verfahren vor, um mit Hilfe von Zeugen – "witnesses" - Aussagen über die Gültigkeit von Formeln zu treffen.

## 1.2 Eine kleine Geschichte der Programmverifikation

### 1.2.1 Hoare-Logik

In den 1960er Jahren entwickelten Floyd und Hoare eine Logik, die ein Beweisverfahren für imperative Sprachen einführt, das auf Zusicherungen beruht. Diese Zusicherungen können dann vom Axiom der Hoare-Logik – der Zuweisung – deduziert werden. Ein Beispiel für ein um Zusicherungen ergänztes Programmstück ist

$$\{ x > 0 \} y := \text{div}(2, x) \{ y = \frac{2}{x} \}$$

<sup>1</sup>Modallogik: Boolesche Logik, die um zwei Operatoren erweitert wird, um die Möglichkeit oder Notwendigkeit einer Aussage auszudrücken.

Die Formel bedeutet, dass wenn unter der Vorbedingung, dass  $x$  größer als 0 ist, die Division  $2 \div x$  ausgeführt wird, die angegebene Nachbedingung für  $y$  gilt. Mit Hilfe der Hoare-Logik kann formal überprüft werden, ob ein Programm von einem gegebenen Anfangszustand in einem gewünschten Endzustand hält. Jedoch sind bestimmte Aussagen, die zur Analyse von komplexen<sup>2</sup> Softwaresystemen nötig sind, nicht möglich. Es lassen sich z.B. keine Aussagen über bestimmte Zustände treffen, die während des Programmablaufs auftreten sollen – was aber gerade bei reaktiven Systemen notwendig ist. Seit den 1960er Jahren hat sich jedoch einiges getan!

### 1.2.2 PDL, HML

PDL<sup>3</sup> und HML<sup>4</sup> sind Modallogiken. Eine Modallogik wird in [3] als eine "Logik des Möglichen und Notwendigen, des Könnens und des Müssens" definiert. Für die Aussagen über das Programm stehen die Operatoren der Modallogik zur Verfügung:

- $\Box w$  bezeichnet eine *notwendige* Aussage  $w$  ("Das Programm  $P$  darf nie terminieren.").
- $\Diamond w$  bezeichnet eine *kontingente* Aussage  $w$  ("Es ist möglich, dass das Programm  $P$  terminiert.").

Das zu analysierende Programm muss als Zustandsübergangsdiagramm dargestellt sein, auf dem Aussagen formuliert werden. Die modalen Operatoren werden zur Verknüpfung von Aktionen und Zuständen verwendet.  $[a]\Phi$  entspricht dem  $\Box$ -Operator aus der Modallogik und bedeutet: "Nach jeder Ausführung von  $a$  muss  $\Phi$  gelten". Dem Operator  $\Diamond$  entspricht  $\langle a \rangle \Phi$ , was "es ist möglich,  $a$  auszuführen und in einem  $\Phi$ -Zustand zu enden" bedeutet.

### 1.2.3 Temporallogik (CTL, CTL\*, ...)

Die Temporallogik, auf der CTL, CTL\* usw. basieren, ist eine Variante der Modallogik. Die Operatoren haben hier eine *zeitliche* Interpretation.

- $\mathbf{G}w$  : "w wird immer gelten." (Invarianzeigenschaft)<sup>5</sup>
- $\mathbf{F}w$  : "w wird einmal gelten." (Lebendigkeitseigenschaft)<sup>6</sup>
- $\mathbf{X}w$  : "w gilt im nächsten Zustand."
- $[P \mathbf{U} Q]$  : "P gilt, bis Q gilt." (until)

**Beispiel:**  $\Diamond \Box w$  bedeutet: "w wird irgendwann Invariante."

## 1.3 Formulierung eines Programms als Transitionssystem

### 1.3.1 Modellbegriff nach S. A. Kripke

In 1.1 habe ich erwähnt, da im  $\mu$ -Kalkül Aussagen über Aktionen und Zustände gemacht werden und deshalb das zu überprüfende System als Zustandsautomat vorliegen muss. Um den Zusammenhang zwischen einem Programm und einem Zustandsautomaten besser zu verstehen, greift Karjoth [2] einen Vorschlag von Kripke auf, ein verständliches Modell für die Modal- und Temporallogik zu bieten:

<sup>2</sup>Hier sind vor allem verteilte/reaktive Systeme aufgrund ihres meist nichtdeterministischen Verhaltens besonders interessant.

<sup>3</sup>Propositional Dynamic Logic

<sup>4</sup>Hennessy-Milner logic

<sup>5</sup>entspricht  $\Box w$  in der Modallogik

<sup>6</sup>entspricht  $\Diamond w$

- Gegeben ist die Menge der möglichen Welten; darunter befindet sich die aktuelle Welt.
- Es existieren Zugänglichkeitsrelationen zwischen diesen Welten.
- Alle Aussagen stehen in Bezug zur aktuellen Welt (wie in der natürlichen Sprache).
- $\Box w$ :  $w$  gilt immer (also in allen Welten, die von der aktuellen Welt aus erreichbar sind)
- $\Diamond w$ :  $w$  gilt irgendwo (also in mindestens einer Welt, die von der aktuellen Welt aus erreichbar ist)
- Deutung:
  - Die Menge aller Welten sind die möglichen Zustände eines Programms.
  - Die Zugänglichkeitsrelation ist die Früher/Später-Beziehung der Zustände; sie ist definiert durch die Reihenfolge der Aktionsausführungen des Programms.

### 1.3.2 Umformung in Aktionen, Eigenschaften, Zustände

In 1.1 habe ich bereits erwähnt, dass die zu betrachtenden Systeme als Zustandsautomaten vorliegen müssen. Der Zustandsraum eines Programms ist die Potenzmenge der möglichen Variablenbelegungen.

#### Beispiel:

```
x:=1;
y:=3;
while (y>0) do x:=x*y; y:=y-1
```

ergibt den Automaten in Abbildung 1, wobei  $S_{(i,j)}$  den Zustand  $\sigma[x \rightarrow i, y \rightarrow j]$  bezeichnet.

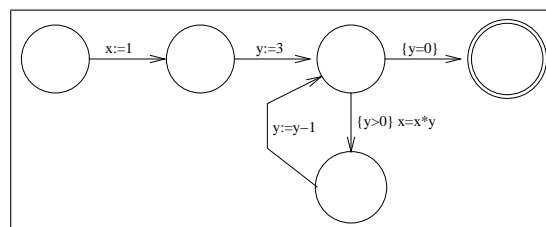


Abbildung 1: Ein Zustandsautomat

Ebenso einfach können Programme in einer Prozesssprache, z.B. CCS, transformiert werden. Der Automaten in Abbildung 1 entspricht folgender Pseudo-Prozesssprache:

$$A(x, 0) = A(x, 0)$$

$$A(x, y) = A(x*y, y-1)$$

Die Transformation eines objektorientierten Programms in einen Automaten ist ungleich schwieriger, da Methodenaufrufe geschachtelt sein können, Attribute und Methoden mit *static* oder *public* verschiedene Sichtbarkeiten haben können und zwischen dynamischer und statischer Bindung unterschieden werden muss. Deshalb werde ich dieses Thema an dieser Stelle auslassen.

## 1.4 Grundlagen der Fixpunkttheorie

$x$  heißt Fixpunkt der Funktion  $f$ , wenn das Ergebnis der Anwendung von  $f$  auf  $x$  wieder  $x$  ist. Es gilt also

$$x = f(x)$$

**Beispiel:**  $x = 1$  ist Fixpunkt der Gleichung  $f(x) = \frac{1}{x}$ .

**Theorem** (nach Knaster-Tarski): Ist  $f$  eine monotone Abbildung über einer geordneten, vollständigen Menge  $A$  mit kleinstem Element, so ist die Gleichung

$$x = f(x)$$

lösbar und es existiert ein eindeutig bestimmter kleinster Fixpunkt von  $f$ .

$f$  heißt *monoton*, wenn gilt:

$$x_1 < x_2 \Rightarrow f(x_1) < f(x_2)$$

Den *kleinsten Fixpunkt* von  $f$  findet man durch Iterieren von  $f$  über ein Element  $\perp$  des Verbandes. Man bekommt so eine aufsteigende<sup>7</sup> Kette für die Approximanten von  $f$ , sodass gilt (der Operator  $\mu$  bezeichnet den kleinsten Fixpunkt):

$$\mu f = \bigcup_{\alpha < \kappa} f^\alpha(\perp)$$

**Beispiel:** Natürliche Zahlen

Die Funktion

$$f(M) = \{0\} \cup \{n+1 \mid n \in M\}$$

definiert die natürlichen Zahlen.  $N_0$  ist  $\emptyset$ , und die Menge  $N$  wird erzeugt, indem man  $f$  immer wieder auf  $N_0$  iteriert:

$$\begin{aligned} N_0 &= \emptyset \\ N_1 &= \{0\} \\ N_2 &= \{0, 1\} \\ N_3 &= \{0, 1, 2\} \\ &\vdots \\ N_\omega &= \mathbb{N} \end{aligned}$$

□

Den *größten Fixpunkt* von  $f$  findet man analog durch Iterieren von  $f$  über einem Element  $\top$ , also über der gesamten Menge selbst. Ein Beispiel ist in 2.2.2 zu sehen: Aus der Menge  $\top$ , also dem Zustandsraum  $\Sigma$ , werden mit jedem Iterationsschritt "falsche" Zustände eliminiert. Es gilt:

$$\nu f = \bigcap_{\alpha < \kappa} f^\alpha(\Sigma)$$

---

<sup>7</sup>Wegen der Monotonie

## 2 Formulierung von Aussagen im $\mu$ -Kalkül

Im folgenden Abschnitt werde ich mit Hilfe von Beispielen schrittweise in die Besonderheiten des  $\mu$ -Kalküls einführen.

### 2.1 Aussagen über Aktionen, Eigenschaften, Zustände

Sei  $a$  eine Aktion und  $\Phi$  eine Eigenschaft. Dann bedeutet  $[a]\Phi$  :  $\Phi$  gilt in allen Zuständen, die durch eine Aktion  $a$  erreichbar sind (" $\Phi$  gilt nach jeder Aktion  $a$ "). In Abbildung 2 ist ein Beispiel eines Transitionssystems mit dieser Eigenschaft abgebildet.

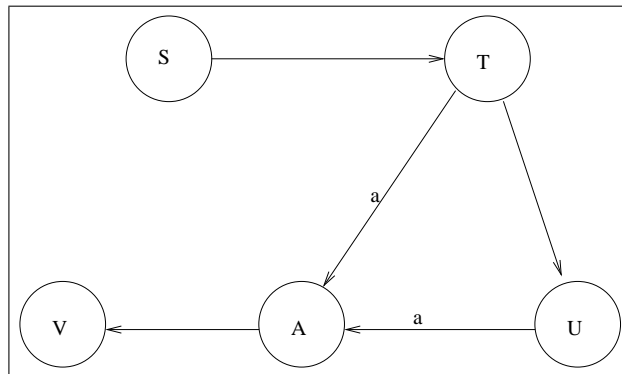


Abbildung 2: Nach jeder Ausführung der Aktion  $a$  gilt  $A$ , also  $[a]A$ .

Die Formel  $\langle a \rangle \Phi$  bedeutet: es gibt eine Aktion  $a$ , nach deren Ausführung  $\Phi$  gilt. Ein Beispiel ist in Abbildung 3 zu sehen. In Abbildung 3 gilt außerdem:  $\neg[a]A$ , denn es kommt

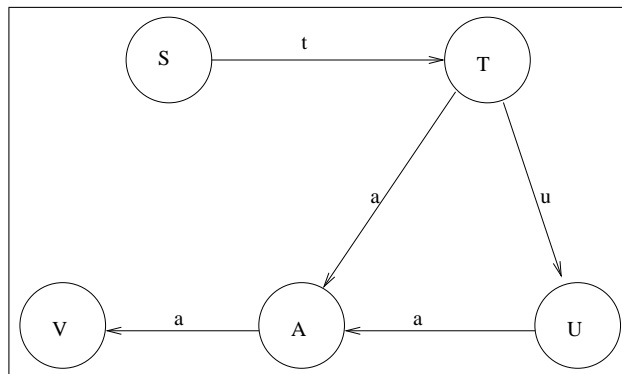


Abbildung 3: Es gibt eine Aktion  $a$ , nach deren Ausführung  $A$  gilt.

auch eine Aktion  $a$  vor, die nicht zu einem Zustand  $A$  führt (sondern  $V$ ).

### 2.2 Aussagen über Lebendigkeit und Sicherheit

Wie ich bereits in 1.1 erwähnt habe, sind Lebendigkeit und Sicherheit eines Systems die beiden wichtigsten Eigenschaften, die mit Hilfe des  $\mu$ -Kalküls nachweisbar sind. Die Formulierung dieser beiden Eigenschaften will ich jeweils wieder mit einem Beispiel zeigen.

### 2.2.1 Der kleinste Fixpunkt

Eine in Zusammenhang mit Zustandsautomaten gerne gestellte Frage ist: *Ist der Zustand  $Z_i$  erreichbar?* Um diese Frage zu beantworten, muss die Menge der erreichbaren Zustände

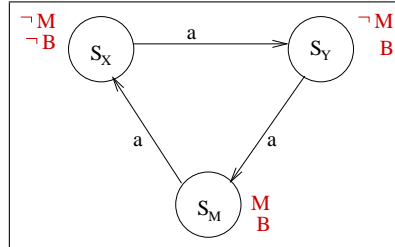


Abbildung 4: Ist der Zustand  $Z_i$  erreichbar?

definiert werden. In diesem Beispiel sei ein Zustand  $Z$  genau dann erreichbar, wenn die Eigenschaft  $M$  in  $Z$  erfüllt ist, oder wenn man zu  $Z$  von einem erreichbaren Zustand aus durch eine  $a$ -Aktion gelangen kann. Wir stellen folgende Formel  $\Phi$  auf:

$$\Phi = M \vee (B \wedge \langle a \rangle Z)$$

$\Phi$  hängt jedoch von  $Z$  ab, deshalb

$$\Phi(Z) = M \vee (B \wedge \langle a \rangle Z) \quad (1)$$

Die Besonderheit des  $\mu$ -Kalküls ist, dass die Semantik der Formeln rekursiv, d.h. über eine Fixpunktgleichung, definiert ist. Zur Lösung des Problems reicht es nun aus, den kleinsten Fixpunkt von (2) zu bestimmen. Dies geschieht durch Iterieren über ein Element  $\perp$ , das in der Mengennotation der leeren Menge entspricht. Also:

1.  $Z_0$  zunächst die leere Menge  $\emptyset$ .
2.  $Z_1 = M \vee (B \wedge \langle a \rangle \emptyset) = \{S_M\}$   
Das sind alle Zustände, in denen  $M$  gilt (also  $\{S_M\}$ ) oder von denen man mit einer  $a$ -Aktion in einem Zustand aus  $\emptyset$  gelangt<sup>8</sup>
3.  $Z_2 = M \vee (B \wedge \langle a \rangle \{S_M\}) = \{S_Y, S_M\}$   
Das sind alle Zustände, in denen  $M$  direkt gilt ( $\{S_M\}$ ) oder Zustände, in denen  $B$  gilt und von denen man durch eine  $a$ -Aktion einen Zustand aus  $\{S_M\}$  erreicht, also  $\{S_Y\}$
4.  $Z_3 = M \vee (B \wedge \langle a \rangle \{S_Y, S_M\}) = \{S_Y, S_M\}$ .  
*Surprise, surprise!* Das sind die gleichen Zustände wie in 3.!

Wir haben mit  $Z_2$  einen kleinsten Fixpunkt für  $\Phi(Z)$  gefunden. Das ist zwar der kleinste Fixpunkt, jedoch nicht die kleinste Menge  $Z$ , die  $\Phi(Z)$  erfüllt, denn das ist die leere Menge  $\emptyset$ :  $\Phi(\emptyset) = \{s_2\}$ .  $\Phi(\emptyset)$  wird *Approximant* von  $\Phi(Z)$  genannt. Durch den induktiven Aufbau ist klar:

Wenn ein Zustand im Fixpunkt der Funktion  $\Phi(Z)$  liegt, dann liegt er auch in einem endlichen Approximanten.<sup>9</sup>

Wir schreiben hierfür

$$\mu Z. \Phi(Z) \equiv \mu Z. M \vee (B \wedge \langle a \rangle Z) \quad (2)$$

<sup>8</sup> $\langle a \rangle \emptyset$  ist immer *false*; wie auch soll ein Zustand aus der leeren Menge, also *kein* Zustand erreichbar sein?

<sup>9</sup>Bei Unklarheit siehe [4].

Eine alternative Deutung dieser Formel ist, dass  $B$  nur endlich lange gilt, bevor  $M$  hält – ” $\mu$ means finite looping”[4]. Somit kann man die CTL-Formel  $\exists[B \text{ U } M]$ <sup>10</sup> übersetzen:  $\mu Z.M \vee (B \wedge \langle - \rangle Z)$ <sup>11</sup>. Diese Formel ist fast identisch mit der Beispielformel (3), nur dass wir in der CTL-Formel den Weg beliebig lassen. In 3.1 werde ich zeigen, wie der kleinste Fixpunkt zum Nachweis der *Lebendigkeit* eines Systems verwendet werden kann.

### 2.2.2 Der größte Fixpunkt

Der größte Fixpunkt einer Funktion  $\Phi(Z)$  ist die größte Menge  $Z_i$ , für die gilt  $\Phi(Z_i) = Z_i$ . Analog zur Bestimmung des kleinsten Fixpunktes ist  $Z_i$  induktiv herleitbar, allerdings startet die Induktion nicht bei einem Element  $\perp$ , sondern bei  $\top$ , was der Menge  $\Sigma$  aller möglichen Zustände entspricht. Mit jedem Schritt werden nun Zustände aus der Menge  $\Sigma$  entfernt, in denen  $\Phi$  nicht hält. Wenn wir mit  $Z_i$  schließlich den größten Fixpunkt erreichen, wissen wir, dass

1.  $Z_i$  eine Menge von Zuständen ist, für die  $\Phi$  gilt, und
2. bei jedem Aufruf von  $\Phi(Z)$  erhalten wir als Ergebnis wieder  $Z_i$  aufgrund der Fixpunkteigenschaft.

Wir haben mit  $Z_i$  also eine Menge von Zuständen gefunden, von denen man bei nochmaliger – einschließlich unendlicher – Iterierung von  $\Phi(Z)$  *immer* zu Zuständen aus  $Z_i$  gelangt. Wir schreiben den größten Fixpunkt als  $\nu Z.\Phi(Z)$ .

#### Beispiel:

Die Formel

$$\nu Z.P \wedge [a]Z \quad (3)$$

bedeutet: ” $P$  gilt in jedem  $a$ -Pfad.” In Abbildung 5 ist ein Automat abgebildet, für den es eine Menge von Zuständen gibt, die (4) erfüllen. Bestimmung des größten Fixpunkts:

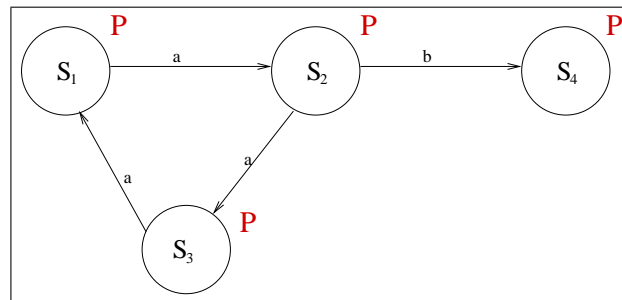


Abbildung 5: In allen Zuständen gilt  $P$

1.  $\Phi(Z_0) = \Sigma = \{s_1, s_2, s_3, s_4\}$
2.  $\Phi(Z_1) = \|\!|P \wedge [a]Z_0\|\!| = \{s_1, s_2, s_3, s_4\} \cap \{s_1, s_2, s_3, s_4\} = \{s_1, s_2, s_3, s_4\}$
3.  $\Phi(Z_2) = \|\!|P \wedge [a]Z_1\|\!| = \{s_1, s_2, s_3, s_4\} \cap \{s_1, s_2, s_3, s_4\} = \{s_1, s_2, s_3, s_4\}$

Somit ist  $Z_1$  der größte Fixpunkt der Gleichung (4), und wie in Abbildung 5 zu sehen ist, können wir für die Zustände in  $Z_1$  tatsächlich garantieren, dass in ihnen  $P$  gilt, und dass

<sup>10</sup>”Irgendwann muss B Until M gelten.”

<sup>11</sup> $\langle - \rangle$  steht für eine beliebige Aktion

man mit einer  $a$ -Aktion wieder in einen  $P$ -Zustand gelangt<sup>12</sup>– unendlich oft. Im Gegensatz zum  $\mu$ -Operator ist hier ein unendliches Loopen ausdrückbar, was sich optimal dazu eignet, Aussagen über die *Sicherheit* eines Systems zu formulieren.

### 2.3 Zusammenfassung der syntaktischen Elemente

Die Formeln des  $\mu$ -Kalküls sind induktiv wie folgt definiert:

- $P$  ist eine Formel (atomare Aussage).
- $Z$  ist eine Formel (Variable).
- $\neg\Phi$  ist eine Formel, wenn  $\Phi$  eine Formel ist (Negation).
- $\Phi_1 \wedge \Phi_2$  ist eine Formel, wenn  $\Phi_1$  und  $\Phi_2$  Formeln sind (Konjunktion).
- $\Phi_1 \vee \Phi_2$  ist eine Formel, wenn  $\Phi_1$  und  $\Phi_2$  Formeln sind (Disjunktion).
- $[a]\Phi$  ist eine Formel, wenn  $\Phi$  eine Formel ist ( $\Phi$  gilt nach jeder  $a$ -Aktion).
- $\langle a \rangle \Phi$  ist eine Formel, wenn  $\Phi$  eine Formel ist ( $\exists$  Aktion  $a$ , nach der  $\Phi$  gilt).
- $\nu Z.\Phi$  ist eine Formel, wenn  $\Phi$  eine Formel ist (größter Fixpunkt).
- $\mu Z.\Phi$  ist eine Formel, wenn  $\Phi$  eine Formel ist (kleinster Fixpunkt).

### 2.4 Semantik der Formeln

Die Semantik des  $\mu$ -Kalküls bildet eine Formel in eine Menge von Zuständen ab, in denen diese Formel gilt. Die Abkürzung  $\|P\|$  steht für  $\|P\|_V^T$ , wobei  $T = (\Sigma, \rightarrow)$  ein beschriftetes Transitionssystem ist mit  $\Sigma$  als Menge der Zustände und  $\rightarrow$  als Zustandsübergangsrelation und einer Interpretationsfunktion  $V$ .

$$\begin{aligned}
\|P\| &= V(P) \\
\|Z\| &= V(Z) \\
\|\neg\Phi\| &= \Sigma - \|\Phi\| \\
\|\Phi_1 \wedge \Phi_2\| &= \|\Phi_1\| \cap \|\Phi_2\| \\
\|\Phi_1 \vee \Phi_2\| &= \|\Phi_1\| \cup \|\Phi_2\| \\
\|[a]\Phi\| &= \{s \mid \forall t. s \xrightarrow{a} t \Rightarrow t \in \|\Phi\|\} \\
\|\langle a \rangle \Phi\| &= \{s \mid \exists t. s \xrightarrow{a} t \wedge t \in \|\Phi\|\} \\
\|\nu Z.\Phi\| &= \bigcup \{s \subseteq \Sigma \mid s \subseteq \|\Phi\|_{[Z:=s]}\} \\
\|\mu Z.\Phi\| &= \bigcap \{s \subseteq \Sigma \mid s \supseteq \|\Phi\|_{[Z:=s]}\}
\end{aligned}$$

wobei  $V[Z := s]$  die Auswertung ist, die  $Z$  durch  $s$  substituiert und die sonst mit  $V$  übereinstimmt. Zur approximativen Programmauswertung, wie in 2.2.1 angesprochen, dienen folgende Äquivalenzen:

$$\begin{aligned}
\|\mu Z^{<\alpha}.\Phi\| &= \bigcup_{\beta < \alpha} \|\mu Z^\beta.\Phi\| \\
\|\mu Z^\alpha.\Phi\| &= \|\Phi\|_{V[Z:=\|\mu Z^{<\alpha}.\Phi\|]} \\
\|\nu Z^{<\alpha}.\Phi\| &= \bigcap_{\beta < \alpha} \|\nu Z^\beta.\Phi\| \\
\|\nu Z^\alpha.\Phi\| &= \|\Phi\|_{V[Z:=\|\nu Z^{<\alpha}.\Phi\|]}
\end{aligned}$$

<sup>12</sup>Aus  $s_4$  führt keine  $a$ -Kante, deshalb führt jede  $a$ -Kante aus  $s_4$  in einen  $P$ -Zustand :-)

## 2.5 Monotonie von $\Phi(Z)$

In 1.4 wurde das Knaster-Tarski-Theorem vorgestellt; es sagt aus, dass eine Funktion  $f$  monoton sein muss, um einen kleinsten Fixpunkt zu besitzen. In den vorigen Abschnitten haben wir munter die Fixpunktoperatoren  $\{\mu, \nu\}$  verwendet, ohne uns mit Fragen der Monotonie zu beschäftigen.

**Definition:** Ein Vorkommen von  $Z$  in  $\Phi$  heißt *positiv*, wenn  $Z$  im Gültigkeitsbereich einer geraden Anzahl von Negationen vorkommt.

Die Monotonie der Formeln  $\Phi(Z)$  im  $\mu$ -Kalkül ist durch folgende syntaktische Wohlformtheitsbedingung gegeben:

Eine Funktion  $\Phi(Z)$  ist monoton,  
falls alle  $Z$  in  $\Phi$  *positiv* vorkommen.

Diese Einschränkung ist sofort klar, wenn man die Semantik des Negationsoperators beachtet. Wenn in einer Formel  $Z$  *negativ* vorkommt und  $Z$  im  $n$ -ten Iterationsschritt klein ist, ist  $Z$  im  $(n+1)$ -ten Schritt sehr groß ( $\Sigma - Z$ ), im  $(n+2)$ -ten Schritt hingegen wieder klein.  $\Phi(Z)$  wächst / schrumpft also nicht kontinuierlich, sondern oszilliert. Somit ist nie ein Fixpunkt erreichbar.

### 3 Beispiele für Formeln im $\mu$ -Kalkül

#### 3.1 Oktoberfest, Lebendigkeit und kleinster Fixpunkt

Wir interessieren uns dafür, ob es möglich ist, das Oktoberfest besuchen zu können (oder anders formuliert: das Oktoberfest *erreichen* zu können):

”Von welchen Orten (*Zuständen*) komme ich auf das Oktoberfest?” (4)

Man kann das Oktoberfest besuchen, wenn man schon in München ist ( $M$ ), oder es am Aufenthaltsort einen Bahnhof gibt ( $B$ ) und ein Zug fährt ( $a$ ), mit dem man nach München reisen kann. In diesem Beispiel sind für uns die Zustände interessant, in denen

- $M$  gilt *oder*
- $B$  gilt *und* über eine  $a$ -Kante ein Zustand erreichbar ist, von dem man in einen  $M$ -Zustand gelangt.

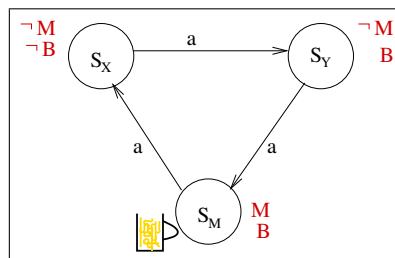


Abbildung 6: Der Weg zum Oktoberfest

Wenn wir die Aussagen in einer Formel  $\Phi$  miteinander verknüpfen, erhalten wir

$$\Phi(Z) := M \vee (B \wedge \langle a \rangle Z) \quad (5)$$

wobei  $Z$  für die Menge der Zustände steht, von denen an einen  $M$ -Zustand erreicht.

Man kann die Frage (5) auch umformen in:

”Besteht die Möglichkeit, zum Oktoberfest zu kommen?” (6)

Das ist eine Frage nach Lebendigkeit eines Systems, und mit dem  $\mu$ -Operator haben wir nun ein Werkzeug, um diese Lebendigkeit nachzuweisen. Das ist im  $\mu$ -Kalkül effektiv berechenbar. In 2.2.1 haben wir bereits eine Lösung für dieses Problem gesehen.

#### Zusammenfassung:

Die Eigenschaft  $\Phi$  wird irgendwann (auf irgendeinem Pfad) gelten:

- $\mu Z. \Phi \vee [-] Z$

Die Eigenschaft  $\Phi$  wird irgendwann auf einem  $a$ -Pfad gelten:

- $\mu Z. \Phi \vee [a] Z$

### 3.2 Sicherheit der elektronischen Geldbörse

In [5] behandeln Barthe, Gurov und Huisman den Nachweis von Sicherheitseigenschaften verteilter Anwendungen. In ihrem Beispiel geht es um eine elektronische Geldbörse EP, die mit einer JavaCard realisiert ist. Die Besonderheit ist, dass man mit dieser Geldbörse an Treueprogrammen von verschiedenen Firmen teilnehmen kann. Bei jedem Einkauf bei einer der teilnehmenden Firmen bekommt man Punkte gutgeschrieben. Auf der JavaCard laufen drei Applets:

- **P**: Dieses Applet kümmert sich um die Auszahlungen. Es speichert zusätzlich die letzten getätigten Transaktionen in einer Tabelle. Da diese Tabelle endlich ist, wird ihr Inhalt gelöscht, sobald die Tabelle voll ist. Damit den "Treue-Applets" keine Transaktionsinformationen verloren gehen, muss P es die anderen Applets benachrichtigen, bevor die Tabelle gelöscht ist. Dies geschieht durch Aufruf einer Methode *logFull*, die alle am Treueprogramm teilnehmenden Applets zur Verfügung stellen müssen. Der *logFull*-Dienst sei aber nun kostenpflichtig, und nicht alle Applets, die am Treueprogramm teilnehmen, nehmen ihn in Anspruch.
- **AF**: Das Applet AF (Air France) nimmt am *logFull*-Dienst teil, wird also von P benachrichtigt, sobald die Transaktionstabelle voll ist. Wenn es benachrichtigt wird, liest es die Transaktionstabelle von P aus und fragt auch den Punktestand von RaC ab, da die beiden Partner sind.
- **RaC**: Rent-A-Car nimmt zwar am Treueprogramm teil, hat sich aber nicht am *logFull*-Dienst angemeldet, da er kostenpflichtig ist. RaC kann jedoch leicht herausfinden, wann die Tabelle voll ist, nämlich dann, wenn AF den Punktestand von RaC abfragt. Jetzt könnte RaC ebenfalls die Tabelle von P auslesen.

Der letzte Punkt beschreibt den unerwünschten Fall: RaC liest die P-Tabelle aus, wenn seine Methode zur Punkteabfrage aufgerufen wird. Zur Verifikation des Systems wird wie

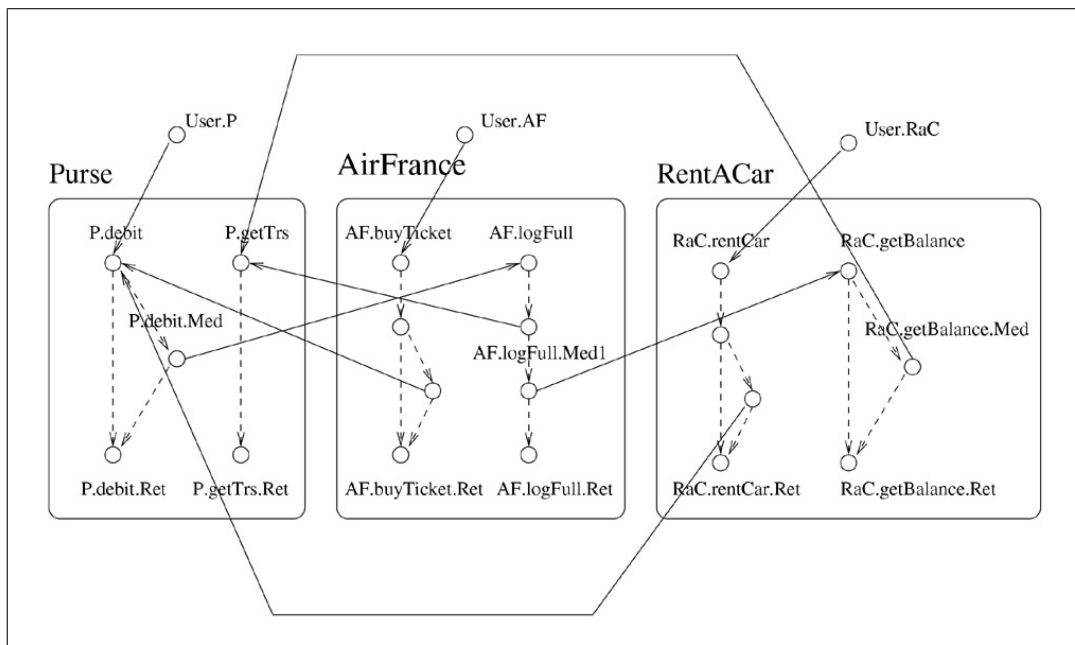


Abbildung 7: Der Aufrufgraph der Applets

folgt vorgegangen:

1. Überprüfung, ob jedes einzelne Applet die Sicherheitseigenschaft erfüllt
2. Überprüfung der Komposition der Applets

Die Eigenschaften der einzelnen Applets sind durch eine Spezifikationsprache beschrieben. Diese Sprache kann in den  $\mu$ -Kalkül kodiert werden und mit Model Checking ausgewertet werden.

```
SPEC_P (X) =
  ALWAYS.
    WITHIN (X.getTrs).
      X CALLS {}
```

```
SPEC_AF (Y, X, Z) =
  ALWAYS.
    WITHIN (Y.logFull).
      Y CALLS (X.getTrs, Z.getBalance)
```

```
SPEC_RaC (Z) =
  ALWAYS.
    WITHIN (Z.getBalance).
      Z CALLS {}
```

```
SPEC_EP (P, AF, RaC) =
  ALWAYS.
    WITHIN (AF.logFull)
      NOT (Z CALLS {X.getTrs})
```

Bei (1) ist beispielsweise zu überprüfen, dass beim Aufruf von *RaC.getBalance* das Applet *RaC* nicht *P.getTrs* aufruft. Um zu zeigen, dass (2) korrekt ist, genügt es zu zeigen, dass in jedem Programmdurchlauf ein externer Knoten (z.B. *User.P*) an erster Stelle steht, und dass höchstens ein Applet zur gleichen Zeit *aktiv* ist. Letzteres können wir im  $\mu$ -Kalkül formulieren:

$$\Phi(Z) = P \wedge [-]Z$$

wobei das Prädikat  $P$  bedeutet: "Höchstens ein Applet ist aktiv.". Wenn es nun erreichbare Zustände gibt, in denen  $\Phi(Z)$  nicht gilt, ist die Sicherheitseigenschaft verletzt.  $\forall Z. \Phi(Z)$  ist die größte Menge der Zustände, für die die Sicherheitseigenschaft gilt. Ist nun ein Zustand  $\in Y$  erreichbar, in dem  $\Phi(Z)$  nicht gilt? Anders gefragt, wird irgendwann einmal eintreffen, dass die Sicherheitseigenschaft nicht gilt? (vgl. 3.1)

$$\mu Y. \neg (\forall Z. P \wedge [-]Z) \vee \langle - \rangle Y \quad (7)$$

Wenn bei der Auswertung der Formel (8) ein kleinster Fixpunkt erreicht wird, ist die Sicherheit des Systems nicht gewährleistet. Wenn hingegen kein kleinster Fixpunkt erreicht wird, ist sichergestellt, dass die Applets richtig komponiert sind.

### Zusammenfassung:

Die Eigenschaft  $\Phi$  soll immer gelten:

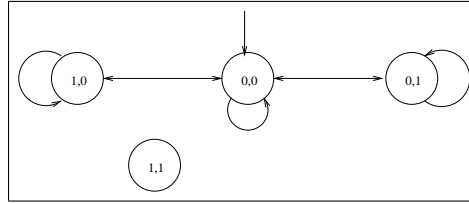
- $\forall Z. \Phi \wedge [-]Z$

Die Eigenschaft  $\Phi$  soll entlang jedes  $a$ -Pfads gelten:

- $\forall Z. \Phi \wedge [a]Z$

### 3.3 Nebenläufige Prozesse

Die Frage, ob ein Zustand in einem Zustandsautomaten erreichbar ist, ist bei der Verifikation mit Zustandsautomaten zentral. Ein Beispiel: Zwei Prozesse a und b haben einen gemeinsamen kritischen Abschnitt. Der Zustand  $(a, b)$  kodiert den Abschnitt, in dem sich die beiden Prozesse jeweils befinden. Eine 0 steht für einen unkritischen, eine 1 für einen kritischen Bereich. Der Startzustand des Systems sei  $(0, 0)$ . Im Zustand  $(1, 1)$  befinden sich beide Prozesse im kritischen Abschnitt – das ist ein Fehler, und deshalb wollen wir vermeiden, in diesen Zustand zu gelangen.



Die Verifikation vereinfacht sich auf die Bestimmung der erreichbaren Zustände und den Test, ob der Fehlerzustand  $(1, 1)$  in dieser Menge liegt. Die vier Prädikate

- $S(x)$  ist wahr gdw.  $x$  ist Startzustand (also  $x = (0, 0)$ )
- $T(x, y)$  ist wahr gdw.  $y$  ist Nachfolger von  $x$
- $E(x)$  ist wahr gdw.  $x$  verletzt die Sicherheitseigenschaft (also  $x = (1, 1)$ )
- $R(x)$  ist wahr gdw.  $x$  ist erreichbar

werden wie folgt definiert:

- $S(x) := \neg x[0] \wedge \neg x[1]$
- $T(x, y) := \neg x[0] \wedge \neg y[0] \vee \neg x[1] \wedge \neg y[1]$
- $E(x) := x[0] \wedge x[1]$

Wie ist jedoch das Prädikat  $R(x)$  zu definieren?  $R(x)$  bedeutet umgangssprachlich: Die Menge der erreichbaren Zustände  $R$  ist die kleinste Menge, in der die Startzustände  $S$  liegen, und die abgeschlossen ist gegenüber Nachfolgerbildung.

Eine äquivalente Formulierung lautet: Die Menge der erreichbaren Zustände  $R$  ist die kleinste Menge, sodass für jedes  $x$  aus  $R$  gilt, dass  $x$  in  $S$  liegt oder es einen Vorgänger  $y$  von  $x$  gibt (es gilt  $T(y, x)$ ), der in  $R$  liegt. Eine mögliche Transformation in den  $\mu$ -Kalkül wäre

$$\mu R. S \vee (\exists y. T(y) \wedge \langle - \rangle R)$$

Zu zeigen ist nun: Wenn ein Zustand erreichbar ist, dann darf er die Sicherheitseigenschaft nicht verletzen:

$$R \rightarrow \neg E$$

### 3.4 Fairness

Wie wir in 3.2 gesehen haben, werden Formeln sehr ausdrucksstark, wenn man die beiden Fixpunktoperatoren alternierend verwendet. In 1.2.3 war ein Beispiel in CTL angegeben:  $\diamond \square w$  bedeutet: "w wird irgendwann Invariante.". Im  $\mu$ -Kalkül kann das wie folgt dargestellt werden:

$$\mu Y. \nu Z. [-] Z \wedge \langle - \rangle Y$$

Diese Formel bedeutet, dass ein Programm nur endlich viele  $Y$ -Zustände durchlaufen kann, bevor schließlich irgendwann einmal immer  $Z$  gelten wird.

Ein sehr komplexes Beispiel für alternierende Fixpunkte erhält man, wenn man *Fairness* ausdrücken will. Eine Aktion  $a$  wird fair behandelt, wenn es keine Pfade gibt, auf denen  $a$  unendlich oft ausgeführt werden kann, aber nur endlich oft auftritt:

$$\forall X. \mu Y. \nu Z. [a]X \wedge (\langle a \rangle tt \Rightarrow [-a]Y) \wedge [-a]Z$$

## Literatur

- [1] Armin Biere. Effiziente Modellprüfung des  $\mu$ -Kalküls mit binären Entscheidungsdiagrammen. Karlsruhe, 1997.
- [2] Günter Karjoth. Prozeßalgebra und temporale Logik - angewandt zur Spezifikation und Analyse von komplexen Protokollen. Stuttgart, 1987.
- [3] G. E. Hughes, M. J. Cresswell. Einführung in die Modallogik. de Gruyter, 1978.
- [4] Julian Bradfield, Colin Stirling. Modal logics and  $\mu$ -calculi: an introduction. Handbook of Process Algebra, Elsevier 2001.
- [5] Gilles Barthe, Dilian Gurov, Marieke Huisman. Compositional specification and verification of control flow based security properties of multi-application programs. <http://www.sics.se/~dilian/Papers/ftfjp01.ps.gz>
- [6] David A. Schmidt. Denotational Semantics. A Methodology for Language Development. Dubuque, Iowa, 1988.
- [7] Rance Cleaveland. Tableau-Based Model Checking in the Propositional Mu-Calculus. Raleigh, North Carolina, 1990. <http://www.cs.sunysb.edu/~Erance/publications/papers/ai90.ps.gz>
- [8] Alexander Kick. Generation of witnesses for global  $\mu$ -calculus model checking. Karlsruhe, <http://liinwww.ira.uka.de/~kick/cavf.ps>