Automaten und formale Sprachen

Mitschrift von www.kuertz.name

Hinweis: Dies ist **kein offizielles Script**, sondern nur eine private Mitschrift. Die Mitschriften sind teweilse **unvollständig, falsch oder inaktuell**, da sie aus dem Zeitraum 2001–2005 stammen. Falls jemand einen Fehler entdeckt, so freue ich mich dennoch über einen kurzen Hinweis per E-Mail – vielen Dank!

Klaas Ole Kürtz (klaasole@kuertz.net)

Inhaltsverzeichnis

Einführung

Anwendungsgebiete der Automatentheorie:

- Compilerbau (Kellerautomaten in Syntaxanalyse etc.)
- Verifikation (Model-Checking)
- XML (Darstellung als Baumstrukturen; Schemas, Typen etc.)
- logische Theorie
- reguläre Ausdrücke (`grep` etc.)
- automatische Spracherkennung (probabilistische Automaten)

1 LTL-Model-Checking

1.1 Einführung

Einsatz von Model-Checking Software zur Verifikation von Hardware oder Software, welche durchgehend läuft und im normalen Betrieb nicht terminiert (z.B. Protokolle, Betriebssysteme, Hardware, ...; beispielsweise ist `SPIN` ein automatisches Verifikationsinstrument).

Definition: Ein *Transitionssystem* ist $T = (P, Q, Q_I, \to, \lambda)$ mit

- einer endlichen, nichtleeren Menge P boolescher Variablen,
- einer nichtleeren Menge Q von Zuständen,
- einer nichtleeren Menge $Q_I \subseteq Q$ von Anfangszuständen,
- einer Transitionsrelation $\to \subseteq Q \times Q$
- einer Beschriftungsfunktion $\lambda\colon Q \to 2^P$, die jedem Zustand die Menge der in ihm gültigen Variablen zuordnet.

Ein System T heißt *endlich*, falls Q endlich ist. Zudem heißt T *lebendig*, falls für alle $q \in Q$ ein $q' \in Q$ existiert mit $q \to q'$.

Definition: Eine *Transitionsfolge* von T ist ein Wort $u \in Q^+ \cup Q^\omega$ mit folgenden Eigenschaften:

- $u[0] \in Q_I$

- $u[i] \to u[i+1]$ für alle i mit $i+1 < |u|$
- Falls $|u| < \infty$ ist, so gibt es kein q mit $u[|u|-1] \to q$.

Die *Beschriftung* einer Transitionsfolge ist[1]

$$\lambda(u) = \lambda(u[0])\lambda(u[1])\lambda(u[2])\ldots \in (2^P)^+ \cup (2^P)^\omega$$

Sei $S(T)$ die *Menge aller Transitionsfolgen* von T; und $L(T)$ sei die *Menge der Beschriftungen aller Transitionsfolgen* von T.

Es werden nun in LTL einige Operatoren definiert, die entsprechend kombiniert werden können:

- Menge aller Beschriftungsfolgen mit der Eigenschaft, daß *irgendwann* p gilt:
$$\mathsf{F}\,p \quad \text{oder} \quad \diamond\, p$$
- Menge aller Beschriftungsfolgen mit der Eigenschaft, daß *nie* p gilt:
$$\neg\mathsf{F}\,p \quad \text{oder} \quad \neg \diamond\, p$$
Alternative ist der Operator *immer* (**g**lobally):
$$\mathsf{G}\,\neg p \quad \text{oder} \quad \Box\neg p$$
- Im *nächsten* Zeitpunkt p (ne**x**t):
$$\mathsf{X}\,p \quad \text{oder} \quad \circ\, p$$
- *Immer wieder* p (unendlich oft p):
$$\mathsf{G}\,\mathsf{F}\,p \quad \text{oder} \quad \mathsf{G}\,\mathsf{X}\,\mathsf{F}\,p$$
Dabei hat die rechte Schreibweise den Vorteil, daß sie wirklich unendliche Folgen garantiert (die linke Eigenschaft gilt auch für endliche Folgen, bei dem im letzten Element p gilt).
- Soll irgendwann p gelten, und *von da an* $\neg r$ gelten, *bis* irgendwann q gilt (**u**ntil), schreibt man:
$$\mathsf{F}\,(p \wedge \neg r\,\mathsf{U}\,q)$$

[1] Notation: Potenzmenge einer Menge M sei 2^M, die Menge der unendlichen Folgen aus M sei M^ω.

- Komplexeres Beispiel: Es sollen drei p erreicht werden, zwischen denen q gilt, d.h. im Prinzip $pq \ldots qpq \ldots qp$:

$$\mathsf{F}\,(p \wedge \mathsf{X}\,(q\,\mathsf{U}\,(p \wedge \mathsf{X}\,(q\,\mathsf{U}\,(p \wedge \mathsf{X}\,(q\,\mathsf{U}\,p))))))$$

- Wenn ψ immer gelten soll, zumindest bis φ es „*erlöst*“, d.h. bis einmal $\varphi \wedge \psi$ gilt:

$$\varphi\,\mathsf{R}\,\psi$$

Definition: Sei P eine Menge boolescher Variablen; sei LTL_P definiert durch

- $\mathtt{tt} \in \mathrm{LTL}_P$ ($\mathtt{tt}$ entspricht *wahr*)
- $P \subseteq \mathrm{LTL}_P$
- Falls $\varphi \in \mathrm{LTL}_P$, so auch $(\neg\varphi) \in \mathrm{LTL}_P$.
- Falls $\varphi, \psi \in \mathrm{LTL}_P$, so auch $(\varphi \vee \psi) \in \mathrm{LTL}_P$.
- Falls $\varphi \in \mathrm{LTL}_P$, so auch $(\mathsf{X}\,\varphi) \in \mathrm{LTL}_P$.
- Falls $\varphi, \psi \in \mathrm{LTL}_P$, so auch $(\varphi\,\mathsf{U}\,\psi) \in \mathrm{LTL}_P$.

Abkürzungen sind dann:

- $\mathsf{F}\,\varphi = \mathtt{tt}\,\mathsf{U}\,\varphi$
- $\mathsf{G}\,\varphi = \neg\mathsf{F}\,\neg\varphi$
- $\varphi\,\mathsf{R}\,\psi = \neg(\neg\varphi\,\mathsf{U}\,\neg\psi)$

Definition: Die **Semantik** von LTL: Sei $u \in (2^P)^+ \cup (2^P)^\omega$. Es gilt

- $u \models \mathtt{tt}$
- $u \models p$ gdw. $p \in u[0]$
- $u \models \neg\varphi$ gdw. $u \not\models \varphi$
- $u \models \varphi \vee \psi$ gdw. $u \models \varphi$ oder $u \models \psi$
- $u \models \mathsf{X}\,\psi$ gdw. $|u| > 1$ und $u[1, *) \models \varphi$
- $u \models \varphi\,\mathsf{U}\,\psi$ gdw. ein $0 \leq j < |u|$ existiert, so daß $u[j, *) \models \psi$ und für alle $i < j$ gilt: $u[i, *) \models \varphi$

Beispiele:

- Sei $u = \{p\}\{p,q\}\{\}\{\}\{p\}$. Dann gilt:
  - $u \not\models q$, $u \not\models \neg p$, $u \models p$
  - $u \not\models \neg p \,\mathsf{U}\, q$, $u \models p \,\mathsf{U}\, q$
  - $u \models \mathsf{XXXXX}T$, $u \not\models \mathsf{XXXXXX}T$
- Eine Folge u ist endlich, falls $u \models \mathsf{F}\,\neg\mathsf{X}\,\mathtt{tt}$.

Bemerkung: In dieser Sprache ist z.B. „gerade Länge“ nicht ausdrückbar, und Zählen (z.B. „gleichviele p und q“) ist unmöglich.

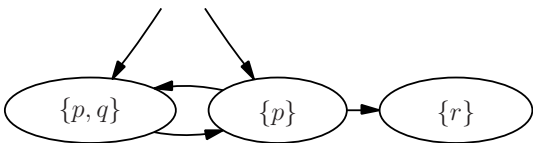
Definitionen:

- Die LTL-*Sprache* über einer Menge P, die die Formel φ erfüllen, sei
$$L_P(\varphi) = \left\{ u \in (2^P)^+ \cup (2^P)^\omega \;\middle|\; u \models \varphi \right\}$$
- $L_P^\omega(\varphi)$ bezeichne $L_P(\varphi) \cap (2^P)^\omega$.
- T *erfüllt* eine Formel φ (geschrieben $T \models \varphi$) genau dann, wenn gilt:
$$L(T) \subseteq L_P(\varphi)$$

Bemerkung: Für das Model-Checking betrachten wir nur *endliche* T.

Ziel/Frage: Das eigentliche Model-Checking-Problem: Seien T (endlich) und $\varphi \in \mathrm{LTL}_p$ gegeben, gilt nun $T \models \varphi$? Eine Beispiel für eine Formel/Eigenschaft, die einfach überprüft werden kann: $T \not\models \mathsf{F}\,\neg\mathsf{X}\,\mathtt{tt}$ genau dann, wenn es im Graphen $(Q, \to)$ einen Zustand $q \in Q_I$ sowie eine starke Zusammenhangskomponente C gibt, die von q aus erreichbar ist.

Beispiel: Gegeben sei der folgende Graph:



Dann gilt $T \models p$, $T \not\models q$, $T \not\models \mathsf{G}\,p$, $T \models \mathsf{F}\,(q \vee r)$ und $T \models \mathsf{GF}\,(q \vee r)$.

1.2 Büchi-Automaten

Hintergrund: Büchi wollte prüfen, ob Formeln über $(\mathbb{N}, \mathrm{succ})$ mit Logik- und Mengenquantoren entscheidbar und berechenbar sind. Hierzu benötigte er Automaten, die unendliche Wörter verarbeiten können.

Definition: Ein *Büchi-Automat (BA)* sei $B = (A, S, S_I, \Delta, F)$ mit

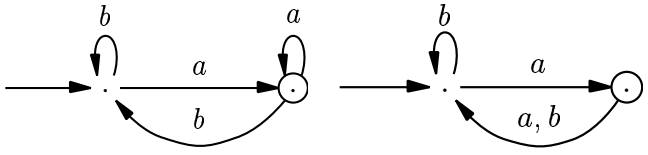
- einem Alphabet A,
- einer endliche Zustandsmenge S,
- der Menge der Anfangszustände $S_I \subseteq S$,
- der Transitionsrelation $\Delta \subseteq S \times A \times S$,
- einer Menge von *akzeptierenden Zuständen* $F \subseteq S$.

Definitionen:

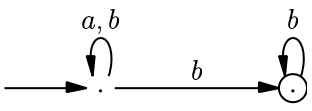
- Ein *Lauf* von $u \in A^\omega$ auf B ist eine Folge $s[0]s[1] \ldots \in S^\omega$ mit $s[0] \in S_I$ und $(s[i], u[i], s[i+1]) \in \Delta$.
- Ein *Lauf* wird *akzeptiert*, wenn es unendlich viele i gibt mit $s[i] \in F$.
- Ein Wort $u \in A^\omega$ wird *akzeptiert*, falls es einen akzeptierenden Lauf von u auf B gibt.
- Entsprechend sei $L(B) = \{\, u \in A^\omega \mid B \text{ akzeptiert } u \}$.

Beispiele:

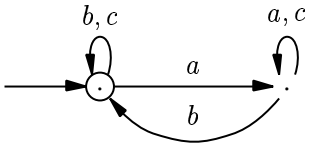
- Folgende Automaten (nicht isomorph!) akzeptieren die Sprache „*unendlich oft* a“ $= [\![(b^*a)^\omega]\!]$:



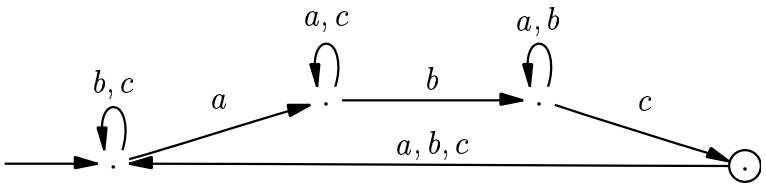
- „*nur endlich viele* a“ $= [\![(a+b)^*b^\omega]\!]$:



- „*hinter jedem a irgendwann ein b*“ mit einem Alphabet $A = \{a, b, c\}$:



- „*a, b und c jeweils unendlich oft*“:



Weiteres Vorgehen: Unser grundsätzliches Ziel ist, zu prüfen, ob $L(T) \subseteq L(\varphi)$ ist für ein Transitionssystem T und eine Formel φ. Wir konstruieren dazu einen Automaten B, so daß $L(B) = L(\neg\varphi)$ gilt. Dann ist nur noch $L(T) \cap L(B) = \emptyset$ zu prüfen. Dazu „kombinieren“ wir die Automaten T und B, Schreibweise $T \ltimes B$, und prüfen $L(T \ltimes B) = \emptyset$.

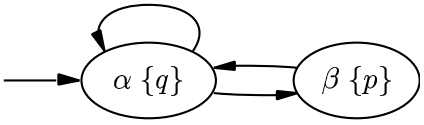
Definition: Seien folgende Automaten gegeben:

$$\begin{aligned} T &= (P, Q, Q_I, \rightarrow, \lambda) \\ B &= (2^P, S, S_I, \Delta, F) \end{aligned}$$

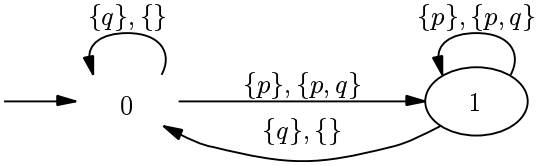
Dann definieren wir $T \ltimes B$ durch

$$\begin{aligned} T \ltimes B &= (Q, Q \times S, Q_I \times S_I, \Delta', Q \times F) \\ \text{mit } \Delta' &= \{\, ((q, s), q, (q', s')) \mid (s, \lambda(q), s') \in \Delta, q \rightarrow q' \} \end{aligned}$$

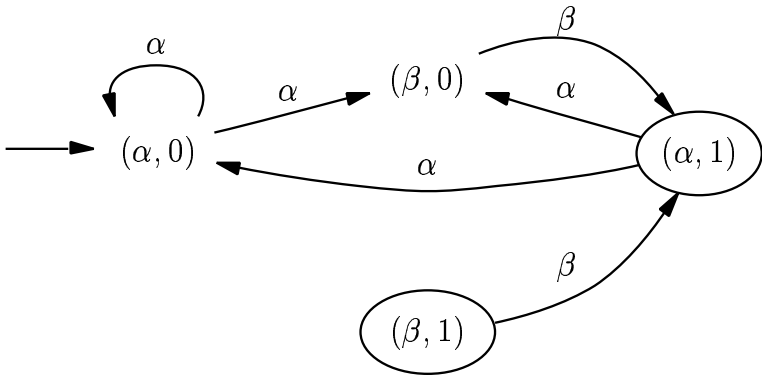
Beispiel: Sei das folgende Transitionssystem T gegeben:



Sei $\varphi = \neg \mathsf{G}\,\mathsf{F}\,p$, der folgende Büchi-Automat beschreibt die Formel $\neg\varphi$:



Der Büchi-Automat $T \ltimes B$ ergibt sich dann als



Satz: Seien T und B wie oben. Dann gilt für alle $u \in Q^\omega$: $u \in L(T \ltimes B)$ genau dann, wenn $u \in S(T)$ ist und $\lambda(u) \in L(B)$.

Beweis:

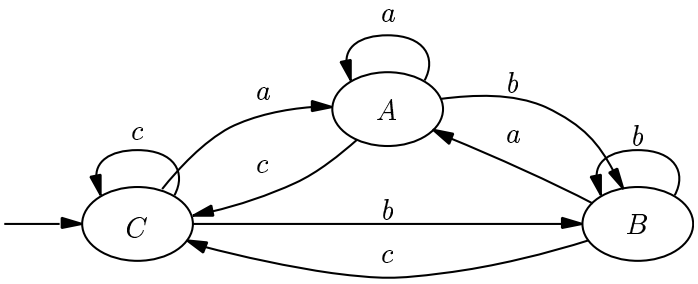
„$\Leftarrow$“ Sei $u \in S(T)$ mit $\lambda(u) \in L(B)$. Dann gibt es $u[0] \in Q_I$, $u[i] \to u[i+1]$ für alle i, und es existiert s mit $s[0] \in S_I$, $(s[i], \lambda(u[i]), s[i+1]) \in \Delta$, und es existieren unendlich viele i mit $s[i] \in F$.

Damit ist $(u[0], s[0])(u[1], s[1])(u[2], s[2]) \ldots$ ein akzeptierender Lauf von $T \ltimes B$ auf u, da gilt: $u[0] \in Q_I$ und $s[0] \in S_I$ (Startzustand); außerdem ist jeweils $u[i] \to u[i+1]$ und $(s[i], \lambda(u[i]), s[i+1]) \in \Delta$ (Übergänge). Für die akzeptierende Zustände sehen wir: Sei $i_0 < i_1 < \ldots$ eine Folge mit $s[i_j] \in F$. Dann ist auch $(u[i_j], s[i_j]) \in Q \times F$.

„$\Rightarrow$“ ähnlich □

Definition: Ein *verallgemeinerter Büchi-Automat (GBA)* ist von der Form $(A, S, S_I, \Delta, \mathcal{F})$, wobei $\mathcal{F} \subseteq 2^S$ ist. In einem *akzeptierenden Lauf* gibt es nun für alle $F \in \mathcal{F}$ unendlich viele i mit $s[i] \in F$.

Beispiel: „a, b und c jeweils unendlich oft“: Betrachte folgenden Automaten mit $\mathcal{F} = \{\{A\}, \{B\}, \{C\}\}$.



Konstruktion: Zur Überführung eines verallgemeinerten Büchi-Automaten in einen „normalen“ Büchi-Automaten gibt es zwei Verfahren. Sei dazu ein verallgemeinerter Büchi-Automat $(A, S, S_I, \Delta, \{F_0, \ldots, F_{k-1}\})$ gegeben.

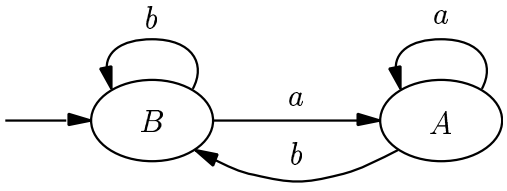
- $(A, S \times \{0,1\}^k, S_I \times \{0\}^k, \Delta', S \times \{1\}^k)$; und für $(q, a, q') \in \Delta$ sei $((q, b_0, \ldots, b_{k-1}), a, (q', b'_0, \ldots, b'_{k-1})) \in \Delta'$ genau dann, wenn
  - $b'_i = 1$, falls $b_i = 1$ oder $q' \in F_i$
  - Ausnahme: falls alle $b_i = 1$, so $b'_i = 0$ für alle i

  Diese Methode führt allerdings zu einer Explosion der Anzahl der Zustände!
- $(A, S \times \{0, \ldots, k\}, S_I \times \{0\}, \Delta', S \times \{k\})$ mit $((s, i), a, (s', j)) \in \Delta$ genau dann, wenn $(s, a, s') \in \Delta$ ist und
  - $i < k$, $s' \in F_i$ und $j = i + 1$
  - oder $i < k$, $s \notin F_i, j = i$
  - oder $i = k$ und $j = 0$

Bemerkungen:

- Die Konstruktion eines Büchi-Automaten aus einem verallgemeinrten Büchi-Automaten bezeichnen wir mit $\mathrm{GBA_2BA}(B)$.
- Die Umwandlung ist bis auf Isomorphie eindeutig.
- Der überführte Automat hat $|S| \cdot (|\mathcal{F}| + 1)$ Zustände.

Beispiel: Sei der folgende verallgemeinerte Büchi-Automat gegeben mit $\mathcal{F} = \{F_0 = \{A\}, F_1 = \{B\}\}$:



Daraus ergibt sich folgender Büchi-Automat aus Bild 1.

Satz: Für B und B' gilt $L(B) = L(B')$.

Lemma: Sei B ein Büchi-Automat. Dann sind äquivalent:

- $L(B) \neq \emptyset$

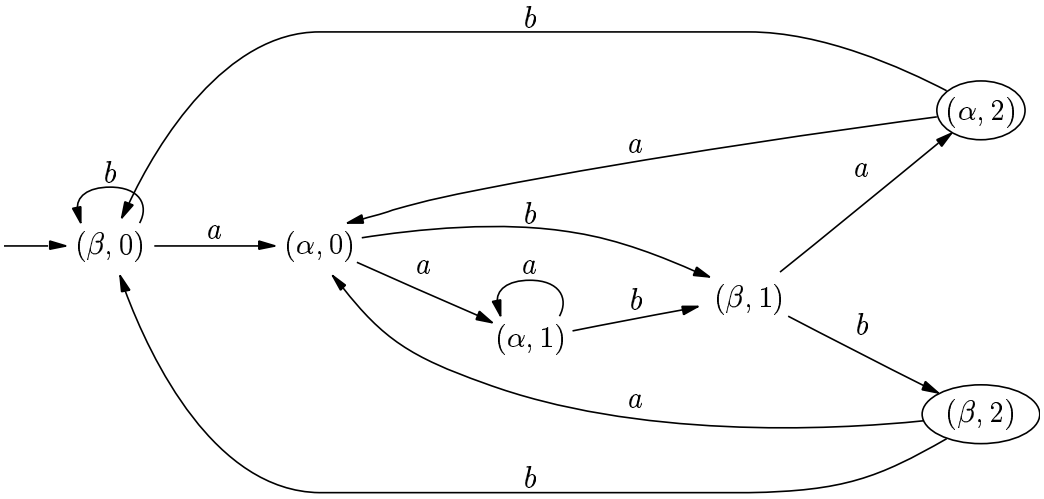


Abbildung 1: Beispiel: Ergebnis von $\mathrm{GBA_2BA}(B)$

- Es gibt eine starke Zusammenhangskomponente, die einen akzeptierenden Zustand enthält und von einem Startzustand aus erreichbar ist.
- Es gibt $s_I \in S_I$ und $f \in F$, so daß f von s_I aus erreichbar und f über einen nicht-trivialen Pfad von f aus erreichbar ist.
- Es gibt ein *schließlich periodisches Wort*, d.h. $u, v \in A^+$ mit $uv^\omega \in L(B)$.

Satz: Der Leerheitstest für Büchi-Automaten ist entscheidbar, sogar lösbar in Zeit $\mathcal{O}(|S| + |\Delta|)$ und liegt in NL[2].

1.3 Übersetzung von LTL in (G)BA

Ziel: Gegen sei $\varphi \in \mathrm{LTL}_P$. Konstruiere nun (G)BA B mit $L_P^\omega(\varphi) = L(B)$.

Erster Schritt: Umwandlung von φ in *positive Normalform (PNF)*, d.h. Negationen dürfen nur vor booleschen Variablen vorkommen.

Notation: Es sei $\varphi \Leftrightarrow_\omega \psi$ genau dann, wenn für alle unendlichen Beschriftungsfolgen u gilt: $u \models \varphi \Leftrightarrow u \models \psi$.

[2] NL: logarithmischer Platzverbrauch auf nicht-deterministischer Turing-Maschine

Lemma: Für alle LTL-Formeln φ und ψ über einer gemeinsamen Menge P von booleschen Variablen gilt:

- $\neg \mathtt{tt} \Leftrightarrow_\omega \mathtt{ff}$ und $\neg \mathtt{ff} \Leftrightarrow_\omega \mathtt{tt}$, außerdem $\neg\neg\varphi \Leftrightarrow_\omega \varphi$
- $\neg(\varphi \wedge \psi) \Leftrightarrow_\omega \neg\varphi \vee \neg\psi$ und $\neg(\varphi \vee \psi) \Leftrightarrow_\omega \neg\varphi \wedge \neg\psi$
- $\neg \mathsf{X}\,\varphi \Leftrightarrow_\omega \mathsf{X}\,\neg\varphi$ (gilt nicht für endliche Wörter!)
- $\neg(\varphi \,\mathsf{U}\, \psi) \Leftrightarrow_\omega \neg\varphi \,\mathsf{R}\, \neg\psi$ und $\neg(\varphi \,\mathsf{R}\, \psi) \Leftrightarrow_\omega \neg\varphi \,\mathsf{U}\, \neg\psi$

Bemerkung: Aus einer beliebigen LTL_P-Formel läßt sich eine äquivalente $\mathrm{LTL}_P[\mathtt{ff}, \wedge, \mathsf{R}]$-Formel in positiver Normalform in Linearzeit berechen, wir schreiben dafür $\mathrm{LTL}_2\mathrm{PNF}(\varphi)$.

Beispiel:

$$\begin{aligned} \neg(p \wedge (q \,\mathsf{U}\, (\neg r \vee p))) &\Leftrightarrow_\omega \neg p \vee \neg(q \,\mathsf{U}\, (\neg r \vee p)) \\ &\Leftrightarrow_\omega \neg p \vee \neg q \,\mathsf{R}\, \neg(\neg r \vee p) \\ &\Leftrightarrow_\omega \neg p \vee \neg q \,\mathsf{R}\, (r \wedge \neg p) \end{aligned}$$

Sei φ nun eine $\mathrm{LTL}_P[\mathtt{ff}, \wedge, \mathsf{R}]$-Formel in PNF. Wir konstruieren daraus einen GBA $B = (2^P, S, S_I, \Delta, \mathcal{F})$, der $L^\omega_P(\varphi)$ erkennt.

Idee: Jeder Zustand beschreibt eine Menge von Teilformeln von φ. Wenn der Automat in diesem Zustand ist, erfüllt die zu lesende Folge an dieser Stelle diese Formeln.

Formal: Sei also $S \subseteq 2^{\mathrm{sub}(\varphi)}$, d.h. die Potenzmenge der Menge der Teilformeln von φ. Konstruiere dann B derart, daß für alle $u \in (2^P)^\omega$ und $\Phi \in S$ gilt: u wird von B mit Anfangszustand Φ akzeptiert genau dann, wenn $\Phi \subseteq \{\psi \in \mathrm{sub}(\varphi) \mid u \models \psi\}$. Dann gilt: Wenn $S_I = \{\Phi \mid \varphi \in \Phi\}$, so ist $L_\omega(B) = L(\varphi)$.

Beispiele:

- Sei $\varphi = \mathsf{G}\,(p \vee q)$. Dann ist $\mathrm{sub}(\varphi) = \{p, q, p \vee q, \mathsf{G}\,(p \vee q)\}$. Der Graph in Abbildung 2 stellt dann alle sinnvollen (d.h. erreichbaren) Zustände dar.
- Sei $\varphi = \mathsf{F}\,(p \vee q)$. Der zugehörige Automat ist in Abbildung 3 dargestellt.

Konstruktion: Wir konstruieren wie oben begonnen $B = (2^P, S, S_I, \Delta, \mathcal{F})$ durch folgende Mengen:

- Zu S: Es gilt $\Phi \in S$ genau dann, wenn

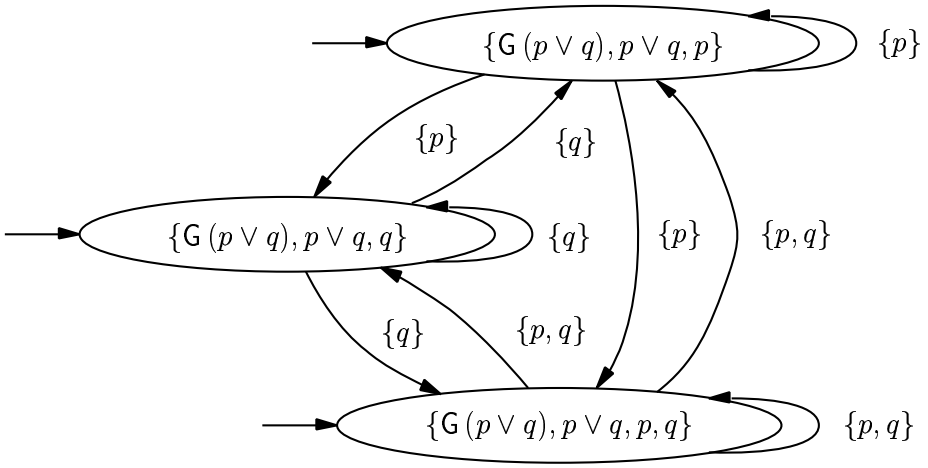


Abbildung 2: Automat zu $\varphi = \mathsf{G}\,(p \vee q)$

  - $\mathtt{ff} \notin \Phi$, $\mathtt{tt} \in \Phi$
  - $p \in \Phi$ genau dann, wenn $\neg p \notin \Phi$
  - wenn $(\psi \vee \xi) \in \Phi$, dann $\psi \in \Phi$ oder $\xi \in \Phi$
  - wenn $(\psi \wedge \xi) \in \Phi$, dann $\psi \in \Phi$ und $\xi \in \Phi$

- Zu S_I: Sei $S_I = \{\,\Phi \mid \varphi \in \Phi\}$.
- Zu Δ: Diese Menge enthält $(\Phi, a, \Psi) \in \Delta$ mit folgenden Bedingungen:
  - es gilt $p \in \Phi$ genau dann, wenn $p \in a$ ist
  - wenn $(\mathsf{X}\,\psi) \in \Phi$ ist, dann $\psi \in \Psi$
  - wenn $(\psi\,\mathsf{U}\,\xi) \in \Phi$, dann

$$(\xi \in \Phi) \quad \text{oder} \quad ((\psi \in \Phi) \text{ und } (\psi\,\mathsf{U}\,\xi \in \Psi))$$

  - wenn $(\psi\,\mathsf{R}\,\xi) \in \Phi$, dann

$$(\xi \in \Phi) \quad \text{und} \quad ((\psi \in \Phi) \text{ oder } (\psi\,\mathsf{R}\,\xi \in \Psi))$$

- Zu $\mathcal{F}$: Für jedes $\psi \in \mathrm{sub}(\varphi)$ der Gestalt $\psi = \xi\,\mathsf{U}\,\xi'$ enthält $\mathcal{F}$ die Menge

$$F_\psi = \{\,\Phi \in S \mid (\psi \notin \Phi) \vee (\xi' \in \Phi)\}$$

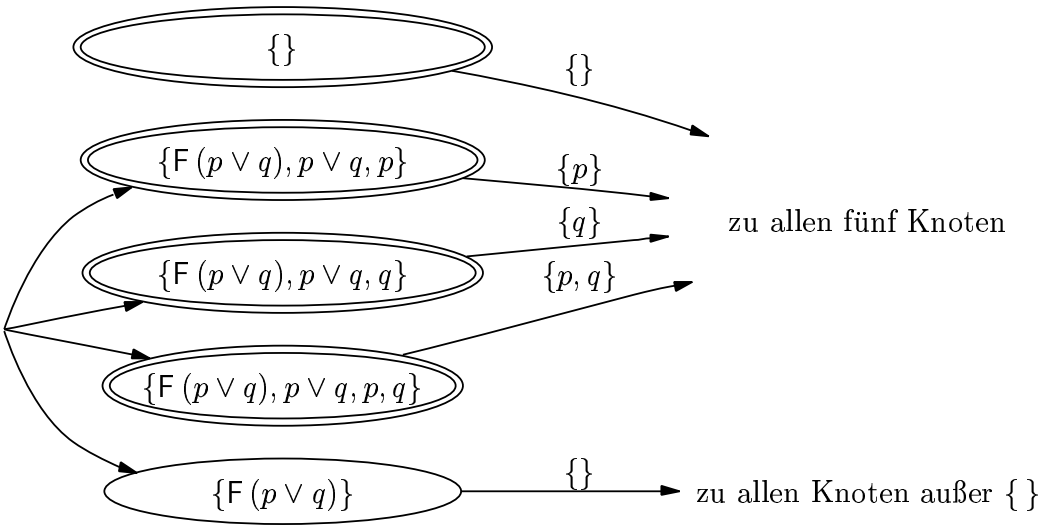


Abbildung 3: Automat zu $\varphi = \mathsf{F}\,(p \vee q)$

Satz: Sei φ eine $\mathrm{LTL}_P[\mathtt{ff}, \wedge, \mathsf{R}]$-Formel in positiver Normalform. Dann gilt:

$$L(B) = L_\omega(\varphi) \;\text{ mit }\; B = \mathrm{LTL}_2\mathrm{GBA}(\varphi)$$

Beweis:

„⇒“ Sei $u \models \varphi$ mit $u \in (2^P)^\omega$. Definiere $v \in S^\omega$ durch

$$v[i] = \{\, \psi \in \mathrm{sub}(\varphi) \mid u[i, *) \models \psi\}$$

Wir zeigen: v ist ein akzeptierender Lauf von u auf B. Zu zeigen ist:

1. $v[i]$ ist ein Zustand des Automaten – dies ist offensichtlich nach Definition des Automaten und der $v[i]$.
2. $v[0]$ ist Anfangszustand – dies ist klar nach Definition von $v[0]$.
3. $(v[i], u[i], v[i+1]) \in \Delta$ für alle i – leicht einzusehen nach Definition der Transitionsmenge Δ
4. v ist akzeptierend, d.h. für alle $\psi = \xi \,\mathsf{U}\, \xi'$ existieren unendlich viele i mit $v[i] \in F_\psi$

Zum vierten Punkt: Sei $\psi = \xi \,\mathsf{U}\, \xi'$. Dann ist

$$F_\psi = \underbrace{\{\, \Phi \mid \psi \notin \Phi\}}_{F_0} \cup \underbrace{\{\, \Phi \mid \xi' \in \Phi\}}_{F_1}$$

– Erster Fall: Es gibt endlich viele i mit $u[i,*) \models \psi$. Also existiert $i^\star$ mit $u[i,*) \not\models \psi$ für alle $i > i^\star$. Damit ist auch $v[i] \in F_0$ für alle $i > i^\star$, also existieren unendlich viele i mit $v[i] \in F_0$.
– Zweiter Fall: Es gibt unendlich viele i mit $u[i,*) \models \psi$. Sei $i_0 < i_1 < i_2 < \ldots$ eine derartige Folge mit $u[i_j,*) \models \psi$ für alle j.

  Dann gibt es i'_0 mit $i'_0 \geq i_0$ und $u[i'_0,*) \models \xi'$, also $v[i'_0] \in F_1$. Wähle k mit $i_k > i'_0$. Dann gibt es i'_k mit $i'_k \geq i_k$ und $u[i'_k,*) \models \xi'$, also $v[i'_k] \in F_1$.

Damit ist v ein akzeptierender Lauf.

„⇐“ Sei v ein akzeptierender Lauf von u auf B. *Behauptung*: Für alle i und für alle $\psi \in v[i]$ gilt $u[i,*) \models \psi$. Damit wären wir fertig, da ja $\varphi \in v[0]$ gilt nach Definition der Anfangszustandsmenge.

Beweis per Induktion über den Aufbau der Formeln in PNF:

– Induktionsanfang für jedes $p \in P$:
  * Falls $\psi = p$ ist, so ist $(v[i], u[i], v[i+1]) \in \Delta$ mit $p \in u[i]$. Also $u[i,*) \models p$.
  * Falls $\psi = \neg p$ ist, so ist $(v[i], u[i], v[i+1]) \in \Delta$ mit $p \notin u[i]$. Also $u[i,*) \models \neg p$.
– Induktionsschritt:
  * Für $\wedge$ und $\vee$ ist der Schluß trivial aufgrund der Konstruktion der Zustände.
  * Bei $\psi = \mathsf{X}\xi$ gilt $(v[i], u[i], v[i+1]) \in \Delta$ mit $\xi \in v[i+1]$. Nach Induktionsvoraussetzung gilt damit $u[i+1,*) \models \xi$, also $u[i,*) \models \mathsf{X}\xi = \psi$.
  * Betrachte den Fall $\psi = \xi \,\mathsf{U}\, \xi'$, es gilt wegen Definition der Transitionsrelation ($(v[i], u[i], v[i+1]) \in \Delta$) einer der beiden Fälle:
    · $\xi' \in v[i]$, also nach Induktionsvoraussetzung $u[i,*) \models \xi'$
    · $\xi \in v[i]$ und $\psi \in v[i+1]$, also $u[i,*) \models \xi$. Dann kann man auf $i+1$ das gleiche Argument anwenden und erhält *entweder* $u[i+1,*) \models \xi'$ (damit auch $u[i,*) \models \psi$) *oder* $u[i+1,*) \models \xi$ (und damit $\psi \in v[i+2]$).

  Übrig bleibt der Fall, daß für jedes $j > i$ gilt: $\psi \in v[j]$ und $u[j,*) \models \xi$. Wegen der Akzeptierbedingung gibt es dann unendlich viele j' mit $\xi' \in v[j']$, also auch ein $j' > i$ mit $\xi' \in v[j']$.

Damit ist nach Induktionsvoraussetzung $u[j', *) \models \xi'$, also $u[i, *) \models \psi$.

* Der Fall $\psi = \xi \,\mathsf{R}\, \xi'$ ist ähnlich. □

1.4 Lösung des LTL-Model-Checking-Problems

Eingabe ist eine Formel φ und ein lebendiges Transitionssystem T.

Schritte:

1. Bringe $\neg\varphi$ in positive Normalform: $\varphi' = \mathrm{LTL_2PNF}(\neg\varphi)$.
2. Erstelle verallgemeinerten Büchi-Automaten für φ': $B = \mathrm{LTL_2GBA}(\varphi')$.
3. Konvertiere B in einen Büchi-Automaten: $B' = \mathrm{GBA_2BA}(B)$.
4. Bilde das Produkt: $B'' = T \ltimes B'$.
5. Bestimme, ob $L(B'') = \emptyset$ gilt – falls ja, gibt „$T \models \varphi$“ aus, andernfalls „$T \not\models \varphi$“.

   Erweiterung: Für $T \not\models \varphi$ wird ein *error trace* $u \in L(B'')$ bestimmt.

Grobe Vorüberlegung zur **Laufzeit**: Der erste Schritt liefert einen doppelt so großen Automaten, Schritt zwei ergibt einen in der Größe der Formel exponentiellen Automaten. Schritt drei ergibt nur einen polynomiellen Faktor, im vierten Schritt ergibt sich das Produkt $|T| \cdot |B'|$, und der letzte Schritt ist linear in Größe des dieses Produkts.

Satz: Der Algorithmus löst das Model-Checking-Problem in Zeit $2^{\mathcal{O}(|\varphi|)} \cdot |T|$.

1.5 Komplexität des LTL-Model-Checking-Problems

Vorbemerkung zur Komplexitätstheorie: PSPACE ist die Menge der Sprachen, die durch eine deterministische Turing-Maschine in polynomiellem Platz entschieden werden. Nach Savitch ist dies gleich der Sprachen, die durch nichtdeterministische Turing-Maschinen in polynomiellem Platz entschieden werden. Es gilt

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

Desweiteren bedeutet PSPACE-vollständig, daß ein Problem PSPACE-schwer ist und in PSPACE liegt.

Wir zeigen nun zunächst, daß das LTL-Model-Checking-Problem in PSPACE liegt. Danach werden wir zeigen, daß es PSPACE-schwer ist, indem wir zeigen,

daß das komplementäre Problem PSPACE-schwer ist und ausnutzen, daß für ein Problem in einer deterministischen Komplexitätsklasse auch das komplementäre Problem in dieser Klasse liegt. Daß das zum Model-Checking komplementäre Problem PSPACE-schwer ist, zeigen wir durch eine Reduktion von Quantified Boolean Formulas auf das LTL-Erfüllbarkeitsproblem, letzteres reduzieren wir dann auf das komplementäre Model-Checking-Problem.

Satz: Das LTL-Model-Checking-Problem ist in PSPACE.

Benutze folgendes Lemma:

Lemma: Sei $f\colon \Sigma^* \to \Gamma^*$ eine Funktion, die in polynomiellem Platz berechnet werden kann und $L \subseteq \Gamma^*$ eine formale Sprache, die zu NL gehört. Dann gilt $f^{-1}(L) \in$ PSPACE.

Wenn nun f die Konstruktion von Transitionssystem und LTL-Formel zu einem Büchi-Automaten beschreibt (in polynomiellem Platz berechenbar) und L die Menge der Büchi-Automaten, die eine leere Sprache liefern, so erhalten wir, daß LTL-Model-Checking in PSPACE liegt.

Wir betrachten nun zunächst das *LTL-Erfüllbarkeitsproblem*, d.h. die Sprache

$$\left\{ \varphi \in \mathrm{LTL}_P \mid \ \exists\, u \in (2^P)^\omega \colon u \models \varphi \right\}$$

Satz: Das LTL-Erfüllbarkeitsproblem ist PSPACE-schwer.

Das heißt, jedes Problem aus PSPACE ist auf das LTL-Erfüllbarkeitsproblem reduzierbar, bzw. ein PSPACE-vollständiges Problem ist auf dieses Problem reduzierbar.

Als PSPACE-vollständiges Problem benutzen wir für den Beweis *Quantified Boolean Formulas (QBF)*. Das sind Formeln, in denen Quantoren über boolesche Variablen benutzt werden, etwa $\forall x\ \exists\ y(x \vee y)$ oder $\forall x\ \exists\ y(x \wedge y)$.

Formal ist QBF die Sprache

$$\{\varphi \text{ quantifizierte, aussagenlogische Formel} \mid \ \varphi \text{ gilt}\}$$

Beweis des obigen Satzes: Sei φ' eine aussagenlogische Formel und

$$\varphi = Q_n x_n Q_{n-1} x_{n-1} \ldots Q_1 x_1 \varphi' \quad \text{mit} \quad Q_i \in \{\exists, \forall\}$$

Frage ist, ob φ wahr ist. Zu φ konstruieren wir eine LTL-Formel ψ (in Polynomzeit), so daß φ wahr ist genau dann, wen ψ erfüllbar ist.

Idee:

- Wir definieren $P = \{p_1, \ldots, p_n, b_0, b_1, \ldots, b_n, c_0, c_1, \ldots, c_n\}$ – dann betrachten wir Wörter über 2^P.
- Wir konstruieren eine LTL-Formel, deren Modelle (d.h. Wörter) ausgeschriebene Belege dafür sind, daß die Formel erfüllbar ist. Dafür konstruieren wir zunächst ein Wort, welches nur existieren kann, falls die Formel φ gilt. Daraus entwickeln wir eine LTL-Formel, die genau dieses Wort erwartet, d.h. die Formel ist genau dann erfüllbar, wenn ein solches Wort existiert, wie wir es induktiv definieren werden.
- Zunächst muß während des ganzen Wortes die Formel φ' gelten, wobei hierin die x_i durch p_i ersetzt wurden.
- Sei nun ξ ein hinterer Abschnitt der Formel, und u gegeben für ξ. Wir definieren jetzt für $\exists x_i \xi$ und $\forall x_i \xi$ entsprechende Wörter u'.
  - Für einen Existenzquantor muß es eine Belegung geben. D.h. falls etwa $\exists x_i \xi$ der hintere Abschnitt der Formel ist, so muß es für jeden Buchstaben in u, der aus ξ resultiert, einen Buchstaben in u' geben, der belegt, daß es ein x_i gibt, welches ξ erfüllt. Es muß also entweder p_i oder $\neg p_i$ in diesem Buchstaben gelten.
  - Für einen Allquantor müssen beide Möglichkeiten ausprobiert werden. D.h. falls $\forall x_i \xi$ der hintere Abschnitt der Formel ist, so muß es für jeden Buchstaben in u, der aus ξ resultiert, *zwei* Buchstaben in u' geben, wobei in einem p_i gilt und im anderen $\neg p_i$.
- Um diese Abschnitte zu markieren und unterscheiden zu können, verwenden wir zudem Markierungen b_i bzw. c_i, wobei b_i für den Anfang des Abschnitts steht, das die Teilformel $Q_i x_i \xi$ repräsentiert, und c_i das Ende dieses Abschnitts markiert. Gleichzeitig soll das ganze Wort über b_0 und c_0 gelten.
- Da nur unendliche Worte betrachtet werden, muß zudem der endliche Abschnitt, der die Gesamtformel φ repräsentiert, immer wieder wiederholt werden.

Formal: Wir definieren $\psi = \bigwedge_{i=0}^{n} \psi_i$, wobei ψ_i definiert ist als:

- $\psi_0 = \mathsf{G}\,(b_0 \wedge c_0 \wedge \varphi')$
- für $Q_i = \exists$ ist

$$\psi_i = \underbrace{\mathsf{G}\,((b_{i-1} \leftrightarrow b_i) \wedge (c_{i-1} \leftrightarrow c_i))}_{(1)} \wedge \underbrace{\mathsf{G}\,(\neg \mathsf{X}\, b_i \rightarrow (p_i \leftrightarrow \mathsf{X}\, p_i))}_{(2)}$$

Diese Formel garantiert durch (1), daß der Existenzquantor die „Grenzen“ der unterliegenden Abschnitte übernimmt, und durch (2), daß im ganzen Abschnitt zwischen b_i und c_i immer das gleiche für x_i gilt (d.h. entweder p_i oder $\neg p_i$).

- für $Q_i = \forall$ ist

$$\begin{array}{rll} \psi_i = & \mathsf{G}\,(\neg \mathsf{X}\, b_{i-1} \rightarrow (p_i \leftrightarrow \mathsf{X}\, p_i)) & (1) \\ \wedge & \mathsf{G}\,(\mathsf{X}\, b_{i-1} \rightarrow (p_i \leftrightarrow \neg \mathsf{X}\, p_i)) & (2) \\ \wedge & \mathsf{G}\,((b_{i-1} \wedge p_i) \leftrightarrow b_i) & (3) \\ \wedge & \mathsf{G}\,((c_{i-1} \wedge \neg p_i) \leftrightarrow c_i) & (4) \end{array}$$

  Dabei stehen die einzelnen Bestandteile für folgende Forderungen:

  (1) Innerhalb der beiden einzelnen Abschnitte muß konstant p_i bzw. $\neg p_i$ gelten.

  (2) Im ersten Abschnitt gilt p_i, im zweiten Abschnitt gilt $\neg p_i$.

  (3) b_i markiert den Anfang des Abschnitts für diesen Quantor (dort gelten b_{i-1} und p_i).

  (4) c_i markiert das Ende des Abschnitts für diesen Quantor (dort gelten c_{i-1} und $\neg p_i$).

Existiert nun ein Wort, das diese Formel ψ erfüllt, so beschreibt dieses alle nötigen Belegungen für die Variablen $x_1, \ldots, x_n$, um φ zu erfüllen. □

Beispiel: Betrachte folgende Formel:

$$\varphi = \forall x_3 \forall x_2 \exists x_1 (x_2 \wedge x_3 \rightarrow \neg x_1)$$

Dann würde sich ein Wort u ergeben mit $u[i] = u[i \bmod 4]$, wobei $u[0,3]$ definiert ist als:

$u[0,3] =$

	$u[0]$	$u[1]$	$u[2]$	$u[3]$
p_1		(✓)	(✓)	(✓)
p_2	✓		✓	
p_3	✓	✓		
b_0	✓	✓	✓	✓
c_0	✓	✓	✓	✓
b_1	✓	✓	✓	✓
c_1	✓	✓	✓	✓
b_2	✓		✓	
c_2		✓		✓
b_3	✓			
c_3				✓

Damit gilt:

Folgerung: Das LTL-Erfüllbarkeitsproblem ist PSPACE-schwer, schon dann, wenn nur X und G zugelassen werden.

Beziehung zum Model-Checking-Problem:

Satz: Das Erfüllbarkeitsproblem für LTL ist in polynomieller Zeit reduzierbar auf das komplementäre Model-Checking-Problem.

Beweis: Erste Idee: φ (mit $p_0, \ldots, p_{n-1}$) erfüllbar genau dann, wenn für ein Transitionssystem T mit $L(T) = (2^P)^\omega$ (d.h. es läßt alle Worte zu!) gilt: $T \not\models \neg\varphi$. Dabei konstruieren wir $T = (P, Q, Q_I, \to, \lambda)$ durch $Q = Q_I = 2^P$, $\to = (2^P)^2$ und $\lambda = \mathrm{id}$.

Problem: Dies ist zu groß: Das Transitionssystem T ist möglicherweise exponentiell in der Länge der Formel!

Lösung: Wir konstruieren einen kleineres Transitionssystem T' mit $(2n+1)$ Zuständen – eine Art „Leiter“, dargestellt in Abbildung 4.

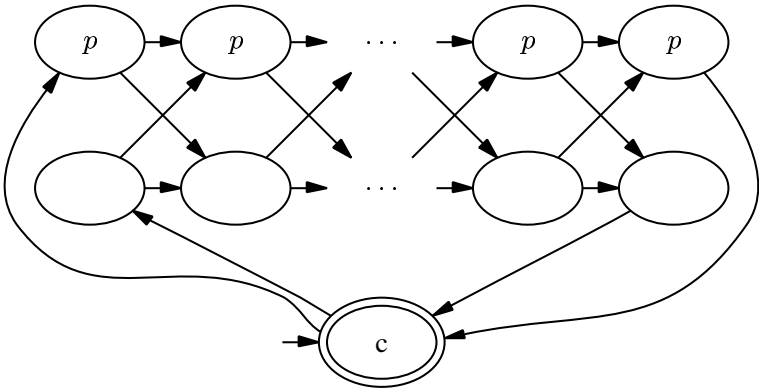


Abbildung 4: Transitionssystem zur Reduktion von LTL-Erfüllbarkeit auf das komplementäre Model-Checking-Problem.

Dabei kodiert ein Pfad von links nach rechts genau eine Belegung: Bei einem Lauf durch einen oberen Knoten in der Spalte i gilt p_i, bei einem unteren Knoten gilt p_i nicht (wobei dies immer durch die gleiche neue Variable p kodiert wird; d.h. p nimmt während eines Laufs von links nach rechts jeweils die Rolle eines p_i an). Es gilt formal: Zu jedem $u \in (2^P)^\omega$ gibt es genau ein $v \in L(T')$, so daß gilt:

- Für jedes i gilt: $c \in v[i]$ genau dann, wenn $i \mod (n+1) = 0$.

- Für jedes i und jedes $j < n$ gilt $p_j \in u[i]$ genau dann, wenn $p \in v[(n+1)i+1+j]$ ist.

Dann können wir die Formel φ, für die wir die Erfüllbarkeit in T prüfen würden, umsetzen in eine Formel ψ, die auf T' läuft:

- Jedes $p_i \in P$ wird durch $\mathsf{X}^{i+1}\, p$ ersetzt.
- Jedes X wird durch X^{n+1} ersetzt.
- Jede Teilformel $\xi \,\mathsf{U}\, \xi'$ wird ersetzt durch $(c \rightarrow \xi) \,\mathsf{U}\, (c \rightarrow \xi')$.

Dann ist φ genau dann erfüllbar, wenn $T' \not\models \neg\psi$ gilt. □

Folgerung: Das zum LTL-Model-Checking-Problem komplementäre Problem ist PSPACE-schwer.

Nun können wir die Tatsache benutzen, daß PSPACEgleich CoPSPACE ist (wie bei allen deterministischen Komplexitätsklassen). Damit ist auch das LTL-Model-Checking-Problem PSPACE-schwer. Zusammen mit der vorhin gezeigten Tatsache, daß LTL-Model-Checking in PSPACE liegt, ergibt sich:

Folgerung: Das LTL-Model-Checking-Problem ist PSPACE-vollständig.

2 Presburger-Arithmetik

2.1 Einführung

Die *Presburger-Arithmetik* ist die Erste-Stufe-Theorie von $(\mathbb{N}, +)$, d.h. die Menge aller prädikatenlogischen Formeln mit $+$ (und $=$), die in $\mathbb{N}$ gelten. Es gilt der folgende Satz[3]:

Satz: Die Presburger-Arithmetik ist entscheidbar.

2.2 Automatische Strukturen

Definitionen:

- Sei A ein Alphabet, bezeichne mit $A_\square$ dieses Alphabet mit einem zusätzlichen Füllzeichen $\square$. Dann sei $A^{\widehat{k}}$ das Alphabet $(A_\square)^k \setminus \{\square\}^k$.
- Die *Projektion*

$$\pi_i\colon \begin{array}{c}((\quad a_{(0,0)} \quad \ldots \quad a_{(0,k-1)} \quad),\\ (\quad \ldots \quad \ldots \quad \ldots \quad),\\ (\quad a_{(r-1,0)} \quad \ldots \quad a_{(r-1,k-1)} \quad))\end{array} \mapsto (a_{(0,i)}, \ldots, a_{(r-1,i)})$$

- Für $u_0, \ldots, u_{k-1} \in A^*$ sei die *Konvolution* $\otimes(u_0, \ldots, u_{k-1})$ das eindeutige Wort v aus $(A^{\widehat{k}})^*$ mit $\pi_i(v) \in u_i\square^*$. Alternative Schreibweise: $u_0 \otimes \ldots \otimes u_{k-1} := \otimes(u_0, \ldots, u_{k-1})$.
- Zu einer k-stelligen Relation $R \subseteq (A^*)^k$ definieren wir hiermit die *Konvolution der Relation* $\otimes R \subseteq (A^k)^*$:

$$\otimes R = \{\, u_0 \otimes \ldots \otimes u_{k-1} \mid (u_0, \ldots, u_{k-1}) \in R\}$$

Ziel der Konvolution ist, aus Tupeln von Wörtern ein Wort über einem anderen Alphabet zu machen.

Beispiel: Sei $u_0 = abba$, $u_1 = ba$ und $u_2 = babaa$. Dann „fülle“ die u_i zu gleicher Länge auf:

$$\begin{array}{lc|c|c|c|c} u_0 = & a & b & b & a & \square \\ u_1 = & b & a & \square & \square & \square \\ u_2 = & b & a & b & a & a \end{array}$$

[3] „Wer weiß, was Quantorenelimination ist? ... Na gut, es ist halt das, was es eben ist.“

Damit ergibt sich folgende Aneinanderreihung der einzelnen Spalten:

$$\otimes(u_0, u_1, u_2) = (a, b, b), (b, a, a), (b, \square, b), (a, \square, a), (\square, \square, a)$$

Definitionen: Eine *Struktur* ist eine Menge (*Universum*) zusammen mit Funktionen und Relationen. Eine *Wortstruktur* ist eine Struktur mit einer Sprache als Universum[4](d.h. $\subseteq A^*$); eine *automatische Wortstruktur* ist eine Struktur mit einem regulären Universum und für alle Relationen R ist $\otimes R$ regulär. *Relationale Strukturen* sind solche, in denen keine Funktionen vorkommen.

Schreibweise: Für eine Struktur S sei U^S das Universum von S, entsprechend für Relationen.

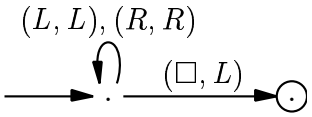
Bemerkung: Wir schränken uns zunächst auf relationale Strukturen ein.

Beispiele:

- Strukturen sind z.B. $(\mathbb{N}, +)$ oder $(\mathbb{R}, +, <)$.
- Eine automatische Wortstruktur ist der unendliche binäre Baum mit den zwei Relationen linker bzw. rechter Nachfolger, kodiert z.B. als

  $$(A^*, \{(u, u\mathsf{L}) \mid u \in A^*\}, \ \{(u, u\mathsf{R}) \mid u \in A^*\}) \ \text{ mit } \ A = \{\mathsf{L}, \mathsf{R}\}$$

  Daß die Konvolution von $R = \{(u, u\mathsf{L}) \mid u \in A^*\}$ regulär ist, zeigt der folgende Automat:



2.2.1 Automatische, relationale Struktur für die Presburger-Arithmetik

Ziel: Wir definieren eine automatische Struktur, die $(\mathbb{N}, +)$ repräsentiert.

Sei $\Sigma_{\text{add}} = \{\text{add}\}$ die Signatur, die das dreistellige Relationssymbol add enthält. Definiere das Alphabet $A = \{0, 1\}$ und die Sprache Z, die die umgekehrte Binärdarstellung natürlicher Zahlen darstellen soll – formal: $Z = [\![A^*1 + \varepsilon]\!]$ mit der Bijektion ν_0:

$$\nu_0 \colon Z \to \mathbb{N} \ \text{ mit } \ \nu_0(u) = \sum_{i<|n|} u[i] \cdot 2^i$$

[4] „Das Universum ist eine Sprache!“

Nun sei S_{add} eine Σ_{add}-Wortstruktur:

$$S_{\text{add}} = (Z, \underbrace{\{(u,v,w) \mid \nu_0(u) + \nu_0(v) = \nu_0(w)\}}_{\text{add}^{S_{\text{add}}}})$$

Satz: S_{add} ist automatisch.

Beweis: Daß Z regulär ist, ist durch die Definition offensichtlich. Zu zeigen: $\otimes\text{add}^{S_{\text{add}}}$ ist regulär, wir konstruieren dazu einen Automaten. Als Beispiel zunächst eine Addition solcher zweier Zahlen (in umgekehrter Binärdarstellung):

$$\begin{array}{l} 0\,0\,1\,1\,0\,1\,0\,1\square\square\square\square \\ 0\,1\,0\,1\,0\,1\,1\,0\,1\;1\,0\,1 \\ 0\,1\,1\,0\,1\,0\,0\,0\,0\;0\,1\,1 \end{array}$$

Den Automaten, der $\otimes\text{add}^{S_{\text{add}}}$ erkennt, definieren wir als

$$D = (A^{\widehat{3}},\ \ (\{0,1\} \times 2^{\{0,1\}}) \cup \{e\},\ \ (0,\emptyset),\ \ \delta,\ \ \{0\} \times 2^{\{0,1\}})$$

Dabei speichern wir in den Zuständen (c, M) in der ersten Komponente den Übertrag, in der zweiten Komponente merken wir uns, wo schon Füllzeichen $\square$ gelesen wurden. Zustand e ist einen Fehlerzustand, von dem aus wir nicht mehr akzeptieren. Die Transitionsfunktion δ ist dann formal gegeben durch folgende Definition (wobei $\square$ in Additionen den Wert 0 hat):

$$\begin{array}{rcll} \delta((c,M),(a_0,a_1,b)) = & (c',M') & \text{mit} & c' = \begin{cases} 0 & \text{falls} \quad a_0+a_1+c<2 \\ 1 & \text{falls} \quad a_0+a_1+c\geq 2 \end{cases} \\ & & \text{und} & M' = \{i \mid a_i = \square\} \\ & & \text{falls} & b = (a_0+a_1+c) \bmod 2 \\ & & \text{und} & \forall\, i \in M\colon a_i = \square \\ & & \text{und} & |M| \leq 1 \\ = & e & \text{sonst} & \end{array}$$

□

2.2.2 Theorie automatischer Strukturen

Satz: Die Theorie $\text{Th}(S)$ ist entscheidbar für ein automatisches S.

Schreibweisen: Wir kürzen eine Folge $x_0, \ldots, x_n$ durch $\bar{x}$ ab[5]. Ist φ eine Formel und $\bar{x}$ ein Tupel, das alle freien Variablen von φ umfaßt, dann schreiben wir $\varphi = \varphi(\bar{x})$.

[5] an der Tafel: $\underline{x}$ statt $\bar{x}$

Beispiel: Für $\varphi = u < v$ sind u.a. (u, v), (v, u) und (u, w, v) mögliche $\bar{x}$ mit $\varphi = \varphi(\bar{x})$ ($\bar{x}$ ist also nicht eindeutig!).

Definition: Für $\varphi = \varphi(\bar{x})$ und eine beliebige Belegung β definieren wir folgende $|\bar{x}|$-stellige Relation.

$$R^S_{\varphi,\bar{x}} = \left\{ (u_0, \ldots, u_{n-1}) \in (U^S)^n \;\middle|\; U^S, \beta \begin{bmatrix} u_0 & \ldots & u_{n-1} \\ x_0 & & x_{n-1} \end{bmatrix} \models \varphi \right\}$$

Lemma: Sei S automatisch und $\varphi = \varphi(\bar{x})$. Dann ist $\otimes R^S_{\varphi,\bar{x}}$ regulär.

Zusatz: Man kann einen entsprechenden Automaten effektiv konstruieren.

Idee: Der Satz über natürliche Zahlen „Zu jedem x existiert ein größeres x“ ließe sich formulieren als

$$\forall\, x \exists\, y \exists\, z(\mathrm{add}(x, z, y) \wedge (\forall\, u((\forall\, u' \colon \mathrm{add}(u, u', u')) \rightarrow \neg z = u)))$$

Falls φ die Teilformel $(\exists\, z \ldots)$ ist, dann können wir die *kleiner*-Relation beschreiben mittels $\mathbb{N}, m, n \models \varphi(x_0, x_1)$, also

$$R^S_{\varphi,x_0,x_1} = \left\{ (m, n) \in \mathbb{N}^2 \;\middle|\; m < n \right\}.$$

Wir zeigen, daß die Konvolution dieser Relation regulär ist. Da aber $\varphi(x_0, x_1, x_2)$ auch möglich ist, müssen wir auch zeigen, daß die Konvolution folgender Relation regulär ist:

$$R^S_{\varphi,x_0,x_1,x_2} = \left\{ (m, n, o) \in \mathbb{N}^3 \;\middle|\; m < n \right\}.$$

Ziel ist also zu zeigen, daß $\otimes R^S_{\varphi,\bar{x}}$ regulär ist für alle möglichen φ und $\bar{x}$.

Beweis: Per Induktion über den Aufbau von φ. Für den Induktionsanfang unterscheiden wir zwei Fälle:

- Falls $\varphi = (x_j = x_{j'})$ ist, so ist

$$R^S_{\varphi,\bar{x}} = \left\{ (u_0, \ldots, u_{k-1}) \in (U^S)^k \;\middle|\; u_j = u_{j'} \right\}$$

  Sei $D = (A, Q, q_I, \delta, F)$ ein Automat für $U^S\square^*$ (das ja regulär ist). Wir konstruieren hieraus einen Automaten D' mit $L(D') = \otimes R^S_{\varphi,\bar{x}}$. Sei dazu $K = 2^{\{0,\ldots,k-1\}}$ und

$$D' = (A^{\widehat{k}}, Q^k \times K \cup \{e\}, (q_I, \ldots, q_I, \emptyset), \delta', F^k \times K)$$

Dann wählen wir folgende Transitionsrelation:

$$\begin{aligned}\delta'((q_0,\ldots,q_{k-1},M),(a_0,\ldots,a_{k-1}))\\ &= (\delta(q_0,a_0),\ldots,\delta(q_{k-1},a_{k-1}),\{i<k \mid a_i=\square\})\\ &\quad \text{falls } a_j=a_{j'} \text{ und } |M|<k \text{ und } \forall\, i\in M\colon a_i=\square\\ &= e \quad \text{sonst}\end{aligned}$$

- Betrachte den Fall $\varphi = R(x_{i_0},\ldots,x_{i_{m-1}})$ und $\bar{x} = x_0,\ldots,x_{k-1}$.

  Beispiel: Wir haben die Struktur S des Binärbaums mit der Relation „linker Nachfolger" $\mathrm{succ}_{\mathsf{L}}$. Falls wir jetzt $\varphi = \mathrm{succ}_{\mathsf{L}}(x_2,x_1)$ und $\bar{x} = x_0,x_1,x_2$ ist, so ist zunächst

  $$R^S_{\varphi,\bar{x}} = \{(u,v,w) \mid \mathrm{succ}_{\mathsf{L}}(w,v)\}$$

  Wir müssen nun einen Automaten konstruieren, der Konvolutionen aus beliebigen (!) x_0 und bestimmten x_1 und x_2 (mit Struktur $x_2 = u$, $x_1 = u\mathsf{L}$) erkennt:

beliebig
$u\cdots u$ L $\square\ldots\square$
$u\cdots u$ $\square$ $\square\ldots\square$

  Formal: Da S automatisch ist, gibt es einen DFA $((A_\square)^m, Q, q_I, \delta, F)$, der die Konvolution der Relation $\otimes R^S$ erkennt, außerdem auch die Verlängerung dieser Konvolution um $\square^m$: $(\otimes R^S)\,((\square^m)^*)$. Zudem gibt es einen Automaten $(A_\square, S, s_I, \sigma, G)$, der $U^S\square^*$ erkennt.

  Wir konstruieren nun einen DFA $(A^{\widehat{k}}, Q\times S^k, (q_I,s_I,\ldots,s_I), \delta', F\times G^k)$, der $\otimes R^S_{\varphi,\bar{x}}$ erkennt, durch

  $$\begin{aligned}\delta'((q,s_0,\ldots,s_{k-1}),(a_0,\ldots,a_{k-1}))\\ &= (\delta(q,(a_{i_0},\ldots,a_{i_{m-1}})),\sigma(s_0,a_0),\ldots,\sigma(s_{k-1},a_{k-1}))\end{aligned}$$

  Da dieser Automat $\otimes R^S_{\varphi,\bar{x}}$ erkennt, ist dies regulär.

Im Induktionsschritt über den Aufbau der Formel konstruieren wir nun neue Automaten aus den Automaten für Teilformeln, die wir aus der Induktionsvoraussetzung erhalten:

- Falls $\varphi = \psi \vee \psi'$ ist, so liefert die Induktionsvoraussetzung zwei DFAs D und D' für $\otimes R^S_{\psi,\bar{x}}$ bzw. $\otimes R^S_{\psi',\bar{x}}$. Hieraus konstruieren wir einen DFA für $\otimes R^S_{\varphi,\bar{x}}$ durch einen einfachen Produktautomaten:

  $$(A^{\widehat{k}}, Q\times Q', (q_I,q'_I), \delta^\vee, (F\times Q')\cup(Q\times F'))$$

mit

$$\delta^{\vee}((q,q'),\bar{a}) = (\delta(q,\bar{a}),\delta'(q',\bar{a}))$$

- Für $\varphi = \neg\psi$ benutzen wir wieder einen Automaten D, der $\otimes R^S_{\psi,\bar{x}}$ erkennt, und einen Automaten $E = (A_\square, S, s_I, \sigma, G)$, der $U^S\square^*$ erkennt. Wir konstruieren folgenden Automaten für $\otimes R^S_{\varphi,\bar{x}}$:

  $$(A^{\widehat{k}}, Q \times S^k, (q_I, s_I, \ldots, s_I), \delta^{\neg}, (Q \setminus F) \times G^k)$$

  Dabei ist

  $$\delta^{\neg}((q, s_0, \ldots, s_{k-1}), \bar{a}) = (\delta(q,\bar{a}), \sigma(s_0,a_0), \ldots, \sigma(s_{k-1},a_{k-1}))$$

- Falls $\varphi = \exists\, y\, \psi$ ist, so unterscheide zwei Fälle:
  - Es ist $y = x_i$ für ein i (d.h. $y \in \bar{x}$) – nun nehmen wir an, daß wir einen DFA für $\otimes R^S_{\psi,\bar{x}}$ haben und wieder E wie oben. Wir konstruieren einen NFA für $\otimes R^S_{\varphi,\bar{x}}$, dieser sei $(A^{\widehat{k}}, Q\times S, (q_I,s_I), \Delta, F\times G)$ mit

    $$\begin{aligned}\Delta \;=\; \{((q,s),\bar{a},(q',\sigma(s,a_i))) \mid \\ \exists\, a' \in A\colon \delta(q,(a_0,\ldots,a_{i-1},a',a_{i+1},\ldots,a_{k-1})) = q'\}\end{aligned}$$

    Hier „rät“ der Automat das a' für jeden konkreten Lauf (und ersetzt damit die ursprüngliche Belegung von x_i), während durch $\sigma(s,a_i)$ sichergestellt wird, daß die ursprüngliche Belegung von x_i ein Wort des Universums ist.

    Zu korrigieren wäre noch, daß dieser Automat nur ein Wort in Länge der Konvolution rät, das gesuchte Wort jedoch auch kürzer oder länger sein kann – siehe Dozenten-Skript.
  - $y \neq x_i$ für ein i (d.h. $y \notin \bar{x}$) – entsprechend. □

Nun können wir obigen Satz beweisen:

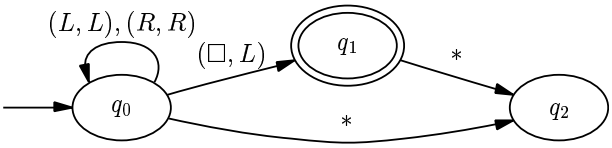
Satz: Th(S) ist entscheidbar für ein automatisches S.

Beweis: Zu einem Satz φ kann ein DFA D für $\otimes R^S_{\varphi,()}$ konstruiert werden. Dann gilt: $\varepsilon \in L(D)$ genau dann, wenn $S \models \varphi$ gilt. □

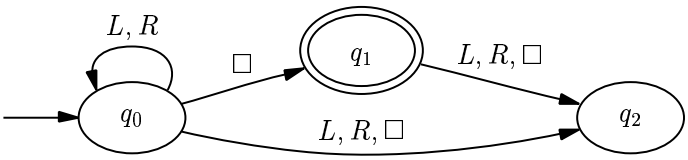
Beispiel: Betrachte wieder die Struktur B des unendlichen Binärbaums. Betrachte folgenden Satz:

$$\varphi = \forall x_0 \underbrace{\exists x_1\, \mathrm{succ}_{\mathsf{L}}(x_0,x_1)}_{\psi}$$

Benutze nun einen DFA für $\otimes R^B_{\mathrm{succ}_{\mathsf{L}(x_0,x_1)},x_0,x_1}$:



Dann entsteht ein NFA[6] für $\otimes R^B_{\psi,x_0}$:



Hinweis: Dieser Automat erkennt nicht (wie gewünscht) $\{\mathsf{L},\mathsf{R}\}^*$, sondern $\{\mathsf{L},\mathsf{R}\}^*\square$ – hier müßte die Automaten-Konstruktion noch korrigiert werden, da bisher z.T. noch Füllzeichen am Ende erforderlich sind.

2.3 Automatisch repräsentierbare Strukturen

Ziel: Gegeben sei eine Signatur Σ und eine relationale Σ-Struktur S – Frage ist, wann S als Wortstruktur aufgefaßt werden kann, damit wir Ergebnisse des letzten Abschnitts anwenden können.

Definition: Sei S eine relationale Σ-Struktur und $\Sigma' = \Sigma \cup \{R_e\}$. Das Tupel (S', ν) mit einer Σ'-Wortstruktur S' und $\nu\colon U^{S'} \to U^S$ heißt *Wortrepräsentation* von S, falls gilt:

- ν ist surjektiv
- $\nu^{-1}(R^S) = R^{S'}$ für alle $R \in \Sigma$
- $R_e^{S'} = \nu^{-1}(=^S)$ – d.h. R_e stellt die S-Gleichheit in S' bereit.

Beispiel: Sei $S = (\mathbb{N}, \mathrm{add}^S)$ und $S' = (\{0,1\}^*, \mathrm{add}^{S'}, R_e)$ mit

$$R_e = \{\,(zu, zv) \mid z \in \{0,1\}^*, u, v \in \{0\}^*\}$$

Hier liefert also R_e die logische Gleichheit in den natürlichen Zahlen, da führende (in umgekehrter Darstellung also am Ende stehende) Nullen der Dezimaldarstellung ignoriert werden.

Satz: Wenn $\mathrm{Th}(S')$ entscheidbar ist, dann ist $\mathrm{Th}(S)$ entscheidbar.

[6] Dieser ist schon um den Automaten erleichert, der das Universum erkennt.

Beweis: Zeige, daß es zu jedem Σ-Satz φ einen Σ'-Satz φ' gibt mit $\varphi \in \mathrm{Th}(S)$ genau dann, wenn $\varphi' \in \mathrm{Th}(S')$ ist. Zeige weiterhin: φ' kann effektiv konstruiert werden, das reicht. Bei der induktiven Konstruktion von φ ist $\varphi' = R_e(x, y)$ für $\varphi = (x = y)$, alles andere trivial.

Zu zeigen bleibt (als Übung), daß für alle Belegungen β' gilt:

$$S', \beta' \models \varphi' \iff S, \nu \circ \beta' \models \varphi$$

□

Definition: Sei eine Signatur $\Sigma = (\mathcal{R}, \mathcal{F}, \mathcal{C})$ gegeben mit Relationssymbolen $\mathcal{R}$, Funktionssymbolen $\mathcal{F}$ und Konstanten $\mathcal{C}$. Dann ist die relationale Variante von S eine Signatur

$$\bar{\Sigma} = (\mathcal{R} \cup \mathcal{R}_{\mathcal{F}} \cup \mathcal{R}_{\mathcal{C}}, \emptyset, \emptyset)$$

mit einem $(n+1)$-stelligen $R_f \in \mathcal{R}_{\mathcal{F}}$ für jedes n-stellige $f \in \mathcal{F}$ und einem einstelligen $R_C \in \mathcal{R}_{\mathcal{C}}$ für jedes $c \in \mathcal{C}$.

Konstruktionen zwischen Σ und $\bar{\Sigma}$:

- Zu einer Σ-Struktur S konstruieren wir eine $\bar{\Sigma}$-Struktur $\bar{S}$ durch
  - $U^{\bar{S}} = U^S$,
  - $R^{\bar{S}} = R^S$ für alle $R \in \mathcal{R}$,
  - für alle $f \in \mathcal{F}$ ist
    $$R_f^{\bar{S}} = \left\{ (a_0, \ldots, a_{n-1}, f(a_0, \ldots, a_{n-1})) \mid a_0, \ldots, a_{n-1} \in U^S \right\},$$
  - $R_c^{\bar{S}} = \{c^S\}$ für alle $c \in \mathcal{C}$.
- Zu einem Σ-Term t und $y \notin \mathrm{var}(t)$ konstruieren wir eine $\bar{\Sigma}$-Formel φ_t^y mit folgenden Eigenschaften:
  - Zu jeder Belegung $\bar{\beta}$ gibt es genau ein $b \in U^{\bar{S}}$ mit $\bar{S}, \bar{\beta}\frac{b}{y} \models \varphi_t^y$.
  - Für dieses b gilt $S, \beta\frac{b}{y} \models y = t$.

  Induktive Konstuktion dieser Formel:
  - für $t = x$ ist $\varphi_t^y = (x = y)$,
  - für $t = c$ ist $\varphi_t^y = R_c(y)$,
  - für $t = f(t_0, \ldots, t_{n-1})$ ist
    $$\varphi_t^y = \exists\, y_0 \ldots \exists\, y_{n-1} \left(\left(\bigwedge_{i<n} \varphi_{t_i}^{y_i} \right) \wedge R_f(y_0, \ldots, y_{n-1}, y) \right)$$
    mit neuen und paarweise verschiedenen Variablen $y_0, \ldots, y_{n-1}$

- Zu einer Σ-Formel φ konstruieren wir eine $\bar{\Sigma}$-Formel $\bar{\varphi}$ durch
  - Ersetzung von $t = t'$ durch $\exists\, y\, (\varphi_t^y \wedge \varphi_{t'}^y)$,
  - Ersetzung von $R(t_0, \ldots, t_{n-1})$ durch

$$\exists\, y_0 \ldots \exists\, y_{n-1} \left(\left(\bigwedge_{i<n} \varphi_{t_i}^{y_i} \right) \wedge R(y_0, \ldots, y_{n-1}) \right)$$

Beispiel: Zum Term $t = (1+x)+x'$ konstruieren wir (wobei hier die Variablen anders benannt sind als in der Konstruktion):

$$\varphi_t^y = \exists\, y_0\, \exists\, y_1\, ((\exists\, y_2\, \exists\, y_3\, (R_1(y_2) \wedge (x = y_2) \wedge R_+(y_2, y_3, y_0))) \\ \wedge (x' = y_1) \wedge R_+(y_0, y_1, y))$$

Lemma: Sei S eine Σ-Struktur. Dann gilt für alle Belegungen β:

$$S, \beta \models \varphi \iff \bar{S}, \beta \models \bar{\varphi}$$

Satz: Ist S eine Σ-Struktur und $\mathrm{Th}(\bar{S})$ entscheidbar, so auch $\mathrm{Th}(S)$.

Definition: S ist *automatisch repräsentierbar*, falls $\bar{S}$ automatisch repräsentierbar ist.

Folgerung: Jede automatisch repräsentierbare Struktur hat eine entscheidbare Theorie. Damit ist auch die Presburger-Arithmetik entscheidbar.

Bemerkung:

- Die Logik ist ohne den Verlust der Entscheidbarkeit erweiterbar z.B. um $\exists^\infty$ oder $\exists^{r \bmod n}$.
- Transitivität[7] oder Mengenquantoren zerstören die Entscheidbarkeit in vielen Fällen.

Beispiel: Sei $S = (\mathbb{N}, +1)$. Dann gibt es keine Formel φ erster Stufe *(FO-Formel)* mit $S \models \varphi(a, b)$ genau dann, wenn $a < b$. Aber es gilt $S \models \mathrm{TC}(x{+}1 = y, x, y)(a, b)$ genau dann, wenn $a \leq b$ ist – d.h. eine FO+TC-Formel kann dies ausdrücken.

[7] Zu jeder Formel $\varphi(\bar{x}, \bar{y})$ ist $\mathrm{TC}(\varphi, \bar{x}, \bar{y})(\bar{u}, \bar{v})$ eine neue Formel mit folgender Semantik: $S, \beta \models \mathrm{TC}(\varphi, \bar{x}, \bar{y})(\bar{u}, \bar{v})$ genau dann, wenn $\beta(\bar{u}) = \bar{a}_0, \ldots, \bar{a}_n = \beta(\bar{v})$ existieren mit $S, \beta \frac{a_i\, a_{i+1}}{\bar{x}\, \bar{y}} \models \varphi(\bar{x}, \bar{y})$ für alle alle $i < n$.

2.4 Komplexitätsbetrachtungen

Satz: Presburger-Arithmetik ist vollständig für doppelt exponentielle Zeit alternierender Turing-Maschinen mit einer linearen Anzahl von Alternierungen.

Satz: Presburger-Arithmetik ist in $\mathsf{TIME}(2^{2^{2^{\mathcal{O}(n)}}})$.

Komplexität unseres Algorithmus': Wenn man jedes mal minimiert, sind alle Automaten $\in 2^{2^{2^{\mathcal{O}(n)}}}$. Es gibt Formeln, für die sogar die minimalen NFAs die Größe $2^{2^{2^{\Omega(n)}}}$ haben.

Betrachte dazu Formeln und Turing-Maschinen mit Arbeitsalphabet $\{0, 1\}$ (d.h. ohne zusätzliche Zeichen, die Eingaben müssen ihr Ende „selbst signalisieren"!).

Satz: Es gibt eine Zahl $c > 0$, so daß für jede NTM T, die $\mathrm{Th}(S_{\mathrm{PrA}})$ entscheidet, gilt: Es gibt n_0, so daß für alle $n \geq n_0$ eine Formel φ existiert, die T nur in mehr als $2^{2^{c \cdot n}}$ Schritten entscheiden kann und für die $|\varphi| = n$ gilt.

Bemerkung: Eine Folgerung daraus ist, daß jedes vernünftige, vollständige Axiomensystem mit Kalkül für Presburger-Arithmetik notwendigerweise doppelt exponentiell lange Beweise für gewisse Sätze hat. Andernfalls könnte eine NTM einfach einen „kurzen" Beweis raten und überprüfen, ob dies ein Beweis ist, dies widerspräche dem Satz.

Vorüberlegungen bzw. Bemerkungen:

1. Sei $f(n) = 2^{2^n}$. Es gibt eine Zahl m_0, so daß für alle $m \geq m_0$ gilt:

$$p(2m) + f(c \cdot (2dm + 1)) \leq f(m) \tag{1}$$

   Wähle dazu $c < (2d)^{-1}$.

2. Zu jeder Turing-Maschine gibt es eine „längere" Turing-Maschine, die äquivalent ist (aber nicht länger für die Berechnungen braucht!).

Beweisidee für den Satz: Als **ersten Teil** benutzen wir eine *Diagonalisierung*, betrachte dazu folgendes Lemma:

Lemma: Sei p ein Polynom und $d > 0$ derart, daß für jede NTM T und jedes Wort u ein Satz in der Presburger-Arithmetik $\varphi_{T,u}$ mit folgenden Eigenschaften existiert:

- $\varphi_{T,u} \in \mathrm{Th}(S_{\mathrm{PrA}})$ genau dann, wenn eine akzeptierende Berechnung von T auf u mit $\leq f(|u|)$ Schritten existiert.
- $|\varphi_{T,u}| \leq d \cdot (|T| + |u|)$
- $\neg\varphi_{T,u}$ läßt sich in Zeit $\leq p(|T| + |u|)$ berechnen.

Dann gilt der obige Satz.

Beweis des Lemmas: Zu einer gegebenen Turing-Maschine T für $\mathrm{Th}(S_{\mathrm{PrA}})$ betrachten wir $T^\star$, die auf einem Eingabewort u wie folgt arbeitet:

1. $T^\star$ konstruiert $\neg\varphi_{u,u}$.
2. $T^\star$ simuliert T auf $\neg\varphi_{u,u}$
3. Falls T akzeptiert, so akzeptiert auch $T^\star$, sonst terminiert $T^\star$ nicht.

Dabei nehmen wir an, daß $T^\star$ so gewählt ist, daß $m_0 \leq |T^\star| \leq |T| + g$ gilt für ein festes g. Sei nun definiert

$$\sigma = \neg\varphi_{T^\star,T^\star}$$

Dann ist σ wahr. Beweis: Angenommen, σ wäre nicht wahr. Dann wäre $\varphi_{T^\star,T^\star}$ wahr, d.h. $T^\star$ angesetzt auf $T^\star$ würde halten, also akzeptiert das simulierte T die Formel $\neg\varphi_{T^\star,T^\star}$, Widerspruch.

Es gilt $|\sigma| \leq d \cdot (|T^\star| + |T^\star|) + 1$ laut zweitem Teil der Voraussetzung des Lemmas (und der 1 für das Negationszeichen). Sei nun $n = |\sigma|$ und $m = |T^\star|$, dann gilt:

$$n \leq 2dm + 1 \tag{2}$$

Da σ wahr ist, gibt es eine akzeptierende Berechnung von T auf σ. Sei t nun die kleinste Anzahl an Schritten, die T braucht, um σ zu akzeptieren. Dann gibt es eine akzeptierende Berechnung von $T^\star$ auf $T^\star$ mit k Schritten, wobei (nach Arbeitsweise von $T^\star$ und dem dritten Teil der Voraussetzung des Lemmas) gilt:

$$p(2m) + t > k$$

Andererseits gilt: σ ist wahr, also ist $\varphi_{T^\star,T^\star}$ falsch, das ergibt (nach erster Voraussetzung im Lemma), daß $T^\star$ auf $T^\star$ mindestens $k \geq f(|T^\star|) = f(m)$ Schritte läuft. Beides kombiniert ergibt:

$$p(2m) + t > k \geq f(m) \tag{3}$$

Es gilt also insgesamt:

$$\begin{array}{rcll} t & > & f(m) - p(2m) & (3) \\ & \geq & (f(c(2dm+1)) + p(2m)) - p(2m) & (1) \\ & = & f(c(2dm+1)) & \\ & \geq & f(cn) & (2) \end{array}$$

Wir haben gezeigt, daß für ein spezielles σ gilt, daß T mehr als $f(cn)$ Schritten benötigt. Für größere σ zeigt man dies durch geschicktes Auffüllen.

Im weiteren Verlauf der Beweisidee des obigen Satzes beschreiben wir ganze Berechnungen von Turing-Maschinen, wobei die Berechnungen 2^{2^n} Schritte haben können, und jede einzelne Konfigurationen auch die Größe 2^{2^n} haben kann. Diese Berechnungen wollen wir durch eine einzige binäre Zahl beschreiben, diese hat daher die Länge $(2^{2^n})^2$, d.h. die zu beschreibenden Zahlen sind $\leq 2^{2^{2^{n+1}}}$.

Zweiter Teil der Beweisidee des obigen Satzes wäre die Konstruktion von Formeln $\varphi_{T,u}$, die die Voraussetzungen des Lemmas erfüllen, zum Umgang mit solchen Folgen – dies wird hier weggelassen.

Im **dritten Teil** der Beweisidee müssen wir es für den zweiten Schritt noch ermöglichen, lange Berechnungen von Turing-Maschinen durch kurze Formeln zu beschreiben.

Lemma: Es gibt $d > 0$, so daß für alle n eine Formel $M_n(x, y, z)$ existiert, so daß folgendes für alle a, b, c gilt:

- $M_n(a, b, c)$ ist wahr genau dann, wenn $a < 2^{2^n}$ und $a \cdot b = c$
- $|M_n(x, y, z)| \leq d(n + 1)$
- M_n kann in Polynomzeit berechnet werden.

Beweis: Konstruktion per Induktion: Für $n = 0$ müssen nur $a < 2^{2^0} = 2$ betrachtet werden, daher:

$$M_0(x, y, z) = (x = 0 \wedge z = 0) \vee (x = 1 \wedge z = y)$$

Für $n > 0$: Jedes $x < 2^{2^{n+1}}$ läßt sich schreiben als $x = x_1x_2 + x_3 + x_4$ mit $x_i < 2^{2^n}$. Damit schreiben wir

$$z = x \cdot y = (\underbrace{x_1x_2}_{u_1} + x_3 + x_4)y = \underbrace{x_1(\overbrace{x_2y}^{u_2})}_{u_3} + \underbrace{x_3y}_{u_4} + \underbrace{x_4y}_{u_5}$$

Dann können wir konstruieren:

$$\begin{aligned} M_{n+1}(x,y,z) = \quad & \exists\, u_1 \ldots \exists\, u_5 \exists\, x_1 \ldots \exists\, x_4 \\ & M_n(x_1,x_2,u_1) \wedge M_n(x_2,y,u_2) \wedge M_n(x_1,u_2,u_3) \\ & \wedge M_n(x_3,y,u_4) \wedge M_n(x_4,y,u_5) \\ & \wedge (x = u_1 + x_3 + x_4) \wedge (z = u_3 + u_4 + u_5) \end{aligned}$$

Diese Formel ist aber zu lang, denn für lineares Wachstum darf nur einmal die Relation $M_n(x,y,z)$ verwendet werden!

Wir verwenden folgenden Trick: Betrachte eine Formel $\varphi(x_1,y_1) \wedge \ldots \wedge \varphi(x_k,y_k)$, diese wird auf *eine* Anwendung von φ reduziert durch

$$\forall\, x \forall\, y\, (((x = x_1 \wedge y = y_1) \vee \ldots \vee (x = x_k \wedge y = y_k)) \rightarrow \varphi(x,y))$$

Durch diesen Trick (und die geschickte Wiederverwendung von Variablen) kann man die gesuchte Formel *kurz* konstruieren. □

Ziel: Wir werden jetzt mit *noch größeren Zahlen* umgehen!

Lemma: Es gibt $d > 0$, so daß für alle n eine Formel $P_n(x,y,z)$ existiert, so daß folgendes für alle a, b, c gilt:

- $P_n(a,b,c)$ ist wahr genau dann, wenn $a, b, c < g(n)$ und $a \cdot b = c$, dabei werden wir $g(n)$ noch definieren mit $g(n) \geq 2^{2^{2^{n+1}}}$.
- $|P_n(x,y,z)| \leq d(n+1)$
- P_n kann in Polynomzeit berechnet werden.

Für den Beweis nutzen wir den Primzahlsatz:

Satz (Primzahlsatz): Sei $\pi(n)$ die Anzahl von Primzahlen kleiner n. Dann gilt für jedes ausreichend große n:

$$\pi(n) > \frac{n}{\log_e n}$$

Beweis des Lemmas: Für $m = 2^{2^{n+2}}$ ist die Anzahl der Primzahlen kleiner m mindestens $2^{2^{n+2}} \cdot 2^{-(n+2)} > 2^{2^{n+1}}$. Definiere $g(n)$ und wende den Chinesischen Restsatz an:

$$g(n) = \prod_{\substack{p<m \\ p \text{ Primzahl}}} p$$

Satz (Chinesischer Restsatz): k Zahlen $m_0, .., m_{k-1}$, paarweise teilerfremd, sei $M = \prod_{i<k} m_i$. Dann ist folgendes ein Isomorphismus:

$$f\colon \mathbb{Z}_m \to \mathbb{Z}_{m_0} + \ldots + \mathbb{Z}_{m_{k-1}} \quad \text{mit} \quad [n]_M \mapsto ([n]_{m_0}, \ldots, [n]_{m_{k-1}})$$

Beispiel: Betrachte $2 \cdot 3 \cdot 5 \cdot 7 = 210$. Dann gilt für alle $x, y < 210$, daß $x = y$ ist genau dann, wenn $x \equiv y (\text{mod } 2, 3, 5, 7)$.

Somit gilt in unserem Fall: Für alle $x, y < g(n)$ ist $x = y$ genau dann, wenn $x \equiv y (\text{mod } p)$ für alle $p < m$. D.h., daß für alle $x, y, z < g(n)$ ist $xy = z$ genau dann, wenn $xy \equiv z (\text{mod } p)$ für alle $p < m$, dies ist genau dann der Fall, wenn

$$((x \bmod p)(y \bmod p)) \bmod p = z \bmod p$$

2.5 Vorführung: MONA

Betrachte *Monadische Logik* mit a, b, c für boolesche Werte (`var0`, `ex0`, `all0` in MONA); x, y, z für natürliche Zahlen (`var1` etc. in MONA); und X, Y, Z für endliche Mengen von natürlichen Zahlen (kodiert z.B. durch Bitvektoren, `var2` etc. in MONA)

3 Beschreibungslogiken

3.1 Einführung

Beschreibungslogiken sind Formalismen zur Repräsentation terminologischen (begrifflichen) Wissens. Beschreibungslogische Systeme erlauben es, automatisch Schlußfolgerungen über das repräsentierte Wissen zu ziehen. Sie stammen ursprünglich aus dem Bereich der künstlichen Intelligenz, darüber hinausgehende Anwendungsgebiete sind aber u.a. folgende:

- Datenbanken (ER- und UML-Diagramme)
- Medizinische Informatik
- digitale Bibliotheken
- Semantisches Web

Beispiel: Sei eine *Familien-Wissensbasis* gegeben (eine informelle Beschreibung von Konzepten), zum Beispiel[8]:

Definitionen:

- Eine Frau ist ein Mensch und ein weibliches Wesen.
- Ein Mann ist ein Mensch und kein weibliches Wesen.
- Eine Mutter ist eine Frau, die ein Kind hat, das ein Mensch ist.
- Ein Vater ist ein Mann, der ein Kind hat, das ein Mensch ist.
- Ein Elternteil ist eine Mutter oder ein Vater.
- Ein Vater mit nur Töchtern ist ein Vater, dessen Kinder Frauen sind.

Wie kann man dies dem Computer so mitteilen, daß er es „versteht“? Das heißt, wir wollen, daß der Computer richtige Schlußfolgerungen aufgrund der Wissensbasis ziehen kann, z.B.

- Jede Mutter ist eine Frau.
- Das Konzept Mutter und Vater ist unerfüllbar, d.h. es gibt kein Individuum, welches sowohl Mutter als auch Vater ist.

[8] „Hier sind nur Männer im Raum... trotzdem wißt Ihr wohl alle, was eine Frau ist, obwohl wir nur Informatiker sind...“

Formal: Wir werden hier *Attributive Language with Complement ($\mathcal{ALC}$)* betrachen.

Definition: Im Folgenden sei N_C eine Menge von Konzeptnamen, wie z.B. Mensch, weiblich etc., desweiteren sei N_R eine Menge von Rollennamen, wie z.B. hat Kind; seien dabei N_C und N_R disjunkt. Die Menge der $\mathcal{ALC}$-Konzeptbeschreibungen ($\mathcal{ALC}$-KBs) ist induktiv definiert:

- $\bot$ (bottom) und $\top$ (top) sind $\mathcal{ALC}$-Konzeptbeschreibungen
- Jeder Konzeptname $A \in N_C$ ist eine $\mathcal{ALC}$-Konzeptbeschreibung
- Sind C und D $\mathcal{ALC}$-Konzeptbeschreibungen und ist $r \in N_R$, so sind folgendes auch $\mathcal{ALC}$-Konzeptbeschreibungen:
  - Konzeptnegation: $\neg C$
  - Konzeptkonjunktion: $C \sqcap D$
  - Konzeptdisjunktion: $C \sqcup D$
  - Werterestriktion: $\forall\, r.C$
  - Existenzrestriktion: $\exists\, r.C$

Definitionen:

- Ein $\mathcal{ALC}$-Axiom ist von der Form $C \sqsubseteq D$ (gesprochen C subsumiert D), wobei C und D $\mathcal{ALC}$-KBs sind.
- Eine $\mathcal{ALC}$-TBox T ist eine endliche Menge von $\mathcal{ALC}$-Axiomen.

Schreibweise: Als Abkürzung verwenden wir $C \equiv D$ für die Axiome $C \sqsubseteq D$ und $D \sqsubseteq C$.

Beispiel: Obiges Beispiel läßt sich jetzt beschreiben als T_{Familie}:

$$\begin{aligned}
\mathsf{Frau} &\equiv \mathsf{Mensch} \sqcap \mathsf{Weiblich} \\
\mathsf{Mann} &\equiv \mathsf{Mensch} \sqcap \neg\mathsf{Weiblich} \\
\mathsf{Mutter} &\equiv \mathsf{Frau} \sqcap \exists\, \mathsf{hatKind.Mensch} \\
\mathsf{Vater} &\equiv \mathsf{Mann} \sqcap \exists\, \mathsf{hatKind.Mensch} \\
\mathsf{Eltern} &\equiv \mathsf{Mutter} \sqcup \mathsf{Vater} \\
\mathsf{VnT} &\equiv \mathsf{Vater} \sqcap \forall\, \mathsf{hatKind.Frau}
\end{aligned}$$

Der Satz „Alle Individuen in der Wissensbasis sind Menschen“ ließe sich dann ausdrücken als $\top \sqsubseteq \mathsf{Mensch}$.

Die **Semantik** ist ähnlich wie in der Prädikatenlogik definiert, wobei Konzeptnamen einstelligen Relationen und Rollen zweistelligen Relationen entsprechen.

Definition: Eine Interpretation I ist ein Tupel $(\Delta^I, \cdot^I)$ bestehend aus einem Interpretationsbereich Δ^I und einer Interpretationsfunktion $\cdot^I$, die

- jedem $A \in N_C$ eine Teilmenge $A^I \subseteq \Delta^I$ und
- jedem $r \in N_R$ eine binäre Relation $r^I \subseteq \Delta^I \times \Delta^I$ zuweist.

Die Funktion $\cdot^I$ wird induktiv auf $\mathcal{ALC}$-KB erweitert durch

$$\begin{aligned}
\bot^I &= \emptyset \\
\top^I &= \Delta^I \\
(C \sqcap D)^I &= C^I \cap D^I \\
(C \sqcup D)^I &= C^I \cup D^I \\
(\neg C)^I &= \Delta^I \setminus C^I \\
(\forall r.C)^I &= \left\{ d \in \Delta^I \;\middle|\; \forall e \in \Delta^I \colon (d,e) \in r^I \rightarrow e \in C^I \right\} \\
(\exists r.C)^I &= \left\{ d \in \Delta^I \;\middle|\; \exists e \in \Delta^I \colon (d,e) \in r^I \wedge e \in C^I \right\}
\end{aligned}$$

Definition: Die Interpretation I ist ein *Modell* der $\mathcal{ALC}$-TBox T (in Zeichen $I \models T$), falls $C^I \subseteq D^I$ für alle Axiome $(C \sqsubseteq D) \in T$.

Beispiel: Sei $I = (\Delta^I, \cdot^I)$ wie folgt definiert: $\Delta^I = \{\mathsf{Tim}, \mathsf{Tom}, \mathsf{Tanja}, \mathsf{Tina}\}$. Nun sein

$$\begin{aligned}
\mathsf{Mensch}^I &= \Delta^I \\
\mathsf{Weiblich}^I &= \{\mathsf{Tanja}, \mathsf{Tina}\} \\
\mathsf{hatKind}^I &= \{(\mathsf{Tim}, \mathsf{Tanja}), (\mathsf{Tina}, \mathsf{Tom})\} \\
\mathsf{Frau}^I &= \{\mathsf{Tanja}, \mathsf{Tina}\} \\
\mathsf{Mann}^I &= \{\mathsf{Tim}, \mathsf{Tom}\} \\
\mathsf{Mutter}^I &= \{\mathsf{Tina}\} \\
\mathsf{Vater}^I &= \{\mathsf{Tim}\} \\
\mathsf{Eltern}^I &= \{\mathsf{Tina}, \mathsf{Tim}\} \\
\mathsf{VnT}^I &= \{\mathsf{Tim}\}
\end{aligned}$$

Dies läßt sich auch graphisch darstellen, für einen Auszug siehe Graphik 5. Dann gilt $I \models T_{\mathsf{Familie}}$.

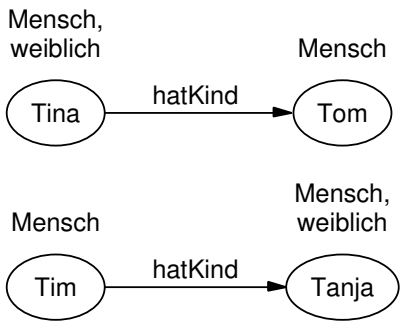


Abbildung 5: Auszug aus einer Interpretation für die Familien-Axiome

3.2 Schlußfolgerungsprobleme

Wir definieren *Erfüllbarkeit* und *Subsumtionsbeziehungen* zwischen Konzeptbeschreibungen bezüglich einer TBox.

Definitionen: C und D seien $\mathcal{ALC}$-KBs und T eine $\mathcal{ALC}$-TBox.

- Dann ist C *erfüllbar bezüglich* T, falls eine Interpretation I existiert mit $I \models T$ und $C^I \neq \emptyset$.
- D *subsumiere* C *bezüglich* T ($T \models C \sqsubseteq D$), falls für alle Interpretationen mit $I \models T$ gilt, daß $C^I \subseteq D^I$ ist.
- C und D sind *äquivalent bezüglich* T ($T \models C \equiv D$), falls $T \models C \sqsubseteq D$ und $T \models D \sqsubseteq C$ gilt.

Beispiel: Alle Konzeptnamen in T_{Familie} sind erfüllbar, siehe obige Interpretation. Einige der Subsumtionsbeziehungen sind in 6 dargestellt.

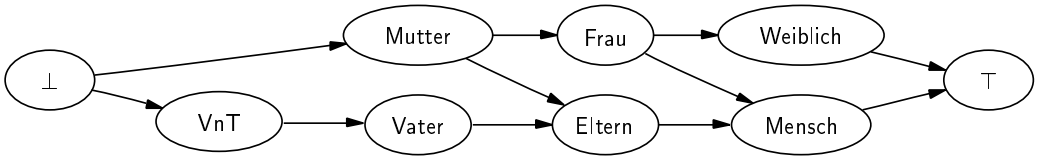


Abbildung 6: Subsumtionsbeziehungen im Axiomensystem *Familie*

Ziel: Wir möchten Verfahren entwickeln, mit denen diese Beziehungen vollautomatisch berechnet werden können. Dazu müssen wir die folgenden Ent-

scheidungsprobleme lösen:

$$\begin{aligned} \mathrm{SAT}_{\mathcal{ALC}} &= \{(C,T) \mid C \text{ erfüllbar bezüglich } T\} \\ \mathrm{SUB}_{\mathcal{ALC}} &= \{(C,D,T) \mid T \models C \sqsubseteq D\} \end{aligned}$$

Wir müssen nur eines der Probleme entscheiden, da sie ineinander überführbar sind:

$$\begin{aligned} (C,T) \in \mathrm{SAT}_{\mathcal{ALC}} &\iff (C,\bot,T) \notin \mathrm{SUB}_{\mathcal{ALC}} \\ (C,D,T) \in \mathrm{SUB}_{\mathcal{ALC}} &\iff (C \sqcap \neg D,T) \notin \mathrm{SAT}_{\mathcal{ALC}} \end{aligned}$$

Um dieses Problem zu lösen, ist es hilfreich, sich das Entscheidungsproblem für eine andere Beschreibungslogik, nämlich $\mathcal{ALC}_{\mathrm{reg}}$, anzusehen, denn $\mathrm{SAT}_{\mathcal{ALC}}$ kann auf das Erfüllbarkeitsproblem für $\mathcal{ALC}_{\mathrm{reg}}$-KB mit leerer TBox reduziert werden.

Definition: Seien N_C und N_R wie oben. *$\mathcal{ALC}_{\mathrm{reg}}$-Konzeptbeschreibungen* sind genau wie $\mathcal{ALC}$-Konzeptbeschreibungen definiert, nur daß Rollennamen r in $\forall\, r.C$ und $\exists\, r.C$ nun (komplexe) Rollenbeschreibungen R sein dürfen. *Rollenbeschreibungen (RB)* sind induktiv definiert durch:

- $r \in N_R$ ist Rollenbeschreibung
- ε (die Identität) ist eine Rollenbeschreibung
- sind R und S Rollenbeschreibungen, so auch folgende Ausdrücke:
  - Konkatenation $R \circ S$
  - reflexiv-transitive Hülle R^*
  - Rollenvereinigung $R \cup S$

Beispiel: Das Konzept Mensch, dessen sämtliche Nachfahren männlich sind:

$$\mathsf{Mensch} \sqcap \forall\, \mathsf{hatKind} \circ \mathsf{hatKind}^*.(\mathsf{Mensch} \sqcap \neg \mathsf{Weiblich})$$

Definition: Sei I eine Interpretation wie oben. Die Interpretationsfunktion $\cdot^I$ wird in offensichtlicher Weise auf $\mathcal{ALC}_{\mathrm{reg}}$-Konzeptbeschreibungen erweitert, wobei die Interpretation von Rollenbeschreibungen induktiv wie folgt

definiert ist:

$$\begin{aligned}
\varepsilon^I &= \{(d,d) \mid d \in \Delta^I\} \\
(R \circ S)^I &= R^I \circ S^I = \{(d_1,d_3) \in (\Delta^I)^2 \mid \\
&\qquad\qquad \exists\, d_2 \in \Delta^I \colon (d_1,d_2) \in R \wedge (d_2,d_3) \in S\} \\
(R^*)^I &= (R^I)^* = \bigcup_{i \geq 0} (\underbrace{R \circ \ldots \circ R}_{i})^I \\
(R \cup S)^I &= R^I \cup S^I
\end{aligned}$$

Wir werden später auch $\mathcal{ALC}_{\text{func}}$ betrachten, in der Rollennamen als partielle Funktionen interpretiert werden.

Definition: Eine $\mathcal{ALC}_{\text{reg}}$-Konzeptbeschreibung C ist erfüllbar, falls eine Interpretation I existiert mit $C^I \neq \emptyset$.

3.3 Reduktion des Erfüllbarkeitsproblems auf $\mathcal{ALC}_{\text{reg}}$

Lemma: Sei C eine $\mathcal{ALC}$-Konzeptbeschreibung und $T = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$ eine $\mathcal{ALC}$-TBox. Dann gilt: C ist erfüllbar bezüglich T genau dann, wenn C erfüllbar ist bezüglich T' mit

$$T' = \{\top \sqsubseteq \neg C_1 \sqcup D_1, \ldots, \top \sqsubseteq \neg C_n \sqcup D_n\}$$

Beweis: Offensichtlich stimmt die Menge der Modellevon T und T' überein. □

Eine Interpretation I kann als ein kanten- und knotenbeschrifteter Graph betrachtet werden. Wir sagen, daß I *Baumstruktur* besitzt, wenn der zu I gehörende Graph ein knoten- und kantenbeschrifteter Baum ist.

Beispiel: Umwandlung eines Graphen in einen Baum durch „Abwickeln“: In Graphik 7 werden die beiden gestrichelten Kanten $(3,1)$ und $(4,4)$ abgewickelt zu den gepunkteten Kanten $(3,1')$, $(4,4')$, $(4',4'')$ usw. mit neuen Knoten $1'$, $4'$, $4''$ usw., dadurch entsteht aus dem Graphen ein Baum.

Lemma: $\mathcal{ALC}$ besitzt die *Baummodelleigenschaft*, d.h. für jede bezüglich einer TBox T erfüllbare $\mathcal{ALC}$-Konzeptbeschreibung C existiert ein Modell I von T mit Baumstruktur, so daß C^I die Wurzel von I enthält und alle Kanten nur mit Rollennamen beschriftet sind, die in C oder T vorkommen.

Beweisidee: Gegeben sei ein beliebiges Modell I von T mit $d \in C^I$ für ein $d \in \Delta^I$. Nun kann I ausgehend von d „abgewickelt“ werden und $r^I = \emptyset$ gesetzt werden für r, die nicht in C oder T vorkommen.

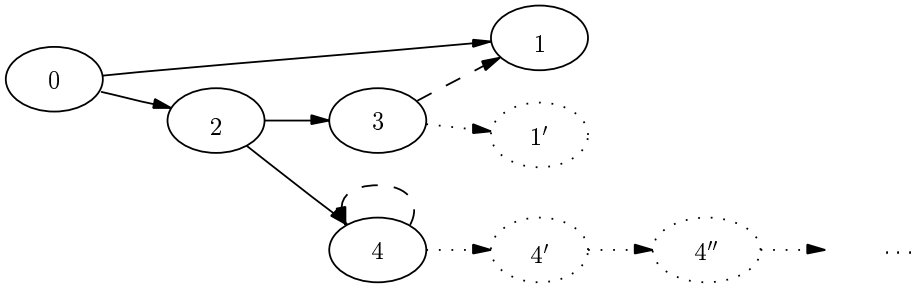


Abbildung 7: Abwicklung eines Graphen zu einem Baum

Lemma: $\mathcal{ALC}_{\text{reg}}$besitzt die Baummodelleigenschaft ebenfalls.

Seien C eine $\mathcal{ALC}$-Konzeptbeschreibung und T eine $\mathcal{ALC}$-TBox. Sei $\{r_1, \ldots, r_k\}$ die Menge der Rollennamen, die in C und T vorkommen. Zu C und T definieren wir die folgende $\mathcal{ALC}_{\text{reg}}$-Konzeptbeschreibung:

$$E_{C,T} = C \sqcap \forall\ (r_1 \cup \ldots r_k)^* .((\neg C_1 \sqcup D_1) \sqcap \ldots \sqcap (\neg C_n \sqcup D_n))$$

Lemma: C ist erfüllbar bezüglich T genau dann, wenn $E_{C,T}$ erfüllbar ist.

Beweis:

„⇒“ Ist C erfüllbar bezüglich T, dann existiert nach dem obigen Lemma ein Modell I von T' mit Baumstruktur, so daß C^I die Wurzel von I enthält. Es ist leicht einzusehen, daß auch $(E_{C,T})^I$ die Wurzel von I enthält.

„⇐“ Analog – hier nutzt man wesentlich aus, daß alle Elemente des Interpretaitonsbereichs von der Wurzel aus erreichbar sind. □

Bemerkung: Statt des Baummodells reichen Modelle aus, die von dem betrachteten Element ausgehend zusammenhängend sind.

Satz: Es gibt eine lineare Reduktion von $\text{SAT}_{\mathcal{ALC}}$ nach $\text{SAT}_{\mathcal{ALC}_{\text{reg}}}$.

3.4 Automaten auf endlichen Bäumen

Ziel: Wir werden nun im Folgenden das Erfüllbarkeitsproblem für $\mathcal{ALC}_{\text{reg}}$ reduzieren auf den Leerheitstest in einem Büchi-Baumautomaten, dies kann wiederum reduziert werden auf die Leerheit eines Automaten über endlichen Bäumen. Sei dazu w eine Menge von natürlichen Zahlen, w^+ die Menge aller Wörter über w mit ε als leerem Wort.

Definition: Ein *Alphabet mit Stelligkeitsfunktion* ist eine Menge Σ mit einer Funktion $\nu : \Sigma \to w$. Es bezeichne $\Sigma_n = \{a \in \Sigma \mid \nu(a) = n\}$, d.h. die Menge aller *n-stelligen Symbole* aus Σ. Σ_0 enthält die *Konstanten*.

Bemerkung: Wir werden oft ν nicht explizit angeben.

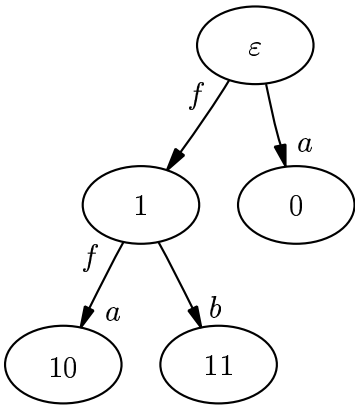
Definition: Sei Σ ein Alphabet mit Stelligkeitsfunktion ν. Ein Σ-*Baum* ist eine partielle Funktion $t\colon w^* \to \Sigma$, deren Definitionsbereich $\mathrm{dom}(t)$ folgende Bedingungen erfüllt:

1. $\varepsilon \in \mathrm{dom}(t)$
2. Für alle $v \in w^*$ und $i \in w$ gilt $v \cdot i \in \mathrm{dom}(t)$ genau dann, wenn $v \in \mathrm{dom}(t)$ und $i < \nu(t(v))$.

Die Elemente in $\mathrm{dom}(t)$ sind die *Knoten* von t. Der Baum t ist *endlich*, falls $\mathrm{dom}(t)$ endlich ist. Die Menge der endlichen Σ-Bäume bezeichnen wir mit T_Σ.

Bemerkung: Man kann Σ-Bäume auch als Σ-Terme definieren.

Beispiel: Folgender endlicher Baum über $\{0,1\}^*$ kann auch als Term $f(f(a,b),a)$ aufgefaßt werden:



Definition: Ein *Blatt-Wurzel-Automat (BW) A* ist ein Tupel (Q, Σ, Δ, F) bestehend aus

- einer endlichen Zustandsmenge Q,
- einem Alphabet Σ mit Stelligkeitsfunktion,
- einer Menge Δ von Transitionen der Form

$$(q, f, q_0, \ldots, q_{n-1}) \in Q \times \Sigma_n \times Q^n \quad \text{für} \quad n \geq 0$$

Man schreibt auch $f(q_0, \ldots, q_{n-1}) \to q$.

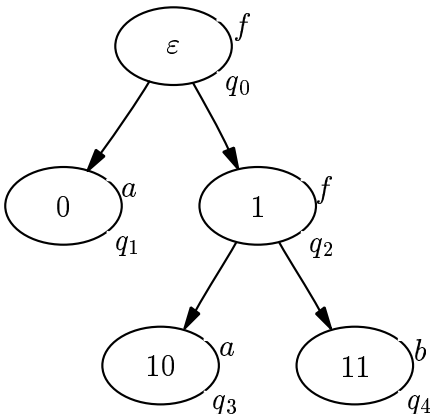
- einer Endzustandsmenge $F \subseteq Q$

Definition: Ein *Lauf* von A ist eine Abbildung $l\colon \mathrm{dom}(t) \to Q$, die jedem Knoten von t einen Zustand zuordnet, so daß gilt:

$$\{(l(v), t(v), l(v0), \ldots, l(v(n-1))) \mid n \geq 0, v \in \mathrm{dom}(t)\colon t(v) \in \Sigma_n\} \subseteq \Delta$$

Der Lauf heißt *erfolgreich*, falls $l(\varepsilon) \in F$ ist.

Beispiel: Im folgenden Baum bzw. Lauf muß für den Automaten gelten: $(q_2, f, q_3, q_4) \in \Delta$



Definition: Der Automat A *akzeptiert* einen Σ-Baum t, falls ein erfolgreicher Lauf von A auf t existiert. Weiter sei die von A erkannte Baumsprache

$$L(A) = \{t \in T_\Sigma \mid A \text{ akzeptiert } t\}$$

Definition: Ein Blatt-Wurzel-Automat A heißt *deterministisch (vollständig)*, falls für alle $n \geq 0$, $f \in \Sigma_n$ und $(q_0, \ldots, q_{n-1}) \in Q^n$ gilt, daß höchstens (mindestens) ein $q \in Q$ existiert, so daß $(q, f, q_0, \ldots, q_{n-1}) \in \Delta$ ist.

Beispiel: Sei Σ ein Alphabet mit $\Sigma_2 = \{f\}$ und $\Sigma_0 = \{a, b\}$. Sei

$$L = \{t \in T_\Sigma \mid t \text{ enthält genau ein } a\}$$

Der folgende Automat erkennt L:

$$\begin{aligned} Q &= \{q_{\text{ja}}, q_{\text{nein}}\} \\ F &= \{q_{\text{ja}}\} \\ \Delta &= \{(q_{\text{ja}}, a), (q_{\text{nein}}, b), (q_{\text{ja}}, f, q_{\text{ja}}, q_{\text{nein}}), \\ &\quad (q_{\text{ja}}, f, q_{\text{nein}}, q_{\text{ja}}), (q_{\text{nein}}, f, q_{\text{nein}}, q_{\text{nein}})\} \end{aligned}$$

Satz: Eine Baumsprache $L \subseteq T_\Sigma$ wird genau dann von einem Blatt-Wurzel-Automaten erkannt, wenn sie von einem vollständigen und deterministischen Blatt-Wurzel-Automaten erkannt wird.

Beweis: über Potenzmengenkonstruktion. □

Definition: Ein *Wurzel-Blatt-Automat (WB)* A ist ein Tupel (Q, Σ, I, Δ) mit Q, Σ und Δ wie oben und einer Menge von Anfangszuständen $I \subseteq Q$.

Definition: Ein *Lauf* ist eine Abbildung $l\colon \text{dom}(t) \to Q$, so daß $l(\varepsilon) \in I$ ist und gilt:

$$\{(l(v), t(v), l(v0), \ldots, l(v(n-1))) \mid n \geq 1, v \in \text{dom}(t)\colon t(v) \in \Sigma_n\} \subseteq \Delta$$

Der Lauf heißt *erfolgreich*, falls $(l(v), t(v)) \in \Delta$ für alle Blätter v von t ist.

Definition: Ein Wurzel-Blatt-Automat A heißt *deterministisch (vollständig)*, falls für alle $n \geq 1$, $f \in \Sigma_n$ und $q \in Q$ gilt, daß höchstens (mindestens) ein Tupel $(q_0, \ldots, q_{n-1}) \in Q^n$ existiert, so daß $(q, f, q_0, \ldots, q_{n-1}) \in \Delta$ ist; für deterministisch muß zusätzlich $|I| = 1$ gelten.

Beispiel: Sei L die Sprache von oben (genau ein a), und $A = (Q, \Sigma, \Delta, F)$ der Blatt-Wurzel-Automat von oben. Dann ist $A' = (Q, \Sigma, I, \Delta)$ mit $I = F$ ein passender Wurzel-Blatt-Automat mit $L(A') = L$.

Satz: Eine Baumsprache $L \subseteq T_\Sigma$ wird genau dann von einem Blatt-Wurzel-Automaten erkannt, wenn sie von einem Wurzel-Blatt-Automaten erkannt wird.

Beweis: in der Übung. □

Bemerkung: Der im Beispiel aus dem Blatt-Wurzel-Automaten konstruierte Wurzel-Blatt-Automat ist nicht deterministisch.

Satz: Zu L aus dem obigen Beispiel gibt es keinen deterministischen Wurzel-Blatt-Automaten.

Beweis: Annahme: Sei $A = (Q, \Sigma, I, \Delta)$ ein deterministischer Wurzel-Blatt-Automat mit $L(A) = L$. Dann akzeptiert A den Baum $f(a, b)$. Also existiert

eine Transition $(q, f, q_1, q_2) \in \Delta$ mit $I = \{q\}$, $(q_1, a) \in \Delta$ und $(q_2, b) \in \Delta$. Da A auch $f(b, a)$ akzeptiert und a deterministisch ist, gilt weiter: $(q_1, b) \in \Delta$ und $(q_2, a) \in \Delta$. Daraus folgt: A akzeptiert auch $f(a, a)$, ein Widerspruch! □

Definition: Eine Baumsprache $L \subseteq T_\Sigma$ heißt *regulär*, wenn sie von einem Blatt-Wurzel- bzw. Wurzel-Blatt-Automaten erkannt wird.

Satz: Das Alphabet Σ enthalte das zweistellige Symbol f und mindestens eine Konstante. Dann ist die folgende Baumsprache nicht regulär:

$$L = \{ f(t, t) \mid t \text{ ist } \Sigma\text{-Baum} \}$$

Beweis: Wir nehmen an, daß L regulär ist. Dann existiert ein deterministischer und vollständiger Blatt-Wurzel-Automat $A = (Q, \Sigma, \Delta, F)$ mit $L = L(A)$. Zu jedem t existiert dann genau ein Lauf l_t von A auf $f(t, t)$. Desweiteren gilt, daß $l_t(0) = l_t(1)$ ist und daß $(l_t(\varepsilon), f, l_t(0), l_t(1)) \in \Delta$ ist. Sei $q_t = l_t(0)$ und $l_t(\varepsilon) \in F$. Da Q endlich und T_Σ unendlich ist, gibt es t und t' mit $t \neq t'$ und $q_t = q_{t'}$. Dann ist klar, daß man einen erfolgreichen Lauf von A auf $f(t, t')$ konstruieren kann. Also wird $f(t, t') \notin L$ von A akzeptiert, dies ist ein Widerspruch! □

Satz: Die Klasse der regulären Baumsprachen ist unter Vereinigung, Schnitt und Komplement abgeschlossen.

3.4.1 Leerheitstest für Automaten auf endlichen Bäumen

Definition: Das Leerheitsproblem für Blatt-Wurzel-Automaten ist wie folgt definiert, für Wurzel-Blatt-Automaten analog:

$$\text{EMPTY}_{\text{BW}} = \{ A \mid A \text{ ist ein BW-Automat mit } L(A) = \emptyset \}$$

Aus dem Beweis eines Satzes aus der Übung folgt, daß Wurzel-Blatt-Automaten in linearer Zeit in Blatt-Wurzel-Automaten überführt werden können. Es reicht also, EMPTY_{BW} zu entscheiden. Dazu definieren wir reduzierte Blatt-Wurzel-Automaten.

Definition: Ein Blatt-Wurzel-Automat $A = (Q, \Sigma, \Delta, F)$ heißt *reduziert*, falls für alle $q \in Q$ ein t und ein (nicht notwendigerweise erfolgreicher) Lauf l von A auf t existiert, so daß $l(\varepsilon) = q$ ist.

Lemma: Für reduzierte Blatt-Wurzel-Automaten $A = (Q, \Sigma, \Delta, F)$ gilt:

$$L(A) = \emptyset \Longleftrightarrow F = \emptyset$$

Beweis:

„⇒“ Falls $F \neq \emptyset$ ist, so existiert $q \in F$, d.h. es existiert t und ein Lauf von l auf t, so daß $l(\varepsilon) = q$ gilt, damit ist $t \in L(A)$. □

Bemerkung: Um EMPTY_{BW} zu entscheiden, reicht es also, einen zu A äquivalenten reduzierten Blatt-Wurzel-Automaten B zu konstruieren (mit $L(A) = L(B)$). Ein solcher Automat kann durch folgenden polynomiellen Algorithmus konstruiert werden:

1. Setze $Q' = \emptyset$.
2. Falls es ein $(q, f, q_0, \ldots, q_{n-1}) \in \Delta$ gibt mit $q_0, \ldots, q_{n-1} \in Q'$ und $q \notin Q'$, dann setze $Q' = Q' \cup \{q\}$ und fahre fort mit Schritt 2, ansonsten fahre fort mit Schritt 3.
3. Gib $B = (Q', \Sigma, \Delta', F \cap Q')$ aus mit Δ' alle Transitionen aus Δ enthält, die nur Zustände aus Q' verwenden.

Lemma: Es existiert ein polynomieller Algorithmus, der zu einem gegebenen Blatt-Wurzel-Automaten in polynomieller Zeit in Größe des Automaten einen äquivalenten reduzierten Blatt-Wurzel-Automaten konstruiert.

Beweis: in der Übung. □

Aus den beiden Lemmata folgt sofort:

Satz: EMPTY_{BW} (EMPTY_{WB}) ist entscheidbar in polynomieller Zeit in Größe des gegebenen Automaten.

3.5 Büchi-Baumautomaten

Im folgenden sei $\Sigma = \Sigma_2$, d.h. Σ bestehe nur aus zweistelligen Symbolen. Es bezeichne T_Σ^ω die Menge aller unendlichen Σ-Bäume, d.h. aller binären Σ-beschrifteten unendlichen Bäume, auch ω-Bäume genannt. Alle hier vorgestellten Resultate lassen sich leicht auf den Fall $\Sigma = \Sigma_k$ für $k \geq 2$ erweitern.

Definitionen:

- Eine Teilmenge von T_Σ^ω heißt *ω-Baumsprache*.
- Ein (unendlicher) *Pfad* π in einem ω-Baum t ist eine Folge v von Knoten aus t mit $v[0] = \varepsilon$ und $v[i] \in v[i-1] \cdot \{0, 1\}$ für alle $i > 0$.
- Für eine Abbildung $\lambda\colon \mathrm{dom}(t) \to S$ und einen Pfad π wie oben bezeichne $\lambda(\pi)$ die unendliche Folge $\lambda(v[0])\lambda(v[1])\ldots$.

Büchi-Baumautomaten verallgemeinern Wurzel-Blatt-Automaten auf den Fall für unendliche Bäume.

Definition: Ein *Büchi-Baumautomat* über dem Alphabet Σ ist ein Tupel von der Form $(Q, \Sigma, I, \Delta, F)$, wobei Q, I und Δ wie für Wurzel-Blatt-Automaten definiert sind und $F \subseteq Q$ eine Menge von Endzuständen ist.

Definition: Ein Lauf l von A auf $t \in T_\Sigma$ ist eine Abbildung $l\colon \mathrm{dom}(t) \to Q$ mit

$$\{\, (l(v), t(v), l(v0), l(v1)) \mid v \in \mathrm{dom}(t)\} \subseteq \Delta$$

Der Lauf l heißt *erfolgreich*, falls $l(\varepsilon) \in I$ ist und für jeden Pfad in L gilt, daß ein Endzustand aus F unendlich oft in $l(\pi)$ auftritt.

Definition: Ein Büchi-Baumautomat A *akzeptiert* t, falls ein erfolgreicher Lauf auf t existiert. Eine Baumsprache $L \subseteq T_\Sigma^\omega$ heißt *Büchi-erkennbar*, falls es einen Büchi-Automaten gibt, der L erkennt.

Beispiel: Sei $\Sigma = \Sigma_2 = \{f, g\}$ und L_1 die Menge aller ω-Bäume, in denen ein Pfad existiert, der unendlich viele f enthält. Der folgende Büchi-Automat A erkennt L_1: der Automat „rät“ einen Pfad und geht jeweils in einen Endzustand $q^\star$, wenn er auf ein f trifft, und danach wieder in einen Warte-Zustand q° bis zum nächsten f. In den restlichen Pfaden bleibt A immer in einem Endzustand q.

$$\begin{aligned} A &= (\{q^\circ, q^\star, q\}, \Sigma, \{q^\circ\}, \Delta, \{q^\star, q\}) \\ \Delta &= \{(q^\circ, f, q^\star, q), (q^\circ, f, q, q^\star), (q^\circ, g, q, q^\circ), (q^\circ, g, q^\circ, q), (q, f, q, q), \\ &\quad (q, q, q, q), (q^\star, f, q^\circ, q), (q^\star, g, q^\circ, q), (q^\star, f, q, q^\circ), (q^\star, g, q, q^\circ)\} \end{aligned}$$

Sei L_2 nun das Komplement zu L_1, d.h. L_2 sei die Menge aller ω-Bäume, in denen kein Pfad existiert, der unendlich viele f enthält.

Satz: L_2 ist nicht Büchi-erkennbar. Insbesondere ist die Klasse der Büchi-erkennbaren Sprachen nicht unter Komplement abgeschlossen.

Beweis: Angenommen, es existiert ein Büchi-Baumautomat $A = (Q, \Sigma, I, \Delta, F)$, der L_2 erkennt. Zu $n \geq |F|$ definieren wir den Σ-Baum t^n wie folgt:

$$t^n(v) = \begin{cases} f & \text{falls} \quad v \in V \\ g & \text{sonst} \end{cases} \quad \text{mit } V = \{\varepsilon\} \cup \bigcup_{i \leq n} (1^+0)^i$$

Der in Graphik 8 dargestellte Baum t^n hat unendlich viele mit f beschriftete Knoten, aber jeder Pfad hat höchstens $n+1$ solche Knoten (und zwar jeweils Pfade der Form $(1^+0)^n 0^\omega$).

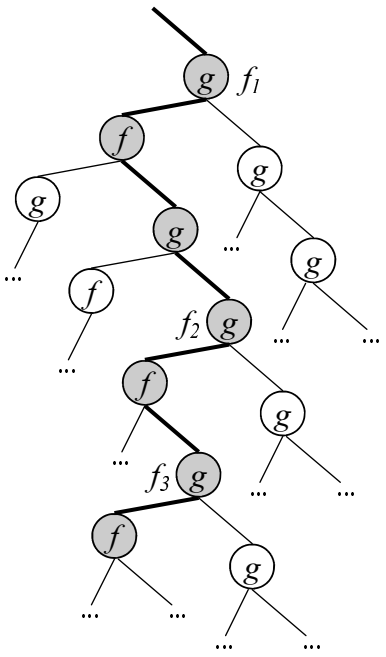


Abbildung 8: Gegenbeispiel für die Büchi-Erkennbarkeit von L_2

Also ist $t^n \in L_2$. Der Automat A erkennt also t_n. Wir definieren nun einen endlichen Pfad wie folgt: Wähle k minimal mit $l(1^k) \in F$ (ein solches k existiert immer!). Setze $f_i = l(1^k)$. Es seien $k_1, \ldots, k_i$ für $i < n$ bereits definiert. Wähle ein k_{i+1} mit

$$f_{i+1} = l(1^{k_1}01^{k_2}0 \cdots 1^{k_i}01^{k_{i+1}}) \in F$$

Da $n > |F|$, gibt es i und j mit $i < j \leq n$ und $f_i = f_j$. Beachte, daß auf dem Pfad von $u = 1^{k_1}0 \cdots 01^{k_i}$ nach $v = 1^{k_1}0 \cdots 01^{k_j}$ ein f vorkommt. Iteriertes Ersetzen des Unterbaums von t^n an der Stelle v durch den Unterbaum von t^n an der Stelle u liefert einen Baum, in dem auf einem Pfad f unendlich oft vorkommt, und der dennoch von A akzeptiert wird. Widerspruch. □

3.5.1 Leerheitstest für Büchi-Baumautomaten

Wir zeigen, daß das Leerheitsproblem für Büchi-Baumautomaten entscheidbar ist:

$$\text{EMPTY}_\omega = \{\, A \mid A \text{ ist ein Büchi-Baumautomat mit } L(A) = \emptyset\}$$

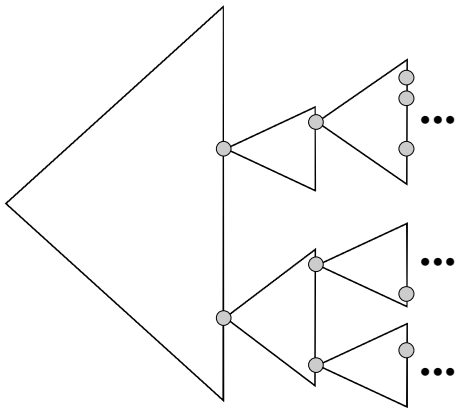
Dazu überlegen wir uns zunächst, wie man endliche Bäume konkateniert und wie durch iterierte Konkatenation (ω-Konkatenation) ω-Bäume entstehen.

Bemerkung: Wir verwenden hier z.T. die Termschreibweise für Bäume, d.h. ein Baum ist beispielsweise $t = f(x, y)$.

Definition: Es sein Γ ein Alphabet mit Stelligkeitsfunktion und $\bar{z} = (z_1, \ldots, z_k)$ sei ein k-Tupel verschiedener Elemente aus Γ_0.

1. Für $t \in T_\Gamma$ und $L_1, \ldots, L_k \subseteq T_\Gamma$ definieren wir $L = t \cdot^{\bar{z}} (L_1, \ldots, L_k)$ induktiv durch:
   (a) Falls $t \in \Gamma_0 \setminus \{z_1, \ldots, z_k\}$, dann ist $L = \{t\}$.
   (b) Falls $t = z_i$ ist für ein i, dann ist $L = L_i$.
   (c) Falls $t = f(t_1, \ldots, t_n)$ ist, dann ist
   $$L = \{\, f(t'_1, \ldots, t'_n) \mid t'_i \in t_i \cdot^{\bar{z}} (L_1, \ldots, L_k)\}\,.$$
2. Für $L_0, \ldots, L_k \subseteq T_\Gamma$ definieren wir
   $$L_0 \cdot^{\bar{z}} (L_1, \ldots, L_k) = \bigcup_{t \in L_0} t \cdot^{\bar{z}} (L_1, \ldots, L_k)$$

Beispiel: Sei $\Gamma = \Gamma_0 \cup \Gamma_2$ mit $\Gamma_2 = \{f\}$, $\Gamma_0 = \{a, b, x\}$. Sei $L = \{a, b\}$. Dann ist $f(x, x) \cdot^x L = \{f(a, a), f(a, b), f(b, a), f(b, b)\}$. Graphisch:



Bemerkung: Verschiedene Vorkommen von z_i in t müssen nicht durch den gleichen Baum ersetzt werden.

Satz: Sind $L_0, \ldots, L_k$ reguläre Baumsprachen, und ist $\bar{z}$ ein k-Tupel wie oben, dann ist auch $L_0 \cdot^{\bar{z}} (L_1, \ldots, L_k)$ eine reguläre Baumsprache.

Beweis: in der Übung. □

Definition: Es sei $\bar{z} = (z_1, \ldots, z_k)$ ein k-Tupel bestehend aus nullstelligen Symbolen, $Z = \{z_1, \ldots, z_k\}$ und $\Sigma = \Sigma_2$ wie immer. Weiter seien $U, U_1, \ldots, U_k \subseteq T_{\Sigma \cup Z}$. Die ω-Baumsprache

$$L = U \cdot^{\bar{z}} (U_1, \ldots, U_k)^{\omega, \bar{z}}$$

besteht formal aus allen ω-Bäumen $t \in T_\Sigma^\omega$, für die es eine Folge $t_0, t_1, \ldots$ von Bäumen aus $T_{\Sigma \cup Z}$ gibt mit

1. $t_0 \in U$, $t_{i+1} \in t_i \cdot^{\bar{z}} (U_1, \ldots, U_k)$ für alle $i \geq 0$
2. t ist Limes dieser Folge, d.h. für alle $v \in \{0,1\}^*$ existiert ein $i \geq 0$ mit $t(v) = t_i(v)$ und $t(v) \in \Sigma$ (insbesondere ist v kein Blatt in t_i)

Beispiel: Sei $\Sigma = \Sigma_2 = \{f, g\}$, $\bar{z} = (x, y)$, $Z = \{x, y\}$, $U = \{f(x, y)\}$, $U_1 = \{f(x, x)\}, U_2 = \{g(y, y)\}$. Dann ist $U \cdot^{\bar{z}} (U_1, U_2)^{\omega, \bar{z}}$ die Sprache $\{f(F, G)\}$, wobei F der Binärbaum bestehend nur aus f-beschrifteten Knoten ist, und G entsprechend für g-beschriftete Knoten.

Vorgehen: Um EMPTY_ω zu entscheiden, charakterisieren wir Büchi-erkennbare ω-Baumsprachen durch ω-Konkatenationen regulärer Baumsprachen.

Für Wörter $v, w \in \{0,1\}^*$ schreiben wir $v < w$ (Präfixordnung), falls $u \in \{0,1\}^+$ existiert mit $vu = w$.

Satz: Für eine ω-Baumsprache $L \subseteq T_\Sigma^\omega$ sind äquivalent:

- L wird von einem Büchi-Baumautomaten mit Endzustandsmenge $F = \{f_1, \ldots, f_k\}$ erkannt.
- Es gibt reguläre Baumsprachen $L_0, \ldots, L_k \subseteq T_{\Sigma \cup F}$, so daß mit $\bar{f} = (f_1, \ldots, f_k)$ gilt:
$$L = L_0 \cdot^{\bar{f}} (L_1, \ldots, L_k)^{\omega, \bar{f}}$$

Beweis:

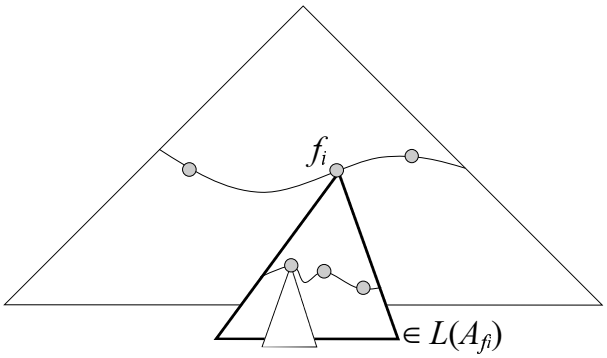
„⇐“ Konstruiere aus den WB-Automaten für $L_0, \ldots, L_k$ einen Büchi-Automaten – siehe Übung.

„⇒“ Sei $A = (Q, \Sigma, I, \Delta, F)$ ein Büchi-Baumautomat für L mit $F = \{f_1, \ldots, f_k\}$. Für $q \in Q$ definiere den WB-Automaten

$$A_q = (Q, \Sigma \cup F, \{q\}, \Delta \cup \Delta') \text{ mit } \Delta' = \{(f_1, f_1), \ldots, (f_k, f_k)\}.$$

Setze $L_0 = \bigcup_{q \in I} L(A_q)$ und $L_i = L(A_{f_i})$. Nach Definition (und wegen der Abgeschlossenheit unter Vereinigung) ist L_i für $i = 0, \ldots, k$ regulär. Mit $L' = L_0 \cdot^{\bar{f}} (L_1, \ldots, L_k)^{\omega, \bar{f}}$ bleibt $L' = L$ zu zeigen – wir zeigen dies durch doppelte Inklusion.

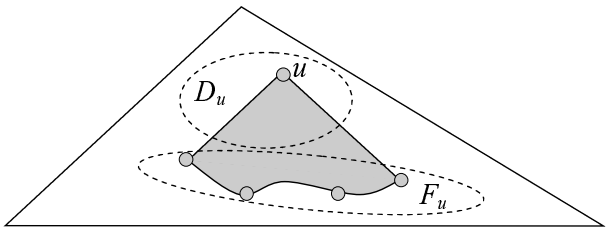
„⊆“ Sei $t \in L'$. Es ist leicht, einen erfolgreichen Lauf von A auf t zu definieren:



„⊇“ Sei $t \in L$ und l ein erfolgreicher Lauf von A auf t. Definiere

$$\begin{aligned} D_u &= \{\, w \in \{0,1\}^* \mid \forall\, v \colon \varepsilon < v \leq w \to l(uv) \notin F \} \\ F_u &= \{\, v\sigma \mid v \in D_u, v\sigma \notin D_u \text{ für } \sigma \in \{0,1\} \} \\ D_u^+ &= D_u \cup F_u \end{aligned}$$

Da l ein erfolgreicher Lauf ist, folgt, daß die Mengen D_u, F_u und D_u^+ endlich sind.



Für $u \in \{0,1\}^*$ definieren wir nun den $(\Sigma \cup F)$-Baum t'_u mit $\mathrm{dom}(t'_u) = D_u^+$ wie folgt: $t'_u(v) = t(uv)$ für alle $v \in D_u$ und $t'_u(v) = l(uv)$ für alle $v \in F_u$. Offensichtlich folgt $t'_u \in L(A_q)$ aus $l(u) = q$ für alle $u \in \{0,1\}^*$ und $q \in Q$. Nun sieht man leicht, daß $t \in L$ ist. □

Bemerkung: Die im obigen Beweis konstruierten $(\Sigma \cup F)$-Baumautomaten A_q werden wir im Leerheitstest für Büchi-Baumautomaten verwenden. Jedoch ist es günstiger, bestimmte Automaten etwas abzuwandeln: Es gilt $q \in L(A_q)$, falls $q \in F$ ist. Besser wäre es, wenn A_q nur Bäume akzeptieren würde, die mit einem Symbol aus Σ beginnen. Dazu ersetzen wir $A_q = (Q, \Sigma, \{q\}, \Delta')$ durch

$$A'_q = (Q \cup \{q_I\}, \Sigma, \{q_I\}, \Delta' \cup \Delta'') \text{ mit } \Delta'' = \{(q_I, f, q_1, q_2) \mid (q, f, q_1, q_2) \in \Delta'\}$$

Offensichtlich gilt $L(A'_q) = L(A_q) \cap \{f(t,t') \in T_{\Sigma \cup F} \mid f \in \Sigma, t, t' \in T_{\Sigma \cup F}\}$. Im Folgenden schreiben wir aber einfach A_q statt A'_q. Man überprüft leicht, daß mit dieser Modifikation der obige Satz weiterhin gilt.

Sei $A = (Q, \Sigma, I, \Delta, F)$ ein Büchi-Baumautomat. Wir eliminieren nun alle Zustände aus A, die nicht in einem erfolgreichen Lauf von A vorkommen können:

Ist $L(A_q) = \emptyset$, dann kann q nicht in einem erfolgreichen Lauf l vorkommen. Angenommen, $L(A_q)$ sei leer und es existiert ein $t \in L(A)$, ein erfolgreicher Lauf l auf t und ein $u \in \{0,1\}^*$ mit $l(u) = q$, dann folgt, daß $t'_u \in L(A_q)$, Widerspruch.

$$\begin{aligned} A_1 &= (Q_1, \Sigma, I_1, \Delta_1, F_1) \\ \text{mit } Q_1 &= \{q \in Q \mid L(A_q) = \emptyset\}, I_1 = I \cap Q_1, F_1 = F \cap Q_1 \\ \text{und } \Delta_1 &= \Delta \cap Q_1 \times \Sigma \times Q_1 \times Q_1 \end{aligned}$$

Nun ist $L(A) = L(A_1)$. **Beachte**, daß es in A_1 einen Zustand q' geben kann, so daß $L((A_1)_{q'}) = \emptyset$, aber $L(A_{q'}) \neq \emptyset$.

Durch Iteration des obigen Verfahrens erhält man aber eine Folge $A_1, A_2, \ldots$ von Büchi-Baumautomanten mit $L(A) = L(A_i)$. Da Q endlich ist, wird spätestens nach dem $|Q|$-ten Schritt kein Zustand aus dem aktuellen Automaten mehr eliminiert.

Satz: Sei $n = |Q|$. Es gilt $L(A) = L(A_n) \neq \emptyset$ genau dann, wenn $I_n \neq \emptyset$.

Beweis:

„⇐“ Sei $I_n \neq \emptyset$. Dann ist $F_n \neq \emptyset$, sonst ist $L((A_n)_q) = \emptyset$ für alle $q \in Q_n$. Sei $F_n = \{f_1, \ldots, f_k\}$. Wir wissen, daß

$$L(A_n) = \left(\bigcup_{q \in I_n} L(A_q) \right) .^{\bar{f}} \left(L((A_n)_{f_1}), \ldots, L((A_n)_{f_k}) \right)^{\omega, \bar{f}} \qquad (\star)$$

Nach Konstruktion von A_n existieren $t_0 \in L((A_n)_q)$ für $q \in I_n$, und $t_i \in L((A_n)_{f_i})$. Dann ist der folgende ω-Baum t ein Element der rechten Seite von $(\star)$, und somit $t \in L(A)$:

$$\{t\} = \{t_0\} .^{\bar{f}} \left(\{t_1\}, \ldots, \{t_k\} \right)^{\omega, \bar{f}}$$

„⇒“ Klar. □

Satz: $\text{EMPTY}_\omega \in \mathsf{P}$

Beweis: obiges Verfahren ist in P.

3.6 Erfüllbarkeit

3.6.1 Erfüllbarkeit von $\mathcal{ALC}_{\text{func}}$

Ziel ist die Reduktion von $\text{SAT}_{\mathcal{ALC}_{\text{func}}}$ auf EMPTY_ω. Zu einem gegebenen C definieren wir A_C, so daß C erfüllbar ist genau dann, wenn $L(A_C)$ nicht leer ist.
Im Folgenden werde für eine $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibung C die Menge der in C vorkommenden Konzeptnamen mit N_C bezeichnet, und $N_R = \{r_0, \ldots, r_{k-1}\}$ sei die Menge der in C vorkommenden Rollennamen. Weiter sei $\neg N_C = \{\neg A \mid A \in N_C\}$.

Lemma: $\mathcal{ALC}_{\text{func}}$ besitzt die Baummodelleigenschaft.

Für $\mathcal{ALC}_{\text{func}}$ betrachten wir eine Interpretation I mit Baumstruktur als einen ω-Baum über $\Sigma = 2^{N_C \cup \neg N_C}$ mit k-stelligen Symbolen, so daß für alle $v \in \{0, \ldots, k-1\}^*$ die folgenden Bedingungen erfüllt sind:

1. $I(v) = \emptyset$ oder ($A \in I(v)$ genau dann, wenn $\neg A \notin I(v)$ für alle $A \in N_C$)
2. Aus $I(v) = \emptyset$ folgt, daß $I(vi) = \emptyset$ für alle $i \in \{0, \ldots, k-1\}$.

Rollenbeschreibungen R betrachten wir im Folgenden als reguläre Ausdrücke über N_R. $L(R) \subseteq N_R^*$ bezeichne die zugehörige Sprache. Für eine Interpretation I und ein Wort $w = w[0] \cdots w[k-1] \in N_R^*$ sei

$$w^I = w[0]^I \circ \ldots \circ w[k-1]^I \subseteq \Delta^I \times \Delta^I$$

Für $L \subseteq N_R^*$ sei $L^I = \bigcup_{w \in L} w^I$.

Ein Quasi-NFA $B = (Q, N_R, \cdot, \Delta, F)$ über dem Alphabet N_R ist ein NFA ohne Anfangszustandsmenge. Mit $q \in Q$ sei $B_q = (Q, N_R, \{q\}, \Delta, F)$ der aus B zu q entstehende NFA. Dann sei $q^I = L(B_q)^I$.

Zu einer $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibung C sei $B = B_c = (Q, N_R, \cdot, \Delta, F)$ ein Quasi-NFA, so daß für alle Ausdrücke in C der Form $(\forall\, R.D)$ oder $(\exists\, R.D)$ ein Zustand $q \in Q$ existiert mit $L(R) = L(B_q)$.

Die Konzeptbeschreibungen, die man aus C erhält, indem man alle Ausdrücke der Form $(\forall\, R.D)$ bzw. $(\exists\, R.D)$ ersetzt durch $(\forall\, q.D)$ oder $(\exists\, q.D)$ mit $L(B_q) = L(R)$, bezeichnen wir mit C_B oder einfach C'. Wir nennen C' eine *zustandsbasierte $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibung*. Die Interpretation von C' ist nun in offensichtlicher Weise definiert. Es gilt: $C \equiv C'$.

Eine $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibung C ist in *Negations-Normalform (NNF)*, falls die Negation nur unmittelbar vor Konzeptnamen auftritt. Offensichtlich läßt sich unter Ausnutzung der folgenden Äquivalenzen jede $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibung mit linearem Aufwand in Negations-Normalform gebracht werden:

$$\neg(C \sqcup D) \equiv \neg C \sqcap \neg D \quad \text{und} \quad \neg(C \sqcap D) \equiv \neg C \sqcup \neg D$$
$$\neg(\forall\, R.C) \equiv \exists R.\neg C \quad \text{und} \quad \neg(\exists\, R.C) \equiv \forall R.\neg C$$

Analog für zustandsbasierte $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibungen.

Definition: Es sei C eine $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibung und $C' = C_B$ die zustandsabasierte Variante von C. Weiter sei Q_B die Zustandsmenge und Δ_B die Transitionsrelation von B. Der *Fischer-Ladner-Abschluß* von C ist die kleinste Menge $\mathcal{FL}(C)$ von zustandsbasierten $\mathcal{ALC}_{\text{func}}$-Konzetbeschreibungen, für die gilt:

1. $C' \in \mathcal{FL}(C)$
2. $C_1 \sqcap C_2 \in \mathcal{FL}(C) \Rightarrow C_1, C_2 \in \mathcal{FL}(C)$
3. $C_1 \sqcup C_2 \in \mathcal{FL}(C) \Rightarrow C_1, C_2 \in \mathcal{FL}(C)$
4. $N_C \cup \neg N_C \subseteq \mathcal{FL}(C)$
5. für alle $(\forall\, q.D) \in \mathcal{FL}(C)$ gilt: $D \in \mathcal{FL}(C)$ und $(\forall\, q'.D) \in \mathcal{FL}(C)$ für alle $q' \in Q_B$ mit $(q, r, q') \in \Delta_B$ für ein $r \in N_R$
6. für alle $(\exists\, q.D) \in \mathcal{FL}(C)$ gilt Entsprechendes.

Definition: Für eine $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibung C sei $\Sigma = 2^{\mathcal{FL}(C)}$ ein Alphabet mit k-stelligen Symbolen, wobei $k = |N_R|$ sei. Ein Σ-Baum t heißt *Hintikka-Baum* für C, falls für alle Knoten $v \in \text{dom}(t)$ gilt:

(1) $C' \in t(\varepsilon)$

(2) entweder $t(v) = \emptyset$, oder es gilt für alle $A \in \mathcal{FL}(C) \cap N_C$: $(A \in t(v)) \Leftrightarrow (\neg A \notin t(v))$

(3) $t(v) = \emptyset \Rightarrow t(v \cdot i) = \emptyset$ für alle $i < k$.

(4) wenn $C_1 \sqcap C_2 \in t(v)$, so auch $C_1, C_2 \in t(v)$

(5) wenn $C_1 \sqcup C_2 \in t(v)$, so auch $C_1 \in t(v)$ oder $C_2 \in t(v)$

(6) wenn $(\forall\, q.D) \in t(v)$, dann gilt:

  (a) falls $\varepsilon \in L(B_q)$, so $D \in t(v)$

  (b) entweder $t(v \cdot i) = \emptyset$ oder es gilt für alle i: $(\forall\, q'.D) \in t(v \cdot i)$ für alle q' mit $(q, r_i, q') \in \Delta_B$

(7) wenn $(\exists\, q.D) \in t(v)$, dann gibt es ein $j \geq 0$ und ein Wort $r_{i_1} \cdots r_{i_j} \in L(B_q)$, so daß $D \in t(v \cdot i_i \cdot \ldots \cdot i_j)$ und $(\exists\, q_h\, D) \in t(v \cdot i_1 \cdot \ldots \cdot i_h)$ für alle $1 \leq h < j$, wobei $(q, r_{i_1}, q_1, r_{i_2}, \ldots, r_{i_j}, q_j)$ ein erfolgreicher Pfad in B sei. Insbesondere ist q_j ein Endzustand von B_q.

Beispiel: Der Zusammenhang zwischen einen Baum mit einer (unvollständigen) Beschriftung und dem Automaten ist in Graphik 9 dargestellt.

Lemma: Eine $\mathcal{ALC}_{\text{func}}$-Konzeptbeschreibung C ist erfüllbar genau dann, wenn es zu C einen Hintikka-Baum gibt.

Beweis:

„⇒“ Ist C erfüllbar, dann existiert ein Baummodell I, so daß $\varepsilon \in C^I$ ist. Definiere einen $2^{\mathcal{FL}(C)}$-Baum t wie folgt: Setze $t(v) = \emptyset$, falls $I(v) = \emptyset$. Sonst setze

$$t(v) = \left\{ D \in \mathcal{FL}(C) \mid v \in D^I \right\}$$

Wir zeigen, daß t ein Hintikka-Baum für C ist:

(1) $\varepsilon \in C^I = C'^I$, also $C' \in t(\varepsilon)$

(2) trivial nach Definition von $t(v)$

(3) trivial nach Definition der Interpreation I

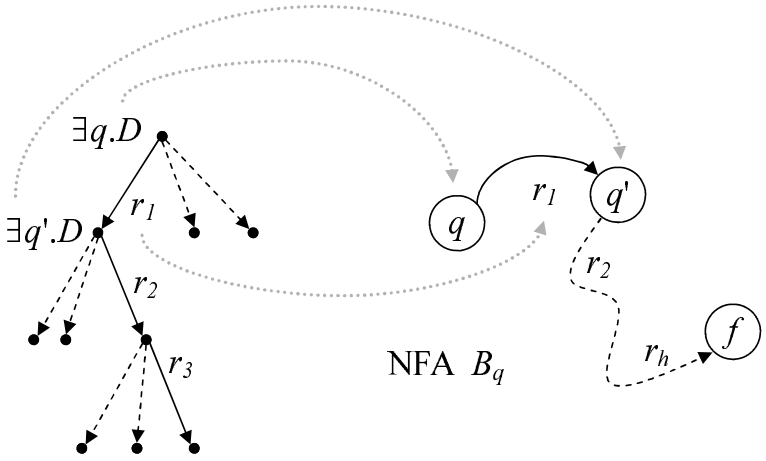


Abbildung 9: Zusammenhang zwischen Baum und Automat

(4) Wenn $C_1 \sqcap C_2 \in t(v)$ ist, ist $v \in (C_1 \sqcap C_2)^I$, also ist $v \in C_1^I$ und $v \in C_2^I$, also auch $C_1 \in t(v)$ und $C_2 \in t(v)$.

(5) Entsprechend.

(6) Sei $(\forall\, q.D) \in t(v)$. Dann ist $v \in (\forall\, q.D)^I$. Betrachte nun $L(B_q)$.

   (a) Falls $\varepsilon \in L(B_q)$, so ist $v \in D^I$, also $D \in t(v)$.

   (b) Falls $t(v \cdot i_1) \neq \emptyset$ ist, so ist insbesondere $I(v \cdot i_1) \neq \emptyset$, d.h. v besitzt in I einen r_{i_1}-Nachfolger. Wegen $v \in (\forall\, q.D)$ gilt für alle erfolgreichen Pfade $(q, r_{i_1}, q_1, r_{i_2}, q_2, \ldots, r_{i_h}, q_h)$ in B_q, daß $v \in (\forall r_{i_1} r_{i_1} \ldots r_{i_h} D)^I$ ist. Insbesondere ist $v \cdot i_1 \in (\forall r_{i_2} \ldots r_{i_h} D)^I$. Dies zeigt, daß $v \cdot i_1 \in (\forall q_1.D)^I$ für alle q_1 mit $(q, r_{i_1}, q_1) \in \Delta_B$. Also gilt $(\forall q_1.D) \in t(v \cdot i_1)$.

(7) Sei $(\exists\, q.D) \in t(v)$. Dann gilt $v \in (\exists\, q.D)^I$. Daraus folgt, daß ein erfolgreicher Pfad $(q, r_{i_1}, q_1, \ldots, r_{i_j}, q_j)$ in B_q, so daß gilt: $(v \cdot i_1 \cdots i_j) \in D^I$. Dann ist $(v \cdot i_1 \cdots i_h) \in (\exists\, q_h.D)^I$ für alle $1 \leq h \leq j$. Also ist sowohl $D \in t(v \cdot i_1 \cdots i_j)$ als auch $(\exists\, q_h.D) \in t(v \cdot i_1 \cdots i_h)$.

„⇐“ Sei t ein Hintikka-Baum für C. Setze

$$I(v) = t(v) \cap (N_C \cup \neg N_C)\ \forall v \in \{0, \ldots, k-1\}^*$$

Zeige per Induktion über den Formelaufbau: Aus $D \in t(v)$ folgt $v \in D^I$; denn mit $C' \in t(\varepsilon)$ und $C \equiv C'$ folgt dann: $\varepsilon \in C^I$.

1. Sei $D \in N_C \cup \neg N_C$. Dann gilt $v \in D^I$ nach Definition von I.

2. Sei $D = D_1 \sqcap D_2$. Sei $(D_1 \sqcap D_2) \in t(v)$. Dann sind $D_1, D_2 \in t(v)$, also nach Induktionsvoraussetzung auch $v \in D_1^I$ und $v \in D_2^I$, damit also $v \in (D_1 \sqcap D_2)^I$.
3. Sei $D = D_1 \sqcup D_2$. Sei $(D_1 \sqcup D_2) \in t(v)$. Dann sind D_1 oder $D_2 \in t(v)$, also nach Induktionsvoraussetzung auch $v \in D_1^I$ oder $v \in D_2^I$, damit also $v \in (D_1 \sqcup D_2)^I$.
4. Sei $D = (\forall\, q.D')$. Sei dann $r_{i_1} \cdots r_{i_j} \in L(B_q)$. Zu zeigen ist, daß aus $(v, v \cdot i_1 \cdots i_j) \in (r_{i_1} \cdots r_{i_j})^I$ folgt: $(v \cdot i_1 \cdots i_j) \in D'^I$. Wir zeigen dies per Induktion über j:
   - Induktionsanfang: Für $j = 0$, so ist nach Definition der Hintikka-Bäumen $D' \in t(v)$, per Induktion über den Formelaufbau folgt dann $v \in D'^I$.
   - Induktionsschritt: Wegen $(v, v \cdot i_1 \cdots i_{j+1}) \in (r_{i_1} \cdot r_{i_{j+1}})^I$ ist $t(vi_1) \neq \emptyset$. Dann gibt es einen Zustand q' mit $(q, r_{i_1}, q') \in \Delta_B$ und $(r_{i_2} \cdots r_{i_{j+1}}) \in L(B_{q'})$.

     Aus der Definition von Hintikka-Bäumen folgt wieder: $(\forall\, q'.D') \in t(v \cdot i_1)$, und nach Induktion über j folgt $(v \cdot i_1 \cdots i_{j+1}) \in D'^I$.
5. Sei $D = (\exists\, q.D')$. Nach der Definition von Hintikka-Bäumen gibt es $j \geq 0$ und ein Wort $(r_{i_1} \cdots r_{i_j}) \in L(B_q)$, so daß $D' \in t(v \cdot i_1 \cdots i_j)$ und $(\exists\, q_h.D') \in t(v \cdot i_1 \cdots i_h)$. Insbesondere ist $t(v \cdot i_1 \cdots i_h) \neq \emptyset$ für alle $1 \leq h \leq j$. Also ist $(v, v \cdot i_1 \cdots v_{i_j}) \in (r_{i_1} \cdots r_{i_j})^I$. Per Induktion über j zeigt man nun, daß $(v \cdot i_1 \cdots i_j) \in D'^I$. □

Vorgehen: Nun können wir einen Büchi-Baumautomaten $A_C = (Q, \Sigma, I, \Delta, F)$ konstruieren, der genau die Hintikka-Bäume zu C erkennt.

Betrachte die einzelnen Punkte (1) bis (7) aus der Definition der Hintikka-Bäume. Davon läßt sich (1) durch die Menge der Startzustände klären; die Punkte (2), (4) und (5) können wir lokal abhandeln (diese sprechen nur über einen Knoten im Baum), die Punkte (3) und (6) werden durch die Transitionen im Büchi-Automaten behandelt (da sie nur über Knoten und ihre Nachfolger sprechen), und für (7) benötigen wir dieBüchi-Akzeptanzbedingung.

Nun können wir die Komponenten des Automaten definieren:

- $\Sigma = 2^{\mathcal{FL}(C)}$
- $Q = Q_L \times Q_G$, wobei Q_L als diejenigen $S \in 2^{\mathcal{FL}(C)}$ sind, die die Punkte (2), (4) und (5) aus der Definition der Hintikka-Bäume erfüllen, und

$$Q_G = 2^{\{\,(\exists q.D) \mid (\exists q.D) \in \mathcal{FL}(C)\}}$$

- $I = \{ S \in Q_L \mid C' \in S \} \times \{\emptyset\}$
- $((S, S'), S, (S_0, S'_0), \ldots, (S_{k-1}, S'_{k-1})) \in \Delta$ genau dann, wenn folgende Bedingungen erfüllt sind:
  - $S, S_0, \ldots, S_{k-1}$ erfüllen (3) und (6) aus der Definition der Hintikka-Bäume.
  - Für $(\exists\, q.D) \in S$ gilt
    - entweder $\varepsilon \in L(B_q)$ und $D \in S$
    - oder es existieren i und q' mit $(q, r_i, q') \in \Delta_B$ und $(\exists\, q'.D) \in S_i$.
  - Falls $S' = \emptyset$, dann existiert für jedes $(\exists\, q.D) \in S$ mit $(\varepsilon \notin L(B_q) \vee D \notin S)$ ein i und ein q', so daß $(\exists q'.D) \in S'_i \cap S_i$ und $(q, r_i, q') \in \Delta_B$.
  - Falls $S' \neq \emptyset$, dann existiert für jedes $(\exists\, q.D) \in S'$ mit $(\varepsilon \notin L(B_q) \vee D \notin S)$ ein i und ein q', so daß $(\exists q'.D) \in S'_i \cap S_i$ und $(q, r_i, q') \in \Delta_B$.
- $F = Q_L \times \{\emptyset\}$

Lemma: A_C erkennt genau die Menge der Hintikka-Bäume zu C.

Beweis: Wir zeigen durch doppelte Inklusion, daß $L(A_C)$ gleich der Menge der Hintikka-Bäume zu C ist.

„$\supseteq$“ Sei t ein Hintikka-Baum zu L. Wir konstruieren einen Lauf von A_C auf t, indem wir jede Beschriftung $t(v)$ mit einer Beschriftung $t'(v)$ analog zur Definition der Transitionen von A_C erweitern und dann $l(v) = (t(v), t'(v))$ setzen.

Zu $(\exists q.D) \in t(v)$ existiert ein erfolgreicher Pfad $(q, r_{i_1}, q_1, \ldots, r_{i_j}, q_j)$ in B, so daß $(\exists q_h.D) \in t(v \cdot i_1 \cdots i_h)$ und $D \in t(v \cdot i_1 \cdots i_j)$.

Wähle einen eindeutigen solchen Pfad, wähle also einen Pfad minimaler Länge, und unter diesen den lexikographisch kleinsten Pfad (für eine lexikographische Ordnung auf N_R und den Zuständen von B). Wir nennen diesen Pfad den mit $(\exists\, q.D)$ in v assoziierten Pfad. Es folgt, daß der zu $(\exists\, q_h.D)$ in $vi_1 \cdots i_h$ assoziierte Pfad $(q_h, r_{i_{h+1}}, \ldots, r_{i_j}, q_j)$ ist.

Wir setzen nun $t'(\varepsilon) = \emptyset$.

- Falls $t'(v) = \emptyset$, dann setzen wir $(\exists\, q'.D) \in t'(v \cdot i)$, falls $(\exists\, q.D) \in t(v)$ ist und falls der mit $(\exists\, q.D)$ in v assoziierte Pfad von der Form $(q, r_i, q', \ldots)$ ist.

– Falls $t'(v) \neq \emptyset$, dann setzen wir $(\exists\, q'.D) \in t'(v \cdot i)$, falls $(\exists\, q.D) \in t'(v)$ ist und falls der mit $(\exists\, q.D)$ in v assoziierte Pfad von der Form $(q, r_i, q', \ldots)$ ist.

Sei nun $l(v) = (t(v), t'(v))$. Wir zeigen, daß l ein erfolgreicher Lauf von A_C auf t ist.

– Es gilt $l(\varepsilon) \in I$, da t ein Hintikka-Baum ist.

– Nach Konstruktion von t' und da t ein Hintikka-Abum ist, ist folgendes eine Transition in A_C:

$$(l(v), t(v), l(v \cdot 0), \ldots, l(v \cdot (k-1)))$$

– Die Büchi-Akzeptanzbedingung ist erfüllt: Auf jedem Pfad wird die zweite Komponente (t') unendlich oft auf $\emptyset$ gesetzt, da t ein Hintikka-Baum ist und die mit den Formeln in $t'(v)$ assoziierten Pfade in jedem Schritt kürzer werden.

„$\subseteq$“ Sei nun l ein erfolgreicher Lauf von A_C auf t. Es ist leicht zu sehen, daß die Bedingungen (1) bis (6) für Hintikka-Bäume erfüllt sind (dies sind lokale Bedingungen, die allein mit Hilfe der Zustände und Transitionen von A_C getestet werden).

Ist $(\exists\, q.D) \in t(v)$ und $l(v) = (S, \emptyset)$, dann existiert nach Konstruktion ein q_1 und i_1, so daß $(\exists\, q_1.D) \in S'_{i_1} \cap S_{i_1}$ für $l(v \cdot i_1) = (S_{i_1}, S'_{i_1})$. Da l erfolgreich ist, existiert ein Pfad $(q, r_{i_1}, q_1, \ldots, r_{i_j}, q_j)$ in B_q, so daß $(\exists\, q_h.D) \in S_h \cap S'_h$ für alle $1 \leq h < j$ mit $l(v \cdot i_1 \cdots i_h) = (S_h, S'_h)$. Außerdem ist $D \in S_j$ und $(\exists\, q_j.D) \notin S'_j = \emptyset$. Damit ist (7) ebenfalls erfüllt.

Der Fall, daß $(\exists\, q.D) \in t(v)$ und $l(v) = (S, S')$ mit $(\exists\, q.D) \in S'$ ist ähnlich. Falls $(\exists\, q.D) \in t(v)$ und $l(v) = (S, S')$ mit $(\exists\, q.D) \notin S'$, dann existiert ein Pfad $(q, r_{i_1}, q_1, \ldots, r_{i_j}, q_j)$, so daß mit $l(v \cdot i_1 \cdots i_h) = (S_h, S'_h)$.

– $S'_h \neq \emptyset$ für alle $1 \leq h \leq j$, $(\exists\, q_h.D) \in S_h$ für alle $1 \leq h < j$, $\varepsilon \in L(B_{q_j})$ und $D \in S_j$. In diesem Fall ist (7) erfüllt.

– $(\exists\, q.D) \in S_h$ für alle $1 \leq h \leq j$ (mit $\varepsilon \notin L(B_{q_j})$ oder $D \notin S_j$), und $S'_j = \emptyset$. Dann existiert ein q_{j+1} und r_{j+1}, so daß $(\exists\, q_{j+1}.D) \in S'_{j+1}$ für $l(v \cdot i_1 \cdots i_{j+1}) = (S_{j+1}, S'_{j+1})$. Da l ein erfolgreicher Lauf ist, kann nun wie oben der Pfad verlängert werden, so daß (7) erfüllt ist. □

Nun erhalten wir:

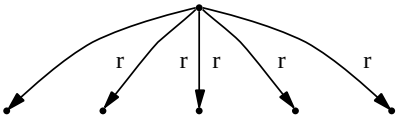
Folgerung: C ist erfüllbar genau dann, wenn $L(A_C) \neq \emptyset$.

Die Größe von A_C ist exponentiell in der Größe von X und kann in exponentieller Zeit aus C konstruiert werden. Daraus folgt:

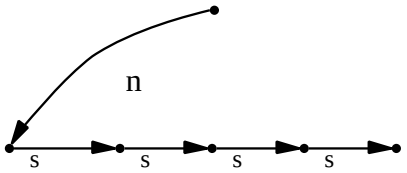
Satz: $\mathrm{SAT}_{\mathcal{ALC}_{\mathrm{func}}} \in \mathsf{EXPTIME}$.

3.6.2 Erfüllbarkeit von $\mathcal{ALC}_{\mathrm{reg}}$

Wir reduzieren $\mathrm{SAT}_{\mathcal{ALC}_{\mathrm{reg}}}$ auf $\mathrm{SAT}_{\mathcal{ALC}_{\mathrm{func}}}$. Sei C eine $\mathcal{ALC}_{\mathrm{reg}}$-Konzeptbeschreibung, in der die Rollennamen $N_R = \{r_0, \ldots, r_{k-1}\}$ vorkommen. Sei C' eine $\mathcal{ALC}_{\mathrm{func}}$-Konzeptbeschreibung, die man aus C erhält, in dem für jedes i jedes Vorkommen von r_i ersetzt wird durch $n_i \circ s_i^*$ – wobei n_i und s_i neue Rollennamen sind. Idee dahinter:



mehrere r_i-Nachfolger



ein n_i-Nachfolger,
danach s_i-Nachfolger

Lemma: C ist erfüllbar genau dann, wenn C' erfüllbar ist.

Beweis: Ist C erfüllbar, dann existiert ein Baummodell I, so daß die Wurzel ε von I zu C^I gehört. Für $v \in \Delta^I$ sei $S_i(v) = \{v_1, \ldots, v_n\} = \left\{ v' \mid (v, v') \in r_i^I \right\}$. Wir konstruieren eine neue Interpretation I': Die Konzeptnamen werden wie für I interpretiert. Die Interpretation von n_i und s_i ist wie folgt: Definiere $(v, v_1) \in n_i^{I'}$ und $(v_j, v_{j+1}) \in s_j^{I'}$ für alle $j = 1, \ldots, n-1$. Es ist leicht zu sehen, daß $\varepsilon \in C'^{I'}$.

Gegeben I' mit $v \in C'^{I'}$ für ein $v \in \Delta^{I'}$. Wir setzen $r_i^I = (n_i \circ s_i^*)^{I'}$. Es ist leicht zu sehen, daß $v \in C^I$. □

Aus diesem Lemma folgt mit obigem Satz:

Satz: $\mathrm{SAT}_{\mathcal{ALC}_{\mathrm{reg}}} \in \mathsf{EXPTIME}$.

Daraus wiederum folgt:

Satz: $\mathrm{SAT}_{\mathcal{ALC}} \in \mathsf{EXPTIME}$.

3.7 Vorführung: Protégé

Vorführung von Protégé, was eine Oberfläche u.a. für Konzeptbeschreibungen mit verschiedenen Backends wie z.B. Racer und Fact ist.

4 Kryptographische Protokolle

4.1 Einführung

Kryptographische Protokolle dienen der sicheren Kommunikation über offene Netzwerke, wie etwa dem Internet. **Konkrete Sicherheitsziele** dabei sind u.a. die *Authentifizierung* von Kommunikationspartnern (mit wem spreche ich?) und der *Schlüsselaustausch* (d.h. am Ende eines erfolgreichen Protokollaufs sollten die beiden beteiligten Kommunikationspartner einen gemeinsamen Schlüssel besitzen, den nur sie kennen und sonst keiner).

Crashkurs: Grundlagen der Kryprographie:

- *symmetrische Verschlüsselung:* Alice (A) und Bob (B) teilen sich einen gemeinsamen Schlüssel k. Alice möchte eine Nachricht x an Bob schicken, sie verschlüsselt diese symmetrisch, Notation $\mathrm{enc}^{\mathsf{s}}_k(x)$.

- *asymmetrische Verschlüsselung:* Alice und Bob haben jeweils einen privaten Schlüssel ($\hat{k}_A$ und $\hat{k}_B$) sowie einen öffentlichen Schlüssel (k_A und k_B). Die Nachricht x wird jetzt von Alice mit Bobs öffentlichem Schlüssel verschlüsselt, Notation $\mathrm{enc}^{\mathsf{a}}_{k_B}(x)$

- *Message Authentication Codes:* Alice möchte Bob eine Nachricht schicken, und diese soll unverändert bei Bob ankommen. Dazu senden Alice neben der Nachricht x auch den Wert $\mathrm{hash}_k(x)$ (für einen gemeinsamen Schlüssel k) – wobei $\mathrm{hash}_k(x)$ zur Vereinfachung der Schreibweise oft auch x schon im Klartext enthalten soll.

Problem: Die Entwicklung sicherer kryptographischer Protokolle ist äußerst fehleranfällig! Auftretende Schwierigkeiten sind u.a.:

- Der Angreifer kontrolliert in unserem Modell das Netzwerk vollständig, kann also Nachrichten verändern, unterdrücken, erzeugen, in der Reihenfolge verändern etc. – siehe Graphik 10.

- Mehrere Instanzen eines Protokolls können gleichzeitig ablaufen und sich gegenseitig beeinflussen.

Beispiel: Das Needham-Schroeder-Protokoll wurde 1976 vorgestellt, und Lowe entdeckte 1995 einen Angriff auf dieses Protokoll.

$$\begin{array}{lll} (1) & A \to B\colon & \mathrm{enc}^{\mathsf{a}}_{k_B}(A, N_A) \\ (2) & B \to A\colon & \mathrm{enc}^{\mathsf{a}}_{k_A}(N_A, N_B) \\ (3) & A \to B\colon & \mathrm{enc}^{\mathsf{a}}_{k_B}(N_B) \end{array}$$

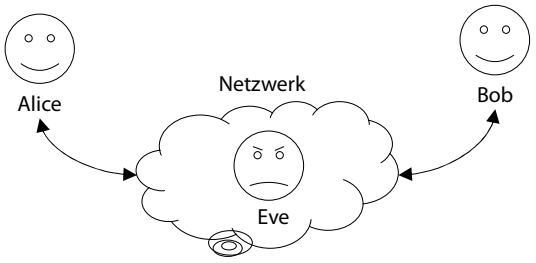


Abbildung 10: das Angreifer-Modell: der Angreifer ist das Netzwerk

Dabei sind N_A und N_B Zufallszahlen (N ist eine Abkürzung für *Nonce*, *number used once*): Man schickt einen zufälligen Bitstring und erwartet, daß dieser wieder zurückgeschickt wird. Im Needham-Schroeder-Protokoll werden zwei Nonces benutzt zur gegenseitigen Authentifizierung; gleichzeitig wird ein Nonce (beispielsweise N_B) als Sitzungsschlüssel benutzt.

Bei Lowes **Angriff** gegen dieses Protokoll glaubt Bob am Ende, mit Alice gesprochen zu haben, und ist bereit, N_B als geheimen und gemeinsamen Sitzungsschlüssel mit A zu benutzen. Dieser ist allerdings dem Angreifer bekannt:

$$
\begin{array}{lllll}
(1) & A \to I\colon & \mathrm{enc}^{\mathsf{a}}_{k_I}(A, N_A) & & \\
(1') & & & I(A) \to B\colon & \mathrm{enc}^{\mathsf{a}}_{k_B}(A, N_A) \\
(2') & & & B \to I(A)\colon & \mathrm{enc}^{\mathsf{a}}_{k_A}(N_A, N_B) \\
(2) & I \to A\colon & \mathrm{enc}^{\mathsf{a}}_{k_A}(N_A, N_B) & & \\
(3) & A \to I\colon & \mathrm{enc}^{\mathsf{a}}_{k_I}(N_B) & & \\
(3') & & & I(A) \to B\colon & \mathrm{enc}^{\mathsf{a}}_{k_B}(N_B)
\end{array}
$$

Beide Protokollläufe sind ordnungsgemäß, aber Bob kann annehmen, daß k ein Sitzungsschlüssel ist, der nur Alice und ihm bekannt ist und daß er mit Alice kommuniziert.

Eine von Lowe vorgeschlagene Modifikation (NSL-Protokoll) fügt in der zweiten Nachricht Bobs Absender ein und verhindert damit den Angriff:

$$
\begin{array}{lll}
(1) & A \to B\colon & \mathrm{enc}^{\mathsf{a}}_{k_B}(A, N_A) \\
(2) & B \to A\colon & \mathrm{enc}^{\mathsf{a}}_{k_A}(N_A, N_B, B) \\
(3) & A \to B\colon & \mathrm{enc}^{\mathsf{a}}_{k_B}(N_B)
\end{array}
$$

Weiteres **Beispiel** für ein Protokoll ist das Bull-Otway-Rekursionsprotokoll: Mehrere Parteien (hier im Beispiel drei Parteien A, B, C) wollen mittels eines Servers S kommunizieren, der ihnen jeweils paarweise symmetrische Schlüssel

(k_{AB}, k_{BC} und k_{CS}) zuteilt. Dabei seien k_X jeweils vorhandene symmetrische Schlüssel von X und dem Server S.

$$
\begin{array}{lll}
(1) & A \to B\colon & m_A := \mathrm{hash}_{k_A}(A, B, N_A, -) \\
(2) & B \to C\colon & m_B := \mathrm{hash}_{k_B}(B, C, N_B, m_A) \\
(3) & C \to S\colon & m_C := \mathrm{hash}_{k_C}(C, S, N_C, m_B) \\
(4) & S \to C\colon & m_S := \mathrm{enc}^{\mathsf{s}}_{k_C}(K_{CS}, S, N_C), \mathrm{enc}^{\mathsf{s}}_{k_C}(K_{BC}, B, N_C), \\
 & & \quad \mathrm{enc}^{\mathsf{s}}_{k_B}(K_{BC}, C, N_B), \mathrm{enc}^{\mathsf{s}}_{k_B}(K_{AB}, A, N_B), \\
 & & \quad \mathrm{enc}^{\mathsf{s}}_{k_A}(K_{AB}, B, N_A) \\
(5) & C \to B\colon & \quad \mathrm{enc}^{\mathsf{s}}_{k_B}(K_{BC}, C, N_B), \mathrm{enc}^{\mathsf{s}}_{k_B}(K_{AB}, A, N_B), \\
 & & \quad \mathrm{enc}^{\mathsf{s}}_{k_A}(K_{AC}, B, N_A) \\
(6) & B \to A\colon & \quad \mathrm{enc}^{\mathsf{s}}_{k_A}(K_{AC}, B, N_A)
\end{array}
$$

Ziel dieses Kapitels ist es nun, ein formales Modell zu definieren, in dem wir die Sicherheit von Protokollen spezifizieren und automatisch analysieren können. Dazu werden wir die einzelnen Empfang-Sende-Schritte eines Teilnehmers durch Baumtransducer beschreiben.

4.2 Baumtransducer

Sei Σ ein Alphabet mit Stelligkeitsfunktion, $V = \{x_0, x_1, \ldots\}$ eine Menge von Variablen mit $\Sigma \cap V = \emptyset$.

Definitionen:

- Seien T_Σ die Menge der Terme über Σ (Grundterme), und $T_\Sigma(V)$ die Terme über $\Sigma \cup V$, wobei Elemente von V die Stelligkeit null haben.
- Ein Term $t \in T_\Sigma(V)$ ist *linear*, falls jede Variable in t genau einmal vorkommt – beispielsweise ist $f(x, x)$ nicht linear, $f(x, g(y))$ ist linear.
- Sei $\mathrm{var}(t)$ die Menge der in t vorkommenden Variablen, und $\mathrm{sub}(t)$ die Menge der Teilterme von t.

Definition: Eine Teilmenge $\tau \subseteq T_\Sigma \times T_\Sigma$ heißt *Transduktion* über Σ. Wir schreiben $\tau(t) = \{t' \mid (t, t') \in \tau\}$; entsprechend ist τ^{-1} das Urbild von τ:

$$\tau^{-1} = \{t \in T_\Sigma \mid \exists t'\colon (t, t') \in \tau\}$$

Definitionen:

- Eine *Substitution* ist eine Funktion $\sigma\colon V \to T_\Sigma(V)$, so daß die Menge $\mathrm{dom}(\sigma) = \{x \in V \mid \sigma(x) \neq x\}$ endlich ist. Es bezeichne $\{x_0 \mapsto$

$t_0, \ldots, x_n \mapsto t_n\}$ eine Substitution mit $\sigma(x_i) = t_i$ für alle $i \leq n$.

- Eine Substitution σ heißt *Grundsubstitution*, falls $\sigma(x) \in T_\Sigma$ für alle $x \in \mathrm{dom}(\sigma)$.
- Die Anwendung einer Substitution σ auf $t \in T_\Sigma(V)$ ist induktiv definiert:
  - $\sigma(c) = c$ für $c \in \Sigma_0$
  - $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$
  - Sei $T_\Sigma^n = T_\Sigma(\{x_0, \ldots, x_{n-1}\})$. Für $t \in T_\Sigma^n$ und $t_i \in T_\Sigma(V)$ sei $t[t_0, \ldots, t_{n-1}] = \sigma(t)$, wobei $\sigma = \{x_0 \mapsto t_0, \ldots, x_{n-1} \mapsto t_{n-1}\}$ ist.

Definition: Sei $s, t \in T_\Sigma(V)$. Eine Substitution σ ist ein *Matcher* für s auf t, falls $\sigma(s) = t$ ist. In diesem Fall sagen wir, daß s mit t matcht.

Wir werden in Transducern einzelne Baumautomaten benutzen, die sich bis auf die Endzustandsmenge gleichen. Daher definieren wir:

Definition: Ein *Semi-Blatt-Wurzel-Baumautomat (Semi-BWBA)* $B = (Q, \Sigma, \Delta)$ ist ein Blatt-Wurzel-Baumautomat ohne Endzustände. Einen deterministischen und vollständigen Semi-BWBA bezeichnen wir mit DBWBA. Sei weiter $B_q = (Q, \Sigma, \Delta, \{q\})$.

Sei nun im Folgenden Q eine Menge Variablen (nullstelliger Symbole) mit $Q \cap \Sigma = \emptyset$.

Definition: Sei $t \in T_\Sigma(Q)$. Nun sein $[t]_B \subseteq Q$ die kleinste Menge, die folgende Bedingungen erfüllt:

- Falls $t \in Q$, dann ist $t \in [t]_B$.
- Falls $t = f(t_0, \ldots, t_{n-1})$ ist mit $(f(q_0, \ldots, q_{n-1}) \to q) \in \Delta$ und $q_i \in [t_i]_B$ für alle $i < n$, dann ist $q \in [t]_B$.

Sei $t \in T_\Sigma^n$ und $S_0, \ldots, S_{n-1} \subseteq Q$. Dann sei

$$[t[S_0, \ldots, S_{n-1}]]_B = \{\, q \in Q \mid \; q \in [t[q_0, \ldots, q_{n-1}]]_B \text{ mit } q_i \in S_i\}$$

Lemma: Sei B ein Semi-BWBA wie oben, $q \in Q$, $t \in T_\Sigma$. Dann gilt $q \in [t]_B$ genau dann, wenn $t \in L(B_q)$ ist.

Beweis: als Übung. □

Wir geben nun zunächst ein Beispiel für einen *Baumtransducer* an, diese werden danach dann formal definiert.

Beispiel: Sei $\Sigma = \Sigma_0 \cup \Sigma_2$ mit $\Sigma_0 = \{a, b\}$ und $\Sigma_2 = \{f, g\}$. Definiere τ so, daß τ jedes Vorkommen von f durch g und jedes Vorkommen von a durch b ersetzt wird. Dann gilt $\tau = \tau_T$ für folgenden Transducer T:

$$\begin{aligned} T &= (\{s\}, \Sigma, \{s\}, \cdot, \Gamma) \\ \text{mit } \Gamma &= \{s(f(x_0, x_1)) \to g(s(x_0), s(x_1)), \\ &\quad\ s(g(x_0, x_1)) \to g(s(x_0), s(x_1)), \\ &\quad\ s(a) \to b, s(b) \to b\} \end{aligned}$$

Definition: Ein *Baumtransducer (BT)* über Σ (mit regulärem Look-Ahead und ε-Transitionen) ist ein Tupel $T = (S, \Sigma, I, \mathcal{A}, \Gamma)$ mit

- einer endlichen Zustandsmenge S
- einer Menge $I \subseteq S$ von Anfangszuständen
- einem Semi-BWBA $\mathcal{A} = (Q, \Sigma, \Delta)$
- einer endlichen Menge Γ von Transitionen der Form

$$s(t) \to^q t'[s_0(t_0), \ldots, s_{r-1}(t_{r-1}), x_{i_0}, \ldots, x_{i_{l-1}}] \qquad (\star)$$

  mit
  - Zuständen $s, s_0, \ldots, s_{r-1} \in S$
  - einem Automaten-Zustand $q \in Q$ zum Look-Ahead,
  - einem linearen Term $t \in T_\Sigma^n$ mit Variablen $x_0, \ldots, x_{n-1}$,
  - einem Term $t' \in T_\Sigma^{r+l}$ mit Variablen $x_0, \ldots, x_{r+l-1}$,
  - Teiltermen $t_j \in \mathrm{sub}(t)$ für alle $j < r$,
  - und mit $\{i_0, \ldots, i_{l-1}\} \subseteq \{\, i \mid\ x_i \in \mathrm{var}(t)\}$.

  Wir erlauben auch Transitionen, bei denen q weggelassen wird.

Wir geben nun die Berechnungen eines Baumtransducers T an.

Definition: Sei $\vdash_T \subseteq T_{\Sigma \cup S} \times T_{\Sigma \cup S}$ die kleinste Relation, die folgende Bedingung erfüllt:

Sei $u = u'[s(v)]$ und $v = t[w_0, \ldots, w_{n-1}] \in T_\Sigma$ mit $s \in S$, $\sigma = \{x_0 \mapsto w_0, \ldots, x_{n-1} \mapsto w_{n-1}\}$ (d.h. $\sigma(t) = v$). Der Transducer T enthalte eine Transition der Form $(\star)$, und es sei $q \in [v]_{\mathcal{A}}$ (mit $v \in L(\mathcal{A}_q)$). Dann gelte:

$$u \vdash_T u'[t'[s_0(\sigma(t_0)), \ldots, s_{r-1}(\sigma(t_{r-1})), \sigma(x_{i_0}), \ldots, \sigma(x_{i_{l-1}})]]$$

Eine Berechnung des Transducers läßt sich wie in Graphik 11 darstellen.

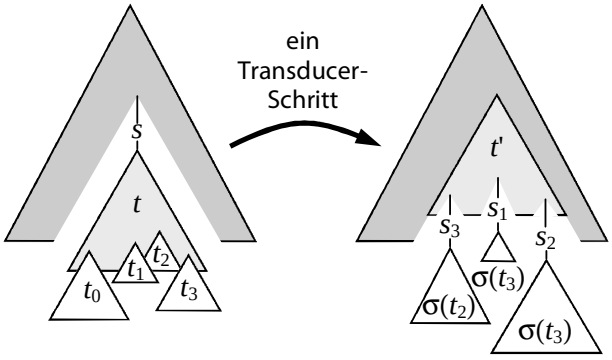


Abbildung 11: Berechnungsschritt eines Transducers

Definitionen:

- Die reflexiv-transitive Hülle von $\vdash_T$ sei $\vdash_T^*$.
- Eine *Berechnung* ist nun eine Folge der folgenden Form (mit $t_I, t' \in T_\Sigma$):
$$s(t_I) \vdash_T t_1 \vdash_T t_2 \vdash_T \ldots \vdash_T t'$$
- Die von T induzierte Transduktion τ_T ist
$$\tau_T = \{(t, t') \in T_\Sigma \times T_\Sigma \mid \exists s \in I \colon s(t) \vdash_T^* t'\}$$
- Eine Transduktion τ heißt BT-realisierbar, falls es einen BT T gibt mit $\tau = \tau_T$.
- Zwei Baumtransducer T und T' sind äquivalent, falls $\tau_T = \tau_{T'}$ ist.

Definition: Ein Braumtransducer heißt *einfach*, falls alle Transitionen von

einer der folgenden Formen sind:

$$\begin{array}{rcll} s(f(x_0,\ldots,x_{n-1})) & \to^q & t'[s_0(t_0),\ldots,s_{r-1}(t_{r-1})] & (\Sigma\text{-Transitionen}) \\ s(x) & \to^q & t'[s_0(x),\ldots,s_{r-1}(x)] & (\varepsilon\text{-Transitionen}) \end{array}$$

Satz: Jeder Baumtransducer T ist äquivalent zu einem (aus T effizient berechenbaren) einfachen Baumtransducer.

Beweis: siehe Übung. □

4.3 Iteriertes-Urbild-Wortproblem

Definition: Gegeben sei ein Alphabet Σ, $t \in T_\Sigma$, ein Blatt-Wurzel-Baumautomat I über Σ und Baumtransducer $T_0,\ldots,T_{n-1}$ (mit $\tau_i = \tau_{T_i}$ über Σ. Das *Iteriertes-Urbild-Wortproblem* ist zu entscheiden, ob $t \in \tau^{-1}(L(I))$, wobei $\tau = \tau_0 \circ \ldots \circ \tau_{n-1}$.

Satz: Das Iterierte-Urbild-Wortproblem ist entscheidbar.

Beweis: Wir zeigen dazu den folgenden Satz, aus dem direkt der obige Satz folgt, da $\tau^{-1}(L(I))$ regulär ist und daß der zugehörige Blatt-Wurzel-Baumautomat P berechnet werden kann.

Satz: Sei $I = (Q_I, \Sigma, \Delta_I, F_I)$ ein Blatt-Wurzel-Baumautomat, $T = (S_T, \Sigma, I_T, A, \Gamma_T)$ ein Baumtransducer mit einem Semi-BWBA $A = (Q_A, \Sigma, \Delta_A)$. Dann kann effektiv ein DBWBA $P = (Q_P, \Sigma, \Delta_P, F_P)$ berechnet werden, der $\tau_T^{-1}(L(I))$ erkennt.

Bemerkung: Der Satz gilt nur für das Urbild von τ_T, nicht für das Bild von τ_T, dies ist nicht regulär – siehe Übung!

Beweis: Wir können nach Satz aus dem letzten Kapitel davon ausgehen, daß T einfach ist. Wir konstruieren nun das gewünschte P und zeigen $L(P) = \tau_T^{-1}(L(I))$.

Idee der Konstruktion von P: Wenn P einen Baum t liest, dann simuliert P die Berechnung von I für alle Ausgaben t' von T auf t. Dies geschieht schrittweise für jede einzelne Transition von T.

- **Zustandsraum** von P: In einer Komponente speichert P den Zustand des simulierten Look-Ahead-Automaten. Eine zweite Komponente speichert eine Zuordnung, bei welchen $s \in S_T$ der Automat I welche Zustände Q_I erreichen kann.

Formal: Der Zustandsraum von P sei

$$Q_P = 2^{Q_A} \times 2^{S_T \times Q_I}$$

Für einen Zustand $b = (L, M) \in Q_P$ bezeichne $\mathrm{LA}(b) = L$ den Look-Ahead (erste Komponente), und $M_s(b) = \{\, q \mid (s, q) \in M \}$ bezeichne für jedes $s \in S_T$ die von I erreichbaren Zustände für den Fall $s(t)$ (zweite Komponente).

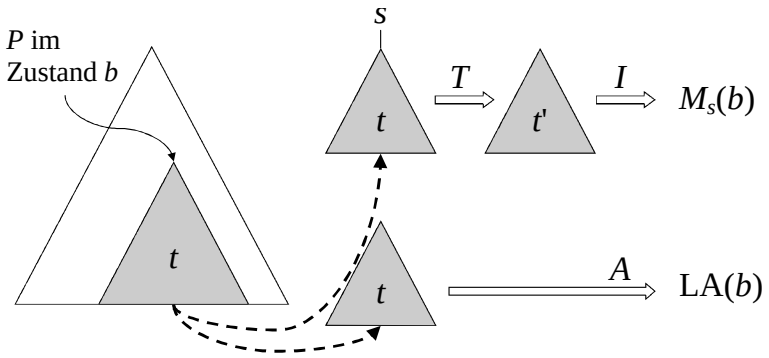


Abbildung 12: der Zustandsraum von P

Ist P nach dem Lesen von t im Zustand b; dann enthält $\mathrm{LA}(b)$ alle Look-Ahead-Zustände, die der Semi-BWBA A nach dem Lesen von t erreichen kann. Die Menge $M_s(b)$ enthält alle Zustände aus Q_I, die bei einem Lauf von I auf einem Term t', den T von $s(t)$ aus erzeugen kann, erreichbar sind.

- **Transitionssystem** von P: Für $f \in \Sigma_n$ und $b_0, \ldots, b_{n-1} \in Q_P$ enthält P die Transition $f(b_0, \ldots, b_{n-1}) \to \hat{b}$, wobei $\hat{b}$ der ε-Abschluß (siehe unten) des Zustandes $b \in Q_P$ ist, der wie folgt definiert ist:

  - $\mathrm{LA}(b) = [f(\mathrm{LA}(b_0), \ldots, \mathrm{LA}(b_{n-1}))]_A$
  - Für jedes $s \in S_T$ setze

$$M_s(b) = \{\, q \in [t'[M_{s_0}(b_{i_0}), \ldots, M_{s_{r-1}}(b_{i_{r-1}})]]_I \mid \\ \text{es gibt ein } q_A \in \mathrm{LA}(b) \text{ und eine } \Sigma\text{-Transition} \\ s(f(x_0, \ldots, x_{n-1})) \to^{q_A} t'[s_0(x_{i_0}), \ldots, s_{r-1}(x_{i_{r-1}})] \in \Gamma_T \}$$

- ε-**Abschluß**: Sei $b \in Q_P$. Wir definieren den ε-Abschluß $\hat{b}$ zu b iterativ wie folgt: Setze zunächst $b_0 = b$. Dann ist

– $\mathrm{LA}(b_{i+1}) = \mathrm{LA}(b_i)$

– Für jedes $s \in S_T$ setze

$$M_s(b_{i+1}) = M_s(b_i) \cup \{\, q \in [t'[M_{s_0}(b_i), \ldots, M_{s_{r-1}}(b_i)]]_I \mid \text{es gibt ein } q_A \in \mathrm{LA}(b_i) \text{ und eine } \varepsilon\text{-Transition } s(x) \to^{q_A} t'[s_0(x), \ldots, s_{r-1}(x)] \in \Gamma_T \}$$

Setze $\hat{b} = b_k$ für ein k mit $b_k = b_{k+1}$. Da Q_P endlich ist, existiert ein solches k.

- **Endzustände** von P:

$$F_P = \{\, b \mid \exists\, s \in I_T, q \in F_I \colon q \in M_s(b) \}$$

Nun fehlt noch der Beweis der Korrektheit der Konstruktion.

Lemma: Sei $t \in T_\Sigma$. Dann existiert genau ein $b \in Q_P$, so daß $b \in [t]_P$, und dieser Zustand ist ε-abgeschlossen, d.h. $\bar{b} = b$.

Beweis: Folgt direkt aus der Konstruktion. □

Lemma: Sei $t \in T_\Sigma$, $b \in [t]_P$, und $s \in S_T$. Dann gilt:

1. $\mathrm{LA}(b) = [t]_A$
2. $M_s(b) = M_{s(t)}$ mit $M_{s(t)} = \{\, q \mid \exists\, t' \in T_\Sigma \colon q \in [t']_I \wedge s(t) \mid\!\!-_T^* t' \}$

Beweis: Die erste Aussage ist beweisbar mittels struktureller Induktion über t, wird hier nicht gezeigt. Die zweite Aussage zeigen wir durch doppelte Inklusion; zunächst die Richtung „$\subseteq$“ per Induktion über t:

- *Induktionsanfang:* Falls $t \in \Sigma_0$ ist, dann enthält P nach Konstruktion die Transition $t \to \hat{b}$, wobei $\hat{b}$ der ε-Abschluß von b ist und b wie oben (bei Definition des Transitionssystems) definiert mit $n = 0$ und $f = t$.

  Mit $b_0 = b$ seien die b_i wie oben definiert beim ε-Abschluß, dann ist $\hat{b} = b_k$ für $b_k = b_{k+1}$. Wir zeigen per Induktion über i, daß $M_s(b_i) \subseteq M_{s(t)}$. Damit folgt sofort, daß $M_s(\hat{b}) \subseteq M_{s(t)}$ ist.

  – *Induktionsanfang:* Im Fall $i = 0$ gilt $M_s(b_i) = M_s(b)$ mit

$$M_s(b) = \{\, q \in [t']_I \mid \text{es gibt ein } q_A \in \mathrm{LA}(b_i) \text{ und eine } \Sigma\text{-Transition } s(t) \to^{q_A} t' \in \Gamma_T \}$$

  Hieraus sieht man $M_s(b_i) = M_s(b) \subseteq M_{s(t)}$.

– *Induktionsvoraussetzung:* Sei $i \in \mathbb{N}$ und $M_s(b_i) \subseteq M_{s(t)}$ für alle $s \in S_T$. Zu zeigen: $M_s(b_{i+1}) \subseteq M_{s(t)}$.

– *Induktionsschluß:* Sei $q \in M_s(b_{i+1}) \setminus M_s(b_i)$. Dann existiert eine ε-Transition

$$s(x) \to^{q_A} t'[s_0(x), \ldots, s_{r-1}(x)] \in \Gamma_T \quad \text{mit } q_A \in \mathrm{LA}(b_i),$$

so daß gilt: $q \in [t'[M_{s_0}(b_i), \ldots M_{s_{r-1}}(b_i)]]_I$. Es gibt also $q_j \in M_{s_j}(b_i)$ mit $q \in t'[q_0, \ldots, q_{r-1}]_I$.

Nach Induktionsvoraussetzung existieren $t'_0, \ldots, t'_{r-1}$ mit $s_i(t) \vdash_T^* t'_j$ und $q_j \in [t'_j]_I$. Also $s(t) \vdash_T^* t'[t'_0, \ldots, t'_{r-1}] = t''$ und $q \in [t'']_I$. Dies zeigt, daß $M_s(b_{i+1}) \subseteq M_{s(t)}$.

- *Induktionsvoraussetzung:* Sei $t = f(t_0, \ldots, t_{n-1})$. Die Behauptung gelte für alle t_j.

- *Induktionsschluß:* Nach obigem Lemma existiert für alle j ein eindeutiger Zustand $b'_j \in [t_j]_P$, und es folgt, daß $f'(b'_0, \ldots, b'_{n-1}) \to \hat{b} \in \Delta_P$.

  Seien b sowie $b_0 = b, b_1, b_2$ wie oben definiert (Σ-Transitionen). Dann zeigt man, analog zum Fall $t \in \Sigma_0$ per Induktion, daß $M_s(b_i) \subseteq M_{s(t)}$ ist. Daraus folgt $M_s(\hat{b}) \subseteq M_{s(t)}$.

Nun die Gegenrichtung („$\supseteq$“) per Induktion über die Länge der Berechnung $s(t) \vdash_T^* t''$.

- *Induktionsanfang:* Falls die Länge null ist, so ist nichts zu zeigen, da keine Ausgabe produziert wird.

- *Induktionsvoraussetzung:* Es sei $t'' \in T_\Sigma$ eine Ausgabe, $q \in Q_I$ mit $q \in [t'']_I$ und $s(t) \vdash_T^* t''$. Wir müssen $q \in M_s(b)$ zeigen.

- *Induktionsschluß:* Wir unterscheiden, ob die erste Transition γ, die in $s(t) \vdash_T^* t''$ angewendet wird, eine Σ- oder eine ε-Transition ist.

  – Σ-Transitionen: Es sei γ von der Form wie oben (Σ-Transition). Damit ist t von der Form $f(t_0, \ldots t_{n-1})$ mit $t_i \in T_\Sigma$. Sei $b_i \in Q_P$ der eindeutige Zustand mit $b_i \in [t_i]_P$.

  Wir wissen, daß $\hat{b}$ der ε-Abschluß von b ist, mit b wie oben (bei den Σ-Transitionen). Nach Anwendung von γ auf $s(t)$ erhalten wir folgenden Term:

$$t'[s_0(t_{i_0}), \ldots, s_{r-1}(t_{i_{r-1}})]$$

Seien $t'_0, \ldots, t'_{r-1} \in T_\Sigma$ mit $s_j(t_{i_j}) \vdash_T^* t'_j$ und $t'' = [t'[q_0, ..., q_{r-1}]]_I$. Nach (1) gilt, daß $q_A \in \mathrm{LA}(b) = \mathrm{LA}(\hat{b}) = [t]_A$.

Nach Induktionsvoraussetzung ist $q_j \in M_{s_j}(b_{i_j})$. Damit folgt

$$t \in [t'[M_{s_{i_0}}(b_{i_0}), ..., M_{s_{i_{r-1}}}(b_{i_{r-1}})]]_I$$

Also gilt $t \in M_s(b) \subseteq M_s(\hat{b})$.

– ε-Transitionen: Analog; benutze, daß $\hat{b}$ ε-abgeschlossen ist. □

Lemma: Es gilt $L(P) = \tau_T^{-1}(L(I))$.

Beweis:

„$\subseteq$“ Sei $t \in L(P)$. Dann existiert ein $b \in F_P$ mit $b \in [t]_P$. Nach Definition von F_P existiert ein $s \in I_T$ und ein $q \in F_I$ mit $q \in M_s(b)$. Nun folgt aus vorigem Lemma, daß ein $t' \in T_\Sigma$ existiert, so daß $s(t) \models^* t'$ und $q \in [t']_I$. Damit gilt $t' \in L(I)$ und $(t, t') \in \tau_T$. Also ist $t \in \tau_T^{-1}(L(I))$.

„$\supseteq$“ Sei $t \in \tau_T^{-1}(L(I))$. D.h. es existiert $t' \in T_\Sigma$, $q \in F_I$ und $s \in I_T$ mit $s(t) \models^* t'$ und $q \in [t']_I$. Nach dem ersten obigem Lemma existiert ein eindeutiger Zustand $b \in [t]_P$. Mit dem zweiten obigen Lemma folgt auch, daß $q \in M_s(b)$ ist. Aus der Definition von F_P folgt damit, daß $b \in F_P$ ist, also gilt auch $t \in L(P)$. □

4.4 Protokoll- und Angreifermodell

Sei A eine endliche Menge von Konstanten (typischerweise Namen von Teilnehmern, Nonces, Schlüssel, ...) mit $\mathsf{secret} \in \mathsf{A}$ ein Geheimnis und $\mathsf{K} \subseteq \mathsf{A}$ der Menge der öffentlichen und privaten Schlüssel. Wir definieren die Bijektion $^{-1}$ auf K, die jedem privaten Schlüssel den öffentlichen Schlüssel zuordnet und umgekehrt.

Definition: Die Signatur Σ_A enthalte neben A noch folgende Mengen von Funktionssymbolen:

- einstellige Funktionssymbole für MACs: $\{\mathrm{hash}_a(\cdot) \mid a \in \mathsf{A}\}$
- einstellige Funktionssymbole für symmetrische und asymmetrische Verschlüsselung: $\{\mathrm{enc}^{\mathsf{s}}_a(\cdot) \mid a \in \mathsf{A}\}$ und $\{\mathrm{enc}^{\mathsf{a}}_k(\cdot) \mid k \in \mathsf{K}\}$
- zweistelliges Funktionssymbol zur Konkatenation $\{\langle \cdot, \cdot \rangle\}$ (nicht assoziativ, d.h. $\langle a, \langle b, c\rangle\rangle \neq \langle \langle a, b\rangle, c\rangle$).

Definition: Sei $\mathsf{M} = T_{\Sigma_\mathsf{A}}$ die *Menge der Nachrichten.*

Beispiel: Eine Nachricht hat beispielsweise die Form $\mathrm{enc}^{\mathsf{a}}_k(\langle b, \mathrm{enc}^{\mathsf{s}}_a(b)\rangle)$.

Um den Angreifer zu modellieren, benötigen wir nun die Menge aller Nachrichten, die er aus einer mitgelesenen Kommunikation extrahieren oder konstruieren kann.

Definition: Es seien $m, m' \in \mathsf{M}$, $a \in \mathsf{A}$, $k, k^{-1} \in \mathsf{K}$. Für $S \subseteq \mathsf{M}$ sei $d(S)$ die kleinste Menge mit $S \subseteq d(S)$, die folgende Bedingungen erfüllt:

(1)	Dekomposition:	$\langle m, m'\rangle \in d(S) \Rightarrow m, m' \in d(S)$
(2)	Sym. Entschl.:	$\mathrm{enc}^{\mathsf{s}}_a(m), a \in d(S) \Rightarrow m \in d(S)$
(3)	Asym. Entschl.:	$\mathrm{enc}^{\mathsf{a}}_k(m), k^{-1} \in d(S) \Rightarrow m \in d(S)$
(4)	Extraktion:	$\mathrm{hash}_a(m) \in d(S) \Rightarrow m \in d(S)$
(1′)	Konkatenation:	$m, m' \in d(S) \Rightarrow \langle m, m'\rangle \in d(S)$
(2′)	Sym. Verschl.:	$m, a \in d(S) \Rightarrow \mathrm{enc}^{\mathsf{s}}_a(m) \in d(S)$
(3′)	Asym. Verschl.:	$m, k \in d(S) \Rightarrow \mathrm{enc}^{\mathsf{a}}_k(m) \in d(S)$
(4′)	Hashing:	$m, a \in d(S) \Rightarrow \mathrm{hash}_a(m) \in d(S)$

Der Abschluß von S unter Verwendung der Bedingungen (1) bis (4) bezeichnen wir mit $\mathrm{an}(S)$, den Abschluß unter (1') bis (4') als $\mathrm{syn}(S)$.

Lemma: Für alle $S \subseteq \mathsf{M}$ gilt $d(S) = \mathrm{syn}(\mathrm{an}(S))$.

Wir definieren nun Protokolle:

Definitionen:

- Ein *Nachrichtentransducer (NT)* oder Empfang-Sende-Schritt ist ein Baumtransducer über Σ_A.
- Eine *Instanz* Π ist eine endliche Folge $T_1, \ldots, T_n$ von Nachrichtentransducern.
- Ein *Protokoll* ist ein Tupel $(\{\Pi_i \mid i < n\}, I)$ mit einer Familie von Instanzen und einem initialen endlichen Angreiferwissen $I \subseteq \mathsf{M}$.

Als nächsten Schritt definieren wir Angriffe; dazu stellen wir uns den Angreifer so vor, daß er Kontrolle über das Netzwerk hat und mit verschiedenen Parteien mit verschiedenen Protokollinstanzen kommuniziert. D.h. er legt jeweils die Nachrichten fest, die an einzelne Transducer geschickt werden – siehe Graphik 13.

Definition: Sei P ein Protokoll wie oben mit $\Pi_i = T^i_0, \ldots, T^i_{n_i-1}$. Ein *Angriff* auf P besteht nun aus den folgenden Komponenten:

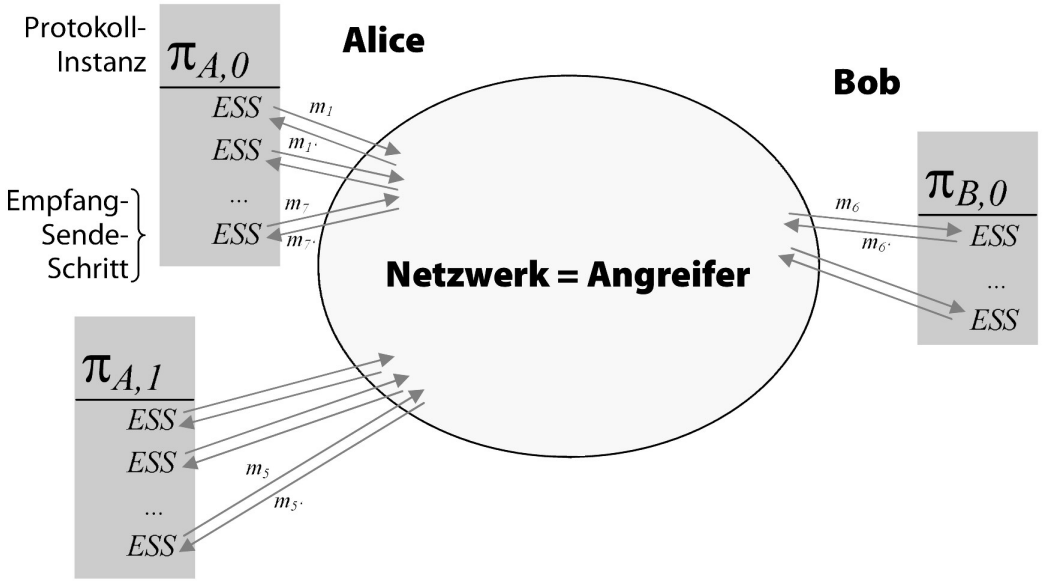


Abbildung 13: das Protokoll- und Angreifer-Modell

- einer Menge $O \subseteq \{(i,j) \mid i < n, j < n_i\}$, so daß für $(i,j') \in O$ auch $(i,j) \in O$ für alle $j < j'$ ist.
- einer totalen Ordnung $<$ auf O, so daß aus $(i,j) < (i',j')$ folgt, daß $j < j'$ ist,
- einer Abbildung ψ, die jedem $(i,j) \in O$ ein Tupel $(r_j^i, s_j^i) \in \tau_{T_j^i}$ zuordnet, so daß mit $S_j^i = I \cup \left\{ s_{j'}^{i'} \;\middle|\; (i',j') < (i,j) \right\}$ gilt:
  1. $r_j^i \in d(S_j^i)$ für alle $(i,j) \in O$ und
  2. $\mathsf{secret} \in d(I \cup \left\{ S_j^i \;\middle|\; (i,j) \in O \right\})$.

Die Menge O gibt die Menge der vom Angreifer verwendeten Protokollschritte an, wobei die Bedingungen (und auch die totale Ordnung auf O) sicherstellen, daß der Angreifer die Protokollschritte in der richtigen Reihenfolge abarbeiten muß und daß die Protokolle lückenlos ausgeführt werden, d.h. daß keine Protokollschritte übergangen werden.

4.5 Entscheidbarkeit der Sicherheit von Protokollen

Wir können nun unsere Suche nach Angriffen auf Protokolle als Entscheidungsproblem formulieren:

Definition: Das Entscheidungsproblem ANGRIFF sei folgendes Problem:

$$\text{ANGRIFF} = \{ P \mid \exists \text{ Angriff auf } P \}$$

Falls $P \notin$ ANGRIFF ist, so nenen wir P *sicher*.

Beispiel: Modell des Servers im Bull-Otway-Rekursionsprotokoll (siehe oben): Seien $P_0, \ldots, P_{n-1}$ Teilnehmer und $S = P_n$. Der symmetrische Schlüssel zwischen P_i und S sei k_i, der vom Server erzeugte Schlüssel für die Kommunikation zwischen P_i und P_j sei s_{i-j}. Der Nachrichtentransducer hat die Form

$$T = (\{\texttt{start}, \texttt{read}\}, \Sigma_{\mathsf{A}}, \{\texttt{start}\}, \cdot, \Gamma)$$

Damit kann S genau einen Empfang-Sende-Schritt ausführen. Sei nun Γ definiert mit $i, j, r \leq n$ und x_0, x_1, x_2 Variablen. Dann verwende folgende Transitionen:

$$\begin{aligned}
\texttt{start}(\text{hash}_{k_i}(P_i, P_n, x_0, x_1)) &\rightarrow \texttt{read}(\text{hash}_{k_i}(P_i, P_n, x_0, x_1)) \\
\texttt{read}(\text{hash}_{k_i}(P_i, P_j, x_0, -)) &\rightarrow \text{enc}^{\mathsf{s}}_{k_i}(s_{ij}, P_j, x_0) \\
\texttt{read}(\text{hash}_{k_i}(P_i, P_j, x_0, & \\
\quad \text{hash}_{k_r}(P_r, P_i, x_1, x_2))) &\rightarrow \big\langle \text{enc}^{\mathsf{s}}_{k_i}(s_{ij}, P_j, x_0), \big\langle \text{enc}^{\mathsf{s}}_{k_i}(s_{ri}, P_r, x_0), \\
& \qquad \texttt{read}(\text{hash}_{k_r}(P_r, P_i, x_1, x_2)) \big\rangle \big\rangle
\end{aligned}$$

Satz: ANGRIFF ist entscheidbar.

Wir werden dieses Problem zurückführen auf das folgende Entscheidungsproblem:

Definition: Gegeben sei eine endliche Menge $I \subseteq \mathsf{M}$ und eine Folge $T_0, \ldots, T_{n-1}$ von Nachrichtentransducern über Σ_{A}. Dann ist $(I, T_0, \ldots, T_{n-1}) \in$ SEQUENZANGRIFF, falls gilt: Es gibt Nachrichten $(r_i, s_i) \in \tau_{T_i}$, so daß mit $r_n = \mathsf{secret}$ gilt: $r_i \in d(I \cup \{s_0, \ldots, s_{i-1}\})$ für alle $i \leq n$.

Ist SEQUENZANGRIFF entscheidbar, dann offensichtlich auch ANGRIFF, indem wir eine Folge von Transducern raten. Wir werden jetzt das Problem SEQUENZANGRIFF noch auf das Iteriertes-Urbild-Problem zurückführen.

Satz: Es existiert ein Nachrichtentransducer T_{der}, so dass $\tau_{T_{\text{der}}}(m) = d(\{m\})$ für alle $m \in \mathsf{M}$.

Im Folgenden konstruieren wir einen solchen Transducer $T_{\text{der}} = (S, \Sigma_{\mathsf{A}}, I, D, \Gamma)$.

- **Look-Ahead-Automat** D: Zunächst definieren wir $D = (Q_D, \Sigma_{\mathsf{A}}, \Delta_D)$. Mit Hilfe dieses (vollständigen und deterministischen) Semi-BWBA werden wir feststellen, welche Schlüssel, d.h. welche atomaren Nachrichten, von m abgeleitet werden können. Die Zustandsmenge Q_D ist die Menge aller Funktionen der Form

  $$Q_D = 2^{\mathsf{A}} \to 2^{\mathsf{A}}$$

  Wir wollen die Transitionen von D so definieren, dass $[m]_D(K) = \text{an}(\{m\} \cup K) \cap \mathsf{A}$ für alle $m \in \mathsf{A}$ und $K \subseteq \mathsf{A}$.

  - Für jedes $a \in \mathsf{A}$ enthält Δ_D die Transition

    $$a \to d \text{ mit } d(K) := K \cup \{a\} \; \forall \; K \subseteq \mathsf{A}$$

  - Für alle $a \in \mathsf{A}$ und $d \in Q_D$ enthält Δ_D eine Transition

    $$\text{enc}^{\mathsf{s}}_a(d) \to d' \text{ mit } d'(K) = \begin{cases} K & \text{falls} \quad a \notin K \\ d(K) & \text{falls} \quad a \in K \end{cases}$$

  - Für alle $k \in \mathsf{K}$ und $d \in Q_D$ enthält Δ_D eine Transition

    $$\text{enc}^{\mathsf{s}}_k(d) \to d' \text{ mit } d'(K) = \begin{cases} K & \text{falls} \quad k^{-1} \notin K \\ d(K) & \text{falls} \quad k^{-1} \in K \end{cases}$$

  - Für alle $a \in \mathsf{A}$ und $d \in Q_D$ enthält Δ_D eine Transition

    $$\text{hash}_a(d) \to d$$

  - Für alle $d, d' \in Q_D$ enthält Δ_D eine Transition

    $$\langle d, d' \rangle \to d''$$

    wobei für alle $K \subseteq \mathsf{A}$ die Menge $d''(K)$ minimal ist, so dass
    - $K \subseteq d''(K)$,
    - $K' \subseteq d''(K) \Rightarrow d(K') \subseteq d''(K)$,
    - $K' \subseteq d''(K) \Rightarrow d'(K') \subseteq d''(K)$.

Lemma: Für alle $m \in \mathsf{M}$ und $K \subseteq \mathsf{A}$ gilt:

$$[m]_D(K) = an(\{m\} \cup K)$$

Beweis: Strukturelle Induktion über m, als Übung.

- **Zustandsmenge:**

$$S = \{s_I\} \cup ((\{s_{\text{syn}}, s_{\text{an}}\}) \times 2^{\mathsf{A}}) \text{ und } I = \{s_I\}$$

- **Transitionen**: Zunächst als Idee. Eingabe sei m, dann ermitteln wir ausgehend von s_I die Schlüssel (Atome), die von m abgeleitet werden können. In der syn-Phase (s_{syn}) konstruieren wir die Nachricht $m_{\text{syn}} = t[m, \ldots, m] \in \mathsf{M}$ für einen linearen Term t. In der an-Phase (s_{an}) wird die Nachricht erzeugt, indem m durch Nachrichten in $\text{an}(\{m\})$ ersetzt wird.

  Dazu enthält Γ die folgenden Transitionen:

  - Für alle $d \in Q_D$: $s_I(x) \to^d (s_{\text{syn}}, d(\emptyset))(x)$.
  - Für alle $K \subseteq \mathsf{A}$:

$$\begin{array}{lcll}
(s_{\text{syn}}, K)(x) & \to & \langle (s_{\text{syn}}, K)(x), (s_{\text{syn}}, K)(x) \rangle & \\
(s_{\text{syn}}, K)(x) & \to & \text{enc}^{\mathsf{s}}_a((s_{\text{syn}}, K)(x)) & \text{für } a \in K \\
(s_{\text{syn}}, K)(x) & \to & \text{enc}^{\mathsf{a}}_k((s_{\text{syn}}, K)(x)) & \text{für } k \in K \cap \mathsf{K} \\
(s_{\text{syn}}, K)(x) & \to & \text{hash}_a((s_{\text{syn}}, K)(x)) & \text{für } a \in K \\
(s_{\text{syn}}, K)(x) & \to & (s_{\text{an}}, K)(x) & \\
(s_{\text{an}}, K)(x) & \to & x & \\
(s_{\text{an}}, K)(x) & \to & a & \text{für } a \in K \\
(s_{\text{an}}, K)(\langle x_0, x_1 \rangle) & \to & (s_{\text{an}}, K)(x_0) & \\
(s_{\text{an}}, K)(\langle x_0, x_1 \rangle) & \to & (s_{\text{an}}, K)(x_1) & \\
(s_{\text{an}}, K)(\text{enc}^{\mathsf{s}}_a(x)) & \to & (s_{\text{an}}, K)(x) & \text{für } a \in K \\
(s_{\text{an}}, K)(\text{enc}^{\mathsf{a}}_k(x)) & \to & (s_{\text{an}}, K)(x) & \text{für } k^{-1} \in K \\
(s_{\text{an}}, K)(\text{hash}_a(x)) & \to & (s_{\text{an}}, K)(x) & \text{für } a \in K
\end{array}$$

Die Korrektheit der Konstruktion ist nun leicht einzusehen. □

Wir können nun die Reduktion von SEQUENZANGRIFF auf das Iteriertes-Urbild-Problem zurückführen. Sei $I = \{m_0, \ldots, m_{n-1}\}$, $T_0, \ldots, T_{l-1}$ wie oben. Wir wollen jetzt im Prinzip feststellen, ob

$$\mathsf{secret} \in \tau_{T_{\text{der}}} \circ \tau_{T_{l-1}} \circ \tau_{T_{\text{der}}} \circ \ldots \circ \tau_{T_{\text{der}}} \circ \tau_{T_0} \circ \tau_{T_{\text{der}}}(I)$$

Hier müssen wir noch ergänzen, daß das wachsende Wissen des Angreifers mittransportiert wird.

- Wir definieren eine Variante $T^\star_{\text{der}}$ von T_{der}, so daß

$$\tau_{T^\star_{\text{der}}} = \{\, (m, \langle m, m' \rangle) \mid\ m' \in d(\{m\}) \}$$

Dies ist möglich, indem $T^\star_{\text{der}}$ einen neuen Anfangszustand $s^\star_I$ enthält sowie eine neue Transition

$$s^\star_I(x) \to \langle x, s_I(x)\rangle\,.$$

Nun schreiben wir $\tau^\star_{\text{der}} = \tau_{T^\star_{\text{der}}}$.

- Wir definieren eine Variante $T^\star_i$ von T_i, so daß

$$\tau_{T^\star_i} = \{\,(\langle m, m'\rangle\,, \langle m, m''\rangle \mid (m', m'') \in \tau_{T_i}\}$$

Dazu ergänzen wir für jeden initialen Zustand s von T_i die Menge der Transitionen von T_i um die Transitionen

$$s^\star(\langle x_0, x_1\rangle) \to \langle x_0, s(x_1)\rangle$$

Die neue Menge der Anfangszustände ist $I^\star_i = \{\, s^\star \mid s \in I_i\}$.

Wir definieren zudem $R = \{\mathsf{secret}\}$, dies ist offensichtlich regulär, und weiter eine Konkatenation aller Nachrichten des initialen Angreiferwissens:

$$m_I = \langle m_0, \langle m_1, \ldots, \langle m_{n-2}, m_{n-1}\rangle \ldots\rangle\rangle$$

Setze nun

$$\tau = \tau_{T_{\text{der}}} \circ \tau^\star_{l-1} \circ \tau^\star_{\text{der}} \circ \tau^\star_{l-2} \circ \tau^\star_{\text{der}} \circ \ldots \circ \tau^\star_{\text{der}} \circ \tau^\star_0 \circ \tau^\star_{\text{der}}$$

Lemma: Es gilt

$$(I, T_0, \ldots, T_{l-1}) \in \text{SEQUENZANGRIFF} \iff m_I \in \tau^{-1}(R)$$

Es folgt:

Satz: SEQUENZANGRIFF ist entscheidbar.

Komplexitätsüberlegungen: Das Verfahren ist nicht elementar, d.h. es ist von der Komplexität

$$2^{2^{\overset{\overset{2}{\vdots}}{}}}\Big\} n$$

5 Syntaxanalyse

5.1 Einführung

Im Compilerbau werden Automaten in der Syntaxanalyse verwendet. Dabei ist der Ablauf wie folgt: Aus dem Quellcode wird mittels der *lexikalischen Analyse* eine Folge von *Tokens* generiert (zum Beispiel $\langle 42, \texttt{int} \rangle$. Mittels der *Syntaxanalyse* erzeugt man dann einen *Syntaxbaum*, d.h. ein Ableitungsbaum für die zugrundeliegende Grammatik.

Der CYK-Algorithmus, der das Wortproblem zu kontextfreien Grammatiken löst, hat die pessimale Laufzeit $\Theta(|n|^3)$. Effizientere allgemeine Verfahren, die ebenfalls alle kontextfreien Grammatiken erkennen, haben mindestens quadratische Laufzeit. Daher schränkt man sich auf gewisse Klassen von kontextfreien Grammatiken ein, um auf diesen Klassen lineare Algorithmen zu entwickeln.

Definitionen:

- Eine Kontextfreie Grammatik ist ein Tupel $G = (A, N, S, P)$ mit einem Terminalalphabet A, einem Nichtterminalalphabet N, einem Startsymbol $S \in N$ und einer Menge von Transitionen $P \subseteq N \times (A \cup N)^*$. Dabei nimmt man an, daß A und N disjunkt sind; weiterhin bezeichne $\Sigma = A \cup N$.
- Eine *Satzform* ist ein Element aus Σ^*.
- Ein *Ableitungsschritt* ist $\alpha_0 B \alpha_1 \vdash \alpha_0 \beta \alpha_1$ für $B \to \beta \in P$. *Rechts-* bzw. *Linksableitungsschritte* sind solche, bei denen zusätzlich $\alpha_1 \in A^*$ bzw. $\alpha_0 \in A^*$ ist, Schreibweise dann $\vdash_r$ bzw. $\vdash_l$.
- Eine *Ableitung* ist eine Folge von Ableitungsschritten $\alpha \vdash^* \beta$ etc.
- Die erzeugte *Sprache* ist gleich

$$\begin{aligned} L(G) &= \{u \in A^* \mid S \vdash^* u\} \\ &= \{u \in A^* \mid S \vdash_r^* u\} \\ &= \{u \in A^* \mid S \vdash_l^* u\} \end{aligned}$$

Bemerkung: Aus Links- und Rechtsableitungen lassen sich schnell Ableitungsbäume gewinnen.

Das **Problem** ist nun, wie man zu einem gegebenen $u \in L(G)$ einen Ableitungsbaum bestimmt, und das möglichst in linearer Zeit. **Methode** dazu ist

die Konstruktion von deterministischen Kellerautomaten – diese sind generell schwächer als nichtdeterministische Kellerautomaten[9].

5.2 Kellerautomaten

Wir werden Kellerautomaten mit Vorausschau und Endmarkierung verwenden.

Definition: Ein Kellerautomat sei ein Tupel $(A, C, Q, \$, q_I, z_I, \Delta)$ mit

- einem Alphabet A,
- einem Kelleralphabet C,
- einer Zustandsmenge Q,
- einer Endmarkierung $\$ \notin A$ (oder $\lhd$),
- einem Anfangszustand q_I,
- einem Kellerstartsymbol z_I,
- einer Übergangsrelation

$$\Delta \subseteq Q \times A^*(\varepsilon + \$) \times C^* \times A^*(\varepsilon + \$) \times C^* \times Q$$

Dabei gibt die zweite Komponente an, was wir (mit Vorausschau) sehen, die vierte Komponente hingegen, was wir lesen. Zusätliche Bedingung daher: Für alle $(q, u, \alpha, v, \beta, q') \in \Delta$ muß v Präfix von u sein, d.h. $v \in \mathrm{pfx}(u)$.

Definitionen:

- Eine *Konfiguration* ist ein Tupel (q, u, α).
- Ein *Berechnungsschritt* ist von der Form $(q, uvw, \alpha\beta) \to (q', vw, \alpha\beta')$, falls $(q, uv, \beta, u, \beta', q') \in \Delta$ ist. Dabe sieht der Automat uv, oben auf dem Keller ist β, der Automat liest u, ersetzt β durch β' und geht in den Zustand q' über.
- Eine *akzeptierende Berechnung* ist eine solche mit $(q_I, u\$, z_I) \vdash^* (q, \varepsilon, \varepsilon)$.

Bemerkung: Damit der Kellerautomat deterministisch ist, darf es keine

[9] wohlgeklammert und $a^n b^n$ ist kein Problem, aber Palindrome.

zwei Transitionen $(q, u, \alpha, \ldots), (q, u', \alpha', \ldots) \in \Delta$ geben mit $u \in \text{pfx}(u')$ und entweder $\alpha \in \text{sfx}(\alpha')$ oder $\alpha' \in \text{sfx}(\alpha)$.

Wenn wir Kellerautomaten mit Ausgabe benötigen, erweitern wir diese um ein zusätzliches Ausgabealphabet D, die Transitionsrelationen sind dann von der Form

$$Q \times A^*(\varepsilon + \$) \times C^* \times A^*(\varepsilon + \$) \times C^* \times \underbrace{D^*}_{\text{Ausgabe}} \times Q$$

Definition: Ein *Kellerautomat mit regulärer Kontrolle* ist ein Tupel (A, B) mit

- einem Kellerautomaten $A = (A, C, Q, \$, z_I, \Delta)$ mit einer reduzierten Transitionsrelation der Form

$$Q \times A^*(\varepsilon + \$) \times C^* \times A^*(\varepsilon + \$) \times C^*$$

- einem DEA $B = (C, Q, q_I, \delta)$

Der aktuelle Zustand ist dann immer gegeben durch $\delta^*(q_I, \alpha)$ mit aktuellem Kellerinhalt α.

Definitionen:

- L ist *deterministisch kontextfrei*, falls L von einem deterministischen Kellerautomaten mit Endmarkierung erkannt wird. L ist *strikt deterministisch kontextfrei*, falls L von einem deterministischen Kellerautomaten ohne Endmarkierung erkannt wird.
- G ist reduziert, wenn für alle $B \in N \setminus \{S\}$ Satzformen α, β, u existieren mit $S \vdash^* \alpha B \beta \vdash^* u$.
- G ist schwach startsepariert, wenn $S \vdash^+ S$ nicht gilt.

Beispiel: Betrachte folgende Grammatik:

$$S \to a_1 B_1 c_1 \mid a_2 B_2 c_2 \quad B_1 \to b \quad B_2 \to b$$

Die Sprache enthält die Worte $a_1 b c_1$ und $a_2 b c_2$. Beim Lesen des Symbols b müßte der Automat dieses zu B_1 oder B_2 reduzieren, das kann er jedoch nur, indem er nach dem bisherigen Keller-Inhalt a_1 bzw. a_2 entscheidet, dies erlaubt die reguläre Kontrolle auf einfache Weise. Nötig ist sie jedoch nicht:

Lemma: Jeder Kellerautomat mit regulärer Kontrolle ist äquivalent zu einem herkömmlichen Kellerautomaten und umgekehrt.

Beweisskizze:

„⇒“ Idee: Kodiere den von einem Automaten „gewünschten“ Zustand im obersten Kellersymbol.

„⇐“ Idee: Kodiere den von B erreichten Zustand in einer zweiten Komponente im Kellersymbol.

Die Umwandlung einer kontextfreien Grammatik G in einen Automaten kann auf zwei Weisen erfolgen:

- *Produce-Shift-Automat*: $(A, N, \{q\}, q, S, \Delta)$ mit

$$\Delta = \left\{ (q, \varepsilon, B, \varepsilon, \overleftarrow{\beta}, q) \;\middle|\; (B \to \beta) \in G \right\} \cup \{ (q, a, a, a, \varepsilon, q) \mid a \in A \}$$

- *Shift-Reduce-Automaten*: Der Automat soll eine umgekehrte Rechtsableitung erzeugen.

  Beispiel: Für die Grammatik $S \to SS \,|\, (S) \,|\, \varepsilon$ und den Ausdruck „(())()“ wäre eine Rechtsableitung:

$$S \vdash SS \vdash S(S) \vdash S() \vdash (S)() \vdash ((S))() \vdash (())()$$

  Der Automat soll diese Ableitung von rechts nach links liefern, das hieße in diesem Fall (mit s für die Shift-Vorgänge, r für Reduce):

			)						)		
		S	S		)			S	S		
	(	(	(	S	S		(	(	(	S	
(	(	(	(	(	(	S	S	S	S	S	S
s	s	r	s	r	s	r	s	r	s	r	r

  Formal: Der Automat sei $(A, N, \{q\}, \$, q, \varepsilon, \Delta)$ mit

$$\begin{aligned} \Delta = \quad & \{ (q, a, \varepsilon, a, a, q) \mid a \in A \} && \text{Shift-Regeln} \\ \cup\, & \{ (q, \varepsilon, \beta, \varepsilon, B, q) \mid (B \to \beta) \in G \} && \text{Reduce-Regeln} \\ \cup\, & \{ (q, \$, S, \varepsilon, \varepsilon, q) \} && \text{Ende} \end{aligned}$$

  Bemerkung: Dies funktioniert so noch nicht vollständig, da in manchen Fällen zu früh die End-Regel angewandt werden kann. Dies wird behoben durch die Einführung eines neuen Zustands.

Bemerkung: Im allgemeinen Fall kann ein Automat gezwungen sein, an einer speziellen Stelle viele ε-Transitionen zu verwenden: Betrachte die Sprache

$$\left\{ 0^i 1^j a 2^k \;\middle|\; (a = 3 \wedge j = k) \vee (a = 4 \wedge i = k) \right\}$$

Hier muß der Automat zunächst alle 0 und 1 auf dem Keller speichern und im Falle $a = 4$ beim Lesen von a durch entsprechende ε-Transitionen alle Vorkommen von 1 vom Keller löschen.

Lemma: Für jeden deterministischen Kellerautomaten gilt: Die Anzahl der Schritte in terminierenden Berechnungen ist $\mathcal{O}(n)$ für eine Eingabe der Länge n.

Beweisskizze:

1. Umbau des Automaten in eine *schwache Normalform*: Veränderungen des Kellers bestehen nur noch im Löschen oder Hinzufügen *eines* Symbols; der neue Automat benötigt dann mindestens so viele Schritte wie der alte.

2. Benutze ein *Pumping-Argument*: Es gibt eine Konstante c, so daß der Kellerautomat nach Hintereinanderausführung von c vielen ε-Schritten entweder den Keller um ein Symbol abgebaut hatoder nicht terminiert.

   Sei dazu eine Konfiguration gegeben mit einem Keller der Länge n, und einem weiteren Lauf, in dem der Automat ab diesem Zeitungpunkt immer mindestens einen Keller der Länge n behält. Sei $d = |C| \cdot |Q|$. Fallunterscheidung:

   - Falls der Automat ab diesem Zeitpunkt immer höchstens $n + d$ Symbole auf dem Keller behält (dargestellt in Graphik 14), dann wiederholt sich irgendwann eine Konfiguration, da es nur endlich viele Möglichkeiten für Konfigurationen mit höchstens d Kellersymbolen und jeweils einem Zustand gibt. Daher gerät der Automat in eine Endlosschleife.
   - Falls die Länge des Kellers zu einem Zeitpunkt $n + d$ übersteigt (dargestellt in Graphik 15), dann wiederholt sich irgendwann eine Konfiguration aus oberstem Kellersymbol und Zustand, wobei zwischen diesen auch nicht mehr auf tiefere Kellersymbole zugegriffen wird. Auch in diesem Fall terminiert die Berechnung nicht. Hier ist $c = d^d$.

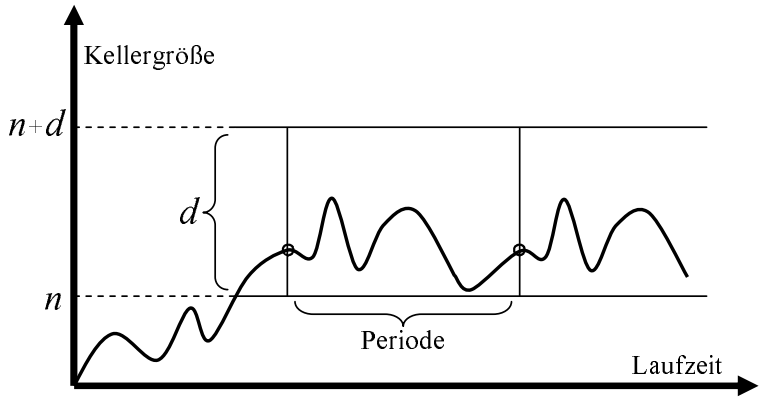


Abbildung 14: Beweis zur linearen Laufzeit: beschränkter Fall

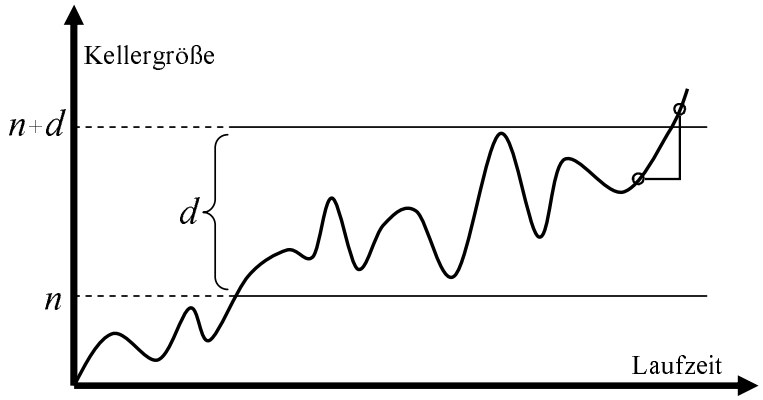


Abbildung 15: Beweis zur linearen Laufzeit: unbeschränkter Fall

5.3 LR-Grammatiken

Ziel: Wir wollen den kanonischen Shift-Reduce-Kellerautomaten durch reguläre Kontrolle und Vorausschau – wenn möglich– deterministisch machen.

Definition: Definiere zu jeder Satzform α dessen *First-Menge*:

$$\text{first}(\alpha) = \{u \in A^*(\varepsilon + \$) \mid \exists\, v\colon \alpha \vdash_r^* uv\}$$

Für alle $k \in \mathbb{N}$ definiere zusätzlich:

$$\text{first}_k(\alpha) = \left\{u \in A^{\leq k} \;\middle|\; \alpha \vdash^* u\right\} \cup \left\{u \in A^k \;\middle|\; \exists v\colon \alpha \vdash^* uv\right\}$$

Wir überlegen uns nun, wie eine Grammatik G beschaffen sein muß, um

mit $k \in \mathbb{N}$ Symbolen Vorausschau einen deterministischen Kellerautomaten konstruieren zu können, der $L(G)$ erkennt.

5.3.1 Charakterisierung durch mögliche Ableitungen

Definition: Sei $k \in \mathbb{N}$, G eine Grammatik. G heiß LR (k)-*Grammatik*, falls gilt: Falls die Grammatik zwei Ableitungen erlaubt der Form

$$\begin{array}{llllllll} S & \vdash_r^* & \alpha_0 B\, u & \vdash & \alpha_0 \beta\, u & = & \alpha u \\ S & \vdash_r^* & \alpha_0' B' u' & \vdash & \alpha_0' \beta' u' & = & \alpha v \end{array} \quad \text{mit } \operatorname{first}_k(u) = \operatorname{first}_k(v), \qquad (1')$$

dann ist

$$\alpha_0 = \alpha_0', \quad B = B', \quad \beta = \beta', \quad u' = v. \qquad (2')$$

Lemma: In die Prämisse darf man auch $|\alpha_0'\beta'| \geq |\alpha_0\beta|$ aufnehmen, ohne daß sich die Klasse der LR (k)-Grammatiken ändert.

Beweis: Zu zeigen: Wenn G keine LR (k)-Grammaitk ist, so wird die neue Bedingung verletzt. Da G keine LR (k)-Grammatik, gibt es Beispiele, die die ursprüngliche Bedingung verletzen.

Wir wählen eines mit minimalem $n := |\alpha_0\beta|$. Falls $|\alpha_0'\beta'| \geq n$, so sind wir fertig – betrachte den anderen Fall. Dann gibt es $x \in A^+$ mit $\alpha_0'\beta' x = \alpha_0\beta$, da $\alpha_0'\beta'$ ein Präfix von α sein muß.

Damit gilt auch: $\alpha_0'\beta' xv = \alpha_0\beta v = \alpha v = \alpha_0'\beta' u'$, also auch $xv = u'$. Also gilt insgesamt:

$$\begin{array}{llllllll} S & \vdash_r^* & \alpha_0' B' u' & \vdash & \alpha_0' \beta' u' & & = & \gamma u' \\ S & \vdash_r^* & \alpha_0 B\, u & \vdash & \alpha_0 \beta\, u & = \alpha_0' \beta' x u & = & \gamma x u \end{array}$$

Weiter wissen wir aus $\operatorname{first}_k(u) = \operatorname{first}_k(v)$ auch $\operatorname{first}_k(xu) = \operatorname{first}_k(xv) = \operatorname{first}_k(u')$. Damit ist die Prämisse aus der Definition der LR (k)-Grammatiken erfüllt, aber $xu \neq u$, d.h. dies ist ein Gegenbeispiel dafür, daß die Grammatik LR (k) ist.

Dieses Gegenbeispiel ist jedoch *kürzer* als das im Beweis verwendete, dies ist ein Widerspruch zur Wahl des kleinsten Gegenbeispiels! □

Beispiel: Betrachte die Grammatik

$$S \to aB_1c_1 \,|\, aB_2c_2 \quad B_1 \to b \quad B_2 \to b$$

Diese Sprache enthält die beiden Wörter abc_1 und abc_2. Es existieren folgende Ableitungen:

$$S \vdash aB_1c_1 \vdash abc_1 \quad \text{und} \quad S \vdash aB_2c_2 \vdash abc_2$$

Da $\text{first}_0(u) = \varepsilon = \text{first}_0(v)$ ist, ist die Prämisse für LR (0)-Grammatiken erfüllt, aber wegen $B_1 \neq B_2$ gilt $B \neq B'$. Damit ist die Grammatik nicht LR (0). Die Grammatik ist aber eine LR (1)-Grammatik.

Definitionen:

- Eine *Rechtssatzform* ist eine Satzform, die in einer Rechtsableitung beginnend mit dem Startsymbol vorkommt. Ein *Rechtspräfix* ist eine Satzform α, für die es ein u gibt, so daß αu eine Rechtssatzform ist.
- Eine *punktierte Regel* ist eine Tripel (B, β_0, β_1) mit $B \to \beta_0\beta_1 \in P$, das als $B \to \beta_0 \cdot \beta_1$ geschrieben wird.
- Ein *Rechtsdatum* ist ein Tupel $[B \to \beta_0 \cdot \beta_1, u]$ mit $u \in A^*$. Die Menge aller Rechtsdaten mit $u \in A^k$ bezeichnen wir mit $\text{P}_k(G)$.
- Wir definieren nun noch $\text{P}_k(\alpha)$ für eine Satzform α: Es gilt

  $$[B \to \beta_0 \cdot \beta_1, u] \in \text{P}_k(\alpha),$$

  falls α_0 mit $\alpha = \alpha_0\beta_0$ und v mit $u \in \text{first}_k(v)$ existieren, so daß es folgende Ableitung gibt:

  $$S \vdash_r^* \alpha_0 B v \vdash_r \underbrace{\alpha_0\beta_0}_{\alpha} \beta_1 v$$

5.3.2 Charakterisierung durch Rechtspräfixe

Satz: Folgende Aussagen sind äquivalent:

(1) G ist LR (k)-Grammatik.

(2) Für jedes Rechtspräfix α und zwei unterschiedlichen Rechtsdaten

$$[B \to \beta_0 \cdot \varepsilon, u] \in \text{P}_k(\alpha), \quad [B' \to \beta_0 \cdot \beta_1, v] \in \text{P}_k(\alpha)$$

gilt: $u \notin \text{first}_k^{\notin}(\beta_1 v)$.

Dies heißt, daß bei der Syntaxanalyse nach dem Lesen von β_0 mit Hilfe der (vorberechneten) $\text{first}_k^{\notin}(\cdot)$-Mengen entschieden werden kann, ob schon reduziert wird (zu B), oder ob noch β_1 gelesen und dann zu B' reduziert wird.

Dafür sei definiert:

Definition: Die Menge $\text{first}_k^{\not\in}(\alpha)$ ist definiert als:

$$\text{first}_k^{\not\in}(\alpha) = \begin{cases} \{ w \in \text{first}_k(\alpha) \mid \exists\, v\colon \alpha \mid\!\!\!\not\vdash_r^+ wv \} & \text{falls} \quad \alpha \in N\Sigma^* \\ \text{first}_k(\alpha) & \text{sonst} \end{cases}$$

Dabei gilt $\alpha \mid\!\!\!\not\vdash_r^+ u$, falls $\beta \notin Nu$ existiert mit $\alpha \vdash_r^* \beta \vdash_r u$.

Beweis des obigen Satzes, zunächst die Hinrichtung durch Kontraposition. Sei eine LR (k)-Grammatik gegeben, für die Bedingung (2) nicht erfüllt ist. Wir werden hieraus einen Widerspruch ableiten.

Folgende Ableitungen sind möglich (mit $\text{first}_k(x) = u \in \text{first}_k^{\not\in}(\beta_1 x')$ und $v \in \text{first}_k(\beta_1 x')$ und $\alpha_0'\beta_0 = \alpha_0\beta = \alpha$):

$$\begin{array}{lllllll} S & \vdash_r^* & \alpha_0 B\, x & \vdash & \alpha_0\beta \quad x & \text{wegen} & [B \to \beta\, \cdot\varepsilon,\ u] \in \mathrm{P}_k(\alpha) \\ S & \vdash_r^* & \alpha_0' B' x' & \vdash & \alpha_0'\beta_0\beta_1 x' & \text{wegen} & [B' \to \beta_0 \cdot \beta_1, v] \in \mathrm{P}_k(\alpha) \end{array}$$

Weiter gilt für ein $y \in A^*$, daß $\beta_1 x' \mid\!\!\!\not\vdash_r^+ uy$. **Fallunterscheidung**:

1. Fall: $\beta_1 = \varepsilon$. Es gilt $u \in \text{first}_k(v) = \{v\}$, d.h. $u = v$. Außerdem gilt $S \vdash_r^* \alpha_0' B' x' \vdash_r \alpha_0'\beta_0 x'$.

   Wende die Bedingung $(1')$ für LR (k)-Grammatiken an, dann gilt: $\alpha_0 = \alpha_0'$, $B = B'$ und $\beta = \beta_0$. Damit sind die zwei Rechtsdaten $[B \to \beta \cdot \varepsilon, u]$ und $[B' \to \beta_0 \cdot \beta_1, v]$ identisch, Widerspruch.

2. Fall: $\beta_1 = z \in A^+$. Dann gilt: $u \in \text{first}_k(zx')$, außerdem $v \in \text{first}_k(zx')$, da diese Menge aber nur ein Element enthält, ist $u = v = \text{first}_k(x) = \text{first}_k(zx')$, außerdem: $S \vdash_r^* \alpha_0' B' x' \vdash_r \alpha_0'\beta_0 z x' = \alpha z x'$. Da G LR (k)-Grammatik, folgt $x' = zx'$, Widerspruch!

3. Fall: In β_1 gibt es ein Nichtterminalsymbol. Dann ist $x' \in \text{sfx}(uy)$, also $uy = u'x'$, so daß $\beta_1 \mid\!\!\!\not\vdash_r^+ u'$, wobei wir u' in $u_0u_1u_2$ mit $u_0u_1 \neq \varepsilon$ zerlegen können, erhalte insgesamt:

   $$S \vdash_r^* \alpha_0' B' x' \vdash_r \alpha_0'\beta_0\beta_1 x' \vdash_r^* \alpha_0'\beta_0 u_0 C u_2 x' \vdash_r \alpha_0'\beta_0 u_0 u_1 u_2 x' = \alpha u' x' = \alpha u y$$

   Dabei gilt $\text{first}_k(x) = u = \text{first}_k(u_0u_1u_2x')$. Da G eine LR (k)-Grammatik ist, erhalten wir $\alpha_0'\beta_0 u_0 = \alpha_0$, $B = C$ und $u_2x' = u_0u_1u_2x'$. Dies impliziert aber insbesondere $u_0u_1 = \varepsilon$ – Widerspruch!

Rückrichtung: Wir nehmen an, wir hätten Ableitungen:

$$\begin{array}{llllll} S & \vdash_r^* & \alpha_0 B\, u & \vdash & \alpha_0\beta\, u & \alpha u \\ S & \vdash_r^* & \alpha_0' B' u' & \vdash & \alpha_0'\beta_0 u' & \alpha v \end{array}$$

mit $w = \text{first}_k(u) = \text{first}_k(v)$ und $|\alpha_0\beta| \leq |\alpha'_0\beta'|$. Außerdem gelte die Bedingung (2), wir konstruieren nun einen Widerspruch.
Zuerst einmal ist $[B \to \beta \cdot \varepsilon, w]$ ein Rechtsdatum für α. Das zweite Rechtsdatum gewinnen wir aus der Rechtsableitung von $\alpha'_0\beta'u'$. In ihr gibt es einen Schritt $\gamma Cy \vdash_r \gamma\delta_0\delta_1 y$ mit $y\delta_0 = \alpha_0\beta$, von dem wir den am weitesten rechts liegenden fixieren. Dann ist $[C \to \delta_0 \cdot \delta_1, w']$ mit $w' = \text{first}_k(y)$ ein Rechtsdatum zu $\alpha_0\beta$.

Wir haben außerdem $\delta_1 y \vdash_r^* v$. Daraus erhalten wir $w = \text{first}_k(v) \in \text{first}_k(\delta_1 y)$. Wir haben sogar $w \in \text{first}_k^{\not\varepsilon}(\delta_1 y)$. Denn wäre die Ableitung $\delta_1 y \vdash_r^* v$ von der Form $\delta_1 y \vdash_r^* Dv \vdash v$, so hätten wir auch $\gamma = \gamma\delta_0$, $C = D$, $y = v$, $\delta_0 = \varepsilon$ und $\delta_1 = v$ wählen können.

AUs der Annahme erhalten wir $B = C$, $\delta_0 = \beta$ und $\delta_1 = \varepsilon$ sowie $w = w'$. Zum einen ist dann $C \to \delta_0\delta_1$ identisch mit $C \to \beta$. Zum anderen wird aus $\delta_1 y \vdash_r^* v$ einfach $y \vdash_r^* v$, also $y = v$. Außerdem haben wir dann $\alpha_0 = \gamma$. Damit erhalten wir

$$S \vdash_r^* \gamma Cv \vdash_r \gamma\beta v = \alpha_0\beta v$$

Daraus können wir $\gamma Cv = \alpha'_0 B'u'$ schließen, was wiederum $\alpha'_0 = \gamma = \alpha_0$, $B' = C = B$, $v = u'$, also $\beta = \beta'$ liefert. □

5.4 LR-Automaten

Konvention: Ab jetzt ist jede Grammatik reduziert und schwach startsepariert.

Wir betrachten den nicht-deterministischen endlichen Automaten $N_k(G)$ ohne Endzustände, der definiert ist durch

$$N_k(G) = (\Sigma \cup \{\$\}, P_k(G), I, \Delta)$$

Dabei sei $I = \{\, [S \to \varepsilon \cdot \alpha, \varepsilon] \mid (S \to \alpha) \in P \}$, und weiter sei

$$\begin{aligned} \Delta = \quad & \{\, ([B \to \beta_0 \cdot C\beta_1, u],\ \varepsilon,\ [C \to \varepsilon \ \cdot \gamma,\ v] \mid C \to \gamma \in G, v \in \text{first}_k(\beta_1 u) \,\} \\ \cup \quad & \{\, ([B \to \beta_0 \cdot a\,\beta_1, u],\ a,\ [B \to \beta_0 a \cdot \beta_1, u] \mid a \in \Sigma \cup \{\$\} \,\} \end{aligned}$$

Lemma: Sei G eine kontextfreie Grammatik und $k \geq 0$. Dann sind die beiden folgenden Aussagen äquivalent für $a \in \Sigma^*$ und $[B \to \beta_0 \cdot \beta_1, u] \in \mathrm{P}_k(G)$:

1. $[B \to \beta_0 \cdot \beta_1, u] \in \mathrm{P}_k(\alpha)$
2. $[B \to \beta_0 \cdot \beta_1, u]$ ist in $N_k(G)$ über α erreichbar.

Beweis: siehe Dozenten-Script, Seite 87f. □

Sei $D_k(G)$ ein zu $N_k(G)$ äquivalenter, deterministischer endliche Automat. Für jedes Wort α sei dann $\delta_G(\alpha)$ der Zustand, der in $D_k(G)$ durch Lesen von α erreicht wird.

Folgerung: Sei G eine kontextfreie Grammatik und $k \geq 0$. Dann gilt für jedes $\alpha \in \Sigma^*$:

$$\mathrm{P}_k(\alpha) = \delta_G(\alpha)$$

Folgerung: Sei G eine kontextfreie Grammatik. Dann sind äquivalent:

1. G ist eine $\mathrm{LR}(k)$-Grammatik.
2. Für jeden in $D_k(G)$ erreichbaren Zustand q gilt: Falls zwei unterschiedliche Rechtsdaten $[B \to \beta \cdot \varepsilon, u]$ und $[B' \to \beta_0 \cdot \beta_1, v] \in \mathrm{P}_k(\alpha)$ sind, so ist $u \notin \mathrm{first}_k^{\notin}(\beta_1 v)$.

Folgerung: Für eine kontextfreie Grammatik läßt sich entscheiden, ob eine gegebene Satzform ein Rechtspräfix ist.

Folgerung: Es ist entscheidbar, ob eine gegebene kontextfreie Grammatik eine $\mathrm{LR}(k)$-Grammatik ist.

Sei nun G eine $\mathrm{LR}(k)$-Grammatik. Wir konstruieren einen deterministischen Kellerautomaten mit Vorausschau k, regulärer Kontrolle und Ausgabe, der $L(G)$ erkennt. Diesen nennen wir den $\mathrm{LR}(k)$-Automaten für G (bzw. L) und bezeichnen ihn mit $\mathrm{A}_k^{\mathrm{LR}}(G)$.

Der Kellerstartinhalt ist ein Startsymbol c_I. Die reguläre Kontrolle ist gegeben durch $\mathrm{A}_k(G)$, der wie folgt abgewandelt wird: Für jedes Wort, das nicht mit c_I beginnt, geht der Automat in einen neuen Zustand q_f. Für jeden Zustand q und jede Vorausschau $u \in A^{\leq k}$ sind die Transitionen wie folgt gegeben:

- *Reduce-Transitionen:* Falls $[B \to \beta \cdot \varepsilon, u] \in q$, so haben wir eine Reduce-Transition
$$(q, u, \beta, \varepsilon, B, B \to \beta)$$
- *Shift-Transitionen:* Falls $[B \to \beta_0 \cdot \beta_1, v] \in q$ mit $v \in \mathrm{first}_k^{\notin}(\beta_1 u)$, so haben wir für jedes $a \in A$ eine Shift-Transition
$$(q, u, \varepsilon, a, \varepsilon, \varepsilon)$$
- *Terminaltransitionen:* Zusätzlich gibt es noch die Terminaltransition
$$(q, \varepsilon, c_I S, \varepsilon, \varepsilon, \varepsilon)$$

Satz: Sei G eine LR (k)-Grammatik. Dann ist $\mathrm{A}_k^{\mathrm{LR}}(G)$ ein deterministischer Kellerautomat, der äquivalent zu G ist und für jedes Wort, das er akzeptiert, eine umgedrehte Rechtsableitung ausgibt.

Beweis: Daß $\mathrm{A}_k^{\mathrm{LR}}(G)$ deterministisch ist, folgt aus dem Satz über Rechtsdaten. Man zeigt leicht per Induktion folgende Behauptung für alle $uv \in A^*$: Falls $(c_I, uv) \to^* (c_I\alpha, v)$, so gilt $\alpha v \vdash_r^* u$. Gleichzeitig wird dabei eine entsprechende Rechtsableitung in umgekehrter Reihenfolge ausgegeben.

Da bei der Berechnung auf einem Wort der Form u mindestens eine Transition ausgeführt werden muß, um den Keller zu leeren, kann jede akzeptierende Berechnung nur enden mit der Terminaltransition. Aus der obigen Behauptung können wir dann aber $u \in L(G)$ schließen.

Für die andere Richtung behaupten wir. Es sei $S \vdash_r^* \alpha_0 Bu \vdash_r \alpha_0\beta u$ eine Rechtsableitung und $\alpha_0\beta = \gamma v$. Dann gibt es eine akzeptierende Berechnung ausgehend von $(c_I\gamma, vu)$. Wir beweisen diese Behauptung per Induktion über die Länge der Rechtsableitung und danach $|\alpha_0\beta| - |\gamma|$.

- Induktionsanfang: Wir betrachten die Ableitung $S \vdash_r \beta$ und wir wollen zeigen, daß es ausgehend von $(c_I\beta, \varepsilon)$ eine akzeptierende Berechnung gibt. Da G eine LR (k)-Grammatik ist, haben wir $[S \to \beta\cdot\varepsilon, \varepsilon] \in \mathrm{P}_k(\beta)$, so daß wir auch $(c_I\beta, \varepsilon) \to (c_I S, \varepsilon)$ haben. Die Terminaltransition führt dann zum Akzeptieren.

- Induktionsanfang: Wir unterscheiden zwei Fälle:

  - Falls $|\alpha_0\beta| = |\gamma|$, so können wir wie beim Induktionsanfang vorgehen und feststellen, daß zuerst einmal $(c_I\alpha_0\beta, u) \to (c_I\alpha_0 B, u)$ gilt. Außerdem haben wir $(c_I\alpha_0 B, u) \to^* (c_I, \varepsilon)$ nach Induktionsannahme, was ausreicht.
  - Falls $|\alpha_0\beta| > |\gamma|$, so folgt aus der Annahme, daß G eine LR (k)-Grammatik ist, daß keine Reduce-Transition anwendbar ist, da an einer späteren Stelle eine Reduktion durchgeführt werden kann. Desweiteren ist deshalb $\mathrm{P}_k(\gamma) \neq \emptyset$, was bedeutet, daß wir mit $v = av'$ nun auch $(c_I\gamma, vu) \to (c_I\gamma a, v'u)$ haben. Wir können also die Induktionsvoraussetzung anwenden, um eine akzeptierende Berechnung zu konstruieren. □

Bemerkungen zu „echten“ Parsern bzw. Parsergeneratoren (YACC etc.):

- Die Parser vermischen die reguläre Kontrolle mit dem eigentlichen Kellerautomaten.

- Parsergeneratoren setzen in der Regel $k = 1$. Die vorgeschaltete lexikalische Analyse faßt jedoch schon einzelne Buchstaben zu Tokens zusammen, so daß ein Token Vorausschau ausreicht.
- In der Praxis werden nach der Analyse häufig Optimierungen (z.B. Zustandsreduktion in der regulären Kontrolle) durchgeführt, YACC erzeugt beispielsweise $LALR(1)$-Automaten.

Beispiel: Betrachte die folgende Grammatik:

$$S \to aA \,|\, bB \quad A \to cAd \,|\, \varepsilon \quad B \to cBd \,|\, \varepsilon$$

Die erzeugte Sprache ist $L = \{\, (a+b)c^n d^n \mid n \in \mathbb{N}\}$.

5.5 Vorführung: YACC

YACC angeworfen auf eine Grammatik liefert einen Parser für die Sprache – entsprechende Tools gibt es in zahlreichen Varianten für zahlreiche Programmiersprachen (u.a. JCup für Java und Happy für Haskell).

5.6 LR (1)-Sprachen

Bemerkung: Jede Sprache, die von einer LR (0)-Grammatik erkannt wird, ist präfixfrei, d.h. aus $u \in L$ folgt $uv \notin L$ für alle $v \in A^+$. Damit kann nicht zu jeder deterministischen Sprache eine äquivalente LR (0)-Grammatik gefunden werden; aber:

Satz:

(1) Zu jeder strikt deterministischen Sprache gibt es eine äquivalente LR (0)-Grammatik.

(2) Zu jeder deterministischen Sprache gibt es eine äquivalente LR (1)-Grammatik.

Beweis von (1): Sei zur strikt deterministischen Sprache L ein Automat $\mathcal{A} = (A, C, Q, c_I, q_I, \Delta)$ in schwacher Normalform gegeben, der den Anfangszustand nur am Anfang annimmt. Nun konstruieren wir eine äquivalente kontextfreie Grammatik G mit folgenden Regeln:

1. $S \to a$ für alle $(q_I, c_I, a, \varepsilon, q) \in \Delta$
2. $S \to [q_I, q]a$ für alle $(q, c_I, a, \varepsilon, q') \in \Delta$

3. $[q_I, q] \to q$ für $(q_I, c_I, a, c_I, q) \in \Delta$
4. $[q_I, q] \to \langle q_I, c_I c, q' \rangle$ für $(q', c, a, \varepsilon, q) \in \Delta$
5. $\langle q, cc', q' \rangle \to a$ für $(q, c, a, cc', q') \in \Delta$
6. $\langle q_I, c_I c, q' \rangle \to [q_I, q'']a$ für $(a'', c_I, a, c_I c, q') \in \Delta$
7. $\langle q, cc', q' \rangle \to \langle q, cc', q'' \rangle\, a$ für $(q'', c', a, c', q') \in \Delta$
8. $\langle q, cc', q' \rangle \to \langle q, cc', q'' \rangle\, \langle q'', c'c'', q''' \rangle\, a$ für $(q''', c'', a, \varepsilon, q') \in \Delta$

Beispiel: Betrachte wieder die Beispielsprache von oben:

$$a^n b^* 0 c^n \$ + a^* b^n 1 c^n$$

Der zugehörige Automat:

$$\begin{array}{llllll}
(q_a, & c_I, & a, & c_I a, & q_a) \\
(q_a, & c_I, & b, & c_I b, & q_b) \\
\\
(q_a, & a, & a, & aa, & q_a) \\
(q_a, & a, & b, & ab, & q_b) \\
(q_b, & b, & b, & bb, & q_b) \\
\\
(q_a, & a, & 0, & a, & q_0) \\
(q_b, & b, & 0, & b, & q_0) \\
(q_a, & a, & 1, & a, & q_1) \\
(q_b, & b, & 1, & b, & q_1) \\
\\
(q_0, & b, & \varepsilon, & \varepsilon, & q_0) \\
(q_0, & a, & \varepsilon, & a, & q_c) \\
(q_0, & c_I, & \varepsilon, & c_I, & q_c)
\end{array}$$

Sei das Wort $x = aaab0ccc\$$ gegeben.

	erzeugter Automat		**sim. Automat**	
Schritt	**Keller**	**Rest-Eingabe**	**Keller**	**Zustand**
	ε	$aaab0ccc\$$	c_I	q_a
shift	a	$aab0ccc\$$		
reduce	$\langle q_a, c_I a, q_a \rangle$	$aab0ccc\$$	$c_I a$	q_a
shift	$\langle q_a, c_I a, q_a \rangle\, a$	$ab0ccc\$$		
reduce	$\langle q_a, c_I a, q_a \rangle\, \langle q_a, aa, q_a \rangle$	$ab0ccc\$$	$c_I aa$	q_a
shift	$\langle q_a, c_I a, q_a \rangle\, \langle q_a, aa, q_a \rangle\, a$	$b0ccc\$$		
reduce	$\langle q_a, c_I a, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, aa, q_a \rangle$	$b0ccc\$$	$c_I aaa$	q_a
shift	$\langle q_a, c_I a, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, aa, q_a \rangle\, b$	$0ccc\$$		
reduce	$\langle q_a, c_I a, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, ab, q_b \rangle$	$0ccc\$$	$c_I aaab$	q_b
shift	$\langle q_a, c_I a, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, ab, q_b \rangle\, 0$	$ccc\$$		
reduce	$\langle q_a, c_I a, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, ab, q_0 \rangle$	$ccc\$$	$c_I aaab$	q_0
reduce	$\langle q_a, c_I a, q_a \rangle\, \langle q_a, aa, q_a \rangle\, \langle q_a, aa, q_0 \rangle$	$ccc\$$	$c_I aaa$	q_0
...	..	...	...	...

Beweis von Teil (1) in mehreren Schritten:

1. Zeige: $[q_I, q] \vdash_r^+ u$ genau dann, wenn $(q_I, u, c_I) \to^+ (q', \varepsilon, c_I)\ \langle q, cc', q' \rangle \vdash_r^* u$ genau dann, wenn $(q, u, c) \to^* (q', \varepsilon, cc')$

2. Es gibt jeweils eine Rechtsableitung mit Satzformen α der folgenden Form:

   - $\alpha = S$
   - $\alpha = [q_I, q]u$
   - $\alpha = \langle q_0, c_0c_1, q_1 \rangle \cdots \langle q_{r-1}, c_{r-1}c_rq_r \rangle\ u$ mit $q_0 = I$ und $c_0 = c_I$

3. Zeige: Die Grammatik ist eindeutig, d.h. es gibt immer nur eine Rechtsableitung.

   Folgerung: Alle Rechtssatzformen, d.h. alle Rechtspräfixe, sind von obiger Form.

4. Zeige, daß G eine LR (0)-Grammatik ist mit Rückgriff auf die Definition.

Beweis von Teil (2): Wir betrachten beliebige LR (0)-Grammatik G für $L\$$. Eine solche gibt es nach Satz 1. Konstruiere Grammatik G' für L mit Regeln $P' = P_0 \cup P_\$$ mit $P_0 = \{ B \to \beta \in P \mid \$ \notin \beta \}$ $P_\$ = \{ B \to \beta \mid B \to \beta\$ \in P \}$. Zeige dann mit Rückgriff auf die Definition, daß G' eine LR (1)-Grammatik für L ist.