Berechenbarkeits- und Komplexitätstheorie

Lerneinheit 2: Entscheidbarkeit

Prof. Dr. Christoph Karg

Studiengang Informatik
Hochschule Aalen

Hochschule Aalen

Wintersemester 2015/2016

Einleitung

Ziel dieser Lerneinheit ist das Ausloten der Grenzen der Informatik.

Folgende Fragen werden diskutiert:

- Was versteht man unter Entscheidbarkeit?
- Gibt es Entscheidungsprobleme, die mit Turing Maschinen nicht lösbar sind?
- Gibt es Techniken zur Analyse der Entscheidbarkeit von Entscheidungsproblemen?

Entscheidbarkeit

Definition. Sei L eine Sprache über dem Alphabet Σ.

- L ist ist entscheidbar (rekursiv), falls es eine Turing Maschine gibt, die L akzeptiert und die auf allen Eingaben hält
- L ist ist semi-entscheidbar (rekursiv aufzählbar), falls es eine Turing Maschine gibt, die L akzeptiert
- Ansonsten ist L unentscheidbar

Ein wichtiger Satz

Satz. Ist L semi-entscheidbar und $\overline{L}$ semi-entscheidbar, dann ist L entscheidbar.

Beweis. Seien $M = (Z, \Gamma, \Sigma, \sqcup, \delta, z_s, z_{acc}, z_{rej})$ und $M' = (Z', \Gamma, \Sigma, \sqcup, \delta', z_s', z_{acc}', z_{rej}')$ (deterministische) Turing Maschinen mit $L(M) = L$ bzw. $L(M') = \overline{L}$.

Der Einfachheit halber sei angenommen, dass jede haltende Berechnung von M bzw. M' im akzeptierenden oder verwerfenden Zustand stoppt

Idee: Simuliere M und M' schrittweise, bis eine der beiden Turing Maschinen akzeptiert

Ein wichtiger Satz (Forts.)

$\textsc{Simulate}_{(M,M')}(x)$

Input: Wort $x = a_1 \dots a_n \in \Sigma^*$

Output: `true`, falls $x \in L$, `false` sonst.

$C := (\llcorner, z_s, a_1 \dots a_n)$; $C' := (\llcorner, z'_s, a_1 \dots a_n)$;
$\mathit{result} := \mathit{unknown}$
while $\mathit{result} = \mathit{unknown}$ **do**
  if C ist eine akzeptierende Konfiguration **then**
    $\mathit{result} :=$ `true`
  elseif C' ist eine akzeptierende Konfiguration **then**
    $\mathit{result} :=$ `false`
  else
    $C :=$ Folgekonfiguration von C bzgl. M
    $C' :=$ Folgekonfiguration von C' bzgl. M'
return result

Ein wichtiger Satz (Forts.)

Korrektheit: Sei $x \in \Sigma^*$.

- Fall 1: $x \in L$.
  - ▷ Dann existiert eine akzeptierende Berechnung von M auf Eingabe x.
  - ▷ Somit durchläuft M ausgehend von der Startkonfiguration $(\sqcup, z_s, a_1 \ldots a_n)$ eine Folge von Konfigurationen, die in einer akzeptierenden Konfiguration endet
  - ▷ Diese Konfigurationsfolge wird von $\textsc{Simulate}_{(M,M')}(x)$ gefunden
  - ▷ Folglich gibt $\textsc{Simulate}_{(M,M')}(x)$ `true` zurück
- Fall 2: $x \notin L$. analog.

Ergebnis: $\textsc{Simulate}_{(M,M')}(x) = \mathit{true}$ genau dann, wenn $x \in L$

Ein wichtiger Satz (Forts.)

Konsequenz. Ist L semi-entscheidbar, aber nicht entscheidbar, dann ist $\overline{L}$ unentscheidbar.

Beweis. Wäre $\overline{L}$ semi-entscheidbar, dann wäre L entscheidbar. Dies ist ein Widerspruch zur Annahme.

Bemerkung: Gelingt der Nachweis der Existenz eines semi-entscheidbaren, aber nicht entscheidbaren Problems, dann hat man automatisch ein unentscheidbares Problem gefunden.

Kodierung von Berechnungsmodellen

Aufgabe: Verarbeitung von DFAs, NFAs, PDAs und Turing Maschinen durch eine Turing Maschine

Voraussetzung: geeignete Kodierung der Automaten

Ansatz: Einsatz eines einheitlichen Kodierungsalphabets

Notation:

- $\langle M \rangle$ steht für die Kodierung des Automaten M
- $\langle M, x \rangle$ steht für die Kodierung des Automaten M und einer zu verarbeitenden Eingabe $x \in \Sigma^*$

Beispiel: Kodierung von NFAs

Gegeben:

- NFA $M = (Z, \Sigma, \delta, z_s, A)$
- Wort $x \in \Sigma^*$

Ziel: Kodierung von M und x über dem Alphabet

$$\Delta = \{\mathtt{z}, \mathtt{b}, \mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}, \#, \mathtt{[}, \mathtt{]}, \mathtt{(}, \mathtt{)}, \mathtt{/}, \mathtt{,}, \mathtt{:}, \mathtt{\{}, \mathtt{\}}\}$$

Beachte: der Inhalt von von Z und Σ ist variabel und muss daher entsprechend dargestellt werden

Beispiel: Kodierung von NFAs (Forts.)

Schritt 1: Kodierung der Zustände

1. Falls $\|Z\| = k$, dann weise jedem Zustand genau einen Index i aus $\{0, 1, \ldots, k-1\}$ zu
2. Definiere die Kodierung des Zustands z_i mit dem Index i als

   $$\langle z_i \rangle = \mathtt{z}i$$

   wobei i als Dezimalzahl ohne führende Nullen dargestellt wird.
3. Definiere die Kodierung der Zustandsmenge Z als

   $$\langle Z \rangle = \{\langle z_0 \rangle, \ldots, \langle z_{k-1} \rangle\}$$

4. Definiere die Kodierung $\langle A \rangle$ der Menge der akzeptierenden Zustände A analog zu $\langle Z \rangle$.

Beispiel: Kodierung von NFAs (Forts.)

Schritt 2: Kodierung des Alphabets

1. Falls $\|\Sigma\| = \ell$, dann weise jedem Buchstaben genau einen Index i aus $\{0, 1, \ldots, \ell - 1\}$ zu
2. Definiere die Kodierung des Buchstabens b_i mit dem Index i als

$$\langle b_i \rangle = \mathtt{b}i$$

   wobei i als Dezimalzahl ohne führende Nullen dargestellt wird.
3. Definiere die Kodierung der Zustandsmenge Σ als

$$\langle \Sigma \rangle = \{\langle b_0 \rangle\, , \ldots\, , \langle b_{\ell-1} \rangle\}$$

Beispiel: Kodierung von NFAs (Forts.)

Schritt 3: Kodierung der Überführungsfunktion $\delta : Z \times \Sigma \mapsto P(Z)$

1. Die Kodierung berücksichtigt ausschließlich $z \in Z$ und $b \in \Sigma$ mit $\delta(z, b) = \{z_1, \ldots, z_n\}$, $n \geq 1$.

   In diesem Fall ist:

$$\langle\delta(z, b)\rangle = [(\langle z\rangle, \langle b\rangle) : \langle z_1\rangle / \ldots / \langle z_n\rangle]$$

   Anderfalls ist $\langle\delta(z, a)\rangle = \varepsilon$

2. Die Kodierung von δ besetzt in der Hintereinanderreihung der einzelnen Einträge:

$$\langle\delta\rangle = \langle\delta(z_0, b_0)\rangle\langle\delta(z_0, b_1)\rangle \ldots \langle\delta(z_i, b_j)\rangle \ldots \\ \ldots \langle\delta(z_{k-1}, b_{\ell-2})\rangle\langle\delta(z_{k-1}, b_{\ell-1})\rangle$$

Beispiel: Kodierung von NFAs (Forts.)

Schritt 4: Kodierung des NFAs

1. Die Kodierung von M ist definiert als:

$$\langle M \rangle = (\langle Z \rangle, \langle \Sigma \rangle, \langle \delta \rangle, \langle z_s \rangle, \langle A \rangle)$$

2. Die Kodierung der Eingabe $x = a_1 \dots a_m$ ist definiert als:

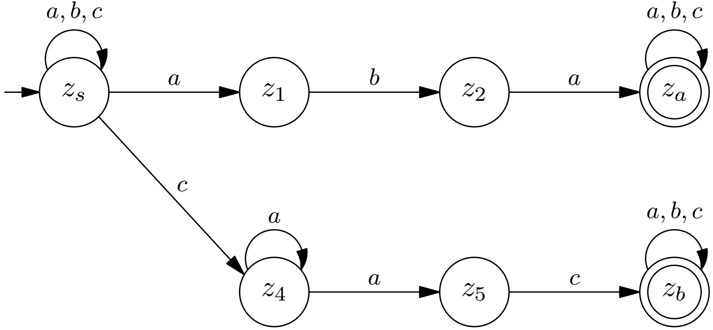
$$\langle x \rangle = (\langle a_1 \rangle, \dots, \langle a_m \rangle)$$

3. Gesamtkodierung:

$$\langle M, x \rangle = \langle M \rangle \# \langle x \rangle$$

Beispiel: Kodierung von NFAs (Forts.)

Beispiel: NFA M, Eingabe $x = aaabac$



Beispiel: Kodierung von NFAs (Forts.)

Indizierung der Zustände:

z	z_s	z_1	z_2	z_a	z_4	z_5	z_b
i	0	1	2	3	4	5	6

Kodierung der Zustände:

z	z_s	z_1	z_2	z_a	z_4	z_5	z_b
$\langle z \rangle$	z0	z1	z2	z3	z4	z5	z6

Kodierung der Zustandsmenge:

$$\langle Z \rangle = \{\texttt{z0,z1,z2,z3,z4,z5,z6}\}$$

Beispiel: Kodierung von NFAs (Forts.)

Indizierung der Buchstaben:

x	a	b	c
i	0	1	2

Kodierung der Buchstaben:

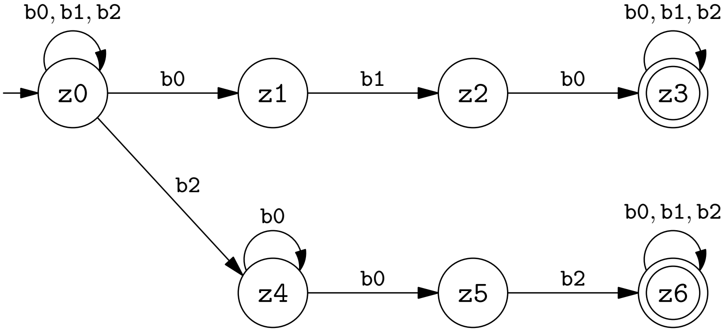
x	a	b	c
$\langle x \rangle$	`b0`	`b1`	`b2`

Kodierung des Alphabets:

$$\langle \Sigma \rangle = \{\texttt{b0,b1,b2}\}$$

Beispiel: Kodierung von NFAs (Forts.)

NFA mit angepasster Beschriftung:



Beispiel: Kodierung von NFAs (Forts.)

Kodierung der Überführungsfunktion:

$$
\begin{array}{lll}
\delta(z_s, a) = \{z_s, z_1\} & \rightsquigarrow & \texttt{[(z0,b0):z0/z1]} \\
\delta(z_s, b) = \{z_s\} & \rightsquigarrow & \texttt{[(z0,b1):z0]} \\
\delta(z_s, c) = \{z_s, z_4\} & \rightsquigarrow & \texttt{[(z0,b2):z0/z4]} \\
\delta(z_1, b) = \{z_1\} & \rightsquigarrow & \texttt{[(z1,b2):z1]} \\
\delta(z_2, a) = \{z_3\} & \rightsquigarrow & \texttt{[(z2,b0):z3]} \\
\delta(z_a, a) = \{z_a\} & \rightsquigarrow & \texttt{[(z3,b0):z3]} \\
\delta(z_a, b) = \{z_a\} & \rightsquigarrow & \texttt{[(z3,b1):z3]} \\
\delta(z_a, c) = \{z_a\} & \rightsquigarrow & \texttt{[(z3,b2):z3]} \\
\delta(z_4, a) = \{z_4, z_5\} & \rightsquigarrow & \texttt{[(z4,b0):z4/z5]} \\
\delta(z_5, c) = \{z_b\} & \rightsquigarrow & \texttt{[(z5,b2):z6]} \\
\delta(z_b, a) = \{z_b\} & \rightsquigarrow & \texttt{[(z6,b0):z6]} \\
\delta(z_b, b) = \{z_b\} & \rightsquigarrow & \texttt{[(z6,b1):z6]} \\
\delta(z_b, c) = \{z_b\} & \rightsquigarrow & \texttt{[(z6,b2):z6]}
\end{array}
$$

Beispiel: Kodierung von NFAs (Forts.)

Kodierung von M und x: $\langle M, x \rangle =$

```
({z0,z1,z2,z3,z4,z5,z6},{b0,b1,b2},
[(z0,b0):z0/z1][(z0,b1):z0][(z0,b2):z0/z4]
[(z1,b2):z1][(z2,b0):z3][(z3,b0):z3][(z3,b1):z3]
[(z3,b2):z3][(z4,b0):z4/z5][(z5,b2):z6]
[(z6,b0):z6][(z6,b1):z6][(z6,b2):z6],
z0,{z3,z6})#(b0,b0,b0,b1,b0,b2)
```

Bemerkungen:

- Die Korrektheit der Kodierung ist mit einer Turing Maschine überprüfbar
- Die Kodierung kann man algorithmisch weiter verarbeiten
- Für DFAs, Kellerautomaten und Turing Maschinen existieren entsprechende Kodierungen

NFA-Kodierungen

Kodierung von NFAs

- Gegeben: Wort $x \in \Delta^*$
- Gefragt: Ist x die Kodierung eines NFAs M?

Zugehörige Sprache:

$$K_{NFA} = \{x \in \Delta^* \mid \text{es gibt einen NFA } M \text{ mit } \langle M \rangle = x\}$$

Analog:

- $K_{DFA} \leadsto$ Kodierung von DFAs
- $K_{PDA} \leadsto$ Kodierung von Kellerautomaten
- $K_{TM} \leadsto$ Kodierung von Einband Turing Maschinen
- Kodierung von weiteren Berechnungsmodellen, Grammatiken, ...

NFA-Kodierungsproblem (Forts.)

Satz. K_{NFA} ist entscheidbar

Beweis. Die Überprüfung eines Wortes $x \in \Delta^*$ erfolgt in zwei Schritten:

1. Überprüfung der syntaktischen Korrektheit
2. Überprüfung der semantischen Korrektheit
   - ▷ Sind die benutzten Zustände konsistent?
   - ▷ Sind die benutzten Buchstaben konsistent?

Die Überprüfung ist mittels einem geeigneten Algorithmus (Parser) effizient durchführbar

Das Wortproblem für DFAs

Wortproblem für DFAs:

- Gegeben: Kodierung $\langle M, x \rangle$ eines DFAs M und einer Eingabe x
- Gefragt: Akzeptiert M die Eingabe x?

Zugehörige Sprache:

$$L_{DFA} = \{\langle M, x \rangle \in \Delta^* \mid \text{der DFA } M \text{ akzeptiert } x\}$$

Satz. Die Sprache L_{DFA} ist entscheidbar.

Das Wortproblem für DFAs (Forts.)

$\textsc{SimulateDFA}(\langle M, x\rangle)$

Input: Kodierung $\langle M, x\rangle$ eines DFAs $M = (Z, \Sigma, \delta, z_s, A)$
und eines Worts $x = a_1 \ldots a_n \in \Sigma^*$

Output: `true`, falls $x \in L(M)$, `false` sonst.

$state := z_s$; $i := 1$;
while $(i \leq n)$ **do**
  if $\delta(state, a_i)$ *ist definiert* **then**
    $state := \delta(state, a_i)$; $i := i + 1$;
  else
    return `false`
if $state \in A$ **then**
  return `true`
else
  return `false`

Das Wortproblem für DFAs (Forts.)

Beweis. Betrachte den Algorithmus $\textsc{SimulateDFA}(\langle M, x\rangle)$.

- Der Algorithmus stoppt auf allen Eingaben
- Der Algorithmus verwirft alle Eingaben, die keine korrekte Kodierung eines DFAs und einer Eingabe darstellen
- Falls $x \in L(M)$, dann gibt es eine Zustandsfolge $z_0, \dots, z_n$ mit
  - ▷ $z_s = z_0$
  - ▷ Für alle $i = 1, \dots, n$ gilt: $z_i = \delta(z_{i-1}, a_i)$
  - ▷ $z_n \in A$

  In diesem Fall durchläuft die **while** Schleife von $\textsc{SimulateDFA}(\langle M, x\rangle)$ exakt diese Zustandsfolge und akzeptiert die Eingabe

Das Wortproblem für DFAs (Forts.)

- Falls $x \notin L(M)$, dann verwirft der Algorithmus die Eingabe, da der DFA M auf Eingabe x keine akzeptierende Berechnung durchläuft
- Somit gilt:

$$\textsc{SimulateDFA}(\langle M, x\rangle) = \begin{cases} \texttt{true} & M \text{ akzeptiert } x, \\ \texttt{false} & \text{sonst.} \end{cases}$$

  d.h., die von dem Algorithmus akzeptierte Sprache ist L_{DFA}

Fazit: L_{DFA} ist entscheidbar

Das Wortproblem für NFAs

Wortproblem für NFAs:

- Gegeben: Kodierung $\langle M, x \rangle$ eines NFAs M und einer Eingabe x
- Gefragt: Akzeptiert M die Eingabe x?

Zugehörige Sprache:

$$L_{NFA} = \{\langle M, x \rangle \in \Delta^* \mid \text{der NFA } M \text{ akzeptiert } x\}$$

Satz. Die Sprache L_{NFA} ist entscheidbar.

Das Wortproblem für NFAs (Forts.)

$\textsc{SimulateNFA}(\langle M, x\rangle)$

Input: Kodierung $\langle M, x\rangle$ eines NFAs $M = (Z, \Sigma, \delta, z_s, A)$
und eines Worts $x = a_1 \dots a_n \in \Sigma^*$

Output: `true`, falls $x \in L(M)$, `false` sonst.

Konvertiere mit der Potenzmengenkonstruktion den NFA M in einen äquivalenten DFA M' ;
if $\textsc{SimulateDFA}(\langle M', x\rangle) =$ `true` **then**
    return `true`
else
    return `false`

Das Wortproblem für NFAs (Forts.)

Beweis.

- Automatentheorie: für jeden NFA M existiert ein DFA M' mit $L(M) = L(M')$
- Mit der Potenzmengenkonstruktion kann M' berechnet werden
- Es gilt:

$$\begin{aligned} & \text{Der Algorithmus } \textsc{SimulateNFA}(\langle M, x\rangle) \text{ akzeptiert} \\ \Leftrightarrow \quad & x \in L(M') \\ \Leftrightarrow \quad & x \in L(M) \end{aligned}$$

- $\textsc{SimulateNFA}(\langle M, x\rangle)$ stoppt auf allen Eingaben

Fazit: L_{NFA} ist entscheidbar

Leerheitstest für DFAs

Leerheitsproblem für DFAs:

- Gegeben: Kodierung $\langle M \rangle$ eines DFAs M
- Gefragt: Ist $L(M) = \emptyset$?

Zugehörige Sprache:

$$E_{DFA} = \{\langle M \rangle \mid M \text{ ist ein DFA mit } L(M) = \emptyset\}$$

Satz. E_{DFA} ist entscheidbar.

Beweis. Beobachtung: Ein DFA akzeptiert mindestens eine Eingabe, wenn es einen Endzustand gibt, der vom Startzustand aus erreichbar ist.

Leerheitstest für DFAs (Forts.)

EMPTYDFA($\langle M \rangle$)

Input: Kodierung $\langle M \rangle$ eines DFAs $M = (Z, \Sigma, \delta, z_s, A)$

Output: `true`, falls $L(M) = \emptyset$, `false` sonst.

$S := \{z_s\}$; *update* := `true`;
while *update* := `true` **and** $S \cap A = \emptyset$ **do**
  update := `false`;
  for jeden Zustand $z \in S$ **do**
    for jeden Buchstaben $b \in \Sigma$ **do**
      if $\delta(z, b) \notin S$ **then**
        $S := S \cup \{\delta(z, b)\}$; *update* := `true`;
if $S \cap A = \emptyset$ **then**
  return `true`
else
  return `false`

Leerheitstest für DFAs (Forts.)

- Die Menge S speichert die bereits gefundenen Zustände
- Die beiden **for** Schleife dienen zur Suche von unentdeckten Zuständen
- Die **while** Schleife wird höchstens $\|Z\| - 1$-mal durchlaufen
- Nach Beendigung der **while** Schleife gilt:
  - Alle Zustände in S sind vom Startzustand z_s aus erreichbar
  - Ist $S \cap A = \emptyset$, dann ist $L(M) = \emptyset$. Andernfalls enthält $L(M)$ mindestens ein Wort

Äquivalenzproblem für DFAs

Äquivalenzproblem für DFAs:

- Gegeben: Kodierung $\langle M, .M_2 \rangle$ zweier DFAs M_1 und M_2
- Gefragt: Ist $L(M_1) = L(M_2)$?

Zugehörige Sprache:

$$EQ_{DFA} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ und } M_2 \text{ sind DFAs mit } L(M_1) = L(M_2)\}$$

Satz. EQ_{DFA} ist entscheidbar.

Äquivalenzproblem für DFAs (Forts.)

$\textsc{DFAEquivalence}(\langle M_1, M_2 \rangle)$

Input: Kodierung $\langle M_1, M_2 \rangle$ zweier DFAs M_1 und M_2

Output: `true`, falls $L(M_1) = L(M_2)$, `false` sonst.

Konstruiere einen DFA M_1^* *mit* $L(M_1^*) = \overline{L(M_1)}$

Konstruiere einen DFA M_2^* *mit* $L(M_2^*) = \overline{L(M_2)}$

Konstruiere einen DFA M *mit*

$$L(M) = (L(M_1) \cap L(M_2^*)) \cup (L(M_2) \cap L(M_1^*))$$

if $\langle M \rangle \in E_{DFA}$ **then**

  return `true`

else

  return `false`

Äquivalenzproblem für DFAs (Forts.)

- Alle DFA-Konstruktionen in obigem Algorithmus sind erlaubt, da reguläre Sprachen abgeschlossen sind unter Komplement, Schnitt und Vereinigung
- Alle DFA-Konstruktionen sind in endlicher Zeit berechenbar
- Die symmetrische Differenz zweier Sprachen L_1 und L_2 ist definiert als $L_1 \triangle L_2 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$
- $L_1 \triangle L_2$ enthält alle Wörter die entweder in L_1 oder in L_2 sind
- $L_1 = L_2$ genau dann, wenn $L_1 \triangle L_2 = \emptyset$
- $\mathrm{DFAE{\scriptsize QUIVALENCE}}(\langle M_1, M_2 \rangle)$ akzeptiert genau dann, wenn $L(M_1) \triangleleft L(M_2) = \emptyset$

Wortproblem für kontextsensitive Grammatiken

Wortproblem für kontextsensitive Grammatiken:

- Gegeben: Kodierung $\langle G, x \rangle$ einer kontextsensitiven Grammatik $G = (V, \Sigma, V, S)$ und eines Worts $x \in \Sigma^*$
- Gefragt: Ist $x \in L(G)$? Oder: $S \Rightarrow_G^* x$?

Zugehörige Sprache:

$$L_{CSL} = \left\{ \langle G, x \rangle \;\middle|\; \begin{array}{l} G \text{ ist eine kontextsensitive} \\ \text{Grammatik und } x \in L(G) \end{array} \right\}$$

Satz. L_{CSL} ist entscheidbar.

Wortproblem für kontextsensitive Gram. (Forts.)

Beweis. Sei $G = (V, \Sigma, V, S)$ eine kontextsensitive Grammatik

Zur Info: Die Regeln von G sind nicht verkürzend, d.h., für alle $u \to v \in P$ gilt: $|u| \le |v|$

Definiere

$$T_m^n = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} \text{Satzform } w \text{ mit } |w| \le n \text{ und} \\ \text{der Eigenschaft, dass } w \text{ aus } S \\ \text{in } \le m \text{ Schritten ableitbar} \end{array} \right\}$$

Definiere für eine Menge X von Satzformen:

$$Abl^n(X) = \left\{ w \in (V \cup \Sigma)^+ \;\middle|\; \begin{array}{l} |w| \le n \text{ und es gibt eine} \\ \text{Satzform } u \in X \text{ mit } u \Rightarrow_G w \end{array} \right\}$$

Wortproblem für kontextsensitive Gram. (Forts.)

Es gilt:

- $T_i^n \subseteq T_{i+1}^n$ für alle $i = 0, 1, 2, \ldots$
- $\bigcup_{i=0}^{\infty} T_i^n$ ist eine endliche Menge, da eine kontextsensitive Grammatik keine verkürzenden Regeln enthält
- Es gibt ein m, so dass $T_m^n = T_{m+1}^n = T_{m+2}^n = T_{m+3}^n = \ldots$
- $x \in L(G)$ genau dann, wenn $x \in \bigcup_{i=0}^{\infty} T_i^n$
- Es gilt: $T_{m+1}^n = T_m^n \cup Abl_G^n(T_m^n)$

Ansatz: Berechne $\bigcup_{i=0}^{\infty} T_i^{|x|}$ und überprüfe, ob x in dieser Menge ist

Wortproblem für kontextsensitive Gram. (Forts.)

$\textsc{CSLWord}(\langle G, x\rangle)$

Input: Kodierung $\langle G, x\rangle$ einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ und $x \in \Sigma^*$

Output: `true`, falls $x \in L(G)$, `false` sonst.

$T' := \{S\}$; $n := |x|$;
repeat
  $T := T'$;
  $T' := T \cup Abl^n(T)$;
until $(T = T')$ **or** $(x \in T')$
if $(x \in T')$ **then**
  return `true`
else
  return `false`

Wortproblem für kontextsensitive Gram. (Forts.)

Beispiel: Betrachte die folgende kontextsensitive Grammatik

$$
\begin{array}{rcl}
S & \rightarrow & CB \mid CDB \\
D & \rightarrow & AB \mid ADB \\
C & \rightarrow & a \\
aA & \rightarrow & aa \\
aB & \rightarrow & ab \\
bB & \rightarrow & bb
\end{array}
$$

Es gilt: $L(G) = \{a^n b^n \mid n \geq 1\}$

Wortproblem für kontextsensitive Gram. (Forts.)

Betrachte $x = aabb$:

$$\begin{aligned}
T_0^4 &= \{S\} \\
T_1^4 &= T_0^4 \cup \{CB, CDB\} \\
T_2^4 &= T_1^4 \cup \{aB, aDB, CABB, \cancel{CADBB}\} \\
T_3^4 &= T_2^4 \cup \{ab, aABB\} \\
T_4^4 &= T_3^4 \cup \{aaBB\} \\
T_5^4 &= T_4^4 \cup \{aabB\} \\
T_6^4 &= T_5^4 \cup \{aabb\}
\end{aligned}$$

Ergebnis: Da $x \in T_6^4$, folgt $x \in L(G)$.

Das Halteproblem

Vereinbarung:

- Es werden ausschließlich deterministische Einband Turing Maschinen untersucht
- Alle Turing Maschinen arbeiten mit dem Eingabealphabet $\Sigma = \{0, 1\}$
- Eine Turing Maschine wird über dem Alphabet $\Delta = \{0, 1\}$ kodiert

Halteproblem für Turing Maschinen

- Gegeben: Turing Maschine $M = (Z, \Gamma, \Sigma, \sqcup, z_s, z_{acc}, z_{rej})$ und ein Wort $x = \Sigma^*$
- Gefragt: Akzeptiert M die Eingabe x?

Zugehörige Sprache:

$$H = \left\{ \langle M, x \rangle \;\middle|\; \begin{array}{l} \text{die Turing Maschine } M \text{ akzeptiert} \\ \text{die Eingabe } x \end{array} \right\}$$

Das Halteproblem (Forts.)

Satz. H ist semi-entscheidbar.

Beweis. Betrachte die Eingabe $\langle M, x\rangle$

Idee: Simuliere unter Einsatz von Konfigurationsfolgen die Turing Maschine M auf Eingabe x, bis ein Ergebnis feststeht.

Fallunterscheidung:

- M akzeptiert $x \rightsquigarrow$ die Simulation stoppt und akzeptiert die Eingabe $\langle M, x\rangle$
- M verwirft $x \rightsquigarrow$ die Simulation stoppt und verwirft die Eingabe $\langle M, x\rangle$
- M rechnet auf Eingabe x unendlich lange $\rightsquigarrow$ die Simulation stoppt nicht

Fazit: H ist semi-entscheidbar

Spezielles Halteproblem

Beobachtung: Die Kodierung einer Turing Maschine M ist ein Binärwort und kann als Eingabe benutzt werden

Spezielles Halteproblem für Turing Maschinen

- Gegeben: Turing Maschine $M = (Z, \Gamma, \Sigma, \sqcup, z_s, z_{acc}, z_{rej})$ und deren Kodierung $\langle M \rangle$
- Gefragt: Akzeptiert M die Eingabe $\langle M \rangle$?

Zugehörige Sprache:

$$H^* = \left\{ \langle M, \langle M \rangle \rangle \;\middle|\; \begin{array}{l} \text{die Turing Maschine } M \text{ akzeptiert} \\ \text{die Eingabe } \langle M \rangle \end{array} \right\}$$

Spezielles Halteproblem (Forts.)

Fakt: $H^* \subseteq H$

Satz. H^* ist semi-entscheidbar

Beweis.. Analog zu obigem Satz

Beobachtung: Ist H entscheidbar, dann ist auch H^* entscheidbar

Umkehrschluss: Wenn H^* nicht entscheidbar ist, dann ist auch H nicht entscheibar.

Ziel: Beweis, dass H^* nicht entscheidbar ist

Voraussetzung: Aufzählung der Kodierungen aller Turing Maschinen

H^* ist nicht entscheidbar

Satz. H^* ist nicht entscheidbar

Beweis. Angenommen H^* ist entscheidbar.

Dann gibt es eine deterministische Einband Turing Maschine D mit folgenden Eigenschaften:

- $L(D) = H^*$
- D hält auf allen Eingaben

Betrachte die Turing Maschine C mit folgendem Verhalten:

$$C(\langle M \rangle) = \begin{cases} \texttt{true} & D(\langle M, \langle M \rangle \rangle) \text{ verwirft} \\ \texttt{false} & D(\langle M, \langle M \rangle \rangle) \text{ akzeptiert} \end{cases}$$

Darüber hinaus C verwirft alle Wörter x, die keine korrekte Kodierung einer Turing Maschine sind

H^* ist nicht entscheidbar (Forts.)

Es gilt:

- C verhält sich komplementär zu D
- C stoppt auf allen Eingaben, da D auf allen Eingaben stoppt
- Die von C akzeptierte Sprache ist entscheidbar

Betrachte die Eingabe $x = \langle C \rangle$:

- Fall 1: D akzeptiert $\langle C, \langle C \rangle \rangle \rightsquigarrow C$ verwirft $\langle C \rangle$
- Fall 2: D verwirft $\langle C, \langle C \rangle \rangle \rightsquigarrow C$ akzeptiert $\langle C \rangle$

H^* ist nicht entscheidbar (Forts.)

Folgerung (Fall 1):

$$\begin{aligned} C \text{ akzeptiert } \langle C \rangle &\Leftrightarrow D \text{ verwirft } \langle C, \langle C \rangle \rangle \\ &\Leftrightarrow C \text{ akzeptiert } \langle C \rangle \text{ nicht} \end{aligned}$$

Widerspruch!

Auch die Analyse von Fall 2 führt zu einem Widerspruch

Ergebnis: H^* kann nicht entscheidbar sein

Somit:

- H^* ist semi-entscheidbar, aber nicht entscheidbar
- $\overline{H^*}$ ist unentscheidbar

Selbiges gilt für das Halteproblem H

Sprachklassen

Definition.

- Eine Sprachklasse $\mathcal{C}$ über dem Alphabet Σ ist eine Menge von Sprachen aus Σ^*, d.h., $\mathcal{C} \subseteq P(\Sigma^*)$
- Eine Sprachklasse $\mathcal{C}$ heißt nicht-trivial, falls $\mathcal{C} \neq \emptyset$ und $\mathcal{C} \neq P(\Sigma^*)$
- Die Menge zur Klasse $\mathcal{C}$ gehörenden Turing Maschinen ist

$$L_{\mathcal{C}} = \{\langle M \rangle \mid \text{Turing Maschine } M \text{ mit } L(M) \in \mathcal{C}\}$$

Beispiel. $\Sigma = \{0, 1\}$

- $\mathcal{C}_1 = \{L \subseteq \Sigma^* \mid L \text{ ist regulär}\}$
- $\mathcal{C}_2 = \{L \subseteq \Sigma^* \mid L \text{ ist kontextfrei}\}$
- $\mathcal{C}_3 = \{L \subseteq \Sigma^* \mid L \text{ ist semi-entscheidbar}\}$

Der Satz von Rice

Satz. (Satz von Rice)
Für jede nicht-triviale Klasse $\mathcal{C}$ von semi-entscheidbaren Sprachen gilt: $L_{\mathcal{C}}$ ist nicht entscheidbar.

Beweis. Sei $\mathcal{C}$ eine nicht-triviale Klasse von semi-entscheidbaren Sprachen.

Der Einfachheit halber sein angenommen, dass $\emptyset \notin \mathcal{C}$

Da $\mathcal{C}$ nicht-trivial ist, existiert eine Sprache $L \in \mathcal{C}$.

Da L semi-entscheidbar ist, existiert eine Turing Maschine M_L mit $L(M_L) = L$.

Angenommen, $L_{\mathcal{C}}$ ist entscheidbar. Dann existiert eine Turing Maschine $M_{\mathcal{C}}$ mit $L(M_{\mathcal{C}}) = L_{\mathcal{C}}$, die auf allen Eingaben stoppt.

Der Satz von Rice (Forts.)

Definiere für eine Turing Maschine M mit zugehöriger Eingabe x den folgenden Algorithmus:

$\mathrm{A}_{\langle M,x\rangle}(w)$

Input: Wort w

Output: `true`, falls M das Wort x akzeptiert und $w \in L$, `false`, sonst.

if M akzeptiert x **then**
    if M_L akzeptiert w **then**
        return `true`
    else
        return `false`
else
    return `false`

Der Satz von Rice (Forts.)

Die von $\mathrm{A}_{\langle M,x\rangle}$ akzeptierte Sprache ist abhängig von der Berechnung der Turing Maschine M auf Eingabe x:

$$L(\mathrm{A}_{\langle M,x\rangle}) = \begin{cases} L & M \text{ akzeptiert } x \\ \emptyset & M \text{ verwirft } x \end{cases}$$

Somit:

$$\langle \mathrm{A}_{\langle M,x\rangle} \rangle \in L_{\mathcal{C}} \Longleftrightarrow \langle M, x\rangle \in H$$

Der Satz von Rice (Forts.)

Betrachte nun den Algorithmus B:

$\mathrm{B}(\langle M, x \rangle)$
Input: Kodierung $\langle M, x \rangle$ einer Turing Maschine M
mit zugehöriger Eingabe x
Output: `true`, falls M die Eingabe x akzeptiert,
`false`, sonst

Berechne die Kodierung $\langle \mathrm{A}_{\langle M,x \rangle} \rangle$ *des Algorithmus* $\mathrm{A}_{\langle M,x \rangle}$
if $M_{\mathcal{C}}$ akzeptiert $\langle \mathrm{A}_{\langle M,x \rangle} \rangle$ **then**
  return `true`
else
  return `false`

Der Satz von Rice (Forts.)

Es gilt:

- $L(B) = H$
- Da $L_{\mathcal{C}}$ entscheidbar ist, stoppt B auf allen Eingaben

Folgerung: Das Halteproblem H ist entscheidbar

Widerspruch!

Somit kann $L_{\mathcal{C}}$ nicht entscheidbar sein.

Bemerkung: Falls $\emptyset \notin \mathcal{C}$, dann wähle stattdessen eine semi-entscheidbare Sprache $S \notin \mathcal{C}$ und passe den Beweis entsprechend an

Schutzzustand

- Der Zustand eines IT-Systems beinhaltet alle aktuellen Daten des Systems, z.B., Inhalt des Hauptspeichers und der Festplatten, angemeldete Benutzer, laufende Prozesse, ...
- Der Schutzzustand beinhaltet eine Auswahl von Daten des Zustands, die sich auf die Sicherheit des Systems beziehen
- Steht $\mathcal{S}$ für die Menge aller möglichen Schutzzustände, dann ist
  - ▷ $\mathcal{S}_{\text{secure}} \subseteq \mathcal{S}$ die Menge aller sicheren Schutzzustände
  - ▷ $\mathcal{S}_{\text{insecure}} = \mathcal{S} - \mathcal{S}_{\text{secure}}$ die Menge aller unsicheren Schutzzustände

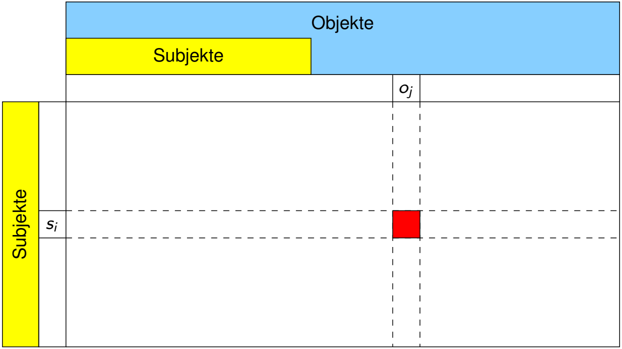
Schutzzustand (Forts.)

- Die Anforderungen an sichere Schutzzustände werden durch eine Sicherheitsstrategie festgelegt
- Ein Sicherheitsmechanismus verhindert, dass ein IT-System von einem sicheren Schutzzustand $Z \in \mathcal{S}_{\text{secure}}$ in einen unsicheren Schutzzustand $Z' \in \mathcal{S}_{\text{insecure}}$ wechseln kann

Zugriffskontrollmatrix

- Die Zugriffskontrollmatrix ist ein Modell zur Bewertung des Schutzzustands eines IT-Systems
- Das Modell der Zugriffskontrollmatrix stammt aus den Forschungsgebieten der Betriebssysteme und Datenbanken
- Das Modell charakterisiert die Berechtigungen von Subjekten gegenüber allen anderen Objekten
- Die Berechtigungen werden in einer Matrix gespeichert

Aufbau einer Zugriffskontrollmatrix



Berechtigungen von Subjekt s_i auf Objekt o_j

Definition Zugriffskontrollmatrix

- O steht für die Menge aller Objekte
- S steht für die Menge aller Subjekte
- R steht für die Menge aller Zugriffsrechte
- Eine Zugriffskontrollmatrix ist eine Matrix A der Dimension $\|S\| \times \|O\|$
- Für alle $s \in S$ und $o \in O$ steht $A[s, o] \subseteq R$ für die Zugriffsrechte des Subjekts s auf das Objekt o
- Das Tripel (S, O, A) beschreibt den Schutzzustand des IT-Systems
- Beachte: Zugriffskontrollmatrizen sind ein Konzept

Beispiel: Zugriffskontrollmatrix

- Das System besteht aus den Objekten Prozess 1, Prozess 2, Datei 1 und Datei 2. Die Prozesse sind die Subjekte
- Die Menge der Zugriffsrechte ist

$$R = \{\text{read}, \text{write}, \text{own}, \text{execute}, \text{append}\}$$

- Eine gültige Zugriffskontrollmatrix ist:

	Datei 1	Datei 2	Prozess 1	Prozess 2
Prozess 1	read, write, own	read	read, write, execute, own	write
Prozess 2	append	read, own	read	read, write, execute, own

Beispiel: UNIX

- Ansatz: Alles ist eine Datei
- Benutzer kann man in Gruppen zusammenfassen
- Der Superuser root besitzt alle Objekte des Systems
- Zugriffsrechte: $R = \{\text{read}, \text{write}, \text{execute}\}$
- Unterscheidung: Datei, Verzeichnis, Prozess
- Zugriffsrechte für Dateien
  - read $\rightsquigarrow$ Lesen der Datei
  - write $\rightsquigarrow$ Schreiben der Datei
  - execute $\rightsquigarrow$ Ausführen der Datei

Beispiel: UNIX (Forts.)

- Zugriffsrechte für Verzeichnisse
  - ▷ read ⇝ Auflisten des Inhalts des Verzeichnisses
  - ▷ write ⇝ Erstellen, Umbenennen und Löschen von Dateien und Unterverzeichnissen
  - ▷ execute ⇝ Zugriff auf die Dateien und Unterverzeichnisse
- Zugriffsrechte für Prozesse
  - ▷ read ⇝ Prozess kann Signale von dem anderen Prozess empfangen
  - ▷ write ⇝ Prozess kann Signale an den anderen Prozess senden
  - ▷ execute ⇝ Prozess kann einen Prozess als Unterprozess ausführen

Beispiel: Netzwerkzugriff

Rechner	tick	trick	track
tick	own	ftp	ftp
trick		ftp, nfs, mail, own	ftp, nfs, mail
track		ftp, mail	ftp, nfs, mail, own

Das Recht own steht für die Möglichkeit, weitere Serverdienste zu starten

Definition Schutzsystem

Definition. Ein Schutzsystem $P = (R, C)$ besteht aus

- einer endlichen Menge R von Berechtigungen und
- einer endlichen Menge C von Befehlen.

Die Konfiguration eines Schutzsystems (zu einem gegebenen Zeitpunkt t) wird durch eine Zugriffskontrollmatrix (S, O, A) dargestellt

Aufbau eines Befehls

Ein Befehl eines Schutzsystems hat folgenden Aufbau:

command $c(p_1, \ldots, p_k)$:
  if $r_1 \in A[s_1, o_1]$ **and**
    $r_2 \in A[s_2, o_2]$ **and**
    $\ldots$
    $r_m \in A[s_m, o_m]$
  then
    op_1
    op_2
    $\ldots$
    op_n
end

Aufbau eines Befehls (Forts.)

Bemerkungen

- Die Werte der Berechtigungen r_i sind konstant
- Die Werte der Subjekte s_i und Objekte o_j werden durch die Parameter $p_1, \ldots, p_k$ bestimmt
- Die Operationen op_i stehen für sogenannte Grundbefehle
- Die Bedingung am Anfang eines Befehls ist optional
- Oder-Verknüpfungen und Negationen von Bedingungen sind *nicht* zulässig
- Ist $n = 1$, dann nennt man den Befehl monooperational
- Ist $m = 1$, dann nennt man den Befehl monokonditional. Falls $m = 2$, dann heißt der Befehl bikonditional

Grundbefehle

Es gibt folgende Grundbefehle:

- Erstellen eines Subjekts
- Erstellen eines Objekts
- Zuweisen eines Zugriffsrechts
- Entziehen eines Zugriffsrechts
- Löschen eines Subjekts
- Löschen eines Objekts

Vereinbarung: Im folgenden stehen (S, O, A) und (S', O', A') für den Schutzzustand vor bzw. nach Ausführung eines Grundbefehls

Erstellen eines Subjekts

Befehl: **create subject** s

Beschreibung:

- Der Befehl erstellt ein neues Subjekt s
- s darf weder als Subjekt noch als Objekt existieren, bevor dieser Befehl ausgeführt wird
- Der Befehl ändert die Grösse der Matrix, fügt aber keine Berechtigungen zur Matrix hinzu

Vorbedingung: $s \notin S$, $s \notin O$

Nachbedingungen:

- $S' = S \cup \{s\}$, $O' = O \cup \{s\}$
- $\forall x \in S' : A'[x, s] = \emptyset$ und $\forall y \in O' : A'[s, y] = \emptyset$
- $\forall x \in S \ \forall y \in O : A'[x, y] = A[x, y]$

Erstellen eines Objekts

Befehl: **create object** o

Beschreibung:

- Der Befehl erstellt ein neues Objekt o
- o darf nicht als Objekt existieren, bevor dieser Befehl ausgeführt wird
- Der Befehl ändert die Grösse der Matrix, fügt aber keine Berechtigungen zur Matrix hinzu

Vorbedingung: $o \notin S$, $o \notin O$

Nachbedingungen:

- $S' = S$, $O' = O \cup \{o\}$
- $\forall x \in S' : A'[x, o] = \emptyset$
- $\forall x \in S \; \forall y \in O : A'[x, y] = A[x, y]$

Hinzufügen von Rechten

Befehl: **enter** r **into** $A[s, o]$

Beschreibung:

- Dieser Befehl gewährt dem Subjekt s das Recht r für Objekt o
- Das Recht r darf in $A[s, o]$ enthalten sein. In diesem Fall ist der Befehl wirkungslos

Vorbedingung: $s \in S$, $o \in O$

Nachbedingungen:

- $S' = S$, $O' = O$
- $A'[s, o] = A[s, o] \cup \{r\}$
- $\forall x \in S \; \forall y \in O : (x, y) \neq (s, o) \rightarrow A'[x, y] = A[x, y]$

Löschen von Rechten

Befehl: **delete** r **from** $A[s, o]$

Beschreibung:

- Dieser Befehl entzieht dem Subjekt s das Recht r für Objekt o
- Das Recht r muss nicht in $A[s, o]$ enthalten sein. In diesem Fall ist der Befehl wirkungslos

Vorbedingung: $s \in S$, $o \in O$

Nachbedingungen:

- $S' = S$, $O' = O$
- $A'[s, o] = A[s, o] - \{r\}$
- $\forall x \in S\ \forall y \in O : (x, y) \neq (s, o) \rightarrow A'[x, y] = A[x, y]$

Löschen eines Subjekts

Befehl: **destroy subject** s

Beschreibung:

- Der Befehl löscht das Subjekt s
- Das Subjekt s muss existieren
- Der Befehl entfernt das Subjekt s und löscht die entsprechende Zeile und Spalte in der Matrix A

Vorbedingung: $s \in S$

Nachbedingungen:

- $S' = S - \{s\}$, $O' = O - \{s\}$
- $\forall x \in S' : A'[s, x] = \emptyset$ und $\forall y \in O' : A'[s, y] = \emptyset$
- $\forall x \in S' \; \forall y \in O' : A'[x, y] = A[x, y]$

Löschen eines Objekts

Befehl: **destroy object** o

Beschreibung:

- Der Befehl löscht das Objekt o
- Das Objekt o muss existieren
- Der Befehl entfernt das Objekt o und löscht die entsprechende Spalte in der Matrix A

Vorbedingung: $o \in O$

Nachbedingungen:

- $S' = S,\ O' = O - \{o\}$
- $\forall x \in S' : A'[s, x] = \emptyset$ und $\forall y \in O' : A'[s, y] = \emptyset$
- $\forall x \in S'\ \forall y \in O' : A'[x, y] = A[x, y]$

Beispiel: Erstellen einer Datei

Vorgang: Ein Prozess p erstellt die Datei f mit den Berechtigungen owner, read und write

Befehl für die Änderung der Zugriffskontrollmatrix:

command *CreateFile*(p, f):
  create object f;
  enter own **into** $A[p, f]$;
  enter read **into** $A[p, f]$;
  enter write **into** $A[p, f]$;
end

Beispiel: Start eines Prozesses

Vorgang: Ein Prozess p startet einen Kindprozess q

Befehl für die Änderung der Zugriffskontrollmatrix:

> **command** *SpawnProcess*(p, q):
> > **create subject** q;
> > **enter** own **into** $A[p, q]$;
> > **enter** read **into** $A[p, q]$;
> > **enter** write **into** $A[p, q]$;
> > **enter** read **into** $A[q, p]$;
> > **enter** write **into** $A[q, p]$;
>
> **end**

Beispiel: Monooperationaler Befehl

Beispiel: Ein Prozess wird Eigentümer einer Datei

command $MakeOwner(p, f)$:
  enter own **into** $A[p, f]$;
end

Beispiel: Bedingter Befehl

Bedingte Befehle werden nur ausgeführt, wenn die an sie geknüpften Bedingungen erfüllt sind

Beispiel: Um einem anderen Prozess q Zugriff auf eine Datei f zu gewähren, muss der Prozess p selbst Eigentümer der Datei sein

> **command** $GrantReadFile(p, f, q)$:
>   **if** own **in** $A[p, f]$ **then**
>   **enter** read **into** $A[q, f]$;
> **end**

Und-Verknüpfungen von Bedingungen

Erlaubt: Und-Verknüpfung von Bedingungen

Beispiel: Angenommen, ein Prozess darf die Leserechte einer Datei verwalten, wenn er selbst die Rechte read und copy für die Datei hat. Dann sieht der Befehl zur Vergabe von Leserechten wie folgt aus.

command *GrantReadFile2*(p, f, q):
  if read **in** $A[p, f]$ **and** copy **in** $A[p, f]$ **then**
    enter read **into** $A[q, f]$;
end

Befehle mit einer Bedingung nennt man monoconditional, Befehle mit zwei Bedingungen nennt man biconditional

Oder-Verknüpfungen von Bedingungen

Nicht erlaubt: Oder-Verknüpfung von Bedingungen

Begründung: Diese Art von Verknüpfung kann durch mehrere separate Bedingte Befehle realisiert werden.

Beispiel: Angenommen, das Recht a ermöglicht die Vergabe von Schreibrechten auf eine Datei.

Betrachte den (nicht erlaubte) Befehl:

> **command** *GrantWriteFile*(p, f, q):
>   **if** own **in** $A[p, f]$ **or** a **in** $A[p, f]$ **then**
>   **enter** write **into** $A[q, f]$;
> **end**

Oder-Verknüpfungen von Bedingungen (Forts.)

Das gewünschte Ergebnis kann durch die Kombination der folgenden Befehle erreicht werden:

command *GrantWriteFile1*(p, f, q):
  if own **in** $A[p, f]$ **then**
    enter write **into** $A[q, f]$;
end

command *GrantWriteFile2*(p, f, q):
  if a **in** $A[p, f]$ **then**
    enter write **into** $A[q, f]$;
end

Negation von Bedingungen

Nicht erlaubt: Negation von Bedingungen

Konsequenz: Man kann nicht festlegen, ob ein Recht *nicht* vorhanden ist

Bemerkung: Das Verbot der Negation führt zu weiteren interessanten Konsequenzen (doch dazu später mehr . . .)

Prinzip der Abschwächung von Privilegien

Prinzip der Abschwächung von Privilegien:

> Ein Subjekt darf Rechte, die es nicht besitzt, nicht an andere Objekte weitergeben

Beachte:

- Der Besitzer eines Objekts kann anderen Subjekten Rechte für das Objekt einräumen, die er selbst nicht besitzt
- Dies widerspricht *nicht* dem obigen Prinzip, da sich der Besitzer vorab selbst die entsprechenden Berechtigungen zuweisen kann

Besondere Berechtigungen

Copy Berechtigung:

- Benutzer darf anderen Benutzern Rechte gewähren
- Das Prinzip der Abschwächung von Privilegien muss beachtet werden
- Es gibt Systeme, bei denen der Benutzer das Recht abgeben muss, wenn er es kopiert

Own Berechtigung:

- Der Eigentümer eines Objekts kann sich selbst alle Rechte gewähren
- Der Eigentümer ist in der Regel das Subjekt, welches das Objekt erstellt hat

Zentrale Fragen

Frage 1: Welche Eigenschaften muss ein Schutzsystem mit zugehörigem Schutzzustand erfüllen, um als sicher eingestuft zu werden?

Frage 2: Kann man algorithmisch feststellen, ob man ein als sicher eingestuftes Schutzsystem mit einer Folge von Befehlen kompromittieren kann?

Transition von Schutzzuständen

- Wenn Subjekte Operationen auf einem IT-System ausführen, dann ändert sich der Schutzzustand
- Der initiale Schutzzustand eines IT-Systems ist $X_0 = (S_0, O_0, A_0)$
- Eine Transaktion $\tau = c(p_1, \ldots, p_k)$ besteht aus einem Befehl c mit zugehörigen Parametern $p_1, \ldots, p_k$
- Die Notation $X_i \vdash_{\tau_{i+1}} X_{i+1}$ steht dafür, dass die Transaktion τ_{i+1} den Schutzzustand X_i in den Schutzzustand X_{i+1} überführt
- Die Notation $X \vdash^* Y$ steht dafür, dass X mittels einer (endlichen) Folge von Transaktionen in den Schutzzustand Y überführbar ist

Beispiel: Prozesse und Dateizugriffe

Szenario: Dateizugriffe von Prozessen

- Subjekte: Prozesse
- Objekte: Prozesse und Dateien
- Anforderung: Jede Datei gehört einem Prozess
- Berechtigungen: `own`, `read`, `write`, `execute`
- Aktionen:
    - ▷ Erstellen einer neuen Datei
    - ▷ Gewähren von Berechtigungen auf eine Datei
    - ▷ Entziehen von Berechtigungen auf eine Datei

Beispiel: Prozesse und Dateizugriffe (Forts.)

Befehle des Schutzsystems:

1. Der Prozess p erstellt eine neue Datei f und wird deren Eigentümer:

   command $\mathit{Create}(p, f)$:
       create object f
       enter `own` **into** $A[p, f]$;
   end

Beispiel: Prozesse und Dateizugriffe (Forts.)

2. Der Eigentümer o der Datei f überträgt ein Recht $r \in \{\texttt{read}, \texttt{write}, \texttt{execute}\}$ an den Prozess p:

   command $Confer_r(o, p, f)$:
       if $\texttt{own}$ **in** $A[o, f]$ **then**
           enter r **into** $A[p, f]$;
   end

3. Der Eigentümer o der Datei f entzieht dem Prozess p das Recht $r \in \{\texttt{read}, \texttt{write}, \texttt{execute}\}$:

   command $Remove_r(o, p, f)$:
       if $\texttt{own}$ **in** $A[o, f]$ **and** r **in** $A[p, f]$ **then**
           delete r **from** $A[p, f]$;
   end

Beispiel: Prozesse und Dateizugriffe (Forts.)

Ausgangssituation: Zwei Prozesse `Sam` und `Joe`

Zugriffskontrollmatrix:

	Sam	Joe
Sam	$\emptyset$	$\emptyset$
Joe	$\emptyset$	$\emptyset$

Vorgang: `Sam` erstellt die zwei Dateien `Code` und `Data` und überträgt `Joe` das Recht, `Code` auszuführen und `Data` zu lesen

Beispiel: Prozesse und Dateizugriffe (Forts.)

Befehlsfolge:

$$\begin{array}{l} \mathit{Create}(\mathtt{Sam}, \mathtt{Code}) \\ \mathit{Create}(\mathtt{Sam}, \mathtt{Data}) \\ \mathit{Confer}_{\mathtt{execute}}(\mathtt{Sam}, \mathtt{Joe}, \mathtt{Code}) \\ \mathit{Confer}_{\mathtt{read}}(\mathtt{Sam}, \mathtt{Joe}, \mathtt{Data}) \end{array}$$

Beispiel: Prozesse und Dateizugriffe (Forts.)

Analyse des Befehls *Create*(Sam, Code):

	Sam	Joe
Sam	$\emptyset$	$\emptyset$
Joe	$\emptyset$	$\emptyset$

$\xRightarrow{\textbf{create object}\ \texttt{Code}}$

	Sam	Joe	Code
Sam	$\emptyset$	$\emptyset$	$\emptyset$
Joe	$\emptyset$	$\emptyset$	$\emptyset$

$\xRightarrow{\textbf{enter}\ \texttt{own}\ \textbf{into}\ \mathrm{A}[\texttt{Sam,Code}]}$

	Sam	Joe	Code
Sam	$\emptyset$	$\emptyset$	{own}
Joe	$\emptyset$	$\emptyset$	$\emptyset$

Beispiel: Prozesse und Dateizugriffe (Forts.)

Darstellung der kompletten Transaktion:

	Sam	Joe
Sam	$\emptyset$	$\emptyset$
Joe	$\emptyset$	$\emptyset$

$\vdash_{\mathit{Create}(\mathtt{Sam},\mathtt{Code})}$

	Sam	Joe	Code
Sam	$\emptyset$	$\emptyset$	$\{\mathtt{own}\}$
Joe	$\emptyset$	$\emptyset$	$\emptyset$

Beispiel: Prozesse und Dateizugriffe (Forts.)

Darstellung der kompletten Transaktionfolge:

$$\begin{array}{c|c|c|}
 & \texttt{Sam} & \texttt{Joe} \\ \hline
\texttt{Sam} & \emptyset & \emptyset \\ \hline
\texttt{Joe} & \emptyset & \emptyset \\ \hline
\end{array} \qquad \vdash_{\mathit{Create}(\texttt{Sam},\texttt{Code})}$$

$$\begin{array}{c|c|c|c|}
 & \texttt{Sam} & \texttt{Joe} & \texttt{Code} \\ \hline
\texttt{Sam} & \emptyset & \emptyset & \{\texttt{own}\} \\ \hline
\texttt{Joe} & \emptyset & \emptyset & \emptyset \\ \hline
\end{array} \qquad \vdash_{\mathit{Create}(\texttt{Sam},\texttt{Data})}$$

$$\begin{array}{c|c|c|c|c|}
 & \texttt{Sam} & \texttt{Joe} & \texttt{Code} & \texttt{Data} \\ \hline
\texttt{Sam} & \emptyset & \emptyset & \{\texttt{own}\} & \{\texttt{own}\} \\ \hline
\texttt{Joe} & \emptyset & \emptyset & \emptyset & \emptyset \\ \hline
\end{array} \qquad \vdash_{\mathit{Confer}_{\texttt{execute}}(\texttt{Sam},\texttt{Joe},\texttt{Code})}$$

Beispiel: Prozesse und Dateizugriffe (Forts.)

	Sam	Joe	Code	Data
Sam	$\emptyset$	$\emptyset$	{own}	{own}
Joe	$\emptyset$	$\emptyset$	{*execute*}	$\emptyset$

$\vdash_{Confer_{\mathtt{read}}(\mathtt{Sam},\mathtt{Joe},\mathtt{Data})}$

	Sam	Joe	Code	Data
Sam	$\emptyset$	$\emptyset$	{own}	{own}
Joe	$\emptyset$	$\emptyset$	{execute}	{read}

Beispiel: Prozesse und Dateizugriffe (Forts.)

Kurzschreibweise:

	Sam	Joe
Sam	$\emptyset$	$\emptyset$
Joe	$\emptyset$	$\emptyset$

$\vdash^*$

	Sam	Joe	Code	Data
Sam	$\emptyset$	$\emptyset$	{own}	{own}
Joe	$\emptyset$	$\emptyset$	{execute}	{read}

Zentrale Fragen

Frage 1: Welche Eigenschaften muss ein Schutzsystem mit zugehörigem Schutzzustand erfüllen, um als sicher eingestuft zu werden?

Frage 2: Kann man algorithmisch feststellen, ob man ein als sicher eingestuftes Schutzsystem mit einer Folge von Befehlen kompromittieren kann?

Stabile Schutzzustände

Definition. Gegeben ist ein Schutzsystem $P = (R, C)$. Der Schutzzustand (S, O, A) ist stabil für die Berechtigung $r \in R$, falls es mit keiner Folge von Befehlen möglich ist, (S, O, A) in einen Schutzzustand (S', O', A') zu überführen, so dass $r \in A'[s, o]$ und $r \notin A[s, o]$ für $(s, o) \in S' \times O'$ gilt

Nebenbedingungen:

- Das Prinzip der Abschwächung von Privilegien wird nicht berücksichtigt
- Es gibt weder eine Copy noch eine Own Berechtigung

Beachte: Stabilität ist nicht identisch mit Sicherheit

Entscheidungsproblem

Instable Security State (ISS)

Gegeben: Schutzsystem $P = (R, C)$, Schutzzustand $X = (S, O, A)$, Berechtigung $r \in R$

Gefragt: Ist der Schutzzustand X nicht stabil für r?
Oder formal: Existiert eine Folge von Befehlen $\tau_i = c_i(p_{i,1}, \ldots, p_{i,\ell_i})$, $1 \leq i \leq k$, so dass

$$X = X_0 \vdash_{\tau_1} X_1 \vdash_{\tau_2} X_2 \vdash_{\tau_3} \ldots \vdash_{\tau_{k-1}} X_{k-1} \vdash_{\tau_k} X_k$$

und $r \in A_k[s, o]$ und $r \notin A_0[s, o]$ für ein $(s, o) \in S_k \times O_k$?

ISS ist semi-entscheidbar

Satz. Das ISS Problem ist semi-entscheidbar

Beweis. (Idee) Man kann algorithmisch

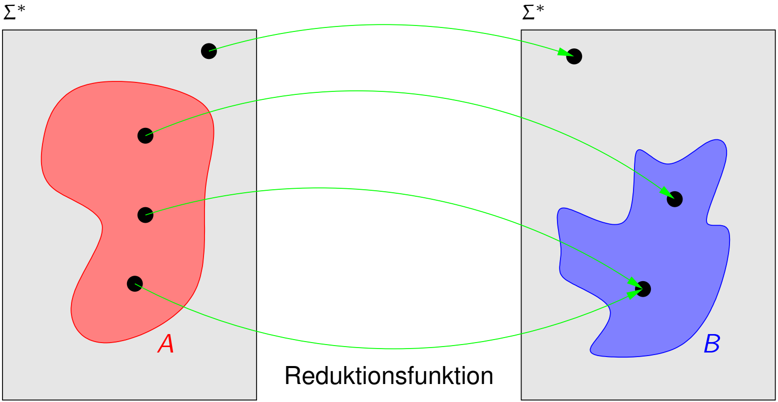
- alle endlichen Folgen von Transaktionen schrittweise erzeugen und
- überprüfen, ob die jeweilige Transaktionsfolge den Schutzzustand X in einen Schutzzustant X_k überführt, in dem das Recht r fälschlicherweise übertragen wurde

Ist X nicht stabil, dann wird eine passende Transaktionsfolge gefunden und der Algorithmus akzeptiert.

Many-One Reduktionen (Idee)

Idee: Löse A unter Verwendung eines Algorithmus für B

Ansatz: Übertrage das Wortproblem für A auf das Wortproblem für B



Many-One Reduktionen

Definition. Seien A und B Sprachen über dem Alphabet Σ.

A ist many-one reduzierbar auf B, symbolisch $A \leq_m B$, falls es eine Funktion $f : \Sigma^* \mapsto \Sigma^*$ gibt mit folgenden Eigenschaften:

- Für alle $x \in \Sigma^*$ gilt: $x \in A \Longleftrightarrow f(x) \in B$
- f ist mittels einer Turing Maschine berechenbar, die auf allen Eingaben stoppt und für jede Eingabe x den Wert $f(x)$ zurückliefert

Wissenswertes zu Reduktionen

Satz. Seien A und B Sprachen. Falls B entscheidbar ist und $A \leq_m B$, dann ist auch A entscheidbar.

Beweis.

- Sei M eine Turing Maschine mit $L(M) = B$, die auf allen Eingaben stoppt
- Sei f eine mittels einer Turing Maschine berechenbare Reduktionsfunktion von A nach B

Der Algorithmus B (siehe nächste Seite) akzeptiert die Sprache A und stoppt auf jeder Eingabe

Somit: A ist entscheidbar

Wissenswertes zu Reduktionen (Forts.)

Entscheidungsalgorithmus für B:

$\mathrm{B}(x)$
Input: Wort x
Output: `true`, falls $x \in A$, `false`, sonst

$y := f(x)$;
if M_B akzeptiert x **then**
  return `true`
else
  return `false`

Konsequenz: Falls $A \leq_m B$ und A nicht entscheidbar, dann ist B ebenfalls nicht entscheidbar

ISS ist nicht entscheidbar

Satz. H $\leq_m$ ISS

Beweis. Betrachte eine beliebige deterministische Einband Turing Maschine $M = (Z, \Gamma, \Sigma, \sqcup, \delta, z_s, z_{acc}, z_{rej})$.

Ansatz: Anhand M und der Eingabe $x \in \Sigma^*$ wird ein Schutzsystem P mit zugehörigem Schutzzustand X konstruiert, der genau dann nicht stabil ist, wenn M die Eingabe x akzeptiert

Konstruktion in zwei Schritten:

1. Konstruktion der Berechtigungen und Befehle des Schutzsystems anhand von M
2. Konstruktion der Zugriffskontrollmatrix anhand von x

ISS ist nicht entscheidbar (Forts.)

Idee:

- Eine Konfiguration von M wird als Zugriffskontrollmatrix dargestellt
- Ein Subjekt repräsentiert eine Bandzelle
- Ein Subjekt ist Eigentümer des Subjekts rechts neben ihm
- Der Inhalt der Bandzellen, der aktuelle Zustand und die Position des L/S-Kopfs wird mittels Berechtigungen festgelegt
- Die Enden des Arbeitsbands werden mit zwei Berechtigungen `endl` und `endr` markiert

ISS ist nicht entscheidbar (Forts.)

Beispiel. Betrachte die Konguration (a, z, bba)

Die entsprechende Zugriffskontrollmatrix ist:

	s_1	s_2	s_3	s_4
s_1	$\{a, \texttt{endl}\}$	$\{\texttt{own}\}$		
s_2		$\{z, b\}$	$\{\texttt{own}\}$	
s_3			$\{b\}$	$\{\texttt{own}\}$
s_4				$\{b, \texttt{endr}\}$

ISS ist nicht entscheidbar (Forts.)

Der Rechenschritt $\delta(z, b) = (z', a, R)$ führt zu folgender Zugriffskontrollmatrix:

	s_1	s_2	s_3	s_4
s_1	$\{a, \texttt{endl}\}$	$\{\texttt{own}\}$		
s_2		$\{a\}$	$\{\texttt{own}\}$	
s_3			$\{z', b\}$	$\{\texttt{own}\}$
s_4				$\{b, \texttt{endr}\}$

ISS ist nicht entscheidbar (Forts.)

Schritt 1: Konstruktion des Schutzsystems $P_M = (R_M, C_M)$.

Die Menge der Berechtigungen ist definiert als:

$$R = Z \cup \Gamma \cup \{\texttt{own}, \texttt{endl}, \texttt{endr}\}$$

Die Befehle werden anhand der Überführungsfunktion festgelegt.

Für jeden möglichen Rechenschritt werden zwei Befehle erzeugt:

- Der erste Befehl wird angewandt wenn sich der L/S-Kopf nicht am Rand des bereits besuchten Teil des Arbeitsbands befindet
- Der zweite Befehl behandelt den Spezialfall, dass der L/S-Kopf auf eine neue Bandzelle positioniert wird

ISS ist nicht entscheidbar (Forts.)

Fall 1: $\delta(z, a) = (z', b, L)$

Befehl 1:

command $C_{(z,a)}(s, s')$:
  if `own` **in** $A[s, s']$ **and**
    z **in** $A[s', s']$ **and**
    a **in** $A[s', s']$
  then
    delete z **from** $A[s', s']$;
    delete a **from** $A[s', s']$;
    enter b **into** $A[s', s']$;
    enter z' **into** $A[s, s]$;
end

ISS ist nicht entscheidbar (Forts.)

Befehl 2:

command $D_{(z,a)}(s, s')$:
  if `endl` **in** $A[s', s']$ **and**
    z **in** $A[s', s']$ **and**
    a **in** $A[s', s']$
  then
    create subject s;
    enter ␣ **into** $A[s, s]$;
    enter `own` **into** $A[s, s']$;
    delete `endl` **from** $A[s', s']$;
    enter `endl` **into** $A[s, s]$;
    delete z **from** $A[s', s']$;
    delete a **from** $A[s', s']$;
    enter b **into** $A[s', s']$;
    enter z' **into** $A[s, s]$;
end

ISS ist nicht entscheidbar (Forts.)

Fall 2: $\delta(z, a) = (z', b, R)$

Befehl 1:

command $C_{(z,a)}(s, s')$:
  if `own` **in** $A[s, s']$ **and**
    z **in** $A[s, s]$ **and**
    a **in** $A[s, s]$
  then
    delete z **from** $A[s, s]$;
    delete a **from** $A[s, s]$;
    enter b **into** $A[s, s]$;
    enter z' **into** $A[s', s']$;
end

ISS ist nicht entscheidbar (Forts.)

Befehl 2:

command $D_{(z,a)}(s, s')$:
  if $\texttt{endr}$ **in** $A[s, s]$ **and**
    z **in** $A[s, s]$ **and**
    a **in** $A[s, s]$
  then
    create subject s';
    enter ␣ **into** $A[s', s']$;
    enter $\texttt{own}$ **into** $A[s, s']$;
    delete $\texttt{endr}$ **from** $A[s, s]$;
    enter $\texttt{endr}$ **into** $A[s', s']$;
    delete z **from** $A[s, s]$;
    delete a **from** $A[s, s]$;
    enter b **into** $A[s, s]$;
    enter z' **into** $A[s, s]$;
end

ISS ist nicht entscheidbar (Forts.)

Schritt 2: Für $x = a_1 \dots a_k$ wird der Schutzzustand $X_{a_1 \dots a_k}$ folgendermaßen definiert:

	s_1	s_2	s_3	$\dots$	s_k
s_1	$\{\mathtt{endl}, a_1, z_s\}$	$\{\mathtt{own}\}$		$\dots$	
s_2		$\{a_2\}$	$\{\mathtt{own}\}$	$\dots$	
$\vdots$					
s_k				$\dots$	$\{\mathtt{endr}, a_k\}$

ISS ist nicht entscheidbar (Forts.)

Bemerkungen:

- Für jeden aus $X_{a_1 \ldots a_k}$ ableitbaren Schutzzustand X ist höchstens ein Befehl anwendbar
- Die Folge der anwendbaren Befehle simuliert die Turing Maschine M auf Eingabe $a_1 \ldots a_k$
- Es gilt: $X_{a_1 \ldots a_k}$ ist nicht stabil für die Berechtigung z_{acc} genau dann, wenn M die Eingabe $a_1 \ldots a_k$ akzeptiert

  Oder formal: $\langle M, a_1 \ldots a_k \rangle \in \mathrm{H}$ genau dann, wenn $\langle P_M, X_{a_1 \ldots a_k}, z_{acc} \rangle \in \mathrm{ISS}$
- Die Transformation von $(M, a_1 \ldots a_k)$ in $(P_M, X_{a_1 \ldots a_k})$ ist berechenbar

Ergebnis: $\mathrm{H} \leq_m \mathrm{ISS}$

ISS ist nicht entscheidbar (Forts.)

Satz. ISS ist nicht entscheidbar

Beweis. Da das Halteproblem H nicht entscheidbar ist und H $\leq_m$ ISS gilt, folgt mit obigem Satz, dass ISS nicht entscheidbar ist

Satz. Die Fragestellung, ob ein Schutzzustand eines Schutzsystems stabil ist, ist unentscheidbar

Satz. Enthält das Schutzproblem nur monooperationale Befehle, dann ist das entsprechende ISS Problem entscheidbar

Zusammenfassung

- Man unterscheidet zwischen
  - entscheidbaren,
  - semi-entscheidbaren und
  - unentscheidbaren

  Sprachen
- Das Halteproblem ist semi-entscheidbar, aber nicht entscheidbar
- Der Satz von Rice liefert als Ergebnis, dass (fast) alle Fragen bezüglich Turing Maschinen nicht entscheidbar sind
- Reduktionen vereinfachen den Nachweis der Nicht-Entscheidbarkeit eines Problems