SS 2005

Einführung in die Informatik IV

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005SS/info4/index.html.de

20. Juni 2005

3. Anwendung der Unentscheidbarkeitsresultate auf kontextfreie Sprachen

Wie wir gesehen haben, gilt:

1. die regulären Sprachen sind unter allen Booleschen Operationen abgeschlossen.
2. die kontextfreien Sprachen sind nicht unter Komplement und Durchschnitt abgeschlossen.

Können wir entscheiden, ob der Durchschnitt zweier kontextfreier Sprachen leer ist?

3. Anwendung der Unentscheidbarkeitsresultate auf kontextfreie Sprachen

Wie wir gesehen haben, gilt:

1. die regulären Sprachen sind unter allen Booleschen Operationen abgeschlossen.
2. die kontextfreien Sprachen sind nicht unter Komplement und Durchschnitt abgeschlossen.

Können wir entscheiden, ob der Durchschnitt zweier kontextfreier Sprachen leer ist?

3. Anwendung der Unentscheidbarkeitsresultate auf kontextfreie Sprachen

Wie wir gesehen haben, gilt:

1. die regulären Sprachen sind unter allen Booleschen Operationen abgeschlossen.
2. die kontextfreien Sprachen sind nicht unter Komplement und Durchschnitt abgeschlossen.

Können wir entscheiden, ob der Durchschnitt zweier kontextfreier Sprachen leer ist?

3. Anwendung der Unentscheidbarkeitsresultate auf kontextfreie Sprachen

Wie wir gesehen haben, gilt:

1. die regulären Sprachen sind unter allen Booleschen Operationen abgeschlossen.
2. die kontextfreien Sprachen sind nicht unter Komplement und Durchschnitt abgeschlossen.

Können wir entscheiden, ob der Durchschnitt zweier kontextfreier Sprachen leer ist?

3. Anwendung der Unentscheidbarkeitsresultate auf kontextfreie Sprachen

Wie wir gesehen haben, gilt:

1. die regulären Sprachen sind unter allen Booleschen Operationen abgeschlossen.
2. die kontextfreien Sprachen sind nicht unter Komplement und Durchschnitt abgeschlossen.

Können wir entscheiden, ob der Durchschnitt zweier kontextfreier Sprachen leer ist?

Sie M eine (beliebige) TM (mit nur einem Band) mit Bandalphabet Σ und Zustandsmenge Q, sei $\# \notin \Sigma \cup Q$.

Definition 155

Definiere die Sprachen

$C_M^{(0)} := \{c_0 \# c_1^R \# c_2 \# c_3^R \ldots c_{2m+1}^R;$ $m \geq 0$, c_i ist Konfiguration von M, c_0 ist Anfangskonfiguration auf leerem Band, c_{2m+1} ist Endkonfiguration oder $c_{2m} = c_{2m+1}$ und c_{2m} ist Endkonfiguration, und c_{2j+1} ist Nachfolgekonfiguration von c_{2j} für alle j $\}$

$C_M^{(1)} := \{c_0 \# c_1^R \# c_2 \# c_3^R \ldots c_{2m+1}^R;$ wie oben, jetzt aber: c_{2j} ist Nachfolgekonfiguration von c_{2j-1} für alle zutreffenden $j \geq 1\}$

Sie M eine (beliebige) TM (mit nur einem Band) mit Bandalphabet Σ und Zustandsmenge Q, sei $\# \notin \Sigma \cup Q$.

Definition 155

Definiere die Sprachen

$C_M^{(0)} := \{c_0 \# c_1^R \# c_2 \# c_3^R \dots c_{2m+1}^R;$ $m \geq 0$, c_i ist Konfiguration von M, c_0 ist Anfangskonfiguration auf leerem Band, c_{2m+1} ist Endkonfiguration oder $c_{2m} = c_{2m+1}$ und c_{2m} ist Endkonfiguration, und c_{2j+1} ist Nachfolgekonfiguration von c_{2j} für alle j $\}$

$C_M^{(1)} := \{c_0 \# c_1^R \# c_2 \# c_3^R \dots c_{2m+1}^R;$ wie oben, jetzt aber: c_{2j} ist Nachfolgekonfiguration von c_{2j-1} für alle zutreffenden $j \geq 1\}$

Bemerkung: $C_M^{(0)}$ enthält nicht nur „echte“ Rechnungen von M, da $c_{2j-1} \longrightarrow c_{2j}$ nicht unbedingt ein Schritt sein muss; das fordern wir jeweils nur für $c_{2j} \longrightarrow c_{2j+1}$.

Lemma 156

Die Sprachen $C_M^{(0)}$ und $C_M^{(1)}$ sind deterministisch kontextfrei.

Beweis:

Es ist einfach, jeweils einen DPDA dafür zu konstruieren. □

Bemerkung: $C_M^{(0)}$ enthält nicht nur „echte“ Rechnungen von M, da $c_{2j-1} \longrightarrow c_{2j}$ nicht unbedingt ein Schritt sein muss; das fordern wir jeweils nur für $c_{2j} \longrightarrow c_{2j+1}$.

Lemma 156

Die Sprachen $C_M^{(0)}$ und $C_M^{(1)}$ sind deterministisch kontextfrei.

Beweis:

Es ist einfach, jeweils einen DPDA dafür zu konstruieren. □

Bemerkung: $C_M^{(0)}$ enthält nicht nur „echte“ Rechnungen von M, da $c_{2j-1} \longrightarrow c_{2j}$ nicht unbedingt ein Schritt sein muss; das fordern wir jeweils nur für $c_{2j} \longrightarrow c_{2j+1}$.

Lemma 156

Die Sprachen $C_M^{(0)}$ und $C_M^{(1)}$ sind deterministisch kontextfrei.

Beweis:

Es ist einfach, jeweils einen DPDA dafür zu konstruieren. □

Bemerkung: Ein Kellerautomat ist lange nicht so mächtig wie eine Turingmaschine. Aber zwei Kellerautomaten (oder eine endliche Kontrolle mit zwei Kellern) sind so mächtig wie eine Turingmaschine (siehe Übung).

Lemma 157

$$w \in H_0 \Leftrightarrow C^{(0)}_{M_w} \cap C^{(1)}_{M_w} \neq \emptyset$$

Beweis:
Unmittelbar aus der Definition der beiden Sprachen! □

Bemerkung: Falls M_w deterministisch ist und $w \in H_0$, dann enthält $C^{(0)}_{M_w} \cap C^{(1)}_{M_w}$ genau ein Element, nämlich die eine Rechnung von M_w auf leerem Band.

Bemerkung: Ein Kellerautomat ist lange nicht so mächtig wie eine Turingmaschine. Aber zwei Kellerautomaten (oder eine endliche Kontrolle mit zwei Kellern) sind so mächtig wie eine Turingmaschine (siehe Übung).

Lemma 157

$$w \in H_0 \Leftrightarrow C^{(0)}_{M_w} \cap C^{(1)}_{M_w} \neq \emptyset$$

Beweis:
Unmittelbar aus der Definition der beiden Sprachen! □

Bemerkung: Falls M_w deterministisch ist und $w \in H_0$, dann enthält $C^{(0)}_{M_w} \cap C^{(1)}_{M_w}$ genau ein Element, nämlich die eine Rechnung von M_w auf leerem Band.

Bemerkung: Ein Kellerautomat ist lange nicht so mächtig wie eine Turingmaschine. Aber zwei Kellerautomaten (oder eine endliche Kontrolle mit zwei Kellern) sind so mächtig wie eine Turingmaschine (siehe Übung).

Lemma 157

$$w \in H_0 \Leftrightarrow C^{(0)}_{M_w} \cap C^{(1)}_{M_w} \neq \emptyset$$

Beweis:

Unmittelbar aus der Definition der beiden Sprachen! □

Bemerkung: Falls M_w deterministisch ist und $w \in H_0$, dann enthält $C^{(0)}_{M_w} \cap C^{(1)}_{M_w}$ genau ein Element, nämlich die eine Rechnung von M_w auf leerem Band.

Bemerkung: Ein Kellerautomat ist lange nicht so mächtig wie eine Turingmaschine. Aber zwei Kellerautomaten (oder eine endliche Kontrolle mit zwei Kellern) sind so mächtig wie eine Turingmaschine (siehe Übung).

Lemma 157

$$w \in H_0 \Leftrightarrow C^{(0)}_{M_w} \cap C^{(1)}_{M_w} \neq \emptyset$$

Beweis:

Unmittelbar aus der Definition der beiden Sprachen! □

Bemerkung: Falls M_w deterministisch ist und $w \in H_0$, dann enthält $C^{(0)}_{M_w} \cap C^{(1)}_{M_w}$ genau ein Element, nämlich die eine Rechnung von M_w auf leerem Band.

Satz 158

Das Schnittproblem für kontextfreie Sprachen ist unentscheidbar!

Beweis:
siehe oben □

Wir haben sogar gezeigt: Das Schnittproblem für deterministisch kontextfreie Sprachen ist unentscheidbar!

Satz 158

Das Schnittproblem für kontextfreie Sprachen ist unentscheidbar!

Beweis:

siehe oben □

Wir haben sogar gezeigt: Das Schnittproblem für deterministisch kontextfreie Sprachen ist unentscheidbar!

Satz 158

Das Schnittproblem für kontextfreie Sprachen ist unentscheidbar!

Beweis:

siehe oben □

Wir haben sogar gezeigt: Das Schnittproblem für deterministisch kontextfreie Sprachen ist unentscheidbar!

In der Literatur wird dieser Satz üblicherweise mit dem Post'schen Korrespondenzproblem (PCP) bewiesen.

Definition 159 (Post'sches Korrespondenzproblem)

Gegeben: $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$ mit $x_i, y_i \in \Sigma^+$
Frage: gibt es eine Folge von Indizes $i_1, i_2, \ldots i_r \in \{1, \ldots, n\}$, so dass

$$x_{i_1} x_{i_2} \ldots x_{i_r} = y_{i_1} y_{i_2} \ldots y_{i_r} \; ?$$

Satz 160

PCP ist unentscheidbar.

In der Literatur wird dieser Satz üblicherweise mit dem Post'schen Korrespondenzproblem (PCP) bewiesen.

Definition 159 (Post'sches Korrespondenzproblem)

Gegeben: $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$ mit $x_i, y_i \in \Sigma^+$
Frage: gibt es eine Folge von Indizes $i_1, i_2, \ldots i_r \in \{1, \ldots, n\}$, so dass

$$x_{i_1} x_{i_2} \ldots x_{i_r} = y_{i_1} y_{i_2} \ldots y_{i_r} \; ?$$

Satz 160

PCP ist unentscheidbar.

Beweis:

Wir skizzieren, wie man mit Hilfe des PCP die Berechnung einer (det.) TM simulieren kann. Wir haben dazu (u.a.) Paare

(a, a) für alle $a \in \Sigma$

$(u_1u_2u_3, aqb)$ gemäß der inversen Übergangsfkt der TM, mit $a, b \in \Sigma, q \in Q$ und $u_1, u_2, u_3 \in \Sigma \cup Q$

Dies bedeutet, dass die TM bei der lokalen Konfiguration aqb diese im nächsten Schritt zu $u_1u_2u_3$ ändert.

Beweis:

Wir skizzieren, wie man mit Hilfe des PCP die Berechnung einer (det.) TM simulieren kann. Wir haben dazu (u.a.) Paare

$$
\begin{aligned}
(a, a) &\text{ für alle } a \in \Sigma \\
(u_1 u_2 u_3, aqb) &\text{ gemäß der inversen Übergangsfkt} \\
&\text{der TM, mit } a, b \in \Sigma, q \in Q \text{ und} \\
&u_1, u_2, u_3 \in \Sigma \cup Q
\end{aligned}
$$

Dies bedeutet, dass die TM bei der lokalen Konfiguration aqb diese im nächsten Schritt zu $u_1 u_2 u_3$ ändert.

Beweis:

Die allgemeine Situation sieht dann so aus, dass eine geeignete Indexfolge $i_1, \ldots, i_k$ folgende Zeichenreihen erzeugt:

x	c_1	$\ldots$	c_{r-1}	x_1	$\ldots$	x_{i-1}	q	x_i	x_{i+1}	$\ldots$	x_s
y	c_1	$\ldots$	c_{r-1}								

Beweis:

Die allgemeine Situation sieht dann so aus, dass eine geeignete Indexfolge $i_1, \ldots, i_k$ folgende Zeichenreihen erzeugt:

x	c_1	$\ldots$	c_{r-1}	x_1	$\ldots$	x_{i-1}	q	x_i	x_{i+1}	$\ldots$	x_s
y	c_1	$\ldots$	c_{r-1}								

Es müssen nun die einzelnen x_i durch Paare der Form (a, a) gematcht werden, lediglich $x_{i-1}qx_i$ kann nur durch (genau bzw. höchstens) ein Paar der zweiten Form gematcht werden.

Beweis:

Die allgemeine Situation sieht dann so aus, dass eine geeignete Indexfolge $i_1, \ldots, i_k$ folgende Zeichenreihen erzeugt:

x	c_1	$\ldots$	c_{r-1}	x_1	$\ldots$	x_{i-1}	q	x_i	x_{i+1}	$\ldots$	x_s
y	c_1	$\ldots$	c_{r-1}								

Es müssen nun die einzelnen x_i durch Paare der Form (a, a) gematcht werden, lediglich $x_{i-1}qx_i$ kann nur durch (genau bzw. höchstens) ein Paar der zweiten Form gematcht werden.
Damit ergibt sich wieder die allgemeine Situation wie oben, mit r um 1 erhöht, und man kann das Argument per Induktion abschließen.

Beweis:

Die allgemeine Situation sieht dann so aus, dass eine geeignete Indexfolge $i_1, \ldots, i_k$ folgende Zeichenreihen erzeugt:

x	c_1	$\ldots$	c_{r-1}	x_1	$\ldots$	x_{i-1}	q	x_i	x_{i+1}	$\ldots$	x_s
y	c_1	$\ldots$	c_{r-1}								

Es müssen nun die einzelnen x_i durch Paare der Form (a, a) gematcht werden, lediglich $x_{i-1}qx_i$ kann nur durch (genau bzw. höchstens) ein Paar der zweiten Form gematcht werden.
Damit ergibt sich wieder die allgemeine Situation wie oben, mit r um 1 erhöht, und man kann das Argument per Induktion abschließen.
Wir überlassen es als Übungsaufgabe herauszufinden, wie auch Anfang und Ende der TM-Berechnung geeignet durch das PCP simuliert werden können. □

Kapitel III Algorithmen und Datenstrukturen

1. Analyse von Algorithmen

Wir wollen die Ressourcen bestimmen, die, in Abhängigkeit von der Eingabe, ein Algorithmus benötigt, z.B.

1. Laufzeit
2. Speicherplatz
3. Anzahl Prozessoren
4. Programmlänge
5. Tinte
6. ...

Kapitel III Algorithmen und Datenstrukturen

1. Analyse von Algorithmen

Wir wollen die Ressourcen bestimmen, die, in Abhängigkeit von der Eingabe, ein Algorithmus benötigt, z.B.

1. Laufzeit
2. Speicherplatz
3. Anzahl Prozessoren
4. Programmlänge
5. Tinte
6. ...

Kapitel III Algorithmen und Datenstrukturen

1. Analyse von Algorithmen

Wir wollen die Ressourcen bestimmen, die, in Abhängigkeit von der Eingabe, ein Algorithmus benötigt, z.B.

1. Laufzeit
2. Speicherplatz
3. Anzahl Prozessoren
4. Programmlänge
5. Tinte
6. ...

Kapitel III Algorithmen und Datenstrukturen

1. Analyse von Algorithmen

Wir wollen die Ressourcen bestimmen, die, in Abhängigkeit von der Eingabe, ein Algorithmus benötigt, z.B.

1. Laufzeit
2. Speicherplatz
3. Anzahl Prozessoren
4. Programmlänge
5. Tinte
6. ...

Kapitel III Algorithmen und Datenstrukturen

1. Analyse von Algorithmen

Wir wollen die Ressourcen bestimmen, die, in Abhängigkeit von der Eingabe, ein Algorithmus benötigt, z.B.

1. Laufzeit
2. Speicherplatz
3. Anzahl Prozessoren
4. Programmlänge
5. Tinte
6. ...

Kapitel III Algorithmen und Datenstrukturen

1. Analyse von Algorithmen

Wir wollen die Ressourcen bestimmen, die, in Abhängigkeit von der Eingabe, ein Algorithmus benötigt, z.B.

1. Laufzeit
2. Speicherplatz
3. Anzahl Prozessoren
4. Programmlänge
5. Tinte
6. . . .

Kapitel III Algorithmen und Datenstrukturen

1. Analyse von Algorithmen

Wir wollen die Ressourcen bestimmen, die, in Abhängigkeit von der Eingabe, ein Algorithmus benötigt, z.B.

1. Laufzeit
2. Speicherplatz
3. Anzahl Prozessoren
4. Programmlänge
5. Tinte
6. . . .

Kapitel III Algorithmen und Datenstrukturen

1. Analyse von Algorithmen

Wir wollen die Ressourcen bestimmen, die, in Abhängigkeit von der Eingabe, ein Algorithmus benötigt, z.B.

1. Laufzeit
2. Speicherplatz
3. Anzahl Prozessoren
4. Programmlänge
5. Tinte
6. ...

Beispiel 161

Prozedur für Fakultätsfunktion

func $fak(n)$
  $m := 1$
  for $i := 2$ **to** n **do**
    $m := m * i$
  do
  return (m)

Diese Prozedur benötigt $O(n)$ Schritte bzw. arithmetische Operationen.

Jedoch: die Länge der Ausgabe ist etwa

$$\lceil \log_{10} n! \rceil = \Omega(n \log n) \text{ Bits.}$$

Beispiel 161

Prozedur für Fakultätsfunktion

func $fak(n)$
  $m := 1$
  for $i := 2$ **to** n **do**
    $m := m * i$
  do
  return (m)

Diese Prozedur benötigt $O(n)$ Schritte bzw. arithmetische Operationen.

Jedoch: die Länge der Ausgabe ist etwa

$$\lceil \log_{10} n! \rceil = \Omega(n \log n) \text{ Bits.}$$

Beispiel 161

Prozedur für Fakultätsfunktion

func $fak(n)$
  $m := 1$
  for $i := 2$ **to** n **do**
    $m := m * i$
  do
  return (m)

Diese Prozedur benötigt $O(n)$ Schritte bzw. arithmetische Operationen.

Jedoch: die Länge der Ausgabe ist etwa

$$\lceil \log_{10} n! \rceil = \Omega(n \log n) \text{ Bits.}$$

Bemerkung:

Um die Zahl $n \in \mathbb{N}_0$ in Binärdarstellung hinzuschreiben, benötigt man

$$\ell(n) := \begin{cases} 1 & \text{für } n = 0 \\ 1 + \lfloor \log_2(n) \rfloor & \text{sonst} \end{cases}$$
$$= \begin{cases} 1 & \text{für } n = 0 \\ \lceil \log_2(n+1) \rceil & \text{sonst} \end{cases}$$

Um die Notation zu vereinfachen, vereinbaren wir im Zusammenhang mit Komplexitätsabschätzungen

$$\log(0) := 0 \,.$$

Bemerkung:

Um die Zahl $n \in \mathbb{N}_0$ in Binärdarstellung hinzuschreiben, benötigt man

$$\ell(n) := \begin{cases} 1 & \text{für } n = 0 \\ 1 + \lfloor \log_2(n) \rfloor & \text{sonst} \end{cases}$$
$$= \begin{cases} 1 & \text{für } n = 0 \\ \lceil \log_2(n+1) \rceil & \text{sonst} \end{cases}$$

Um die Notation zu vereinfachen, vereinbaren wir im Zusammenhang mit Komplexitätsabschätzungen

$$\log(0) := 0\,.$$

1.1 Referenzmaschine

Wir wählen als Referenzmaschine die Registermaschine (engl. random access machine, RAM) oder auch WHILE-Maschine, also eine Maschine, die WHILE-Programme verarbeiten kann, erweitert durch

- IF ... THEN ... ELSE ... FI
- Multiplikation und Division
- indirekte Adressierung
- arithmetische Operationen wie $\sqrt{n}$, $\sin n$,...

1.1 Referenzmaschine

Wir wählen als Referenzmaschine die Registermaschine (engl. random access machine, RAM) oder auch WHILE-Maschine, also eine Maschine, die WHILE-Programme verarbeiten kann, erweitert durch

- IF ... THEN ... ELSE ... FI
- Multiplikation und Division
- indirekte Adressierung
- arithmetische Operationen wie $\sqrt{n}$, $\sin n$,...

1.1 Referenzmaschine

Wir wählen als Referenzmaschine die Registermaschine (engl. random access machine, RAM) oder auch WHILE-Maschine, also eine Maschine, die WHILE-Programme verarbeiten kann, erweitert durch

- IF ... THEN ... ELSE ... FI
- Multiplikation und Division
- indirekte Adressierung
- arithmetische Operationen wie $\sqrt{n}$, $\sin n$,...

1.1 Referenzmaschine

Wir wählen als Referenzmaschine die Registermaschine (engl. random access machine, RAM) oder auch WHILE-Maschine, also eine Maschine, die WHILE-Programme verarbeiten kann, erweitert durch

- IF ... THEN ... ELSE ... FI
- Multiplikation und Division
- indirekte Adressierung
- arithmetische Operationen wie $\sqrt{n}$, $\sin n$,...

1.1 Referenzmaschine

Wir wählen als Referenzmaschine die Registermaschine (engl. random access machine, RAM) oder auch WHILE-Maschine, also eine Maschine, die WHILE-Programme verarbeiten kann, erweitert durch

- IF ... THEN ... ELSE ... FI
- Multiplikation und Division
- indirekte Adressierung
- arithmetische Operationen wie $\sqrt{n}$, $\sin n$,...

1.2 Zeit- und Platzkomplexität

Beim Zeitbedarf zählt das uniforme Kostenmodell die Anzahl der von der Registermaschine durchgeführten Elementarschritte, beim Platzbedarf die Anzahl der benutzten Speicherzellen.

Das logarithmische Kostenmodell zählt für den Zeitbedarf eines jeden Elementarschrittes

$$\ell(\text{größter beteiligter Operand}),$$

beim Platzbedarf

$$\sum_x \ell(\text{größter in } x \text{ gespeicherter Wert}),$$

also die maximale Anzahl der von allen Variablen x benötigten Speicherbits.

1.2 Zeit- und Platzkomplexität

Beim Zeitbedarf zählt das uniforme Kostenmodell die Anzahl der von der Registermaschine durchgeführten Elementarschritte, beim Platzbedarf die Anzahl der benutzten Speicherzellen.

Das logarithmische Kostenmodell zählt für den Zeitbedarf eines jeden Elementarschrittes

$$\ell(\text{größter beteiligter Operand}),$$

beim Platzbedarf

$$\sum_x \ell(\text{größter in } x \text{ gespeicherter Wert}) \,,$$

also die maximale Anzahl der von allen Variablen x benötigten Speicherbits.

Beispiel 162

Wir betrachten die Prozedur

func $dbexp(n)$
  $m := 2$
  for $i := 1$ **to** n **do**
    $m := m^2$
  do
  return (m)

Die Komplexität von $dbexp$, die $n \mapsto 2^{2^n}$ berechnet, ergibt sich bei Eingabe $n$zu

	Zeit	Platz
uniform	$O(n)$	$O(1)$
logarithmisch	$O(2^n)$	$O(2^n)$

Beispiel 162

Wir betrachten die Prozedur

func $dbexp(n)$
  $m := 2$
  for $i := 1$ **to** n **do**
    $m := m^2$
  do
  return (m)

Die Komplexität von $dbexp$, die $n \mapsto 2^{2^n}$ berechnet, ergibt sich bei Eingabe $n$zu

	Zeit	Platz
uniform	$O(n)$	$O(1)$
logarithmisch	$O(2^n)$	$O(2^n)$

Bemerkung:
Das (einfachere) uniforme Kostenmodell sollte also nur verwendet werden, wenn alle vom Algorithmus berechneten Werte gegenüber den Werten in der Eingabe nicht zu sehr wachsen, also z.B. nur polynomiell.