Grundlagen Theoretischer Informatik 2

WiSe 2011/12 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Grundlagen Theoretischer Informatik 2 Gesamtübersicht

- Ersetzungsverfahren: Grammatiken und Automaten
- Rekursionstheorie-Einführung
- Komplexitätstheorie-Einführung

Formale Sprachen Technische Sicht

Automatentheorie

Endliche Automaten
zur Erkennung von Wörtern oder Sprachen
zur Übersetzung (siehe Mealy- / Moore-Automaten Technische Informatik)

Endliche Automaten mit zusätzlichen Speicherstrukturen
(Zählerautomaten)
Kellerautomaten
Turingmaschinen

Hinweis: (Pro-)Seminar über im kommenden Sommersemester
Freie Themenwahl, “Endliche Automaten” wäre eine mögliche Stoßrichtung.
Bei Interesse bitte Kontaktaufnahme per Email.

Lesen lernen... (aus Wikipedia 2009/10)

Gegeben sei ein durch ein 6-Tupel $(Q, \Sigma, \Omega, \delta, \lambda, q_0)$ definierter, deterministischer *Moore-Automat* mit:

$Q = \{q_0, q_1, q_2, q_3\}$,

$\Sigma = \{x, y, z\}$,

$\Omega = \{a, b, c\}$

und Anfangszustand q_0.

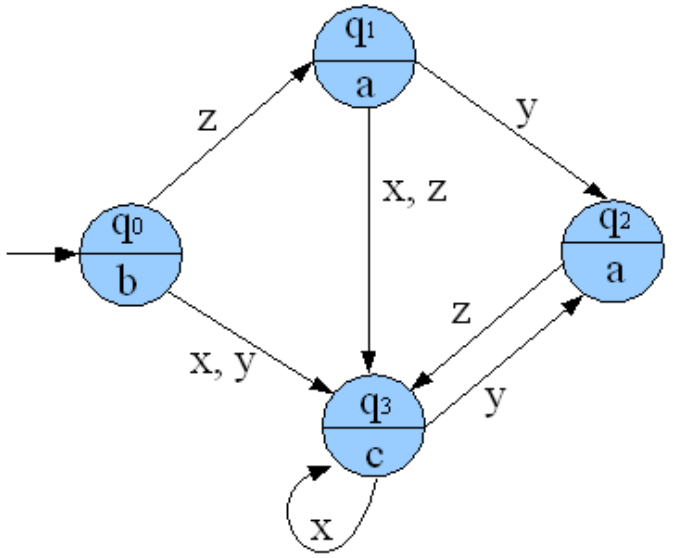
Die Übergangsfunktion δ sowie die Ausgabefunktion λ können

durch einen Graphen bzw.

durch eine Automatentafel

dargestellt werden.

Darstellung von δ und λ durch einen Graphen:



Übersetzende Automaten

kennen Sie (Mealy / Moore) aus “Rechnerstrukturen”
Sie sind also wichtig für die Technische Informatik.

“Bei uns” in der Theorie meist präsent in allgemeinererer Form als so genannte *(rationale) Transduktoren*.

Viele “einfache Reduktionen” der Komplexitätstheorie lassen sich durch übersetzende Automaten realisieren.

Auch bei übersetzenden Automaten ist das Aufsetzen weiterer Speicherstrukturen möglich.
“Übersetzende Turingmaschinen” wurden in der VL betrachtet!

Formale Sprachen Linguistische Sicht

Definition von “Sprachen” ein natürlicher Begriff.

Zur Sprachbeschreibung werden (formale) Grammatiken betrachtet.
Auf welcher Stufe der Chomsky-Hierarchie “liegen” natürliche Sprachen?
Sind Chomsky-Grammatiken für natürliche Sprachen adäquat?
Sprachbeschreibungsmechanismen können deutlich anders aussehen als bei uns: Nicht immer sind Ersetzungssysteme die Grundlage.

Formale Sprachen | Algorithmische Sicht

“Probleme” sind (mit geeigneter Codierung) Formale Sprachen.

Ziel: Klassifikation von Problemen gemäß ihrer Komplexität
$\leadsto$ Gegenstand der Komplexitätstheorie (und für die ganz schwierigen Fälle: auch der Rekursionstheorie)

Durch Codierung sind “Problemklassen” (genannt Komplexitätsklassen) nichts anderes als Klassen Formaler Sprachen.

Umgekehrt eignen sich klassische Bereiche aus den Formalen Sprachen, um leicht algorithmische Probleme zu formulieren, die wiederum selbst klassifiziert werden wollen!

Lesen lernen... (aus einer Vorlesung im Internet; Uni Düsseldorf)

Entscheidbare Fragen

Probleme

- Wortproblem: Gegeben eine reguläre Sprache L und $w \in \Sigma^*$. Gilt $w \in L$?
- Leerheitsproblem: Gegeben eine reguläre Sprache L. Gilt $L = \emptyset$?
- Endlichkeitsproblem: Gegeben eine reguläre Sprache L. Ist L endlich?
- Schnittproblem: Gegeben zwei reguläre Sprachen L_1, L_2. Gilt $L_1 \cap L_2 = \emptyset$?
- Inklusionsproblem: Gegeben zwei reguläre Sprachen L_1, L_2. Gilt $L_1 \subseteq L_2$?
- Äquivalenzproblem: Gegeben zwei reguläre Sprachen L_1, L_2. Gilt $L_1 = L_2$?

Lesen lernen. . . (aus einer Vorlesung im Internet; Uni Düsseldorf)

Was soll das bedeuten?

Wie sind “reguläre Sprachen gegeben” ??

Wir erinnern verschiedene mögliche “Eingabe-Formate”:
NEAs / NFAs
DEAs / DFAs
reguläre Ausdrücke
rechtslineare Grammatiken

Lesen wir also weiter, um zu verstehen, was gemeint sein könnte. . .

Lesen lernen... (aus einer Vorlesung im Internet; Uni Düsseldorf)

Entscheidbare Fragen

Wortproblem ($w \in L$?)

Gegeben sind L und $w \in \Sigma^*$.

Lösung: Bestimme einen DFA M für L und verfolge die Zustandsübergänge von M, wie durch w vorgegeben.
Endzustand wird erreicht $\rightsquigarrow$ $w \in L$
Nicht-Endzustand wird erreicht $\rightsquigarrow$ $w \notin L$

Lesen lernen... (aus einer Vorlesung im Internet; Uni Düsseldorf)

Was soll das bedeuten?

Offenbar werden doch “reguläre Sprachen” eingegeben in unseren Algorithmus.

Das klingt seltsam (und ist genau genommen falsch).

Aber wir sehen auch: Hier werden DFAs als “Eingabe” angesehen.

Das folgende Problem lässt sich also in deterministischer Polynomzeit lösen:
Gegeben sei eine geeignete Codierung eines DFAs A und ein Wort w über dem Eingabealphabet von A. Entscheide, ob $w \in L(A)$ liegt.

Dies nennt man auch *uniformes Wortproblem*.

Lesen lernen... (aus einer Vorlesung im Internet; Uni Düsseldorf)

Entscheidbare Fragen

Leerheitsproblem ($L = \emptyset$?)

Gegeben ist L.

Lösung: Bestimme einen NFA M für L.

$L = \emptyset \iff$ es gibt keinen Pfad von einem Start- zu einem Endzustand.

Lesen lernen... (aus einer Vorlesung im Internet; Uni Düsseldorf)

Was soll das bedeuten?

Jetzt geht man wohl von NFAs als Eingaben aus.

Würden DFAs irgendetwas einfacher machen?!

Hinweis: Dijkstra-Algorithmus zum Auffinden kürzester Wege in Graphen.

Lesen lernen... (aus einer Vorlesung im Internet; Uni Düsseldorf)

Entscheidbare Fragen

Endlichkeitsproblem (Ist L endlich?)

Gegeben ist L.

Lösung: Bestimme einen NFA M für L.
L ist unendlich
$\iff$ in M gibt es unendlich viele Pfade von einem Start- zu einem Endzustand
$\iff$ es gibt einen erreichbaren Zyklus in M, von dem aus wiederum ein Endzustand erreichbar ist.

Lesen lernen... (aus einer Vorlesung im Internet; Uni Düsseldorf)

Was soll das bedeuten?
OK: Hier wieder NFAs als Eingaben...

Wie kann man solche Kreise denn finden?!

Dies ist genau das, was das reguläre Pumping-Lemma "leistet":
Wenn $L(A)$ unendlich ist, lassen sich "hinreichend lange Wörter" $w \in L(A)$ zerlegen in: $w = xyz$, sodass y einen Kreis beschreibt im Automatengraphen, in den x hineinführt und z herausführt.
$\rightsquigarrow \forall n \geq 0$: $xy^n z \in L(A)$.
Und wenn für gewisse x, y, z gilt: $\forall n \geq 0$: $xy^n z \in L(A)$,
so ist $L(A)$ natürlich unendlich!

Lesen lernen... (Wir springen ein wenig...)

Entscheidbare Fragen

Effizienzgesichtspunkte:

Je nach dem in welcher Darstellung eine Sprache L gegeben ist, kann die Komplexität der oben beschriebenen Verfahren sehr unterschiedlich ausfallen.

Beispiel Äquivalenzproblem:

- L_1, L_2 gegeben als DFAs $\rightsquigarrow$ Komplexität $O(n^2)$
  (d.h., quadratisch viele Schritte in der Größe der Eingabe)
- L_1, L_2 gegeben als Grammatiken, reguläre Ausdrücke oder NFAs $\rightsquigarrow$ Komplexität NP-hart
  Das bedeutet unter anderem: es ist nicht bekannt, wie dieses Problem effizient gelöst werden kann.
- Mehr zum Thema Komplexität in der Vorlesung „Berechenbarkeit und Komplexität“ im Wintersemester.

Lesen lernen... (aus einer Vorlesung im Internet; Uni Düsseldorf)

Was soll das bedeuten?

Hier wird bei DFAs auf das (in GTI1 nur kurz) besprochene Zustandsminimieren bei DFAs angesprochen.

Dadurch erhält man eine Normalform, bei der "leicht" auf Äquivalenz getestet werden kann.

Weshalb ist denn das NICHT-Äquivalenzproblem in NP ?
$\leadsto$ Guess and Check (für einen Zeugen für die Nicht-Äquivalenz)

Alles zu abstrakt?

Anwendungsbeispiel: Systemverifikation

Dabei liegt die Sprache L_{spez} der korrekten Programmverläufe vor (Spezifikation), und es soll geprüft werden, ob das vorzuliegende (technische) System Programmläufe liefert, die der Spezifikation genügen, d.h.: $L_{spez} = L_{sys}$ oder wenigstens $L_{sys} \subseteq L_{spez}$.

Bei kleinen Systemen können die (teilweise) bekannten Techniken für reguläre Sprachen helfen!

Lesen lernen... (Weiter im Text...)

Anwendung: Verifikation

Ein abschließendes ausführliches Beispiel:

- Wir betrachten zwei Prozesse P_1, P_2, die auf eine gemeinsame Ressource zugreifen wollen.
- Jeder Prozess hat einen sogenannten kritischen Bereich, in dem auf die Ressource zugegriffen wird. Es darf sich jeweils nur ein Prozess im kritischen Bereich befinden.
- Es stehen gemeinsame Variable zur Verfügung, über die sich die Prozesse synchronisieren können. Diese Variablen sind jedoch keine Semaphore, d.h., eine atomare Operation, bei der gleichzeitig gelesen und geschrieben wird, ist nicht möglich.

Wir möchten zeigen, dass der wechselseitige Ausschluß gewährleistet ist und dass gewisse Fairnessbedingungen (jeder Prozess kommt irgendwann an die Reihe) eingehalten werden.

Lesen lernen... (Wechselseitiger Ausschluss 1. Versuch)

Anwendung: Verifikation

Versuch 1: Beide Prozesse P_1, P_2 verwenden eine gemeinsame Boolesche Variable f, die mit false initialisiert wird.

Programmcode für P_1, P_2

```
   while true do
1:    if (f = false?) then
2:       f := true
3:       Betrete kritischen Bereich
         ...
4:       Verlasse kritischen Bereich
5:       f := false
      end
   end
```

Lesen lernen... (Wechselseitiger Ausschluss 1. Versuch)

Anwendung: Verifikation

Wir verwenden folgendes Alphabet, bestehend aus den Programm-Befehlen und den Abfragen der Booleschen Variablen:

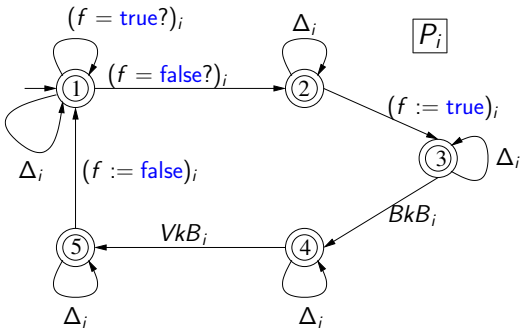
$$\begin{aligned}\Sigma = \{&(f := \text{true})_i, (f := \text{false})_i, (f = \text{true}?)_i, (f = \text{false}?)_i \\ &\mid i \in \{1,2\}\} \\ &\text{(Synchronisation von Prozess } i \text{ mit Variable } f\text{)} \\ \cup \{&BkB_i, VkB_i \mid i \in \{1,2\}\} \\ &\text{(Prozess } i \text{ betritt/verläßt kritischen Bereich).}\end{aligned}$$

Der Index $i \in \{1,2\}$ gibt an, ob die jeweilige Aktion vom ersten oder vom zweiten Prozess ausgeführt wird.

Lesen lernen... (Wechselseitiger Ausschluss 1. Versuch)

Anwendung: Verifikation

Beschreibung der Abläufe eines Prozesses i als endlicher Automat:



mit $\Delta_i = \{(f{:=}\text{true})_i, (f{:=}\text{false})_i, (f{=}\text{true}?)_i, (f{=}\text{false}?)_i BkB_j, VkB_j\}$
wobei $j = 2$, falls $i = 1$, und $j = 1$, falls $i = 2$.

Anwendung: Verifikation

Bemerkungen:

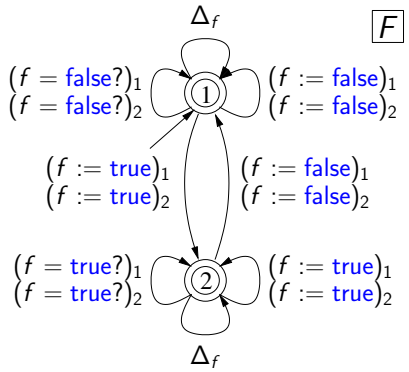
- Bedeutung der Zustände 1, 2, 3, 4, 5: diese entsprechen den mit Labels markierten Programmzeilen
- Bedeutung der Schleifen mit Alphabetsymbolen aus Δ_i: da wir später das Kreuzprodukt bilden werden, um mehrere Automaten zu synchronisieren, dürfen Übergänge anderer Automaten, die den Prozess i nicht betreffen, nicht ausgeschlossen werden. Sie werden einfach „mitgehört“ und haben keinen Einfluss auf die Zustandsübergänge.

Etwas stimmt hier nicht! Wie sind denn Δ_i etc. definiert?!
Die Indizes in der Mengenklammer müssen konsequent j sein, sonst stimmt das “Mithören” nicht!
“Kreuzprodukt”: Produktmenge der Zustandsmengen wie beim Schnitt-Abschluss reg. Sprachen.

Lesen lernen... (Wechselseitiger Ausschluss 1. Versuch)

Anwendung: Verifikation

Beschreibung der Booleschen Variable f durch einen Automaten:

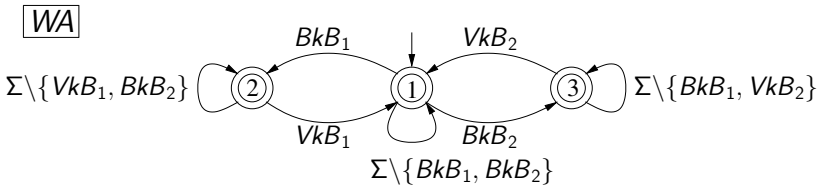


mit $\Delta_f = \{BkB_1, VkB_1, BkB_2, VkB_2\}$.

Anwendung: Verifikation

Die Sprache aller Abläufe des Gesamtsystems ist $T(P_1) \cap T(P_2) \cap T(F)$.

Der Automat WA, der alle Abläufe beschreibt, die den wechselseitigen Ausschluß erfüllen (beide Prozesse sind nicht gleichzeitig im kritischen Bereich) sieht folgendermaßen aus:



Damit ist zu zeigen $T(P_1) \cap T(P_2) \cap T(F) \subseteq T(WA)$. (Wir überprüfen das mit Hilfe des Tools Grail.)

Hinweis: Grail: `http://www.csd.uwo.ca/Research/grail/`

Lesen lernen... (Wechselseitiger Ausschluss 1. Versuch)

Anwendung: Verifikation

```
$ fmcross p1.aut < p2.aut > psynch.aut
$ fmcross f.aut < psynch.aut > sys.aut
$ fmcment wa.aut > wa-cment.aut
$ fmcross sys.aut < wa-cment.aut > errors.aut
$ fmenum errors.aut
DdAXax
DdAaXx
DdAaxX
DdaAXx
[...]
```


Anwendung: Verifikation

Die entstehende Sprache ist nicht leer! Es gibt also Abläufe, die die Bedingung des wechselseitigen Ausschluss verletzen.

Einer davon ist `DdAXax`. Übersetzt ins ursprüngliche Alphabet:

$$(f = \text{false}?)_2\ (f = \text{false}?)_1\ (f := \text{true})_2\ BkB_2\ (f := \text{true})_1\ BkB_1.$$

Grund für die Verletzung des wechselseitigen Ausschlusses: Es gibt keine atomare Schreib- und Leseoperation. Daher können beide Prozessen nacheinander die Variable auslesen, anschließend setzen beide die Variable und betreten den kritischen Bereich.

Der Algorithmus ist also falsch!

Wir haben also einen Fehler gegenüber der Spezifikation gefunden!
Wir mussten in Grail die Eingabezeichen umdefinieren...

Lesen lernen... (Wechselseitiger Ausschluss 2. Versuch von Leslie Lamport)

Anwendung: Verifikation

Prozess P_1:

```
   while true do
1:     f1 := true
2:     while (f2 = true?) do
           skip
       end
3:     Betrete kritischen Bereich
       ...
4:     Verlasse kritischen Bereich
5:     f1 := false
   end
```

skip : Null-Operation (hat keine Auswirkungen)

Anwendung: Verifikation

Prozess P_2

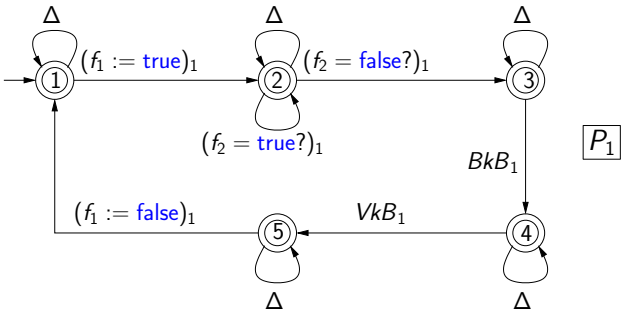
```
   while true do
1:     f2 := true
2:     if (f1 = true?) then do
3:         f2 := false
4:         while (f1 = true?) do skip end
       else
5:         Betrete kritischen Bereich
           ...
6:         Verlasse kritischen Bereich
7:         f2 := false
       end
   end
```


Es werden zwei Boolesche Variablen benötigt, um wechselseitigen Ausschluss zu garantieren.

Lesen lernen... (Wechselseitiger Ausschluss 2. Versuch von Leslie Lamport)

Anwendung: Verifikation

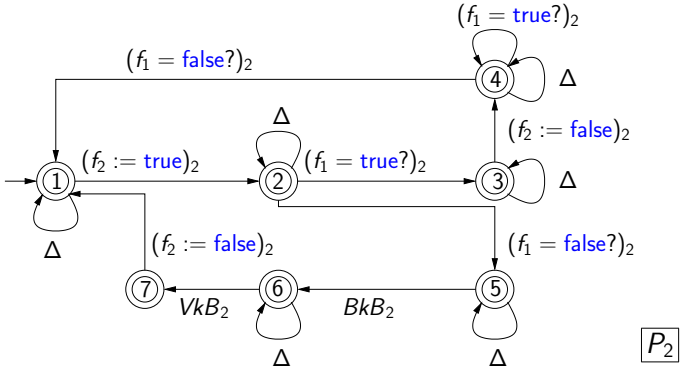
Automat für den Prozess P_1:



Dabei gilt, dass Δ all mögliche Aktionen von Prozess 2 enthält.

Anwendung: Verifikation

Automat für den Prozess P_2:



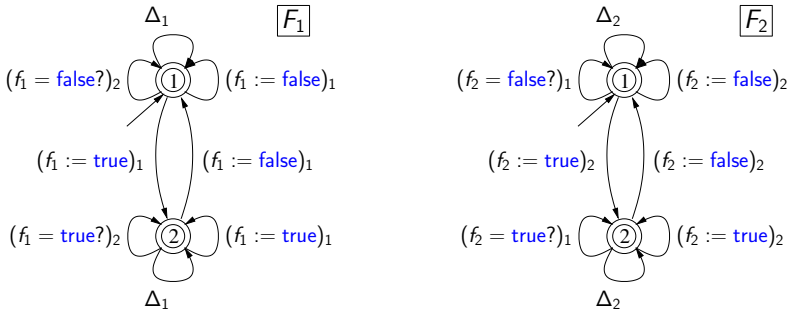
Dabei gilt, dass Δ all mögliche Aktionen von Prozess 1 enthält.

Die zugehörigen endlichen Automaten sehen schon komplizierter aus.

Lesen lernen... (Wechselseitiger Ausschluss 2. Versuch von Leslie Lamport)

Anwendung: Verifikation

Automaten für die beiden Variablen:



$$\Delta_1 = \{(f_2 := \text{true})_2, (f_2 := \text{false})_2, (f_2 = \text{true}?)_1, (f_2 = \text{false}?)_1 \\ BkB_1, BkB_2, VkB_1, VkB_2$$

Analog für Δ_2.

Anwendung: Verifikation

In diesem Fall ist der wechselseitige Ausschluss erfüllt, d.h., es gilt $T(P_1) \cap T(P_2) \cap T(F_1) \cap T(F_2) \subseteq T(WA)$.

Überprüfung mit Grail:

```
$ fmcross p1.aut < p2.aut > psynch.aut
$ fmcross f1.aut < psynch.aut  > psynch1.aut
$ fmcross f2.aut < psynch1.aut  > sys.aut
$ fmcment wa.aut > wa-cment.aut
$ fmcross sys.aut < wa-cment.aut > errors.aut
$ fmenum errors.aut
[keine Ausgabe]
```

Lesen lernen... Zusammenfassung

Anwendung: Verifikation

Zusammenfassung Verifikation:

- Wir haben mit Hilfe von endlichen Automaten zwei Protokolle modelliert, die wechselseitigen Ausschluss realisieren sollen.
- Mit Hilfe der Lösungsverfahren für das Inklusions- bzw. Schnittproblem konnten wir überprüfen, ob diese Protokolle tatsächlich wechselseitigen Ausschluss realisieren.

Das bedeutet: die vorgestellten Verfahren können zur Programmverifikation eingesetzt werden.

Bemerkung: Bei realen Programmen hat man allerdings noch damit zu kämpfen, dass der Zustandsraum des Programms unendlich ist. Damit wird vieles unentscheidbar und muss durch approximative Verfahren gelöst werden.

Moment: Gab es da nicht Unentscheidbarkeitsaussagen bei der Verifikation?!

Eine formalsprachliche Anwendung des PCP aus VL 10:

Unter dem *Schnittleerheitsproblem* einer Grammatikfamilie $\mathcal{G}$ versteht man:
Gegeben $G_1, G_2 \in \mathcal{G}$, gilt $L(G_1) \cap L(G_2) = \emptyset$?

Satz: Das Schnittleerheitsproblem für Typ-2-Grammatiken ist unentscheidbar.

Beweisidee:

Gegeben Postsches Korrespondenzproblem P über $\{0, 1\}$ mit

$$P = \{(x_1, y_1), ..., (x_k, y_k)\}$$

Konstruiere zwei kontextfreie Grammatiken $G_1 = (N_1, T, P_1, S)$ und $G_2 = (N_2, T, P_2, S)$ wie folgt:

$$N_1 = \{S, A, B\}, \quad N_2 = \{S, R\}, \quad T = \{0, 1, \$, a_1, ..., a_k\}$$

...

Die Grammatik G_1 besitzt die Ableitungsregeln P_1

$$\begin{aligned} S &\rightarrow A\$B \\ A &\rightarrow a_1Ax_1 \mid ... \mid a_kAx_k \mid a_1x_1 \mid ... \mid a_kx_k \\ B &\rightarrow y_1^rBa_1 \mid ... \mid y_k^rBa_k \mid y_1^ra_1 \mid ... \mid y_k^ra_k \end{aligned}$$

wobei mit w^r das gespiegelte Wort zu w bezeichnet wird.

Diese Grammatik G_1 erzeugt die Sprache

$$L_1 = \{a_{i_1}...a_{i_n}x_{i_n}...x_{i_1}\$y_{j_1}^r...y_{j_m}^r a_{j_m}...a_{j_1} \mid n, m \geq 1, i_\nu, j_\mu \in \{1, ..., k\}\}$$

Wir führen eine zweite Grammatik G_2 mit den folgenden Regeln P_2 ein:

$$\begin{aligned} S &\rightarrow a_1Sa_1 \mid ... \mid a_kSa_k \mid R \\ R &\rightarrow 0R0 \mid 1R1 \mid \$ \end{aligned}$$

Sie erzeugt die Sprache

$$L_2 = \{u\nu\$\nu^ru^r \mid u \in \{a_1, ..., a_k\}^*, \nu \in \{0, 1\}^*\}$$

Beide Sprachen sind übrigens sogar deterministisch kontextfrei.

Weitere formalsprachliche Anwendungen des PCP:

Unter dem *Schnittunendlichkeitsproblem* einer Grammatikfamilie $\mathcal{G}$ versteht man: Gegeben $G_1, G_2 \in \mathcal{G}$, gilt $|L(G_1) \cap L(G_2)| = \infty$?

Satz: Das Schnittunendlichkeitsproblem für Typ-2-Grammatiken ist unentscheidbar.

Beweisidee: Verwende gleiche Konstruktion $P \mapsto (G_1, G_2)$ wie soeben:

$$\begin{aligned} P \in PCP \quad &\Leftrightarrow \quad \text{es gibt eine Lösung } \mathcal{I} = (i_1, ..., i_n) \text{ für } P \\ &\Leftrightarrow \quad \text{es gibt } \infty\text{-viele Lösungen } \mathcal{I}, \mathcal{I}\mathcal{I}, \mathcal{I}\mathcal{I}\mathcal{I}, ... \text{ für } P \\ &\Leftrightarrow \quad |L_1 \cap L_2| = \infty \\ &\rightsquigarrow \quad (L_1 \cap L_2 \neq \emptyset \quad \Leftrightarrow \quad P \in PCP) \end{aligned}$$

Weitere formalsprachliche Anwendungen des PCP:

Unter dem *Schnittkontextfreiheitsproblem* einer Grammatikfamilie $\mathcal{G}$ versteht man: Gegeben $G_1, G_2 \in \mathcal{G}$, gilt $L(G_1) \cap L(G_2)$ ist vom Typ 2 ?

Satz: Das Schnittkontextfreiheitsproblem für Typ-2-Grammatiken ist unentscheidbar.

Beweis; Wieder gleiche Konstruktion $P \mapsto (G_1, G_2)$, setze $L := L_1 \cap L_2$.

Jedes Wort in L hat Form $ab\$c^r d^r$, mit $a = d$, $b = c$ und 'Index-Übereinstimmung' zwischen a und b bzw. c und d

Also: a oder d bekannt $\Rightarrow$ $ab\$c^r d^r$ eindeutig festgelegt!

Behauptung: Falls $|L| = \infty$, dann L ist nicht kontextfrei.

Annahme: L sei kontextfrei (und unendlich), d.h. Pumping-Lemma für kontextfreie Sprachen sei anwendbar, n sei die Pumpingzahl.

Wähle $ab\$c^r d^r \in L$ mit $n < |a| = |d| \leq |b| = |c|$

Betrachte beliebige Zerlegung $ab\$c^r d^r = uvwxy$ mit
$uwy = uv^0wx^0y \in L$, $|vx| \neq 0$ und $|vwx| \leq n$

Fall (1): $|u| \geq |a|$: damit a Präfix von uwy,
aber b, c, d durch a festgelegt
$\Rightarrow$ Widerspruch zu $|vx| > 0$ und $uwy \in L$.

Fall (2): $|u| < |a|$: damit $|uvwx| < |ab|$ und d Suffix von uwy,
aber a, b, c durch d festgelegt
$\Rightarrow$ wieder Widerspruch zu $|vx| > 0$ und $uwy \in L$.

Damit also: $|L| = \infty \Rightarrow L$ nicht kontextfrei

Zusammen:

$$\begin{array}{lcll} P \in PCP & \Rightarrow & |L| = \infty & \Rightarrow \quad L \text{ nicht kontextfrei} \\ P \notin PCP & \Rightarrow & L = \emptyset & \Rightarrow \quad L \text{ kontextfrei} \end{array}$$

Inklusionsproblem und Äquivalenzproblem sind ebenso unentscheidbar für kontextfreie Grammatiken:

Nutze, dass L_1, L_2 deterministisch kontextfrei sind.
Deterministisch kontextfreie Sprachen sind effektiv unter Komplementbildung abgeschlossen:

> Zu G_1 und G_2 kann man effektiv Grammatiken G_1' und G_2' konstruieren mit $L(G_1') = \text{Komplement}(L_1)$ und $L(G_2') = \text{Komplement}(L_2)$.

Dann folgt

$$L_1 \cap L_2 = \emptyset \quad \Leftrightarrow \quad L_1 \subseteq L(G_2')$$

D.h. $P \mapsto (G_1, G_2')$ reduziert PCP auf das Inklusionsproblem kontextfreier Sprachen.
Aus G_1 und G_2' konstruiere G_3 mit $L(G_3) = L_1 \cup L(G_2')$, dann

$$\begin{aligned} L_1 \cap L_2 = \emptyset \quad &\Leftrightarrow \quad L_1 \cup L(G_2') = L(G_2') \\ &\Leftrightarrow \quad L(G_3) = L(G_2') \end{aligned}$$

D.h. $P \mapsto (G_3, G_2')$ reduziert PCP auf das Äquivalenzproblem kontextfreier Sprachen.

Viel Erfolg bei der Prüfung!