0 Wer, Was, Wann

Vorlesungen:

- Termersetzungssysteme (Giesl, SS 04)
- Compilerbau (Indermark, SS 04)
- Logikprogrammierung (Indermark, WS 03/04)

Prüfer: Prof. Dr. Indermark

Beisitzer: Herr Bollig

Datum der Prüfung: 14.10.2004

Prüfling: Matthias Raffelsieper

Bemerkung: Das folgende Protokoll entspricht sicherlich nicht dem wirklich Gesagtem. Es kann durchaus sein, dass ich hier einige Teile verkürzt oder anders wiedergegeben habe.
Alle fett gedruckten Texte sind von Herrn Indermark, normal Gedrucktes von mir.

1 TES

- **Fangen wir direkt mal mit Termersetzung an.**
  Was war denn das Anliegen?

  ➠ Das Wortproblem $s \equiv_{\mathcal{E}} t$ zu lösen

- **Was heißt denn das?**

  ➠ Jedes Modell der Gleichungsmenge $\mathcal{E}$ ist auch Modell von $s \equiv t$

- **Das kann man ja nicht wirklich automatisieren. Dazu benutzt man dann ja die Ersetzungsrelation. Wie sieht die denn aus?**

  ➠ Aufgeschrieben:
  Es ist $s \rightarrow_{\mathcal{E}} t$, gdw. $\exists \pi \in \mathrm{Occ}(s)$ mit $s|_\pi = l\sigma$ [**Was ist denn** l, **und wo kommt** σ **her?** Ich mach erst mal weiter, dann erklär ich das...],
  $t = s[r\sigma]_\pi$ für ein $l \equiv r \in \mathcal{E}$, $\sigma \in \mathrm{SUB}(\Sigma, \mathcal{V})$

- **Aha, da steht es ja. Aber gilt da nicht $l \equiv r$ oder $r \equiv l \in \mathcal{E}$?**

  ➠ Ich meine, Herr Giesl hätte das so definiert, da bildet man dann ja die transitiv, reflexiv, symmetrische Hülle von; die hat dann quasi $l \equiv r$ oder $r \equiv l \in \mathcal{E}$.

- **Hmm, na gut, ich weiß jetzt nicht wie Herr Giesl das gemacht hat, ist ja letztendlich auch egal. Denn uns interessiert ja $\leftrightarrow^*_{\mathcal{E}}$. Da gibt es ja den Satz von Birkhoff zu, was sagt der denn?**

  ➠ $\equiv_{\mathcal{E}} = \leftrightarrow^*_{\mathcal{E}}$

- **Ja. Aber der Suchraum ist ja immernoch ziemlich groß. Wie versuchen wir denn, das Ganze praktikabel zu machen?**

➠ Wir suchen ein zu $\mathcal{E}$ äquivalentes, konvergentes TES $\mathcal{R}$.

- **Was heißt äquivalent?**

  ➠ korrekt und adäquat:
  Wenn alle linken Seiten von Gleichungen in $\mathcal{E}$ mit der transitiv, reflexiv, symmetrischen Hülle der Relation, die von $\mathcal{R}$ beschrieben wird, ineinander überführbar sind;
  und wenn die linken Seiten aller Regeln in $\mathcal{R}$ in $\equiv_{\mathcal{E}}$-Relation stehen.

- **OK, was heißt konvergent?**

  ➠ Ein TES ist konvergent, wenn es terminierend und konfluent ist.

- **Was heißt Konfluenz?**

  ➠ Die Indeterminismen sind zusammenführbar.

- **Nehmen wir mal an, wir haben ein solches TES. Wie prüft man nun $s \equiv_{\mathcal{E}} t$?**

  ➠ Man bildet die Normalformen von s und t. Wenn $s \downarrow = t \downarrow$, dann gilt $s \equiv_{\mathcal{E}} t$.

- **Warum gilt denn da, dass wegen $s \equiv_{\mathcal{E}} t$, bzw. $s \leftrightarrow^*_{\mathcal{E}} t$ auch $s \downarrow = t \downarrow$ gilt?**

  ➠ Naja, hmm, weil bei Konfluenz...

- **Nein, nicht Konfluenz, sondern was wird da benutzt?**

  ➠ Achso, Church-Rosser-Eigenschaft.

- **Was besagt die, und warum können wir die hier benutzen?**

  ➠ Wenn für zwei Terme $s \leftrightarrow^*_{\mathcal{E}} t$ gilt, dann gilt auch $s \downarrow t$. Und es gibt einen Satz, der besagt das Church-Rosser und Konfluenz äquivalent sind.

- **Denn wollen wir doch mal beweisen. Die eine Richtung ist ja trivial. Welche ist denn das?**

  ➠ Church-Rosser $\curvearrowright$ Konfluenz

- **Genau. Und was ist andersherum?**

  ➠ Wir haben Konfluenz, also

$$\begin{tikzcd}
 & p \arrow[dl, "*"'] \arrow[dr, "*"] & \\
s \arrow[dr, "*"] & & t \arrow[dl, "*"'] \\
 & q &
\end{tikzcd}$$

  Zu zeigen ist $s \leftrightarrow^* t \curvearrowright s \downarrow t$
  Wir machen Induktion über die Länge der Herleitung, also über $s \leftrightarrow^n_{\mathcal{E}} t$.
  [Ja, wie sieht da denn der Induktionsschritt aus?]
  Also wir haben dann ja $s \leftrightarrow^n s' \leftrightarrow t$.
  Nach IV existiert dann $s \rightarrow^* q^* \leftarrow s'$. Dann zwei Fälle:

  * $s' \leftarrow t$ ist trivial, weil $t \rightarrow^* q$
  * $s' \rightarrow t$ gilt, weil q und t einen gemeinsamen Vorgänger s' haben. Daher muss wegen Konfluenz q' ex. mit $s' \rightarrow^* q \rightarrow^* q'^* \leftarrow t$

- **OK. Gibt es denn immer ein solches äquiv. konvergentes TES?**

➠ Nein, denn das Wortproblem ist im Allgemeinen ja unentscheidbar.

- **Können Sie mir denn eine Klasse von Problemen nennen, wo das Wortproblem entscheidbar ist?**

  ➠ Die Menge der Probleme, bei denen keine Variablen in den Gleichungen vorkommen. Da kann das Wortproblem mit dem Kongruenzabschluß gelöst werden.

- **Wie funktioniert der Kongruenzabschluß?**

  ➠ Man bildet den Kongruenzabschluß nur bezüglich der Menge S der Subterme, dies sind alle Teilterme der Terme auf linken und rechten Seiten der Gleichungsmenge sowie der zu untersuchenden Gleichung.
  Da $\equiv_{\mathcal{E}}$ reflexiv, symmetrisch, transitiv [**Was ist das also für eine Relation?** Eine Äquivalenzrelation, bzw. weil auch noch kongruent eine Kongruenzrelation.] stabil (was uns hier aber nicht interessiert, da wir ja eben keine Variablen haben) und monoton (und damit kongruent) ist, bilden wir schrittweise die Gleichungen, die aufgrund einer dieser Eigenschaften aus einer Gleichungsmenge folgen. Die resultierende Gleichungsmenge wird dann mit $S \times S$ geschnitten, daher terminiert das Verfahren immer.

2 Logikprogrammierung

- **Dann gehen wir mal zur Logikprogrammierung. Schreiben Sie mir mal die FO-Resolution auf.**

  ➠ Also seien K_1 und K_2 zwei Klauseln. Dann brauchen wir zwei Variablenumbenennungen s_1 und s_2 [**Ja, hmm, wir könnten auch voraussetzen, daß die Klauseln variablenfremd sind, aber ok**] und erhalten dann zwei Klauseln $K_1 s_1$ und $K_2 s_2$, die keine gemeinsamen Variablen mehr haben.
  Dann existieren $\{\bar{L_1}, \ldots, \bar{L_n}\} \subseteq K_1 s_1$ und $\{L'_1, \ldots, L'_m\} \subseteq K_2 s_2$ die unifizierbar sind [**Ja, nur funktioniert das ja noch nicht so ganz. Da haben sie ja noch die Negation in der ersten Menge...** Die muss man dann weglassen, sonst kann natürlich nicht unifiziert werden. **Was steht da noch?** Da wollte ich eigentlich gerade die Unifikation hinschreiben (habe ich dann auch gemacht)]
  es ex. mgu σ mit $|\{L_1, \ldots, L_n, L'_1, \ldots, L'_m\}\sigma| = 1$ [**Ah, da haben Sie ja auch die Negation berücksichtigt.**]
  Die Resolvente ist dann

$$R = ((K_1 s_1 \backslash \{\bar{L_1}, \ldots, \bar{L_n}\}) \cup (K_2 s_2 \backslash \{L'_1, \ldots, L'_m\}))\sigma$$

- **Wir haben uns dann ja auf Horn-Logik beschränkt. Was zeichnet denn die Horn-Logik aus?**

  ➠ Dort haben wir nur negative und definite Klauseln, also in jeder Klausel höchstens ein positives Literal.

- **(irgendwas mit SLD-Baum)**

  ➠ Dort sind die Knoten mit Konfigurationen beschriftet und ein Konfiguration ist direktes Kind eines anderen, wenn ich mit einem kanonischen Resolutionsschritt dort angelange.

- **Bei der SLD-Resolution beschränkt man sich ja auf Hornlogik; die SLD-Resolution ist ja eine Spezialform der linearen Resolution. Was ist denn da der Unterschied?**

➠ Die SLD-Resolution ist eine lineare Resolution, die mit einer negativen Klausel startet. Diese ist dann nur mit einer definiten Klausel unifizierbar, da sonst keine positiven Literale auftreten. Das Resultat ist dann wieder eine negative Klausel, daher haben wir bei der SLD-Resolution auch keine Rückgriffe auf vorherige Resolventen, mit denen dann resolviert wird.

- **Wir hatten immer wieder mit Unifikation zu tun. Ist die eigentlich entscheidbar?**

  ➠ Ja, mit dem Algorithmus von Robinson.

- **Warum terminiert der immer?**

  ➠ Bei jedem Schleifendurchlauf wird die Zahl der Konflikte kleiner, oder aber es wird ein Clash- oder Occur-Failure festgestellt.

- **Was meinen Sie mit Konflikte? Wie funktioniert denn der Algorithmus?**

  ➠ Es wird gleichzeitig durch die Terme gelaufen und angehalten, wenn zwei Symbole verschieden sind. Wenn kein Fehler besteht, so ist eine der Positionen eine Variable. Der Unifikator wird dann um die Substitution der Variablen durch den entsprechenden Term verlängert. Also ist ein Konflikt aufgelöst und damit weniger.

- **Naja, es wird die Zahl der Variablen in jedem Schleifendurchlauf kleiner.** (diese Antwort war wohl gesucht)

- **Bei der prozeduralen Semantik haben wir ja die Unifikation nicht wie im allgemeinen Fall sondern eine Einschränkung...**

  ➠ Dort haben wir nur binäre Resolution, die aber für Horn-Logik vollständig ist.

- **Es gibt dort aber zwei Nichtdeterminismen, welche?**

  ➠ Zum einen kann ausgewählt werden, mit welchem negtiven Literal der Zielklausel resolviert werden soll. Dies ist aber unerheblich aufgrund des Vertauschungslemmas. Zum anderen kann die definite Klausel gewählt werden, mit der resolviert wird.

- **Kommen wir dann mal zur Anwendung... Ach nein, machen wir erst noch was davor: Bei der Universalität von Logikprogrammen haben wir die Komposition von Funktionen betrachtet. Gegeben seien zwei einstellige Funktionen $f, g : \mathbb{N} \to \mathbb{N}$. Wir wollen jetzt $h(x) := (g \circ f)(x) = g(f(x))$ berechnen. Für g und f seien die Logikprogramme π_f und π_g gegeben.**

  ➠ In Logikprogrammen muss Funktionen durch ihre Graphen darstellen, n-stellige Funktionen werden dabei zu $n+1$-stelligen Prädikaten. Wir haben also

  ```
  f(X,Y) % in den jeweiligen
  g(X,Y) % Programmen
  h(X,Y) :- f(X,Y'), g(Y',Y).
  ```

- **In der Vorlesung hatten wir ja DCGs als Darstellung von CFGs als Logikprogramm betrachtet. Was passiert dabei mit den Nichtterminalsymbolen?**

  ➠ Die Nichtterminalsymbole werden zu zweistelligen Prädikaten...

- **Betrachten wir ersteinmal den einfachen Fall, der in der Vorlesung zuerst behandelt wurde. Dort sind diese Prädikate ja einstellig. Was bedeuten diese?**

➠ Ein solches Prädikat ist erfüllt, wenn das Wort W, das als Argument übergeben wird, von diesem Nichtterminalsymbol aus ableitbar ist.

- **Und warum ist das schlecht?**

  ➠ Weil wir dabei viele append-Operationen ausführen müssen.

- **Daher benutzt Prolog ja Differenzlisten. Diese werden aber nicht wie bei uns dargestellt, sondern Prolog macht das wie?**

  ➠ Prolog benutzt ein zweites Argument, das den Subtrahend der Differenzliste enthält.

- **Was bedeutet es hier, wenn ein Prädikat erfüllbar ist?**

  ➠ Dann ist ein Wort w von diesem Nichtterminalsymbol aus ableitbar, das in der Darstellung als Differenzliste das erste Argument minus das zweite Argument ist.

- **Welche Beziehung gilt dabei für das zweite Argument?**

  ➠ (Was ich da gesagt habe weiß ich überhaupt nicht mehr, ich habe irgendwas von „Fortsetzung“ und „hinterer Teil“ erzählt, was aber alles nicht gewünscht war.)
  [**Ein Wort kann doch Präfix eines anderen sein...**]
  Aha, die zweite Liste ist ein Suffix der ersten.
  [**Ja, genau.**] (das Wort Suffix war wohl gefragt)

3 Compilerbau

- **Gehen wir nun zum Compilerbau. Dort hatten wir ja die Bottom-Up Analyse. Wie kann man für eine Grammatik feststellen, ob diese in LR(1) ist?**

  ➠ Man bildet die LR(1)-Mengen und testet diese auf Konfliktfreiheit.

- **Wie viele Mengen gibt es denn?**

  ➠ Es gibt nur endlich viele verschiedene.

- **Ja. Welche Konflikte können denn auftreten?**

  ➠ Zum einen ein Shift-Reduce- und zum anderen ein Reduce-Reduce-Konflikt.
  Ein Shift-Reduce-Konflikt liegt vor, wenn in einer LR(1)-Menge Auskünfte der Form

  $$[A \to \beta_1 \cdot a\beta_2, x]$$
  $$[A' \to \gamma\cdot, a]$$

  enthalten sind. [**Ja, das a ist im Shift-Fall nicht das Symbol hinten, sondern das hinter dem Punkt. Ja, und Reduce-Reduce ist ja damit klar...**]

- **Dann zur semantischen Analyse. Dort können ja Zirkularitäten auftreten. Wann ist das der Fall?**

  ➠ Wenn eine aktuelle Attributvariable von sich selbst abhängt.

- **Stimmt, aber formulieren Sie das mal ohne „aktuelle Attributvariable“.**

  ➠ Es existiert ein Ableitungsbaum, bei dem im Abhängigkeitsgraph ein Zyklus auftritt.

- **Und wieviele Ableitungsbäume gibt es?**
  - ➠ Im Prinzip unendlich viele.
- **Stimmt. Aber man kann die Zirkularität trotzdem feststellen, wie?**
  - ➠ Man bildet für eine Regel die unterhalb abhängigen Paare von inheriten und synthetischen Attributvariablen...
- **Wir hatten das ja induktiv angegeben, machen Sie das auch mal so.**
  - ➠ Wenn wir eine Regel haben, auf deren rechter Seite keine Nicht-Terminale mehr vorkommen, dann gehen nur die Abhängigkeiten synthetischer von inheriten Attributvariablen in die Menge.
    Bei einer Regel mit Nicht-Terminalen auf der rechten Seite wählen wir einen Ableitungsbaum mit seinen Abhängigkeiten, hängen diesen im Prinzip an der entsprechenden Stelle darunter und berechnen nun an den Oberknoten die entsprechenden Abhängigkeiten.
- **Was ist dabei zu beachten?**
  - ➠ Das man die einzelnen Bäume auch einzeln betrachtet.
- **Was hat man sonst?**
  - ➠ Sonst hat man nur die starke Nichtzirkularität.
- **Wir hatten dann ja noch die spezielle Klasse der L-Attributgrammatiken. Was gilt bei diesen?**
  - ➠ Wenn eine inherite von einer synthetischen Attributvariablen abhängig ist, dann ist der Index der synthetischen Attributvariablen kleiner als der der inheriten.
- **Und wie werden diese Attributvariablen berechnet?**
  - ➠ Man benutzt einen Stack-Automaten, der als Kelleralphabet die LR(0)-Auskünfte und eine Belegungsfunktion der Attributvariablen hat.
- **Warum werden da überhaupt LR(0)-Auskünfte verwendet?**
  - ➠ Weil der Punkt die Markierung der aktuellen Stelle erlaubt.
- **Aber warum überhaupt der Kellermechanismus? Warum nicht die lineare Ableitung wie beim TD-Analyseautomaten?**
  - ➠ Weil wir bei LAGs jeden Knoten zweimal besuchen müssen: einmal um die inheriten Attributvariablen auszurechnen und das zweite Mal, um die synthetischen Attributvariablen beim reduce-Schritt zu auszurechnen.
- **Auf was muss man denn dabei achten?**
  - ➠ Da in den Belegungen nur die formalen Attributvariablen verwendet werden, muss man darauf achten, dass der Index jeweils ein anderer ist: (wollte gerade ausholen und das an einem Beispiel erklären)
- **OK, das reicht dazu. Kommen wir noch zum Zwischencode. Da hatten wir ja zum einen die Variante, wo Prozeduren keine Parameter hatten und viel in den Befehlen wie CALL versteckt war. Ich denke da zum Beispiel an die base-Funktion. Bei Prozeduren mit Parametern ist das ja nicht mehr so. Was wird denn verwendet, um die base-Funktion zum Beispiel nachzustellen, welches Register?**

➠ Da wird das Indexregister R verwendet, über das man rekursiv die Adresse der entsprechenden Speicherzelle errechnen kann.

- **Das reicht. Wenn Sie bitte kurz draußen warten würden...**

Fazit

Am Anfang der Prüfung war ich recht nervös, was sich aber eigentlich relativ schnell gegeben hat. Es herrschte eine recht lockere Atmosphäre, die mir das Ganze erleichtert hat. Es war aber manchmal nicht ganz einfach herauszubekommen, worauf Herr Indermark eigentlich herauswollte. Ich habe bei manchen Fragen dann ersteinmal mit etwas Anderem gestartet, bis dann eine Präzisierung kam, was denn verlangt war.