I. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Christian Schulte

Ausgabe: 23.4.93 Abgabe: 30.4.93

Aufgabe I.1 (Die Struktur $\mathcal{N}$ (4T)) Gegeben ist die Signatur $\Sigma = \{0, 1, +, *\}$, wobei $0, 1$ Konstanten und $+, *$ zweistellige Funktionszeichen sind. Die Σ-Struktur $\mathcal{N}$, deren Universum die natürlichen Zahlen (inklusive Null) sind, interpretiert $0, 1$ als die entsprechenden natürlichen Zahlen und $*, +$ als die entsprecheden Funktionen auf den natürlichen Zahlen.
Geben Sie Formeln ϕ_i an, wobei die Notation $\phi(x, y, z)$ bedeutet, daß ϕ genau die freien Variablen x, y, z hat, so daß:

1. $\mathcal{N}, \alpha \models \phi_1(x, y) \iff \alpha(x)$ ist kleiner oder gleich $\alpha(y)$.
2. $\mathcal{N}, \alpha \models \phi_2(x, y) \iff \alpha(x)$ ist echt kleiner $\alpha(y)$.
3. $\mathcal{N}, \alpha \models \phi_3(x, y) \iff \alpha(x)$ teilt $\alpha(y)$.
4. $\mathcal{N}, \alpha \models \phi_4(x) \iff \alpha(x)$ ist prim.

Aufgabe I.2 (Die Struktur $\mathcal{H}$ (8T)) In der Vorlesung haben Sie die Σ-Struktur $\mathcal{H}$ kennengelernt.

1. Zeigen Sie, daß die folgenden Formeln in $\mathcal{H}$ gültig sind:

$$\begin{array}{ll} f(t_1, \ldots, t_n) \doteq f(s_1, \ldots, s_n) \rightarrow \bigwedge_{i=1}^n t_i \doteq s_i & f\ n\text{-stellig} \\ f(t_1, \ldots, t_n) \not\doteq g(s_1, \ldots, s_m) & f\ n\text{-stellig},\ g\ m\text{-stellig},\ f \neq g \\ x \not\doteq t & x \in \mathcal{V}(t), x \neq t \end{array}$$

2. Sei $x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n$ eine gelöste Form, daß heißt die x_i sind paarweise verschieden und kein x_i kommt in einem t_j vor. Zeigen Sie:

$$\mathcal{H} \models \tilde{\forall} \exists x_1 \cdots \exists x_n\ (x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n)$$

Aufgabe I.3 (Multimengen (2T, 10P)) In der Vorlesung sind *Multimengen* erwähnt worden, wir werden diese hier untersuchen. Eine *Multimenge* M über einer Grundmenge $\mathcal{G}$ ist eine Funktion $M : \mathcal{G} \rightarrow \mathbb{N}_0$. Ein Element $m \in G$ ist n-mal in M enthalten, falls $M(g) = n$, insbesondere ist es also nicht enthalten, falls $n = 0$.
Seien M, N Multimengen über $\mathcal{G}$ und $g \in \mathcal{G}$. Dann definieren wir die folgenden Operationen

$$\begin{array}{rcll} (M \cup N)(g) & = & M(g) + N(g) & \text{Vereinigung} \\ (M \cap N)(g) & = & \min(M(g), N(g)) & \text{Durchschnitt} \\ (M - N)(g) & = & M(g) \ominus N(g) & \text{Differenz}, n \ominus m = \max(0, n - m) \end{array}$$

und die folgenden Relationen

$$\begin{array}{rcll} M \subseteq N & \Leftrightarrow & M(g) \leq N(g) \quad \text{für alle } g \in \mathcal{G} & \text{Teilmultimenge} \\ g \in_n M & \Leftrightarrow & M(g) \geq n & n\text{-faches Enthaltensein} \end{array}$$

Also zum Beispiel $\{\!|1, 1, 2, 2|\!\} \cap \{\!|1, 2, 3|\!\} = \{\!|1, 2|\!\}$, $\{\!|1, 1, 2, 2|\!\} \cup \{\!|1, 2, 3|\!\} = \{\!|1, 1, 1, 2, 2, 2, 3|\!\}$.

1. Zeigen Sie, daß $\cup$ und $\cap$ assoziativ und kommutativ sind.

2. Zeigen oder widerlegen Sie: Für Multimengen M, N, O über $\mathcal{G}$ gilt:

$$M \cup (N \cap O) = (M \cup N) \cap (M \cup O)$$

$$M \cap (N \cup O) = (M \cap N) \cup (M \cap O)$$

3. Welche Funktion repräsentiert die leere Multimenge?

Eine endliche Multimenge M mit $\{\langle x_1, n_1\rangle, \ldots, \langle x_k, n_k\rangle\} = \{\langle x, M(x)\rangle | M(x) > 0\}$ mit $x_i < x_{i+1}$ für $1 \leq i < k$ uber den natürlichen Zahlen läßt sich in Prolog durch die folgende Liste $[\![M]\!]$ darstellen $[\![M]\!] = [[\overline{x_1}, \overline{n_1}], \ldots, [\overline{x_k}, \overline{n_k}]]$, wobei für eine natürliche Zahl n $\overline{n}$ als $\mathsf{s}^n(\mathsf{0})$ definiert ist.

4. Schreiben Sie in Prolog Prädikate, die Addition, die Operation $\ominus$, Multiplikation, Potenzierung, Minimum und die kleiner gleich Relation berechnen. Also zum Beispiel: $\mathsf{add}(\overline{i}, \overline{j}, \overline{k})$ wird von Prolog genau dann erfolgreich abgearbeitet, falls $i + j = k$ ist.

5. Schreiben Sie in Prolog Prädikate, die die Vereinigung, Durchschnitt, Differenz und die Multiteilmengen-Relation berechnen. Verwenden Sie dabei die oben spezifizierte Darstellungsweise!

6. Eine andere Möglichkeit zur Darstellung ist, eine endliche Multimenge einfach als Liste ihrer Elemente darzustellen, also zum Beispiel $\{\!|1, 1, 1, 2|\!\}$ als $[\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(\mathsf{0}))]$. Schreiben Sie ein Prolog-Prädikat, daß festellt, ob zwei dieser Darstellungen derselben Multimenge entsprechen.

Aufgabe I.4 (Unifizierbarkeit von Gleichungssystemen (10T, 5B)) Wir wollen ein Verfahren herleiten, daß die Unifizierbarkeit von Gleichungssystemen entscheidet.

1. Geben Sie ein Gleichungssystem an, dessen gelöste Form exponentielle Komplexität in Abhängigkeit von der Komplexität des Ausgangssytems besitzt.

2. Geben Sie ein Verfahren an, daß zu einem Gleichungssystem E eine gelöste Form berechnet, falls E unifizierbar ist, sonst $\perp$ zurückliefert. **Hinweis:** Denken Sie an die Formeln der Aufgabe I.2! Geben Sie das Verfahren in Form von Ableitungsregeln an, also zum Beispiel wird das Verfahren die folgende Regel enthalten:

$$\frac{x \doteq x \wedge E}{E}$$

3. Zeigen Sie, daß Ihr Verfahren korrekt ist.

4. Zeigen Sie: Wenn Ihr Verfahren terminiert, ist das resultierende Gleichungssystem in gelöster Form oder gleich $\perp$.

5. **(Bonus)** Zeigen Sie, daß Ihr Verfahren terminiert.

Aufgabe I.5 (Wir färben die EG! (5P)) Schreiben Sie ein Prolog-Programm, das jedes Land der EG mit einer Farbe aus $\{\mathsf{rot}, \mathsf{grün}, \mathsf{blau}, \mathsf{gelb}\}$ versieht, so daß zwei auf der Europa-Landkarte benachbarte Länder immer unterschiedlich gefärbt sind. **Hinweis:** Stellen Sie die Landkarte als Liste von Termen $\mathsf{nachbar}(\mathsf{Land}_1, \mathsf{Land}_2)$ dar.

```
exists Server In Ask in

  Server = pred {In Ask}
    local
      LookIn = pred {Message AskQ ?Agent ?NewAskQ}
         case AskQ
         of question(Pattern GetAgent)|AskQRest then
           if {Pattern Message} then
             Agent=GetAgent
             NewAskQ=AskQRest
           else
              NewAskQ = question(Pattern GetAgent)
                        | {LookIn Message AskQRest Agent}
           fi
         else
           false
         end
      end


      LookAsk = pred {Pattern InQ ?Message ?NewInQ}
         case InQ
         of GetMessage|InQRest then
           if {Pattern GetMessage} then
             Message=GetMessage
             NewInQ = InQRest
           else
              NewInQ = GetMessage | {LookAsk Pattern InQRest Message}
           fi
         else
           false
         end
      end


      DoServe = pred {InMess AskMess InQ AskQ}
          {Show InQ#AskQ}
          if Message in
            InMess = message(Message)
          then
             if Agent NewAskQ in
               {LookIn Message AskQ Agent NewAskQ}
             then
                process
                    {Agent Message}
                end
                {DoServe {Get $ In} AskMess InQ NewAskQ}
             else
                {DoServe {Get $ In} AskMess {Append InQ [Message]} AskQ}
             fi
```

```
          [] Pattern Agent in
            AskMess = question(Pattern Agent)
          then
             if Message NewInQ in
                {LookAsk Pattern InQ Message NewInQ}
             then
                process
                   {Agent Message}
                end
                {DoServe InMess {Get $ Ask} NewInQ AskQ}
             else
                {DoServe InMess {Get $ Ask}
                 InQ {Append AskQ [question(Pattern Agent)]}}
             fi
          fi
      end
   in
      process
         {DoServe {Get $ In} {Get $ Ask} nil nil}
      end
   end
end

{NewChannel In}
{NewChannel Ask}
{Server In Ask}

{Put message(a(aa)) In}
{Put message(b(bb)) In}
{Put message(c(cc)) In}
{Put
 question(pred {$ Pattern} Pattern=a(_) end
          pred {$ Message} {Show a#Message} end)
 Ask}
{Put
 question(pred {$ Pattern} Pattern=d(_) end
          pred {$ Message} {Show d#Message} end)
 Ask}
{Put message(c(cc)) In}
{Put message(d(cc)) In}
```

X. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte
Ausgabe: 29.6.93 Abgabe: 6.7.93 Abgabe der Programme: 13.7.93

Aufgabe X.1 (Vollständige Theorien (10T)) Zeigen Sie, daß die Theorie

$$\mathrm{CET}(\Sigma) \cup \{\mathrm{DCA}(\Sigma)\} \cup \{\forall \mathsf{X}\, (\mathsf{list}(\mathsf{X}) \leftrightarrow \exists \mathsf{Y} \exists \mathsf{Z}\, (\mathsf{X} \doteq \mathsf{cons}(\mathsf{Y}, \mathsf{Z}) \wedge \mathsf{list}(\mathsf{Z})))\}$$

nicht vollständig ist, wobei die Signatur Σ das einstellige Relationssymbol list, die Konstanten nil und 0, das einstellige Funktionssymbol s, und das zweistellige Funktionssymbol cons enthält. **Hinweis:** Denken Sie an die unendliche Liste, die alle natürliche Zahlen in der Darstellung $\mathsf{s}^n(\mathsf{0})$ enthält; Sie kennen diese bereits aus der Vorlesung.

Aufgabe X.2 (Speisende Informatiker (6P)) In der Mensa sitzen drei Informatiker (in der klassischen Formulierung nach Hoare: Philosophen) nach der Vorlesung um einen runden Tisch. Zwischen je zweien liegt eine Gabel, vor jedem steht ein Teller Spaghetti. Zum Essen werden zwei Gabeln benötigt. Deshalb kann jeweils nur einer der drei essen, während die anderen über das Universum der Bäume nachdenken. Ist der Essende satt, legt er die Gabeln zurück, und einer der Anderen ist an der Reihe. Der Gesättigte beginnt seinerseits zu denken, und ist sogleich wieder an den Gabeln interessiert – Denken macht hungrig!

Implementieren Sie die Situation durch ein Oz-Programm: Dabei sollen die drei Informatiker in indeterministischer aber fairer Abfolge Zugriff auf die Gabeln erhalten. Weiterhin soll es nicht zu einer Verklemmung kommen (Stichwort: Semaphoren), bei der jeder Informatiker je eine Gabel hat, und alle drei weder denken noch essen können. Modellieren Sie

- jeden Philosophen durch einen Prozeß,
- die Gabeln mit Hilfe von Kanälen, und
- machen Sie den Ablauf sichtbar mittels {Show} oder {Browse}.

Aufgabe X.3 (Merge (6P)) Ein Oz-Kanal C ist für eine $n:m$–Kommunikation mit dem folgenden Protokoll geeignet: Alle Prozesse, die C kennen, können schreibend und lesend auf C zugreifen, jede Nachricht kann jedoch nur einmal gelesen werden. Das sieht man etwa an der "Verschmelzungsregel" für komplementäre Kommunikationstoken:

$$y!x \wedge z?x \wedge \Box x \rightarrow y \doteq z \wedge \Box x.$$

Schreiben Sie ein Oz-Programm, das einen $n:m$-Mailverteiler realisiert: Das Programm soll n Eingabekanäle und m Ausgabekanäle so verknüpfen, daß jede Nachricht die auf einem der Eingabekanäle erscheint, auf *jedem* der Ausgangskanäle sichtbar wird! Also sollte bei dem folgenden Beispiel

```
exists Out1 Out2 In1 In2 In3 X Y Z in
   {MailNM [Out1 Out2] [In1 In2 In3]}

   {Put message Out2}
   if det X then {Show 'In1'} fi
   if det Y then {Show 'In2'} fi
   if det Z then {Show 'In3'} fi
   {Get X In1} {Get Y In2} {Get Z In3}
```

'In1', 'In2' und 'In3' ausgegeben werden. Achten Sie insbesondere darauf, daß Ihr Mailverteiler *fair* ist, das heißt, daß jede Nachricht, egal auf welchem Kanal sie ankommt, auch verteilt wird.

Aufgabe X.4 (Server (10P)) Gegeben Sei das folgende Problem: Auf einem Kanal In kommen Nachrichten der Form message(M) an, auf einem anderen Kanal Ask kommen Anfragen der Form question(*Pattern Agent*) an, wobei *Pattern* und *Agent* zwei einstellige Prädikate sind.

Trifft jetzt eine Nachricht message(M) auf In ein, so soll $\{Agent\ M\}$ ausgeführt werden, falls $\{Pattern\ M\}$ zu **true** reduziert. Wurde auf dem Kanal Ask jedoch keine solche Anfrage bisher gesendet, so soll die Nachricht M gespeichert werden, und in der oben geschilderten Weise verfahren werden, wenn die richtige Anfrage auf Ask eintrifft.

Umgekehrt sollen natürlich auch Anfragen gespeichert werden, wenn noch keine passende Nachricht abgeliefert wurde.

Programmieren Sie ein Prädikat {Serve In Ask}, daß die obige Funktionalität implementiert. Achten Sie wiederum auf Fairness bei der Bedienung der Nachrichten und Anfragen!

Aufgabe X.5 (Endliche Listen (6P)) Schreiben Sie ein Oz-Prädikat {FinList L}, daß zu **true** reduziert, falls L eine endliche Liste ist.

Aufgabe X.6 (Smart–Append (8P 4B)) Wenn man bezüglich der Standard–Definition von Append, also

```
pred {Append Xs Ys Zs}
   or Xs=nil Ys=Zs
   [] X Xr Zr in Xs=X|Xr Zs=X|Zr then
      {Append Xr Ys Zr}
   ro
end
```

die Anfrage

```
exists L in
   {Append _|L _ L}
```

stellt, so divergiert dieser Aufruf (Wieso?). Programmieren Sie ein Prädikat {SmartAppend Xs Ys Zs} an, daß die Listenkonkatenation implementiert und bei diesem Aufruf terminiert.

Bonus: Schreiben Sie Ihr SmartAppend so um, daß es bei dem obigen Aufruf zu false reduziert.

Aufgabe X.7 (Normierer (6T)) Zeigen Sie, daß für einen Normierer θ gilt:

$$\theta x = \theta y \iff \theta \models x \doteq y.$$

XI. Übungsblatt zur Vorlesung "Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte

Ausgabe: 6.7.93 Abgabe: 13.7.93 Abgabe der Programme: 20.7.93

Aufgabe XI.1 (Subsumtion (4T)) Welche der beiden Aussagen gilt, welche ist falsch?

$$\mathsf{X} \doteq \mathsf{f}(\mathsf{g}(\mathsf{X}), \mathsf{Y}) \models_{\mathrm{RAT}} \exists \mathsf{Y}\, \mathsf{X} \doteq \mathsf{f}(\mathsf{Y}, \mathsf{g}(\mathsf{Y}))$$

$$\mathsf{X} \doteq \mathsf{f}(\mathsf{g}(\mathsf{X}), \mathsf{Y}) \models_{\mathrm{RAT}} \forall \mathsf{Y}\, \mathsf{X} \not\doteq \mathsf{f}(\mathsf{Y}, \mathsf{g}(\mathsf{Y}))$$

Aufgabe XI.2 (Baumbereiche (4T)) Zu einem Σ–Baum $T : B \to \Sigma$ definieren wir die Sprache $T^* \subseteq \Sigma^*$ wie folgt:

$$T^* := \{T(n_1)T(n_1n_2)\cdots T(n_1\cdots n_k) \mid n_1\cdots n_k \in B, n_i \in \mathbb{N}\}.$$

Zum Beispiel, für den Baum

$$T := \{(\epsilon, \mathsf{f}), (1, \mathsf{a}), (2, \mathsf{g}), (21, \mathsf{b}), (22, \mathsf{c})\}$$

ist

$$T^* = \{\mathsf{fa}, \mathsf{fgb}, \mathsf{fgc}\}.$$

Welche der Richtungen $\{\, \substack{\Rightarrow \\ \Leftarrow} \,\}$ gilt in der folgenden Aussage:

$$T^* \text{ ist regulär} \quad \Longleftrightarrow \quad B \text{ ist regulär?}$$

Aufgabe XI.3 (Elementare Äquivalenz (4T)) Zwei Modelle $\mathcal{A}$ und $\mathcal{B}$ heißen **elementar äquivalent**, wenn für jede Formel σ gilt:

$$\mathcal{A} \models \sigma \iff \mathcal{B} \models \sigma.$$

Zeigen Sie: Eine Theorie Δ ist genau dann vollständig, wenn je zwei Modelle von Δ elementar äquivalent sind.

Aufgabe XI.4 (Normalformen (6T)) In der Vorlesung haben Sie die Operation $\overline{\cdots}$ kennengelernt, die aus einem RAT–Constraint φ alle trivialen Gleichungen $x \doteq x$ und alle Duplikate entfernt. Zum Beispiel:

$$\overline{\mathsf{Y} \doteq \mathsf{Y} \land \mathsf{X} \doteq \mathsf{f}(\mathsf{X}) \land \mathsf{X} \doteq \mathsf{f}(\mathsf{X})} \quad = \quad \mathsf{X} \doteq \mathsf{f}(\mathsf{X}).$$

Geb Sie einen Kalkül an, der aus einem gegebenen Constraint φ den Constraint $\overline{\varphi}$ herstellt. Beweisen Sie Ihr Ergebnis.

Aufgabe XI.5 ($\overline{x}$–orientierter Normierer (6T)) Mit dem Simplifikationssystem COL kann man kanonische Formen herstellen, die es gestatten, daraus Normierer abzulesen. Erweitern Sie das Simplifikationssystem so, daß automatisch zu gegebenem $\overline{x}$ ein $\overline{x}$–orientierter Normierer berechnet wird. Beweisen Sie Ihr Ergebnis.

Aufgabe XI.6 (Die RAT–Maschine (10T)) Gegeben sei die Formel

$$\sigma = \mathsf{f}(\mathsf{X}, \mathsf{g}(\mathsf{X}, \mathsf{c})) \doteq \mathsf{f}(\mathsf{g}(\mathsf{X}, \mathsf{c}), \mathsf{Y}).$$

1. Transformieren Sie die Formel in einen kanonischen Constraint.

2. Geben Sie Maschineninstruktionen an, die diesen kanonischen Constraint in der Maschine realisieren. Skizzieren Sie dabei nach jeder Maschineninstruktion den Zustand der Maschine.

3. Stellen Sie mit Hilfe der Maschine fest, ob

$$\sigma \models_{\mathrm{RAT}} \mathsf{X} \doteq \mathsf{Y}$$

gilt.

Aufgabe XI.7 (Der RAT**–Compiler (20P))** Schreiben Sie ein Oz–Programm, das eine Liste von Constraints der Form

$$x \doteq y \qquad \text{oder} \qquad x \doteq f(\overline{y})$$

entgegennimmt, und diese in Maschineninstruktionen der RAT–Maschine übersetzt.

XII. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte

Ausgabe: 13.7.93 Abgabe: 20.7.93 Abgabe der Programme: 20.7.93

Aufgabe XII.1 (Das Propagierungsprinzip (6T)) In der Vorlesung haben Sie das Propagierungsprinzip kennengelernt. Eine operationale Formulierung lautet wie folgt, wobei σ, τ und π logische Formeln sind:

$$\frac{}{\tau \xrightarrow{\pi} \tau} \qquad\qquad \frac{}{\tau \xrightarrow{\pi} \tau \wedge \pi}$$

und

$$\frac{\tau \xrightarrow{\pi} \tau' \quad \sigma \xrightarrow{\pi} \sigma'}{\tau \wedge \sigma \xrightarrow{\pi} \tau' \wedge \sigma'} \qquad \frac{\tau \xrightarrow{\pi} \tau'}{\neg\tau \xrightarrow{\pi} \neg\tau'} \qquad \frac{\tau \xrightarrow{\pi} \tau'}{\exists x\, \tau \xrightarrow{\pi} \exists x\, \tau'}\, x \notin \mathcal{V}(\pi)$$

Zeigen Sie:

$$\sigma \xrightarrow{\pi} \tau \quad \Rightarrow \quad \pi \wedge \sigma \models\!\!\!\!\dashv \pi \wedge \tau$$

Aufgabe XII.2 (Die logische Teilsprache von Oz (2T)) In der Vorlesung haben Sie gehört, daß Konditionale mit mehr als einer Klausel und die Kommunikationsprimitive **nicht** zur logischen Teilsprache von Oz gehören. Überlegen Sie sich Beispiele, an denen man die Problematik dieser Konstrukte erkennen kann!

Aufgabe XII.3 ($\exists$ in der RAT–Maschine (16T)) Die in der Vorlesung vorgestellte RAT–Maschine ist nicht in der Lage, existentiell quantifizierte Variablen zu behandeln. Wir wollen hier überlegen, wie diese integriert werden können.

1. Was muß an der Maschine geändert werden, um Existenzquantoren auf Toplevel abzuarbeiten?

2. Um Existenzquantoren in Wächtern von Konditionalen, also zum Beispiel

$$\langle \exists \mathsf{Y} \exists \mathsf{Z}\; \mathsf{X} \doteq \mathsf{pair}(\mathsf{Y}, \mathsf{Z}); \cdots; \cdots \rangle\,,$$

   zu behandeln, führen wir **lokale Variablen** in der Maschine ein, also Variablen, die nur lokal in dem Wächter eines Konditionals erzeugt und benutzt werden.

   (a) Erweitern Sie die Datentypen der Maschine um lokale Variablen!

   (b) Wie muß das Binden von Variablen erweitert werden? **Hinweis:** Denken Sie an den Begriff eines orientierten Normierers! Berücksichtigen Sie auch die Behandlung des Trail–Stapels!

   (c) Wie kann jetzt bei der then–Instruktion Entailment, Disentailment oder keines von beiden (also Notwendigkeit zur Suspension) festgestellt werden? Erweitern Sie diese Maschineninstruktion entsprechend!

3. Unsere obige Erweiterung war so, daß sich die existentielle Quantifizierung nur über den Wächter erstreckte. Es ist jedoch wünschenswert, daß sich die Quantifizierung auch über den then–Teil des Konditionals erstreckt, denn typischerweise benutzt man das Konditional wie in dem folgenden Beispiel:

   $$\langle \exists \mathsf{H} \exists \mathsf{R}\ \mathsf{L} \doteq \mathsf{cons}(\mathsf{H},\mathsf{R}); \text{Verwendung von } \mathsf{H} \text{ und } \mathsf{R}; \text{Keine Verwendung von } \mathsf{H} \text{ und } \mathsf{R}\rangle\,.$$

   Die logische Semantik des Konditionals wäre dann

   $$\langle \exists \overline{x}\, \varphi; \sigma; \tau\rangle \quad \rightsquigarrow \quad \exists \overline{x}(\varphi \wedge \sigma) \vee (\neg \exists \overline{x}\, \varphi \wedge \tau).$$

   Für den Test der Subsumtion müssen diese Variablen natürlich weiterhin als lokal behandelt werden. Wenn jedoch der Wächter subsumiert ist, müssen die lokalen Variablen zu globalen Variablen geändert werden. Überlegen Sie sich eine Lösung, die konstante Laufzeit hat (unabhängig von Anzahl der quantifizierten Variablen)! **Hinweis:** Ordnen Sie die nötige Information dem Wächter und nicht jeder einzelnen Variablen zu!

Aufgabe XII.4 (Saraswats tell–Operatoren (10T)) Bei der Vorstellung des Kalküls K kamen auch Saraswats Operatoren etell [·] und atell [·] zur Sprache.

1. Rekapitulieren Sie die Semantik der beiden Operatoren!
2. Was muß man unternehmen, um den Operator etell [·] in der RAT–Maschine zu realisieren?
3. Integrieren Sie die atell [·] Operation in die RAT–Maschine!
4. Zeigen Sie, daß On–Line Constraintverarbeitung mit Ihrer erweiterten Maschine nur dann möglich ist, wenn Constraints nicht auf Toplevel, sondern nur in Wächtern von Konditionalen und in etell [·]– und atell [·]–Operatoren vorkommen!

Aufgabe XII.5 (Wörterbuch (10P)) Implementieren Sie ein Wörterbuch als ein Oz–Objekt, daß auf einer Sammlung von Termen die folgenden Operationen zur Verfügung stellt:

insert(T): Legt den Term T in der Sammlung ab.

read($Pattern\ T$): Sobald in der Sammlung ein Term S vorhanden ist, für den $\{Pattern\ S\}$ gilt, wird T an S gebunden.

remove($Pattern\ T$): Analog zu read, nur wird anschließend der Term aus der Sammlung entfernt.

check($Pattern$): Testet auf das Vorhandensein eines Termes T in der Sammlung mit $\{Pattern\ T\}$.

Beachten Sie dabei, daß *Pattern* dabei jeweils ein einstelliges Prädikat ist!

Aufgabe XII.6 (Binärbäume (10P)) Implementieren Sie den abstrakten Datentyp eines geordneten Binärbaumes. Sehen Sie dazu die Operationen Einfügen, Löschen und Ausgabe als eine sortierte Liste vor.

1. Schreiben Sie ein Oz–Objekt, das diesen Datentyp implementiert, wobei der Baum im Zustand des Objektes gespeichert ist. Bei dem Erzeugen des Objektes soll ein zweistelliges Prädikat P in der init–Methode angegeben werden, das die Ordnung auf den Elementen des Baumes realisiert.
2. Implementieren Sie eine alternative Lösung, in der jeder Knoten des Baumes ein eigenes Objekt ist!

XIII. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte

Keine Abgabe

Aufgabe XIII.1 (Faires Mischen von Strömen) Schreiben Sie ein dreistelliges Prädikat pred {Merge Xs Ys Zs}, so daß Elemente der beiden Ströme Xs und Ys auf den Strom Zs geschrieben werden. Realisieren Sie eine **faire** Lösung, das heißt, daß jedes Element auf einem der Eingabeströme nach endlich vielen Berechnungsschritten auf den Ausgabestrom geschrieben wird.

Aufgabe XIII.2 (Graphik für Binärbäume) Auf dem letzten Übungsblatt haben Sie einen abstrakten Datentyp für Binärbäume implementiert. Erweitern Sie Ihre Lösung in der jeder Knoten des Binärbaumes ein eigenes Objekt ist, so, daß die Binärbäume graphisch mit Hilfe des Oz–Windowsystems dargestellt werden. Entwerfen Sie Ihre Lösung so, daß sich bei Änderung des Baumes automatisch die graphische Darstellung dieses Baumes ändert!

Aufgabe XIII.3 (Property–Listen) In der Vorlesung haben Sie Records in Oz kennengelernt. Hier wollen wir diese Records als Property-Listen implementieren. Dabei ist PL eine Property-Liste, falls

$$\mathsf{PL} = [a_1\#t_1 \ \cdots \ a_n\#t_n]$$

ist. Dabei ist „#" der Tupel–Konstruktor in Oz (Schauen Sie in der On–Line Dokumentation nach!). Diese Property–Liste implementiert den Record mit den Feldern $a_1, \ldots, a_n$ und den Einträgen $t_1, \ldots, t_n$. Implementieren Sie die Oz–Record Operationen Select (das ist der Punkt in Oz) und AdjoinAt, deren Semantik Sie auch wiederum in der On–Line Dokumentation nachschlagen können.

Aufgabe XIII.4 (Sicherheit in Oz) Gegeben Sei folgendes Problem: Sie haben einen Oz–Term T, den Sie geheim halten wollen: Zugriff auf T ist nur dann erlaubt, wenn ein eindeutiger Schlüssel Key vorhanden ist.

Schreiben Sie ein Prädikat {MkTreasure T Key Treasure}, daß zu einem gegebenen Term T folgendes leistet: Key wird an einen eindeutigen Schlüssel und Treasure wird an ein zweistelliges Prädikat gebunden, so daß der Aufruf {Treasure Key' T'}, T′ genau dann an T bindet, wenn in Oz Key′ = Key gilt.

Welche Datenstruktur eignet sich in Oz für die Darstellung von Key?

Aufgabe XIII.5 (Pipeline des Eratosthenes) Sie kennen ja das Sieb des Eratosthenes zur Primzahlberechnung. Sie sollen hier eine „zeitgemäßere" Lösung des Problems in Oz implementieren.

Dabei brauchen wir zwei verschiedene Arten von Prozessen:

- Einen Prozeß, der alle natürliche Zahlen erzeugt und auf einen Kanal schreibt.
- Einen Prozeß, der eine Zahl n von einem Kanal C_{in} entgegennimmt, diese ausdruckt und dann einen weiteren Prozeß erzeugt. Dieser Prozeß liest Zahlen von dem Kanal C_{in} und gibt diese auf einem Kanal C_{out} wieder aus, falls die Zahlen nicht durch n teilbar sind. Der ursprüngliche Prozeß nimmt jetzt die Eingaben auf dem Kanal C_{out} entgegen.

Diese Prozesse werden jetzt so miteinander verbunden, daß nach und nach alle Primzahlen ausgedruckt werden.

1. Implementieren Sie die Pipeline des Eratosthenes.
2. Erweitern Sie Ihre Lösung so, daß zwischen den einzelnen Prozessen nur noch synchrone Kommunikation stattfindet.
3. Ändern Sie Ihre Lösung dahingehend, daß Listen an Stelle von Kanälen verwendet werden.

Aufgabe XIII.6 (Game of Life) Sie alle werden ja das Game of Life vom amerikanischen Mathematiker John Conway kennen, das die Dynamik von Zellpopulationen wiederspiegelt. Hier noch einmal kurz die Regeln:

- Gegeben sei als Lebensraum der Population ein Boolesches Array der Größe $m \times n$. Ein Feld dieses Arrays hat den Wert **true**, falls es eine lebendige Zelle enthält, ansonsten den Wert **false**. Jedes Feld bietet höchstens einer Zelle Platz.
- Als *Generation* k wollen wir eine bestimmte Belegung dieses Arrays zu einem Zeitpunkt k bezeichnen. Die *Nachbarn* eines Feldes $a(i,j)$ sind durch die folgende Skizze gegeben (8-Nachbarschaft):

	$j-1$	j	$j+1$
$i-1$	$\times$	$\times$	$\times$
i	$\times$		$\times$
$i+1$	$\times$	$\times$	$\times$

  Die *Anzahl der Nachbarn* der Zelle $a^k(i,j)$ zum Zeitpunkt k sei $n^k(i,j)$.
- Der Übergang von einer Generation k zur folgenden $k+1$ ist wie folgt definiert.
  1. $n^k(i,j) < 2$: $a^{k+1}(i,j) = 0$ (die Zelle stirbt an Vereinsamung)
  2. $n^k(i,j) = 2$: $a^{k+1}(i,j) = a^k(i,j)$ (die Zelle bleibt aus Langeweile am Leben)
  3. $2 < n^k(i,j) < 4$: $a^{k+1}(i,j)$ (zwei oder drei lebendige Zellen zeugen eine neue)
  4. $3 < n^k(i,j)$: $a^{k+1}(i,j) = 0$ (die Zelle stirbt an Platzangst)

Jede Generation entspricht einem Schwarz-Weiß-Muster, die Generationswechsel einer Veränderung dieser Muster.

1. Schreiben Sie ein Oz–Objekt Matrix, das den Datentyp eines zweidimensionalen Arrays implementiert.
2. Implementieren Sie ein Oz–Objekt Population, daß auf Zusendung einer Methode step die Berechnung einer neuen Zellpopulation nach den obigen Regeln vornimmt.

   Ererben Sie dabei vom Matrix–Objekt.
3. Erweitern Sie das Objekt Population um eine graphische Ausgabe für die gerade aktuelle Zellpopulation.
4. **Challenge:** Entwerfen Sie eine alternative Realisierung des Game of Life, indem Sie jede Zelle als ein eigenes Objekt darstellen. Überlegen Sie sich, wie sie den kontrollierten Übergang zu einer neuen Generation von Zellen realisieren können.

II. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Christian Schulte
Ausgabe: 30.4.93   Abgabe: 7.5.93

Aufgabe II.1 ($\mathcal{H}$–Strukturen (8T)) Wir nennen eine Struktur eine **Herbrand-** oder **$\mathcal{H}$–Struktur**, falls Universum und Interpretation der Funktionszeichen mit der Struktur $\mathcal{H}$ übereinstimmen. Zeigen Sie:

1. Ein Programmm P hat ein $\mathcal{H}$–Modell, das heißt eine Herbrand-Struktur, die Modell von P ist.
2. Wenn $(\mathcal{H}_i)_{i \in I}$ eine Familie von Herbrand–Modellen von P ist, so ist der Durchschnitt $\bigcap_{i \in I} \mathcal{H}_i$ wieder ein Herbrand–Modell von P (Der Durchschnitt von Herbrand-Strukturen ist gerade durch den Durchschnitt der Interpretation ihrer Prädikatszeichen bestimmt.)
3. Jedes Programm P hat ein kleinstes (bezüglich $\subseteq$) Herbrand-Modell $\mathcal{H}(P)$.
4. $P \models \exists G \iff \mathcal{H}(P) \models \exists G$.
5. Aussage 2. gilt nicht für allgemeine Formelmengen.

Mit ANT_i bezeichnen wir die Antwortmenge des Kalküls i, mit $\longrightarrow_i$ die definierte Ableitungsrelation des Kalküls i.

Aufgabe II.2 (Korrektheit von Kalkül 1 (6T)) Beweisen Sie die Korrektheit von Kalkül 1, das heißt:

$$\theta \in ANT_1(G, \overline{x}) \quad \Longrightarrow \quad P \models \theta(G).$$

Hinweis: Benutzen Sie Induktion über die Ableitung im Kalkül 1.

Aufgabe II.3 (Die Kalküle 1 und 2 (6T)) Zeigen Sie:

1. Wenn $A \wedge G\theta H \wedge \theta G$ und $A \wedge G \longrightarrow \theta' H \wedge \theta' G$ mit Anwendung derselben Klausel, dann existiert ein Gleichungssystem E, so daß $\theta \models\!\mid_{\mathcal{H}} E \wedge \theta'$.
2. $ANT_2(G, \overline{x}) \subseteq ANT_1(G, \overline{x})$.

Aufgabe II.4 (Terme (5P)) In der Vorlesung haben wir die Konvention eingeführt, Funktionszeichen (a, n) in Termen einfach als a zu schreiben, zum Beispiel anstatt $(\mathsf{cons}, 2)\Big((\mathsf{0}, 0), (\mathsf{nil}, 0)\Big)$ kürzer $\mathsf{cons}\,(\mathsf{0}, \mathsf{nil})$. Einen Term $t = (a, n)(t_1, \ldots, t_n)$ in dieser "langen" Form kann man in Prolog zum Beispiel durch die Liste

$$[\![t]\!] = [[a|n], [\![t_1]\!], \ldots, [\![t_n]\!]]$$

darstellen. Der obige Term sähe also so aus: $[[\mathsf{cons}|0], [0|0], [\mathsf{nil}|0]]$. Schreiben Sie ein Prolog-Prädikat, daß einen Prolog-Term in so eine Liste umrechnet (und umgekehrt). **Hinweis:** Sie können die Standardprädikate functor/3, arg/3 und = ../2 benutzen.

Aufgabe II.5 (Differenzlisten (7P)) Eine Anwendung unvollständiger Datenstrukturen in PROLOG sind die sogenannten **Differenzlisten**. Eine Differenzliste L ist ein Term der Form $\mathsf{L} = [s_1, \ldots, s_n|\mathsf{L}'] - \mathsf{L}'$, intuitiv: Wenn ich die Liste L' von der Liste L abziehe, bekomme ich die Liste $[s_1, \ldots, s_n]$, oder graphisch:

$$\begin{array}{rlll} & [t_1, \ldots, t_n, & t_n + 1, t_n + 2, \ldots] & = \mathsf{L} \\ & - & [t_n + 1, t_n + 2, \ldots] & = \mathsf{L}' \\ = & [t_1, \ldots, t_n] & & = \mathsf{L} - \mathsf{L}' \end{array}$$

1. Schreiben Sie ein Prädikat $\mathsf{dlist}(\mathsf{L})$, das genau dann gilt, wenn L eine Differenzliste ist.
2. Schreiben Sie ein Prädikat $\mathsf{dappend}$, daß die Konkatenation von Differenzlisten berechnet.
3. Schreiben Sie den Quicksort-Algorithmus unter Verwendung von Differenzlisten.
4. Schreiben Sie den Quicksort-Algorithmus unter Verwendung von Differenzlisten, ohne jedoch daß Prädikat $\mathsf{dappend}$ aus 2. zu verwenden.
5. Eine Teilliste einer Liste $[t_1, \ldots, t_n]$ ist definiert als $[t_i, \ldots, t_j]$ mit $1 \leq i \leq j \leq n$. Schreiben Sie ein Programm in PROLOG, ohne jedoch setof, bagof oder ähnliches zu benutzen, das zu einer gegebenen Liste L die Liste alle Teillisten von L berechnet. Ist es möglich und von Vorteil bei diesem Problem Differenzlisten zu benutzen?

Aufgabe II.6 (Turingvollständigkeit von PROLOG (8P)) Zeigen Sie, daß PROLOG Turing-vollständig ist. **Hinweis:** Schreiben Sie ein Programm, daß eine Turingmaschine simuliert. Der Begriff der Turingmaschine dürfte Ihnen bekannt sein, Sie können diesen zum Beispiel in [LP81, Seite 168 ff.] finden.

Literatur

[LP81] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall. 1981.

III. Übungsblatt zur Vorlesung "Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Christian Schulte

Ausgabe: 7.5.93 Abgabe: 14.5.93 Abgabe der Programme: 21.5.93

Aufgabe III.1 (Kongruenzen (14T)) Gegeben sei eine prädikatenlogische Signatur Σ, die nur Funktionssymbole enthält. Die Menge $GT(\Sigma)$ bezeichnet die Menge der Σ-Grundterme, das heißt der variablenfreien Terme. Nun haben wir eine Menge $Ax \subseteq GT(\Sigma) \times GT(\Sigma)$ von Paaren ("Axiome") von Σ-Grundtermen. Ein Paar $(s,t) \in Ax$ schreiben wir als $s \doteq t$.

Eine Äquivalenzrelation $\sim$ über $GT(\Sigma)$ heißt Kongruenzrelation, wenn für jedes n-stellige Funktionszeichen aus Σ gilt: Aus $s_i \sim t_i$ $(1 \leq i \leq n)$ folgt $f(s_1, \ldots, s_n) \sim f(t_1, \ldots, t_n)$.

Sei $\equiv$ die kleinste Kongruenzrelation auf $GT(\Sigma)$, die die Menge Ax (als Relation) umfaßt.

Wir definieren jetzt den Kalkül B[1], der aus den folgenden Axiomen

$$\frac{}{s \doteq s} \qquad \frac{}{s \doteq t}(s \doteq t \in Ax)$$

und Regeln besteht

$$\frac{s \doteq t}{t \doteq s} \qquad \frac{r \doteq s \quad s \doteq t}{r \doteq t} \qquad \frac{s_1 \doteq t_1 \quad \cdots \quad s_n \doteq t_n}{f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n)}(f \in \Sigma_n)$$

$s \doteq t$ ist jetzt im Kalkül B ableitbar, wenn $s \doteq t$ entweder der Konklusion eines Axioms entspricht, oder der Konklusion einer Regel, und die Prämissen der Regel wiederum ableitbar sind.

1. Sei $\Sigma = \{(\mathsf{pair}, 2), (\mathsf{a}, 0), (\mathsf{b}, 0)\}$ und $Ax = \{\mathsf{pair}(\mathsf{pair}(r,s),t) = \mathsf{pair}(r, \mathsf{pair}(s,t)) \mid r,s,t \in GT(\Sigma)\}$. Welche der folgenden Gleichungen sind herleitbar (Geben Sie Ableitungen an):

$$\mathsf{pair}(\mathsf{pair}(\mathsf{a},\mathsf{b}),\mathsf{b}) \doteq \mathsf{pair}(\mathsf{pair}(\mathsf{a},\mathsf{b}),\mathsf{b}) \tag{1}$$
$$\mathsf{pair}(\mathsf{pair}(\mathsf{b},\mathsf{a}),\mathsf{b}) \doteq \mathsf{pair}(\mathsf{pair}(\mathsf{a},\mathsf{b}),\mathsf{b}) \tag{2}$$
$$\mathsf{pair}(\mathsf{pair}(\mathsf{b},\mathsf{pair}(\mathsf{a},\mathsf{b})),\mathsf{a}) \doteq \mathsf{pair}(\mathsf{pair}(\mathsf{b},\mathsf{a}),\mathsf{pair}(\mathsf{b},\mathsf{a})) \tag{3}$$

2. Zeigen Sie, daß folgendes gilt: $\equiv$ ist gleich dem Durchschnitt aller Kongruenzrelationen über $GT(\Sigma)$, die die Menge Ax umfassen. **Hinweis:** Denken Sie an Aufgabe II.1 auf dem letzten Übungsblatt.

3. Zeigen Sie:

$$s \doteq t \text{ ableitbar in B} \iff s \equiv t$$

Aufgabe III.2 (Umdrehen (6 T)) Geben Sie ein Programm P im Kalkül 3 an, das ein zweistelliges Prädikat $\mathsf{reverse}$ definiert. Dabei soll $P, \alpha \models \mathsf{reverse}(\mathsf{X}, \mathsf{Y})$ genau dann gelten, wenn $\alpha(\mathsf{X})$ und $\alpha(\mathsf{Y})$ Listen sind, und wenn $\alpha(\mathsf{X})$ die Elemente von $\alpha(\mathsf{Y})$ in umgekehrter Reihenfolge enthält.

Zudem soll gelten, daß nur $O(n)$ Reduktionsschritte (im Kalkül 3) benötigt werden, um eine Liste mit n Elementen umzudrehen. Zeigen Sie, daß Ihr Prädikat dieses leistet.

[1] Nach dem amerikanischen Mathematiker G. Birkhoff

Geben Sie außerdem die Clark'sche Vervollständigung dieses Programmes an.

Aufgabe III.3 (Mastermind (15P)) Sie sollen ein PROLOG-Programm schreiben, das das Spiel Mastermind spielen kann.

Zur Erinnerung: Es gibt Stifte in den Farben rot, grün, gelb, blau, weiß und schwarz. Spieler A denkt sich ein Quadrupel von Stiften aus, und Spieler B versucht dieses Quadrupel folgendermaßen zu erraten: Er sagt Spieler A ein Quadrupel von Stiften, und Spieler A antwortet zwei Zahlen. Die erste Zahl gibt an, wieviele Stifte in beiden Quadrupeln an derselben Position dieselbe Farbe haben. Die zweite Zahl gibt an, wieviele Stifte dieselbe Farbe haben, sich aber nicht an derselben Position befinden. Diese Information nutzt Spieler B für seinen nächsten Versuch. Dies geht so weiter, bis Spieler B das Quadrupel von Spieler A erraten hat.

1. Schreiben Sie ein PROLOG-Prädikat bewerte, so daß bewerte(Q_A, Q_B, Z_1, Z_2) gilt: Z_1 ist die erste, Z_2 die zweite Zahl (nach der obigen Beschreibung) für die Quadrupel Q_A und Q_B. Überlegen Sie sich eine günstige Datenstruktur für die Darstellung der Quadrupel!

2. Schreiben Sie ein PROLOG-Prädikat rate(Q, V), so daß Q das zu erratende Quadrupel, und V die Liste der gemachten Versuche bis zum Erraten ist. **Hinweis:** Der Computer soll nach folgender Strategie vorgehen: Wenn die Liste der bisherigen Versuche gleich $[Q_1, \ldots, Q_n]$ ist, denkt er sich ein Quadrupel Q' aus, für das gilt:

   (a) $Q' \neq Q_i$ für $(1 \leq i \leq n)$.

   (b) Wenn bewerte(Q, Q_i, p, q) und bewerte(Q', Q_i, p', q') gilt, dann gilt $p = p'$ und $q = q'$ für $(1 \leq i \leq n)$. Das heißt, das Quadrupel Q' wird bezüglich der bisher gemachten Versuche Q_i genauso bewertet wie das Quadrupel, das zu erraten ist.

IV. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Christian Schulte

Ausgabe: 14.5.93 Abgabe: 21.5.93 Abgabe der Programme: 28.5.93

Aufgabe IV.1 (Kontextfreie Grammatiken (6T)) In der Vorlesung haben Sie ordnungssortierte Signaturen und Kalküle kennengelernt. Wir werden dieses auf den Ihnen bekannten Formalismus der kontextfreien Grammatiken (KFG) (siehe zum Beispiel [**?**, Seite 95 ff.]) anwenden.

1. Geben Sie eine ordnungssortierte Signatur Σ für KFG an. **Hinweis:** Modellieren Sie folgende Elemente: Terminalsymbole, Nichtterminalsymbole, Wörter aus Symbolen und Produktionen.
2. Geben Sie einen ordnungssortierten Σ–Kalkül an, der die übliche Ableitungsrelation einer KFG modelliert.
3. Ändern Sie Ihren Kalkül so, daß in ihm nur noch Linksableitungen (das heißt, bei jedem Ableitungsschritt wird das am weitesten links stehende Nichtterminalzeichen ersetzt) möglich sind.

Aufgabe IV.2 ($P^{\leftrightarrow}$ (3T)) Sei P das folgende Programm:

$$\begin{array}{lcl}
\mathsf{list(nil)} & \leftarrow & \top \\
\mathsf{list(cons(X, Xs))} & \leftarrow & \mathsf{list(Xs)} \\
\mathsf{app(nil, Ys, Ys)} & \leftarrow & \mathsf{list(Ys)} \\
\mathsf{app(cons(X, Xs), Ys, cons(X, Zs))} & \leftarrow & \mathsf{app(Xs, Ys, Zs)}
\end{array}$$

Geben Sie $P^{\leftrightarrow}$ an.

Aufgabe IV.3 (CET **(8T))** In der Vorlesung haben wir bei der Definition von CET vorrausgesetzt, daß die Signatur immer unendlich viele Funktionssymbole und mindestens ein Konstantensymbol enthält. In dieser Aufgabe nehmen wir jetzt an, daß die Signatur nur noch **endlich** viele Funktions- und Konstantensymbole, aber weiterhin mindestens ein Konstantensymbol, enthält.

1. Ist CET unter dieser Voraussetzung eine endliche Formel menge?
2. Ist $\mathcal{H}$ noch ein Modell von CET?
3. Ist CET weiterhin vollständig? **Hinweis:** Das ist nicht schwer! Wenn Sie glauben, daß CET jetzt unvollständig ist, überlegen Sie (aber ohne Beweis), welche Sätze Sie zu CET hinzufügen würden, um es wieder vollständig zu machen!

Aufgabe IV.4 (Transformation von Termen (12T 8P)) Es gibt ein Transformationsschema $[\![\cdot]\!]$, daß zu einer Gleichung eine logisch äquivalente Formel ϕ berechnet, die nur noch folgendermassen aufgebaut sein darf:

$$\begin{array}{lcl}
\phi, \psi & ::= & u \doteq v \quad | \quad \phi \wedge \psi \quad | \quad \exists x\, \phi \\
u, v & ::= & x \quad | \quad f(y_1, \ldots, y_n)
\end{array}$$

Also zum Beispiel:

$$[\![\mathsf{f(g(a),X)} \doteq \mathsf{f(Y,X)}]\!] = \exists \mathsf{Z}\ (\mathsf{f(Z,X)} \doteq \mathsf{f(Y,X)} \wedge \exists \mathsf{W}(\mathsf{Z} \doteq \mathsf{g(W)} \wedge \mathsf{W} \doteq \mathsf{a})).$$

1. Geben Sie so ein Transformationsschema $[\![\cdot]\!]$ an.

2. Zeigen Sie: $s \doteq t \models\!\!\!\!\models [\![s \doteq t]\!]$.

3. Wir definieren jetzt zwei Theorien CET′ und CET″. Bei CET′ ändern sich die Axiomenschemata Ax1 und Ax2 wie folgt:

$$\begin{array}{llll} \text{Ax1}': & \forall(\, f(\overline{x}) \doteq f(\overline{y}) \rightarrow \overline{x} \doteq \overline{y}\,) & & (f \in \Sigma) \\ \text{Ax2}': & \forall(\, f(\overline{x}) \doteq g(\overline{y}) \rightarrow \bot\,) & & (f, g \in \Sigma, f \neq g) \end{array}$$

   Bei CET″ ändert sich zusätzlich noch Ax3:

$$\text{Ax3}: \quad \forall(\, x \doteq f(\overline{y}) \rightarrow \bot\,) \qquad (f \in \Sigma, x \in \mathcal{V}(\overline{y}))$$

   Zeigen oder widerlegen Sie die beiden folgenden Aussagen:

$$\text{CET} \models\!\!\!\!\models \text{CET}' \qquad \text{und} \qquad \text{CET} \models\!\!\!\!\models \text{CET}''.$$

4. Implementieren Sie Ihr Transformationsschema $[\![\cdot]\!]$ in einem PROLOG-Programm. **Hinweis:** Überlegen Sie sich eine günstige Darstellungsform für Formeln und Terme. Achten Sie insbesondere auf die Darstellung von Variablen: Sie können diese entweder als PROLOG–Variablen darstellen und zu Ihrer Manipulation nicht-logische Prozeduren[1] von PROLOG benutzen, oder aber Variablen als normale Terme codieren.

Aufgabe IV.5 (Das n–Damen Problem (8P)) Sie kennen ja bestimmt das n–Damen Problem: Auf einem Schachbrett mit $n \times n$–Feldern sollen n Damen so positioniert werden, daß keine der Damen eine andere bedroht.

1. Schreiben Sie ein PROLOG–Programm, daß dieses Problem für beliebiges n löst.

2. Schreiben Sie ein besonders **schnelles** und **schönes** Programm. **Belohnung:** Die beste Lösung wird prämiert.

Literatur

[LP81] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall. 1981.

[1] Zum Beispiel: var/1, ==/2 und ähnliches.

V. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte

Ausgabe: 28.5.93 Abgabe: 4.6.93 Abgabe der Programme: 11.6.93

Aufgabe V.1 (Parsen mit Prolog (8P))

Schreiben Sie einen Parser für die folgende Grammatik, die arithmetische Ausdrücke mit Addition und Subtraktion erzeugt. Diese Grammatik besteht aus den Terminalen $\{0, 1, 2, 3, ..., 9, (,), +, -\}$, den Nichtterminalen $\{Expr, Num\}$ und den Produktionen:

$$\begin{array}{lll} Expr & ::= & Num + Num \mid Num - Num \\ Num & ::= & (Expr) \mid 0 \mid 1 \mid 2 \mid 3 \mid ... \mid 9 \,. \end{array}$$

Kodieren Sie terminale Wörter als Differenzlisten von Terminalen in 'Quotes' (Z.B. das Wort $2 + *34$ durch die Differenzliste $['2' , '+' , '*' , '3' , '4' \mid X] - X$).

1. Schreiben Sie einen Akzeptor bestehend aus den Prädikaten `expr/2` und `num/2`. Dabei soll `expr(X,Y)` genau dann gültig sein, wenn `X-Y` eine Differenzliste ist, die einen gültigen Ausdruck kodiert (Analog für `num`).

   Beispiel: `expr(['(','2','-','3',')','+','8'],[]) ist gültig.`

2. Transformieren Sie den Akzeptor zu einem Parser, d.h. berechenen Sie zusätzlich den konkreten Syntaxbaum.

   Beispiel: `expr(['(','2','-','3',')','+','8'],[],Abl)` liefert
   `Abl = e(n(left,e(int(2),minus,int(3)),right),plus,n(int(8)))`.

3. Transformieren Sie den Akzeptor erneut, so daß Sie die korrekten arithmetischen Ausdrücke auswerten, ohne den Ableitungsbaum zu berechnen.

   Beispiel: `expr(['(','2','-','3',')','+','8'],[],Val)` liefert `Val = 7`.

Aufgabe V.2 (Definite Clause Grammar (DCG) (8T))

Grammatiken, die in Form einer sogenannten *Definite Clause Grammar* (DCG) spezifiziert sind, können automatisch in ein PROLOG-Programm transformiert werden: So werden etwa die Produktionen

$$S ::= NP \;\; VP \qquad N ::= hans$$

von der DCG Notation

```
s --> np, vp.    n --> ['hans'].
```

in die Klauseln

```
s(X0,X2) :- np(X0,X1), vp(X1,X2).    n(['hans'|X],X).
```

(resp.) übersetzt. Das Prädikat `s/2` beschreibt einen Akzeptor für die von S erzeugten Sprache, mit der in der letzten Aufgabe verwendeten Differenzlistentechnik.

In dieses Übersetzungsschema lassen sich nun Erweiterungen unmittelbar einbetten:

1. Die Berechnung von Werten oder den Aufbau von Strukturen während des Parsens.
2. Die Verarbeitung von Extrabedingungen.
3. Die Kodierung kontextsensitiver Grammatiken.

Allgemein:

- Ein **DCG-Nichterminal** ist ein Prologterm `p(`$\mathtt{s}_1$`,...,`$\mathtt{s}_n$`)`, in dem `p` das Nichtterminalzeichen P, und die jedes $\mathtt{s}_i$ ein(e) Attribut(struktur) des mit P assoziierten Syntaxbaums beschreibt.
- Ein **DCG-Terminal(wort)** ist eine Liste von Prologatomen `['`$\mathtt{x}_1$`',...,'`$\mathtt{x}_n$`']`.
- Eine **DCG-Regel** ist von der Form *head* –> *body*, wobei *head* ein Nichterminal und *body* eine durch Kommata getrennte Sequenz von Terminalen, Nichtterminalen und **Aufrufen** ist.

  NB: Die Übersetzung dieser Sequenz in den Rumpf einer PROLOG-Klausel verwendet essentiell die links-nach-rechts Strategie von PROLOG!
- Ein **Aufruf** ist ein beliebiges PROLOG-Ziel in {}.

Beispielsweise wird

```
dat(d(T,M,J)) --> tag(T), mon(M), jahr(J), { ...0 < M, M < 13 ...}
```

zu

```
dat(d(T,M,J),X0,X3) :- tag(T,X0,X1),mon(M,X1,X2),jahr(J,X2,X3), ..., 0<M, M<13, ....
```

Aufgabe:

1. Lesen Sie zur Ergänzung dieser knappen Einführung in [SS86] den Abschnitt über DCGs.
2. Schreiben Sie (auf dem Papier) eine DCG für die bekannte kontextsensitive Sprache $\{a^n b^n c^n | n \leq 0\}$.

Aufgabe V.3 (Entailment/Disentailment(8T))
Wir betrachten die folgenden Constraints über der Herbrandstruktur $\mathcal{H}$:

$$\phi, \psi ::= t \doteq s \mid \phi \wedge \psi \mid \top \mid \bot \quad .$$

Geben Sie ein Verfahren an, das für zwei beliebige Constraints ϕ, ψ über $\mathcal{H}$ entscheidet, ob:

- $\phi \models_{\mathcal{H}} \psi$ (Entailment)
- $\phi \models_{\mathcal{H}} \neg\psi$ (Disentailment)

gilt, und beweisen Sie die Korrektheit Ihres Verfahrens.

Literatur

[SS86] Leon Sterling and Ehud Shapiro. *The Art of Prolog.* The MIT Press, 1986.

VI. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte

Ausgabe: 1.6.93 Abgabe: 8.6.93

Aufgabe VI.1 (Abhängigkeit und CET (10T))

Sei $\Sigma = \{f_1, \ldots, f_2\}$ eine prädikatenlogische Signatur, die mindestens eine Konstante und ein ein- oder mehrstelliges Funktionssymbol enthält. Wir betrachten das Constraintsystem

$$\Gamma = (\Sigma, CET(\Sigma) \cup DCA(\Sigma), CON),$$

wobei die Menge der Constraints CON durch

$$\varphi, \psi ::= s \doteq t \mid \exists x \varphi \mid \varphi \wedge \psi \mid \top \mid \bot$$

gegeben ist. $DCA(\Sigma)$ ist die einelementige Menge mit dem sogenannten 'domain closure axiom'

$$\forall x\, (\exists \overline{y}_1\, x \doteq f_1(\overline{y}_1)) \vee \ldots \vee (\exists \overline{y}_n\, x \doteq f_n(\overline{y}_n)),$$

wobei $\overline{y}_i$ Folgen von Variablen mit geeigneter Länge sind.

1. Überzeugen Sie sich, daß die Termalgebra $\mathcal{T}(\Sigma)$ ein Modell von Γ ist.
2. Zeigen Sie, daß CET keine vollständige Theorie ist.
3. Zeigen Sie, daß Γ **nicht** unabhängig ist, d.h. es gibt Constraints $\varphi, \psi_1, \ldots \psi_n$, so daß $\varphi \models_\Gamma \vee_{i=1}^n \psi_i$ gilt und für alle i $\varphi \not\models_\Gamma \psi_i$ erfüllt ist.

   Hilfe: Setzen sie $\varphi = \top$ und verwenden Sie existentielle Quantifizierung zur Angabe der ψ_i.

Aufgabe VI.2 (Ganzzahlige Constraints (16T)) Sei $\mathbb{Z}$ die Struktur der ganzen Zahlen mit der Signatur $\Sigma = \{\leq, 0, -1, 1, -2, 2, \cdots\}$ und $\mathsf{VAR} = \{\mathsf{X}, \mathsf{Y}, \ldots\}$ eine Menge von Variablen. Zudem sei die abstrakte Syntax

$$\begin{array}{rcll} x, y, z & ::= & \mathsf{X}, \mathsf{Y}, \ldots & \textit{Variablen} \\ m, n & ::= & 0 \mid -1 \mid 1 \mid -2 \mid \cdots & \textit{Konstanten} \\ s, t & ::= & x \mid m & \textit{Terme} \\ \varphi, \psi & ::= & s \,\dot{\leq}\, t \mid x \,\dot{\neq}\, m \mid \bot \mid \top \mid \varphi \wedge \psi & \textit{Constraints} \end{array}$$

gegeben. Die Formel $x \,\dot{\neq}\, m$ entspricht der Formel $\neg x \doteq m$. Im folgenden betrachten wir das Constraintsystem

$$ZD \;=\; (\Sigma, Th(\mathbb{Z}), [\varphi]).$$

1. Ein Constraint φ heißt singulär, wenn er höchstens eine Variable enthält, d.h. die Menge $\mathcal{V}(\varphi)$ entweder ein Singleton oder leer ist. Sei $\{m_1, \cdots, m_k\}$ eine endliche Menge von Zahlen. Zeigen Sie, daß es einen singulären Constraint φ gibt, so daß $\varphi \models\!\!\!\dashv_{\mathbb{Z}} x \doteq m_1 \vee \ldots \vee x \doteq m_k$.

2. Wir geben nun ein Simplifikationssystem S für das Constraintsystem ZD an. Die Kongruenz von S ist durch die Axiomenschemata

$$
\begin{array}{rcll}
\varphi \wedge \psi & \equiv & \psi \wedge \varphi & \textit{Kommutativität} \\
(\varphi \wedge \psi) \wedge \xi & \equiv & \varphi \wedge (\psi \wedge \xi) & \textit{Assoziativität} \\
\varphi \wedge \top & \equiv & \varphi & \textit{Identität}
\end{array}
$$

gegeben. Die Reduktion von S ist durch die folgenden Regeln definiert:

$$
\begin{array}{lrcll}
(1) & m \dot{\leq} x \wedge n \dot{\leq} x \wedge \varphi & \rightarrow & n \dot{\leq} x \wedge \varphi & \textit{falls } m \leq n \\
(2) & x \dot{\leq} m \wedge x \dot{\leq} n \wedge \varphi & \rightarrow & x \dot{\leq} m \wedge \varphi & \textit{falls } m \leq n \\
(3) & m \dot{\leq} x \wedge x \dot{\neq} m \wedge \varphi & \rightarrow & m+1 \dot{\leq} x \wedge \varphi & \\
(4) & x \dot{\leq} m \wedge x \dot{\neq} m \wedge \varphi & \rightarrow & x \dot{\leq} m-1 \wedge \varphi & \\
(5) & x \dot{\leq} m \wedge n \dot{\leq} x \wedge \varphi & \rightarrow & \perp & \textit{falls } m < n \\
(6) & m \dot{\leq} n \wedge \varphi & \rightarrow & \perp & \textit{falls } m > n \\
(7) & \perp \wedge \varphi & \rightarrow & \perp & \textit{falls } \varphi \not\equiv \top
\end{array}
$$

(a) Probieren Sie die Regeln an dem Ausdruck

$$\mathsf{0} \dot{\leq} \mathsf{X} \wedge \mathsf{X} \dot{\neq} -\mathsf{1} \wedge \mathsf{X} \dot{\neq} \mathsf{0} \wedge \mathsf{X} \dot{\neq} \mathsf{3} \wedge -\mathsf{4} \dot{\leq} \mathsf{X}\,.$$

(b) Zeigen Sie, daß die Reduktion $\varphi \rightarrow \psi$ die Äquivalenz $\varphi \models\!\!\dashv_{\mathbb{Z}} \psi$ impliziert.

(c) Beweisen Sie, daß S terminiert.

(d) Sei φ irreduzibel und enthalte keine Gleichungen $x \dot{\leq} y$. Dann ist φ erfüllbar in ZD genau dann, wenn $\varphi \not\equiv \perp$.

(e) Welche Regeln können Sie wegfallen lassen, ohne die Vollständigkeit zu verlieren?

3. Wir erweitern nun S zu einem Simplifikationssystem S', das für ZD vollständig ist. S' geht aus S durch Hinzufügen der folgenden Reduktionsregeln hervor:

$$
\begin{array}{lrcll}
(8) & m \dot{\leq} x \wedge x \dot{\leq} y \wedge n \dot{\leq} y \wedge \varphi & \rightarrow & m \dot{\leq} x \wedge x \dot{\leq} y \wedge m \dot{\leq} y \wedge \varphi & \textit{falls } n < m \\
(9) & x \dot{\leq} m \wedge x \dot{\leq} y \wedge y \dot{\leq} n \wedge \varphi & \rightarrow & x \dot{\leq} n \wedge x \dot{\leq} y \wedge y \dot{\leq} n \wedge \varphi & \textit{falls } n < m \\
(10) & m \dot{\leq} x \wedge x \dot{\leq} y \wedge \varphi & \rightarrow & m \dot{\leq} x \wedge x \dot{\leq} y \wedge m \dot{\leq} y \wedge \varphi & \textit{falls } \neg\exists n : n \dot{\leq} y \in \varphi \\
(11) & x \dot{\leq} y \wedge y \dot{\leq} m \wedge \varphi & \rightarrow & x \dot{\leq} m \wedge x \dot{\leq} y \wedge y \dot{\leq} m \wedge \varphi & \textit{falls } \neg\exists n : x \dot{\leq} n \in \varphi
\end{array}
$$

Dabei schreiben wir $\varphi \in \psi$ genau dann, wenn ein ψ' existiert, so daß $\psi \equiv \varphi \wedge \psi'$.

(a) Reduzieren Sie den Ausdruck $\mathsf{5} \dot{\leq} \mathsf{X} \wedge \mathsf{X} \dot{\leq} \mathsf{Y} \wedge \mathsf{Y} \dot{\leq} \mathsf{Z} \wedge \mathsf{Z} \dot{\leq} \mathsf{X} \wedge \mathsf{Y} \dot{\neq} \mathsf{5} \wedge \mathsf{Z} \dot{\neq} \mathsf{6} \wedge \mathsf{X} \dot{\leq} \mathsf{6}$.

(b) Zeigen Sie: $\varphi \rightarrow \psi$ impliziert $\varphi \models\!\!\dashv_{\mathbb{Z}} \psi$.

(c) Beweisen Sie, daß S' terminiert.

(d) Sei φ irreduzibel. Zeigen Sie, daß φ in $\mathbb{Z}$ genau dann erfüllbar ist, wenn $\varphi \not\equiv \perp$ gilt.

4. Zeigen Sie:

(j) Wenn φ singulär ist, dann existiert ein singulärer Constraint ψ, so daß $\neg\varphi \models\!\!\dashv_{\mathbb{Z}} \psi$.

(k) Wenn ω eine Boolsche Kombination (mit $\neg, \vee, \wedge$) von singulären Constraints mit der gleichen Variable ist, dann gibt es einen singulären Constraint φ mit $\omega \models\!\!\dashv_{\mathbb{Z}} \varphi$.

Zusatzaufgabe VI.3 (*Challenge*: **Unabhängigkeit linearer Arithmetik (16T)**)
Wir betrachten das Constraintsystem $\Gamma = (\Sigma, Th(\mathbb{Z}), [\varphi])$ der linearen Arithmetik über den ganzen Zahlen. Dabei ist Σ die prädikatenlogische Signatur $\Sigma = \{+, -, \cdot, 0, -1, +1, -2, +3, \ldots\}$ und die abstrakte Syntax

$$\begin{array}{rcll} x, y, z & ::= & \mathsf{X}, \mathsf{Y}, \ldots & \textit{Variablen} \\ m, n & ::= & 0 \mid -1 \mid 1 \mid -2 \mid \cdots & \textit{Konstanten} \\ s, t & ::= & x \mid m \mid s + t \mid s - t \mid m \cdot t & \textit{Terme} \\ \varphi, \psi & ::= & s \doteq t \mid \varphi \wedge \psi \mid \bot \mid \top & \textit{Constraints}\,. \end{array}$$

1. Eine Normalform ist ein Constraint der Form $\top$ oder

$$n_1 \cdot x_1 \doteq t_1 \wedge \ldots \wedge n_k \cdot x_k \doteq t_k \qquad (n \geq 1)$$

mit paarweise verschiedenen x_i und $\{x_1, \ldots, x_k\} \cap (\cup_{i=1}^{k} \mathcal{V}(t_i)) = \emptyset$.

   (a) Es gibt einen Algorithmus, der Constraints in äquivalente transformiert und entweder in einer Normalform oder $\bot$ endet. Wie heißt dieser Algorithmus? Welche Struktureigenschaft von $\mathbb{Z}$ nutzt dieser Algorithmus aus?

   (b) Eine Normalform φ heißt x–orientiert, falls $x \notin \mathcal{V}(\varphi)$ oder $x \in \{x_1, \ldots, x_n\}$ gilt. Berechnen Sie eine y–orientierte Normalform des Constraints

$$3 \cdot \mathsf{X} + \mathsf{Y} \doteq 27 \cdot \mathsf{Z} - 1 \;\wedge\; 2 \cdot \mathsf{Z} \doteq 5 \cdot \mathsf{Y} + 3.$$

   Ist dieser Constraint in $\mathbb{Z}$ lösbar?

   (c) Zeigen Sie, daß keine Normalform außer $\top$ in $\mathbb{Z}$ gültig ist.

2. Beweisen Sie, daß die Unabhängigkeit von Γ äquivalent dazu ist, daß für beliebige Normalformen $\psi_1, \ldots, \psi_n$ gilt:

$$\models_{\mathbb{Z}} \vee_{i=1}^{n} \psi_i \quad \Leftrightarrow \quad \exists i \in \{1, \ldots, k\} : \; \psi_i = \top\,.$$

Hinweis: Eine Normalform φ kann als 'Quasi-Substitution' θ verwendet werden, so daß $\varphi \models_{\mathbb{Z}} \psi$ äquivalent ist zu $\models_{\mathbb{Z}} \theta\psi$.

3. Beweisen Sie die Unabhängigkeit von Γ mittels des folgenden Lemmas:

Seien $\psi_1, \ldots, \psi_n$ Normalformen verschieden von $\top$ und $n \geq 0$. Dann gibt es eine Variablenbelegung $\alpha : Var \to \mathbb{Z}$, die keineLösung von $\vee_{i=1}^{n} \psi_i$ in $\mathbb{Z}$ ist.

Hinweis: Benutzen Sie Induktion über die Anzahl der Variablen in $\vee_{i=1}^{n} \psi_i$ und verwenden Sie die Existenz orientierter Normalformen.

VII. Übungsblatt zur Vorlesung
"Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte

Ausgabe: 8.6.93 Abgabe: 15.6.93 Abgabe der Programme: 22.6.93

Aufgabe VII.1 (Saraswats Tell–Operatoren (2T)) Der Student K. Lug weist entrüstet zurück, daß Saraswats $\mathsf{atell}\,[\cdot]$– und $\mathsf{etell}\,[\cdot]$–Operatoren nötig sind. Denn er behauptet, folgende Konstruktionen im Kalkül $K(\Gamma)$ simulieren Saraswats Kombinatoren:

$$\begin{array}{lcl} \mathsf{atell}\,[\varphi] & \rightsquigarrow & \langle \varphi; \varphi; \top \rangle \\ \mathsf{etell}\,[\varphi] & \rightsquigarrow & \langle \top; \varphi; \bot \rangle \end{array}$$

Beweisen oder widerlegen Sie die Behauptung von Herrn Lug.

Aufgabe VII.2 (Erfüllbarkeitsvollständige Constraintsysteme (6T)) Sei $\Gamma = (\Sigma, \Delta, \mathrm{CON})$ ein Constraintsystem.

1. Zeigen Sie die Äquivalenzen der Proposition 5.1 auf Seite 49 des Skriptes.
2. Welche Richtungen ($\Rightarrow$ oder/und $\Leftarrow$) gelten in der folgenden Aussage (Beweisen Sie Ihre Aussage):
$$\Gamma \text{ ist erfüllbarkeitsvollständig} \quad \Longleftrightarrow \quad \Delta \text{ ist vollständig.}$$

Aufgabe VII.3 (Der Kalkül $K(\Gamma)$ (10T)) Sei Γ ein Constraintsystem.

1. Zeigen Sie, daß die Reduktionsrelation von $K(\Gamma)$ terminierend ist.
2. Ist $K(\Gamma)$ Turing-vollständig?
3. In dem Kalkül $K(\Gamma)$ sind die Wächter **flach**, das heißt in ihnen dürfen nur Constraints φ, jedoch keine Agenten α auftreten. Wir wollen nun studieren, was wir erreichen, wenn wir **tiefe Wächter** zulassen, das heißt die Syntax des Kalküls $K'(\Gamma)$ wird erweitert zu:

$$\begin{array}{lcl} \varphi,\, \psi & : & \text{Constraint von } \Gamma \\ \sigma,\, \tau & ::= & \varphi \quad | \quad \alpha \quad | \quad \sigma \wedge \tau \\ \alpha,\, \beta & ::= & \langle \sigma; \tau; \tau' \rangle \quad | \quad \top \quad | \quad \alpha \wedge \beta \end{array}$$

Bei der Ableitungsrelation müssen wir jetzt dafür sorgen, daß auch in den Wächtern der Konditionale selber wieder Konditionale reduziert werden können. Um dieses jedoch durchführen zu können, müssen alle Constraints der Tafel eventuell auch in den Wächtern explizit sichtbargemacht werden.

(a) Geben Sie eine Kongruenz $\equiv$ an, die dieses „Sichtbarmachen“ der Constraints formalisiert.

(b) Geben Sie die Ableitungsrelation $\longrightarrow$ an. **Hinweis:** Bei Dissubsumption sollte Reduktion so früh (?!) wie möglich stattfinden.

(c) Argumentieren Sie, daß der Kalkül weiterhin terminierend ist!

4. Wie kann man die Disjunktion $\sigma \vee \tau$ im Kalkül $K'(\Gamma)$ durch Negation darstellen? Was muß man dabei für Anforderungen an das Constraintsystem Γ stellen? Welche Probleme besitzt Ihre Darstellung von $\vee$ im Kalkül $K'(\Gamma)$?

Aufgabe VII.4 (Unifikation (8T)) In dieser Aufgabe wollen wir ein paar Eigenschaften von Substitutionen und Unifikatoren untersuchen. **Zur Erinnerung:** Eine Substitution ist ein Unifikator der Terme s und t, falls $\theta s = \theta t$. Ein Unifikator θ_1 von s, t ist allgemeiner als ein Unifikator θ_2 von s, t, wenn es eine Substitution ρ gibt, so daß $\theta_2 = \rho \circ \theta_1$. Ein Unifikator θ von s, t ist allgemeinst, wenn er allgemeiner als jeder Unifikator von s, t ist. Eine Substitution θ ist idempotent, wenn $\theta = \theta \circ \theta$.

Ein neuer Begriff ist der folgende. Ein Unifikator zweier Terme s, t heißt **normiert**, wenn

$$x \in \mathcal{D}om(\theta) \;\Rightarrow\; x \in \mathcal{V}(s) \cup \mathcal{V}(t).$$

1. Für eine Substitution θ sind die folgenden Aussagen äquivalent:
   (a) θ ist idempotent.
   (b) $\mathcal{D}om(\theta) \cap \mathcal{V}(\mathcal{R}an(\theta)) = \emptyset$.
   (c) Für je zwei Variablen x, y mit $y \in \mathcal{V}(\theta(x))$ gilt $\theta(y) = y$.
2. Seien θ_1 und θ_2 zwei allgemeinste, normierte Unifikatoren von s und t. Dann gibt es eine Variablenumbennenung ρ mit $\theta_1 = \rho \circ \theta_2$. Zeigen Sie auch, daß die Normiertheit essentiell ist für die Richtigkeit der Aussage.
3. Sind prinzipale Unifikatoren eindeutig bestimmt?

Aufgabe VII.5 (Subsumption für CET **(4T 16P 10B))** In dieser Aufgabe wollen wir uns genauer mit dem Problem beschäftigen, die Subsumptionsrelation $\models_{\mathrm{CET}}$ auf Booleschen Kombinationen zu entscheiden.

1. Geben Sie einen Kalkül (die Ableitungsrelation reicht) an, der zu einer gegebenen Booleschen Kombination von Constraints die disjunktive Normalform berechnet.
2. **Challenge:** Zeigen Sie, daß die Ableitungsrelation Ihres Kalküles terminierend ist (Hierfür gibt es 10 Bonuspunkte).
3. Implementieren Sie ein Entscheidungsverfahren in Prolog, das sowohl Subsumption als auch Dissubsumption von Booleschen Kombinationen von Constraints von CET entscheidet. **Hinweis:** Benutzen Sie dabei den oben entwickelten Kalkül zum Herstellen disjunktiver Normalformen. Gehen Sie modular wie bei der Herleitung in der Vorlesung vor.

VIII. Übungsblatt zur Vorlesung "Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte

Ausgabe: 15.6.93   Abgabe: 22.6.93

Aufgabe VIII.1 (Formalisierung von Komplexitätsmaßen (2T)) In der Vorlesung haben Sie die Formulierung: „Summe der Tiefen der Auftreten von existentiell quantifizierten Variablen $\overline{x}$ in $\exists\overline{x}\, M$" kennengelernt. Geben Sie eine exakte Definition dieses Komplexitätsmaßes an.

Aufgabe VIII.2 (Die Theorie FCET (2T)) Zeigen Sie, daß die Struktur $\mathcal{H}$ ein Modell von FCET ist.

Aufgabe VIII.3 (Simplifikation zu $\left\{\begin{smallmatrix}\bot\\\top\end{smallmatrix}\right\}$ (6T)) In der Vorlesung fehlte die Angabe eines Verfahrens, das Boolesche Kombinationen von $\top$ und $\bot$ zu $\left\{\begin{smallmatrix}\bot\\\top\end{smallmatrix}\right\}$ simplifiziert. Geben Sie einen kompletten Kalkül an, der dieses leistet. Zeigen Sie, daß die Reduktionsrelation Ihres Kalküls terminierend ist, und daß irreduzible Ausdrücke gerade $\bot$ oder $\top$ sind.

Aufgabe VIII.4 (Normalformen (4T)) Geben Sie zu der Formel

$$\exists \mathsf{X}_2\,(\mathsf{p}(\mathsf{X},\mathsf{f}(\mathsf{Y}) \wedge \neg\,(\exists \mathsf{X}_1\, \mathsf{X}_2 \doteq \mathsf{X}_1 \vee \neg\forall \mathsf{X}_1 \exists \mathsf{X}_2\,(\mathsf{q}(\mathsf{X}_2) \wedge \mathsf{X}_1 \not\doteq \mathsf{f}(\mathsf{X}_2) \vee \mathsf{Z} \doteq \mathsf{X})))$$

eine logisch äquivalente Formel in Pränex–Normalform an, deren Matrix in DNF ist.

Aufgabe VIII.5 (Quantoreneliminierung in FCET (6T)) Sei Σ die Signatur, die die Konstante c, das einstellige Funktionszeichen f und das zweistellige Funktionszeichen g enthält. Ist die Σ–Formel

$$\exists \mathsf{Z} \neg \exists \mathsf{X}_1 \left(\begin{array}{l} \forall \mathsf{Y}_1 \forall \mathsf{Y}_2\, \mathsf{g}(\mathsf{f}(\mathsf{Z}),\mathsf{c}) \not\doteq \mathsf{g}(\mathsf{f}(\mathsf{f}(\mathsf{Y}_1)),\mathsf{c}) \wedge \\ \exists \mathsf{X}_2 \forall \mathsf{Y}_1\,(\mathsf{X}_1 \doteq \mathsf{Y}_1 \vee \forall \mathsf{Y}_2\,(\mathsf{Y}_1 \not\doteq \mathsf{f}(\mathsf{Y}_2) \vee \mathsf{f}(\mathsf{Y}_2) \doteq \mathsf{f}(\mathsf{f}(\mathsf{X}_2)) \vee \mathsf{X}_1 \doteq \mathsf{f}(\mathsf{X}_2))) \end{array} \right)$$

gültig, erfüllbar oder unerfüllbar in FCET?

Aufgabe VIII.6 (Konjunktive Normalform (4T 8B)) Geben Sie einen Kalkül an, der beliebige Formeln (also nicht unbedingt in Negationsnormalform!) in konjunktive Normalform bringt, und zeigen Sie die Terminierung analog zu dem Beweis im Skript. **Bonus:** Geben Sie auch einen Kalkül ohne Kongruenz an. Das heißt, dieser Kalkül muß durch Ableitungsregeln die Assoziativität der Konjunktion und Disjunktion reflektieren!

Aufgabe VIII.7 (Multimengen–Ordnung (4T)) Sei $(M,\leq)$ eine partielle Ordnung. Zeigen Sie:

1. $(M,\leq)$ wohlfundiert $\Rightarrow$ $(FMS(M),\leq_\mu)$ wohlfundiert.
2. $(M,\leq)$ total $\Rightarrow$ $(FMS(M),\leq_\mu)$ total. Dabei heißt eine partielle Ordnung $(M,\leq)$ total, wenn zusätzlich für alle $n,m\in M$ gilt: $n\leq m$ oder $m\leq n$.

IX. Übungsblatt zur Vorlesung "Logische Programmierung" SS 93

Prof. Gert Smolka
Martin Müller, Joachim Niehren, Christian Schulte

Ausgabe: 22.6.93 Abgabe: 29.6.93 Abgabe der Programme: 6.7.93

Aufgabe IX.1 (Das Reduktionssystem COL (4T))
Simplifizieren Sie das folgende Gleichungssystem mittels COL:

$$Y \doteq f(g(X), g(X)) \wedge Z \doteq f(V, W) \wedge W \doteq g(g(W)) \wedge V \doteq g(g(g(V))) \wedge Y \doteq Z$$

Veranschaulichen Sie sich zunächst anhand einer Skizze die Bäume Y und Z.

Ist das Gleichungssystem in CET erfüllbar?

Aufgabe IX.2 (Erfüllbarkeit von Determinanten (6T))
Geben Sie einen Algorithmus an, der entscheidet, ob eine Determinante δ in CET erfüllbar ist.

Hinweis: Der Algorithmus ist Ihnen aus den Grundstudium bekannt.

Aufgabe IX.3 (Das Constraintsystem RAT (6T))
Die Algebren der rationalen $\mathcal{R}(\Sigma)$ und der unendlichen Bäume $\mathcal{J}(\Sigma)$ sind in der Vorlesung definiert worden. Sie unterscheiden sich von der Termalgebra nur dadurch, daß sie im Unterschied zu dieser über den Universen der rationalen bzw. unendlichen Bäume definiert sind.

Rekapitulieren Sie die Axiomatisierung der Theorie der rationalen Bäume RAT(Σ), und zeigen Sie, daß $\mathcal{R}(\Sigma)$ und $\mathcal{J}(\Sigma)$ Modelle von RAT(Σ) sind.

Aufgabe IX.4 (Konditional versus Disjunktion (2P))
Die beiden unten angegebenen Prädikate implementieren den Minimumsoperator in Oz. Inwiefern unterscheiden sich die beiden Versionen?

```
exists Min1 Min2 in

pred {Min1 X Y Z}
  if
      X ≤ Y then Z = X
   [] Y ≤ X then Z = Y
  fi
end

pred {Min2 X Y Z}
  or
      X ≤ Y Z = X
   [] Y ≤ X Z = Y
  ro
end
```

Aufgabe IX.5 (Prädikate höherer Stufe (8P))
Schreiben Sie ein Prädikat {`ForSome List Prop`}, dessen Argumente eine Liste bzw. ein einstelliges Prädikat sind, und das genau dann zu `true` reduziert, wenn *Prop* für mindestens ein Element $e \in List$ gilt. Beispielsweise soll

exists P **in**

pred {P X}
  X $>$ 2
end

if {ForSome 1|2|3|4|nil P} **then** {Show 'y'} **else** {Show 'n'} **fi**

die Ausgabe `y` erzeugen.

Aufgabe IX.6 (Quicksort (16P))
Implementieren Sie Quicksort in Oz. Dabei sollte die Top-Level Prozedur mit einem zweistelligen Ordnungsprädikat `LessThan` parametrisierbar sein.

Beispiel: {`QuickSort 2|1|4|0|nil`} `LessThan 0|1|2|4|nil`}

Modifizieren Sie Ihr Programm so, daß das `QuickSort`-Prädikat “als Produzent” der sortierten Liste fungiert, d.h., daß nur “bei Bedarf” das nächste Element der sortierten Liste generiert wird:

Beispiel:

`exists X in`	
`{QuickSort 2|1|4|0|nil LessThan X}`	suspendiert
`X = Y|Z`	liefert `Y=0` etc.

Aufgabe IX.7 (Synchrone Kommunikation (8P))
Die Kommunikation in Oz ist vollständig asynchron. Das heißt, daß weder `Put` noch `Get` suspendieren, sondern beide (wenn die Kommunikation ermöglicht ist) sofort reduzieren, unabhängig davon, ob von der anderen Seite ein Kommunikationswunsch vorliegt. Schreiben Sie zwei Oz-Prädikate {`SynchPut Msg Ch`} und {`SynchGet Msg Ch`}, mit der entsprechenden synchronen Funktionalität. D.h. ein {`SynchPut X Ch`} etwa suspendiert solange, bis ein anderer Agent ein `Get` oder `SynchGet` auf diesen Kanal abgesetzt hat.