Übung zur Vorlesung

Automatisierte Logik und Programmierung

Prof. Chr. Kreitz

Universität Potsdam, Theoretische Informatik — Sommersemester 2014

Blatt 4 — Abgabetermin: 9.7.2014

Aufgabe 4.1 (Optimierung)

In Übung 3 haben wir einen Globalsuch-Algorithmus für das n-Damen Problem synthetisiert.

```
FUNCTION queens(n:ℤ):Set(Seq(ℤ)) WHERE n≥1
   RETURNS {nq | perm(nq,{1..n}) ∧ safe(nq)}
≡ if nodups([]) ∧ safe([]) then queens_gs(n,[]) else ∅
FUNCTION queens_gs(n:ℤ,V:Seq(ℤ)):Set(Seq(ℤ))
  WHERE n≥1 ∧ range(V)⊆{1..n} ∧ nodups(V) ∧ safe(V)
  RETURNS {nq | perm(nq,{1..n}) ∧ V⊑nq ∧ safe(nq)}
≡     {nq | nq∈{V} ∧ perm(nq,{1..n}) ∧ safe(nq)}
  ∪ ⋃{ queens_gs(n,W) | W∈{V·i|i∈{1..n}} ∧ nodups(W) ∧ safe(W)}
```

Optimieren Sie diesen Algorithmus mit den in der Vorlesung vorgestellten Techniken.

Aufgabe 4.2 (Rückblick, 1. Teil)

Die folgenden Aufgaben sind vorgesehen als Kontrollfragen zur Überprüfung des eigenen Kenntnisstandes. Sie entsprechen in ihrer Thematik dem Spektrum einer mündlichen Prüfung. Die Antworten sind größtenteils im Skript bzw. auf den Folien enthalten, allerdings nur selten an auffälliger Stelle. Versuchen sie, diese zunächst ohne Ihre Unterlagen zu beantworten.

4.2–a Beschreiben Sie die wichtigsten Bestandteile eines Beweisassistenzsystems.

4.2–b Welche prinzipiellen Möglichkeiten einer Computerunterstützung für formaler Beweise gibt es?

4.2–c Welche Doppelrolle spielt die Sprache ML für interaktiver Beweissysteme?

4.2–d Durch welche Datenstruktur werden Sequenzenbeweise in ML repräsentiert?
Nennen sie auch die wichtigsten Zugriffs- und Manipulationsoperatoren.

4.2–e Welcher Mechanismus ist erforderlich, um konservative Erweiterungen der Typentheorie durch Abstraktionen fehlerfrei zu ermöglichen?

4.2–f Erklären Sie die besondere Rolle von Taktiken bei der Beweisführung.

4.2–g Warum sind Beweise, die mit Taktiken geführt werden, immer korrekt?

4.2–h Auf welche Arten kann ein Benutzer Taktiken konstruieren/programmieren? Nennen Sie Beispiele.

4.2–i Was ist der Unterschied zwischen Rewriting und taktischer Beweisführung? Wie kann man die beiden Techniken kombinieren?

4.2–j Unter welchen Voraussetzungen kann man Entscheidungsprozeduren in Beweissystemen einsetzen?

4.2–k Warum gibt es keine Entscheidungsprozedur für die gesamte Arithmetik?

4.2–l Welche Art von Problemen entscheidet die Prozedur `arith`?

4.2–m Erklären Sie die grundsätzliche Arbeitsweise der Prozedur `arith`.

4.2–n Erklären Sie die grundsätzliche Arbeitsweise der Prozedur `supinf`.

4.2–o Erklären Sie die grundsätzliche Arbeitsweise der Prozedur `equality`.

4.2–p Welche Probleme besitzen Beweissysteme, die sich ausschließlich auf Entscheidungsprozeduren und Benutzerinteraktion stützen?

4.2–q Welche Arten von Beweisführung gibt es für die Logik erster Stufe und wie funktionieren sie?

4.2–r Beschreiben Sie die Funktionsweise von JProver.

4.2–s Aus welchen Kompoenenten besteht eine formale Theorie? Geben Sie ein beispiel.

4.2–t Beschreiben Sie die wichtigsten Aufgaben formaler Wissensbanken?
Auf welche Arten kann man formale Wissensbanken gestalten?

4.2–u Auf welche Arten kann man das Benutzerinterface von Beweisassistenten gestalten. Geben Sie Vor- und Nachteile an.
Welche Besonderheit besitzt der Termeditor von NuPRL und welche Vorteile ergeben sich daraus?

Aufgabe 4.3 (Rückblick, 2.Teil)

4.3–a Warum kann es ein vollständiges Programmsynthesesystem niemals geben?

4.3–b Beschreiben Sie die Phasen einer formalen Entwicklung von Software.

4.3–c Welche Komponenten von Programmen und Spezifikationen sind für Programmsynthese von Bedeutung?

4.3–d Geben Sie eine präzise Definition der Begriffe "Programmkorrektheit" und "Erfüllbarkeit von Spezifikationen" auf der Basis der oben genannten Komponenten.

4.3–e Welche grundsätzlichen Paradigmen für Programmsynthese kennen Sie? Beschreiben Sie die grundsätzlichen Unterschiede bei der Darstellung des Problems und der Vorgehensweise zur Lösung.

4.3–f Welches Problem führt dazu, daß das Prinzip "Beweise-als-Programme" in der Praxis nur von geringer Bedeutung ist?

4.3–g Was ist die charakterisierende Eigenschaft einer Algorithmentheorie?

4.3–h Beschreiben Sie das generelle Verfahren zur Synthese von Algorithmen mit Hilfe von Algorithmenschemata. Wodurch wird dieses Vorgehen gerechtfertigt?

4.3–i Welche Vorteile hat die Synthese mit Algorithmenschemata gegenüber anderen Verfahren?

4.3–j Beschreiben Sie die allgemeine Struktur von Divide & Conquer Algorithmen und die Voraussetzungen für ihre Korrektheit.
Beschreiben Sie eine Strategie zur Erzeugung von Divide & Conquer Algorithmen (an einem Beispiel)

4.3–k Beschreiben sie die allgemeine Struktur von Globalsuchalgorithmen und die Voraussetzungen für ihre Korrektheit. Beschreiben Sie die allgemeine Strategie zur Erzeugung von Globalsuchalgorithmen.

4.3–l Auf welche Art könnte man Synthesestrategien in das formale Konzept typentheoretischer Schlußfolgerungen integrieren?

4.3–m Beschreiben Sie die allgemeine Struktur von Problemreduktionsgeneratoren und die Voraussetzungen für ihre Korrektheit. Beschreiben Sie die Strategie zur Erzeugung von Problemreduktionsgeneratoren.

4.3–n Beschreiben Sie die allgemeine Struktur von Lokalsuchalgorithmen und die Voraussetzungen für ihre Korrektheit. Beschreiben Sie die allgemeine Strategie zur Erzeugung von Lokalsuchalgorithmen.

4.3–o Beschreiben Sie die wichtigsten Techniken zur Optimierung von synthetisierten Algorithmen.

Lösung 4.1

Die Lösung folgt der Vorgehensweise des Costas-Arrays Problems (animierte Version der Folien). Die Lemmata sind ähnlich - wir geben hier nur wesentlichen die Schritte an.

```
FUNCTION queens(n:ℤ):Set(Seq(ℤ)) WHERE n≥1
   RETURNS {nq | perm(nq,{1..n}) ∧ safe(nq)}
≡ if nodups([]) ∧ safe([]) then queens_gs(n,[]) else ∅
FUNCTION queens_gs(n:ℤ,V:Seq(ℤ)):Set(Seq(ℤ))
  WHERE n≥1 ∧ range(V)⊆{1..n} ∧ nodups(V) ∧ safe(V)
  RETURNS {nq | perm(nq,{1..n}) ∧ V⊑nq ∧ safe(nq)}
≡     {nq | nq∈{V} ∧ perm(nq,{1..n}) ∧ safe(nq)}
  ∪ ⋃{ queens_gs(n,W) | W∈{V·i|i∈{1..n}} ∧ nodups(W) ∧ safe(W)}
```

- Wir beginnen mit der Optimierung der Hauptfunktion

  `nodups([])` und `safe([])` werden mit Lemma B.2.2.4.1 und B.1.11.1. zu `true` vereinfacht. Damit kann die `if`-Anweisung zu einem einfachen Aufruf der Hilfsfunktion im Hauptteil vereinfacht werden und wir erhalten

  ```
  FUNCTION queens(n:ℤ):Set(Seq(ℤ)) WHERE n≥1
     RETURNS {nq | perm(nq,{1..n}) ∧ safe(nq)}
  ≡ queens_gs(n,[])
  ```

- Die erste Menge der Hilfsfunktion wählt `nq` aus einer einelementigen Menge aus und testet dann Bedingungen an `nq`. Die resultierende Menge ist {`nq`}, wenn die bedingungen erfüllt sind und ansonsten leer. Diese Erkenntnis wird ausgedrückt durch das Lemma

  $\{z \mid z \in \{x\} \wedge P[z]\} \quad \equiv \quad \texttt{if } P[x] \texttt{ then } \{x\} \texttt{ else } \emptyset$

  und damit vereinfacht sich der erste Teil der Hilfsfunktion zu dem Ausdruck

  ```
  if perm(V),{1..n}) ∧ safe(V)  then {V}  else ∅
  ```

  Dabei wurde der Kontext noch nicht berücksichtigt. Da `range(V)⊆{1..n} ∧ nodups(V)` und `safe(V)` bereits in den Vorbedingungen steht, kann `safe(V)` entfallen und `perm(V),{1..n})` mit Lemma B.1.13.13 vereinfacht werden zu `{1..n}⊆range(V)`. Übrig bleibt damit nur

  ```
  if {1..n}⊆range(V)  then {V}  else ∅
  ```

- Im anderen Teil können wir mit Lemma B.1.15.6

  $\{\, f[x,t] \mid t \in \{g[x,y] \mid y \in S\,\} \wedge h[t]\} \quad \equiv \quad \{\, f[x,\, g[x,y]\,] \mid y \in S \wedge h[\,g[x,y]\,]\}$

  den geschachtelten Mengenausdruck vereinfachen. Dabei wird jeweils `W` durch `V·i` ersetzt und wir erhalten

  ```
  { queens_gs(n,V·i) | i∈{1..n} ∧ nodups(V·i) ∧ safe(V·i)}
  ```

  Nun können wir wiederum die Vorbedingung verwenden und Lemmata über `nodups`, `append` und beschränkte Allquantoren einsetzen (B.2.24.6 / B.1.11.2)

  ```
  nodups(V·i)  ⇔  nodups(V)  ∧  i∉range(V)
  safe(V·i)    ⇔  safe(V)  ∧  ∀k<|V|.  |V[k]-i|≠ |V|+1-k
  ```

  Man beachte, daß `V·i[|V|+1]=i` und `V·i[k]=V[k]` ist (Lemma B2.14.6/5).

  Da `nodups(V)` und `safe(V)` bereits im Kontext der Vorbedingungen stehen, können wir uns diese aufwendigen Berechnungen ersparen und erhalten

  ```
  { queens_gs(n,V·i) | i∈{1..n} ∧ i∉range(V) ∧ ∀k<|V|. |V[k]-i|≠ |V|+1-k}
  ```

Die Hilfsfunktion lautet nun

```
FUNCTION queens_gs(n:ℤ,V:Seq(ℤ)):Set(Seq(ℤ))
  WHERE n≥1 ∧ range(V)⊆{1..n} ∧ nodups(V) ∧ safe(V)
  RETURNS {nq | perm(nq,{1..n}) ∧ V⊑nq ∧ safe(nq)}
≡    if {1..n}⊆range(V)  then {V}  else ∅
  ∪ ⋃{ queens_gs(n,V·i) | i∈{1..n} ∧ i∉range(V) ∧ ∀k<|V|. |V[k]-i|≠ |V|+1-k}
```

- An dieser Stelle bietet sich endliche Differenzierung an. Der Ausdruck `i∈{1..n} ∧ i∉range(V)` stellt eine ständig wiederkehrende iterative Berechnung dar, die besser inkrementell berechnet wird. Gleiches gilt für die Berechnung von `|V|`. Um dies vorzubereiten vereinfachen wir weiter

  Mit den Lemmata $x \in M \wedge x \notin M' \equiv x \in M \backslash M'$ und $M \subseteq M' \equiv M \backslash M' = \emptyset$ können wir in beiden Teilen der Hilfsfunktion den gleichen Mengendifferenzausdruck {1..n}\range(V) einführen, den wir dann für das endliche Differenzieren verwenden. Wir erhalten

```
FUNCTION queens_gs(n:ℤ,V:Seq(ℤ)):Set(Seq(ℤ))
  WHERE n≥1 ∧ range(V)⊆{1..n} ∧ nodups(V) ∧ safe(V)
  RETURNS {nq | perm(nq,{1..n}) ∧ V⊑nq ∧ safe(nq)}
≡    if {1..n}\range(V)=∅  then {V}  else ∅
  ∪ ⋃{ queens_gs(n,V·i) | i∈{1..n}\range(V) ∧ ∀k<|V|. |V[k]-i|≠ |V|+1-k}
```

  Nun differenzieren wir über die Ausdrücke {1..n}\range(V) (Variable `pool`) und `|V|` (Variable `vs`), was zu folgendem veränderten Funktionenpaar führt

```
FUNCTION queens(n:ℤ):Set(Seq(ℤ)) WHERE n≥1
   RETURNS {nq | perm(nq,{1..n}) ∧ safe(nq)}
≡ queens_gs(n,[],{1..n},0)
FUNCTION queens_gs(n:ℤ,V:Seq(ℤ),pool:Seq(ℤ),vs:ℤ):Set(Seq(ℤ))
  WHERE n≥1 ∧ range(V)⊆{1..n} ∧ nodups(V) ∧ safe(V) ∧ pool={1..n}\range(V) ∧ vs=|V|
  RETURNS {nq | perm(nq,{1..n}) ∧ V⊑nq ∧ safe(nq)}
≡    if pool=∅  then {V}  else ∅
  ∪ ⋃{ queens_gs(n,V·i,pool\{i},vs+1) | i∈pool ∧ ∀k<vs. |V[k]-i|≠ vs+1-k}
```

- Nun bietet sich an, die Fallunterscheidung `if pool=∅ ..` über beide Teile zu distributieren und dann den Ausdruck weiter zu vereinfachen

```
if pool=∅
   then {V} ∪ ⋃{ queens_gs(n,V·i,pool\{i},vs+1) | i∈pool ∧ ∀k<vs. |V[k]-i|≠ vs+1-k}
   else  ∅  ∪ ⋃{ queens_gs(n,V·i,pool\{i},vs+1) | i∈pool ∧ ∀k<vs. |V[k]-i|≠ vs+1-k}
```

  Im ersten Fall verwenden wir das Lemma $\bigcup\{ f(i) \mid i \in \emptyset\} = \emptyset$ um den zweiten Teil der Vereinigung zu ∅ zu vereinfachen. Mit $S \cup \emptyset = \emptyset = \emptyset \cup S$ ergibt sich

```
if pool=∅  then {V}
   else ⋃{ queens_gs(n,V·i,pool\{i},vs+1) | i∈pool ∧ ∀k<vs. |V[k]-i|≠ vs+1-k}
```

- Als Endprodukt aller Optimierungen erhalten wir

```
FUNCTION queens(n:ℤ):Set(Seq(ℤ)) WHERE n≥1
   RETURNS {nq | perm(nq,{1..n}) ∧ safe(nq)}
≡ queens_gs(n,[],{1..n},0)
FUNCTION queens_gs(n:ℤ,V:Seq(ℤ),pool:Seq(ℤ),vs:ℤ):Set(Seq(ℤ))
  WHERE n≥1 ∧ range(V)⊆{1..n} ∧ nodups(V) ∧ safe(V) ∧ pool={1..n}\range(V) ∧ vs=|V|
  RETURNS {nq | perm(nq,{1..n}) ∧ V⊑nq ∧ safe(nq)}
≡    if pool=∅  then {V}
   else ⋃{ queens_gs(n,V·i,pool\{i},vs+1) | i∈pool ∧ ∀k<vs. |V[k]-i|≠ vs+1-k}
```

  Zum Abschluß könnte man noch Datentypverfeinerung durchführen, was – wie bei den Costas-Arrays – dazu führt `V` als umgekehrt verkettete Liste und `pool` als Bitvektor zu implementieren.

Lösung 4.2

4.2–a Beschreiben Sie die wichtigsten Bestandteile eines Beweisassistenzsystems.
Inferenzmaschine, Bibliothek, Benutzerinterface + optionale Komponenten wie Evaluator, Programmextraktion bei konstruktiven Systemen

4.2–b Welche prinzipiellen Möglichkeiten einer Computerunterstützung für formaler Beweise gibt es?
Siehe Seite 183/184: Proof Checking, Proof Editoren, Taktiken, Entscheidungsprozeduren, Theorembeweiser-Suchstrategien

4.2–c Welche Doppelrolle spielt die Sprache ML für interaktiver Beweissysteme?
Siehe Seite 185/186: formale Metasprache des Kalküls und Programmiersprache für Implementierung

4.2–d Durch welche Datenstruktur werden Sequenzenbeweise in ML repräsentiert?
Nennen sie auch die wichtigsten Zugriffs- und Manipulationsoperatoren.
Siehe Seite 191/192: abstrakter Datentyp – i.w. rekursive Baumstruktur. Manipulation NUR durch `refine`. Zugriffe auf Deklarationen, Konklusion, Regel und Nachfolger.

4.2–e Welcher Mechanismus ist erforderlich, um konservative Erweiterungen der Typentheorie durch Abstraktionen fehlerfrei zu ermöglichen?
Siehe Seite 198/199: Substitution und Matching zweiter Stufe

4.2–f Erklären Sie die besondere Rolle von Taktiken bei der Beweisführung.
Siehe Seite 200f: flexible, benutzerdefinierte Erweiterung des Inferenzsystems ohne Sicherheitsprobleme

4.2–g Warum sind Beweise, die mit Taktiken geführt werden, immer korrekt?
Siehe Seite 210: sie können Beweise NUR mithilfe der festen Regeln der Theorie manipulieren.

4.2–h Auf welche Arten kann ein Benutzer Taktiken konstruieren/programmieren? Nennen Sie Beispiele.
Direkt aus Regeln mit `refine`, Kombination vordefinierter Taktiken, Tacticals (was ist das)?, explizite Metalevel-Steuerung für ausgefeilte Taktiken

4.2–i Was ist der Unterschied zwischen Rewriting und taktischer Beweisführung? Wie kann man die beiden Techniken kombinieren?
Rewriting ist Termersetzung ohne Korrektheitsbeweis. Dahinter steht aber immer eine etablierte Gleichheit/Implikatio die als Lemma formuliert zur Validierung der Termersetzung in Taktiken genuzt werden können.

4.2–j Unter welchen Voraussetzungen kann man Entscheidungsprozeduren in Beweissystemen einsetzen?
Siehe Seite 211/212: entscheidbare Teiltheorie, maschinennahe Charakterisierung der Gültigkeit, Konsistenz mit Rest der Theorie... dabei Entscheidungsprozeduren als solche erklären.

4.2–k Warum gibt es keine Entscheidungsprozedur für die gesamte Arithmetik?
Siehe Seite 213: Theorie der rekursiven Funktionen ist darin enthalten.

4.2–l Welche Art von Problemen entscheidet die Prozedur `arith`?
Siehe Seite 214: elementar-arithmetische Ausdrücke – genauer erklären.

4.2–m Erklären Sie die grundsätzliche Arbeitsweise der Prozedur `arith`.
Siehe Seite 218

4.2–n Erklären Sie die grundsätzliche Arbeitsweise der Prozedur `supinf`.
Siehe ....

4.2–o Erklären Sie die grundsätzliche Arbeitsweise der Prozedur `equality`.
Siehe Seite 220

4.2–p Welche Probleme besitzen Beweissysteme, die sich ausschließlich auf Entscheidungsprozeduren und Benutzerinteraktion stützen?
Siehe Seite 223: längere Erklärung

4.2–q Welche Arten von Beweisführung gibt es für die Logik erster Stufe und wie funktionieren sie?
Siehe Kapitel 14, Folie 1 und folgende

4.2–r Beschreiben Sie die Funktionsweise von JProver.
Siehe Kapitel 14, Folien 10ff

4.2–s Aus welchen Kompoenenten besteht eine formale Theorie? Geben Sie ein beispiel.
Notation für Datentypen und Operationen, Axiome, Implementierung durch Abstützung auf bestehende Konzepte

4.2–t Beschreiben Sie die wichtigsten Aufgaben formaler Wissensbanken?
Auf welche Arten kann man formale Wissensbanken gestalten?
Siehe Kapitel 15, Folien 6ff

4.2–u Auf welche Arten kann man das Benutzerinterface von Beweisassistenten gestalten. Geben Sie Vor- und Nachteile an.
Welche Besonderheit besitzt der Termeditor von NuPRL und welche Vorteile ergeben sich daraus?
Siehe Kapitel 15, Folien 17ff
Siehe Seite 194: Struktureditor: kein parser nötig, flexible Syntax.

Lösung 4.3

4.3–a Warum kann es ein vollständiges Programmsynthesesystem niemals geben?
Unentscheidbarkeiten des Syntheseproblems

4.3–b Beschreiben Sie die Phasen einer formalen Entwicklung von Software.
mit Compilierung 6 Phasen

4.3–c Welche Komponenten von Programmen und Spezifikationen sind für Programmsynthese von Bedeutung?

D,R,I,O

4.3–d Geben Sie eine präzise Definition der Begriffe “Programmkorrektheit” und “Erfüllbarkeit von Spezifikationen” auf der Basis der oben genannten Komponenten.
Siehe §17

4.3–e Welche grundsätzlichen Paradigmen für Programmsynthese kennen Sie? Beschreiben Sie die grundsätzlichen Unterschiede bei der Darstellung des Problems und der Vorgehensweise zur Lösung.
Beweise als Programme, Synthese durch Transformationen, Synthese mit Hilfe von Algorithmenschemata.
Dazu die Details angeben

4.3–f Welches Problem führt dazu, daß das Prinzip “Beweise-als-Programme” in der Praxis nur von geringer Bedeutung ist?
niedriges Abstraktionsniveau

4.3–g Was ist die charakterisierende Eigenschaft einer Algorithmentheorie?
Siehe §17 Folie 20: kanonische Erweiterbarkeit zu Programm

4.3–h Beschreiben Sie das generelle Verfahren zur Synthese von Algorithmen mit Hilfe von Algorithmenschemata. Wodurch wird dieses Vorgehen gerechtfertigt?
Siehe §17 Folie 22: Verfahren + Satz

4.3–i Welche Vorteile hat die Synthese mit Algorithmenschemata gegenüber anderen Verfahren?
Siehe §17 Folie 29: Effizienteres Verfahren, bessere Lösungsqualität durch Verarbeitung von Wissen

4.3–j Beschreiben Sie die allgemeine Struktur von Divide & Conquer Algorithmen und die Voraussetzungen für ihre Korrektheit.
Beschreiben Sie eine Strategie zur Erzeugung von Divide & Conquer Algorithmen (an einem Beispiel)
Siehe §18

4.3–k Beschreiben sie die allgemeine Struktur von Globalsuchalgorithmen und die Voraussetzungen für ihre Korrektheit. Beschreiben Sie die allgemeine Strategie zur Erzeugung von Globalsuchalgorithmen.
Siehe §19

4.3–l Auf welche Art könnte man Synthesestrategien in das formale Konzept typentheoretischer Schlußfolgerungen integrieren?
Das haben wir nicht besprochen, aber es steht etwas im Skript dazu und es gibt auch nur einen sinnvollen Weg – Theoreme, die den Übergang beschreiben von der Frage nach der Erfüllbarkeit von Spezifikationen in die Frage nach der Existenz von Komponenten, wleche bestimmte Axiome erfüllen, müssen aufgerufen werden und als Extraktterm die gewünschten Algorithmenstrukturen instantiieren.

4.3–m Beschreiben Sie die allgemeine Struktur von Problemreduktionsgeneratoren und die Voraussetzungen für ihre Korrektheit. Beschreiben Sie die Strategie zur Erzeugung von Problemreduktionsgeneratoren.
Siehe §20

4.3–n Beschreiben Sie die allgemeine Struktur von Lokalsuchalgorithmen und die Voraussetzungen für ihre Korrektheit. Beschreiben Sie die allgemeine Strategie zur Erzeugung von Lokalsuchalgorithmen.
Siehe §21

4.3–o Beschreiben Sie diewichtigsten Techniken zur Optimierung von synthetisierten Algorithmen.
Siehe §22