Übung zur Vorlesung

Automatisierte Logik und Programmierung

Prof. Chr. Kreitz

Universität Potsdam, Theoretische Informatik — Sommersemester 2004

Blatt 3 — Abgabetermin: —

Das dritte Übungsblatt befaßt sich mit der Formalisierung von Problemstellungen, der Erstellung formaler Spezifikationen. und der Synthese durch Transformationen, ohne daß dabei eine spezielle Strategie betrachtet wird.

Aufgabe 3.1 (Formale Spezifikationen)

Geben Sie formale Spezifikationen für die folgenden Probleme an.

3.1–a **Sortierung**: Eine Liste `L` ganzer Zahlen ist geordnet, wenn alle ihre Elemente in aufsteigender Reihenfolge angeordnet sind. Sie ist Sortierung einer anderen Liste `L'`, wenn `L` einer Permutation der Liste `L'` und geordnet ist.

1. Gesucht ist ein Programm, welches eine Liste `L'` ohne doppelte Elemente sortiert.
2. Gesucht ist ein Programm, welches eine beliebige Liste `L'` sortiert.

Hinweis: der Begriff der Permutation ist im zweiten Fall etwas komplexer als im ersten und muß formalisiert werden

3.1–b **8-Dame Problem**: Gegeben ein Schachbrett mit 8x8 Feldern und 8 Dame Figuren. Eine Dame kann alle Figuren schlagen, die sich auf derselben waagerechten oder senkrechten Linie befinden sowie alle Figuren, die sich auf der von ihr ausgehenden Diagonale befinden.

Gesucht ist ein Programm, welches *alle* möglichen Plazierungen der 8 Damen auf dem Schachbrett bestimmt, so daß keine Dame eine andere schlagen kann.

Stellen Sie eventuell notwendige Definitionen für neue Begriffe auf und beschreiben Sie die Gesetze dieser neuen Konzepte.

Aufgabe 3.2

In der Vorlesung haben wir die formalen Grundbegriffe der Programmsynthese beschrieben. Zur Vorbereitung eines formalen Schließens über Programme und ihre Eigenschaften müssen auch diese formalisiert werden. Geben Sie formale Definitionen für die folgenden Konzepte an:

3.2–a Die Klasse aller Spezifikationen als ein Datentyp `SPECIFICATIONS`

3.2–b Die Klasse aller Programme als ein Datentyp `PROGRAMS`

3.2–c Programmkorrektheit als ein "Prädikat" p `ist korrekt` *(beachten Sie Terminierung!)*

3.2–d Erfüllbarkeit von Spezifikationen als ein "Prädikat" *spec* `ist erfüllbar`

3.2–e Die Notation für Programme
`FUNCTION` f`(x:D):R WHERE I(x) RETURNS y SUCH THAT O(x,y) = body(x)`

Aufgabe 3.3 (Synthese durch Transformationen)

Gegeben sei die folgende Spezifikation des Sortierproblems

```
FUNCTION sort(L:Seq(ℤ)):Seq(ℤ)  WHERE true
  RETURNS S  SUCH THAT rearranges(L,S) ∧ ordered(S)
```

wobei das Prädikat `ordered` wie folgt definiert sei

```
ordered(S)      ≡ ∀i∈{1..|S|-1}.S[i]≤S[i+1]
```

Synthetisieren Sie ein Sortierprogramm durch Anwendung von Transformationen.
Stellen Sie dazu eine Spezifikationsformel auf und transformieren Sie die Ausgabebedingung durch Äquivalenzumformungen bis ein rekursives Sortierprogramm entsteht. Begründen Sie, warum das erzeugte Programm terminiert.

Hinweise: Je nach Art der Umformungen können sehr unterschiedliche Algorithmen erzeugt werden. Stellen Sie die nötigen Lemmata, die nicht bereits im Appendix B des Skriptes zu finden sind, separat auf, ohne sie formal zu beweisen.

Aufgabe 3.4 (Korrektheit von Algorithmenerzeugungsregeln)

Die Programme f, f' und g seien bereits als korrekt bekannt und `P` sei ein beliebiges Prädikat.

```
FUNCTION f(x:D):R    WHERE I(x)   RETURNS y  SUCH THAT O(x,y)   =  body_f(x)
FUNCTION f'(x:D):R   WHERE I'(x)  RETURNS y  SUCH THAT O'(x,y)  =  body_f'(x)
FUNCTION g(x,y:R):R' WHERE J(x,y) RETURNS z  SUCH THAT O'(x,y,z) = body_g(x,y)
```

Zeigen Sie (semiformal), daß dann auch die folgenden Programme korrekt sind.

3.4–a
```
FUNCTION F1(x:D):R  WHERE I(x) ∨ I'(x)
    RETURNS y  SUCH THAT O(x,y) ∨ O'(x,y)
  = if I(x) then body_f(x) else body_f'(x)
```

3.4–b
```
FUNCTION F2(x:D):R  WHERE I(x) ∧ P(x,body_f(x))
    RETURNS y  SUCH THAT O(x,y) ∧ P(x,y)
  = body_f(x)
```

3.4–c
```
FUNCTION F3(x:D):R' WHERE I(x) ∧ J(x,body_f(x))
    RETURNS z SUCH THAT ∃y:R. O(x,y) ∧ O'(x,y,z)
  = body_g(x,body_f(x))
```

Lösung 3.0 *Die Loesungen sind z.T. von alten Musterloesungen uebernommen. Ich habe ein paar kleinere Korrekturen vorgenommen aber nicht jedes kleine Detail überprüft.*

Lösung 3.1 *Ziel dieser Aufgabe ist es, die Formalisierung von Problemspezifikationen einzuüben und dabei eventuell fehlende Begriffe zu formalisieren, sofern dies für die Problemstellung sinnvoller ist als die Verwendung der entsprechenden komplexeren Ausdrücke.*

3.1–a Das Problem ist leicht zu beschreiben, wenn man die Begriffe "geordnet" und "Permutation" formalisiert. Eine Liste L ist *geordnet*, wenn ihre Elemente in aufsteigender Reihenfolge angeordnet sind. Dies legt z.B. die folgende Definition nahe.

$\texttt{ordered}(L) \equiv \forall \texttt{i} \in \{\texttt{1..}|L|\texttt{-1}\}.L\texttt{[i]} \leq L\texttt{[i+1]}$

1. Bei der Sortierung von Listen ohne doppelte Elemente kann man auf die bekannte Definition der Permutationen einer Menge S zurückgreifen, denn eine Permutation der Elemente einer Liste L' ist in diesem Fall eine Permutation der *Menge* $\texttt{range}(L')$. Dies führt zu folgender Formalisierung des Sortierproblems

   FUNCTION $sort$(L':Seq($\mathbb{Z}$)):Seq($\mathbb{Z}$) WHERE nodups(L')
   RETURNS L SUCH THAT perm(L,range(L')) $\wedge$ ordered(L)

2. Enthält die Liste L' doppelte Vorkommen von Elementen, so ist obige Beschreibung unbrauchbar. Wir müssen nun eine *echte* Umordnung von Listen definieren, wofür wir ein Prädikat `rearranges(L',L)` neu definieren. Die naheliegendste Definition hierfür besagt, daß `L` aus `L'` durch eine Permutation der Indizes entsteht, also sich beschreiben läßt als $\texttt{[L'[k]} \mid \texttt{k} \in \texttt{I]}$, wobei `I` Permutation von `domain(L')` ist.

   $\texttt{rearranges(L',L)} \equiv \exists \texttt{I:Seq}(\mathbb{Z})\texttt{.perm(I,domain(L'))} \wedge \texttt{L=[L'[k]} \mid \texttt{k} \in \texttt{I]}$

   Dies führt dann zu folgender Problemspezifikation, die natürlich die erste umfaßt.

   FUNCTION $sort$(L':Seq($\mathbb{Z}$)):Seq($\mathbb{Z}$) WHERE true
   RETURNS L SUCH THAT rearranges(L',L) $\wedge$ ordered(L)

3.1–b Die Bedingung dafür, daß die 8 Damen einander nicht schlagen können ist, daß in jeder waagerechten, senkrechten, aufwärts- und abwärts-diagonalen Reihe höchstens eine Dame steht. Da es nur 8 waagerechte und senkrechte Reihen gibt, muß jede Dame in genau einer dieser Reihen stehen. Damit genügt es, anstelle einer Repräsentation des gesamten Schachbretts die waagerechten Positionen der Damen in jeder senkrechten Reihe darzustellen, also eine Folge L von 8 Zahlen zwischen 1 und 8 zu verwalten. Da in jeder waagerechten Zeilee nur eine Dame stehen kann, darf diese Folge keine doppelten Vorkommen enthalten, muß also eine Permutation der Zahlen $\{1..8\}$ sein.

Zusätzlich müssen die aufwärts- und abwärts-diagonalen Reihen sicher sein: steht in Reihe `i` an Position L`[i]` eine Dame, so darf die Position L`[j]` der Dame in Reihe `j` nicht genau L`[i]+(j-i)` oder L`[i]-(j-i)` sein. Dies beschreiben wir durch zwei neue Einzelbegriffe, die wir insgesamt mit dem Begriff `safe(`L`)` zusammenfassen.

$\texttt{free\_up\_diagonal}(L) \equiv \forall \texttt{i} \in \texttt{domain}(L).\forall \texttt{j} \in \texttt{domain}(L).\ \texttt{i} \neq \texttt{j} \Rightarrow L\texttt{[i]+(j-i)} \neq L\texttt{[j]}$
$\texttt{free\_down\_diagonal}(L) \equiv \forall \texttt{i} \in \texttt{domain}(L).\forall \texttt{j} \in \texttt{domain}(L).\ \texttt{i} \neq \texttt{j} \Rightarrow L\texttt{[i]-(j-i)} \neq L\texttt{[j]}$
$\texttt{safe}(L) \equiv \texttt{free\_up\_diagonal}(L) \wedge \texttt{free\_down\_diagonal}(L)$

Die Spezifikation, die sich leicht auf beliebig große Schachbretter verallgemeinern läßt und ein beliebtes Synthesebeispiel ist, lautet nun

FUNCTION $queens$():Set(Seq($\mathbb{Z}$)) WHERE true

RETURNS {nq | perm(nq,{1..8}) ∧ safe(nq)}

Lösung 3.2 *Ziel dieser Aufgabe ist es, die Grundbegriffe der Programmsynthese so zu formalisieren, daß man später formale Beweise über die Synthetisierbarkeit von Programmen führen und die entsprechenden Theoreme innerhalb eines Syntheseprozesses verwenden kann.*

3.2–a Die Klasse aller Spezifikationen ist die Klasse aller 4-Tupel *spec* = (D,R,I,O), wobei D und R Datentypen (also Elemente von TYPES ≡ U_1, I ein Prädikat über D und O ein Prädikat über D×R ist.

Diese Klasse läßt sich als ein Datentyp SPECIFICATIONS beschreiben, der allerdings von einer höheren Ordnung ist (d.h. zu U_2 gehört).

SPECIFICATIONS ≡ D:TYPES × R:TYPES × D→$\mathbb{B}$ × D×R→$\mathbb{B}$

3.2–b Die Klasse aller Programme ergu]ibt sich aus derjenigen der Spezifikationen durch Hinzunahme eines Programmkörpers body:D↛R. Um dies beschreiben zu können, geben wir Selektoren D(spec) und R(spec) für den Domain und Range einer Spezifikation an

D(*spec*) ≡ let (D,R,I,O)= *spec* in D
R(*spec*) ≡ let (D,R,I,O)= *spec* in R
PROGRAMS ≡ spec:SPEC × D(spec)↛R(spec)

3.2–c Die Formalisierung von Programmkorrektheit ergibt sich unmittelbar, da Terminierung durch das dom-Prädikat der rekursiven Funktionstypen beschrieben werden kann.

body hält auf x ≡ dom(*body*)(x)
p ist korrekt ≡ let ((D,R,I,O),body) = p in
∀x:D. I(x) ⇒ body hält auf x ∧ O(x,body(x))

3.2–d Erfüllbarkeit von Spezifikationen folgt ebenfalls unmittelbar

spec ist erfüllbar ≡ let((D,R,I,O),body) = p in
∃body:D↛R. (spec,body) ist korrekt

3.2–e Die syntaktisch aufbereitete Notation für Programme läßt sich relativ leicht auf die Tupelschreibweise abbilden

FUNCTION $f(x{:}D){:}R$ WHERE I_x RETURNS y SUCH THAT $O_{x,y}$ = $body_{f,x}$
≡
(D, R, $\lambda x.I_x$, $\lambda x,y.O_{x,y}$, letrec $f(x)$= $body_{f,x}$)

Dabei soll I_x einen beliebiger Ausdruck kennzeichnen, in dem x frei vorkommen darf.

Lösung 3.3 *Diese Aufgabe hat mehrere Ziele. Zum einen soll sie zeigen, daß Lösungen, die durch Nachdenken erzeugt wurden, sich tatsächlich mit Hilfe von Äquivalenztransformationen ableiten (und damit verifizieren) läßt. Zum anderen macht sie relativ schnell deutlich, daß die Anwendung von Äquivalenztransformationen zur Lösung eines Programmierproblems ohne eine Strategie nicht zum Ziele führt.*
In diesem Übungsblatt steht die Korrektheit einer Ableitung im Vordergrund und nicht die Methode, wie diese gefunden wird. Der Algorithmus soll als logische Formel beschrieben werden und es ist zu zeigen, mit welchen Äquivalenztransformationen man zu diesem Algorithmus kommen kann. Im Laufe dieses Prozesses lohnt es, sich Gedanken über Systematik zu machen.

Entsprechend der Denkweise der Synthese durch Transformationen führen wir als erstes ein neues Prädikat SORT ein, das durch die folgende Äquivalenz definiert ist:

$\forall$L,S:Seq($\mathbb{Z}$). true $\Rightarrow$ SORT(L,S) $\Leftrightarrow$ rearranges(L,S) $\wedge$ ordered(S)

Da Sortierverfahren notwendigerweise rekursiv sind, muß das Ziel der Transformationen eine rekursive Formel sein, in der auf der rechten Seite außer SORT nur Ausdrücke vorkommen, die sich leicht in Algorithmen umwandeln lassen. Wir wollen 4 Algorithmen ableiten.

Selection Sort:

```
sort(L) = if L=[]  then []  else let y=min(L) in  y.sort(L-y)
```

Insertion Sort:

```
sort(L) = if L=[]  then []  else let x=first(L) and L'=rest(L)
                                in  insert(x,sort(L'))
```

Quicksort:

sort(L) = if L=[] then [] else let x=first(L) and L'=rest(L)
in sort($L'_{<x}$) $\circ$ x.sort($L'_{\geq x}$)

Mergesort:

sort(L) = if L=[] then [] else let x=first(L) and L'=rest(L) and k=|L'|$\div$2
in insert(x, merge(sort($L'_{[1..k]}$),sort($L'_{[k+1..|L'|]}$)))

Diese Algorithmen entsprechen den folgenen logischen Formeln.

$\forall$L,S:Seq($\mathbb{Z}$). true $\Rightarrow$
SSORT(L,S) $\Leftrightarrow$ L=[] $\wedge$ S=[]
$\vee$ $\exists$a:$\mathbb{Z}$.$\exists$S':Seq($\mathbb{Z}$). a$\in$L $\wedge$ $\forall$x$\in$L.x$\geq$a $\wedge$ SSORT(L-a,S') $\wedge$ S=a.S'

$\forall$L,S:Seq($\mathbb{Z}$). true $\Rightarrow$
ISORT(L,S) $\Leftrightarrow$ L=[] $\wedge$ S=[]
$\vee$ $\exists$a:$\mathbb{Z}$.$\exists$L',S':Seq($\mathbb{Z}$). L=a.L' $\wedge$ ISORT(L',S') $\wedge$ S = $S'_{<a} \circ$ x.$S'_{\geq a}$

$\forall$L,S:Seq($\mathbb{Z}$). true $\Rightarrow$
QSORT(L,S) $\Leftrightarrow$ L=[] $\wedge$ S=[]
$\vee$ $\exists$a:$\mathbb{Z}$.$\exists$L',S_1,S_2:Seq($\mathbb{Z}$). L=a.L'
$\wedge$ QSORT($L'_{<a}$,S_1) $\wedge$ QSORT($L'_{\geq a}$,S_2) $\wedge$ S = $S_1 \circ$ a.S_2

$\forall$L,S:Seq($\mathbb{Z}$). true $\Rightarrow$
MSORT(L,S) $\Leftrightarrow$ L=[] $\wedge$ S=[]
$\vee$ $\exists$a,k:$\mathbb{Z}$.$\exists$L',S',S_1,S_2:Seq($\mathbb{Z}$). L=a.L' $\wedge$ k=|L'|$\div$2
$\wedge$ MSORT($L'_{[1..k]}$,S_1) $\wedge$ MSORT($L'_{[k+1..|L'|]}$,S_2)
$\wedge$ S'=merge(S_1,S_2) $\wedge$ S = $S'_{<a} \circ$ a.$S'_{\geq a}$

Dabei sei definiert

insert(x,S) $\equiv$ $S_{<x} \circ x . S_{\geq x}$
merge(S_1,S_2) $\equiv$ if S_1=[] then S_2 else if S_2=[] then S_1
else let x.S_1'=S_1 and y.S_2'=S_2 in
if x<y then x.merge(S_1',S_2) else y.merge(S_1,S_2')

(Genau besehen wird merge *als rekursive Funktion* letrec merge(S_1,S_2)... *definiert)*

Wie man sieht haben alle Formeln eine first/rest-Zerlegung gemeinsam, die entweder auf der Ausgabevariablen S oder auf der Eingabe L stattfindet. Diese Zerlegung wird in allen Fällen auf die gleiche Art begonnen werden. Die letzten beiden Formeln nehmen zusätzlich einen Split der Eingaben vor, bevor sortiert wird. Es ist daher zu erwarten, daß ihre Ableitung sich zu einem Teil aus der des Insertion-Sort Algorithmus ergibt, zu dem nun Argumente über Splitting und zusammenmischen hinzukommen werden. Alle Ableitungen werden Lemmata über ordered benötigen, die wir zum Schluß ergänzen müssen.

Wir beginnen mit der Herleitung der first/rest-Zerlegung und der Lösung für den Basisfall. Der Einfachheit unterdrücken wir ab sofort die Zeile $\forall\texttt{L,S:Seq}(\mathbb{Z}).\ \texttt{true}\ \Rightarrow$

$$\forall\texttt{L,S:Seq}(\mathbb{Z}).\ \ \texttt{true}\ \Rightarrow\ \ \texttt{SORT(L,S)}\ \Leftrightarrow\ \texttt{rearranges(L,S)}\ \wedge\ \texttt{ordered(S)}$$

$$\boxed{\textit{B.2.3.1 mit } \mathbb{Z} \textit{ und } \texttt{L} + \textit{Zerlegung}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{rearranges(L,S)}\ \wedge\ \texttt{ordered(S)}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{L':Seq}(\mathbb{Z})\texttt{.L=a.L}\ \wedge\ \texttt{rearranges(L,S)}\ \wedge\ \texttt{ordered(S)}\end{aligned}$$

$$\boxed{\textit{Substitution}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{rearranges([],S)}\ \wedge\ \texttt{ordered(S)}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{L':Seq}(\mathbb{Z})\texttt{.L=a.L'}\ \wedge\ \texttt{rearranges(a.L',S)}\ \wedge\ \texttt{ordered(S)}\end{aligned}$$

$$\boxed{\textit{B.2.26.1/2}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\ \wedge\ \texttt{ordered(S)}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{L':Seq}(\mathbb{Z})\texttt{.L=a.L'}\ \wedge\ \texttt{a}\in\texttt{S}\ \wedge\ \texttt{rearranges(L',S-a)}\ \wedge\ \texttt{ordered(S)}\end{aligned}$$

$$\boxed{\textit{Substitution}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\ \wedge\ \texttt{ordered([])}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{L':Seq}(\mathbb{Z})\texttt{.L=a.L'}\ \wedge\ \texttt{a}\in\texttt{S}\ \wedge\ \texttt{rearranges(L',S-a)}\ \wedge\ \texttt{ordered(S)}\end{aligned}$$

$$\boxed{\textit{ordered.1}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{L':Seq}(\mathbb{Z})\texttt{.L=a.L'}\ \wedge\ \texttt{a}\in\texttt{S}\ \wedge\ \texttt{rearranges(L',S-a)}\ \wedge\ \texttt{ordered(S)}\ \boxed{\mathbf{1}}\end{aligned}$$

Selection Sort

Da `rearranges` kommutativ ist, gilt $\boxed{1}$ auch für `S` anstelle von `L` und umgekehrt.

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{S':Seq}(\mathbb{Z})\texttt{.S=a.S'}\ \wedge\ \texttt{a}\in\texttt{L}\ \wedge\ \texttt{rearranges(L-a,S')}\ \wedge\ \texttt{ordered(S)}\end{aligned}$$

$$\boxed{\textit{ordered.2}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{S':Seq}(\mathbb{Z})\texttt{.S=a.S'}\ \wedge\ \texttt{a}\in\texttt{L}\ \wedge\ \texttt{rearranges(L-a,S')}\\ & \quad\wedge\ \texttt{ordered(S')}\ \wedge\ \forall\texttt{x}\in\texttt{S'.a}\leq\texttt{x}\end{aligned}$$

$$\boxed{\textit{B.2.26.12/3}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{S':Seq}(\mathbb{Z})\texttt{.S=a.S'}\ \wedge\ \texttt{a}\in\texttt{L}\ \wedge\ \texttt{rearranges(L-a,S')}\\ & \quad\wedge\ \texttt{ordered(S')}\ \wedge\ \forall\texttt{x}\in\texttt{L-a.a}\leq\texttt{x}\end{aligned}$$

$$\boxed{\textit{Definition von } \texttt{SORT}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{S':Seq}(\mathbb{Z})\texttt{.S=a.S'}\ \wedge\ \texttt{a}\in\texttt{L}\ \wedge\ \texttt{SORT(L-a,S')}\ \wedge\ \forall\texttt{x}\in\texttt{L-a.a}\leq\texttt{x}\end{aligned}$$

$$\boxed{\textit{Arithmetik-Ergänzung}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{S':Seq}(\mathbb{Z})\texttt{.S=a.S'}\ \wedge\ \texttt{a}\in\texttt{L}\ \wedge\ \texttt{SORT(L-a,S')}\ \wedge\ \forall\texttt{x}\in\texttt{L-a.a}\leq\texttt{x}\ \wedge\ \texttt{a}\leq\texttt{a}\end{aligned}$$

$$\boxed{\textit{B.1.11.6 (analog)}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{S':Seq}(\mathbb{Z})\texttt{.S=a.S'}\ \wedge\ \texttt{a}\in\texttt{L}\ \wedge\ \texttt{SORT(L-a,S')}\ \wedge\ \forall\texttt{x}\in\texttt{L.a}\leq\texttt{x}\end{aligned}$$

$$\boxed{\textit{Umsortieren, der Lesbarkeit wegen}}$$

$$\begin{aligned}\texttt{SORT(L,S)}\ \Leftrightarrow\ & \texttt{L=[]}\ \wedge\ \texttt{S=[]}\\ & \vee\ \exists\texttt{a:}\mathbb{Z}.\exists\texttt{S':Seq}(\mathbb{Z})\texttt{.a}\in\texttt{L}\ \wedge\ \forall\texttt{x}\in\texttt{L.a}\leq\texttt{x}\ \wedge\ \texttt{SORT(L-a,S')}\ \wedge\ \texttt{S=a.S'}\end{aligned}$$

Insertion Sort

Wir beginnen wieder bei $\boxed{1}$

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{a}\in\texttt{S} \;\wedge\; \texttt{rearranges(L',S-a)} \;\wedge\; \texttt{ordered(S)}
\end{aligned}
$$

ordered 5

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{a}\in\texttt{S} \;\wedge\; \texttt{rearranges(L',S-a)} \\
& \;\;\wedge\; \texttt{ordered(S-a)} \;\wedge\; \texttt{S} = \texttt{(S-a)}_{<a} \circ \texttt{a.(S-a)}_{\geq a}
\end{aligned}
\qquad \boxed{\mathbf{2}}
$$

Quantifizierung von `S-a`

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L',S'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{a}\in\texttt{S} \;\wedge\; \texttt{rearranges(L',S')} \\
& \;\;\wedge\; \texttt{ordered(S')} \;\wedge\; \texttt{S} = \texttt{S'}_{<a} \circ \texttt{a.(S')}_{\geq a} \;\wedge\; \texttt{S'=S-a}
\end{aligned}
$$

S'=S-a *ist überflüssig nach 6.,* a ∈ S *ist überflüssig nach B.2.9.11/2*

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L',S'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{rearranges(L',S')} \\
& \;\;\wedge\; \texttt{ordered(S')} \;\wedge\; \texttt{S} = \texttt{S'}_{<a} \circ \texttt{a.S'}_{\geq a}
\end{aligned}
\qquad \boxed{\mathbf{3}}
$$

Definition von `SORT`

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L',S'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{SORT(L',S')} \;\wedge\; \texttt{S} = \texttt{S'}_{<a} \circ \texttt{a.S'}_{\geq a}
\end{aligned}
$$

Quicksort

Die Ableitung ist bis zum Punkt $\boxed{2}$ identisch mit Insertion Sort, muß dann allerdings noch ein zusätzliches Splitting der Ausgabe durchführen, die genau am Punkt `a` gespalten wird, der bereits für das Einfügen vorgesehen ist.

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{a}\in\texttt{S} \;\wedge\; \texttt{rearranges(L',S-a)} \\
& \;\;\wedge\; \texttt{ordered(S-a)} \;\wedge\; \texttt{S} = \texttt{(S-a)}_{<a} \circ \texttt{a.(S-a)}_{\geq a}
\end{aligned}
$$

B.2.26.12, B.2.26.16 mit `p(x)≡x<a` `p(x)≡x≥a`

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{a}\in\texttt{S} \;\wedge\; \texttt{rearranges(L',S-a)} \\
& \;\;\wedge\; \texttt{rearranges(L'}_{<a}\texttt{,(S-a)}_{<a}\texttt{)} \;\wedge\; \texttt{rearranges(L'}_{\geq a}\texttt{,(S-a)}_{\geq a}\texttt{)} \\
& \;\;\;\wedge\; \texttt{ordered(S-a)} \;\wedge\; \texttt{S} = \texttt{(S-a)}_{<a} \circ \texttt{a.(S-a)}_{\geq a}
\end{aligned}
$$

ordered 8

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{a}\in\texttt{S} \;\wedge\; \texttt{rearranges(L',(S-a)}_{<a}\circ\texttt{(S-a)}_{\geq a}\texttt{)} \\
& \;\;\wedge\; \texttt{rearranges(L'}_{<a}\texttt{,(S-a)}_{<a}\texttt{)} \;\wedge\; \texttt{rearranges(L'}_{\geq a}\texttt{,(S-a)}_{\geq a}\texttt{)} \\
& \;\;\;\wedge\; \texttt{ordered(S-a)} \;\wedge\; \texttt{S} = \texttt{(S-a)}_{<a} \circ \texttt{a.(S-a)}_{\geq a}
\end{aligned}
$$

ordered 7

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L'}:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{a}\in\texttt{S} \;\wedge\; \texttt{rearranges(L',(S-a)}_{<a}\circ\texttt{(S-a)}_{\geq a}\texttt{)} \\
& \;\;\wedge\; \texttt{rearranges(L'}_{<a}\texttt{,(S-a)}_{<a}\texttt{)} \;\wedge\; \texttt{rearranges(L'}_{\geq a}\texttt{,(S-a)}_{\geq a}\texttt{)} \\
& \;\;\;\wedge\; \texttt{ordered((S-a)}_{<a}\texttt{)} \;\wedge\; \texttt{ordered((S-a)}_{\geq a}\texttt{)} \;\wedge\; \texttt{S} = \texttt{(S-a)}_{<a} \circ \texttt{a.(S-a)}_{\geq a}
\end{aligned}
$$

Quantifizierung von $\texttt{(S-a)}_{<a}$, $\texttt{(S-a)}_{\geq a}$

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L',,S}_1\texttt{,S}_2:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \;\wedge\; \texttt{a}\in\texttt{S} \;\wedge\; \texttt{rearranges(L',S}_1\circ\texttt{S}_2\texttt{)} \\
& \;\;\wedge\; \texttt{rearranges(L'}_{<a}\texttt{,S}_1\texttt{)} \;\wedge\; \texttt{rearranges(L'}_{\geq a}\texttt{,S}_2\texttt{)} \\
& \;\;\;\wedge\; \texttt{ordered(S}_1\texttt{)} \;\wedge\; \texttt{ordered(S}_2\texttt{)} \;\wedge\; \texttt{S} = \texttt{S}_1 \circ \texttt{a.S}_2 \;\wedge\; \texttt{S}_1\texttt{=(S-a)}_{<a} \;\wedge\; \texttt{S}_2\texttt{=(S-a)}_{\geq a}
\end{aligned}
$$

rearranges(L',$S_1 \circ S_2$) *ist überflüssig nach B.2.26.7/14/13*

$S_1=(S\text{-}a)_{<a} \wedge S_2=(S\text{-}a)_{\geq a}$ *ist überflüssig nach B.2.26.3/B.2.12.*

a ∈ S *ist überflüssig nach B.2.9.11/2*

$$
\begin{aligned}
\texttt{SORT(L,S)} \;\Leftrightarrow\;\; & \texttt{L=[]} \;\wedge\; \texttt{S=[]} \\
& \vee\; \exists \texttt{a}:\mathbb{Z}.\exists \texttt{L',S}_1\texttt{,S}_2:\texttt{Seq}(\mathbb{Z}).\,\texttt{L=a.L'} \\
& \;\;\wedge\; \texttt{rearranges(L'}_{<a}\texttt{,S}_1\texttt{)} \;\wedge\; \texttt{rearranges(L'}_{\geq a}\texttt{,S}_2\texttt{)}
\end{aligned}
$$

$$\land\ \mathtt{ordered}(S_1)\ \land\ \mathtt{ordered}(S_2)\ \land\ S = S_1 \circ a.S_2$$

Definition von SORT

$$\begin{aligned}\mathtt{SORT}(L,S)\ \Leftrightarrow\ & L=[\,]\ \land\ S=[\,]\\ &\lor\ \exists a{:}\mathbb{Z}.\exists L',S_1,S_2{:}\mathtt{Seq}(\mathbb{Z}).\ L=a.L'\\ &\land\ \mathtt{SORT}(L'_{<a},S_1)\ \land\ \mathtt{SORT}(L'_{\geq a},S_2)\ \land\ S = S_1 \circ a.S_2\end{aligned}$$

Mergesort

Die Ableitung ist bis zum Punkt $\boxed{3}$ identisch mit Insertion Sort, muß dann allerdings noch ein zusätzliches Splitting der (restlichen) Eingabe durchführen.

$$\begin{aligned}\mathtt{SORT}(L,S)\ \Leftrightarrow\ & L=[\,]\ \land\ S=[\,]\\ &\lor\ \exists a{:}\mathbb{Z}.\exists L',S'{:}\mathtt{Seq}(\mathbb{Z}).\ L=a.L'\ \land\ \mathtt{rearranges}(L',S')\\ &\land\ \mathtt{ordered}(S')\ \land\ S = S'_{<a} \circ a.(S')_{\geq a}\end{aligned}$$

Splitting.1 mit k=|L'|÷2

$$\begin{aligned}\mathtt{SORT}(L,S)\ \Leftrightarrow\ & L=[\,]\ \land\ S=[\,]\\ &\lor\ \exists a,k{:}\mathbb{Z}.\exists L',S'{:}\mathtt{Seq}(\mathbb{Z}).\ L=a.L'\ \land\ k=|L'|\div 2\\ &\land\ \mathtt{rearranges}(L'_{[1..k]}\circ L'_{[k+1..|L'|]},S')\ \land\ \mathtt{ordered}(S')\ \land\ S = S'_{<a} \circ a.(S')_{\geq a}\end{aligned}$$

merge.3

$$\begin{aligned}\mathtt{SORT}(L,S)\ \Leftrightarrow\ & L=[\,]\ \land\ S=[\,]\\ &\lor\ \exists a,k{:}\mathbb{Z}.\exists L',S',S_1,S_2{:}\mathtt{Seq}(\mathbb{Z}).\ L=a.L'\ \land\ k=|L'|\div 2\\ &\land\ \mathtt{rearranges}(L'_{[1..k]},S_1)\ \land\ \mathtt{rearranges}(L'_{[k+1..|L'|]},S_2)\\ &\land\ \mathtt{ordered}(S_1)\ \land\ \mathtt{ordered}(S_2)\ \land\ S'=\mathtt{merge}(S_1,S_2)\ \land\ S = S'_{<a} \circ a.(S')_{\geq a}\end{aligned}$$

Definition von SORT

$$\begin{aligned}\mathtt{SORT}(L,S)\ \Leftrightarrow\ & L=[\,]\ \land\ S=[\,]\\ &\lor\ \exists a,k{:}\mathbb{Z}.\exists L',S',S_1,S_2{:}\mathtt{Seq}(\mathbb{Z}).\ L=a.L'\ \land\ k=|L'|\div 2\\ &\land\ \mathtt{SORT}(L'_{[1..k]},S_1)\ \land\ \mathtt{SORT}(L'_{[k+1..|L'|]},S_2)\ \land\ S'=\mathtt{merge}(S_1,S_2)\\ &\land\ S = S'_{<a} \circ a.(S')_{\geq a}\end{aligned}$$

Lemmata

Die folgenden Lemmata sind im Verlaufe der Ableitungen erforderlich geworden.

ordered

1. $\mathtt{ordered}([\,])$
2. $\mathtt{ordered}(a.S')\ \Leftrightarrow\ \mathtt{ordered}(S')\ \land\ \forall x \in S'.\, a \leq x$
3. $\mathtt{ordered}(S)\ \Rightarrow\ \forall i,k \in \mathtt{domain}(S).\ i \leq k\ \Rightarrow\ \mathtt{ordered}(S_{[i..k]})$
4. $a \in S\ \Rightarrow\ \mathtt{ordered}(S)\ \Rightarrow\ \exists k \in \mathtt{domain}(S).\, S_{<a}=S_{[1..k]}\ \land\ S_{\geq a}=S_{[k+1..|S|]}$
5. $a \in S\ \Rightarrow\ \mathtt{ordered}(S)\ \Leftrightarrow\ \mathtt{ordered}(S-a)\ \land\ S = (S-a)_{<a} \circ a.(S-a)_{\geq a}$ *abgeleitet aus 4 und B.2.25.4*
6. $a \in S\ \land\ \mathtt{ordered}(S)\ \Rightarrow\ S'=S-a\ \Leftrightarrow\ S = S'_{<a} \circ a.S'_{\geq a}$ *abgeleitet aus 5 und B.2.25.4*
7. $a \in S\ \Rightarrow\ \mathtt{ordered}(S)\ \Leftrightarrow\ \mathtt{ordered}(S_{<a})\ \land\ \mathtt{ordered}(S_{\geq a})$
8. $a \in S\ \Rightarrow\ \mathtt{ordered}(S)\ \Rightarrow\ S = S_{<a} \circ S_{\geq a}$

splitting

1. $\forall k \in \mathtt{domain}(L).\ L = L_{[1..k]} \circ L_{[k+1..|L|]}$

merge

1. $\texttt{rearranges(merge(S}_1\texttt{,S}_2\texttt{), S}_1\circ\texttt{S}_2\texttt{)}$
2. $\texttt{ordered(S}_1\texttt{)} \wedge \texttt{ordered(S}_2\texttt{)} \Rightarrow \texttt{ordered(merge(S}_1\texttt{,S}_2\texttt{))}$
3. $\texttt{ordered(S)} \wedge \texttt{rearranges(L}_1\circ\texttt{L}_2\texttt{,S)} \Leftrightarrow \exists\texttt{S}_1\texttt{,S}_2\texttt{:Seq(}\mathbb{Z}\texttt{). ordered(S}_1\texttt{)} \wedge \texttt{ordered(S}_2\texttt{)}$
   $\wedge \texttt{rearranges(L}_1\texttt{,S}_1\texttt{)} \wedge \texttt{rearranges(L}_2\texttt{,S}_2\texttt{)} \wedge \texttt{S=merge(S}_1\texttt{,S}_2\texttt{)}$

S_1/S_2 *entsteht durch Filterung. Aufwendiger Beweis*

B.2.25 (neu!!)

3. $\texttt{a}\in\texttt{L} \Leftrightarrow \exists\texttt{k}\in\texttt{domain(L).L=L}_{\texttt{[1..k-1]}}\circ\texttt{a.L}_{\texttt{[k+1..|L|]}}$
4. $\texttt{a}\in\texttt{L} \Rightarrow \texttt{L'=L-a} \Leftrightarrow \exists\texttt{k}\in\texttt{domain(L').L=L'}_{\texttt{[1..k]}}\circ\texttt{a.L'}_{\texttt{[k+1..|L'|]}}$
5. $\texttt{a}\in\texttt{S} \Rightarrow \texttt{S}_{<a}\texttt{=(S-a)}_{<a} \wedge \texttt{S}_{\geq a}\texttt{=a.(S-a)}_{\geq a}$

Im Appendix B war ein TeX-Fehler, was $\texttt{L}_{\texttt{[k..j]}}$ betrifft. Die Indizes sind nicht zu sehen.

Lösung 3.4 *Ziel dieser Aufgabe ist es, den Zusammenhang zwischen logischen Formeln und den durch sie beschriebenen Algorithmen zu verstehen. Es geht dabei besonders um die Analyse der syntaktischen Struktur der Ausgabebedingungen.*

3.4–a $\texttt{FUNCTION } F_1\texttt{(x:D):R WHERE I(x)}\vee\texttt{I'(x) RETURNS y SUCH THAT O(x,y)}\vee\texttt{O'(x,y)}$
$= \texttt{if I(x) then body}_f\texttt{(x) else body}_{f'}\texttt{(x)}$

Eine Disjunktion in der Ausgabebedingung entspricht somit einer Fallunterscheidung im Algorithmus, wobei als Unterscheidungsbedingung die Eingabebedingung `I(x)` zu wählen ist, welche die erste Ausgabebedingung `O(x,y)` erfüllbar macht. Da hier nur die Korrektheit überprüft werden soll, können wir davon ausgehen, daß diese Informationen bereits gefunden sind.

Zu zeigen ist: $\forall\texttt{x:D. I(x)}\vee\texttt{I'(x)} \Rightarrow \texttt{O(x,y)}\vee\texttt{O'(x,y)}$,
wobei $\texttt{y = if I(x) then body}_f\texttt{(x) else body}_{f'}\texttt{(x)}$ ist.

Sei $\texttt{x}\in\texttt{D}$. Wir unterscheiden die Fälle `I(x)` und $\neg$`I(x)` und müssen – nach Auswertung von `y` – zwei Fälle untersuchen.

1. $\texttt{I(x)}\wedge\texttt{(I(x)}\vee\texttt{I'(x))} \Rightarrow \texttt{O(x,body}_f\texttt{(x))}\vee\texttt{O'(x,body}_f\texttt{(x))}$
   Nach Reduktion der Vorbedingung erhalten wir
   $\texttt{I(x)} \Rightarrow \texttt{O(x,body}_f\texttt{(x))}\vee\texttt{O'(x,body}_f\texttt{(x))}$
   Dies folgt aus der Korrektheitsannahme für f: $\texttt{I(x)} \Rightarrow \texttt{O(x,body}_f\texttt{(x))}$
2. $\neg\texttt{I(x)}\wedge\texttt{(I(x)}\vee\texttt{I'(x))} \Rightarrow \texttt{O(x,body}_{f'}\texttt{(x))}\vee\texttt{O'(x,body}_{f'}\texttt{(x))}$
   Nach Reduktion der Vorbedingung erhalten wir
   $\neg\texttt{I(x)}\wedge\texttt{I'(x)} \Rightarrow \texttt{O(x,body}_{f'}\texttt{(x))}\vee\texttt{O'(x,body}_{f'}\texttt{(x))}$
   Dies folgt aus der Korrektheitsannahme für f': $\texttt{I'(x)} \Rightarrow \texttt{O'(x,body}_{f'}\texttt{(x))}$

P.S.: der Beweis könnte auch formal geführt werden.

3.4–b $\texttt{FUNCTION } F_2\texttt{(x:D):R WHERE I(x)}\wedge\texttt{P(x,body}_f\texttt{(x)) RETURNS y SUCH THAT O(x,y)}\wedge\texttt{P(x,y)}$
$= \texttt{body}_f\texttt{(x)}$

Eine Konjunktion in der Ausgabebedingung entspricht somit einer Einschränkung legaler Eingaben unter Beibehaltung der Berechnungsmethode. Das Ziel ist nun, solche Berechnungsmethoden zu wählen, daß die Einschrankung von allen Werten erfüllt wird.

Der Beweis für $\forall\texttt{x:D. I(x)}\wedge\texttt{P(x,body}_f\texttt{(x))} \Rightarrow \texttt{O(x,body}_f\texttt{(x))}\wedge\texttt{P(x,body}_f\texttt{(x))}$ folgt unmittelbar aus der Korrektheitsannahme für f.

3.4–c $\texttt{FUNCTION } F_3\texttt{(x:D):R' WHERE I(x)} \wedge \texttt{J(x,body}_f\texttt{(x))}$
$\texttt{RETURNS z SUCH THAT } \exists\texttt{y:R. O(x,y)} \wedge \texttt{O'(x,y,z)}$
$= \texttt{body}_g\texttt{(x,body}_f\texttt{(x))}$

Ein Existenzquantor in der Ausgabebedingung entspricht einer Generalisierung des Problems: zuerst wird ein Zwischenwert berechnet und dann allgemein gezeigt, wie man aus Zwischenwerten eine Lösung baut.

Zu zeigen: $\forall\texttt{x:D. I(x)} \wedge \texttt{J(x,body}_f\texttt{(x))} \Rightarrow \exists\texttt{y:R. O(x,y)} \wedge \texttt{O'(x,y,body}_g\texttt{(x,body}_f\texttt{(x)))}$

Es sei $\texttt{x} \in \texttt{D}$ und es gelte $\texttt{I(x)} \wedge \texttt{J(x,body}_f\texttt{(x))}$. Aus der Korrektheitsannahme für f – $\texttt{I(x)} \Rightarrow \texttt{O(x,body}_f\texttt{(x))}$ – folgt $\texttt{O(x,body}_f\texttt{(x))}$ und wir wählen $\texttt{y:=body}_f\texttt{(x)}$.

Zu zeigen bleibt $\texttt{O'(x,body}_f\texttt{(x),body}_g\texttt{(x,body}_f\texttt{(x)))}$, was direkt aus der Korrektheitsannahme für g – $\texttt{J(x,body}_f\texttt{(x))} \Rightarrow \texttt{O'(x,body}_f\texttt{(x),body}_g\texttt{(x,body}_f\texttt{(x)))}$ – folgt.