

1 Strukturelle Induktion und Rekursion

Definition 1 (Rekursive Konstruktoren, Rekursiv definierte Datentypen)

Für einen Datentyp $\text{struc}[@T_1, \dots, @T_n]$ definiert durch

$$\begin{aligned} \text{structure struc}[@T_1, \dots, @T_n] <= \\ \text{cons}_1(\text{sel}_{1,1} : \text{struc}_{1,1}, \dots, \text{sel}_{1,n_1} : \text{struc}_{1,n_1}), \\ \vdots \\ \text{cons}_k(\text{sel}_{k,1} : \text{struc}_{k,1}, \dots, \text{sel}_{k,n_k} : \text{struc}_{k,n_k}) \end{aligned} \quad (1)$$

und einen Konstruktor cons_i von $\text{struc}[@T_1, \dots, @T_n]$ mit

$$\text{cons}_i(\text{sel}_{i,1} : \text{struc}_{i,1}, \dots, \text{sel}_{i,n_i} : \text{struc}_{i,n_i})$$

ist

- $h \in \{1, \dots, n_i\}$ eine Rekursionsposition von cons_i gdw. $\text{struc}_{i,h} = \text{struc}[@T_1, \dots, @T_n]$,
- cons_i rekursiv gdw. cons_i mindestens eine Rekursionsposition besitzt,
- $\text{struc}[@T_1, \dots, @T_n]$ ein rekursiv definierter Datentyp gdw. $\text{struc}[@T_1, \dots, @T_n]$ mindestens einen rekursiven Konstruktor besitzt. ■

Formale Grundlagen der Informatik 3 –

7. Strukturelle Induktion und Induktion nach Rekursion von Prozeduren

Christoph Walther
TU Darmstadt

Beispiel 1 (Rekursive Konstruktoren und Datentypen)

- `bool` – keine rekursiven Konstruktoren, also nicht rekursiv definiert.
- `nat` – `succ` ist rekursiv, also ist `nat` rekursiv definiert.
- `list[@T]` – `::` ist rekursiv, also ist `list[@T]` rekursiv definiert.
- `structure sexpr[@T] <=`
`nil, atom(data:@T), cons(car:sexpr[@T], cdr:sexpr[@T])`
 – `cons` ist rekursiv, also ist `sexpr[@T]` rekursiv definiert.
- `structure tree[@T] <=`
`leaf, node(left:tree[@T], key:@T, right:tree[@T])`
 – `node` ist rekursiv, also ist `tree[@T]` rekursiv definiert.

Definition 2 (Relationenbeschreibung eines Datentyps)

Für einen Datentyp (gegeben wie in Definition 1) wird

- jedem nicht-rekursiven Konstruktor cons die atomare Relationenbeschreibung

$$A_{\text{cons}} = \langle \{? \text{cons}(u)\}, \emptyset \rangle$$

und

- jedem rekursiven Konstruktor cons mit den Selektoren $\text{sel}_1, \dots, \text{sel}_h$ an den Rekursionspositionen von cons die atomare Relationenbeschreibung

$$A_{\text{cons}} = \langle \{? \text{cons}(u)\}, \{ \{u/\text{sel}_1(u)\}, \dots, \{u/\text{sel}_h(u)\} \} \rangle$$

zugeordnet.

- Dem Datentyp $\text{struc}[@T_1, \dots, @T_n]$ wird die Relationenbeschreibung

$$R_{\text{struc}[@T_1, \dots, @T_n]} := \{ A_{\text{cons}} \mid \text{cons ist Konstruktor von } \text{struc}[@T_1, \dots, @T_n] \}$$

zugeordnet. ■

Beispiel 2 (Relationenbeschreibung eines Datentyps)

(1) Man erhält für `structure nat <= 0, succ(pred : nat)`

$$R_{\text{nat}} := \left\{ \left\langle \{ \{ ?0(u) \}, \emptyset \}, \{ \{ ?\text{succ}(u) \}, \{ \{ u/\text{pred}(u) \} \} \right\rangle \right\}.$$

(2) Man erhält für

`structure list[@T] <= 0, [infixr, 100] :: (hd : @T, tl : list[@T])`

$$R_{\text{list}[@T]} := \left\{ \left\langle \{ \{ ?\emptyset(u) \}, \emptyset \}, \{ \{ ? :: (u) \}, \{ \{ u/\text{tl}(u) \} \} \right\rangle \right\}.$$

(3) Man erhält für `structure sexpr[@T] <=`

`atom(data : @T), nil, cons(car : sexpr[@T], cdr : sexpr[@T])`

$$R_{\text{sexpr}[@T]} := \left\{ \left\langle \{ \{ ?\text{atom}(u) \}, \emptyset \}, \{ \{ ?\text{nil}(u) \}, \emptyset \}, \{ \{ ?\text{cons}(u) \}, \{ \{ u/\text{car}(u) \}, \{ u/\text{cdr}(u) \} \} \right\rangle \right\}.$$

(4) Man erhält für `structure tree[@T] <=`

`leaf, node(left : tree[@T], key : @T, right : tree[@T])`

$$R_{\text{tree}[@T]} := \left\{ \left\langle \{ \{ ?\text{leaf}(u) \}, \emptyset \}, \{ \{ ?\text{node}(u) \}, \{ \{ u/\text{left}(u) \}, \{ u/\text{right}(u) \} \} \right\rangle \right\}.$$

Die Relationenbeschreibung eines monomorphen Datentyps ist die syntaktische Repräsentation der strukturellen Ordnung des Datentyps:

Definition 3 (Strukturelle Ordnung eines monomorphen Datentyps)

Sei

- `struc[@T1, ..., @Tn]` ein Datentyp (gegeben wie in Definition 1),
- $R_{\text{struc}[@T_1, \dots, @T_n]}$ die Relationenbeschreibung von `struc[@T1, ..., @Tn]`,
- `struc'` eine monomorphe Instanz von `struc[@T1, ..., @Tn]`, und
- $R_{\text{struc}'}$ die entsprechende monomorphe Instanz von $R_{\text{struc}[@T_1, \dots, @T_n]}$.

Dann heißt $>_{R_{\text{struc}'}, u}$ die strukturelle Ordnung des monomorphen Datentyps `struc'`.^{1,2} ■

¹ Zur Erinnerung: $>_{R_{\text{struc}'}, u}$ ist die durch $R_{\text{struc}'}$ definierte Relation (s. Definition 7 in Kapitel 6).

² Die strukturelle Ordnung eines Datentyps ist keine Ordnung im formalen Sinne, denn $>_{R_{\text{struc}'}, u}$ ist nicht transitiv.

Beispiel 3 (Strukturelle Ordnung)

(1) Für den Datentyp `nat` gilt `succ(n) >nat n`, also z.B. `succ(succ(0)) >nat succ(0) >nat 0` $\not>_{\text{nat}}$ Es gilt jedoch *nicht* `succ(succ(0)) >nat 0`, denn $>_{\text{nat}}$ ist *nicht transitiv*.

(2) Für den Datentyp `list[nat]` gilt `n :: k >list[nat] k`, also z.B.

`n1 :: n2 :: 0 >list[nat] n2 :: 0 >list[nat] 0` $\not>_{\text{list[nat]}}$ Es gilt jedoch *nicht* `n1 :: n2 :: 0 >list[nat] 0`, denn $>_{\text{list[nat]}}$ ist *nicht transitiv*.

(3) Für den Datentyp `bool` gilt $>_{\text{bool}} = \emptyset$, d.h. Daten vom Typ `bool` sind mittels $>_{\text{bool}}$ unvergleichbar. Grund: `bool` besitzt nur Konstruktorkonstanten.

Satz 4

Für jeden Datentyp `struc[@T1, ..., @Tn]` ist $R_{\text{struc}[@T_1, \dots, @T_n]}$ eine fundierte Relationenbeschreibung.

Beweis: Sei $R_{\text{struc}'}$ eine beliebige monomorphe Instanz von $R_{\text{struc}[@T_1, \dots, @T_n]}$. Mit $u >_{R_{\text{struc}'}} v$ gilt dann “ v ist echter Teilterm von u ”. Wäre $>_{R_{\text{struc}'}}$ nicht fundiert, so gäbe es Terme mit unendlich vielen echten Teiltermen. Da dies nicht möglich ist, ist $>_{R_{\text{struc}'}}$ fundiert. Folglich ist jede monomorphe Instanz $R_{\text{struc}'}$ von $R_{\text{struc}[@T_1, \dots, @T_n]}$ fundiert, und damit auch $R_{\text{struc}[@T_1, \dots, @T_n]}$.³ ■

³ Siehe Definition 8 in Kapitel 6.

Sprechweisen:

- Wir definieren eine Prozedur `p` durch *strukturelle Rekursion* über `struc[@T1, ..., @Tn]`, falls die optimierte (\Rightarrow Abschnitt 3) Relationenbeschreibung R_p von `p` (modulo Umbenennung der Variablen von R_p) identisch der Relationenbeschreibung $R_{\text{struc}[@T_1, \dots, @T_n]}$ ist.
- Wir führen den Beweis eines Lemmas durch *strukturelle Induktion* über `struc[@T1, ..., @Tn]`, falls die Induktionsformeln des Beweises aus der Relationenbeschreibung $R_{\text{struc}[@T_1, \dots, @T_n]}$ gewonnen wurden.⁴

⁴ Siehe Abschnitt 2.3 in Kapitel 6.

Beispiel 4 (Strukturelle Rekursion und Induktion)

In der Fallstudie “Sortieren durch Einfügen” wurden

- die Prozeduren
 - \leq durch strukturelle Rekursion über nat ,
 - `insert`, `isort` und `ordered` durch strukturelle Rekursion über `list[nat]`, sowie
 - `count` durch strukturelle Rekursion über `list[@ITEM]`
 definiert;
- die Lemmata
 - `<_is_total` durch strukturelle Induktion über nat ,
 - `insert_keeps_ordered` und `isort_sorts` durch strukturelle Induktion über `list[nat]`
 bewiesen.

Zusammenfassung:

- Aus der *Definition* von monomorphen Datentypen gewinnt man *uniform* (= “automatisch”) eine *fundierte Relation* (= strukturelle Ordnung, => Definitionen 2 und 3, Satz 4).
- Die *strukturelle Ordnung* eines monomorphen Datentyps `struc'` wird durch dessen *Relationenbeschreibung* $R_{\text{struc}'}$ syntaktisch *repräsentiert* (=> Definition 3).
- Durch *Rekursion* nach der strukturellen Ordnung (= *strukturelle Rekursion*) können wir *terminierende* Prozeduren definieren (=> **Kapitel 8**).
- Durch *Induktion* nach der strukturellen Ordnung (= *strukturelle Induktion*) können wir *Lemmata* über Datentypen und Prozeduren *beweisen* (=> Beispiel 4).
- Aus der *Relationenbeschreibung* eines Datentyps können *uniform* (= “automatisch”) *Induktionsformeln* zum Nachweis von Lemmata erzeugt werden (=> Beispiel 4, => Abschnitt 2.3 in **Kapitel 6**).

2 Induktion nach Rekursion von Prozeduren**Begriffe:**

- Ein Term der Form $if\{b, r_1, r_2\}$ wird *if-Term* genannt.
- Ein Term der Form $case\{b; d_1 : r_1, \dots, d_k : r_k\}$ wird *case-Term* genannt.
- Das erste Argument b eines *if*- oder *case*-Terms wird *Bedingungsterm* genannt.
- Jedes andere Argument r_i eines *if*- oder *case*-Terms wird *Ergebnisterm* genannt, falls r_i weder *if*- noch *case*-Terme enthält.
- Ein Term t heißt *normalisiert* gdw. für jeden Teilterm t' von t gilt:
 - falls t' ein *if*- oder *case*-Term ist, so enthält der Bedingungsterm von t' weder *if*- noch *case*-Terme (=> z.B. `if(if(...), ..., ...)` verboten), und
 - falls t' die Form $f(\dots, t'', \dots)$ mit $f \notin \{if, case\}$ hat, so enthält t'' weder *if*- noch *case*-Terme (=> z.B. `func(..., if(...), ...)` verboten).
- Für einen Teilterm t' eines normalisierten Terms t sei $cond(t', t)$ die Menge aller Literale, die in t “zu t' führen” (= “aufsammeln” von Bedingungstermen bzw. den Negaten davon).

Zur Erinnerung: Prozedurrümpfe sind *normalisierte* Terme !

Definition 5 (Relationenbeschreibung einer Prozedur)

Für eine rekursiv definierte Prozedur

$$\text{function proc}(x_1 : \text{struc}_1, \dots, x_k : \text{struc}_k) : \text{struc} \leq \text{body}_{\text{proc}}$$

wird

- jedem Ergebnisterm t in $\text{body}_{\text{proc}}$, der keine rekursiven Aufrufe enthält, die atomare (*nicht-rekursive*) Relationenbeschreibung

$$A_t = \langle \text{cond}(t, \text{body}_{\text{proc}}), \emptyset \rangle$$

zugeordnet und

- jedem Ergebnis- oder Bedingungsterm t in $\text{body}_{\text{proc}}$, der n rekursiven Aufrufe

$$\text{proc}(t_{1,1}, \dots, t_{k,1}), \dots, \text{proc}(t_{1,n}, \dots, t_{k,n})$$

enthält, die atomare (*rekursive*) Relationenbeschreibung

$$A_t = \langle \text{cond}(t, \text{body}_{\text{proc}}), \{\{x_1/t_{1,1}, \dots, x_k/t_{k,1}\}, \dots, \{x_1/t_{1,n}, \dots, x_k/t_{k,n}\}\} \rangle$$

zugeordnet.

- Der Prozedur `proc` wird die Relationenbeschreibung

$$R_{\text{proc}} := \{A_t \mid t \text{ ist Teilterm von } \text{body}_{\text{proc}}, \text{ so daß } A_t \text{ definiert ist}\}$$

zugeordnet. ■

Beispiel 5 (Relationenbeschreibungen für Prozeduren aus der InsertionSort-Fallstudie)

Es gilt

$$(1) R_{\leq} := \left\{ \begin{array}{l} \langle \{?0(n)\}, \emptyset \rangle, \\ \langle \{-?0(n), ?0(m)\}, \emptyset \rangle, \\ \langle \{-?0(n), \neg?0(m)\}, \{n/\text{pred}(n), m/\text{pred}(m)\} \rangle \end{array} \right\}.$$

$$(2) R_{\text{ordered}} := \left\{ \begin{array}{l} \langle \{?0(k)\}, \emptyset \rangle, \\ \langle \{-?0(k), ?0(\text{tl}(k))\}, \emptyset \rangle, \\ \langle \{-?0(k), \neg?0(\text{tl}(k)), \neg\text{hd}(k) \leq \text{hd}(\text{tl}(k))\}, \emptyset \rangle, \\ \langle \{-?0(k), \neg?0(\text{tl}(k)), \text{hd}(k) \leq \text{hd}(\text{tl}(k))\}, \{k/\text{tl}(k)\} \rangle \end{array} \right\}$$

$$(3) R_{\text{insert}} := \left\{ \begin{array}{l} \langle \{?0(k)\}, \emptyset \rangle, \\ \langle \{-?0(k), n \leq \text{hd}(k)\}, \emptyset \rangle, \\ \langle \{-?0(k), \neg n \leq \text{hd}(k)\}, \{n/n, k/\text{tl}(k)\} \rangle \end{array} \right\}.$$

$$(4) R_{\text{isort}} := \left\{ \begin{array}{l} \langle \{?0(k)\}, \emptyset \rangle, \\ \langle \{-?0(k)\}, \{k/\text{tl}(k)\} \rangle \end{array} \right\}.$$

$$(5) R_{\text{count}} := \left\{ \begin{array}{l} \langle \{?0(k)\}, \emptyset \rangle, \\ \langle \{-?0(k), i = \text{hd}(k)\}, \{i/i, k/\text{tl}(k)\} \rangle, \\ \langle \{-?0(k), \neg i = \text{hd}(k)\}, \{i/i, k/\text{tl}(k)\} \rangle \end{array} \right\}.$$

Die Relationenbeschreibung einer monomorphen Prozedur ist die syntaktische Repräsentation der Rekursionsordnung der Prozedur:

Definition 6 (Rekursionsordnung einer Prozedur)

Sei

- proc eine Prozedur (gegeben wie in Definition 5),
- R_{proc} die Relationenbeschreibung von proc ,
- proc' eine monomorphe Instanz von proc und
- $R_{\text{proc}'}$ die entsprechende monomorphe Instanz von R_{proc} .

Dann heißt $>_{R_{\text{proc}', x_1 \dots x_k}}$ die Rekursionsordnung der monomorphen Prozedur proc' .^{5,6} ■

Sprechweise:

- Wir führen den Beweis eines Lemmas durch *Induktion nach der Rekursionsordnung* einer Prozedur proc , falls die Induktionsformeln des Beweises aus der Relationenbeschreibung R_{proc} gewonnen wurden.⁷

⁵ Zur Erinnerung: $>_{R_{\text{proc}', x_1 \dots x_k}}$ ist die durch $R_{\text{proc}'}$ und $x_1 \dots x_k$ definierte Relation (s. Definition 7 in **Kapitel 6**).

⁶ Die Rekursionsordnung einer Prozedur ist keine Ordnung im formalen Sinne, denn $>_{R_{\text{proc}', x_1 \dots x_k}}$ ist nicht transitiv.

⁷ Siehe Abschnitt 2.3 in **Kapitel 6**.

Beispiel 6 (Rekursionsordnung einer Prozedur)

Für die Prozeduren aus der InsertionSort-Fallstudie (\Rightarrow **Kapitel 2**) gilt:

$$(1) (5, 3) >_{R_{\leq, nm}} (4, 2) >_{R_{\leq, nm}} (3, 1) >_{R_{\leq, nm}} (2, 0) \not>_{R_{\leq, nm}} \text{ sowie } (3, 5) >_{R_{\leq, nm}} (2, 4) >_{R_{\leq, nm}} (1, 3) >_{R_{\leq, nm}} (0, 2) \not>_{R_{\leq, nm}}.$$

$$(2) 1 :: 3 :: 4 :: 8 :: \emptyset >_{R_{\text{ordered}, k}} 3 :: 4 :: 8 :: \emptyset >_{R_{\text{ordered}, k}} 4 :: 8 :: \emptyset >_{R_{\text{ordered}, k}} 8 :: \emptyset \not>_{R_{\text{ordered}, k}} \text{ sowie } 1 :: 3 :: 2 :: 8 :: \emptyset >_{R_{\text{ordered}, k}} 3 :: 2 :: 8 :: \emptyset \not>_{R_{\text{ordered}, k}}.$$

$$(3) (5, 1 :: 3 :: 4 :: 8 :: 7 :: \emptyset) >_{R_{\text{insert}, nk}} (5, 3 :: 4 :: 8 :: 7 :: \emptyset) >_{R_{\text{insert}, nk}} (5, 4 :: 8 :: 7 :: \emptyset) >_{R_{\text{insert}, nk}} (5, 8 :: 7 :: \emptyset) \not>_{R_{\text{insert}, nk}}.$$

Bemerkung 1 (Rekursionsordnung einer Prozedur)

Für

- monomorphe Datentypen $\text{struc}'_1, \dots, \text{struc}'_k$,
- die Mengen $\mathcal{C}(P)_{\text{struc}'_1}, \dots, \mathcal{C}(P)_{\text{struc}'_k}$ der Konstruktorgrundterme dieser Datentypen und
- eine monomorphe Prozedur

$$\text{function } \text{proc}'(x_1 : \text{struc}'_1, \dots, x_k : \text{struc}'_k) : \text{struc} \leq \text{body}_{\text{proc}'}$$

ist die Rekursionsordnung $>_{R_{\text{proc}', x_1 \dots x_k}}$ der Prozedur proc' eine binäre Relation auf $\mathcal{C}(P)_{\text{struc}'_1} \times \dots \times \mathcal{C}(P)_{\text{struc}'_k}$ definiert durch

$$(u_1, \dots, u_k) >_{R_{\text{proc}', x_1 \dots x_k}} (v_1, \dots, v_k)$$

gdw.

bei Berechnung von $\text{eval}_P(\text{proc}'(u_1, \dots, u_k))$ muß $\text{eval}_P(\text{proc}'(v_1, \dots, v_k))$ im Zuge eines direkten rekursiven Aufrufs berechnet werden.

- Vorgriff auf **Kapitel 8**:

Definition 1 (Terminierung von \mathcal{L} -Programmen und Prozeduren)

...

Satz 2 (Fundierte Relationenbeschreibungen und terminierenden Prozeduren)

Die Relationenbeschreibung R_p einer \mathcal{L} -Prozedur p ist *fundiert* gdw. die \mathcal{L} -Prozedur p *terminiert*.

Beweis: ...

3 Optimierte Relationenbeschreibungen

- In *VeriFun* werden die Relationenbeschreibungen von Prozeduren *optimiert*.
- **Ziel:** Möglichst einfache Relationenbeschreibungen.
- **Genauer:** Bilde *fundierte Obermengen* der Rekursionsordnung.
- **Zweck:** Einfachere Basisfälle, stärkere Induktionshypothesen.
- **Generelles Vorgehen:**
 - (a) *Eliminiere Bedingungen* in $\text{cond}(t)$, die für die Fundiertheit der Relation nicht erforderlich sind (\Rightarrow *Domain Generalization*),
 - (b) *Eliminiere Substitutionspaare* in $\delta \in \Delta$, die für die Fundiertheit der Relation nicht erforderlich sind (\Rightarrow *Range Generalization*),
 - (c) Fasse atomare Relationenbeschreibungen mit identischen Hypothesenmengen H zusammen (um Disjunktheit der atomare Relationenbeschreibungen beizubehalten).
- **Wie wird festgestellt, welche Bedingungen bei *Domain Generalization* nicht erforderlich sind?** *VeriFun* führt bei einem Terminierungsbeweis für eine Prozedur Protokoll, welche Bedingungen in $\text{cond}(t)$ zum Terminierungsnachweis *nicht* benötigt werden. Diese Bedingungen werden dann bei *Domain Generalization* eliminiert (Warum darf man das? \Rightarrow *Übung*).

Beispiel 7 (Optimierte Relationenbeschreibungen für Prozeduren)

- (1) In der Prozedur *ordered* ist nur die Bedingung $\neg ?\phi(k)$ für den Terminierungsnachweis relevant. Mit *Domain Generalization* erhält man

$$R_{\text{ordered}}^{(1)} := \left\{ \begin{array}{l} \langle \{?\phi(k)\}, \emptyset \rangle, \\ \langle \{\neg ?\phi(k)\}, \{k/\tau 1(k)\} \rangle \end{array} \right\}.$$

Range Generalization ist nicht anwendbar – es gibt keine formalen Parameter außer k .

- (2) In der Prozedur *insert* ist nur die Bedingung $\neg ?\phi(k)$ für den Terminierungsnachweis relevant. Mit *Domain Generalization* erhält man

$$R'_{\text{insert}} := \left\{ \begin{array}{l} \langle \{?\phi(k)\}, \emptyset \rangle, \\ \langle \{\neg ?\phi(k)\}, \{n/n, k/\tau 1(k)\} \rangle \end{array} \right\}.$$

Der formale Parameter n ist für den Terminierungsnachweis von *insert* irrelevant. Mit *Range Generalization* erhält man folglich

$$R_{\text{insert}}^{(1)} := \left\{ \begin{array}{l} \langle \{?\phi(k)\}, \emptyset \rangle, \\ \langle \{\neg ?\phi(k)\}, \{k/\tau 1(k)\} \rangle \end{array} \right\}.$$

- (3) In der Prozedur *count* ist nur die Bedingung $\neg ?\phi(k)$ für den Terminierungsnachweis relevant. Mit *Domain Generalization* erhält man

$$R'_{\text{count}} := \left\{ \begin{array}{l} \langle \{?\phi(k)\}, \emptyset \rangle, \\ \langle \{\neg ?\phi(k)\}, \{i/i, k/\tau 1(k)\} \rangle \end{array} \right\}.$$

Der formale Parameter i ist für den Terminierungsnachweis von *count* irrelevant. Mit *Range Generalization* erhält man folglich

$$R_{\text{count}}^{(1)} := \left\{ \begin{array}{l} \langle \{?\phi(k)\}, \emptyset \rangle, \\ \langle \{\neg ?\phi(k)\}, \{k/\tau 1(k)\} \rangle \end{array} \right\}.$$

- (4) R_{isort} ist bereits optimal.

Beobachtung: Die *optimierten* Relationenbeschreibungen von *ordered*, *insert*, *isort* und *count* sind *identisch* !

(5) In der Prozedur \leq ist entweder nur die Bedingung $\neg 0(n)$ oder aber die Bedingung $\neg 0(m)$ für den Terminierungsnachweis relevant. Mit *Domain Generalization* und anschließender *Range Generalization* bzgl. $\neg 0(m)$ erhält man

$$R_{\leq}^{(1)} := \left\{ \left\langle \{ \{ ?0(n) \}, \emptyset \right\rangle, \left\langle \{ \{ \neg ?0(n) \}, \{ \{ n/\text{pred}(n) \} \} \right\rangle \right\}.$$

Mit *Domain Generalization* und anschließender *Range Generalization* bzgl. $\neg 0(n)$ erhält man

$$R_{\leq}^{(2)} := \left\{ \left\langle \{ \{ ?0(m) \}, \emptyset \right\rangle, \left\langle \{ \{ \neg ?0(m) \}, \{ \{ m/\text{pred}(m) \} \} \right\rangle \right\}.$$

Bemerkung 2 (Mehrere Relationenbeschreibungen für eine Prozedur)

- In Beispiel 7(5) erhält man 2 Relationenbeschreibungen für die Prozedur \leq , die jedoch bis auf Umbenennung von Variablen identisch sind.
- Generell kann man jedoch für eine Prozedur mehrere Relationenbeschreibungen erhalten, die *nicht durch Umbenennung* auseinander hervorgehen.
- **Ursache:** Eine Prozedur p kann aus verschiedenen Gründen terminieren, d.h. es gibt dann prinzipiell unterschiedliche Terminierungsbeweise für p und damit prinzipiell verschiedene Relationenbeschreibungen für p .

Beispiel 8 (Mehrere Relationenbeschreibungen für eine Prozedur)

Die Prozedur

```
function ntl(n : ℕ, k : list[@ITEM]) : list[@ITEM] <=
  if ?∅(k)
  then k
  else if ?0(n)
        then k
        else ntl(pred(n), tl(k))
  end_if
end_if
```

löscht die ersten n Elemente aus der Liste k (falls $n \leq |k|$).

- Relationenbeschreibung von ntl :

$$R_{\text{ntl}} := \left\{ \left\langle \{ \{ ?\emptyset(k) \}, \emptyset \right\rangle, \left\langle \{ \{ \neg ?\emptyset(k), ?0(n) \}, \emptyset \right\rangle, \left\langle \{ \{ \neg ?\emptyset(k), \neg ?0(n) \}, \{ \{ n/\text{pred}(n), k/\text{tl}(k) \} \} \right\rangle \right\}.$$

- Relationenbeschreibung nach *Domain Generalization* bzgl. $\neg ?\emptyset(k)$ und anschließender *Range Generalization* bzgl. k :

$$R_{\text{ntl}}^{(1)} := \left\{ \left\langle \{ \{ ?0(n) \}, \emptyset \right\rangle, \left\langle \{ \{ \neg ?0(n) \}, \{ \{ n/\text{pred}(n) \} \} \right\rangle \right\}.$$

- Relationenbeschreibung nach *Domain Generalization* bzgl. $\neg ?0(n)$ und anschließender *Range Generalization* bzgl. n :

$$R_{\text{ntl}}^{(2)} := \left\{ \left\langle \{ \{ ?\emptyset(k) \}, \emptyset \right\rangle, \left\langle \{ \{ \neg ?\emptyset(k) \}, \{ \{ k/\text{tl}(k) \} \} \right\rangle \right\}.$$

• Anschaulich:

Die Terminierung von ntl kann auf *zwei* Weisen begründet werden:

- (a) ntl terminiert, weil in jedem rekursiven Aufruf n bzgl. $>_{R_{\text{nat}}}$ kleiner wird.
- (b) ntl terminiert, weil k für jede *monomorphe Instanz* list' von $\text{list}[@\text{ITEM}]$ in jedem rekursiven Aufruf bzgl. $>_{R_{\text{list}'}}$ kleiner wird.

Die optimierte Relationenbeschreibung einer monomorphen (terminierenden) Prozedur repräsentiert eine (fundierte) Obermenge der (fundierten) Rekursionsordnung dieser Prozedur:

Satz 7

Für die Relationenbeschreibung R_{proc} einer Prozedur proc und eine Optimierung $R_{\text{proc}}^{(n)}$ von R_{proc} gilt

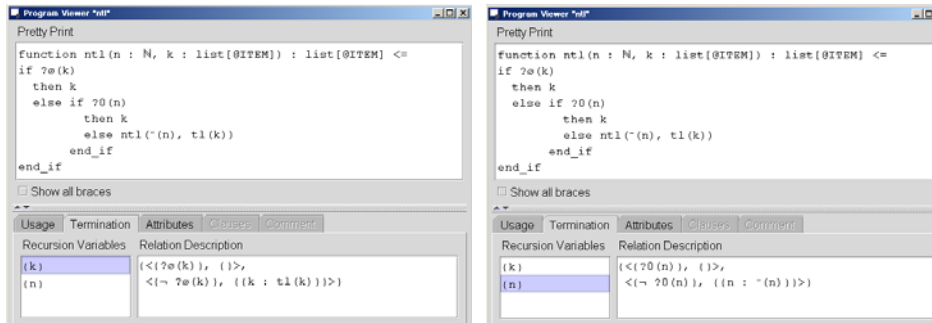
- (1) $>_{R_{\text{proc}'}, x_1 \dots x_k} \subseteq >_{R_{\text{proc}'}, x_1 \dots x_k}^{(n)}$ für jede monomorphe Instanz $R_{\text{proc}'}$ von R_{proc} und die entsprechende monomorphe Instanz $R_{\text{proc}'}^{(n)}$ von $R_{\text{proc}}^{(n)}$ sowie
- (2) R_{proc} fundiert $\curvearrowright R_{\text{proc}}^{(n)}$ fundiert.

Damit:

- *Induktionsbeweise* nach einer *optimierten* Relationenbeschreibung sind i.A. *einfacher* zu führen, denn wegen $>_{R_{\text{proc}'}, x_1 \dots x_k} \subseteq >_{R_{\text{proc}'}, x_1 \dots x_k}^{(n)}$ erhält man
 - weniger Basisfälle *sowie*
 - stärkere Induktionshypothesen.

Anzeige der optimierten Relationenbeschreibungen in VeriFun:

- Program Viewer\Termination



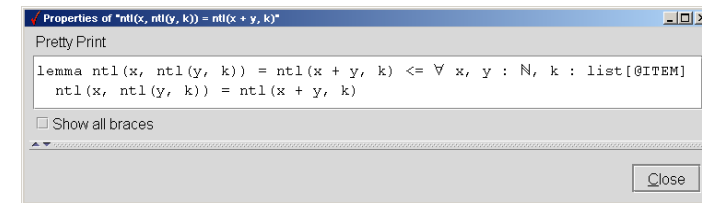
Zusammenfassung:

- Aus der *Definition* einer *terminierenden* Prozedur `proc` gewinnt man *uniform* (= “automatisch”) *fundierte Relationen* (= *Rekursionsordnung* jeder monomorphen Instanz von `proc`, => Definition 5, => **Kapitel 8**).
- Die *Rekursionsordnung* einer monomorphen Instanz von `proc` wird durch die entsprechende monomorphe Instanz der *Relationenbeschreibung* R_{proc} von `proc` syntaktisch *repräsentiert* (=> Definitionen 5 und 6).
- Durch *Domain-* und *Range Generalisierung* der Relationenbeschreibung von `proc` erhält man weitere Relationenbeschreibungen für `proc`, deren monomorphe Instanzen *fundierte Obermengen* der Rekursionsordnungen der monomorphen Instanzen von `proc` repräsentieren (=> Beispiele 7 und 8, => Satz 7).
- Aus diesen *Relationenbeschreibungen* können *uniform* (= “automatisch”) “starke” *Induktionsformeln* zum Beweis von *Lemmata* erzeugt werden (=> Abschnitt 2.3 in **Kapitel 6**).

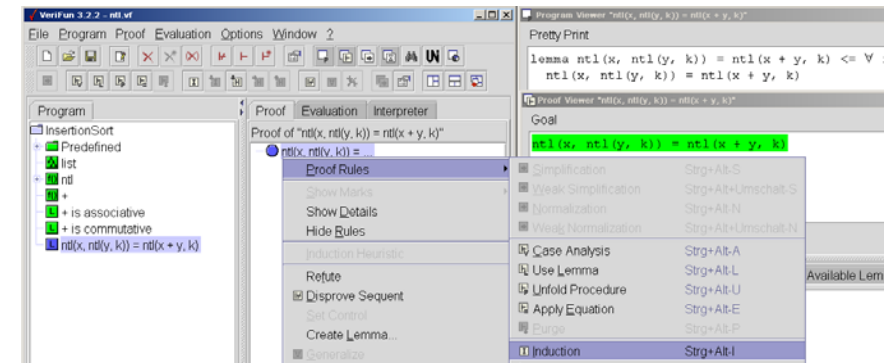
4 Induktionsbeweise in VeriFun

- Für *Induktionsbeweise* stehen in *VeriFun* zur Verfügung
 - die *Relationenbeschreibungen* aller *Datentypen* sowie
 - die *optimierten Relationenbeschreibungen* aller *Prozeduren* deren *Terminierung* bewiesen wurde (=> Status = “verified”, *Proz.-Icon* = “grün”).
- *Interaktiver Induktionsbeweis* eines Lemmas:
 - Benutzer:** Selektiert *Wurzel* des Beweisbaums im *Proof Window*,
 - Benutzer:** Wählt *HPL-Regel Induction* in Menue *Proof\Proof Rules*,
 - Benutzer:** Selektiert *Relationenbeschreibung* im Dialogfenster der *HPL-Regel Induction*,
 - Benutzer:** Wählt *Variable*, über die *Induktion* geführt werden soll,
 - System:** Erzeugt *Induktionsformeln* (= *Basis-* und *Schrittfälle* der gewählten *Induktion*, => Abschnitt 2.3 in **Kapitel 6**).
- *Automatischer Induktionsbeweis* (*Verify-Taktik*, => Abschn. 2.3 in **Kapitel 3**):
 - Benutzer:** Selektiert *Lemma* im *Program Window*,
 - Benutzer:** Wählt *Verify* in Menue *Program*,
 - System:** Übernimmt die Rolle des Benutzers (= Schritte (a) – (d) bei interaktivem *Induktionsbeweis*).

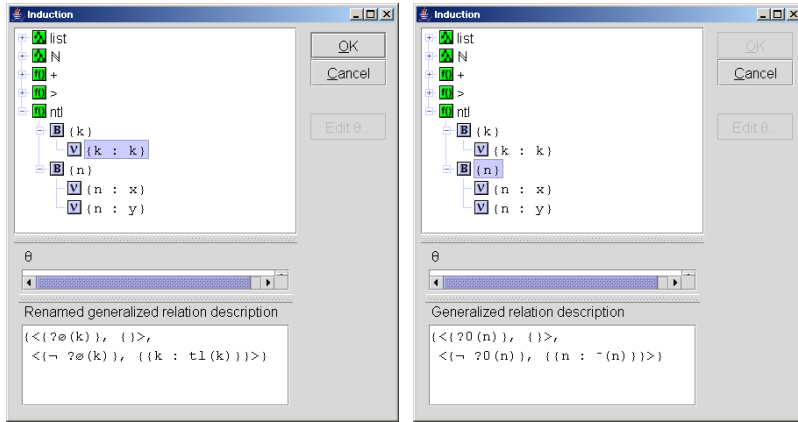
Beispiel: Beweise



Aufruf HPL-Regel Induction:



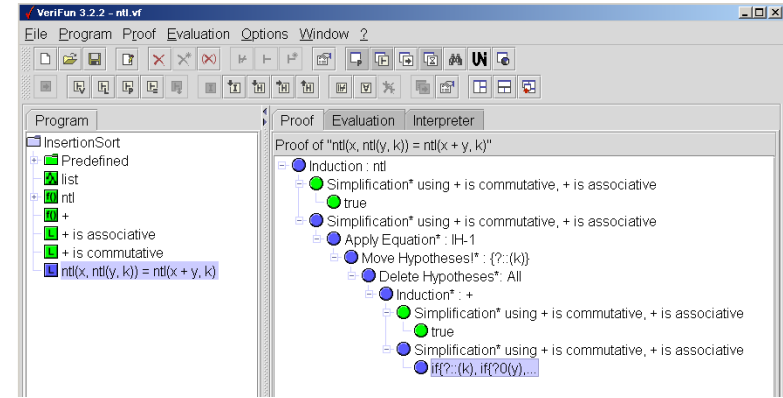
Dialog *HPL-Regel Induction*:



Die Prozedur *nt1* schlägt 2 Induktionsschemata zum Beweis vor:

- Strukturelle Induktion bzgl. *list*[@ITEM] sowie
- Strukturelle Induktion bzgl. *nat*

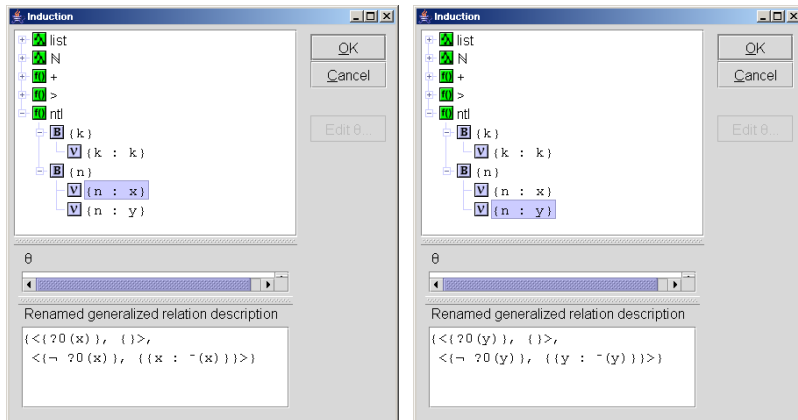
Wir wählen strukturelle Induktion bzgl. *list*[@ITEM]:



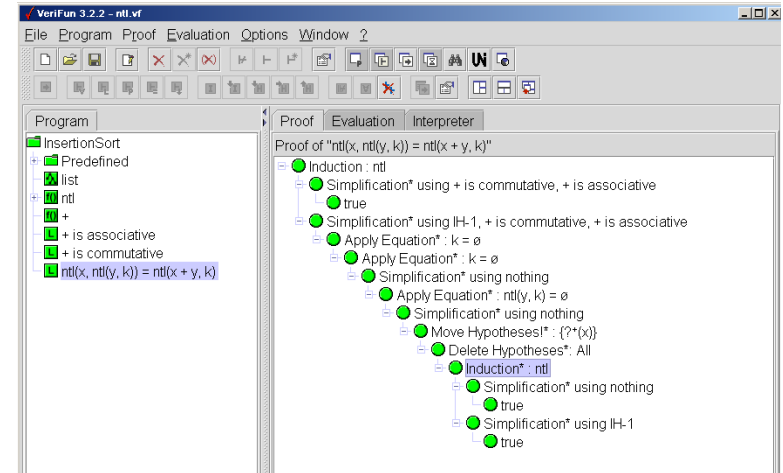
Nicht erfolgreich :-)

Wir wählen strukturelle Induktion bzgl. *nat*:

Hier müssen wir entscheiden: Induktion über *x* oder über *y* ?

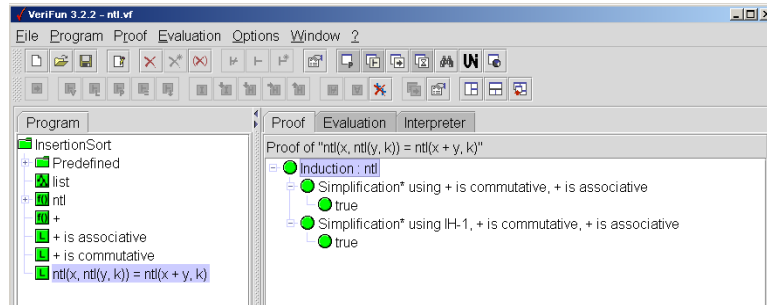


Wir wählen strukturelle Induktion bzgl. *nat* über *x*:



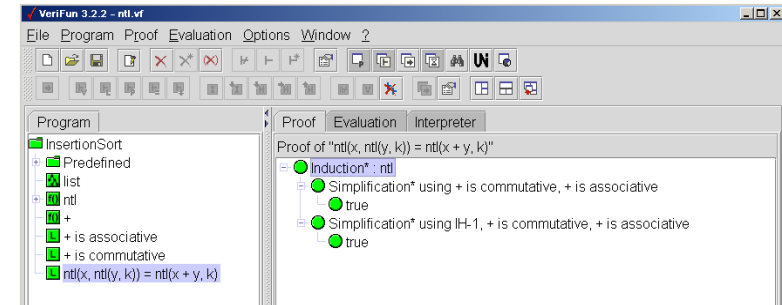
- Umständlicher Beweis – *geschachtelte* Induktion
- Aber immerhin: *Erfolgreich* :-)

Wir wählen strukturelle Induktion bzgl. nat über y:



- *Erfolgreich* und einfacher Beweis :-))
- *Frage*: Warum führt Induktion über y eigentlich zu einem einfacheren Beweis als Induktion über x ? (=> Übung)

Wir arbeiten *nicht interaktiv* sondern starten den Beweis mit *Verify*:



- *Verify* wählt strukturelle Induktion bzgl. nat über y :-)))

Inside VeriFun: Die Induktionsheuristik

- Für ein Lemma, in dem ein Prozeduraufruf $p(\dots)$ vorkommt, ist
 - ein *Induktionsbeweis* nach (einer Obermenge) der *Rekursionsordnung* von p (=> optimierte Relationenbeschreibung $R_p^{(n)}$ von p) “oft” *erfolgreich*,
 - vorausgesetzt, die *aktuellen Parameter* in $p(\dots)$ an Stellen von *Rekursionsvariablen* von $R_p^{(n)}$ sind *Variable*.⁸
- Wurde zuerst ab 1971 im NQTHM-System von R.S. Boyer & J.S. Moore erfolgreich realisiert (<http://www.cs.utexas.edu/users/moore/best-ideas/nqthm/>).
- Die *Induktionsheuristik* ist nicht nur für maschinelle Beweiser nützlich, sondern auch für *menschliche Beweisversuche* !
- *Was passiert, wenn ein Lemma* Aufrufe verschiedener Prozeduren *enthält*?
 - Kein Problem solange die *optimierten* Relationenbeschreibungen *übereinstimmen* (wie in der Fallstudie *InsertionSort*).
 - Andernfalls muß heuristisch ausgewählt werden (=> *Tie-breaking Heuristik*).

⁸ Eine Variable x ist eine *Rekursionsvariable* einer Relationenbeschreibung $R = \{A_1, \dots, A_h\}$ gdw. $x \in DEF(\delta)$ für ein $i \in \{1, \dots, h\}$ mit $A_i = \langle H_i, \Delta_i \rangle$ und $\delta \in \Delta_i$.

Werden mit den *Relationenbeschreibungen* eines \mathcal{L} -Programms *immer alle Induktionsaxiome zur Verfügung gestellt, die zum Korrektheitsnachweis des \mathcal{L} -Programms erforderlich sind?*

- **Nein!** Es gibt Lemmata über Programme P , für deren Beweis Induktionsschemata erforderlich sind, die *weder* durch *strukturelle Rekursion* der Datentypen noch durch (Obermengen der) *Rekursionsordnungen* der Prozeduren von P gegeben sind.
- **Aber:** Sehr seltene Fälle.
- **Was dann tun:** Schreibe eine *terminierende* “Dummy”-Prozedur, aus deren Rekursionsordnung das gewünschte Induktionsschema (per Relationenbeschreibung) gewonnen wird.
- **Damit:** Das gewünschte Induktionsaxiom steht jetzt über die Relationenbeschreibung der “Dummy”-Prozedur zur Verfügung.
- **Beispiel:** Siehe Beweis des Lemmas $y \neq 0 \rightarrow 2 * (y)^2 \neq (x)^2$ mittels Induktion nach der “Dummy”-Prozedur \mathcal{R} (=> Abschnitt 4 in **Kapitel 4** sowie Fallstudie *sqrt_2-is-irrational.vf* auf der Web-Seite der Vorlesung).