Formale Grundlagen der Informatik 3 –

7. Strukturelle Induktion und Induktion nach Rekursion von Prozeduren

Christoph Walther
TU Darmstadt

1 Strukturelle Induktion und Rekursion

Definition 1 (Rekursive Konstruktoren, Rekursiv definierte Datentypen)
Für einen Datentyp $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$ *definiert durch*

$$\begin{array}{l}\textbf{structure}\ \texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]\ \texttt{<=} \\ \quad \texttt{cons}_1\texttt{(sel}_{1,1}\texttt{:struc}_{1,1}\texttt{,} \ldots \texttt{,sel}_{1,n_1}\texttt{:struc}_{1,n_1}\texttt{)}, \\ \qquad \vdots \\ \quad \texttt{cons}_k\texttt{(sel}_{k,1}\texttt{:struc}_{k,1}\texttt{,} \ldots \texttt{,sel}_{k,n_k}\texttt{:struc}_{k,n_k}\texttt{)}\end{array} \tag{1}$$

und einen Konstruktor $\texttt{cons}_i$ *von* $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$ *mit*

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

ist

- $h \in \{1, \ldots, n_i\}$ eine *Rekursionsposition* von $\texttt{cons}_i$ gdw. $\texttt{struc}_{i,h} = \texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$,
- $\texttt{cons}_i$ *rekursiv* gdw. $\texttt{cons}_i$ mindestens eine Rekursionsposition besitzt,
- $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$ ein *rekursiv definierter Datentyp* gdw. $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$ mindestens einen rekursiven Konstruktor besitzt. ■

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

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

und

- jedem *rekursiven Konstruktor* `cons` mit den Selektoren $\texttt{sel}_1, \ldots, \texttt{sel}_h$ an den *Rekursionspositionen* von `cons` die atomare Relationenbeschreibung

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

zugeordnet.

- Dem Datentyp $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$ wird die Relationenbeschreibung

$R_{\texttt{struc}[@\texttt{T}_1,\ldots,@\texttt{T}_n]} := \{A_{\texttt{cons}} \mid \texttt{cons}$ ist Konstruktor von $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]\}$

zugeordnet. ■

Beispiel 2 *(Relationenbeschreibung eines Datentyps)*

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

$$R_{\texttt{nat}} := \left\{ \begin{array}{l} \langle \{\texttt{?0}(u)\}, \emptyset \rangle, \\ \langle \{\texttt{?succ}(u)\}, \{\{u/\texttt{pred}(u)\}\} \rangle \end{array} \right\}.$$

(2) Man erhält für
`structure list[@T] <= ø, [infixr,100] ::(hd : @T, tl : list[@T])`

$$R_{\texttt{list}[@\texttt{T}]} := \left\{ \begin{array}{l} \langle \{\texttt{?ø}(u)\}, \emptyset \rangle, \\ \langle \{\texttt{?::}(u)\}, \{\{u/\texttt{tl}(u)\}\} \rangle \end{array} \right\}.$$

(3) Man erhält für `structure sexpr[@T] <=`
`atom(data:@T), nil, cons(car:sexpr[@T], cdr:sexpr[@T])`

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

(4) Man erhält für `structure tree[@T] <=`
`leaf, node(left:tree[@T], key:@T, right:tree[@T])`

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

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

Definition 3 (Strukturelle Ordnung eines monomorphen Datentyps)
Sei

- $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$ ein Datentyp (gegeben wie in Definition 1),
- $R_{\texttt{struc}[@\texttt{T}_1,\ldots,@\texttt{T}_n]}$ die Relationenbeschreibung von $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$,
- $\texttt{struc}'$ eine monomorphe Instanz von $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$, und
- $R_{\texttt{struc}'}$ die entsprechende monomorphe Instanz von $R_{\texttt{struc}[@\texttt{T}_1,\ldots,@\texttt{T}_n]}$.

Dann heißt $>_{R_{\texttt{struc}'},u}$ *die* strukturelle Ordnung *des monomorphen Datentyps* $\texttt{struc}'$. [1,2] ■

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

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

Beispiel 3 *(Strukturelle Ordnung)*

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

(2) Für den Datentyp $\texttt{list[nat]}$ gilt $\texttt{n :: k} >_{\texttt{list[nat]}} \texttt{k}$, also z.B. $\texttt{n}_1 \texttt{ :: } \texttt{n}_2 \texttt{ :: ø} >_{\texttt{list[nat]}} \texttt{n}_2 \texttt{ :: ø} >_{\texttt{list[nat]}} \texttt{ø} \not>_{\texttt{list[nat]}} \ldots$. Es gilt jedoch *nicht* $\texttt{n}_1 \texttt{ :: } \texttt{n}_2 \texttt{ :: ø} >_{\texttt{list[nat]}} \texttt{ø}$, denn $>_{\texttt{list[nat]}}$ ist *nicht transitiv*.

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

Satz 4

Für jeden Datentyp $\texttt{struc}[@\texttt{T}_1, \ldots, @\texttt{T}_n]$ *ist* $R_{\texttt{struc}[@\texttt{T}_1,\ldots,@\texttt{T}_n]}$ *eine* fundierte *Relationenbeschreibung.*

Beweis: *Sei* $R_{\texttt{struc}'}$ *eine beliebige monomorphe Instanz von* $R_{\texttt{struc}[@\texttt{T}_1,\ldots,@\texttt{T}_n]}$. *Mit* $u >_{R_{\texttt{struc}'}} v$ *gilt dann "*v *ist* echter Teilterm *von* u*". Wäre* $>_{R_{\texttt{struc}'}}$ *nicht fundiert, so gäbe es Terme mit unendlich vielen echten Teiltermen. Da dies nicht möglich ist, ist* $>_{R_{\texttt{struc}'}}$ *fundiert. Folglich ist jede monomorphe Instanz* $R_{\texttt{struc}'}$ *von* $R_{\texttt{struc}[@\texttt{T}_1,\ldots,@\texttt{T}_n]}$ *fundiert, und damit auch* $R_{\texttt{struc}[@\texttt{T}_1,\ldots,@\texttt{T}_n]}$.[3] ■

[3] Siehe Definition 8 in **Kapitel 6**.

Sprechweisen:

- Wir definieren eine Prozedur $\mathtt{p}$ durch *strukturelle Rekursion* über $\mathtt{struc}[@\mathtt{T}_1, \ldots, @\mathtt{T}_n]$, falls die optimierte (=> Abschnitt 3) Relationenbeschreibung $R_{\mathtt{p}}$ von $\mathtt{p}$ (modulo Umbenennung der Variablen von $R_{\mathtt{p}}$) identisch der Relationenbeschreibung $R_{\mathtt{struc}[@\mathtt{T}_1,\ldots,@\mathtt{T}_n]}$ ist.

- Wir führen den Beweis eines Lemmas durch *strukturelle Induktion* über $\mathtt{struc}[@\mathtt{T}_1, \ldots, @\mathtt{T}_n]$, falls die Induktionsformeln des Beweises aus der Relationenbeschreibung $R_{\mathtt{struc}[@\mathtt{T}_1,\ldots,@\mathtt{T}_n]}$ 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 $\texttt{struc}'$ wird durch dessen *Relationenbeschreibung* $R_{\texttt{struc}'}$ 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 $case\ \{b; d_1 : r_1, \ldots, d_k : r_k\}$ 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 $f(\ldots, t'', \ldots)$ 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*

$$\textbf{function}\ \texttt{proc(x}_1\texttt{:struc}_1, \ldots, \texttt{x}_k\texttt{:struc}_k\texttt{):struc} <= \mathit{body}_{\texttt{proc}}$$

wird

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

  $$A_t = \left\langle \mathit{cond}\left(t, \mathit{body}_{\texttt{proc}}\right), \emptyset \right\rangle$$

  zugeordnet und

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

  $$\texttt{proc(t}_{1,1}, \ldots, \texttt{t}_{k,1}\texttt{)},\ \ldots\ , \texttt{proc(t}_{1,n}, \ldots, \texttt{t}_{k,n}\texttt{)}$$

  enthält, die atomare (*rekursive*) Relationenbeschreibung

$$A_t = \left\langle \mathit{cond}\left(t, \mathit{body}_{\texttt{proc}}\right), \left\{\left\{x_1/t_{1,1}, \ldots, x_k/t_{k,1}\right\}, \ldots, \left\{x_1/t_{1,n}, \ldots, x_k/t_{k,n}\right\}\right\} \right\rangle$$

  zugeordnet.

- Der Prozedur `proc` wird die Relationenbeschreibung

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

  zugeordnet. ■

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

Es gilt

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

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

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

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

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

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),
- $R_{\texttt{proc}}$ die Relationenbeschreibung von `proc`,
- `proc′` eine monomorphe Instanz von `proc` und
- $R_{\texttt{proc}'}$ die entsprechende monomorphe Instanz von $R_{\texttt{proc}}$.

Dann heißt $>_{R_{\texttt{proc}'},x_1 \ldots x_k}$ *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 $R_{\texttt{proc}}$ gewonnen wurden.[7]

---

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

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

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

Beispiel 6 *(Rekursionsordnung einer Prozedur)*

Für die Prozeduren aus der InsertionSort*-Fallstudie (=> **Kapitel 2**) gilt:*

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

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

$1 :: 3 :: 2 :: 8 :: \text{ø} >_{R_{\texttt{ordered},k}}$
$3 :: 2 :: 8 :: \text{ø} \not>_{R_{\texttt{ordered},k}}$.

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

Bemerkung 1 (Rekursionsordnung einer Prozedur)
Für

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

$$\textbf{function}\ \texttt{proc}'\texttt{(}\texttt{x}_1\texttt{:struc}'_1, \ldots, \texttt{x}_k\texttt{:struc}'_k\texttt{):struc} <= \mathit{body}_{\texttt{proc}'}$$

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

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

gdw.

bei Berechnung von $eval_P(\texttt{proc(}u_1, \ldots, u_k\texttt{)})$ *muß* $eval_P(\texttt{proc(}v_1, \ldots, v_k\texttt{)})$ *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 $R_{\mathtt{p}}$ einer $\mathcal{L}$-Prozedur $\mathtt{p}$ ist *fundiert* gdw. die $\mathcal{L}$-Prozedur $\mathtt{p}$ *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 $cond(t)$, die für die Fundiertheit der Relation nicht erforderlich sind (=> *Domain Generalization*),
  (b) *Eliminiere Substitutionspaare* in $\delta \in \Delta$, die für die Fundiertheit der Relation nicht erforderlich sind (=> *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 $cond(t)$ zum Terminierungsnachweis *nicht* benötigt werden. Diese Bedingungen werden dann bei *Domain Generalization* eliminiert (Warum darf man das? => *Übung*).

Beispiel 7 *(Optimierte Relationenbeschreibungen für Prozeduren)*

(1) In der Prozedur `ordered` ist nur die Bedingung $\neg\texttt{?ø}(k)$ für den Terminierungsnachweis relevant. Mit *Domain Generalization* erhält man

$$R^{(1)}_{\texttt{ordered}} := \left\{ \begin{array}{l} \langle \{\texttt{?ø(}k\texttt{)}\}, \emptyset \rangle, \\ \langle \{\neg\texttt{?ø(}k\texttt{)}\}, \{\{k/\texttt{tl(}k\texttt{)}\}\} \rangle \end{array} \right\}.$$

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

(2) In der Prozedur `insert` ist nur die Bedingung $\neg\texttt{?ø}(k)$ für den Terminierungsnachweis relevant. Mit *Domain Generalization* erhält man

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

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

$$R^{(1)}_{\texttt{insert}} := \left\{ \begin{array}{l} \langle \{\texttt{?ø(}k\texttt{)}\}, \emptyset \rangle, \\ \langle \{\neg\texttt{?ø(}k\texttt{)}\}, \{\{k/\texttt{tl(}k\texttt{)}\}\} \rangle \end{array} \right\}.$$

(3) In der Prozedur `count` ist nur die Bedingung $\neg\texttt{?ø}(k)$ für den Terminierungsnachweis relevant. Mit *Domain Generalization* erhält man

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

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

$$R^{(1)}_{\texttt{count}} := \left\{ \begin{array}{l} \langle \{\texttt{?ø(}k\texttt{)}\}, \emptyset \rangle, \\ \langle \{\neg\texttt{?ø(}k\texttt{)}\}, \{\{k/\texttt{tl(}k\texttt{)}\}\} \rangle \end{array} \right\}.$$

(4) $R_{\texttt{isort}}$ 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 $\neg\mathtt{0}(n)$ oder aber die Bedingung $\neg\mathtt{0}(\mathtt{)}m\mathtt{)}$ für den Terminierungsnachweis relevant. Mit *Domain Generalization* und anschließender *Range Generalization* bzgl. $\neg\mathtt{0}(\mathtt{)}m\mathtt{)}$ erhält man

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

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

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

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 ?ø(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* $\mathtt{n} \leq |\mathtt{k}|$*).*

- Relationenbeschreibung von ntl:

$$R_{\mathtt{ntl}} := \left\{ \begin{array}{l} \langle \{ \mathtt{?ø}(k) \}, \emptyset \rangle, \\ \langle \{ \neg \mathtt{?ø}(k), \mathtt{?0}(n) \}, \emptyset \rangle, \\ \langle \{ \neg \mathtt{?ø}(k), \neg \mathtt{?0}(n) \}, \{ \{ n/\mathtt{pred}(n), k/\mathtt{tl}(k) \} \} \rangle \end{array} \right\}.$$

- Relationenbeschreibung nach *Domain Generalization* bzgl. $\neg\texttt{?ø(}k\texttt{)}$ und anschließender *Range Generalization* bzgl. k :

$$R^{(1)}_{\texttt{ntl}} := \left\{ \begin{array}{l} \langle \{\texttt{?0(}n\texttt{)}\}, \emptyset \rangle, \\ \langle \{\neg\texttt{?0(}n\texttt{)}\}, \{\{n/\texttt{pred(}n\texttt{)}\}\} \rangle \end{array} \right\}.$$

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

$$R^{(2)}_{\texttt{ntl}} := \left\{ \begin{array}{l} \langle \{\texttt{?ø(}k\texttt{)}\}, \emptyset \rangle, \\ \langle \{\neg\texttt{?ø(}k\texttt{)}\}, \{\{k/\texttt{tl(}k\texttt{)}\}\} \rangle \end{array} \right\}.$$

- **Anschaulich:**
  Die Terminierung von `ntl` kann auf *zweierlei* Weisen begründet werden:

  (a) `ntl` terminiert, weil in jedem rekursiven Aufruf `n` bzgl. $>_{R_{\texttt{nat}}}$ kleiner wird.

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

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

Satz 7

Für die Relationenbeschreibung $R_{\texttt{proc}}$ *einer Prozedur* `proc` *und eine Optimierung* $R^{(n)}_{\texttt{proc}}$ *von* $R_{\texttt{proc}}$ *gilt*

(1) $>_{R_{\texttt{proc}'},x_1 \ldots x_k} \;\subseteq\; >_{R^{(n)}_{\texttt{proc}'},x_1 \ldots x_k}$ für jede monomorphe Instanz $R_{\texttt{proc}'}$ von $R_{\texttt{proc}}$ und die entsprechende monomorphe Instanz $R^{(n)}_{\texttt{proc}'}$ von $R^{(n)}_{\texttt{proc}}$ sowie

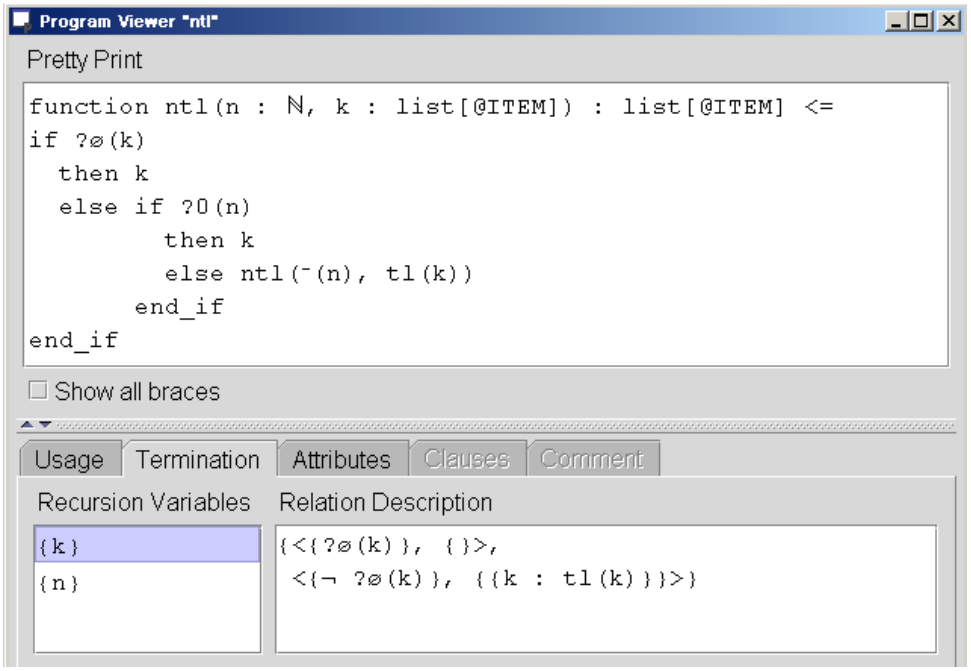
(2) $R_{\texttt{proc}}$ fundiert $\curvearrowright$ $R^{(n)}_{\texttt{proc}}$ fundiert.

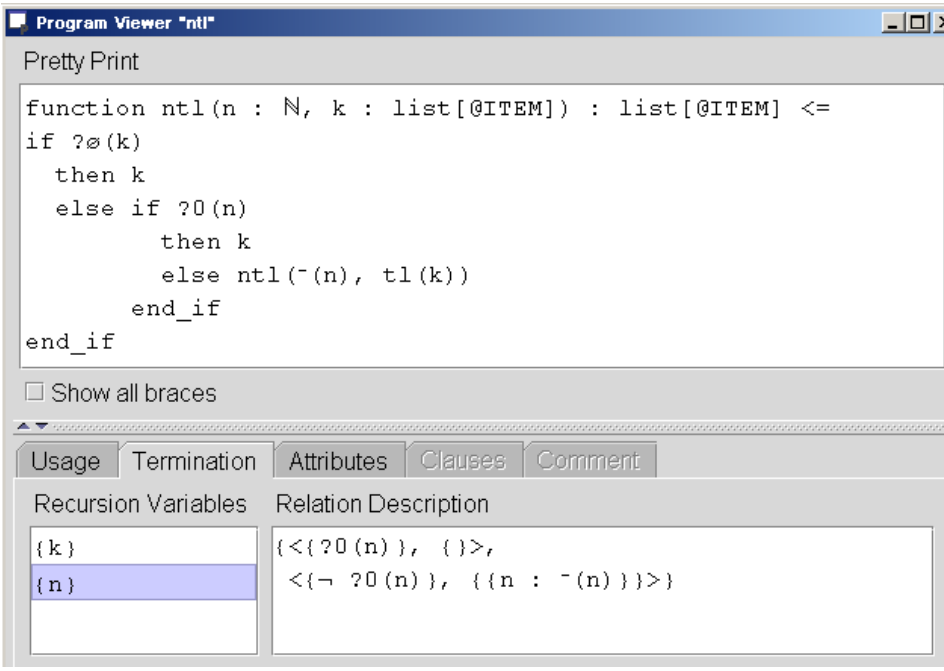
Damit:

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

Anzeige der optimierten Relationenbeschreibungen in *VeriFun*:

- *Program Viewer\Termination*





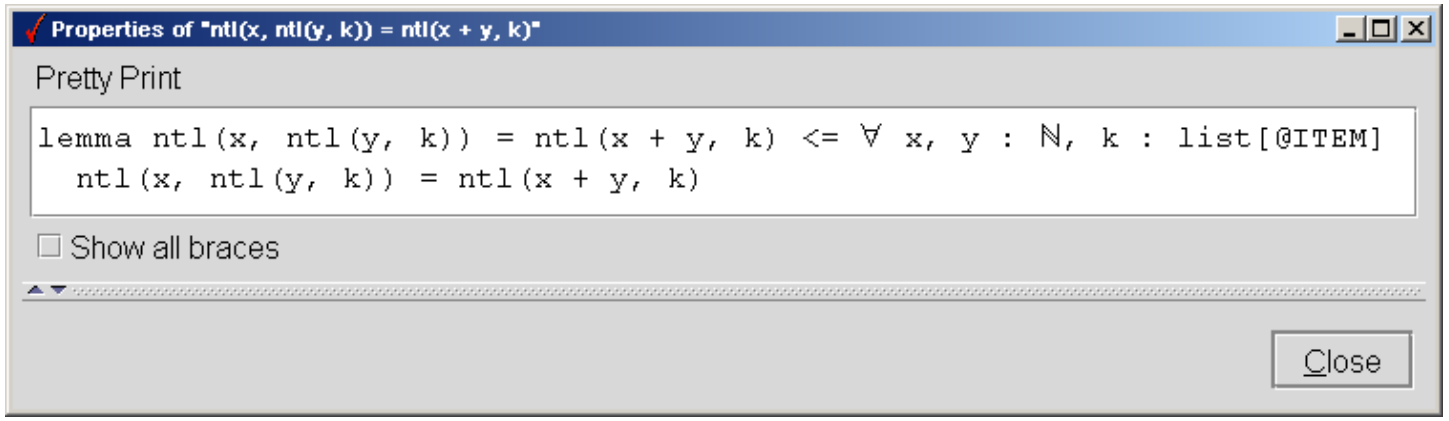
Zusammenfassung:

- 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* $R_{\texttt{proc}}$ 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**).

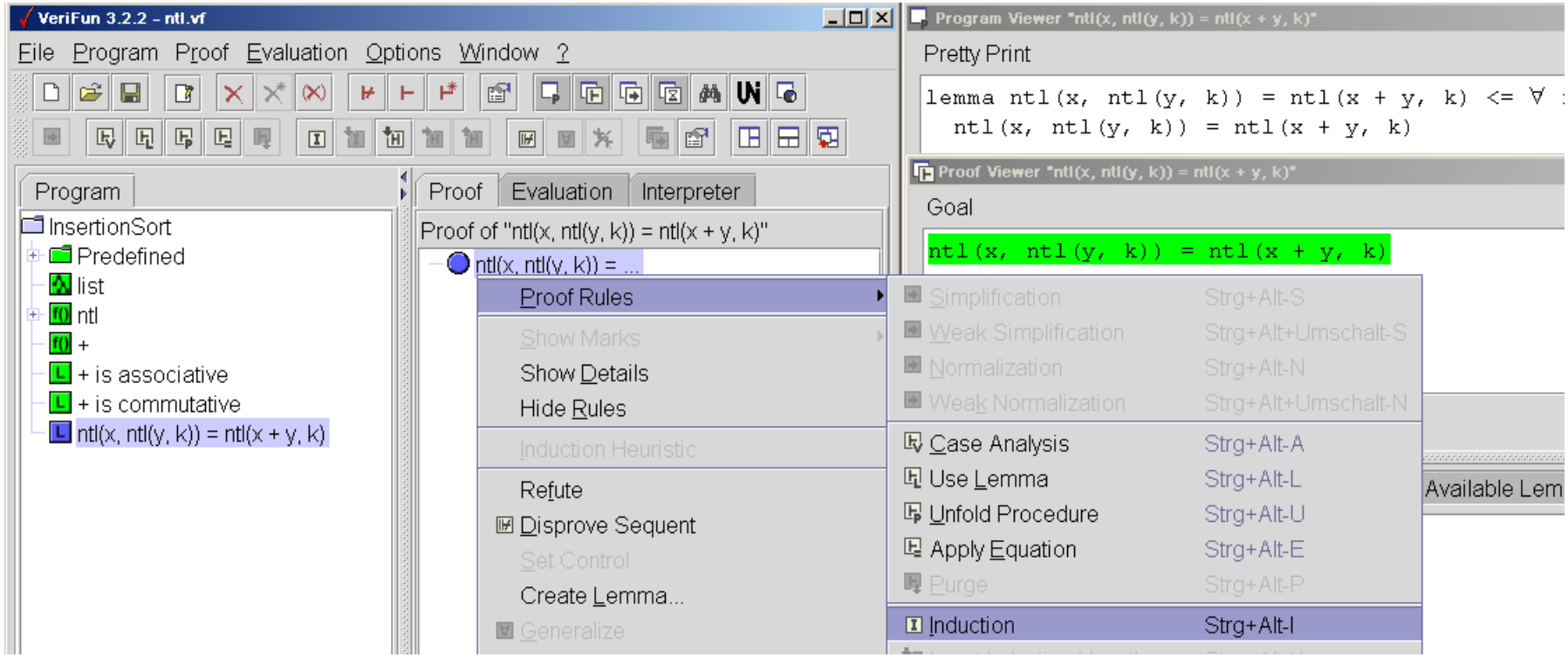
4 Induktionsbeweise in *VeriFun*

- 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:
  (a) **Benutzer:** Selektiert *Wurzel* des Beweisbaums im *Proof Window*,
  (b) **Benutzer:** Wählt *HPL*-Regel *Induction* in Menue *Proof\Proof Rules*,
  (c) **Benutzer:** Selektiert Relationenbeschreibung im Dialogfenster der *HPL*-Regel *Induction*,
  (d) **Benutzer:** Wählt *Variable*, über die *Induktion* geführt werden soll,
  (e) **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**):
  (a) **Benutzer:** Selektiert *Lemma* im *Program Window*,
  (b) **Benutzer:** Wählt *Verify* in Menue *Program*,
  (c) **System:** Übernimmt die Rolle des Benutzers (= Schritte (a) – (d) bei interaktivem Induktionsbeweis).

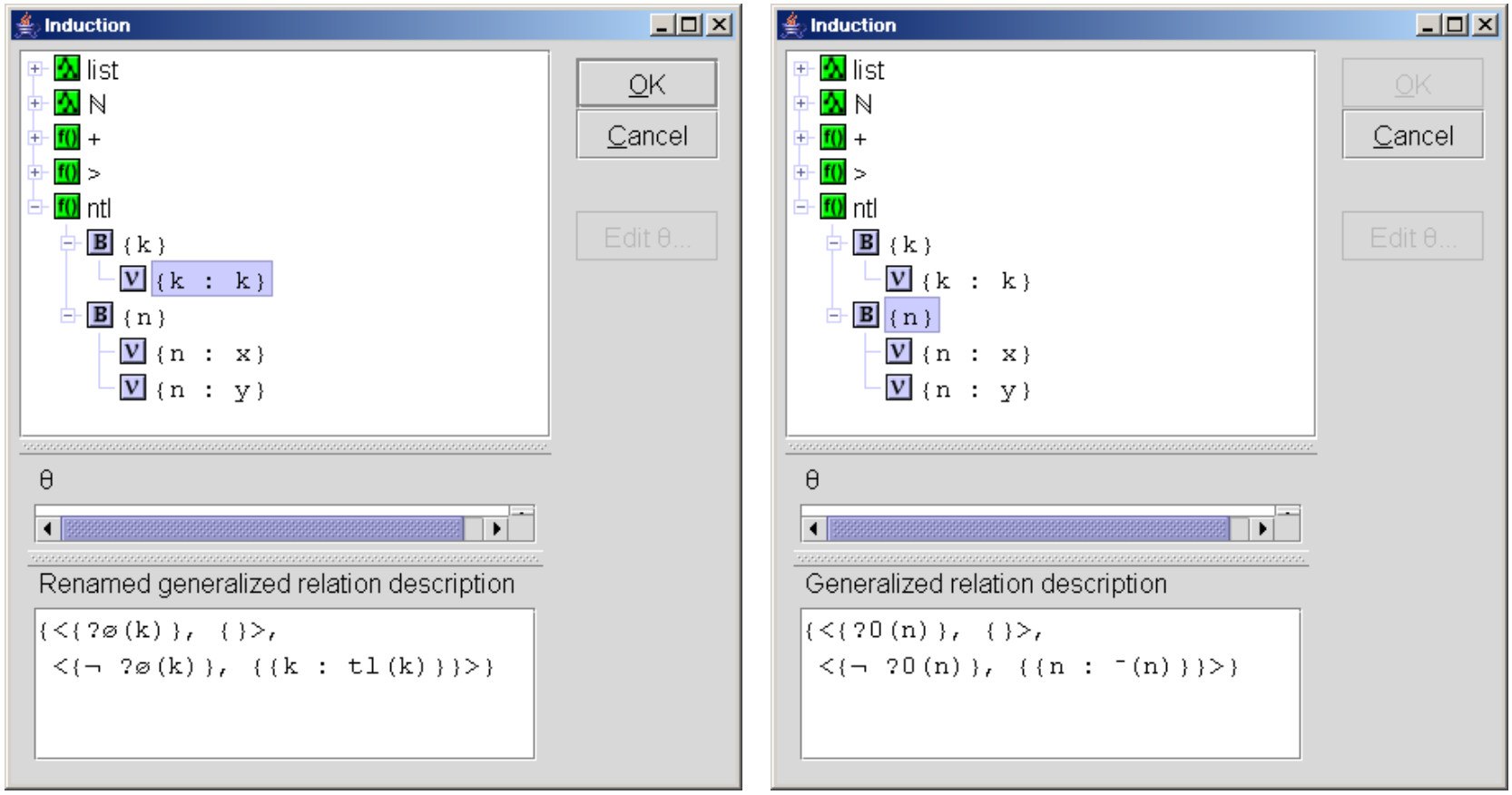
Beispiel: Beweise



Aufruf *HPL*-Regel *Induction*:



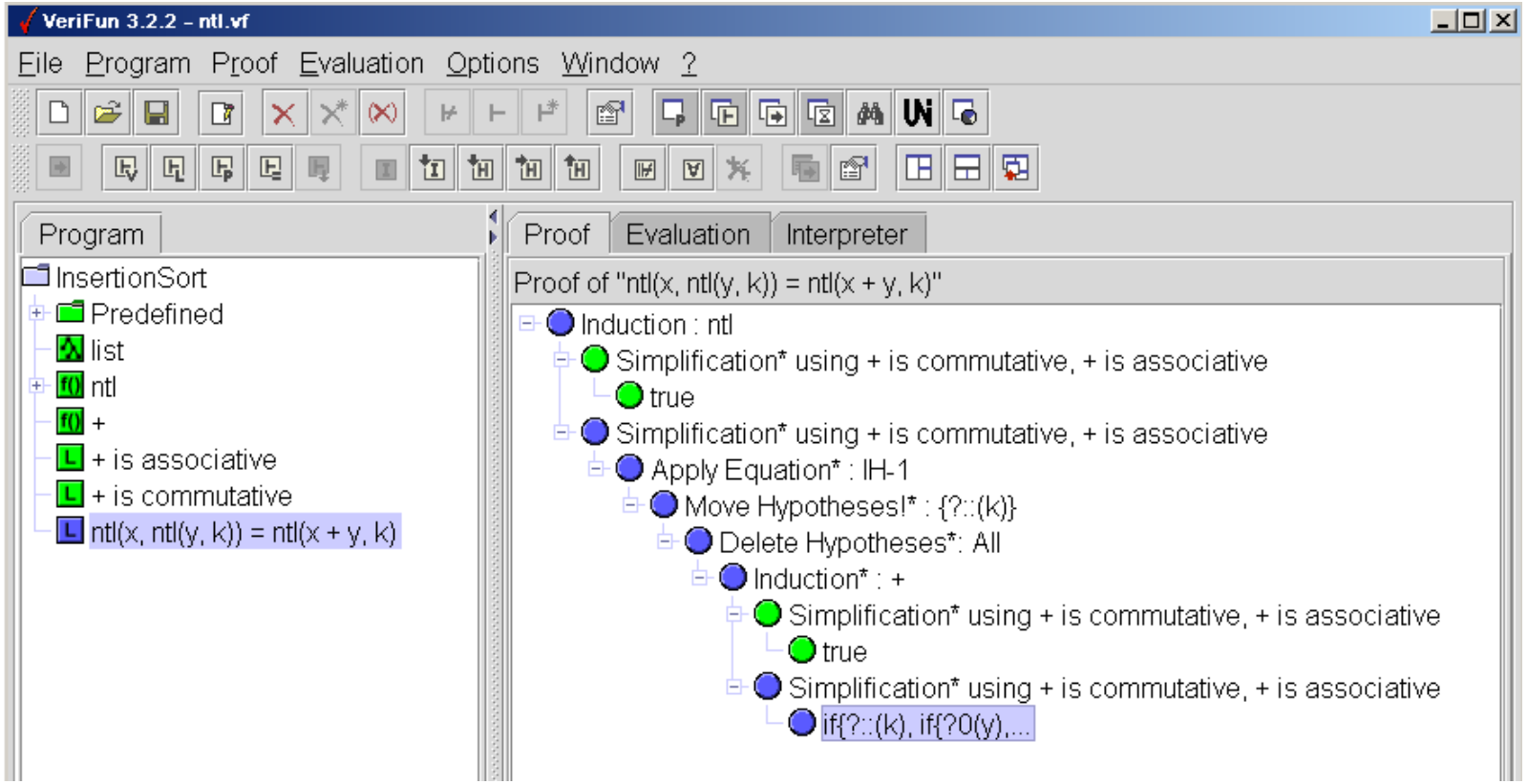
Dialog *HPL*-Regel *Induction*:



Die Prozedur `ntl` schlägt 2 Induktionsschemata zum Beweis vor:

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

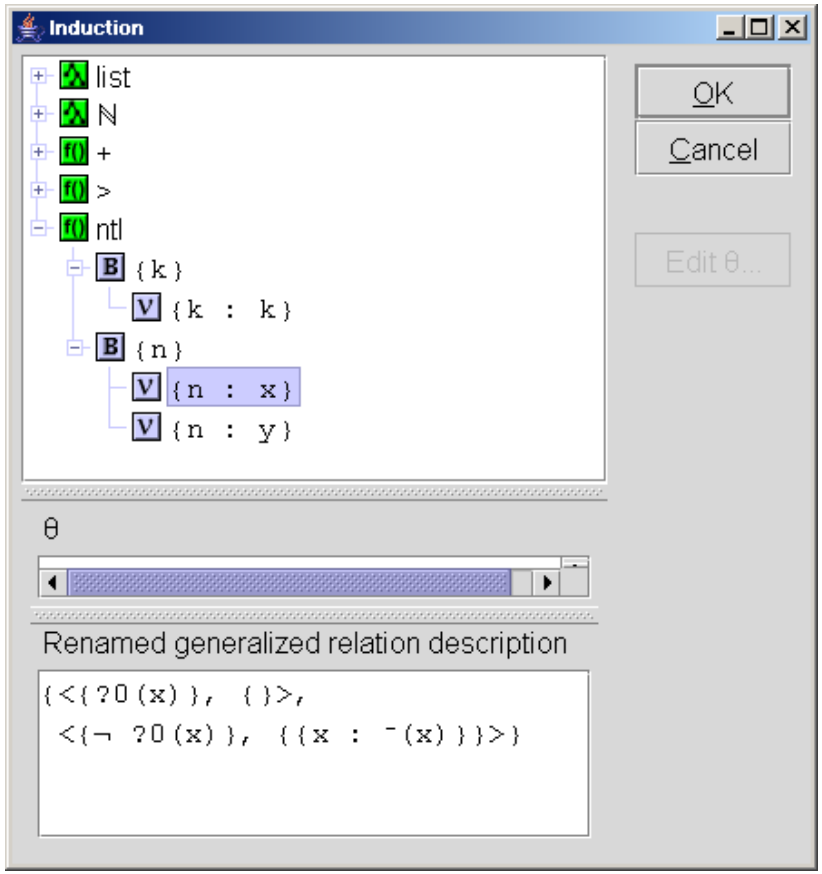
Wir wählen strukturelle Induktion bzgl. `list[@ITEM]`:

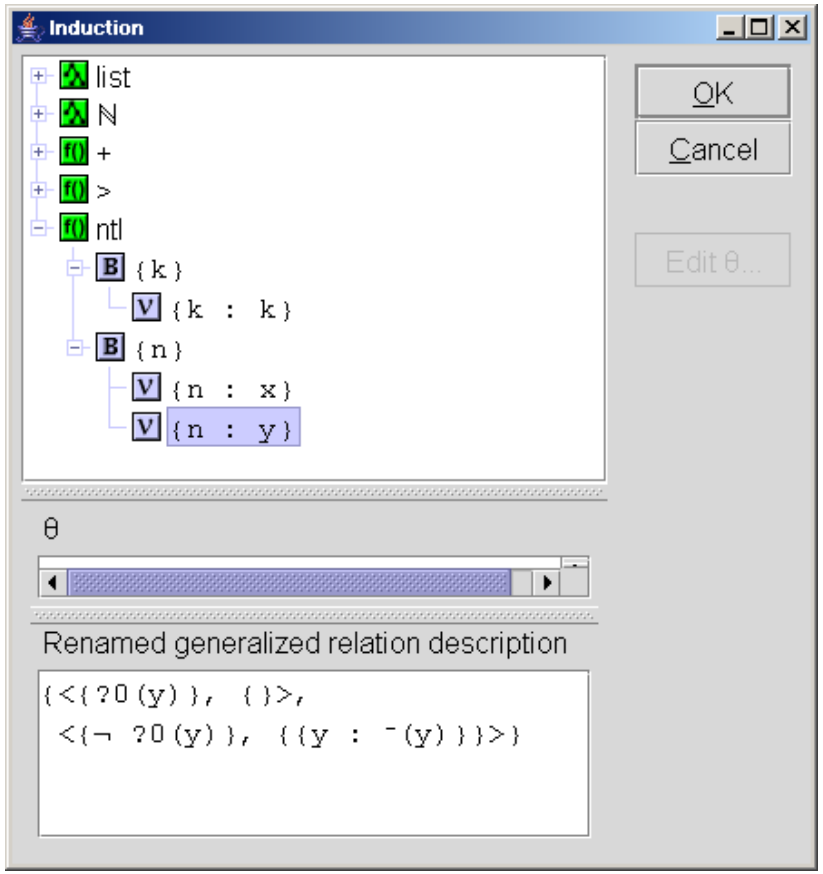


Nicht erfolgreich **:–(**

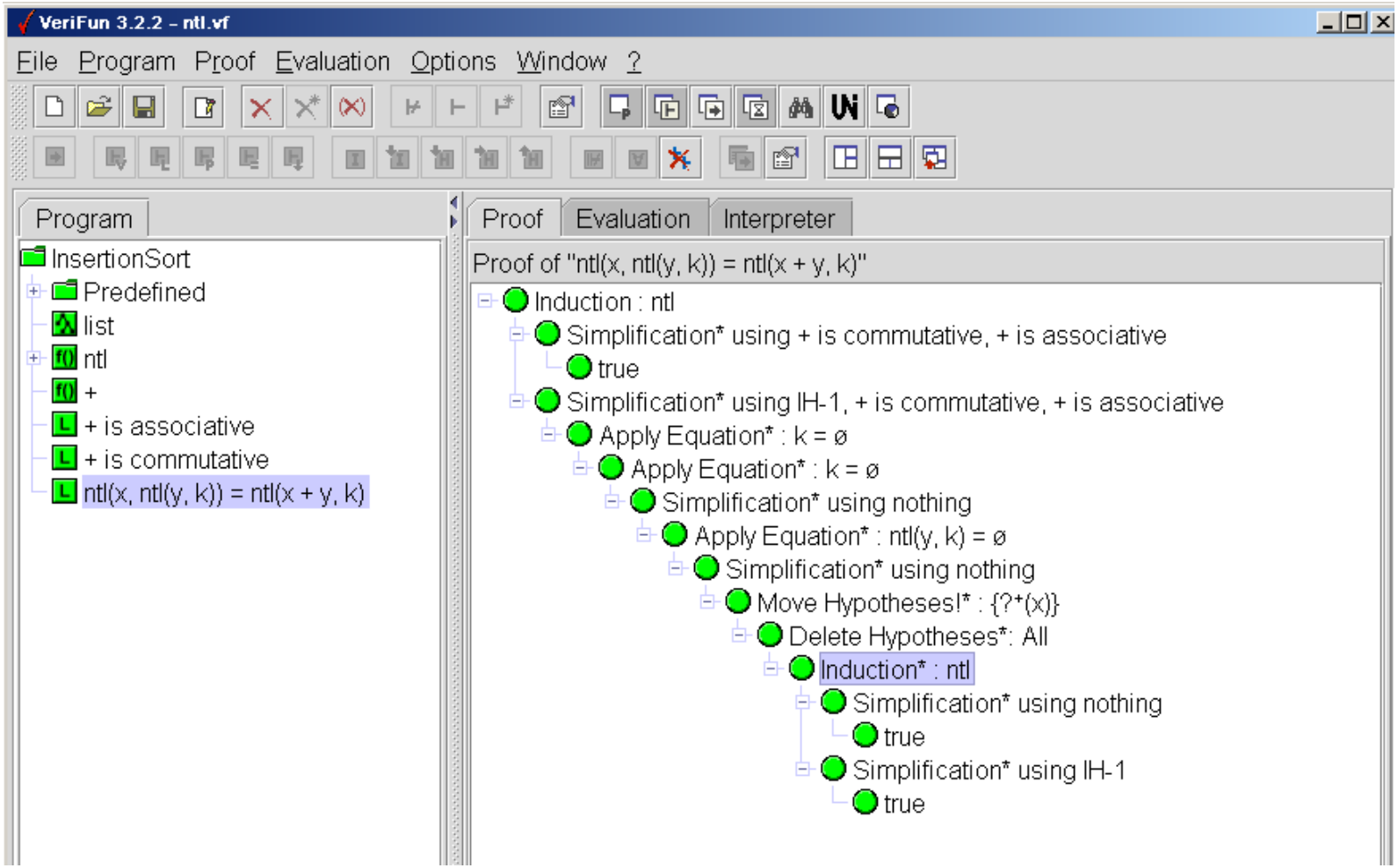
Wir wählen strukturelle Induktion bzgl. `nat`:

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



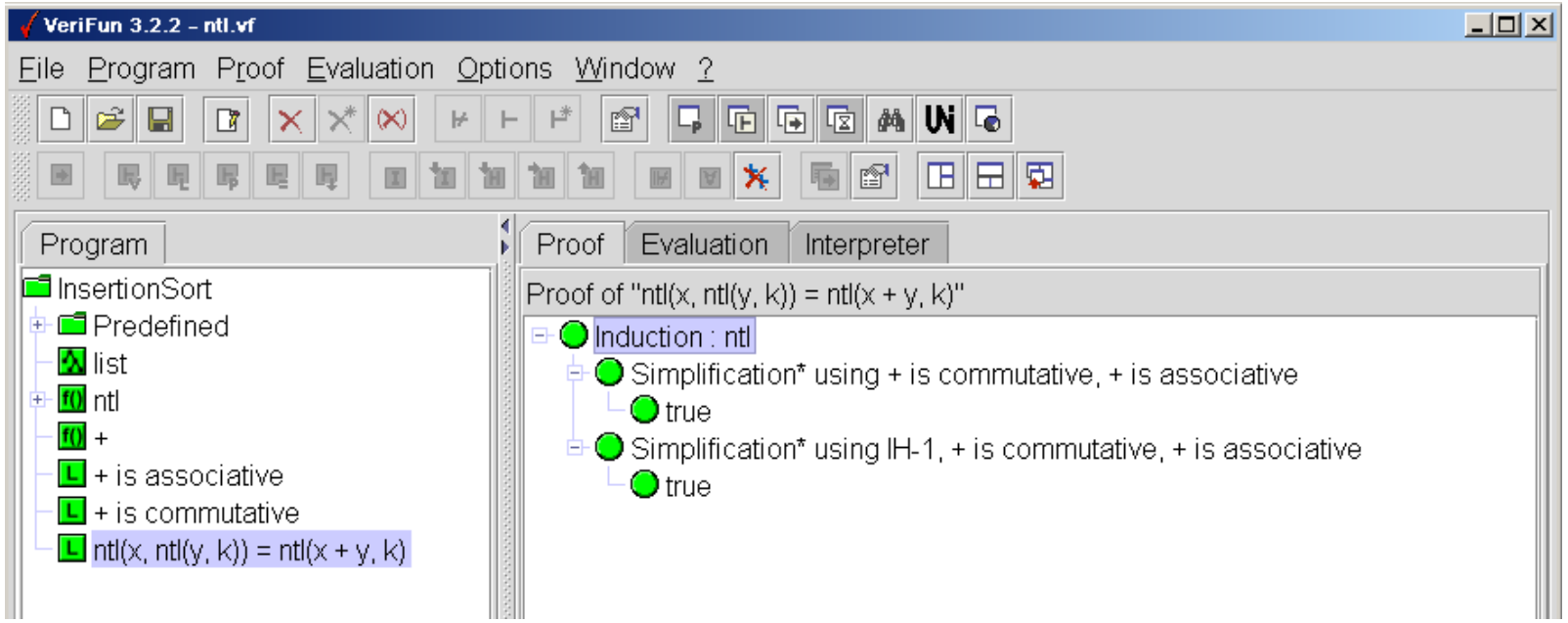


Wir wählen strukturelle Induktion bzgl. `nat` über `x`:



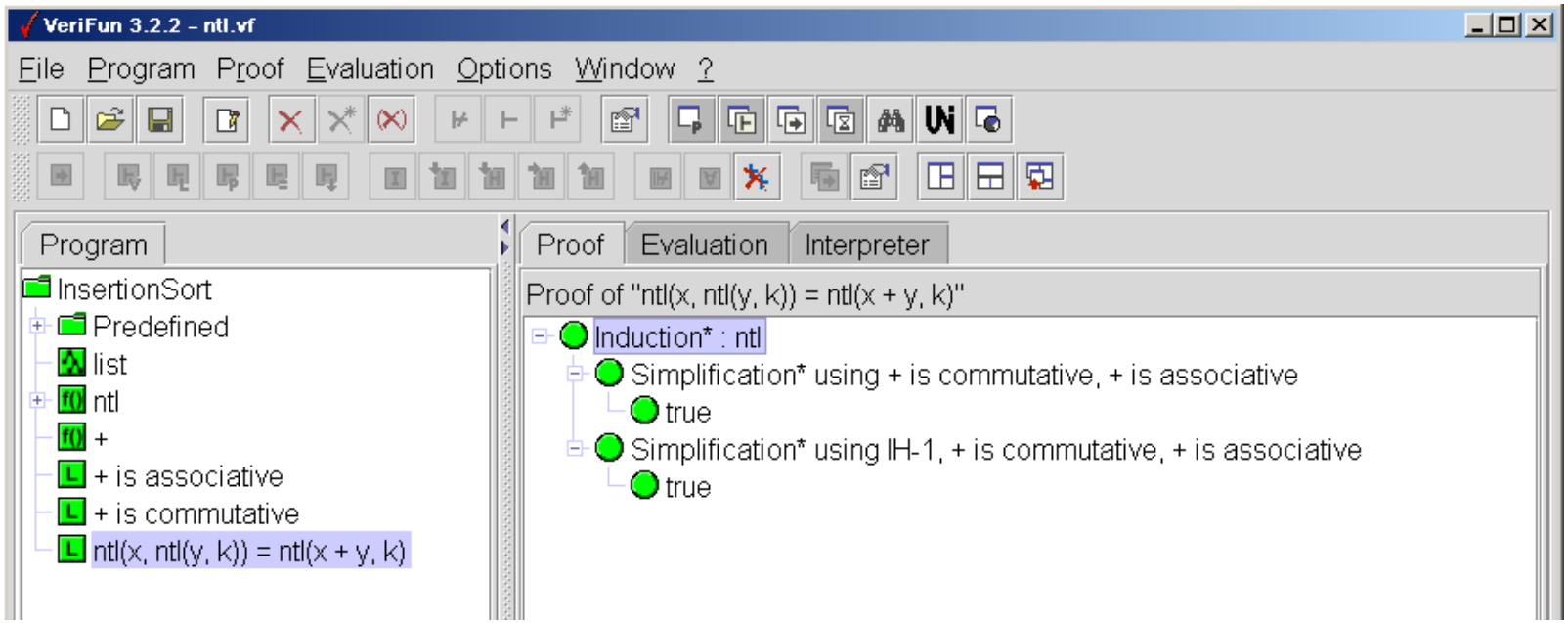
- Umständlicher Beweis – *geschachtelte* Induktion
- Aber immerhin: *Erfolgreich* **:–)**

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(...)` vorkommt, ist
  - ein *Induktionsbeweis* nach (einer Obermenge) der *Rekursionsordnung* von `p` (=> optimierte Relationenbeschreibung $R_{\mathtt{p}}^{(n)}$ von `p`) "oft" *erfolgreich*,
  - vorausgesetzt, die *aktuellen Parameter* in `p(...)` an Stellen von *Rekursionsvariablen* von $R_{\mathtt{p}}^{(n)}$ 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 $R = \{A_1, \ldots, A_h\}$ gdw. $x \in DEF(\delta)$ für ein $i \in \{1, \ldots, h\}$ 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 $\Re$ (=> Abschnitt 4 in **Kapitel 4** sowie Fallstudie `sqrt_2-is-irrational.vf` auf der Web-Seite der Vorlesung).