

## Formale Grundlagen der Informatik 3 –

### 10. Symbolische Auswertung

Christoph Walther  
TU Darmstadt

## 1 Übersicht

- Der *Symbolic Evaluator* ist ein System, das die Axiome (= Gleichungen) aus Datentyp- und Prozedurdefinitionen sowie aus *bewiesenen* Lemmata und Induktionshypothesen mittels *Auswertungsregeln* auf Beweisziele (= boolesche Terme) anwendet, und diese dadurch umformt.
  - *Beispiel*: Auswertungsregel **Execute procedure call** – Anwendung des Axioms einer Prozedurdefinition (vgl. Berechnungsregel (19) in **Kapitel 5**).
  - *Beispiel*: Auswertungsregel **Keep then-part** – Anwendung eines Axioms einer Datentypdefinition ( $\text{if}\{\text{true}, x, y\} \vdash x$ , vgl. Berechnungsregel (9) in **Kapitel 5**).
  - *Beispiel*: Auswertungsregel **Appropriate selector** – Anwendung eines Axioms einer Datentypdefinition ( $\text{pred}(\text{succ}(t)) \vdash t$ , vgl. Berechnungsregel (5) in **Kapitel 5**).
- Des weiteren werden (aus Axiomen) *abgeleitete* Lemmata verwendet
  - *Beispiel*: Auswertungsregel **Skip condition** –  $\text{if}\{a, b, b\} \vdash b$ .
  - *Beispiel*: Auswertungsregel **Skip alternatives** –  $\text{if}\{a, \text{true}, \text{false}\} \vdash a$ .

- “*Evaluator*”: die Arbeitsweise orientiert sich an der Arbeitsweise des Interpreters  $\text{eval}_P$  (= Ausrechnen durch Umformungen; s. **Kapitel 5**)
- “*Symbolic*”: es wird mit Ausdrücken “gerechnet”, in denen auch *Variable* anstelle konkreter Werte stehen.
- Der *Symbolic Evaluator* löst die *Indeterminismen* bei Umformungen durch geeignete *Heuristiken* auf:
  - *Entscheide*: Auf *welchen* Teilterm (= *Redex*) des Beweisziels wird *welche* Auswertungsregel angewendet.
- *Heuristisches Haltekriterium* – wann sollen *keine* Auswertungsregeln mehr angewendet werden:
  - *Beispiel*: Auf den Aufruf einer rekursiv definierten Prozedur kann *Execute procedure call* unendlich oft ausgeführt werden – die Heuristik muß entscheiden, wann es sinnvoll ist einen Prozeduraufruf durch den Prozedurrumpf zu ersetzen bzw. diesen zu belassen.

## 2 HPL-Kalkül: Computed Proof Rules

- Der *HPL-Kalkül* besitzt 5 sogenannte *Computed Proof Rules*, die (mit einer Ausnahme) über das Menue *Proof\Proof Rules* angewendet werden:
  1. **Simplification**: Bei Anwendung dieser Beweisregel auf eine *HPL-Sequenz*  $\langle H, IH \Vdash \text{goal} \rangle$  wird der *Symbolic Evaluator* aufgerufen, um *goal* unter Verwendung von *H*, *IH*, den Datentyp- und Prozedurdefinitionen sowie den bereits bewiesenen Lemmata symbolisch auszuwerten
  2. **Weak Simplification**: Wie *Simplification*, jedoch werden *rekursiv definierte* Prozeduren *nicht* “*geöffnet*” (kein *Execute procedure call*)
  3. **Normalization**: Wie *Simplification*, jedoch werden *keine* Prozeduren “*geöffnet*” (kein *Execute procedure call*)
  4. **Weak Normalization**: Wie *Normalization*, jedoch werden weder Induktionshypothesen noch Lemmata angewendet (kein *Affirmative assumption*)
  5. **Inconsistency**: Wertet eine Hypothese  $h \in H$  unter  $H \setminus \{h\}$  zu *false* aus, falls dies möglich ist. (In diesem Fall ist  $\langle H, IH \Vdash \text{goal} \rangle$  trivialerweise bewiesen)
    - \* Die Anwendbarkeit von *Inconsistency* wird vom System automatisch überprüft, deshalb steht diese Regel in *Proof\Proof Rules* nicht zur Verfügung

- \* *Beispiel:* Für  $seq = \langle \{x>y, \neg x>y\}, IH \Vdash 1=2 \rangle$  erhält man  $1=2 \vdash_{AXP, Lem_{verified}, seq} true$
- \* *Beispiel:* Für  $seq = \langle \{?0(x), ?^+(x)\}, IH \Vdash 1=2 \rangle$  erhält man  $1=2 \vdash_{AXP, Lem_{verified}, seq} true$

- Der *Symbolic Evaluator* arbeitet automatisch, Benutzerinteraktionen sind nicht möglich
- Die Arbeit des *Symbolic Evaluator* kann jedoch in *Proof Control* angehalten oder abgebrochen werden

### 3 Symbolische Auswertung – Ein Beispiel

```

• function [infixr, 30] \(k : list[@T], i : @T) : list[@T] <=
  if ?∅(k)
  then ∅
  else if i = hd(k)
        then tl(k)
        else hd(k) :: (tl(k) \ i)
  end_if
end_if

```

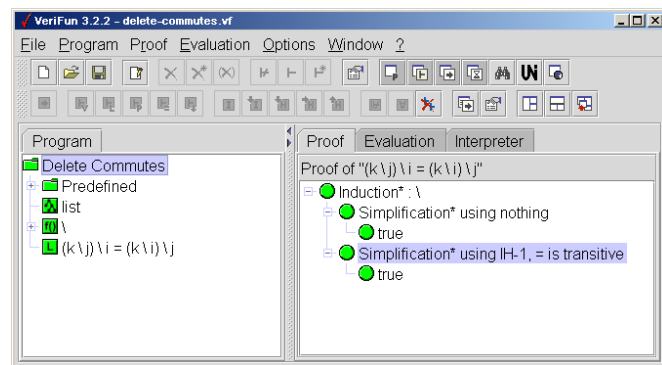
löscht das erste (von links gelesen) Vorkommen von  $i$  in der Liste  $k$ .

**Behauptung:** Es gilt

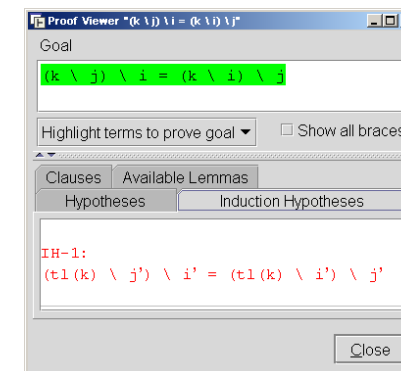
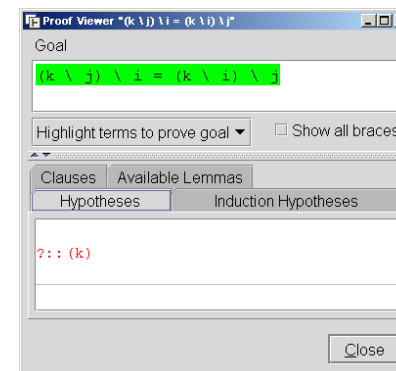
• lemma  $(k \setminus j) \setminus i = (k \setminus i) \setminus j <=$   
 $\forall k : list[@ITEM], i, j : @ITEM \ (k \setminus j) \setminus i = (k \setminus i) \setminus j$

d.h., die Reihenfolge, in der Listenelemente gelöscht werden, ist irrelevant.

- **Benutzer:** Aufruf *Verify*
- **System:**
  - *Induction* (strukturelle Induktion über  $list[@T]$   
 $\Rightarrow$  optimierte Relationenbeschreibung von  $\setminus$ )
  - $2 \times$  *Simplification* (2 Induktionsformeln = 1 Basisfall und 1 Schrittfall)
  - *fertig!*



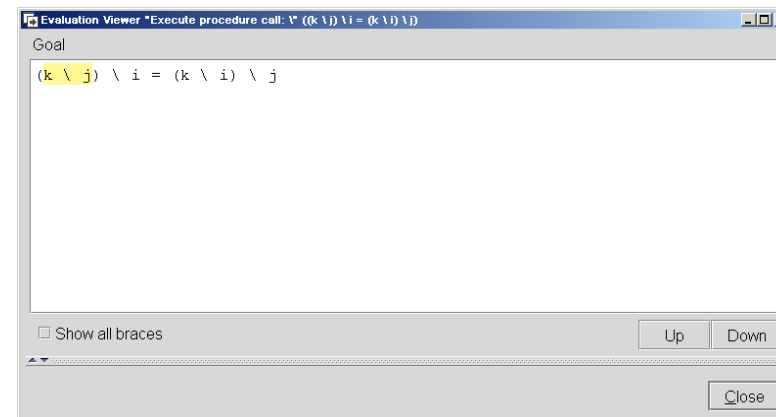
- **HPL-Sequenz**  $\langle H, IH \Vdash goal \rangle$  im Schrittfall



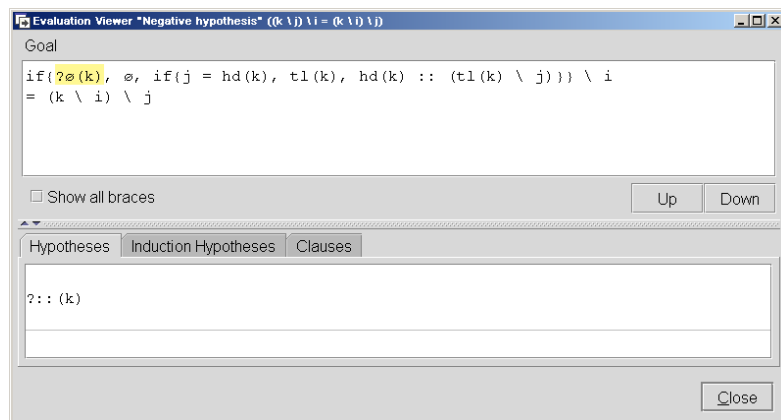
- **Beweisziel goal:**  $(k \setminus j) \setminus i = (k \setminus i) \setminus j$
- Strukturelle Induktion über  $list[@ITEM]$ , also
  - *Hypothese(n) H:*  $? :: (k)$  – wir sind im Schrittfall, d.h. Liste  $k$  nicht leer
  - *Induktionshypothese(n) IH:*  
 $\forall i', j' : @ITEM \ (tl(k) \setminus j') \setminus i' = (tl(k) \setminus i') \setminus j'$

**Bemerkung 1** Hypothesen und Induktionshypothesen werden rot angezeigt, wenn sie im Beweis verwendet wurden (andernfalls schwarz).

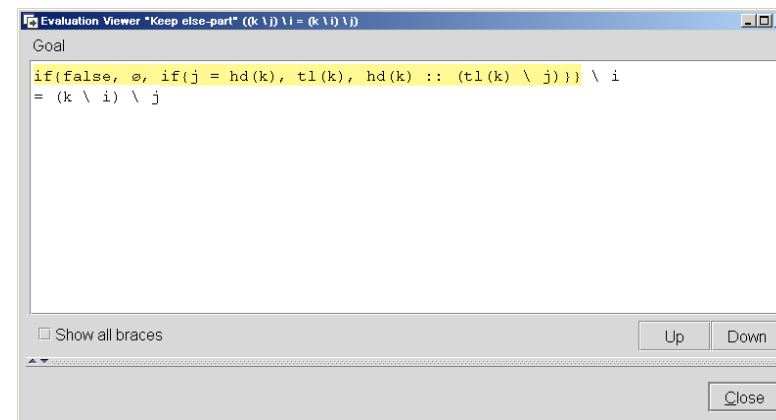
- **Beweis einer Induktionsformel** durch den *Symbolic Evaluator*
  - Beweise (= symbolische Auswertungen) werden durch Anwendung sogenannter *Auswertungsregeln* geführt
  - der *Symbolic Evaluator* verwendet die *Axiome*, die aus den Definitionen der *Datentypen*, *Prozeduren* und *Lemmata* gewonnen werden (=> **Kapitel 9**)
  - der *Symbolic Evaluator* verwendet (nur) Axiome *bewiesener* Lemmata
  - *Induktionshypothesen* werden wie (bewiesene) Lemmata behandelt
  - die einzelnen Schritte einer symbolischen Auswertung werden im *Evaluation Viewer* angezeigt:
    - \* Öffnen durch Doppelklick auf einen mit *Simplification* (oder *Weak Simplification* oder ...) bezeichneten *HPL*-Beweisknoten im *Proof Window* und dann
    - \* mit den *Up/Down*-Buttons oder den  $\downarrow / \uparrow$ -Pfeiltasten den nächsten/vorherigen Auswertungsschritt anzeigen lassen
    - \* *Name der Auswertungsregel* in der Kopfzeile, der *Redex* der Regelanwendung ist *gelb* unterlegt



- *Execute procedure call*:
  - Ersetze Prozeduraufruf durch instantiierten (formale Parameter  $\rightarrow$  aktuelle Parameter) Prozedurrumpf.



- *Negative Hypothesis*:
  - Wir sind im Induktionsschritt, d.h. es gilt  $? :: (k)$  (in Worten “k ist nicht die leere Liste”).
  - Also darf  $?\emptyset(k)$  durch *false* ersetzt werden.



- *Keep else-part*:
  - Semantik von *if* –  $\text{if}\{\text{false}, X, Y\}$  darf durch *Y* ersetzt werden.

Goal

```
if(j = hd(k), tl(k), hd(k) :: (tl(k) \ j)) \ i = (k \ i) \ j
```

Buttons: Show all braces, Up, Down, Close

- *Distribute if-argument:*
  - Ausdrücke der Form  $g(\text{if}\{a, b, c\}, d)$  dürfen durch  $\text{if}\{a, g(b, d), g(c, d)\}$  ersetzt werden.

Goal

```
if(j = hd(k), tl(k) \ i, hd(k) :: (tl(k) \ j) \ i) = (k \ i) \ j
```

Buttons: Show all braces, Up, Down, Close

- *Execute procedure call:*
  - Ersetze Prozeduraufruf durch instantiierten (formale Parameter  $\rightarrow$  aktuelle Parameter) Prozedurrumpf.

Goal

```
if(j = hd(k),
  tl(k) \ i,
  if(?empty(hd(k) :: (tl(k) \ j)),
    empty,
    if(i = hd(hd(k) :: (tl(k) \ j)),
      tl(hd(k) :: (tl(k) \ j)),
      hd(hd(k) :: (tl(k) \ j)) :: (tl(hd(k) :: (tl(k) \ j)) \ i)))
  = (k \ i) \ j
```

Buttons: Show all braces, Up, Down, Close

- *Negative structure test:*
  - Keine Liste der Form  $n :: l$  ist leer.
  - Damit darf  $?empty(hd(k) :: \dots)$  durch `false` ersetzt werden.

Goal

```
if(j = hd(k),
  tl(k) \ i,
  if(false,
    empty,
    if(i = hd(hd(k) :: (tl(k) \ j)),
      tl(hd(k) :: (tl(k) \ j)),
      hd(hd(k) :: (tl(k) \ j)) :: (tl(hd(k) :: (tl(k) \ j)) \ i)))
  = (k \ i) \ j
```

Buttons: Show all braces, Up, Down, Close

- *Keep else-part:*
  - Semantik von `if` – `if{false, X, Y}` darf durch `Y` ersetzt werden.

```

Evaluation Viewer "Appropriate selector" ((k \ i) \ j)
Goal
if(j = hd(k),
  tl(k) \ i,
  if(i = hd(hd(k) :: (tl(k) \ j)),
    tl(hd(k) :: (tl(k) \ j)),
    hd(hd(k) :: (tl(k) \ j)) :: (tl(hd(k) :: (tl(k) \ j)) \ i)))
= (k \ i) \ j

 Show all braces
Up Down
Close

```

- *Appropriate selector*:
  - Selektoren sind “invers” zu den Konstruktoren, zu denen sie gehören.
  - Also darf jeder Ausdruck der Form  $hd(n :: 1)$  durch  $n$  ersetzt werden.

```

Evaluation Viewer "Appropriate selector" ((k \ i) \ j)
Goal
if(j = hd(k),
  tl(k) \ i,
  if(i = hd(k),
    tl(hd(k) :: (tl(k) \ j)),
    hd(hd(k) :: (tl(k) \ j)) :: (tl(hd(k) :: (tl(k) \ j)) \ i)))
= (k \ i) \ j

 Show all braces
Up Down
Close

```

- *Appropriate selector*:
  - Selektoren sind “invers” zu den Konstruktoren, zu denen sie gehören.
  - Also darf jeder Ausdruck der Form  $tl(n :: 1)$  durch  $n$  ersetzt werden.

```

Evaluation Viewer "Appropriate selector" ((k \ i) \ j)
Goal
if(j = hd(k),
  tl(k) \ i,
  if(i = hd(k),
    tl(k) \ j,
    hd(hd(k) :: (tl(k) \ j)) :: (tl(hd(k) :: (tl(k) \ j)) \ i)))
= (k \ i) \ j

 Show all braces
Up Down
Close

```

- *Appropriate selector*:
  - Selektoren sind “invers” zu den Konstruktoren, zu denen sie gehören.
  - Also darf jeder Ausdruck der Form  $hd(n :: 1)$  durch  $n$  ersetzt werden.

```

Evaluation Viewer "Appropriate selector" ((k \ i) \ j)
Goal
if(j = hd(k),
  tl(k) \ i,
  if(i = hd(k), tl(k) \ j, hd(k) :: (tl(hd(k) :: (tl(k) \ j)) \ i)))
= (k \ i) \ j

 Show all braces
Up Down
Close

```

- *Appropriate selector*:
  - Selektoren sind “invers” zu den Konstruktoren, zu denen sie gehören.
  - Also darf jeder Ausdruck der Form  $tl(n :: 1)$  durch  $n$  ersetzt werden.

Goal

```

if(j = hd(k),
  tl(k) \ i,
  if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ j) \ i)))
= (k \ i) \ j

```

Show all braces      Up    Down

Close

- Genauso wie auf der linken Seite der Gleichung ...

- |                          |                           |
|--------------------------|---------------------------|
| - Execute procedure call | - Execute procedure call  |
| - Negative Hypothesis    | - Negative structure test |
| - Keep else-part         | - Keep else-part          |
| - Distribute if-argument | - 4× Appropriate selector |

Goal

```

if(j = hd(k),
  tl(k) \ i,
  if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ j) \ i)))
=
if(i = hd(k),
  tl(k) \ j,
  if(j = hd(k), tl(k) \ i, hd(k) :: ((tl(k) \ i) \ j)))

```

Show all braces      Up    Down

Close

- *Distribute if-argument*:
  - Ausdrücke der Form  $if\{a, b, c\} = if\{d, e, f\}$  dürfen durch  $if\{a, b = if\{d, e, f\}, c = if\{d, e, f\}\}$  ersetzt werden.

Goal

```

if(j = hd(k),
  tl(k) \ i =
  if(i = hd(k),
    tl(k) \ j,
    if(j = hd(k), tl(k) \ i, hd(k) :: ((tl(k) \ i) \ j))),
  if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ j) \ i)) =
  if(i = hd(k),
    tl(k) \ j,
    if(j = hd(k), tl(k) \ i, hd(k) :: ((tl(k) \ i) \ j)))

```

Show all braces      Up    Down

Close

- *Affirmative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks  $a$  im *then*-Teil eines Ausdrucks der Form  $if\{a, b[a], c\}$  darf durch  $true$  ersetzt werden, also  $if\{a, b[a], c\} \rightarrow if\{a, b[true], c\}$ .
  - In diesem Fall wird damit
    - \*  $if\{j = hd(k), tl(k) \ i = if\{\dots j = hd(k) \dots\}, \dots\}$  durch
    - \*  $if\{j = hd(k), tl(k) \ i = if\{\dots true \dots\}, \dots\}$  ersetzt.

Goal

```

if(j = hd(k),
  tl(k) \ i
  =
  if(i = hd(k), tl(k) \ j, if(true, tl(k) \ i, hd(k) :: ((tl(k) \ i) \ j))),
  if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ j) \ i)) =
  if(i = hd(k),
    tl(k) \ j,
    if(j = hd(k), tl(k) \ i, hd(k) :: ((tl(k) \ i) \ j)))

```

Show all braces      Up    Down

Close

- *Keep then-part*:
  - Semantik von  $if$  –  $if\{true, X, Y\}$  darf durch  $X$  ersetzt werden.

```

Evaluation Viewer "Distribute if argument" ((k \ j) \ i = (k \ i) \ j)
Goal
if(j = hd(k),
  t1(k) \ i = if(i = hd(k), t1(k) \ j, t1(k) \ i),
  if(i = hd(k), t1(k) \ j, hd(k) :: ((t1(k) \ j) \ i)) =
  if(i = hd(k),
    t1(k) \ j,
    if(j = hd(k), t1(k) \ i, hd(k) :: ((t1(k) \ i) \ j))))
 Show all braces
Up Down
Close

```

- **Distribute if-argument:**
  - Ausdrücke der Form  $a = \text{if}\{b, c, d\}$  dürfen durch  $\text{if}\{b, a = c, a = d\}$  ersetzt werden.

```

Evaluation Viewer "Reflexivity" ((k \ j) \ i = (k \ i) \ j)
Goal
if(j = hd(k),
  if(i = hd(k), t1(k) \ i = t1(k) \ j, t1(k) \ i = t1(k) \ i),
  if(i = hd(k), t1(k) \ j, hd(k) :: ((t1(k) \ j) \ i)) =
  if(i = hd(k),
    t1(k) \ j,
    if(j = hd(k), t1(k) \ i, hd(k) :: ((t1(k) \ i) \ j))))
 Show all braces
Up Down
Close

```

- **Reflexivity:**
  - Gleichheit ist *reflexiv*.
  - Damit darf jeder Ausdruck der Form  $t = t$  durch *true* ersetzt werden.

```

Evaluation Viewer "Negative hypothesis" ((k \ j) \ i = (k \ i) \ j)
Goal
if(j = hd(k),
  if(i = hd(k), t1(k) \ i = t1(k) \ j, true),
  if(i = hd(k), t1(k) \ j, hd(k) :: ((t1(k) \ j) \ i)) =
  if(i = hd(k),
    t1(k) \ j,
    if(j = hd(k), t1(k) \ i, hd(k) :: ((t1(k) \ i) \ j))))
 Show all braces
Up Down
Close

```

- **Negative hypothesis:**
  - Jedes Vorkommen eines Ausdrucks  $a$  im *else*-Teil eines Ausdrucks der Form  $\text{if}\{a, b, c[a]\}$  darf durch *false* ersetzt werden, also  $\text{if}\{a, b, c[a]\} \rightarrow \text{if}\{a, b, c[\text{false}]\}$ .
  - In diesem Fall wird damit
    - \*  $\text{if}\{j = \text{hd}(k), \dots, \text{if}\{\dots j = \text{hd}(k) \dots\}\}$  durch
    - \*  $\text{if}\{j = \text{hd}(k), \dots, \text{if}\{\dots \text{false} \dots\}\}$  ersetzt.

```

Evaluation Viewer "Keep else-part" ((k \ j) \ i = (k \ i) \ j)
Goal
if(j = hd(k),
  if(i = hd(k), t1(k) \ i = t1(k) \ j, true),
  if(i = hd(k), t1(k) \ j, hd(k) :: ((t1(k) \ j) \ i)) =
  if(i = hd(k),
    t1(k) \ j,
    if(false, t1(k) \ i, hd(k) :: ((t1(k) \ i) \ j))))
 Show all braces
Up Down
Close

```

- **Keep else-part:**
  - Semantik von *if* –  $\text{if}\{\text{false}, X, Y\}$  darf durch  $Y$  ersetzt werden.

```

Evaluation Viewer "Distribute if argument" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ j) \ i)))
= if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ i) \ j)))
  
```

- **Distribute if-argument:**
  - Ausdrücke der Form  $if\{a, b, c\} = if\{d, e, f\}$  dürfen durch  $if\{a, b = if\{d, e, f\}, c = if\{d, e, f\}\}$  ersetzt werden.

```

Evaluation Viewer "Affirmative hypothesis" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k),
    tl(k) \ j = if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ i) \ j)),
    hd(k) :: ((tl(k) \ j) \ i)
    = if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ i) \ j))))
  
```

- **Affirmative hypothesis:**
  - Jedes Vorkommen eines Ausdrucks  $a$  im *then*-Teil eines Ausdrucks der Form  $if\{a, b[a], c\}$  darf durch  $true$  ersetzt werden, also  $if\{a, b[a], c\} \rightarrow if\{a, b[true], c\}$ .

```

Evaluation Viewer "Keep then-part" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k),
    tl(k) \ j = if(true, tl(k) \ j, hd(k) :: ((tl(k) \ i) \ j)),
    hd(k) :: ((tl(k) \ j) \ i)
    = if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ i) \ j))))
  
```

- **Keep then-part:**
  - Semantik von  $if - if\{true, X, Y\}$  darf durch  $X$  ersetzt werden.

```

Evaluation Viewer "Reflexivity" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k),
    tl(k) \ j = tl(k) \ j,
    hd(k) :: ((tl(k) \ j) \ i)
    = if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ i) \ j))))
  
```

- **Reflexivity:**
  - Gleichheit ist *reflexiv*.
  - Damit darf jeder Ausdruck der Form  $t = t$  durch  $true$  ersetzt werden.



```

Evaluation Viewer "Negative hypothesis" ((k \ j) \ i = (k \ i) \ j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k),
    true,
    hd(k) :: ((tl(k) \ j) \ i)
  ) = if(i = hd(k), tl(k) \ j, hd(k) :: ((tl(k) \ i) \ j)))
  
```

- **Negative hypothesis:**
  - Jedes Vorkommen eines Ausdrucks  $a$  im *else*-Teil eines Ausdrucks der Form  $if\{a, b, c[a]\}$  darf durch  $false$  ersetzt werden, also  $if\{a, b, c[a]\} \rightarrow if\{a, b, c[false]\}$ .

```

Evaluation Viewer "Keep else-part" ((k \ j) \ i = (k \ i) \ j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k),
    true,
    hd(k) :: ((tl(k) \ j) \ i)
  ) = if(false, tl(k) \ j, hd(k) :: ((tl(k) \ i) \ j)))
  
```

- **Keep else-part:**
  - Semantik von  $if$  –  $if\{false, X, Y\}$  darf durch  $Y$  ersetzt werden.

```

Evaluation Viewer "Injectivity" ((k \ j) \ i = (k \ i) \ j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k),
    true,
    hd(k) :: ((tl(k) \ j) \ i) = hd(k) :: ((tl(k) \ i) \ j)
  )
  
```

- **Injectivity:**
  - Für eine 2-stellige *injektive* Funktion  $g$  gilt  $g(a, b) = g(c, d)$  gdw.  $a = c \wedge b = d$ , also darf  $g(a, b) = g(c, d)$  durch  $if\{a = c, b = d, false\}$  ersetzt werden.
  - Da Konstruktoren *injektiv* sind, wird hier
    - \*  $hd(k) :: ((tl(k) \ j) \ i) = hd(k) :: ((tl(k) \ i) \ j)$  ersetzt durch
    - \*  $if\{hd(k) = hd(k), (tl(k) \ j) \ i = (tl(k) \ i) \ j, false\}$ .

```

Evaluation Viewer "Reflexivity" ((k \ j) \ i = (k \ i) \ j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k),
    true,
    if(hd(k) = hd(k), (tl(k) \ j) \ i = (tl(k) \ i) \ j, false)
  )
  
```

- **Reflexivity:**
  - Gleichheit ist *reflexiv*.
  - Damit darf jeder Ausdruck der Form  $t = t$  durch  $true$  ersetzt werden.

```

Evaluation Viewer "Keep then-part" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k), true, if(true, (tl(k) \ j) \ i = (tl(k) \ i) \ j, false)))
  
```

- **Keep then-part:**
  - Semantik von `if` – `if{true, X, Y}` darf durch `X` ersetzt werden.

```

Evaluation Viewer "Affirmative assumption using IH-1: \forall j', i' : @ITEM ((k \j) \j' \i' = (tl(k) \j) \j') ((k \i) \i) \j'"
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k), true, (tl(k) \ j) \ i = (tl(k) \ i) \ j))
  
```

Hypotheses InductionHypotheses Clauses

```

IH-1: \forall j', i' : @ITEM
(tl(k) \ j') \ i' = (tl(k) \ i') \ j'
  
```

- **Affirmative assumption using ... :**
  - Anwendung eines Lemmas oder einer *Induktionshypothese*, jede *Instanz* davon darf durch `true` ersetzt werden.
  - Die Induktionshypothese lautet hier:  $(tl(k) \setminus j') \setminus i' = (tl(k) \setminus i') \setminus j'$
  - *Bedeutung:* Die Variablen mit “'” sind *allquantifiziert*, dürfen also durch *beliebige Terme* ersetzt werden.
  - Variable ohne “'” sind *Induktionsvariable*, dürfen also *nicht ersetzt* werden.

```

Evaluation Viewer "Skip if condition" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k), tl(k) \ i = tl(k) \ j, true),
  if(i = hd(k), true, true))
  
```

- **Skip if-condition:**
  - Irrelevante Fallunterscheidung, denn *then-* und *else-*Teil sind identisch.
  - Damit darf jeder Ausdruck der Form `if{a, b, b}` durch `b` ersetzt werden.

```

Evaluation Viewer "Affirmative left non-boolean functionality" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k), if(i = hd(k), tl(k) \ i = tl(k) \ j, true), true)
  
```

- **Affirmative left non-boolean functionality:**
  - $g(a, b) = g(c, d)$  darf durch `true` ersetzt werden, falls  $a = c \wedge b = d$ , also darf  $g(a, b) = g(c, d)$  durch `if{if{a = c, b = d, false}, true, g(a, b) = g(c, d)}` ersetzt werden.
  - Konkret wird hier
    - \*  $tl(k) \setminus i = tl(k) \setminus j$  ersetzt durch
    - \* `if{if{i = j, tl(k) = tl(k), false}, true, tl(k) \ i = tl(k) \ j}`.

```

Evaluation Viewer "Affirmative assumption using = is transitive" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k),
    if(i = j, tl(k) = tl(k), false), true, tl(k) \ i = tl(k) \ j),
    true),
  true)
  
```

- *Affirmative assumption using ...* :
  - Anwendung eines Lemmas oder einer Induktionshypothese.
  - Hier wird das Transitivitätsgesetz für = verwendet, d.h.  $\forall x, y, z: @T \text{ if } \{x = y, \text{ if } \{y = z, x = z, \text{ true}\}, \text{ true}\}$ .
  - Damit darf
    - \*  $i = j$  ersetzt werden durch
    - \*  $\text{if}\{i = j, \text{ true}, \text{if}\{j = \text{hd}(k), \text{hd}(k) = i, \text{ false}\}\}$ .

```

Evaluation Viewer "Affirmative hypothesis" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k),
    if(
      if(i = j, true, if(j = hd(k), hd(k) = i, false)),
      tl(k) = tl(k),
      false),
    true,
    tl(k) \ i = tl(k) \ j),
  true)
  
```

- *Affirmative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks  $a$  im *then*-Teil eines Ausdrucks der Form  $\text{if}\{a, b[a], c\}$  darf durch  $\text{true}$  ersetzt werden, also  $\text{if}\{a, b[a], c\} \rightarrow \text{if}\{a, b[\text{true}], c\}$ .

```

Evaluation Viewer "Keep then-part" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k),
    if(
      if(i = j, true, if(true, hd(k) = i, false)), tl(k) = tl(k), false),
      true,
      tl(k) \ i = tl(k) \ j),
    true),
  true)
  
```

- *Keep then-part*:
  - Semantik von  $\text{if} - \text{if}\{\text{true}, X, Y\}$  darf durch  $X$  ersetzt werden.

```

Evaluation Viewer "Affirmative hypothesis" ((k \j) \i = (k \i) \j)
Goal
if(j = hd(k),
  if(i = hd(k),
    if(if(i = j, true, hd(k) = i), tl(k) = tl(k), false),
      true,
      tl(k) \ i = tl(k) \ j),
    true),
  true)
  
```

- *Affirmative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks  $a$  im *then*-Teil eines Ausdrucks der Form  $\text{if}\{a, b[a], c\}$  darf durch  $\text{true}$  ersetzt werden, also  $\text{if}\{a, b[a], c\} \rightarrow \text{if}\{a, b[\text{true}], c\}$ .
  - Wir dürfen  $\text{hd}(k) = i$  auch unter der Bedingung  $i = \text{hd}(k)$  durch  $\text{true}$  ersetzen, denn = ist *symmetrisch* !

```

Evaluation Viewer "Skip if condition" ((k \i) \i = (k \i) \i)
Goal
if(j = hd(k),
  if(i = hd(k),
    if(if(i = j, true, true), tl(k) = tl(k), false),
      true,
      tl(k) \ i = tl(k) \ j),
    true),
  true)
  
```

- *Skip if-condition:*
  - Irrelevante Fallunterscheidung, denn *then-* und *else-*Teil sind identisch.
  - Damit darf jeder Ausdruck der Form  $if\{a, b, b\}$  durch  $b$  ersetzt werden.

```

Evaluation Viewer "Keep then-part" ((k \i) \i = (k \i) \i)
Goal
if(j = hd(k),
  if(i = hd(k),
    if(true, tl(k) = tl(k), false), true, tl(k) \ i = tl(k) \ j),
    true),
  true)
  
```

- *Keep then-part:*
  - Semantik von  $if$  –  $if\{true, X, Y\}$  darf durch  $X$  ersetzt werden.

```

Evaluation Viewer "Reflexivity" ((k \i) \i = (k \i) \i)
Goal
if(j = hd(k),
  if(i = hd(k), if(tl(k) = tl(k), true, tl(k) \ i = tl(k) \ j), true),
  true)
  
```

- *Reflexivity:*
  - Gleichheit ist *reflexiv*.
  - Damit darf jeder Ausdruck der Form  $t = t$  durch  $true$  ersetzt werden.

```

Evaluation Viewer "Keep then-part" ((k \i) \i = (k \i) \i)
Goal
if(j = hd(k),
  if(i = hd(k), if(true, true, tl(k) \ i = tl(k) \ j), true),
  true)
  
```

- *Keep then-part:*
  - Semantik von  $if$  –  $if\{true, X, Y\}$  darf durch  $X$  ersetzt werden.

Goal

```
if(j = hd(k), if(i = hd(k), true, true), true)
```

Show all braces    Up    Down

Close

- *Skip if-condition:*
  - Irrelevante Fallunterscheidung, denn *then-* und *else-*Teil sind identisch.
  - Damit darf jeder Ausdruck der Form  $if\{a, b, b\}$  durch  $b$  ersetzt werden.

Goal

```
if(j = hd(k), true, true)
```

Show all braces    Up    Down

Close

- *Skip if-condition:*
  - Irrelevante Fallunterscheidung, denn *then-* und *else-*Teil sind identisch.
  - Damit darf jeder Ausdruck der Form  $if\{a, b, b\}$  durch  $b$  ersetzt werden.

Goal

```
true
```

Show all braces    Up    Down

Close

- *Bewiesen, fertig !*