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 (= boolsche 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 (`if{true, x, y}` $\vdash$ `x`, vgl. Berechnungsregel (9) in **Kapitel 5**).
  - *Beispiel*: Auswertungsregel **Appropriate selector** – Anwendung eines Axioms einer Datentypdefinition (`pred(succ(t))` $\vdash$ `t`, vgl. Berechnungsregel (5) in **Kapitel 5**).
- Des weiteren werden (aus Axiomen) *abgeleitete* Lemmata verwendet
  - *Beispiel*: Auswertungsregel **Skip condition** – `if{a, b, b}` $\vdash$ `b`.
  - *Beispiel*: Auswertungsregel **Skip alternatives** – `if{a, true, false}` $\vdash$ `a`.

- "*Evaluator*": die Arbeitsweise orientiert sich an der Arbeitsweise des Interpreters $eval_P$ (= 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 $\langle H, IH \Vdash goal \rangle$ 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 $\langle H, IH \Vdash goal \rangle$ trivialerweise bewiesen)
     - Die Anwendbarkeit von *Inconsistency* wird vom System automatisch überprüft, deshalb steht diese Regel in *Proof\Proof Rules* nicht zur Verfügung

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

- 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 ?ø(k)
   then ø
   else if i = hd(k)
          then tl(k)
          else hd(k) :: (tl(k) \ i)
         end_if
  end_if
  ```

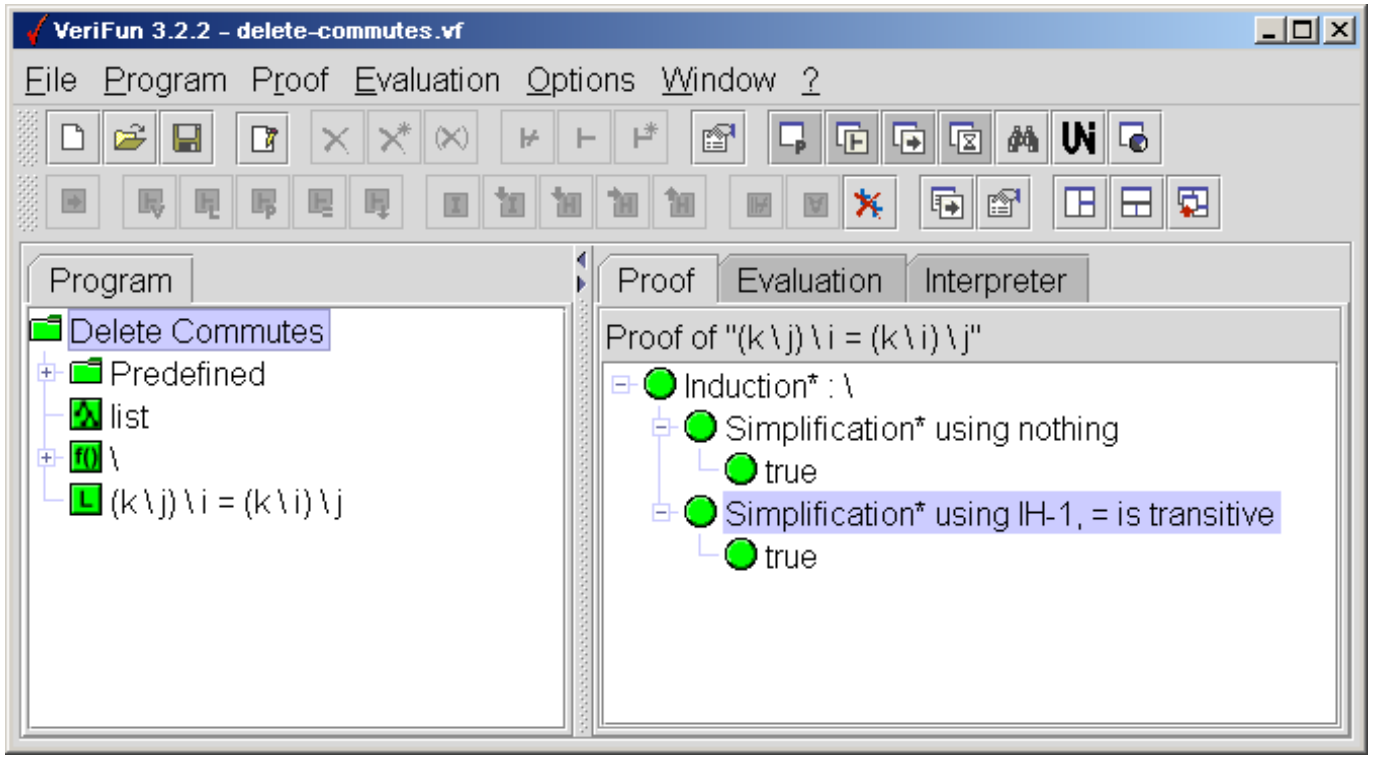
  löscht das erste (von links gelesen) Vorkommen von `i` in der Liste `k`.

Behauptung: Es gilt

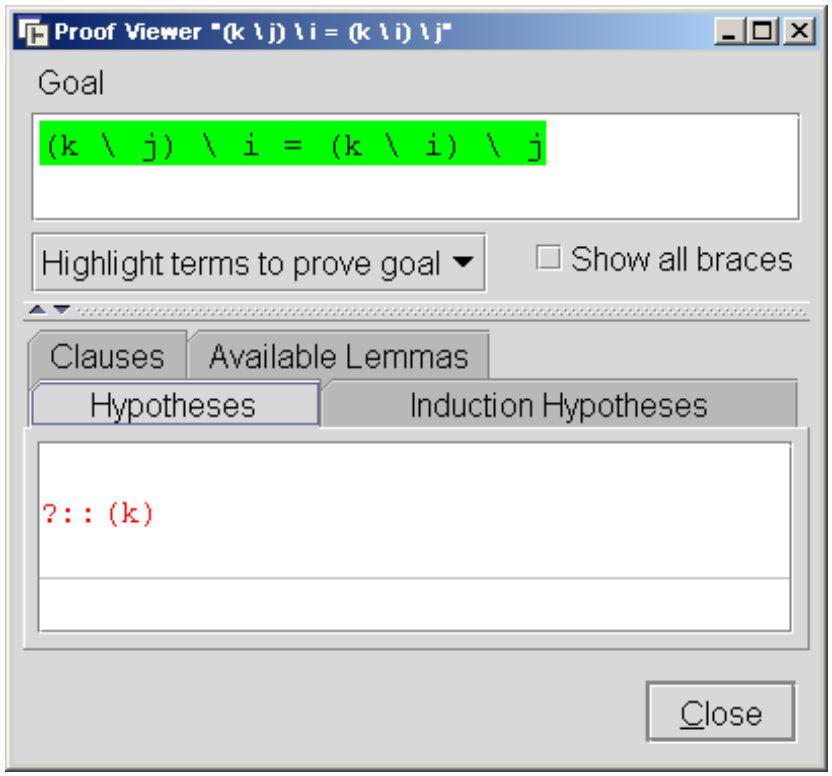
- ```
  lemma (k \ j) \ i = (k \ i) \ j <=
  ∀k : list[@ITEM], i, j : @ITEM (k \ j) \ i = (k \ i) \ j
  ```

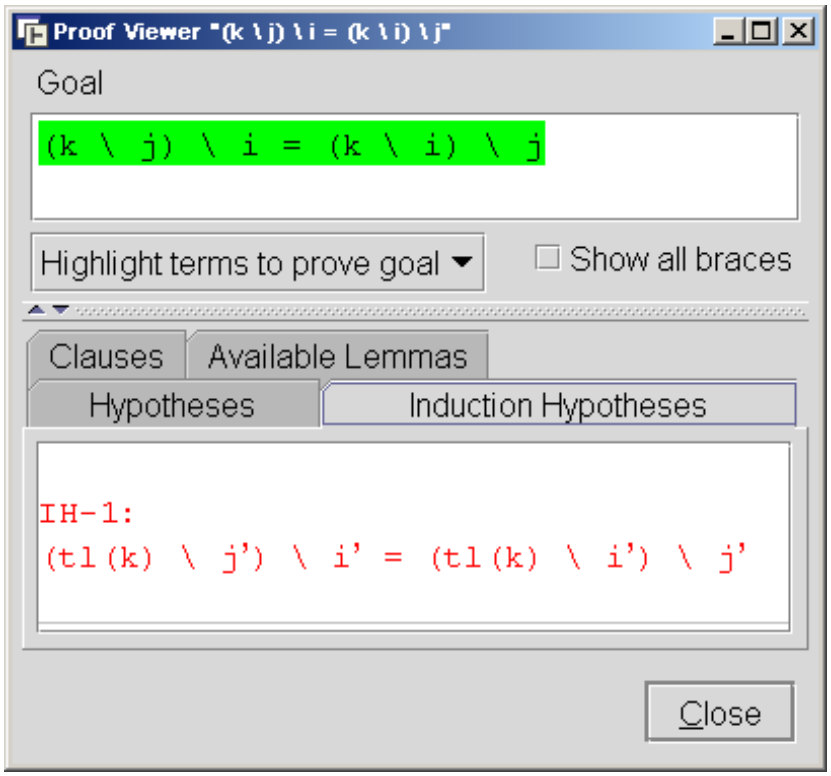
  d.h., die Reihenfolge, in der Listenelemente gelöscht werden, ist irrelevant.

- **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**

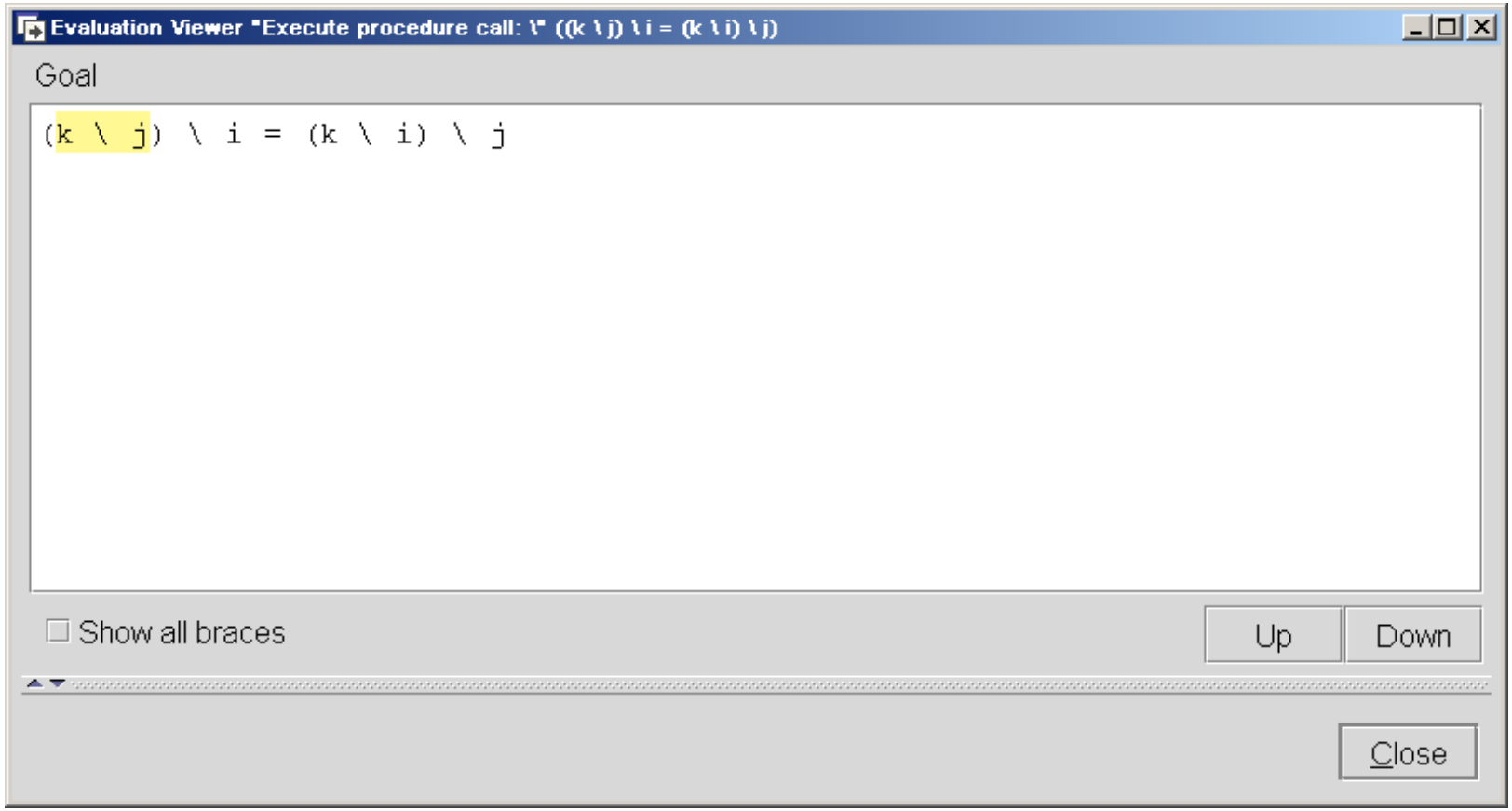




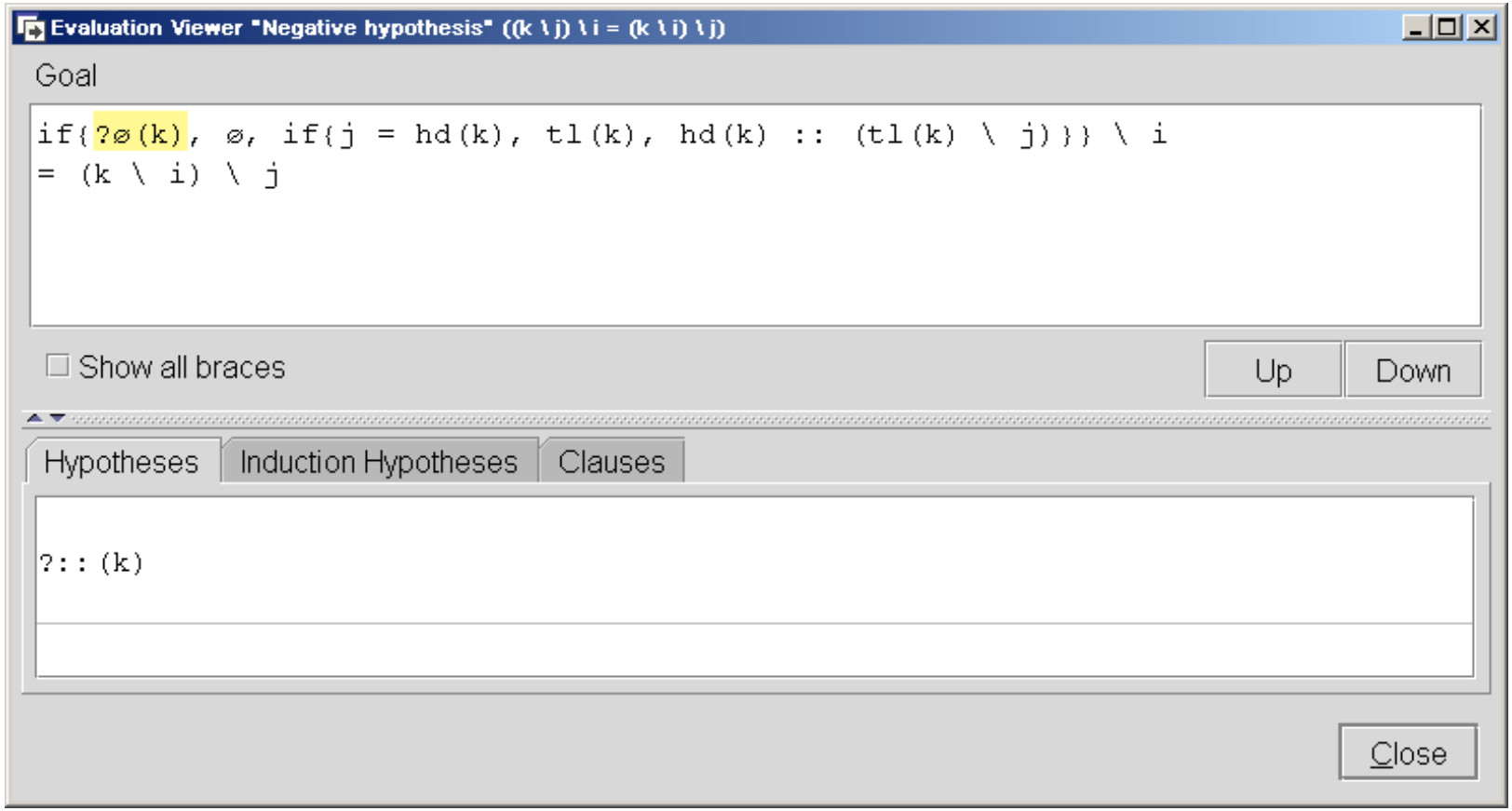
- *Beweisziel* $goal$: $\texttt{(k \ j) \ i} = \texttt{(k \ i) \ j}$
- Strukturelle Induktion über `list[@ITEM]`, also
  - *Hypothese*(n) H: `?::(k)` – wir sind im Schrittfall, d.h. Liste `k` nicht leer
  - *Induktionshypothese*(n) IH:
    $\forall \texttt{i', j' : @ITEM (tl(k) \ j') \ i' = (tl(k) \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

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

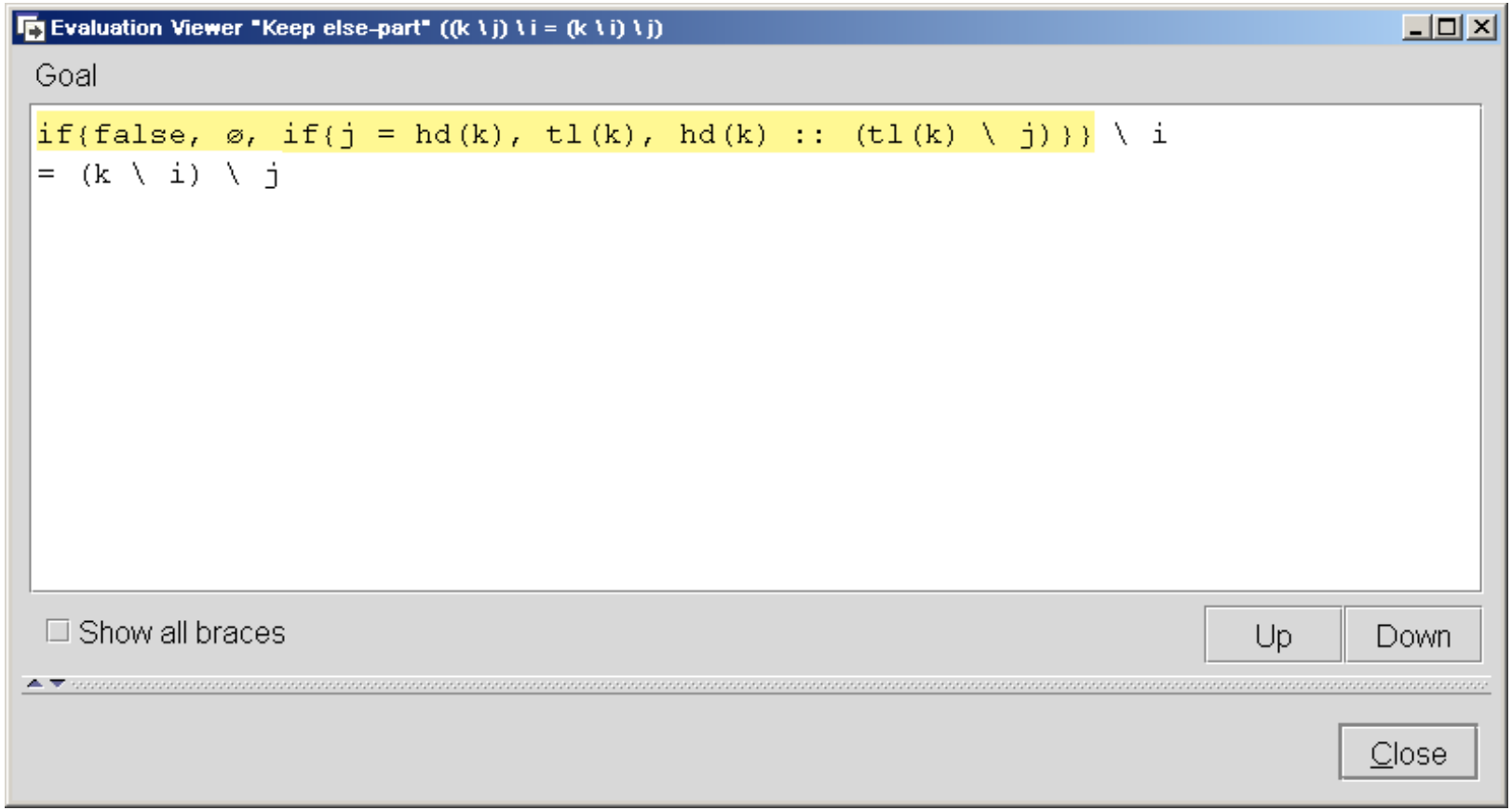
- **Beweis einer Induktionsformel** durch den *Symbolic Evaluator*
  - Beweise (= *symbolische Auswertungen*) werden durch Anwendung sogenannter *Auswertungsregeln* geführt
  - der *Symbolic Evaluator* verwendet die *Axiome*, die aus den Definitionen der *Datentypen*, *Prozeduren* und *Lemmata* gewonnen 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



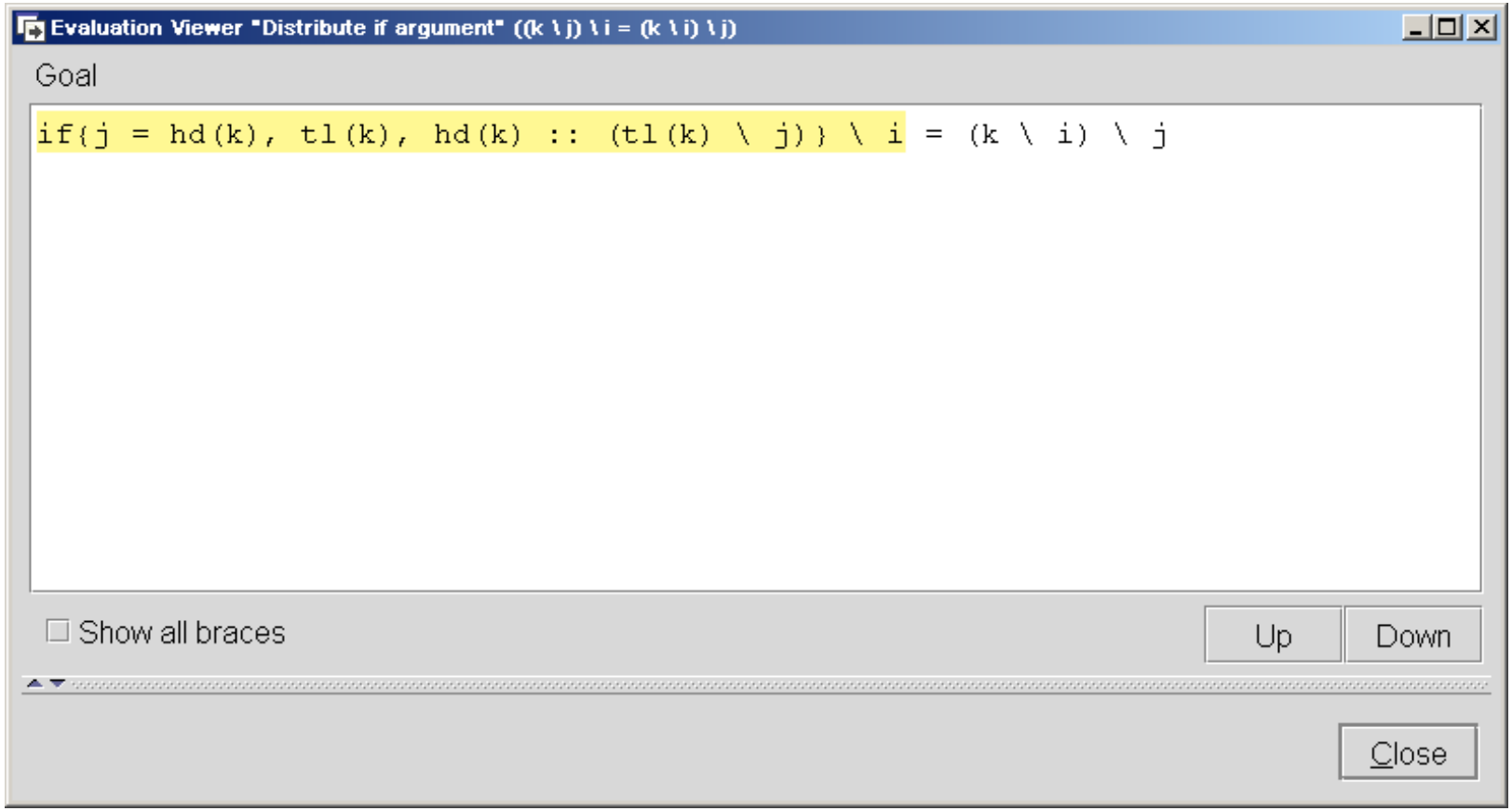
- *Execute procedure call*:
  - Ersetze Prozeduraufruf durch instantiierten (formale Parameter → aktuelle Parameter) Prozedurrumpf.



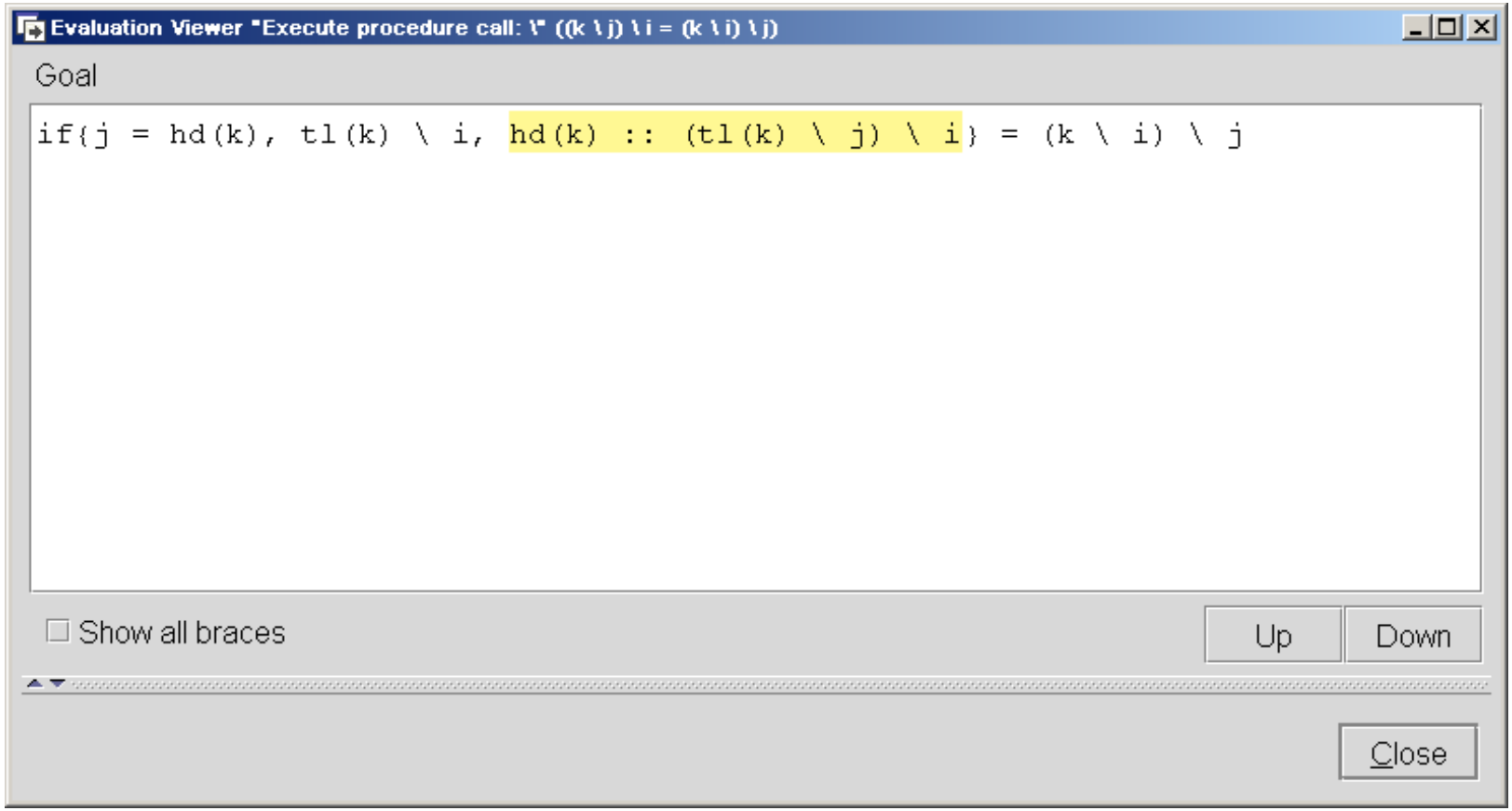
- *Negative Hypothesis*:
  - Wir sind im Induktionsschritt, d.h. es gilt `?::(k)` (in Worten "`k` ist nicht die leere Liste").
  - Also darf `?ø(k)` durch `false` ersetzt werden.



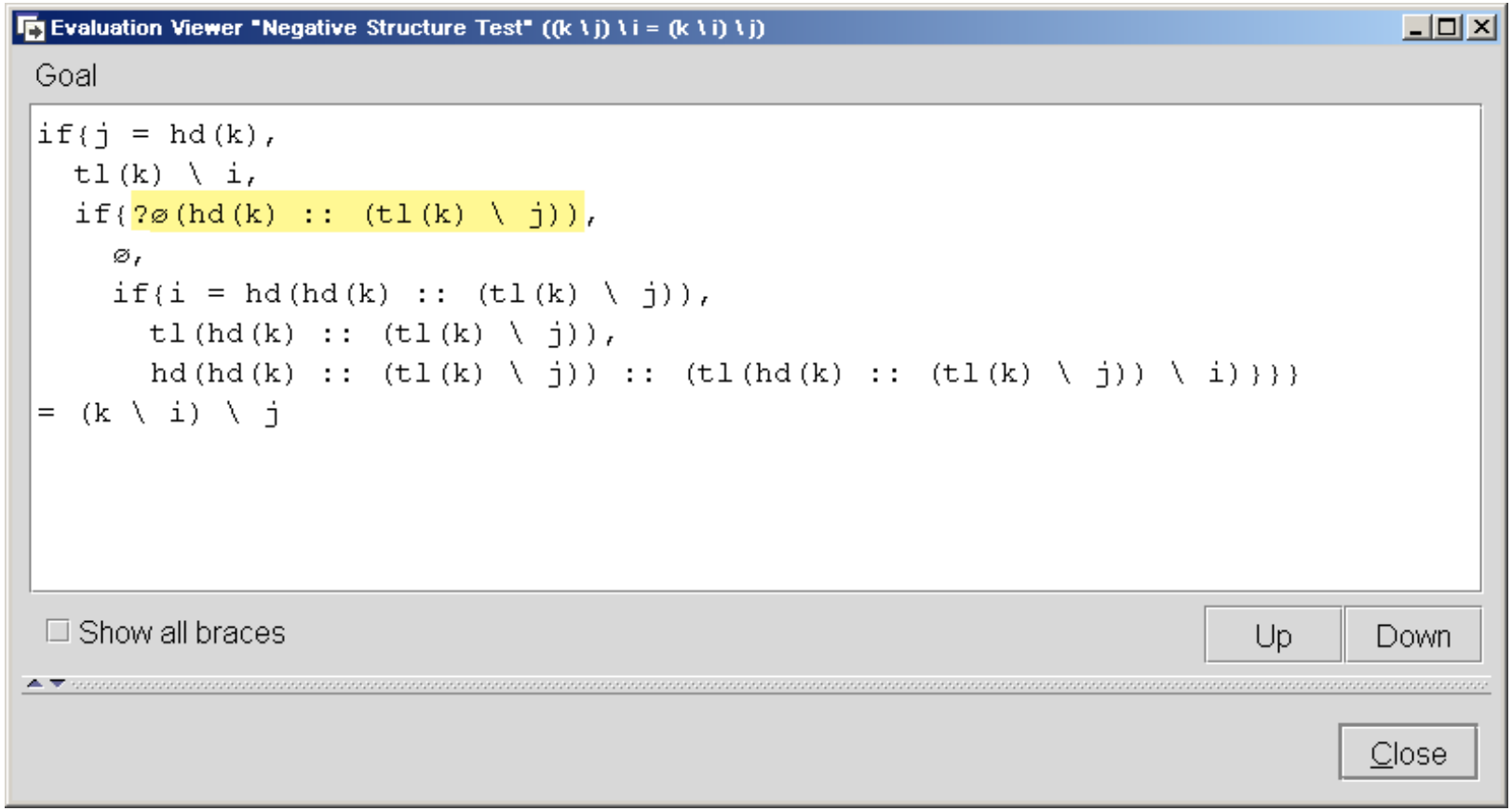
- *Keep else-part*:
  - Semantik von `if` – `if{false,`X`,`Y`}` darf durch Y ersetzt werden.



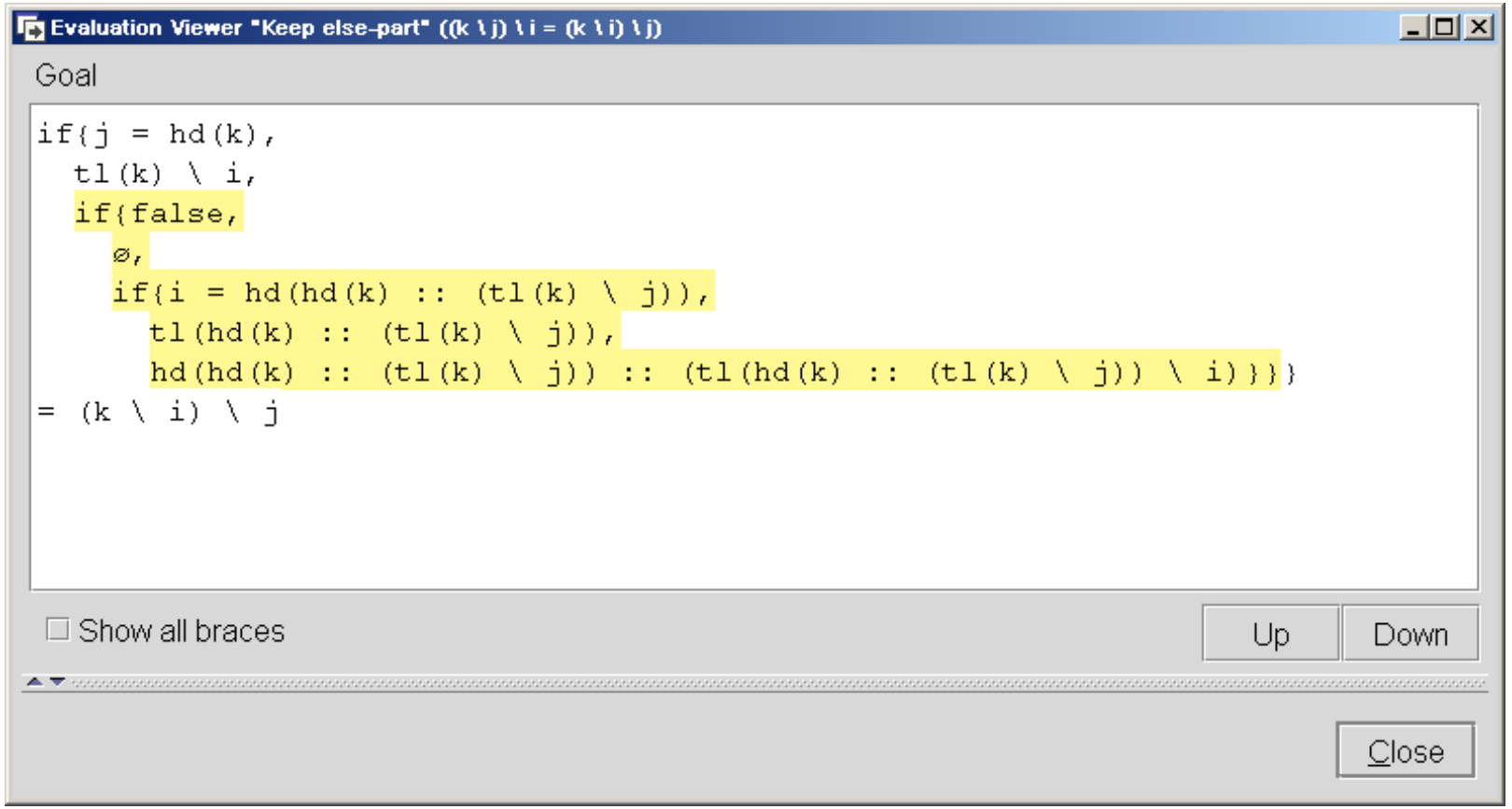
- *Distribute if-argument*:
  - Ausdrücke der Form $g(if\{a, b, c\}, d)$ dürfen durch $if\{a, g(b, d), g(c, d)\}$ ersetzt werden.



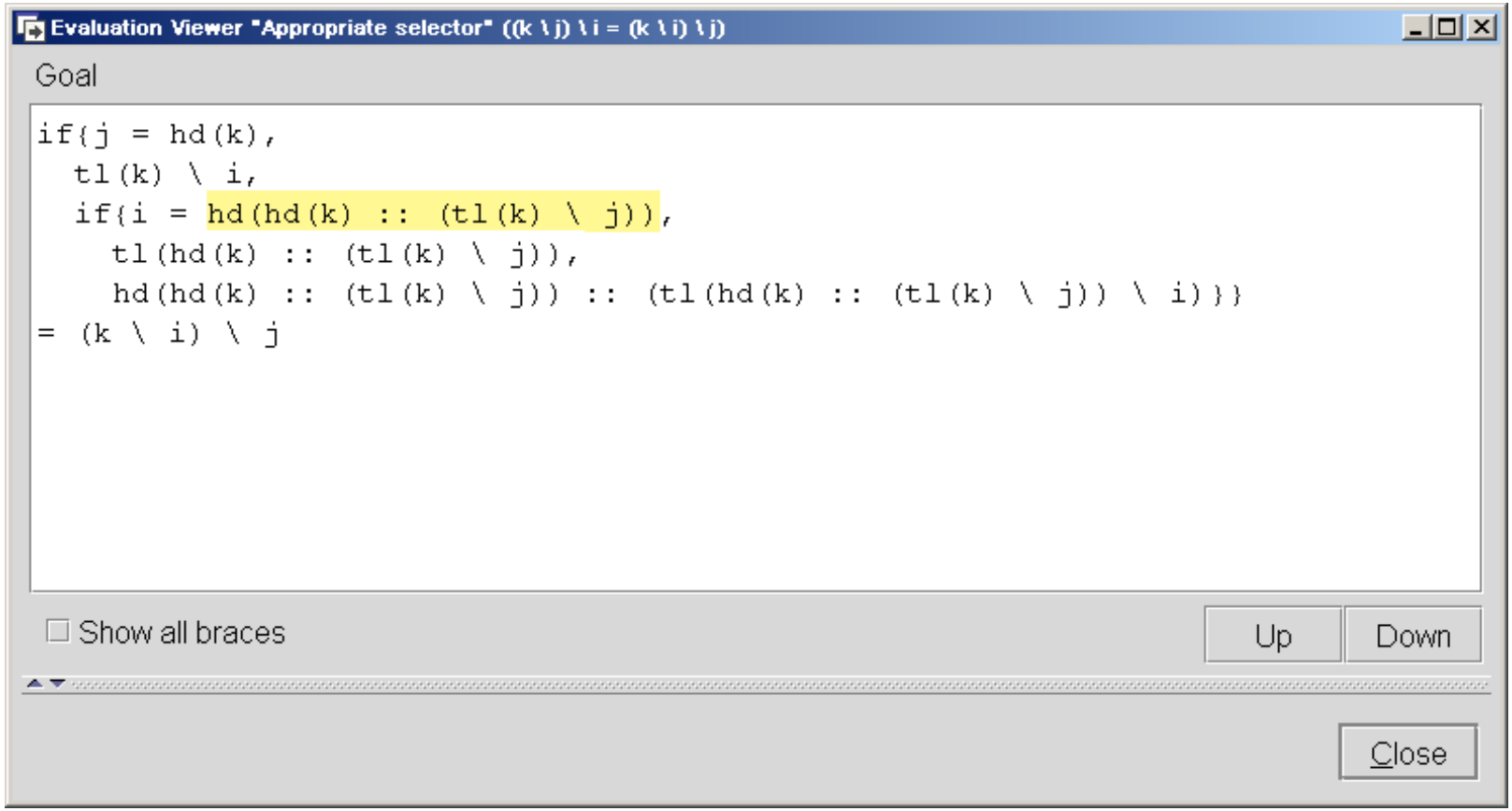
- *Execute procedure call*:
  - Ersetze Prozeduraufruf durch instantiierten (formale Parameter → aktuelle Parameter) Prozedurrumpf.



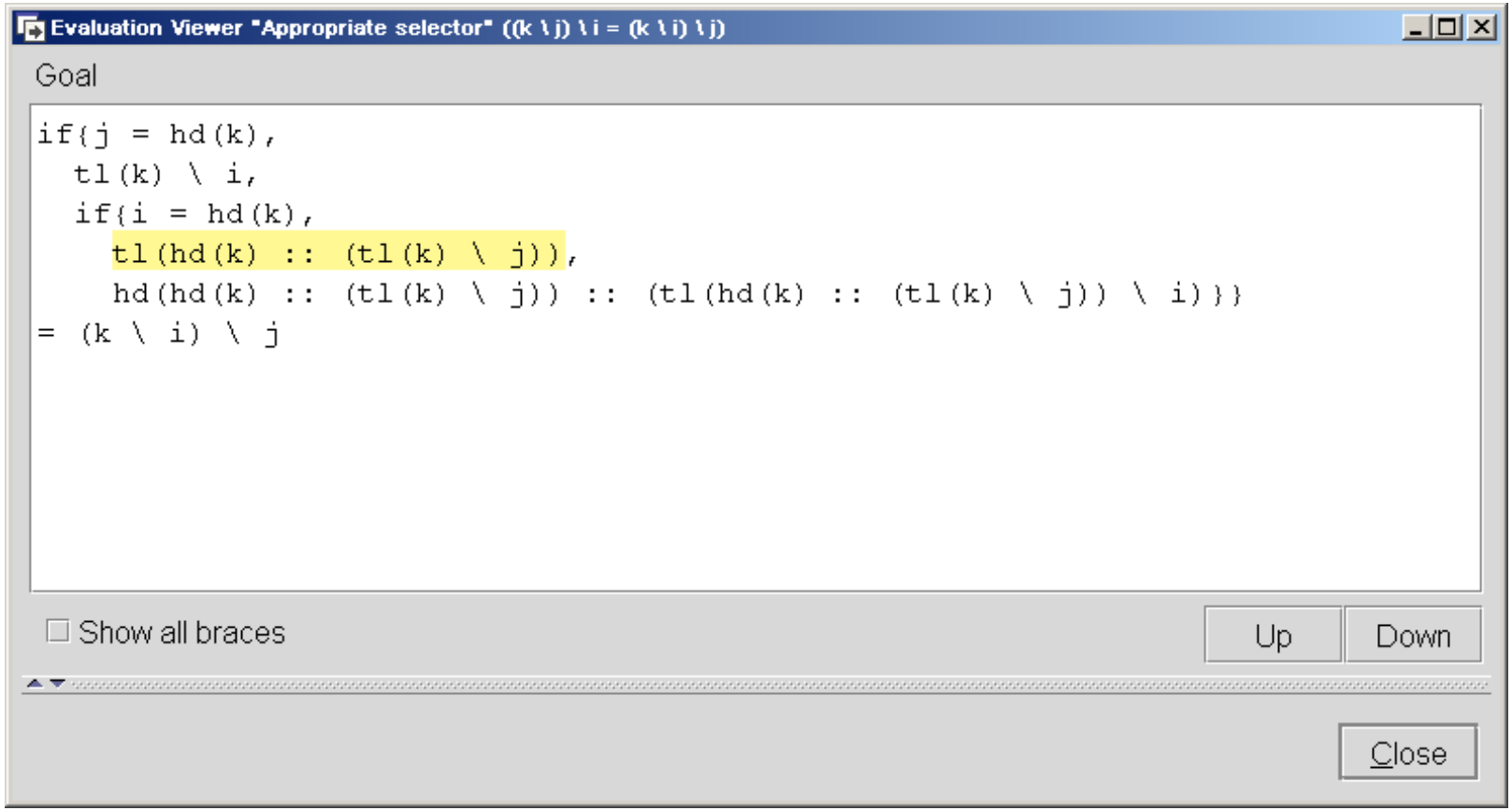
- *Negative structure test*:
  - Keine Liste der Form `n :: l` ist leer.
  - Damit darf `?ø(hd(k) :: ...)` durch `false` ersetzt werden.



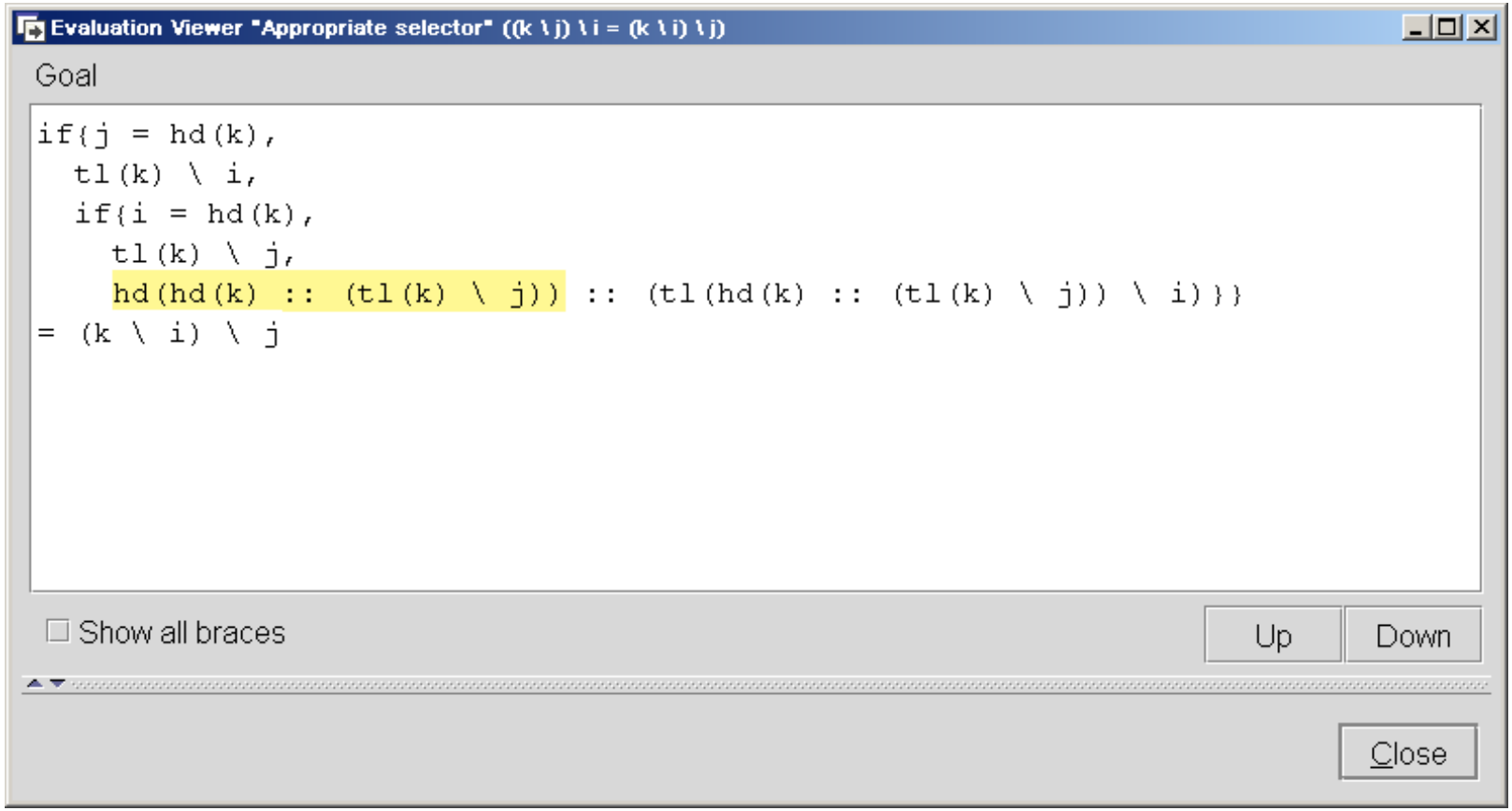
- *Keep else-part*:
  - Semantik von `if` – `if{false,`X`,`Y`}` darf durch Y ersetzt werden.



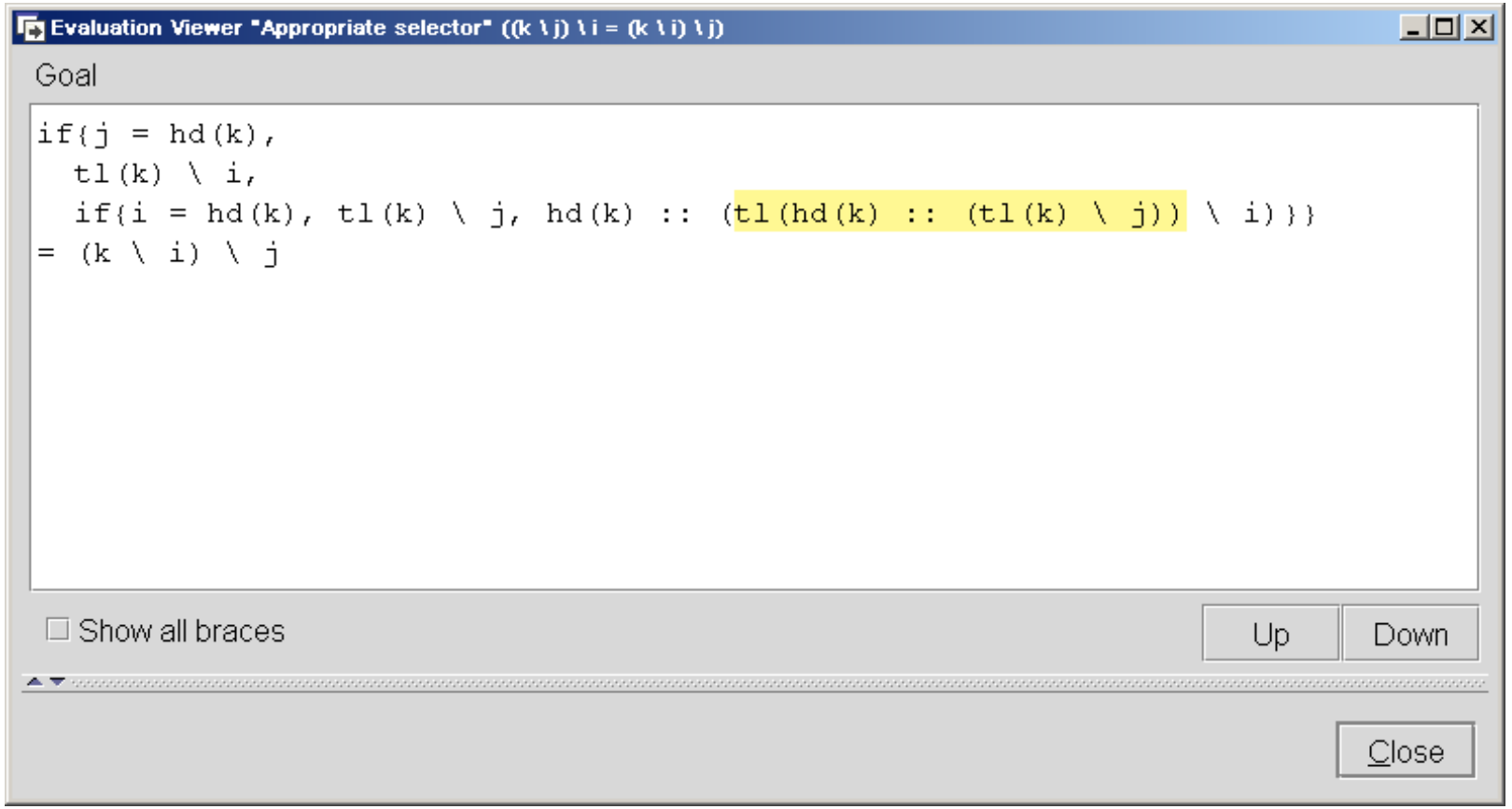
- *Appropriate selector*:
  - Selektoren sind “invers” zu den Konstruktoren, zu denen sie gehören.
  - Also darf jeder Ausdruck der Form `hd(n :: l)` durch `n` ersetzt werden.



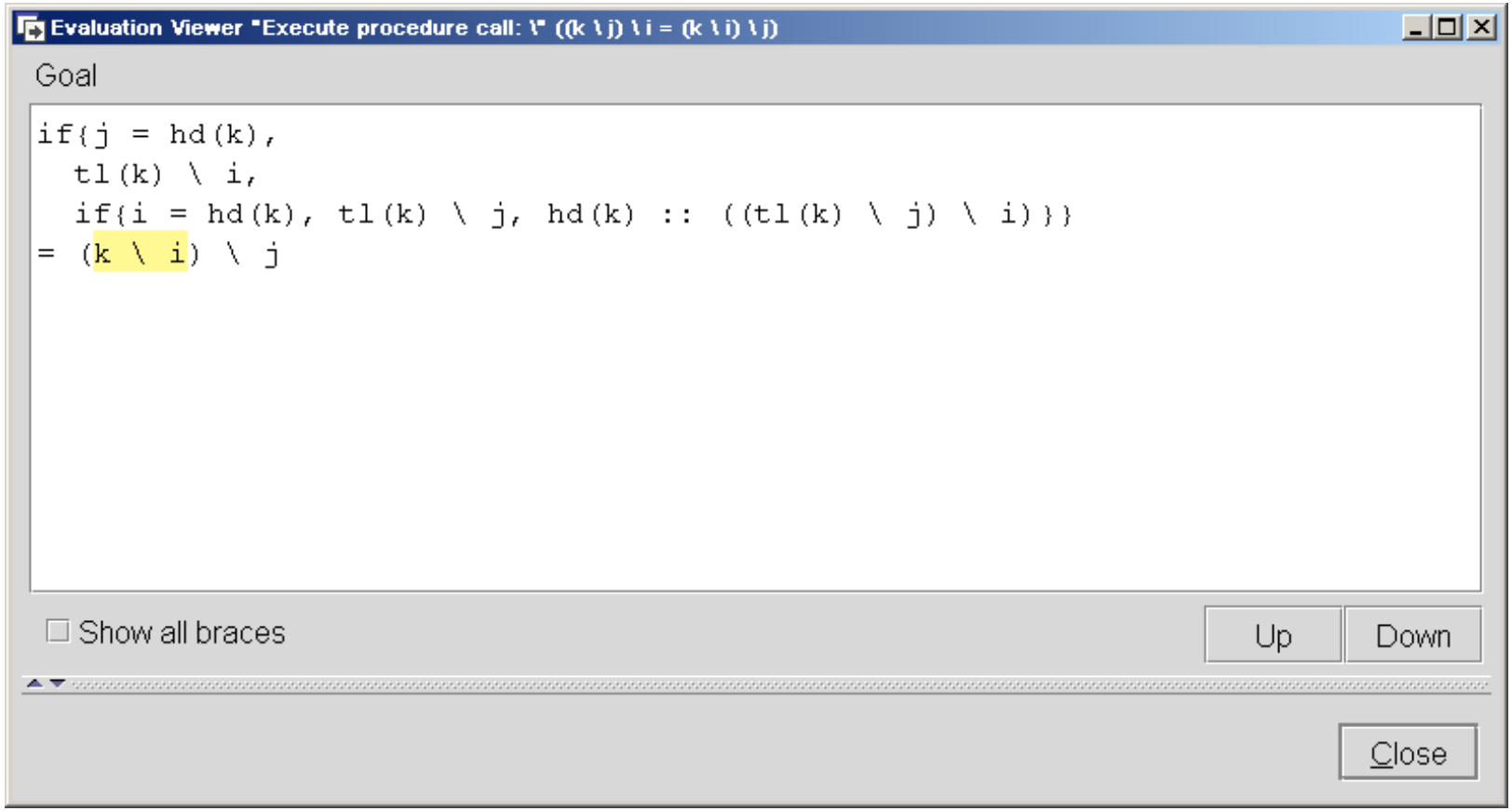
- *Appropriate selector*:
  - Selektoren sind “invers” zu den Konstruktoren, zu denen sie gehören.
  - Also darf jeder Ausdruck der Form `tl(n :: l)` durch `l` ersetzt werden.



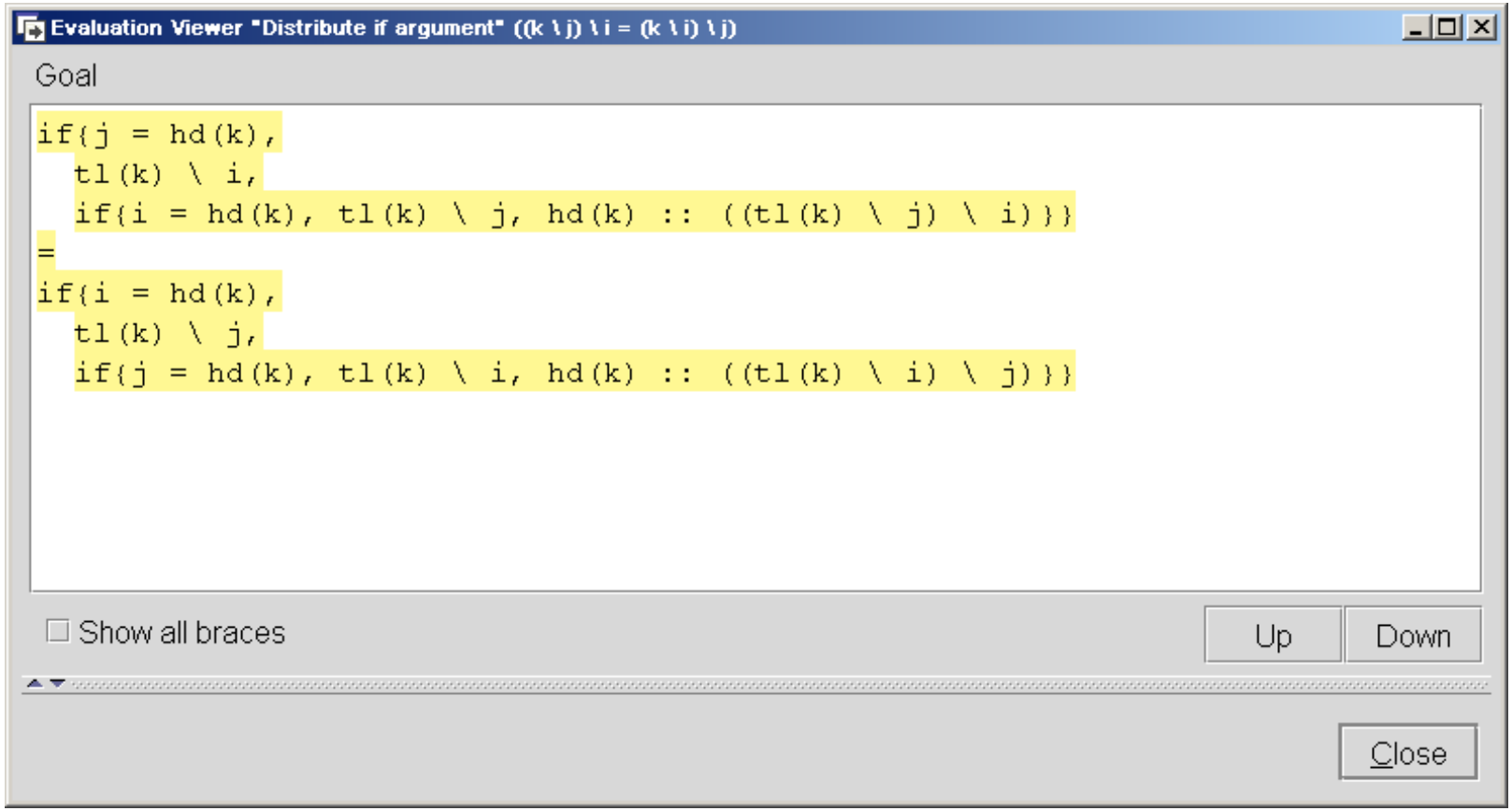
- *Appropriate selector*:
  - Selektoren sind “invers” zu den Konstruktoren, zu denen sie gehören.
  - Also darf jeder Ausdruck der Form `hd(n :: l)` durch `n` ersetzt werden.



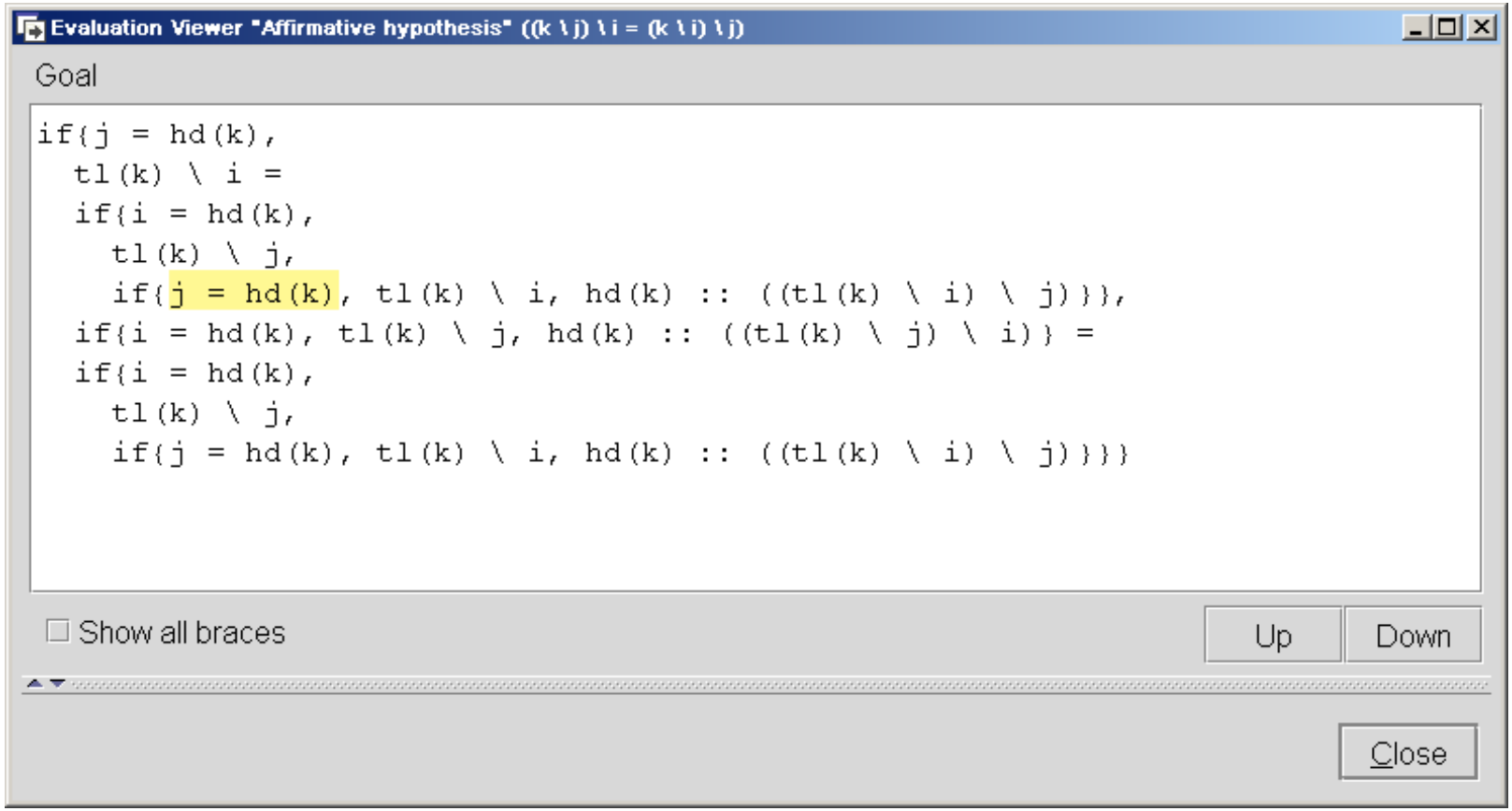
- *Appropriate selector*:
  - Selektoren sind “invers” zu den Konstruktoren, zu denen sie gehören.
  - Also darf jeder Ausdruck der Form `tl(n :: l)` durch `l` ersetzt werden.



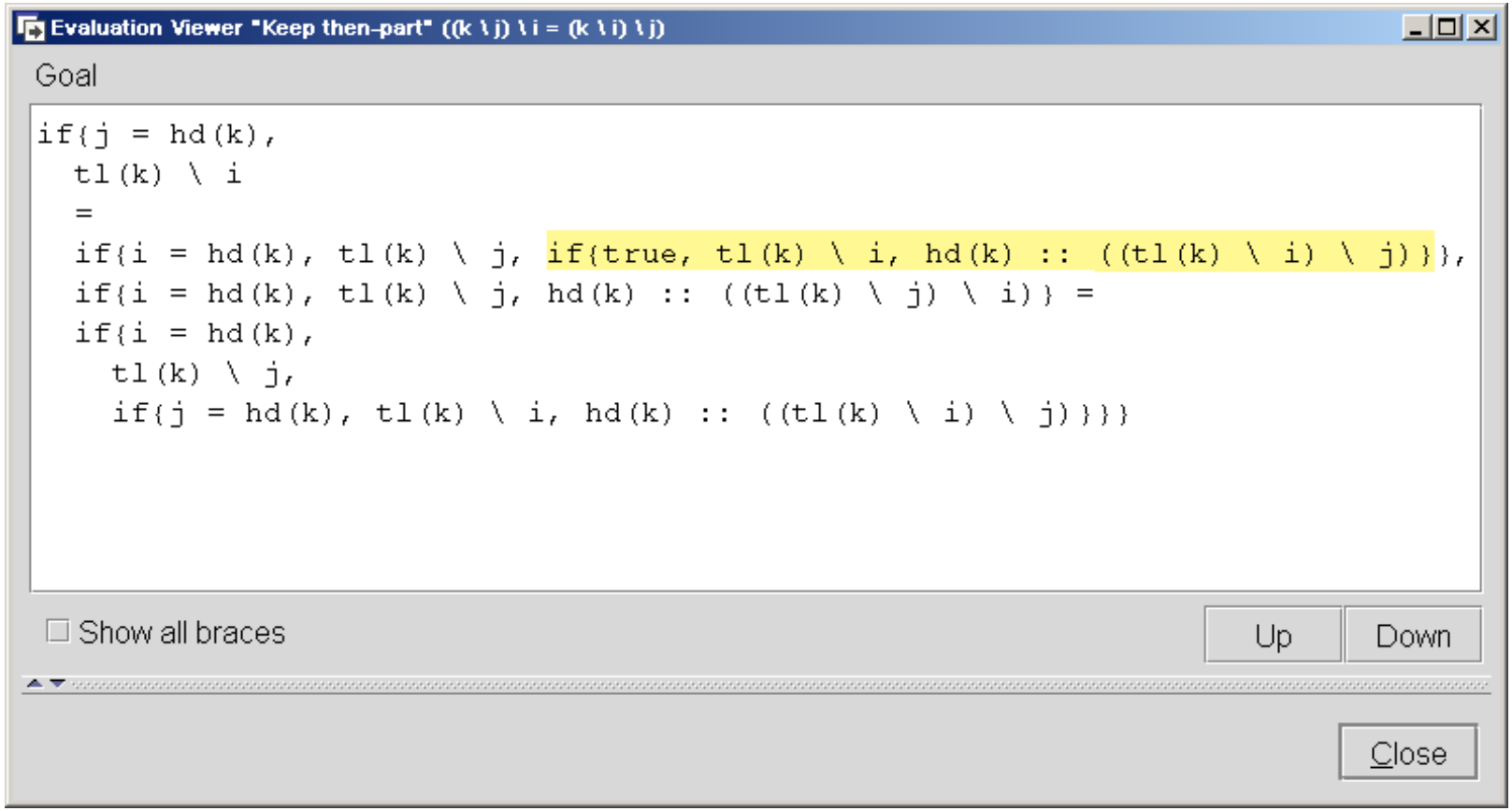
- Genauso wie auf der linken Seite der Gleichung ...
  - *Execute procedure call*
  - *Negative Hypothesis*
  - *Keep else-part*
  - *Distribute if-argument*
  - *Execute procedure call*
  - *Negative structure test*
  - *Keep else-part*
  - $4\times$ *Appropriate selector*



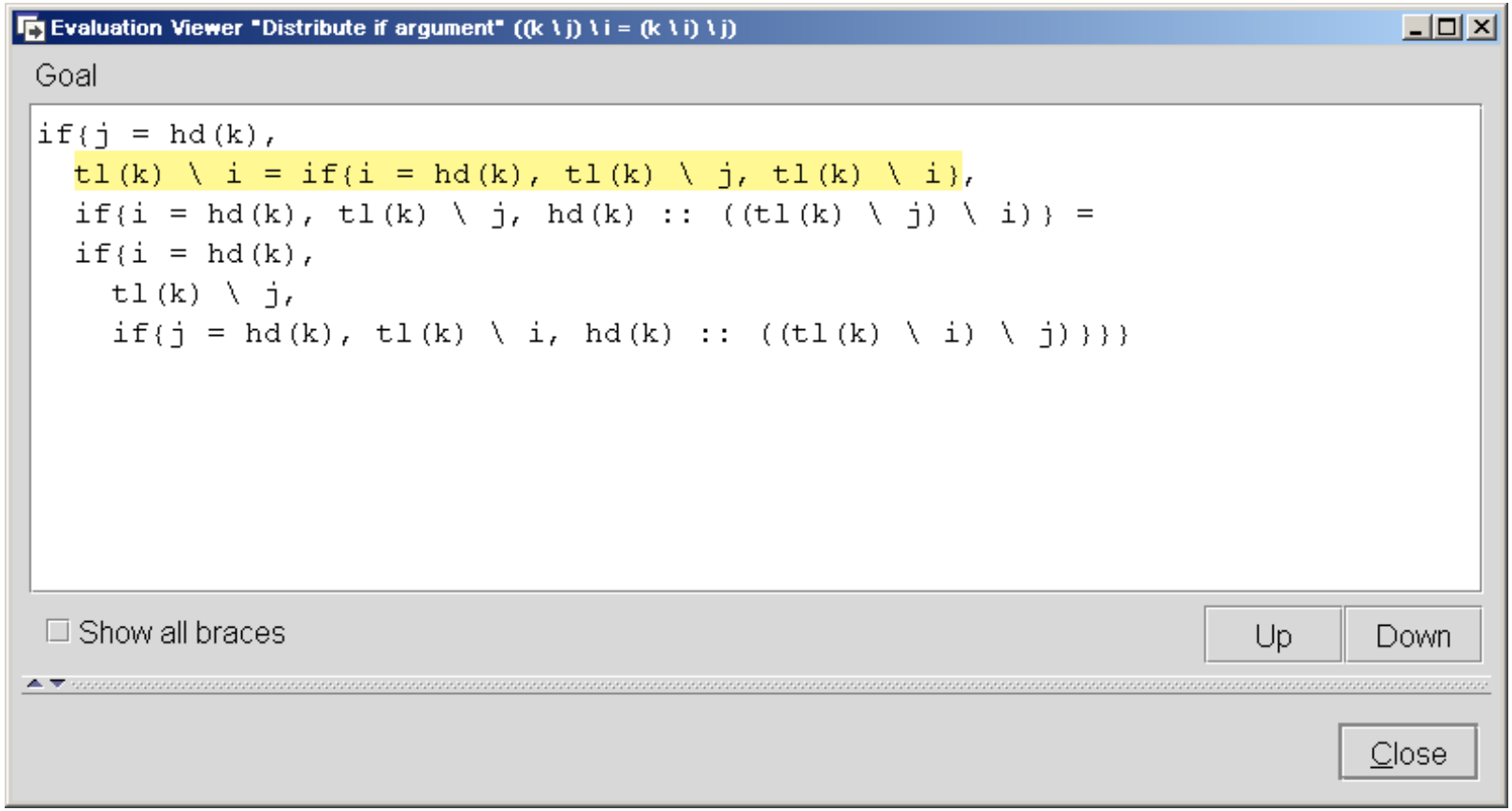
- *Distribute if-argument*:
  - Ausdrücke der Form $if\{a, b, c\} = if\{d, e, f\}$ dürfen durch $if\{a, b = if\{d, e, f\}, c = if\{d, e, f\}\}$ ersetzt werden.



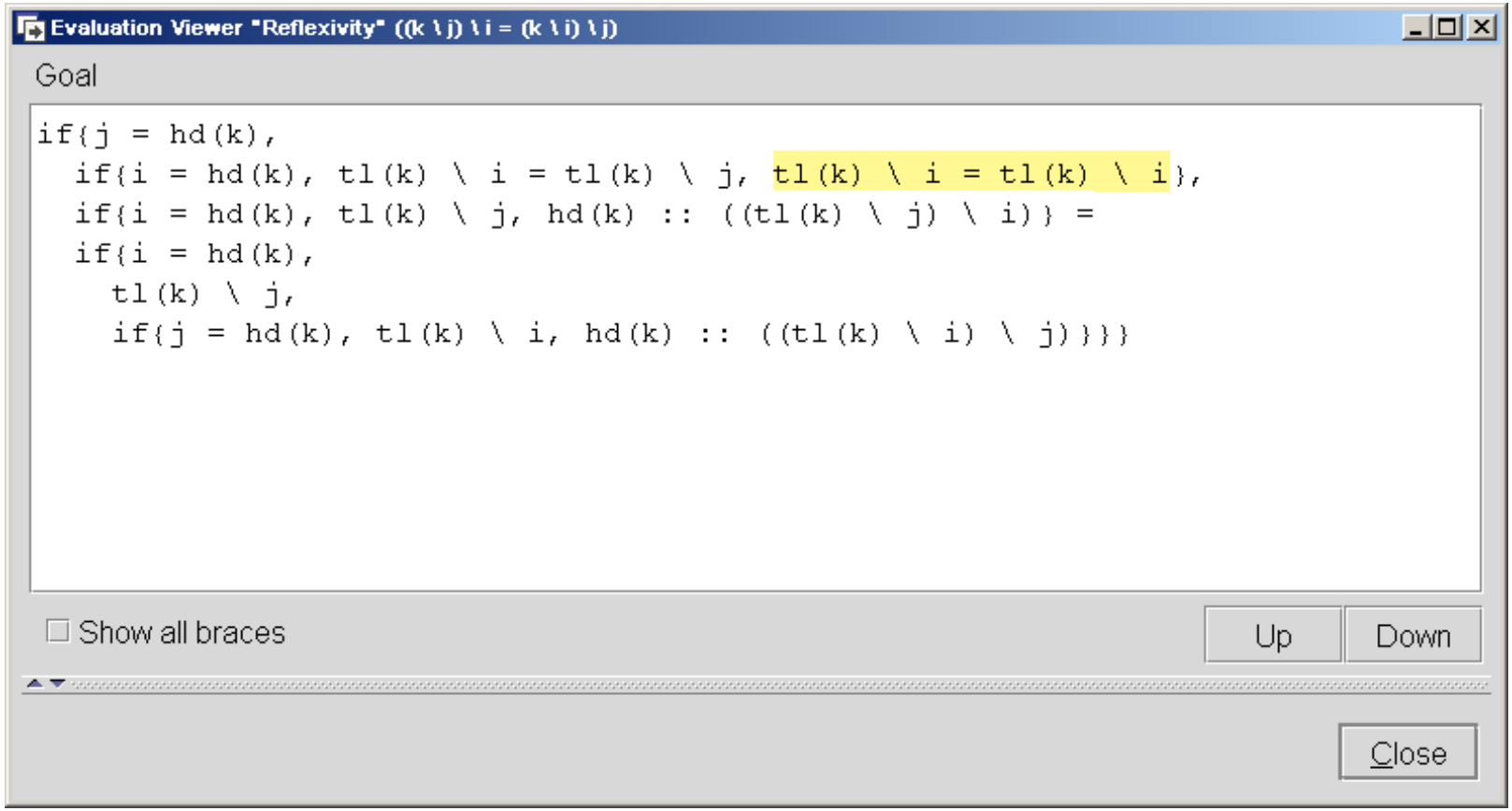
- *Affirmative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks a im *then*-Teil eines Ausdrucks der Form $if\{a, b[a], c\}$ darf durch $true$ ersetzt werden, also $if\{a, b[a], c\} \rightarrow if\{a, b[true], c\}$.
  - In diesem Fall wird damit
    - `if{j = hd(k), tl(k)\i = if{... j = hd(k) ...}, ...}`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



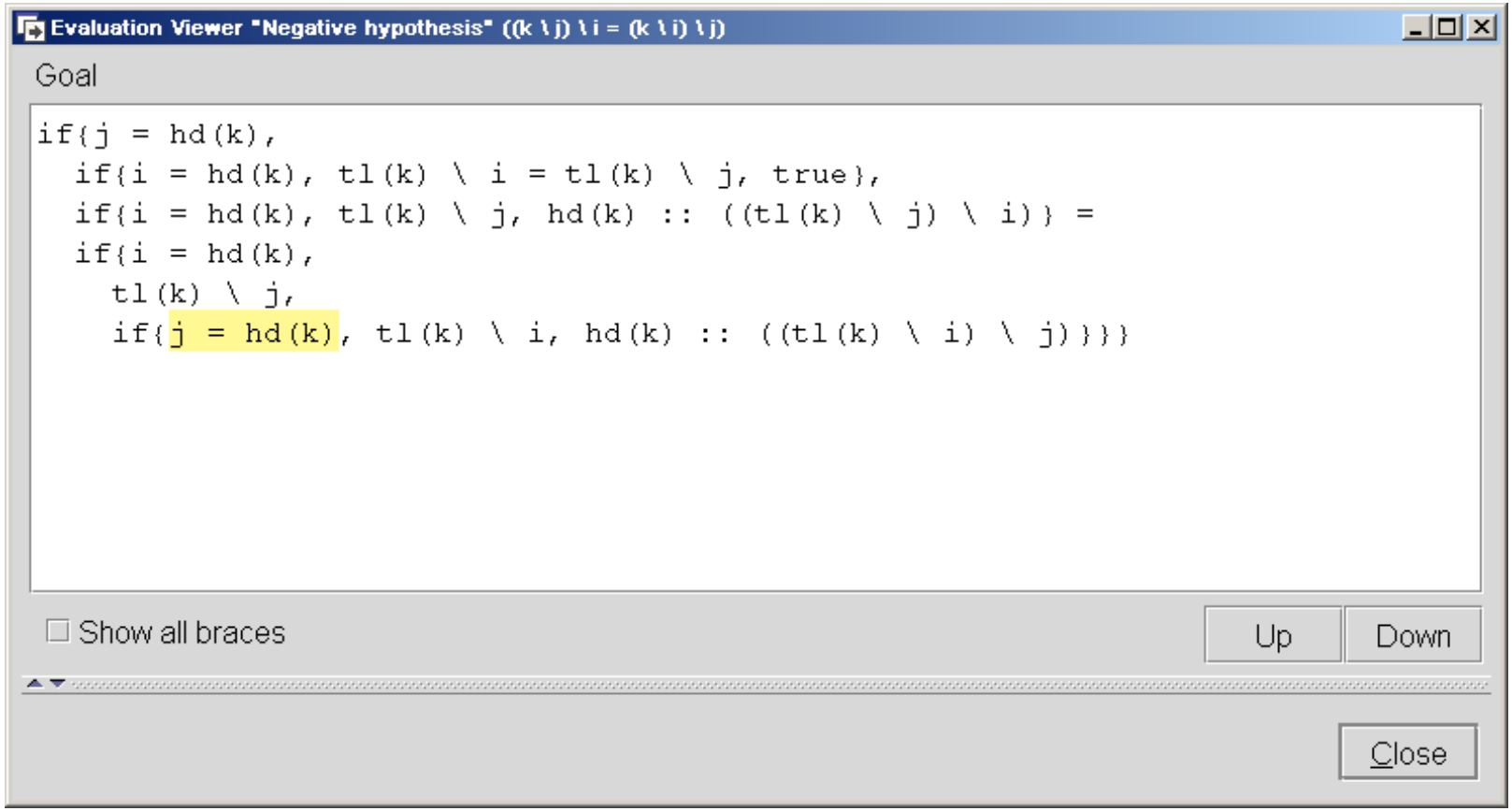
- *Keep then-part*:
  - Semantik von `if` – `if{true,`X`,`Y`}` darf durch X ersetzt werden.



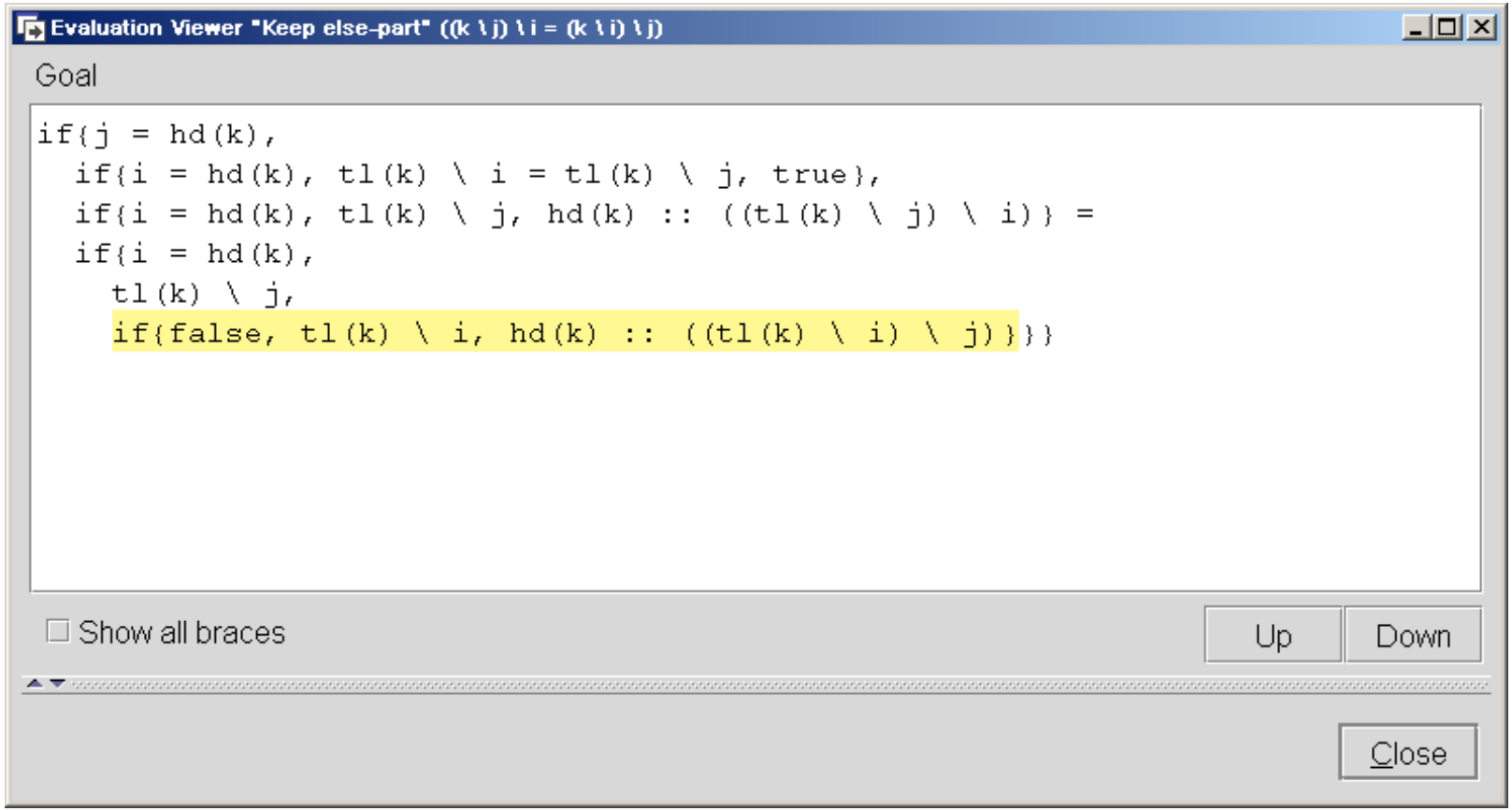
- *Distribute if-argument*:
  - Ausdrücke der Form $a = if\{b, c, d\}$ dürfen durch $if\{b, a = c, a = d\}$ ersetzt werden.



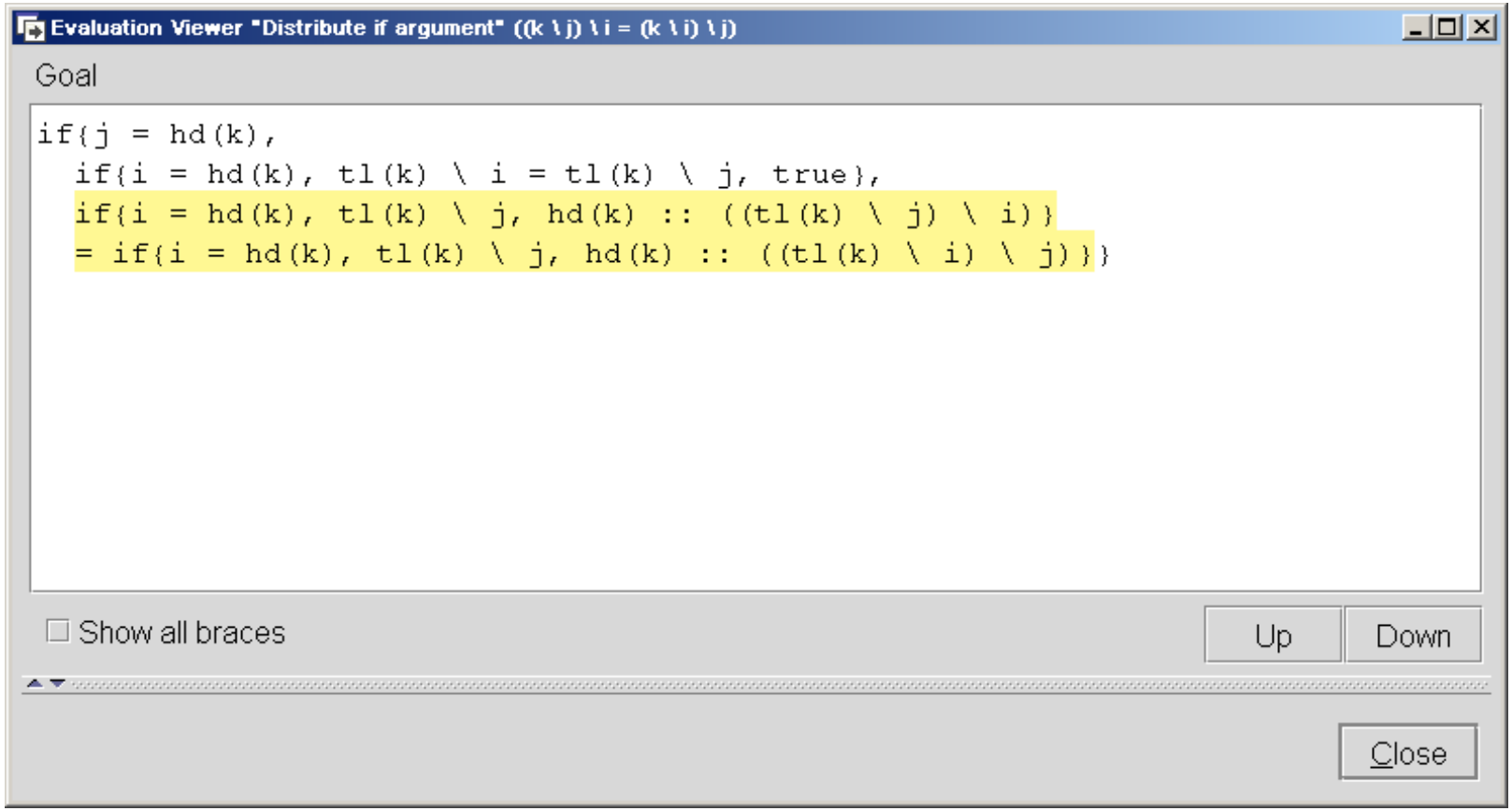
- *Reflexivity*:
  - Gleichheit ist *reflexiv*.
  - Damit darf jeder Ausdruck der Form $t = t$ durch $true$ ersetzt werden.



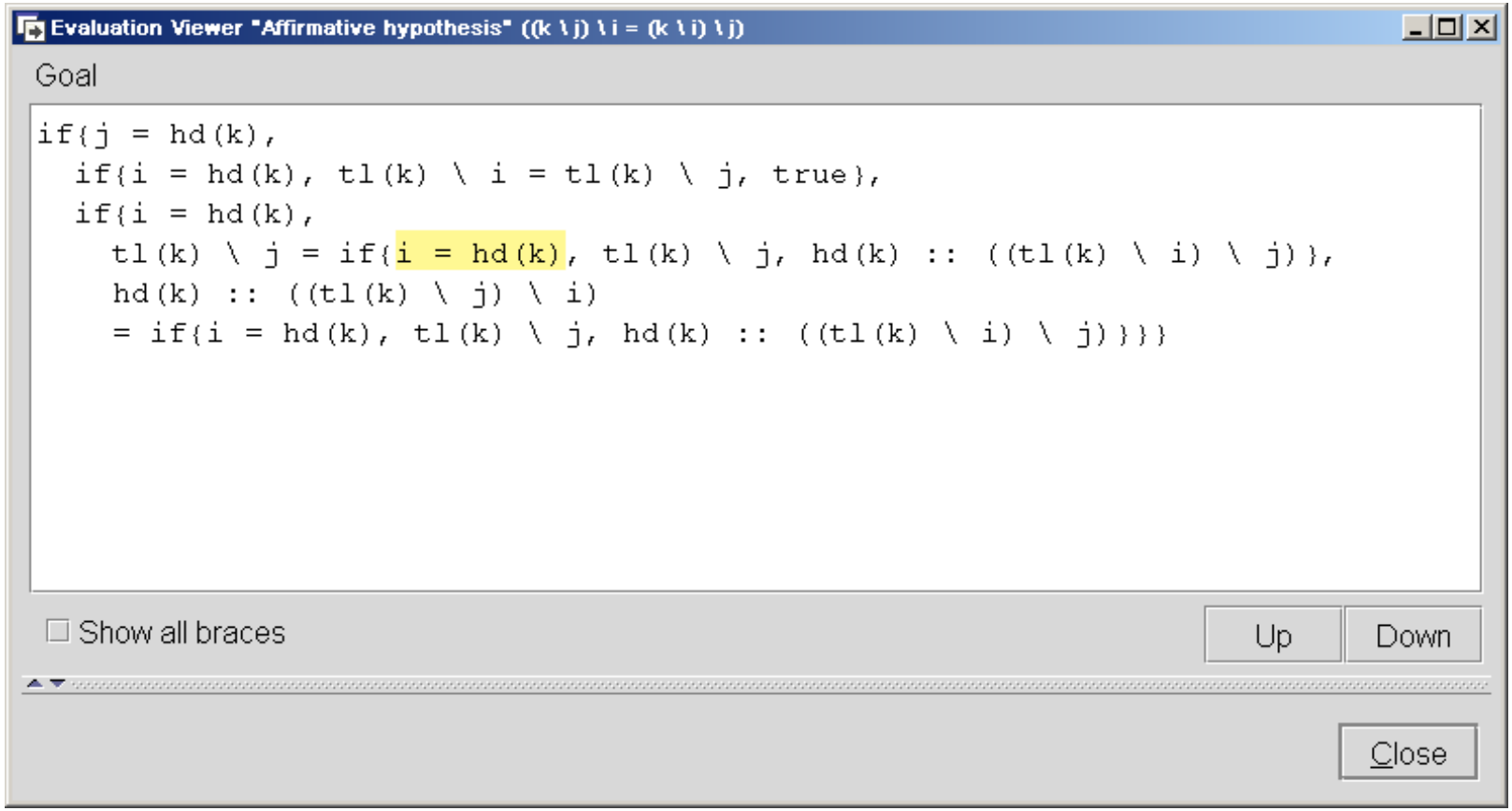
- *Negative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks a im *else*-Teil eines Ausdrucks der Form $if\{a, b, c[a]\}$ darf durch $false$ ersetzt werden, also $if\{a, b, c[a]\} \rightarrow if\{a, b, c[false]\}$.
  - In diesem Fall wird damit
    - `if{j = hd(k),...,if{... j = hd(k) ...}}` durch
 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



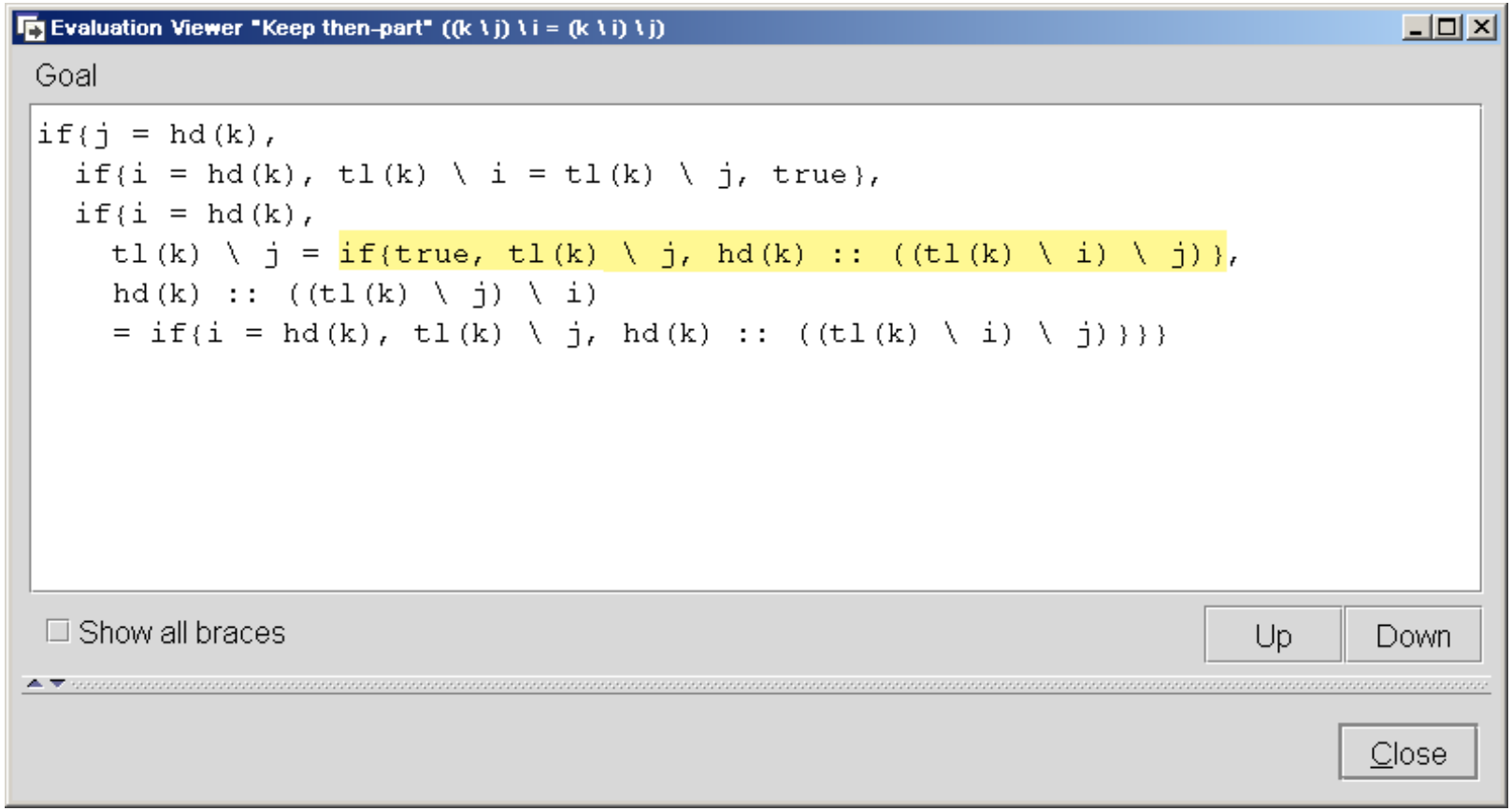
- *Keep else-part*:
  - Semantik von `if` – `if{false,`X`,`Y`}` darf durch Y ersetzt werden.



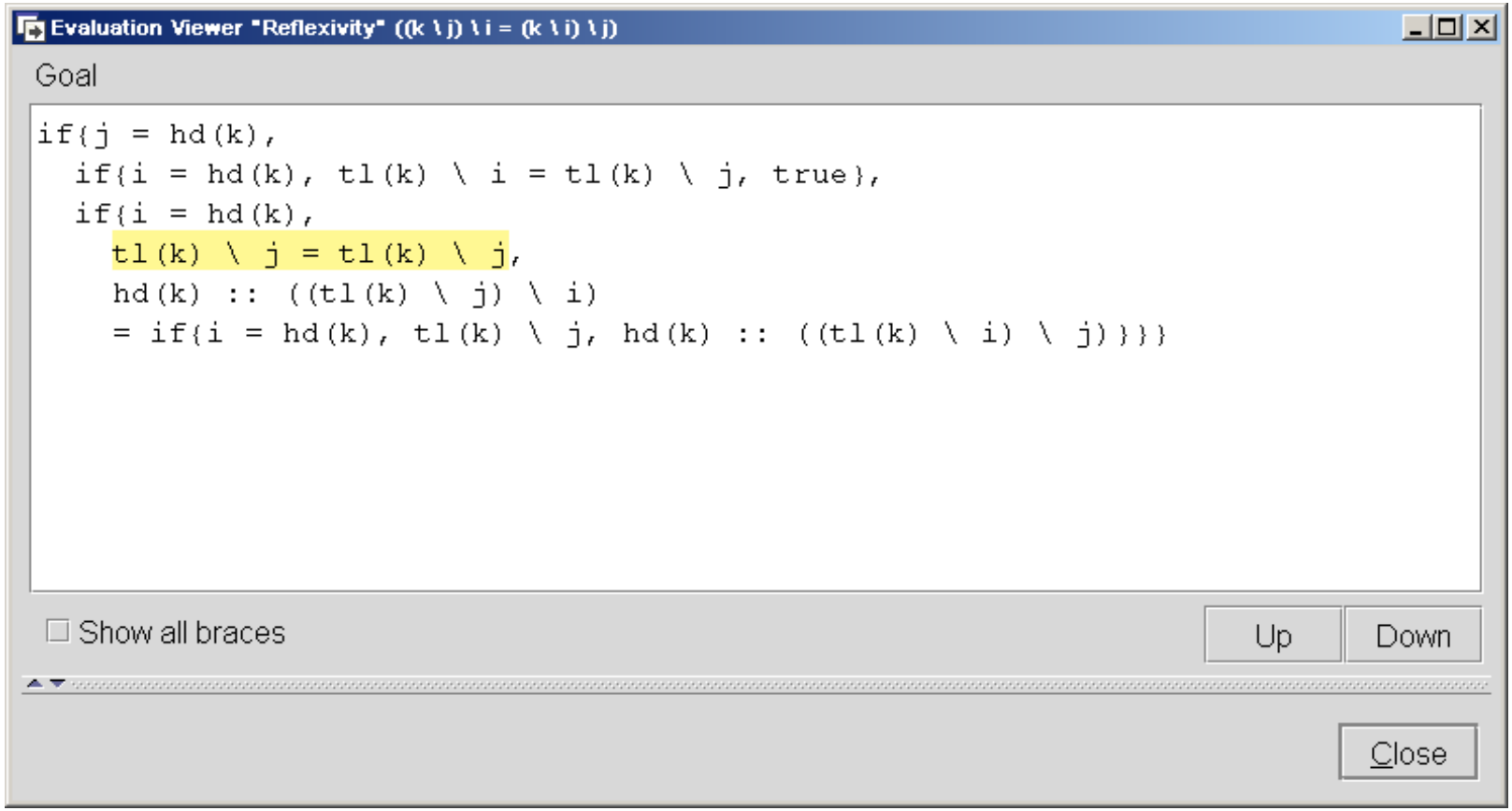
- *Distribute if-argument*:
  - Ausdrücke der Form $if\{a, b, c\} = if\{d, e, f\}$ dürfen durch $if\{a, b = if\{d, e, f\}, c = if\{d, e, f\}\}$ ersetzt werden.



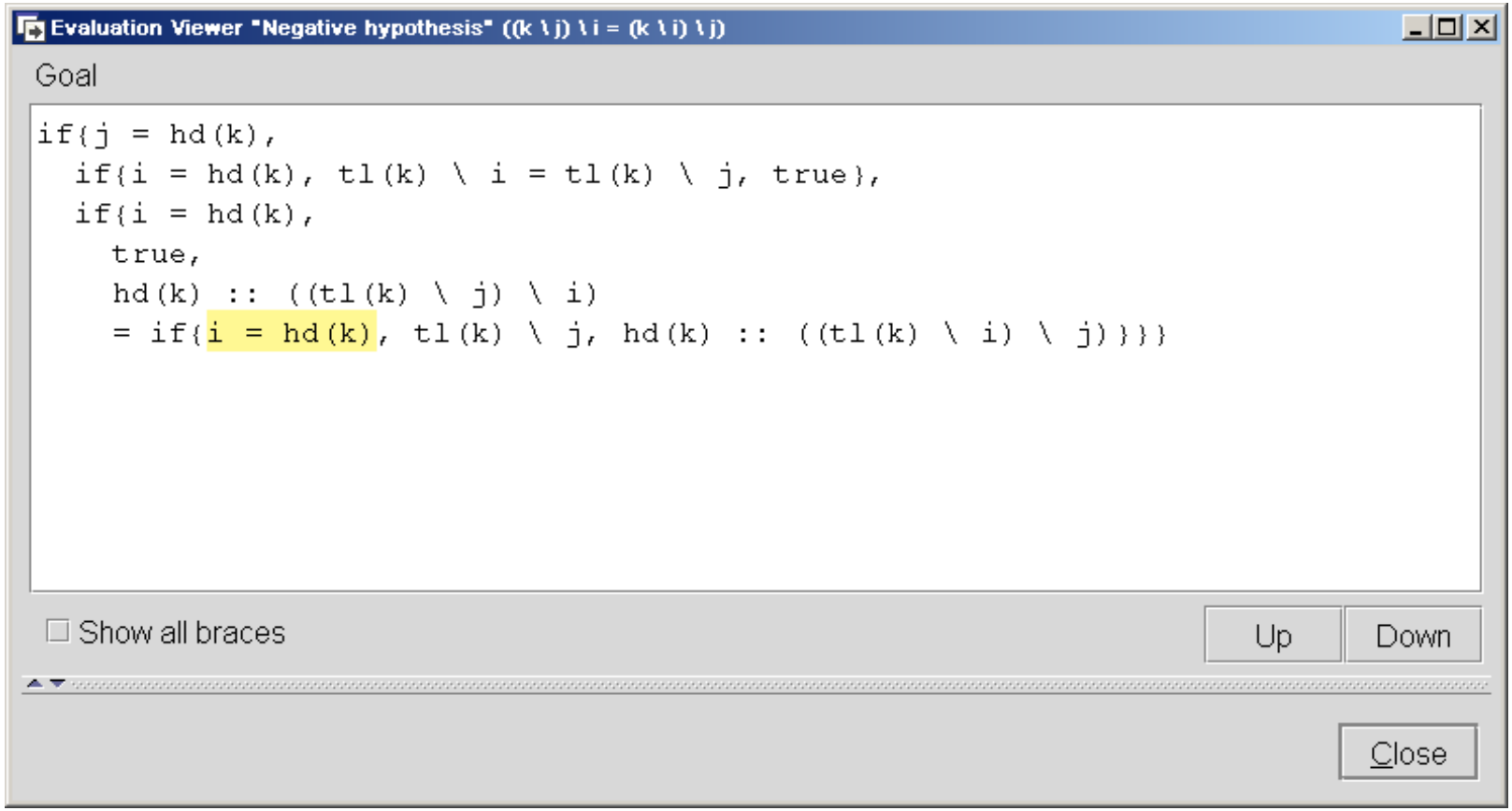
- *Affirmative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks a im *then*-Teil eines Ausdrucks der Form $if\{a, b[a], c\}$ darf durch $true$ ersetzt werden, also $if\{a, b[a], c\} \rightarrow if\{a, b[true], c\}$.



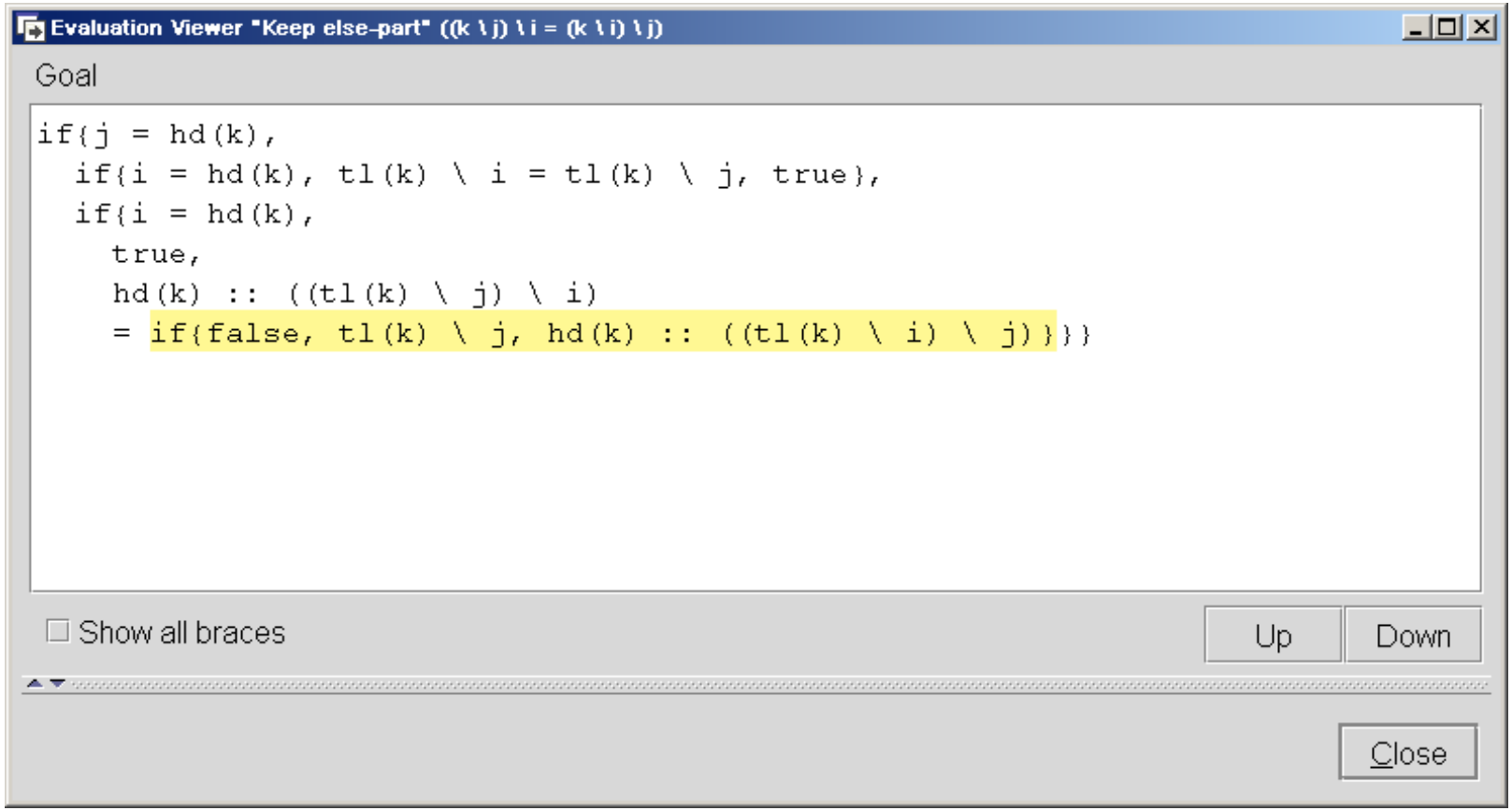
- *Keep then-part*:
  - Semantik von `if` – `if{true,`X`,`Y`}` darf durch X ersetzt werden.



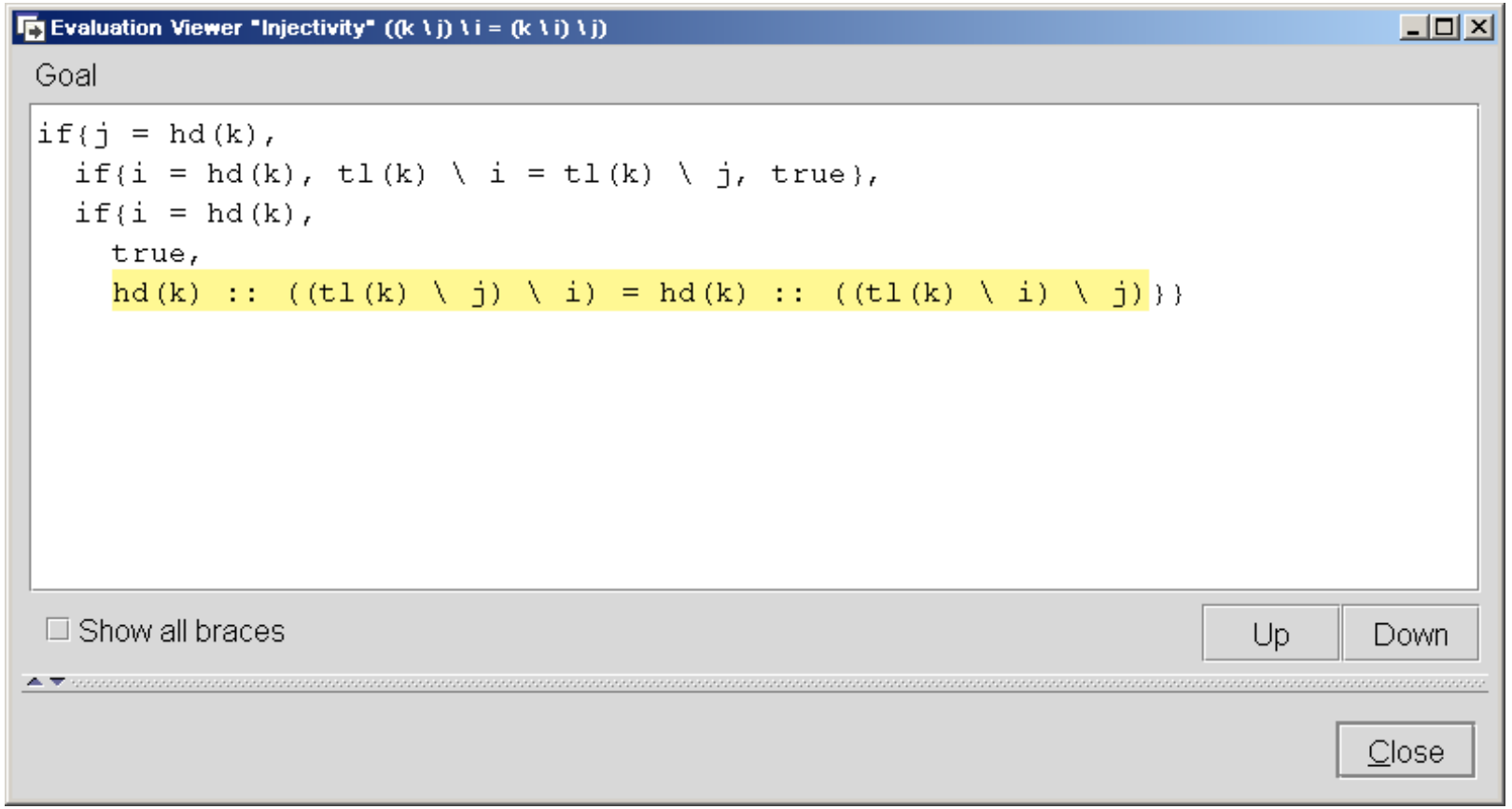
- *Reflexivity*:
  - Gleichheit ist *reflexiv*.
  - Damit darf jeder Ausdruck der Form $t = t$ durch $true$ ersetzt werden.



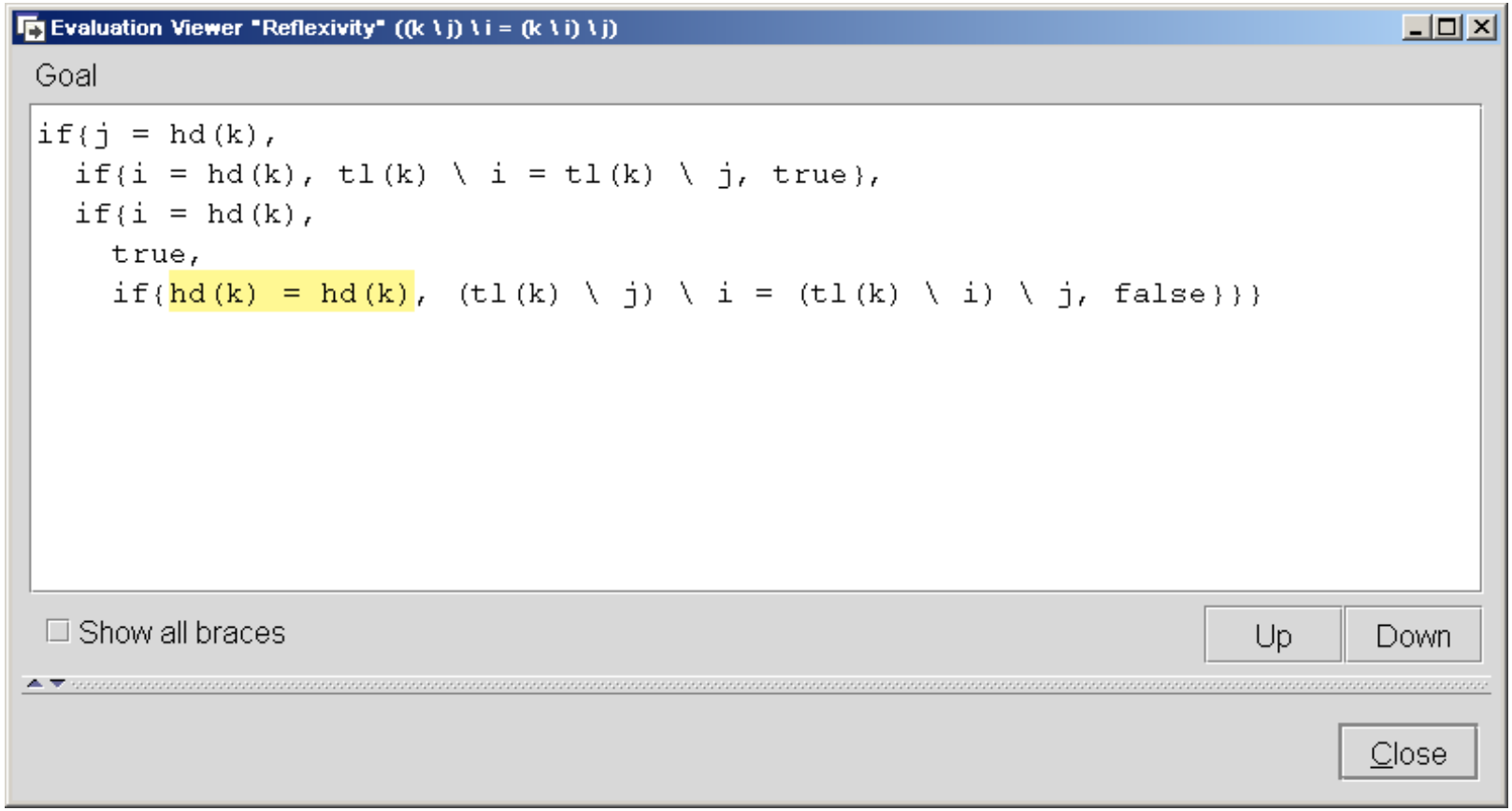
- *Negative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks a im *else*-Teil eines Ausdrucks der Form $if\{a, b, c[a]\}$ darf durch $false$ ersetzt werden, also $if\{a, b, c[a]\} \rightarrow if\{a, b, c[false]\}$.



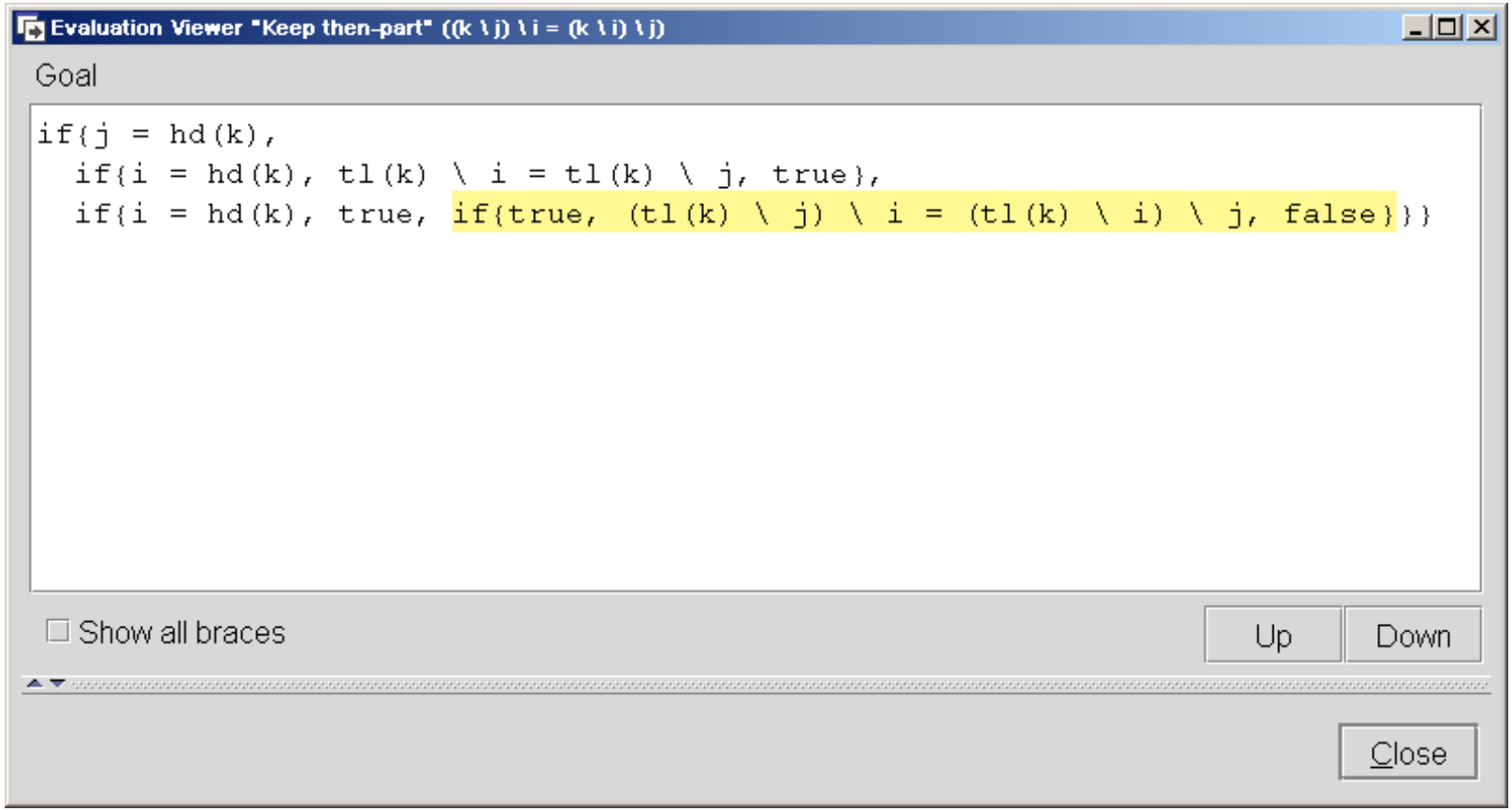
- *Keep else-part*:
  - Semantik von `if` – `if{false,`X`,`Y`}` darf durch Y ersetzt werden.



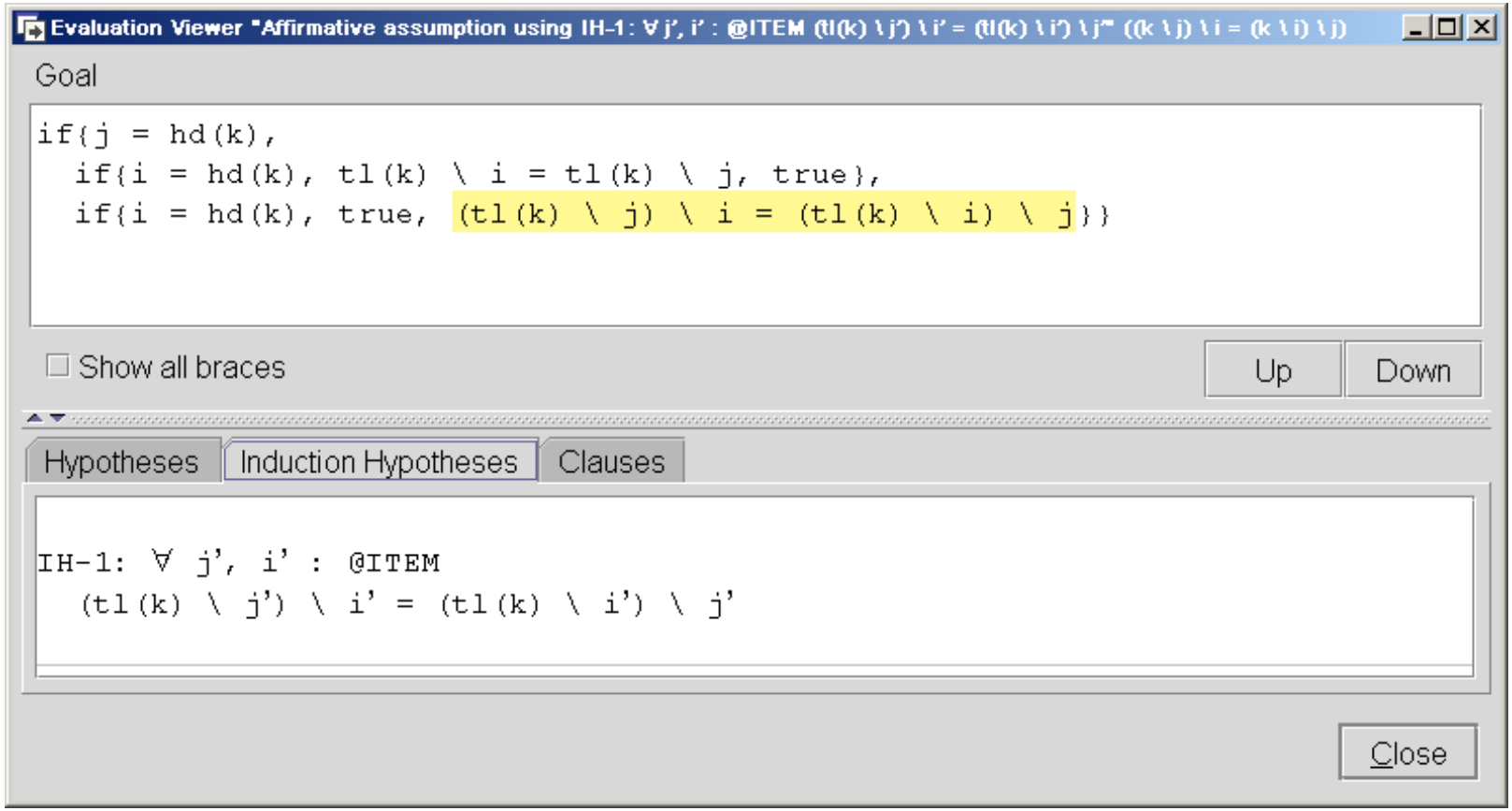
- *Injectivity*:
  - Für eine 2-stellige *injektive* Funktion g gilt $g(a,b) = g(c,d)$ gdw. $a = c \wedge b = d$, also darf $g(a,b) = g(c,d)$ durch $if\{a = c, b = d, false\}$ ersetzt werden.
  - Da Konstruktoren *injektiv* sind, wird hier
    - `hd(k)::((tl(k)\j)\i) = hd(k)::((tl(k)\i)\j)` ersetzt durch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



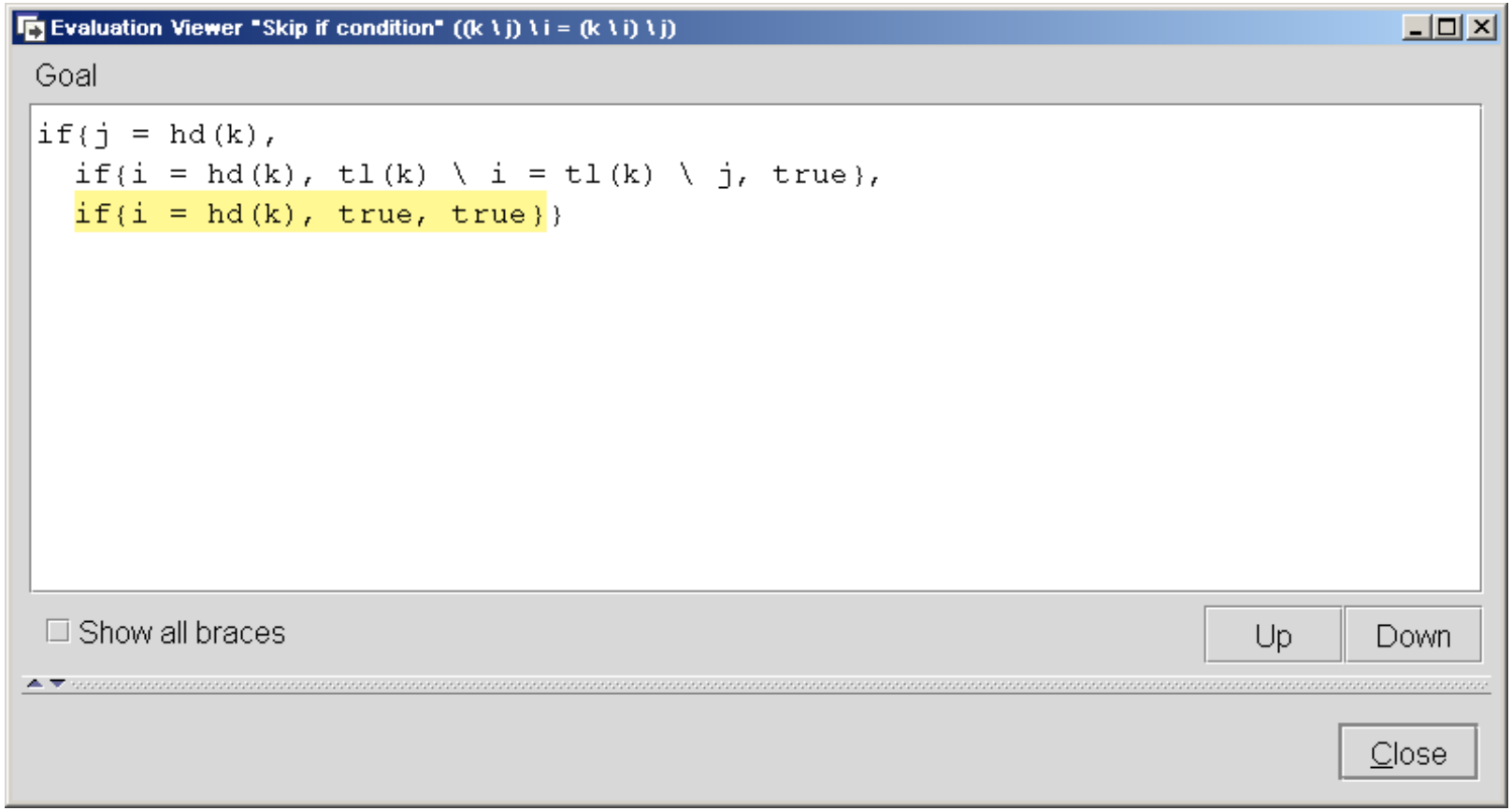
- *Reflexivity*:
  - Gleichheit ist *reflexiv*.
  - Damit darf jeder Ausdruck der Form $t = t$ durch $true$ ersetzt werden.



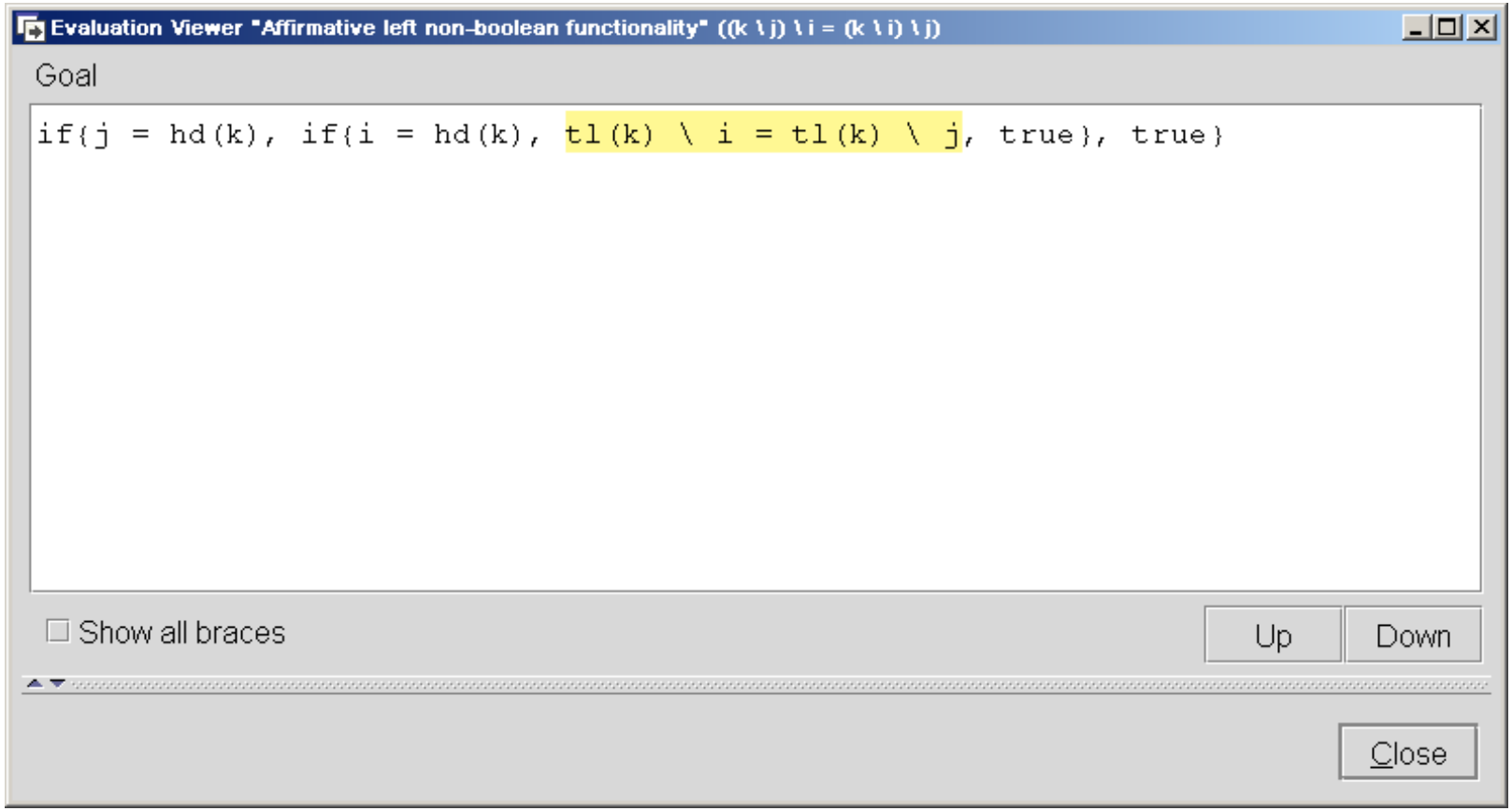
- *Keep then-part*:
  - Semantik von `if` – `if{true,`X`,`Y`}` darf durch X ersetzt werden.



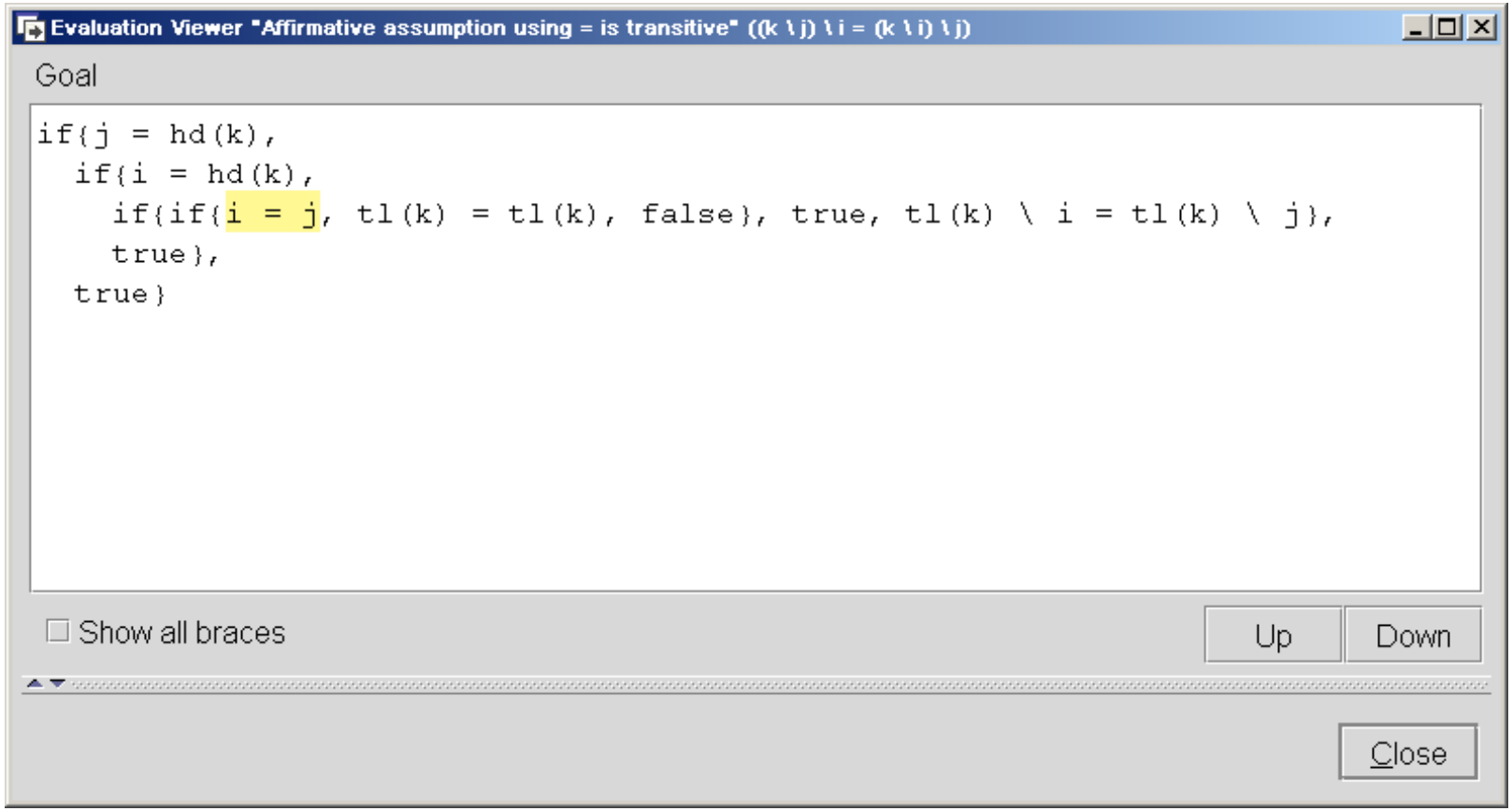
- *Affirmative assumption using . . .* :
  - Anwendung eines Lemmas oder einer *Induktionshypothese*, jede *Instanz* davon darf durch $true$ ersetzt werden.
  - Die Induktionshypothese lautet hier: `(tl(k)\j')\i' = (tl(k)\i')\j'`
  - *Bedeutung*: Die Variablen mit "′" sind *allquantifiziert*, dürfen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



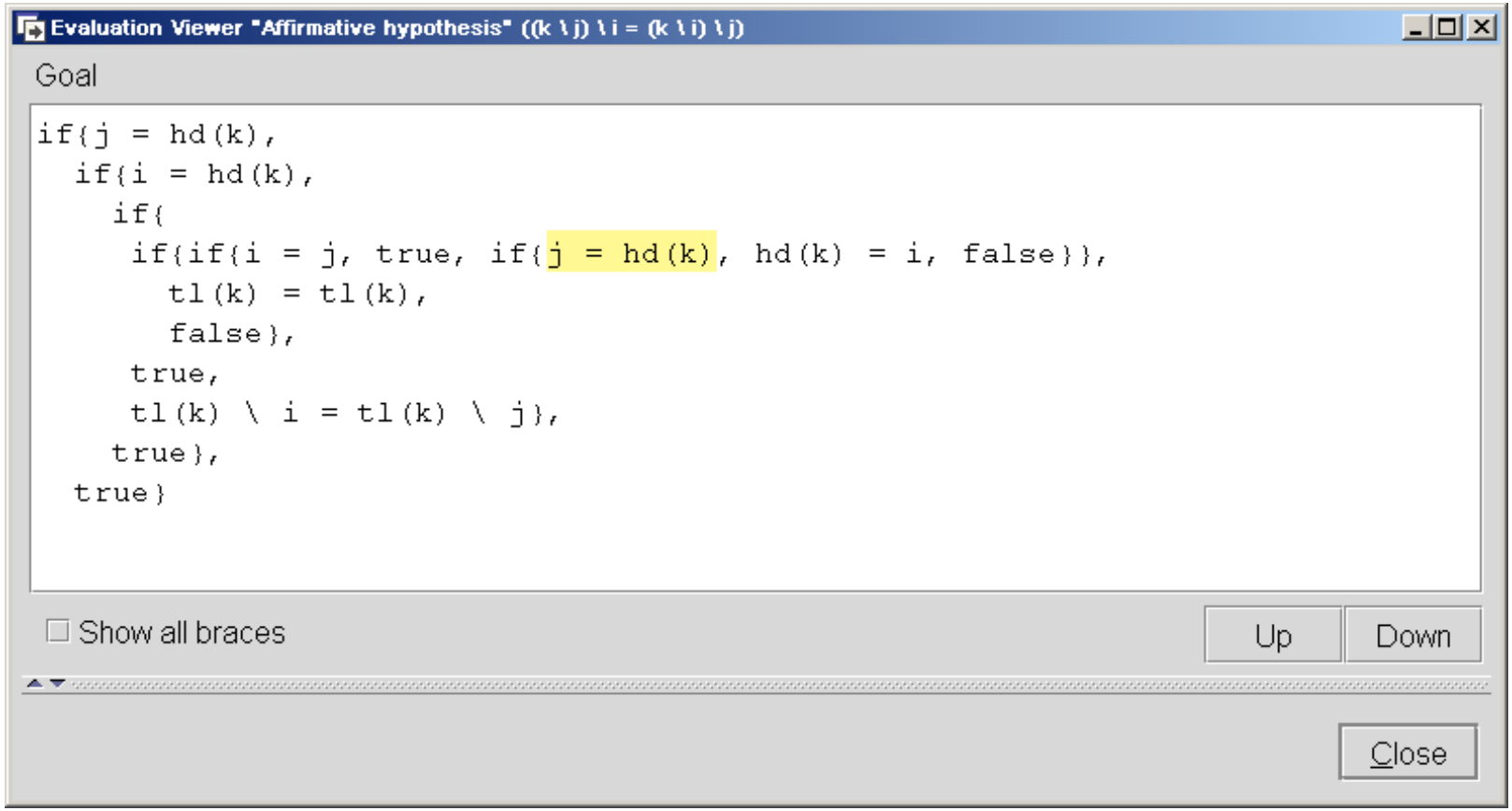
- *Skip if-condition*:
  - Irrelevante Fallunterscheidung, denn *then*- und *else*-Teil sind identisch.
  - Damit darf jeder Ausdruck der Form $if\{a, b, b\}$ durch b ersetzt werden.



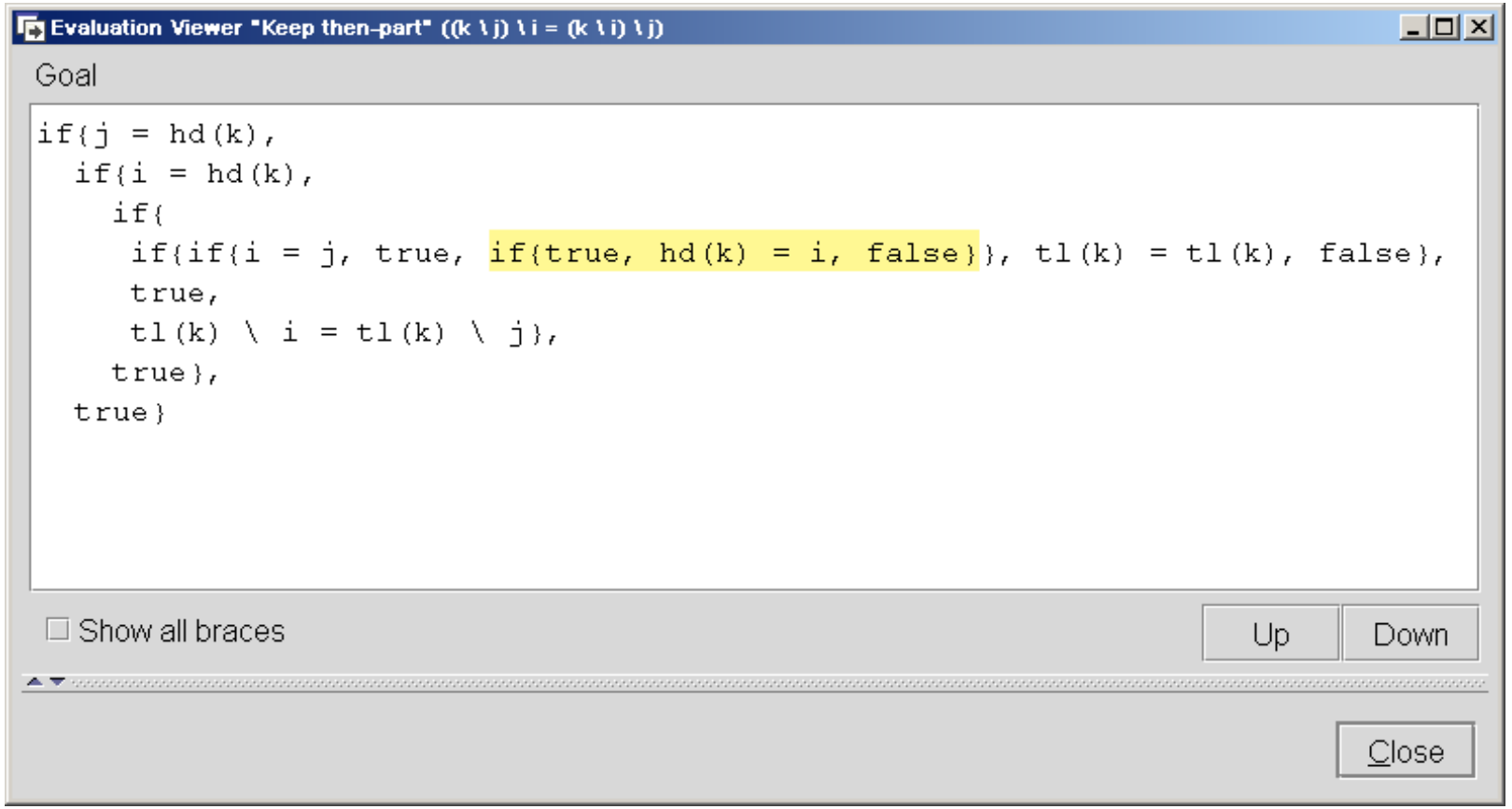
- *Affirmative left non-boolean functionality*:
  - $g(a,b) = g(c,d)$ darf durch $true$ ersetzt werden, falls $a = c \wedge b = d$, also darf $g(a,b) = g(c,d)$ durch $if\{if\{a = c, b = d, false\}, true, g(a,b) = g(c,d)\}$ ersetzt werden.
  - Konkret wird hier
    - `tl(k)\i = tl(k)\j` ersetzt durch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



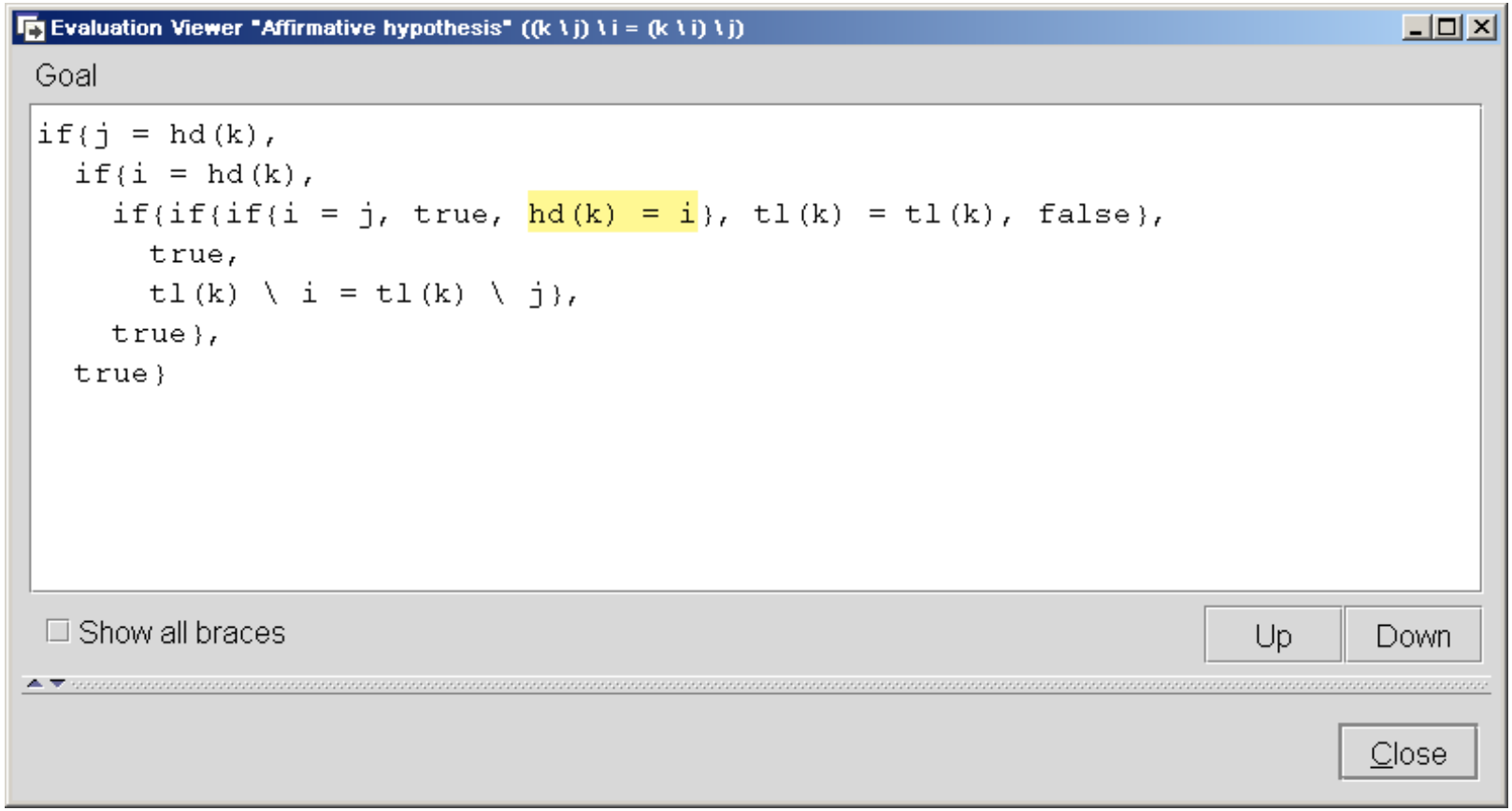
- *Affirmative assumption using* . . . :
  - Anwendung eines *Lemmas* oder einer Induktionshypothese.
  - Hier wird das Transitivitätsgesetz für $=$ verwendet, d.h. $\forall x, y, z{:}@T\ if\{x = y, if\{y = z, x = z, true\}, true\}$.
  - Damit darf
    - `i = j` ersetzt werden durch
    - `if{i = j,true,if{j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



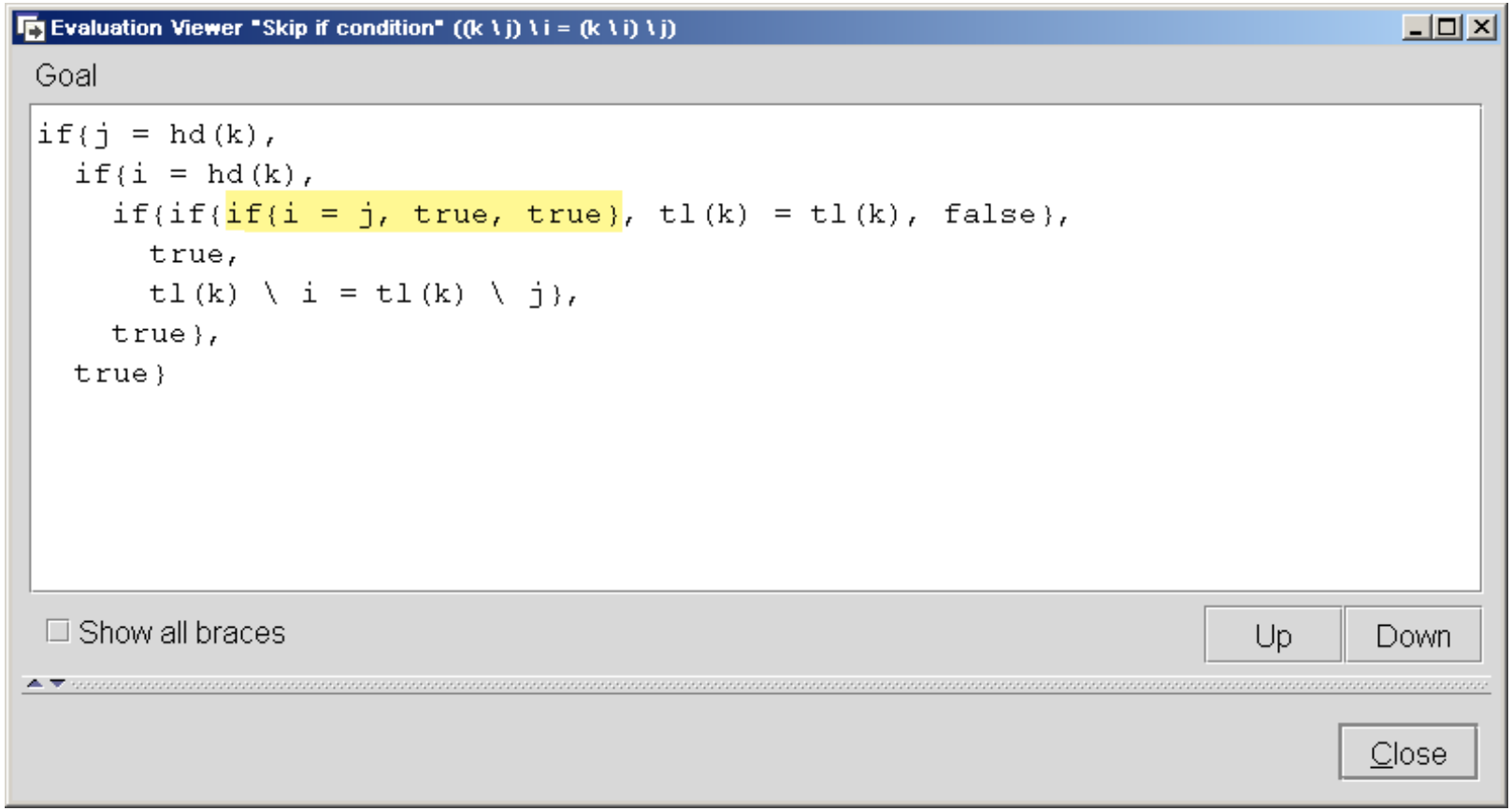
- *Affirmative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks a im *then*-Teil eines Ausdrucks der Form $if\{a, b[a], c\}$ darf durch $true$ ersetzt werden, also $if\{a, b[a], c\} \rightarrow if\{a, b[true], c\}$.



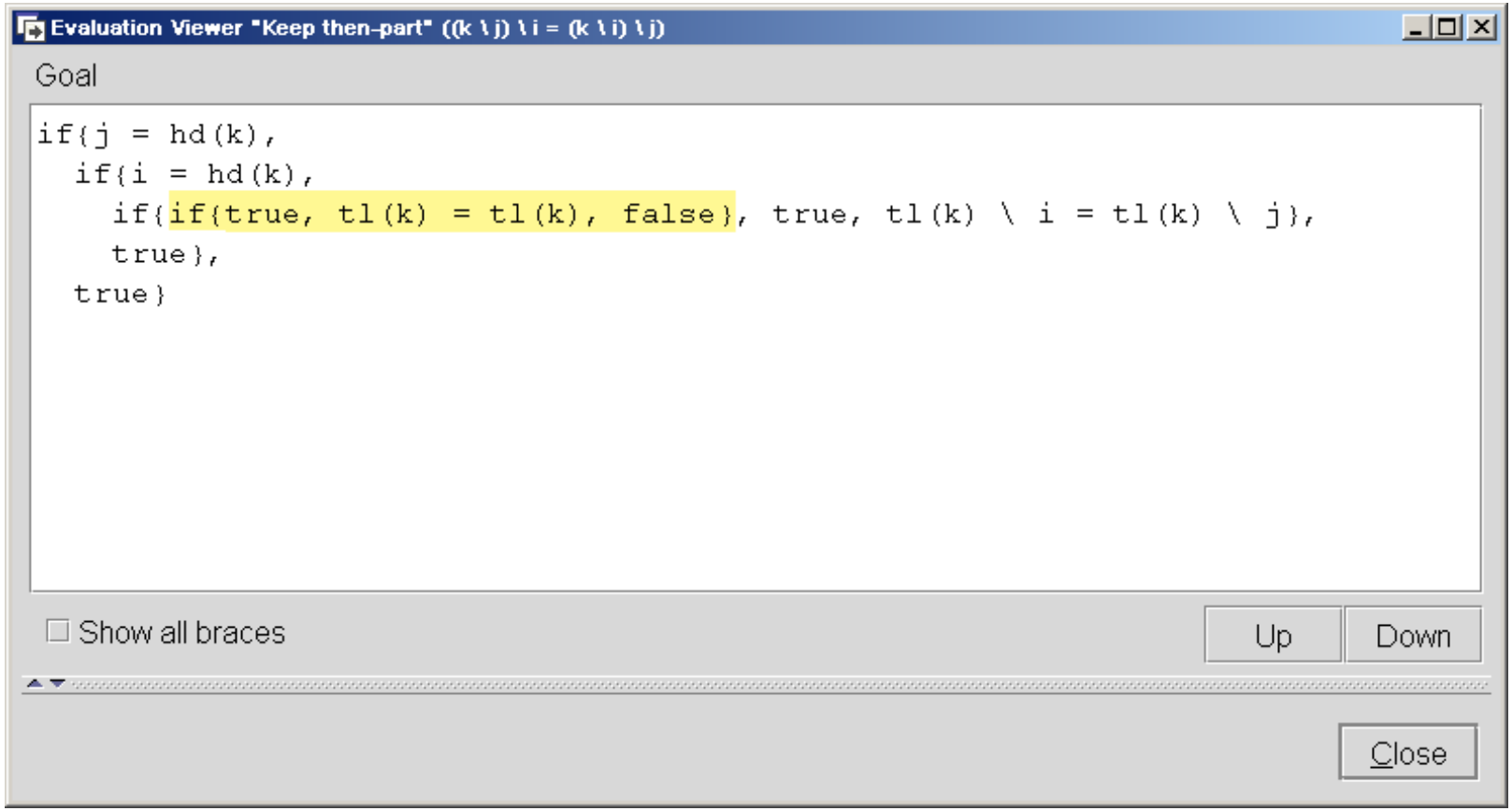
- *Keep then-part*:
  - Semantik von `if` – `if{true,`X`,`Y`}` darf durch X ersetzt werden.



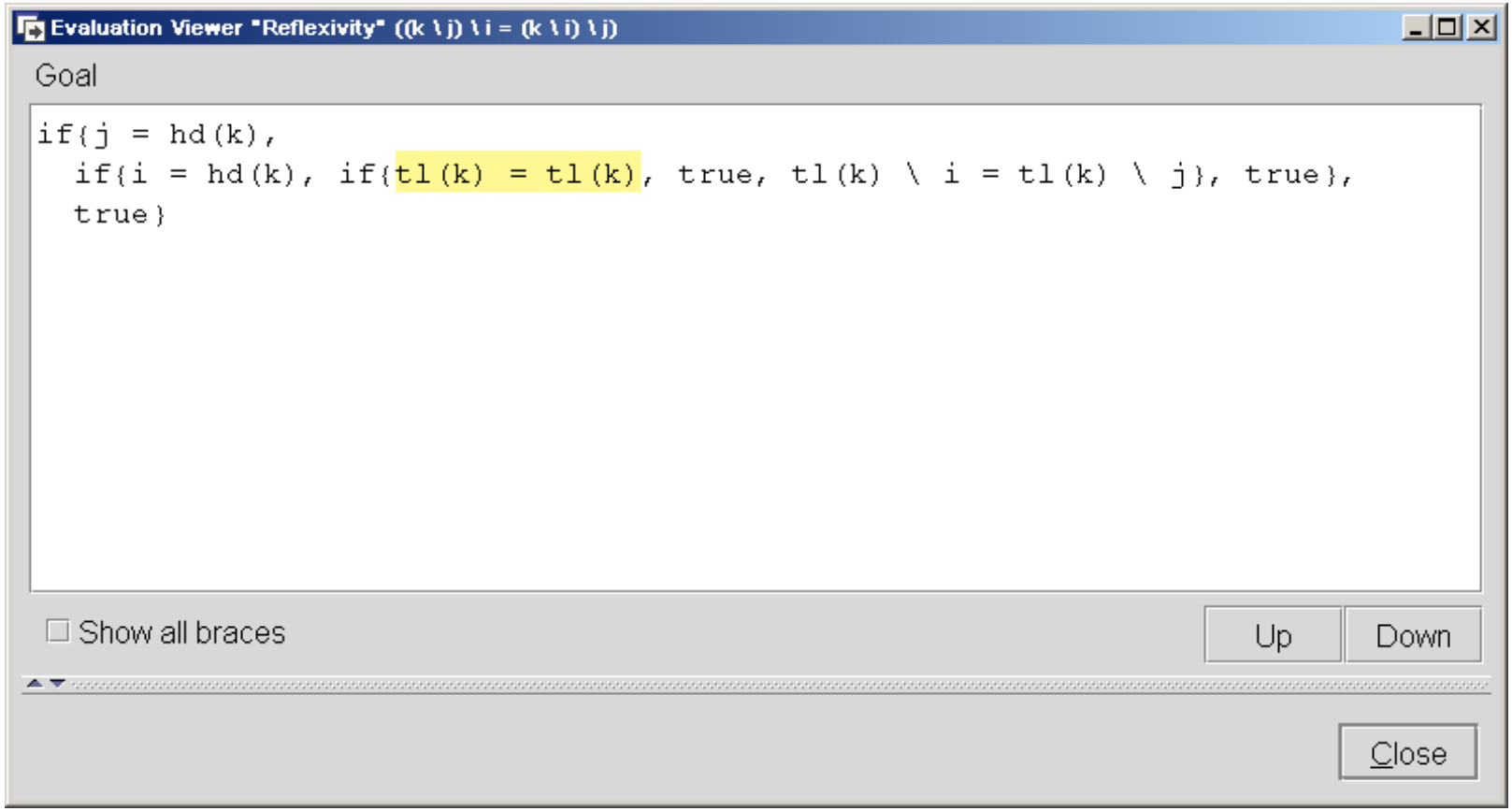
- *Affirmative hypothesis*:
  - Jedes Vorkommen eines Ausdrucks a im *then*-Teil eines Ausdrucks der Form $if\{a, b[a], c\}$ darf durch $true$ ersetzt werden, also $if\{a, b[a], c\} \rightarrow if\{a, b[true], c\}$.
  - Wir dürfen `hd(k) = i` auch unter der Bedingung `i = hd(k)` durch `true` ersetzen,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



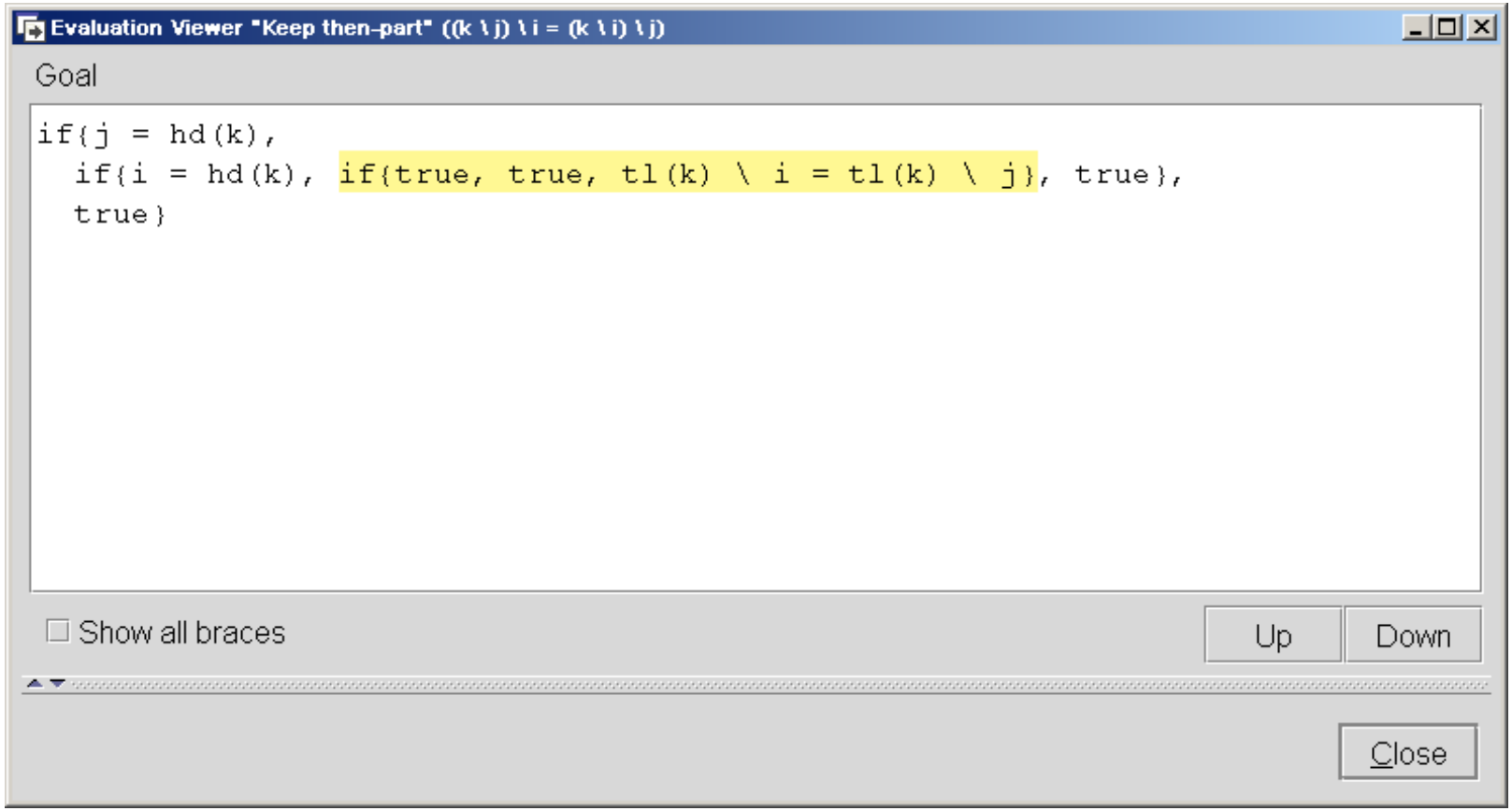
- *Skip if-condition*:
  - Irrelevante Fallunterscheidung, denn *then*- und *else*-Teil sind identisch.
  - Damit darf jeder Ausdruck der Form $if\{a, b, b\}$ durch b ersetzt werden.



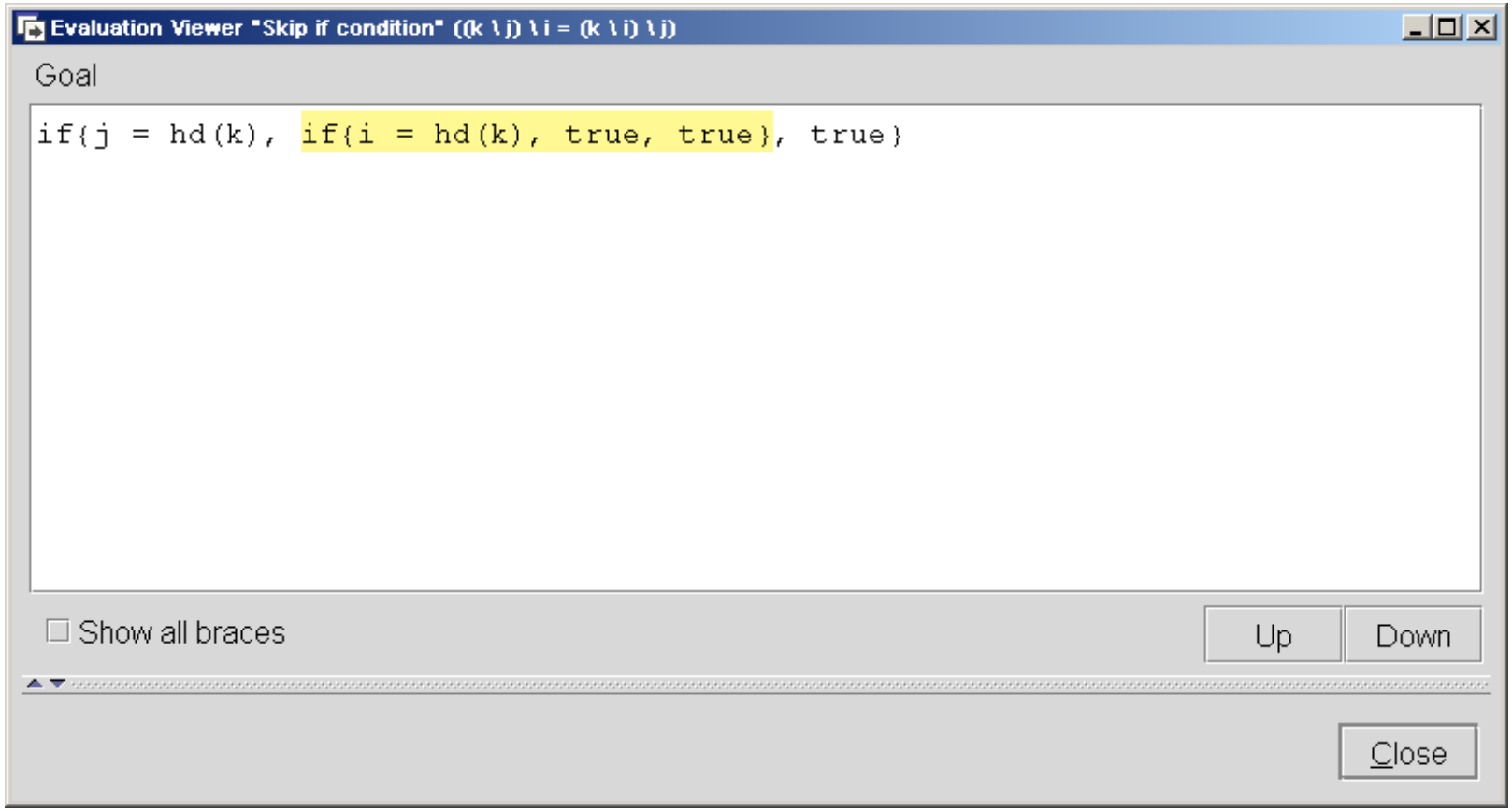
- *Keep then-part*:
  - Semantik von `if` – `if{true,`X`,`Y`}` darf durch X ersetzt werden.



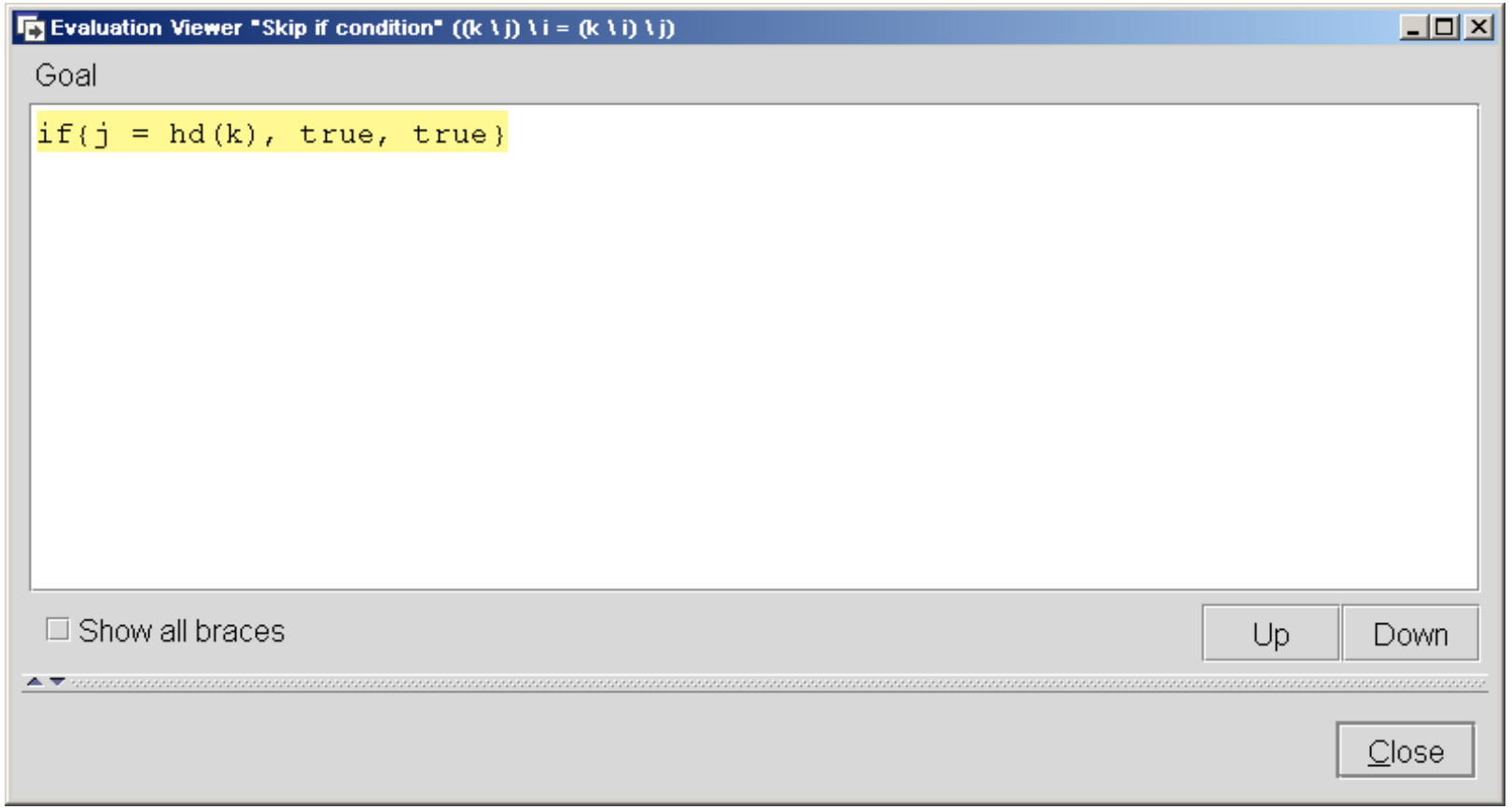
- *Reflexivity*:
  - Gleichheit ist *reflexiv*.
  - Damit darf jeder Ausdruck der Form $t = t$ durch $true$ ersetzt werden.



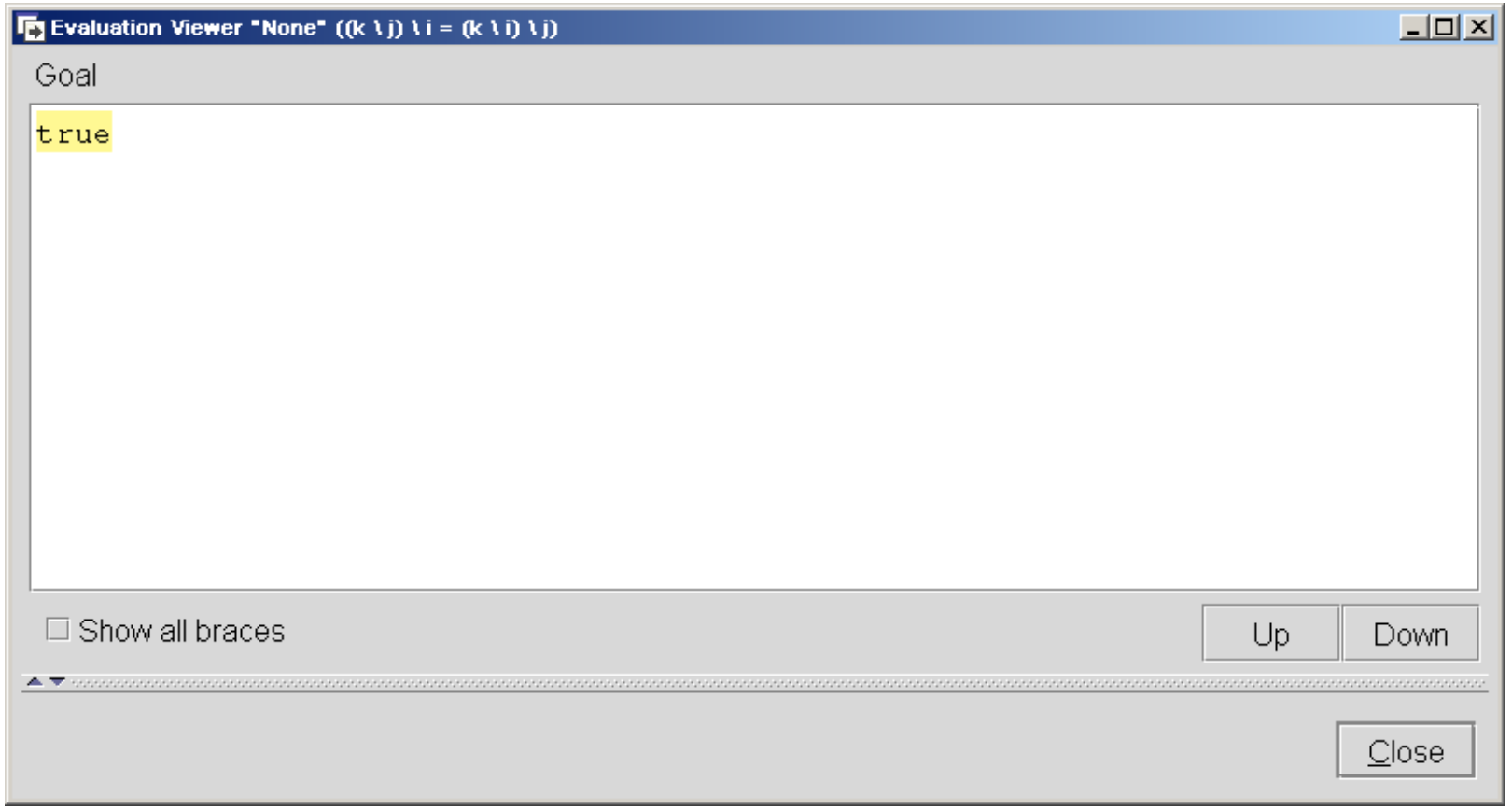
- *Keep then-part*:
  - Semantik von `if` – `if{true,`X`,`Y`}` darf durch X ersetzt werden.



- *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.



- *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.



- *Bewiesen, fertig !*