AN INTRODUCTION TO SEPARATION LOGIC

5. Trees and Dags

John C. Reynolds
Carnegie Mellon University

February 18, 2011

S-expressions (a la LISP)

$\tau \in$ S-exps iff $\tau \in$ Atoms

or $\tau = (\tau_0 \cdot \tau_1)$ where $\tau_0, \tau_1 \in$ S-exps.

Representing S-expressions by Trees

For $\tau \in$ S-exps, we define the assertion $\mathsf{tree}\ \tau\ (i)$ by structural induction:

$\mathsf{tree}\ a\ (i)$ iff $\mathbf{emp} \wedge i = a$ when a is an atom

$\mathsf{tree}\ (\tau_0 \cdot \tau_1)\ (i)$ iff

$\exists i_0, i_1.\ \ i \mapsto i_0, i_1\ *\ \mathsf{tree}\ \tau_0\ (i_0)\ *\ \mathsf{tree}\ \tau_1\ (i_1).$

One can show that the assertions $\mathsf{tree}\ \tau\ (i)$ and $\exists \tau.\ \mathsf{tree}\ \tau\ (i)$ are precise.

—

Copying Trees

We will show that

$$\begin{array}{l}\mathsf{copytree(j;i)} = \\ \quad \mathbf{if}\ \mathsf{isatom(i)}\ \mathbf{then}\ \mathsf{j} := \mathsf{i}\ \mathbf{else} \\ \quad\quad \mathbf{newvar}\ \mathsf{i}_0, \mathsf{i}_1, \mathsf{j}_0, \mathsf{j}_1\ \mathbf{in} \\ \quad\quad\quad \big(\mathsf{i}_0 := [\mathsf{i}]\ ;\ \mathsf{i}_1 := [\mathsf{i}+1]\ ; \\ \quad\quad\quad \mathsf{copytree}(\mathsf{j}_0;\mathsf{i}_0)\ ;\ \mathsf{copytree}(\mathsf{j}_1;\mathsf{i}_1)\ ;\ \mathsf{j} := \mathbf{cons}(\mathsf{j}_0,\mathsf{j}_1)\big).\end{array}$$

satisfies

$$\{\mathsf{tree}\ \tau(\mathsf{i})\}\ \mathsf{copytree(j;i)}\{\tau\}\ \{\mathsf{tree}\ \tau(\mathsf{i})\ *\ \mathsf{tree}\ \tau(\mathsf{j})\}.$$

$$\begin{array}{l}\{\mathsf{tree}\ \tau(\mathsf{i})\}\ \mathsf{copytree(j;i)}\{\tau\}\ \{\mathsf{tree}\ \tau(\mathsf{i})\ *\ \mathsf{tree}\ \tau(\mathsf{j})\} \vdash \\ \quad \{\mathsf{tree}\ \tau(\mathsf{i})\} \\ \quad\quad \mathbf{if}\ \mathsf{isatom(i)}\ \mathbf{then} \\ \quad\quad \{\mathsf{isatom}(\tau) \wedge \mathbf{emp} \wedge \mathsf{i} = \tau\} \\ \quad\quad \{\mathsf{isatom}(\tau) \wedge ((\mathbf{emp} \wedge \mathsf{i} = \tau)\ *\ (\mathbf{emp} \wedge \mathsf{i} = \tau))\} \\ \quad\quad \mathsf{j} := \mathsf{i} \\ \quad\quad \{\mathsf{isatom}(\tau) \wedge ((\mathbf{emp} \wedge \mathsf{i} = \tau)\ *\ (\mathbf{emp} \wedge \mathsf{j} = \tau))\} \\ \quad\quad\quad \vdots\end{array}$$

—

$$
\begin{array}{l}
\quad \vdots \\
\mathbf{else} \\
\{\exists \tau_0, \tau_1.\ \ \tau = (\tau_0 \cdot \tau_1) \wedge \mathsf{tree}\,(\tau_0 \cdot \tau_1)(\mathsf{i})\} \\
\quad \mathbf{newvar}\ \mathsf{i}_0, \mathsf{i}_1, \mathsf{j}_0, \mathsf{j}_1\ \mathbf{in}\ \Big(\mathsf{i}_0 := [\mathsf{i}]\ ;\ \mathsf{i}_1 := [\mathsf{i}+1]\ ; \\
\qquad \{\mathsf{tree}\,\tau_0\,(\mathsf{i}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{i}_1)\} \\
\qquad \left.\begin{array}{l} \{\mathsf{tree}\,\tau_0(\mathsf{i}_0)\} \\ \mathsf{copytree}(\mathsf{j}_0; \mathsf{i}_0)\{\tau_0\} \\ \{\mathsf{tree}\,\tau_0(\mathsf{i}_0)\ *\ \mathsf{tree}\,\tau_0(\mathsf{j}_0)\} \end{array}\right\} *\ \mathsf{tree}\,\tau_1\,(\mathsf{i}_1) \\
\qquad \left.\begin{array}{l} \{\mathsf{tree}\,\tau_1(\mathsf{i}_1)\} \\ \mathsf{copytree}(\mathsf{j}_1; \mathsf{i}_1)\{\tau_1\} \\ \{\mathsf{tree}\,\tau(\mathsf{i}_1)\ *\ \mathsf{tree}\,\tau_1(\mathsf{j}_1)\} \end{array}\right\} * \left(\begin{array}{c} \mathsf{tree}\,\tau_0\,(\mathsf{i}_0) \\ * \\ \mathsf{tree}\,\tau_0\,(\mathsf{j}_0) \end{array}\right) \\
\qquad \{\mathsf{tree}\,\tau_0\,(\mathsf{i}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{i}_1)\ * \\
\qquad\qquad \mathsf{tree}\,\tau_0\,(\mathsf{j}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{j}_1)\} \\
\qquad \mathsf{j} := \mathbf{cons}(\mathsf{j}_0, \mathsf{j}_1) \\
\qquad \{\mathsf{tree}\,\tau_0\,(\mathsf{i}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{i}_1)\ * \\
\qquad\qquad \mathsf{j} \mapsto \mathsf{j}_0, \mathsf{j}_1\ *\ \mathsf{tree}\,\tau_0\,(\mathsf{j}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{j}_1)\} \\
\qquad\qquad\qquad\qquad\qquad\qquad * \left(\begin{array}{c} \tau = (\tau_0 \cdot \tau_1) \\ \wedge \\ \mathsf{i} \mapsto \mathsf{i}_0, \mathsf{i}_1 \end{array}\right) \Bigg\} \exists \tau_0, \tau_1 \\
\quad \{\exists \tau_0, \tau_1.\ \ \tau = (\tau_0 \cdot \tau_1) \wedge (\mathsf{tree}\,(\tau_0 \cdot \tau_1)\,(\mathsf{i})\ *\ \mathsf{tree}\,(\tau_0 \cdot \tau_1)\,(\mathsf{j}))\}\Big) \\
\{\mathsf{tree}\,\tau(\mathsf{i})\ *\ \mathsf{tree}\,\tau(\mathsf{j})\}. \\
\text{—}
\end{array}
$$

Representing S-expressions by Dags

For $\tau \in$ S-exps, we define

$$\mathsf{dag}\ \tau\ (i)$$

by:

$$\mathsf{dag}\ a\ (i) \text{ iff } i = a \qquad \text{when } a \text{ is an atom}$$

$$\mathsf{dag}\ (\tau_0 \cdot \tau_1)\ (i) \text{ iff}$$

$$\exists i_0, i_1.\ \ i \mapsto i_0, i_1\ *\ (\mathsf{dag}\ \tau_0\ (i_0) \wedge \mathsf{dag}\ \tau_1\ (i_1)).$$

—

Some Intuitionistic Assertions

Proposition 13 $(1)\ \mathsf{dag}\ \tau\ (i)$ *and* $(2)\ \exists\tau.\ \mathsf{dag}\ \tau\ (i)$ *are intuitionistic assertions.*

PROOF (1) The proof is by induction on the structure of τ, and uses the fact that p is intuitionistic if $p\ *\ \mathbf{true} \Rightarrow p$. If τ is an atom a,

$$\begin{aligned}\mathsf{dag}\ a\ (i)\ *\ \mathbf{true} &\Rightarrow i = a\ *\ \mathbf{true}\\ &\Rightarrow i = a\\ &\Rightarrow \mathsf{dag}\ a\ (i).\end{aligned}$$

since $i = a$ is pure. Otherwise, $\tau = (\tau_0 \cdot \tau_1)$, and

$$\begin{aligned}&\mathsf{dag}\ (\tau_0 \cdot \tau_1)\ (i)\ *\ \mathbf{true}\\ &\quad\Rightarrow \exists i_0, i_1.\ \ i \mapsto i_0, i_1\ *\ (\mathsf{dag}\ \tau_0\ (i_0) \wedge \mathsf{dag}\ \tau_1\ (i_1))\ *\ \mathbf{true}\\ &\quad\Rightarrow \exists i_0, i_1.\ \ i \mapsto i_0, i_1\ *\\ &\qquad((\mathsf{dag}\ \tau_0\ (i_0)\ *\ \mathbf{true}) \wedge (\mathsf{dag}\ \tau_1\ (i_1)\ *\ \mathbf{true}))\\ &\quad\Rightarrow \exists i_0, i_1.\ \ i \mapsto i_0, i_1\ *\ (\mathsf{dag}\ \tau_0\ (i_0) \wedge \mathsf{dag}\ \tau_1\ (i_1))\\ &\quad\Rightarrow \mathsf{dag}\ (\tau_0 \cdot \tau_1)\ (i),\end{aligned}$$

by the induction hypothesis for τ_0 and τ_1.

(2) $\exists\tau.\ \mathsf{dag}\ \tau\ (\mathsf{i})$ is intuitionistic, since

$$\begin{aligned}(\exists\tau.\ \mathsf{dag}\ \tau\ (i))\ *\ \mathbf{true} &\Rightarrow \exists\tau.\ (\mathsf{dag}\ \tau\ (i)\ *\ \mathbf{true})\\ &\Rightarrow \exists\tau.\ \mathsf{dag}\ \tau\ (\mathsf{i}).\end{aligned}$$

END OF PROOF

—

Some Supported Assertions

Proposition 14 (1) *For all* i, τ_0, τ_1, h_0, h_1, *if* $h_0 \cup h_1$ *is a function, and*

$$[\,\mathsf{i}\colon i \mid \tau\colon \tau_0\,], h_0 \vDash \mathsf{dag}\ \tau\ (\mathsf{i}) \quad \textit{and} \quad [\,\mathsf{i}\colon i \mid \tau\colon \tau_1\,], h_1 \vDash \mathsf{dag}\ \tau\ (\mathsf{i}),$$

then $\tau_0 = \tau_1$ *and*

$$[\,\mathsf{i}\colon i \mid \tau\colon \tau_0\,], h_0 \cap h_1 \vDash \mathsf{dag}\ \tau\ (\mathsf{i}).$$

(2) $\mathsf{dag}\ \tau\ \mathsf{i}$ *is a supported assertion.*
(3) $\exists\tau.\ \mathsf{dag}\ \tau\ (\mathsf{i})$ *is a supported assertion.*

PROOF We first note that: (a) When a is an atom, $[\,\mathsf{i}\colon i \mid \tau\colon a\,], h \vDash \mathsf{dag}\ \tau\ (\mathsf{i})$ iff $i = a$.

(b) $[\,\mathsf{i}\colon i \mid \tau\colon (\tau_l \cdot \tau_r)\,], h \vDash \mathsf{dag}\ \tau\ (\mathsf{i})$ iff i is not an atom and there are i_l, i_r, and h' such that

$$h = [\,i\colon i_l \mid i+1\colon i_r\,] \cdot h'$$
$$[\,\mathsf{i}\colon i_l \mid \tau\colon \tau_l\,], h' \vDash \mathsf{dag}\ \tau\ (\mathsf{i})$$
$$[\,\mathsf{i}\colon i_r \mid \tau\colon \tau_r\,], h' \vDash \mathsf{dag}\ \tau\ (\mathsf{i}).$$

(1) The proof is by structural induction on τ_0. For the base case, suppose τ_0 is an atom a. Then by (a), $i = a$.

Moreover, if τ_1 were not an atom, then by (b) we would have the contradiction that i is not an atom. Thus τ_1 must be atom a', and by (a), $i = a'$, so that $\tau_0 = \tau_1 = i = a = a'$. Then, also by (a), $[\,\mathsf{i}\colon i \mid \tau\colon \tau_0\,], h \vDash \mathsf{dag}\ \tau\ (\mathsf{i})$ holds for any h.

—

Some Supported Assertions (continued)

For the induction step suppose $\tau_0 = (\tau_{0l} \cdot \tau_{0r})$. Then by (b), i is not an atom, and there are i_{0l}, i_{0r}, and h'_0 such that

$$h_0 = [\, i\colon i_{0l} \mid i+1\colon i_{0r} \,] \cdot h'_0$$
$$[\, \mathsf{i}\colon i_{0l} \mid \tau\colon \tau_{0l} \,], h'_0 \models \mathsf{dag}\ \tau\ (\mathsf{i})$$
$$[\, \mathsf{i}\colon i_{0r} \mid \tau\colon \tau_{0r} \,], h'_0 \models \mathsf{dag}\ \tau\ (\mathsf{i}).$$

Moreover, if τ_1 were an atom, then by (a) we would have the contradiction that i is an atom. Thus, τ_1 must have the form $(\tau_{1l} \cdot \tau_{1r})$, so that by (b) there are i_{1l}, i_{1r}, and h'_1 such that

$$h_1 = [\, i\colon i_{1l} \mid i+1\colon i_{1r} \,] \cdot h'_1$$
$$[\, \mathsf{i}\colon i_{1l} \mid \tau\colon \tau_{1l} \,], h'_1 \models \mathsf{dag}\ \tau\ (\mathsf{i})$$
$$[\, \mathsf{i}\colon i_{1r} \mid \tau\colon \tau_{1r} \,], h'_1 \models \mathsf{dag}\ \tau\ (\mathsf{i}).$$

Since $h_0 \cup h_1$ is a function, h_0 and h_1 must map i and $i+1$ into the same values. Thus $[\, i\colon i_{0l} \mid i+1\colon i_{0r} \,] = [\, i\colon i_{1l} \mid i+1\colon i_{1r} \,]$, so that $i_{0l} = i_{1l}$ and $i_{0r} = i_{1r}$, and also,

$$h_0 \cap h_1 = [\, i\colon i_{0l} \mid i+1\colon i_{0r} \,] \cdot (h'_0 \cap h'_1).$$

—

Then, since

$[\,\mathsf{i}\colon i_{0l} \mid \tau\colon \tau_{0l}\,], h'_0 \vDash \mathsf{dag}\,\tau\ (\mathsf{i})$ and $[\,\mathsf{i}\colon i_{1l} \mid \tau\colon \tau_{1l}\,], h'_1 \vDash \mathsf{dag}\,\tau\ (\mathsf{i})$, the induction hypothesis for τ_{0l} gives

$$\tau_{0l} = \tau_{1l} \quad \text{and} \quad [\,\mathsf{i}\colon i_{0l} \mid \tau\colon \tau_{0l}\,], h'_0 \cap h'_1 \vDash \mathsf{dag}\,\tau\ (\mathsf{i}),$$

and the induction hypothesis for τ_{0r} gives

$$\tau_{0r} = \tau_{1r} \quad \text{and} \quad [\,\mathsf{i}\colon i_{0r} \mid \tau\colon \tau_{0r}\,], h'_0 \cap h'_1 \vDash \mathsf{dag}\,\tau\ (\mathsf{i}).$$

Thus, (b) gives

$$[\,\mathsf{i}\colon i \mid \tau\colon (\tau_{0l}{\cdot}\tau_{0r})\,], h_0 \cap h_1 \vDash \mathsf{dag}\,\tau\ (\mathsf{i}),$$

which, with $\tau_0 = (\tau_{0l}{\cdot}\tau_{0r}) = (\tau_{1l}{\cdot}\tau_{1r}) = \tau_1$, establishes (1).

—

(2) Since τ and i are the only free variables in $\mathsf{dag}\,\tau\,(\mathsf{i})$, we can regard s and $[\,\mathsf{i}\colon i \mid \tau\colon \tau\,]$, where $i = s(\mathsf{i})$ and $\tau = s(\tau)$, as equivalent stores. Then (2) follows since $h_0 \cap h_1$ is a subset of both h_0 and h_1.

(3) Since i is the only free variable in $\exists\tau.\ \mathsf{dag}\,\tau\,(\mathsf{i})$, we can regard s and $[\,\mathsf{i}\colon i\,]$, where $i = s(\mathsf{i})$, as equivalent stores. Then we can use the semantic equation for the existential quantifier to show that there are S-expressions τ_0 and τ_1 such that the assumptions for (1) hold. Then (1) and the semantic equation for existentials shows that $[\,\mathsf{i}\colon i\,], h_0 \cap h_1 \models \mathsf{dag}\,\tau\,(\mathsf{i})$, and (3) follows since $h_0 \cap h_1$ is a subset of both h_0 and h_1. END OF PROOF

—

Precise Versions of dag

We can use the "precising" operation,

$$\mathsf{Pr}\ p \stackrel{\mathsf{def}}{=} p \wedge \neg(p\ *\ \neg\,\mathbf{emp}),$$

to convert $\mathsf{dag}\ \tau\ (\mathsf{i})$ and $\exists\tau.\ \mathsf{dag}\ \tau\ (\mathsf{i})$ into the precise assertions

$$\mathsf{dag}\ \tau\ (\mathsf{i}) \wedge \neg(\mathsf{dag}\ \tau\ (\mathsf{i})\ *\ \neg\,\mathbf{emp})$$
$$(\exists\tau.\ \mathsf{dag}\ \tau\ (\mathsf{i})) \wedge \neg((\exists\tau.\ \mathsf{dag}\ \tau\ (\mathsf{i}))\ *\ \neg\,\mathbf{emp}),$$

each of which asserts that the heap contains the dag at i and nothing else.

—

A Problem

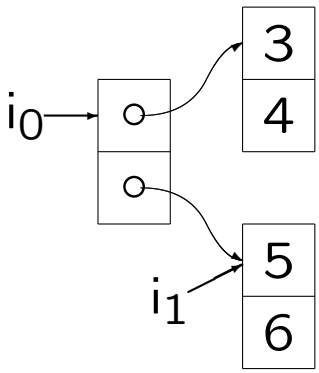
Suppose we wish to prove that

$$\{\mathsf{dag}\ \tau(\mathsf{i})\}\ \mathsf{copytree}(\mathsf{j};\mathsf{i})\{\tau\}\ \{\mathsf{dag}\ \tau(\mathsf{i})\ *\ \mathsf{tree}\ \tau(\mathsf{j})\}$$

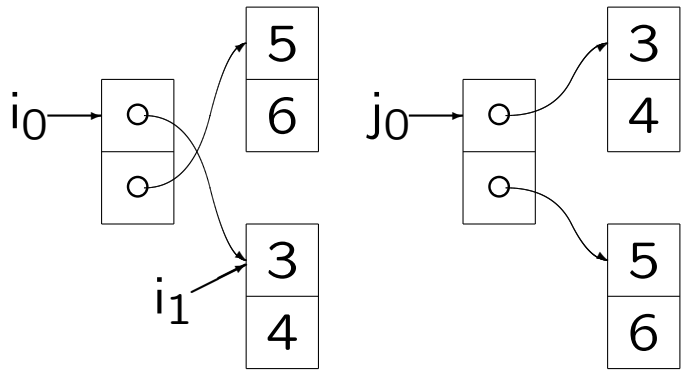
Then, we must use this specification as a hypothesis in proving that the first recursive call in the procedure body satisfies:

$$\{\mathsf{i} \mapsto \mathsf{i}_0, \mathsf{i}_1\ *\ (\mathsf{dag}\ \tau_0(\mathsf{i}_0) \wedge \mathsf{dag}\ \tau_1(\mathsf{i}_1))\}$$
$$\mathsf{copytree}(\mathsf{j}_0;\mathsf{i}_0)\{\tau_0\}$$
$$\{\mathsf{i} \mapsto \mathsf{i}_0, \mathsf{i}_1\ *\ (\mathsf{dag}\ \tau_0(\mathsf{i}_0) \wedge \mathsf{dag}\ \tau_1(\mathsf{i}_1))\ *\ \mathsf{tree}\ \tau_0(\mathsf{j}_0)\}.$$

But the hypothesis is not strong enough to imply this. For example, suppose $\tau_0 = ((3 \cdot 4) \cdot (5 \cdot 6))$ and $\tau_1 = (5 \cdot 6)$. Then $\mathsf{copytree}(\mathsf{j}_0;\mathsf{i}_0)$ might change the state from



into



where $\mathsf{dag}\ \tau_1\ (\mathsf{i}_1)$ is false.

—

Possible Solutions

1. Introduce ghost variables denoting heaps, e.g.

$$\{\mathbf{this}(\mathsf{h}_0) \wedge \mathsf{dag}\ \tau(\mathsf{i})\}\ \mathsf{copytree}(\mathsf{j};\mathsf{i})\{\tau, \mathsf{h}_0\}\ \{\mathbf{this}(\mathsf{h}_0) * \mathsf{tree}\ \tau(\mathsf{j})\}$$

2. Introduce ghost variables denoting assertions, e.g.

$$\{\mathsf{p} \wedge \mathsf{dag}\ \tau(\mathsf{i})\}\ \mathsf{copytree}(\mathsf{j};\mathsf{i})\{\tau, \mathsf{p}\}\ \{\mathsf{p} * \mathsf{tree}\ \tau(\mathsf{j})\}$$

3. Introduce fractional permissions. Then one could define an assertion $\mathsf{passdag}\ \tau(\mathsf{i})$ describing a read-only heap containing a dag, and use it to specify:

$$\{\mathsf{passdag}\ \tau(\mathsf{i})\}\ \mathsf{copytree}(\mathsf{j};\mathsf{i})\{\tau\}\ \{\mathsf{passdag}\ \tau(\mathsf{i}) * \mathsf{tree}\ \tau(\mathsf{j})\}.$$

We will explore the second approach.

—

Assertion Variables

We extend the concept of state to include an *assertion store* mapping assertion variables into properties of heaps:

$$\mathsf{AStores}_A = A \to (\mathsf{Heaps} \to \mathbf{B})$$
$$\mathsf{States}_{AV} = \mathsf{AStores}_A \times \mathsf{Stores}_V \times \mathsf{Heaps},$$

where A denotes a finite set of *assertion variables*.

Assertion stores have no effect on the execution of commands, but they affect the meaning of assertions. Thus we write

$$as, s, h \models p$$

(instead of $s, h \models p$) to indicate that the state as, s, h *satisfies* p.

Then, when an assertion variable is used as an assertion:

$$as, s, h \models a \text{ iff } \ as(a)(h).$$

—

The Substitution Law for Assertions, Revisited

Proposition 15 *Suppose p is an assertion, and let δ abbreviate the substitution*

$$a_1 \to p_1, \ldots, a_m \to p_m, v_1 \to e_1, \ldots, v_n \to e_n,$$

Then let s be a store such that

$$(\mathsf{FV}(p) - \{v_1, \ldots, v_n\}) \cup \mathsf{FV}(p_1, \ldots, p_m, e_1, \ldots, e_n) \subseteq \mathsf{dom}\, s,$$

and let as be an assertion store such that

$$(\mathsf{AV}(p) - \{a_1, \ldots, a_m\}) \cup \mathsf{AV}(p_1, \ldots, p_m) \subseteq \mathsf{dom}\, as,$$

(where $\mathsf{AV}(p)$ is the set of assertion variables in p). Then let

$$\widehat{s} = [\, s \mid v_1 \colon [\![e_1]\!]_{\mathsf{exp}} s \mid \ldots \mid v_n \colon [\![e_n]\!]_{\mathsf{exp}} s \,]$$
$$\widehat{as} = [\, as \mid a_1 \colon \lambda h.\, (as, s, h \models p_1) \mid \ldots \mid a_m \colon \lambda h.\, (as, s, h \models p_m) \,]$$

Then

$$as, s, h \models (p/\delta) \text{ iff } \widehat{as}, \widehat{s}, h \models p.$$

—

Specifications Revisited

The definition of Hoare triples remains unchanged, except that one uses — and quantifies over — the new enriched notion of states. Command execution neither depends upon nor alters the new assertion-store component of these states.

—

The Substitution Rules Revisited

- Substitution (SUB)

$$\frac{\{p\}\ c\ \{q\}}{\{p/\delta\}\ (c/\delta)\ \{q/\delta\},}$$

where δ is the substitution

$$\delta = a_1 \to p_1, \ldots, a_m \to p_m, v_1 \to e_1, \ldots, v_n \to e_n;$$

$a_1, \ldots, a_m$ are the assertion variables occurring in p or q; $v_1, \ldots, v_n$ are the variables occurring free in p, c, or q; and, if v_i is modified by c, then e_i is a variable that does not occur free in any other e_j or in any p_j.

In $\{\mathsf{a}\}\ \mathsf{x} := \mathsf{y}\ \{\mathsf{a}\}$, we can substitute $\mathsf{a} \to (\mathsf{y} = \mathsf{z}), \mathsf{x} \to \mathsf{x}, \mathsf{y} \to \mathsf{y}$ to obtain

$$\{\mathsf{y} = \mathsf{z}\}\ \mathsf{x} := \mathsf{y}\ \{\mathsf{y} = \mathsf{z}\},$$

but we cannot substitute $\mathsf{a} \to (\mathsf{x} = \mathsf{z}), \mathsf{x} \to \mathsf{x}, \mathsf{y} \to \mathsf{y}$ to obtain

$$\{\mathsf{x} = \mathsf{z}\}\ \mathsf{x} := \mathsf{y}\ \{\mathsf{x} = \mathsf{z}\}.$$

—

- Substitution (SUBan)

$$\frac{\mathcal{A} \gg \{p\}\ c\ \{q\}}{\{\mathcal{A}\}/\delta \gg \{p/\delta\}\ (c/\delta)\ \{q/\delta\},}$$

where δ is the substitution

$$\delta = a_1 \to p_1, \ldots, a_m \to p_m, v_1 \to e_1, \ldots, v_n \to e_n;$$

$a_1, \ldots, a_m$ are the assertion variables occurring in p or q; $v_1, \ldots, v_n$ are the variables occurring free in p, c, or q; and, if v_i is modified by c, then e_i is a variable that does not occur free in any other e_j or in any p_j.

—

Copying Dags to Trees

We will prove that the procedure

$$
\begin{array}{l}
\mathsf{copytree}(\mathsf{j};\mathsf{i}) = \\
\quad \mathbf{if}\ \mathsf{isatom}(\mathsf{i})\ \mathbf{then}\ \mathsf{j} := \mathsf{i}\ \mathbf{else} \\
\quad\quad \mathbf{newvar}\ \mathsf{i}_0, \mathsf{i}_1, \mathsf{j}_0, \mathsf{j}_1\ \mathbf{in} \\
\quad\quad\quad \big(\mathsf{i}_0 := [\mathsf{i}]\ ;\ \mathsf{i}_1 := [\mathsf{i}+1]\ ; \\
\quad\quad\quad \mathbf{copytree}(\mathsf{j}_0;\mathsf{i}_0)\,;\mathbf{copytree}(\mathsf{j}_1;\mathsf{i}_1)\,;\mathsf{j}{:=}\mathbf{cons}(\mathsf{j}_0,\mathsf{j}_1)\big)
\end{array}
$$

satisfies

$$\{\mathsf{p} \wedge \mathsf{dag}\ \tau(\mathsf{i})\}\ \mathsf{copytree}(\mathsf{j};\mathsf{i})\{\tau, \mathsf{p}\}\ \{\mathsf{p}\ *\ \mathsf{tree}\ \tau(\mathsf{j})\}.$$

We can take p to be $\mathsf{dag}\ \tau(\mathsf{i})$, to obtain the specification

$$\{\mathsf{dag}\ \tau(\mathsf{i})\}\ \mathsf{copytree}(\mathsf{j};\mathsf{i})\{\tau, \mathsf{dag}\ \tau(\mathsf{i})\}\ \{\mathsf{dag}\ \tau(\mathsf{i})\ *\ \mathsf{tree}\ \tau(\mathsf{j})\},$$

but this is too weak to serve as a recursion hypothesis.

$$
\begin{array}{l}
\{\mathsf{p} \wedge \mathsf{dag}\ \tau(\mathsf{i})\}\ \mathsf{copytree}(\mathsf{j};\mathsf{i})\{\tau, \mathsf{p}\}\ \{\mathsf{p}\ *\ \mathsf{tree}\ \tau(\mathsf{j})\} \vdash \\
\quad \{\mathsf{p} \wedge \mathsf{dag}\ \tau(\mathsf{i})\} \\
\quad \mathbf{if}\ \mathsf{isatom}(\mathsf{i})\ \mathbf{then} \\
\quad\quad \{\mathsf{p} \wedge \mathsf{isatom}(\tau) \wedge \tau = \mathsf{i}\} \\
\quad\quad \{\mathsf{p}\ *\ (\mathsf{isatom}(\tau) \wedge \tau = \mathsf{i} \wedge \mathbf{emp})\} \\
\quad\quad \mathsf{j} := \mathsf{i} \\
\quad\quad \{\mathsf{p}\ *\ (\mathsf{isatom}(\tau) \wedge \tau = \mathsf{j} \wedge \mathbf{emp})\} \\
\quad\quad\quad \vdots
\end{array}
$$

—

$$
\begin{array}{l}
\quad \vdots \\
\mathbf{else} \\
\{\exists \tau_0, \tau_1.\ \ \tau = (\tau_0 \cdot \tau_1) \wedge \mathsf{p} \wedge \mathsf{dag}\,(\tau_0 \cdot \tau_1)(\mathsf{i})\} \\
\left.\begin{array}{l}
\mathbf{newvar}\ \mathsf{i}_0, \mathsf{i}_1, \mathsf{j}_0, \mathsf{j}_1\ \mathbf{in}\Big(\mathsf{i}_0 := [\mathsf{i}]\ ;\ \mathsf{i}_1 := [\mathsf{i}+1]\ ; \\
\quad \{\mathsf{p} \wedge (\mathsf{i} \mapsto \mathsf{i}_0, \mathsf{i}_1\ *\ (\mathsf{dag}\,\tau_0\,(\mathsf{i}_0) \wedge \mathsf{dag}\,\tau_1\,(\mathsf{i}_1)))\} \\
\quad \{\mathsf{p} \wedge (\mathbf{true}\ *\ (\mathsf{dag}\,\tau_0\,(\mathsf{i}_0) \wedge \mathsf{dag}\,\tau_1\,(\mathsf{i}_1)))\} \\
\quad \{\mathsf{p} \wedge ((\mathbf{true} * \mathsf{dag}\,\tau_0\,(\mathsf{i}_0)) \wedge (\mathbf{true} * \mathsf{dag}\,\tau_1\,(\mathsf{i}_1)))\} \\
\quad \{\mathsf{p} \wedge \mathsf{dag}\,\tau_1\,(\mathsf{i}_1) \wedge \mathsf{dag}\,\tau_0\,(\mathsf{i}_0)\} \qquad (*) \\
\quad \mathsf{copytree}(\mathsf{j}_0; \mathsf{i}_0)\{\tau_0, \mathsf{p} \wedge \mathsf{dag}\,\tau_1\,(\mathsf{i}_1)\} \\
\quad \{(\mathsf{p} \wedge \mathsf{dag}\,\tau_1\,(\mathsf{i}_1))\ *\ \mathsf{tree}\,\tau_0(\mathsf{j}_0)\} \\
\quad \left.\begin{array}{l} \{\mathsf{p} \wedge \mathsf{dag}\,\tau_1(\mathsf{i}_1)\} \\ \mathsf{copytree}(\mathsf{j}_1; \mathsf{i}_1)\{\tau_1, \mathsf{p}\} \\ \{\mathsf{p}\ *\ \mathsf{tree}\,\tau_1(\mathsf{j}_1)\} \end{array}\right\} *\ \mathsf{tree}\,\tau_0\,(\mathsf{j}_0) \\
\quad \{\mathsf{p}\ *\ \mathsf{tree}\,\tau_0\,(\mathsf{j}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{j}_1)\} \\
\quad \mathsf{j} := \mathbf{cons}(\mathsf{j}_0, \mathsf{j}_1) \\
\quad \{\mathsf{p}\ *\ \mathsf{j} \mapsto \mathsf{j}_0, \mathsf{j}_1\ *\ \mathsf{tree}\,\tau_0\,(\mathsf{j}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{j}_1)\} \\
\qquad\qquad\qquad\qquad\qquad\qquad * \begin{pmatrix} \tau = (\tau_0 \cdot \tau_1) \\ \wedge \\ \mathbf{emp} \end{pmatrix}
\end{array}\right\} \exists \tau_0, \tau_1 \\
\{\exists \tau_0, \tau_1.\ \ (\tau = (\tau_0 \cdot \tau_1) \wedge \mathsf{p})\ *\ \mathsf{tree}\,(\tau_0 \cdot \tau_1)\,(\mathsf{j})\}\Big) \\
\{\mathsf{p}\ *\ \mathsf{tree}\,\tau(\mathsf{j})\}. \\
\text{—}
\end{array}
$$

Substitution of an S-expression for an Atom

$$
\begin{aligned}
a/a \to \tau' &= \tau' \\
b/a \to \tau' &= b \qquad \text{when } b \in \mathsf{Atoms} - \{a\} \\
(\tau_0 \cdot \tau_1)/a \to \tau' &= ((\tau_0/a \to \tau') \cdot (\tau_1/a \to \tau')).
\end{aligned}
$$

Substitution by Copying

$\mathsf{subst}(\mathsf{i}; \mathsf{a}, \mathsf{j}) =$

$\quad \mathbf{if}\ \mathsf{isatom}(\mathsf{i})\ \mathbf{then\ if}\ \mathsf{i} = \mathsf{a}\ \mathbf{then}\ \mathsf{copytree}(\mathsf{i}\ ;\ \mathsf{j})\ \mathbf{else\ skip}$

$\quad \mathbf{else\ newvar}\ \mathsf{i}_0, \mathsf{i}_1\ \mathbf{in}\ (\mathsf{i}_0 := [\mathsf{i}]\ ;\ \mathsf{i}_1 := [\mathsf{i} + 1]\ ;$

$\quad\quad \mathsf{subst}(\mathsf{i}_0; \mathsf{a}, \mathsf{j})\ ;\ \mathsf{subst}(\mathsf{i}_1; \mathsf{a}, \mathsf{j})\ ;\ [\mathsf{i}] := \mathsf{i}_0\ ;\ [\mathsf{i} + 1] := \mathsf{i}_1).$

satisfies

$$
\begin{aligned}
&\{\mathsf{tree}\ \tau\ (\mathsf{i})\ *\ \mathsf{dag}\ \tau'\ (\mathsf{j})\} \\
&\mathsf{subst}(\mathsf{i}; \mathsf{a}, \mathsf{j})\{\tau, \tau'\} \\
&\{\mathsf{tree}\ (\tau/\mathsf{a} \to \tau')\ (\mathsf{i})\ *\ \mathsf{dag}\ \tau'\ (\mathsf{j})\}.
\end{aligned}
$$

—

Substitution by Copying (continued)

since (writing D for dag τ' (j))

$$\{\mathsf{tree}\,\tau\,(\mathsf{i})\ *\ D\}\ \mathsf{subst}(\mathsf{i};\mathsf{a},\mathsf{j})\{\tau,\tau'\}\ \{\mathsf{tree}\,(\tau/\mathsf{a}\to\tau')\,(\mathsf{i})\ *\ D\}\ \vdash$$

$$\begin{array}{l}
\quad \{\mathsf{tree}\,\tau\,(\mathsf{i})\ *\ D\}\\
\quad \mathbf{if}\ \mathsf{isatom}(\mathsf{i})\ \mathbf{then}\\
\qquad \{(\mathsf{isatom}(\tau)\wedge\tau=\mathsf{i}\wedge\mathbf{emp})\ *\ D\}\\
\qquad \mathbf{if}\ \mathsf{i}=\mathsf{a}\ \mathbf{then}\\
\qquad\quad \{(\mathsf{isatom}(\tau)\wedge\tau=\mathsf{a}\wedge\mathbf{emp})\ *\ D\}\\
\qquad\quad \{((\tau/\mathsf{a}\to\tau')=\tau'\wedge\mathbf{emp})\ *\ D\}\\
\qquad\quad \left.\begin{array}{l}\{D\}\\ \mathsf{copytree}(\mathsf{i};\mathsf{j})\{\tau'\}\\ \{D\ *\ \mathsf{tree}\,\tau'(\mathsf{i})\}\end{array}\right\}\ *\ ((\tau/\mathsf{a}\to\tau')=\tau'\wedge\mathbf{emp})\\
\qquad\quad \{\mathsf{tree}\,(\tau/\mathsf{a}\to\tau')\,(\mathsf{i})\ *\ D\}\\
\qquad \mathbf{else}\\
\qquad\quad \{(\mathsf{isatom}(\tau)\wedge\tau\neq\mathsf{a}\wedge\tau=\mathsf{i}\wedge\mathbf{emp})\ *\ D\}\\
\qquad\quad \{((\tau/\mathsf{a}\to\tau')=\tau\wedge\mathsf{isatom}(\tau)\wedge\tau=\mathsf{i}\wedge\mathbf{emp})\ *\ D\}\\
\qquad\quad \mathbf{skip}\\
\qquad\quad \{\mathsf{tree}\,(\tau/\mathsf{a}\to\tau')\,(\mathsf{i})\ *\ D\}\\
\qquad\quad \vdots
\end{array}$$

—

$$
\begin{array}{l}
\vdots \\
\textbf{else} \\
\{\exists \tau_0, \tau_1, \mathsf{i}_0, \mathsf{i}_1.\ \tau = (\tau_0 \cdot \tau_1) \wedge \\
\qquad (\mathsf{i} \mapsto \mathsf{i}_0, \mathsf{i}_1\ *\ \mathsf{tree}\,\tau_0\,(\mathsf{i}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{i}_1)\ *\ D)\} \\
\textbf{newvar}\ \mathsf{i}_0, \mathsf{i}_1\ \textbf{in}\ \Big(\mathsf{i}_0 := [\mathsf{i}]\ ;\ \mathsf{i}_1 := [\mathsf{i}+1]\ ; \\
\quad \left.\begin{array}{l}
\{\mathsf{tree}\,\tau_0\,(\mathsf{i}_0)\ *\ \mathsf{tree}\,\tau_1\,(\mathsf{i}_1)\ *\ D)\} \\
\quad \left.\begin{array}{l}
\{\mathsf{tree}\,\tau_0\,(\mathsf{i}_0)\ *\ D\} \\
\mathsf{subst}(\mathsf{i}_0; \mathsf{a}, \mathsf{j})\{\tau_0, \tau'\}\ ; \\
\{\mathsf{tree}\,(\tau_0/\mathsf{a} \to \tau')\,(\mathsf{i}_0) * D\}
\end{array}\right\} * \mathsf{tree}\,\tau_1\,(\mathsf{i}_1) \\
\quad \left.\begin{array}{l}
\{\mathsf{tree}\,\tau_1\,(\mathsf{i}_1)\ *\ D\} \\
\mathsf{subst}(\mathsf{i}_1; \mathsf{a}, \mathsf{j})\{\tau_1, \tau'\}\ ; \\
\{\mathsf{tree}\,(\tau_1/\mathsf{a} \to \tau')\,(\mathsf{i}_1) * D\}
\end{array}\right\} * \mathsf{tree}\,(\tau_0/\mathsf{a} \to \tau')\,(\mathsf{i}_0) \\
\quad \{\mathsf{tree}\,(\tau_0/\mathsf{a} \to \tau')\,(\mathsf{i}_0)\ *\ \mathsf{tree}\,(\tau_1/\mathsf{a} \to \tau')\,(\mathsf{i}_1)\ *\ D\}
\end{array}\right\} \exists \tau_0, \tau_1 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad * \begin{pmatrix} \tau = (\tau_0 \cdot \tau_1) \\ \wedge \\ \mathsf{i} \mapsto -, - \end{pmatrix} \\
\quad [\mathsf{i}] := \mathsf{i}_0\ ;\ [\mathsf{i}+1] := \mathsf{i}_1 \\
\quad \{\tau = (\tau_0 \cdot \tau_1) \wedge (\mathsf{i} \mapsto \mathsf{i}_0, \mathsf{i}_1\ * \\
\qquad \mathsf{tree}\,(\tau_0/\mathsf{a} \to \tau')\,(\mathsf{i}_0)\ *\ \mathsf{tree}\,(\tau_1/\mathsf{a} \to \tau')\,(\mathsf{i}_1)\ *\ D)\} \Big) \\
\{\exists \tau_0, \tau_1.\ \tau = (\tau_0 \cdot \tau_1) \wedge \\
\qquad (\mathsf{tree}\,((\tau_0/\mathsf{a} \to \tau') \cdot (\tau_1/\mathsf{a} \to \tau'))\,(\mathsf{i})\ *\ D)\}\Big) \\
\{\mathsf{tree}\,(\tau/\mathsf{a} \to \tau')\,(\mathsf{i})\ *\ D\}. \\
\text{—}
\end{array}
$$

More Derived Inference Rules

To derive: $\dfrac{}{\mathbf{emp} \Rightarrow (p \mathrel{-\!\!*} p)}$

1. $(\mathbf{emp} * p) \Rightarrow p$ $(p * \mathbf{emp} \Rightarrow p)$
2. $\mathbf{emp} \Rightarrow (p \mathrel{-\!\!*} p)$ (currying, 1)

—

More Derived Inference Rules (continued)

To derive: $$\frac{p \Rightarrow q \qquad p \Rightarrow r}{p \Rightarrow (q \wedge r)}$$

1. $p \Rightarrow q$ (assumption)
2. $p \Rightarrow r$ (assumption)
3. $q \Rightarrow (r \Rightarrow (q \wedge r))$ $(p \Rightarrow (q \Rightarrow (p \wedge q)))$
4. $p \Rightarrow (r \Rightarrow (q \wedge r))$ (trans impl, 1, 3)
5. $(p \Rightarrow (r \Rightarrow (q \wedge r))) \Rightarrow ((p \Rightarrow r) \Rightarrow (p \Rightarrow (q \wedge r)))$
   $((p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r)))$
6. $(p \Rightarrow r) \Rightarrow (p \Rightarrow (q \wedge r))$ (modus ponens, 4, 5)
7. $p \Rightarrow (q \wedge r)$ (modus ponens, 2, 6)

—

More Derived Inference Rules (continued)

To derive: $\dfrac{}{(p \mathrel{-\!\!*} i_0) * (p \mathrel{-\!\!*} i_1) \Rightarrow (p \mathrel{-\!\!*} (i_0 \wedge i_1))}$

when i_0 and i_1 are intuitionistic.

1. $(p \mathrel{-\!\!*} i_0) \Rightarrow (p \mathrel{-\!\!*} i_0)$ $(p \Rightarrow p)$
2. $(p \mathrel{-\!\!*} i_0) * p \Rightarrow i_0$ (decurrying, 1)
3. $(p \mathrel{-\!\!*} i_1) \Rightarrow \mathbf{true}$ $(p \Rightarrow \mathbf{true})$
4. $(p \mathrel{-\!\!*} i_0) * (p \mathrel{-\!\!*} i_1) * p \Rightarrow i_0 * \mathbf{true}$
   (monotonicity, 2, 3)
5. $i_0 * \mathbf{true} \Rightarrow i_0$ $(i * p \Rightarrow i)$
6. $(p \mathrel{-\!\!*} i_0) * (p \mathrel{-\!\!*} i_1) * p \Rightarrow i_0$ (trans impl, 4, 5)
7. $(p \mathrel{-\!\!*} i_0) * (p \mathrel{-\!\!*} i_1) * p \Rightarrow i_1$ (similarly)
8. $(p \mathrel{-\!\!*} i_0) * (p \mathrel{-\!\!*} i_1) * p \Rightarrow (i_0 \wedge i_1)$ (above, 6, 7)
9. $(p \mathrel{-\!\!*} i_0) * (p \mathrel{-\!\!*} i_1) \Rightarrow (p \mathrel{-\!\!*} (i_0 \wedge i_1))$ (currying, 8)

—

More Derived Inference Rules (continued)

To derive: $\dfrac{}{p * (q \mathrel{-\!\!*} r) \Rightarrow (q \mathrel{-\!\!*} (p * r))}$

1. $p \Rightarrow p$ $(p \Rightarrow p)$
2. $(q \mathrel{-\!\!*} r) \Rightarrow (q \mathrel{-\!\!*} r)$ $(p \Rightarrow p)$
3. $(q \mathrel{-\!\!*} r) * q \Rightarrow r)$ (decurrying, 2)
4. $p * (q \mathrel{-\!\!*} r) * q \Rightarrow p * r)$ (monotonicity, 1, 3)
5. $p * (q \mathrel{-\!\!*} r) \Rightarrow (q \mathrel{-\!\!*} (p * r))$ (currying, 4)

—

Substitution without Copying

$$\begin{array}{l}\mathsf{subst2}(\mathsf{i};\mathsf{a},\mathsf{j}) = \\ \quad \mathbf{if}\ \mathsf{isatom}(\mathsf{i})\ \mathbf{then\ if}\ \mathsf{i} = \mathsf{a}\ \mathbf{then}\ \mathsf{i} := \mathsf{j}\ \mathbf{else\ skip} \\ \quad \mathbf{else\ newvar}\ \mathsf{i}_0, \mathsf{i}_1\ \mathbf{in}\ (\mathsf{i}_0 := [\mathsf{i}]\ ;\ \mathsf{i}_1 := [\mathsf{i}+1]\ ; \\ \qquad \mathsf{subst2}(\mathsf{i}_0;\mathsf{a},\mathsf{j})\ ;\ \mathsf{subst2}(\mathsf{i}_1;\mathsf{a},\mathsf{j})\ ;\ [\mathsf{i}] := \mathsf{i}_0\ ;\ [\mathsf{i}+1] := \mathsf{i}_1)\end{array}$$

satisfies

$$\begin{array}{l}\{\mathsf{tree}\,\tau\,(\mathsf{i})\ *\ \mathsf{dag}\,\tau'\,(\mathsf{j})\} \\ \left.\begin{array}{|l|}\hline \{\mathsf{tree}\,\tau\,(\mathsf{i})\} \\ \mathsf{subst2}(\mathsf{i};\mathsf{a},\mathsf{j})\{\tau,\tau'\} \\ \{\mathsf{dag}\,\tau'\,(\mathsf{j}) \mathrel{-\!\!*} \mathsf{dag}\,(\tau/\mathsf{a} \to \tau')\,(\mathsf{i})\} \\ \hline\end{array}\right\} *\ \mathsf{dag}\,\tau'\,(\mathsf{j})\} \\ \{(\mathsf{dag}\,\tau'\,(\mathsf{j}) \mathrel{-\!\!*} \mathsf{dag}\,(\tau/\mathsf{a} \to \tau')\,(\mathsf{i}))\ *\ \mathsf{dag}\,\tau'\,(\mathsf{j})\} \\ \{\mathsf{dag}\,(\tau/\mathsf{a} \to \tau')\,(\mathsf{i}) \wedge (\mathsf{dag}\,\tau'\,(\mathsf{j})\ *\ \mathbf{true})\} \\ \{\mathsf{dag}\,(\tau/\mathsf{a} \to \tau')\,(\mathsf{i}) \wedge \mathsf{dag}\,\tau'\,(\mathsf{j})\},\end{array}$$

where we have used

$$(q \mathrel{-\!\!*} p)\ *\ q \Rightarrow p \wedge (q\ *\ \mathbf{true})$$

$$i\ *\ \mathbf{true} \Rightarrow i \quad \text{when } i \text{ is intuitionistic.}$$

We will prove the boxed specification.

—

Substitution without Copying (continued)

Let D abbreviate $\mathsf{dag}\ \tau'\ (\mathsf{j})$. Then the body of the procedure subst2 will meet the specification

$$
\begin{array}{l}
\{\mathsf{tree}\ \tau\ (\mathsf{i})\}\ \mathsf{subst2}(\mathsf{i};\mathsf{a},\mathsf{j})\{\tau,\tau'\}\ \{D \mathrel{-\!\!*} \mathsf{dag}\ (\tau/\mathsf{a} \to \tau')\ (\mathsf{i})\} \vdash \\
\quad \{\mathsf{tree}\ \tau\ (\mathsf{i})\} \\
\quad \mathbf{if}\ \mathsf{isatom}(\mathsf{i})\ \mathbf{then} \\
\qquad \{\mathsf{isatom}(\tau) \wedge \tau = \mathsf{i} \wedge \mathbf{emp}\} \\
\qquad \mathbf{if}\ \mathsf{i} = \mathsf{a}\ \mathbf{then} \\
\qquad\quad \{\mathsf{isatom}(\tau) \wedge \tau = \mathsf{a} \wedge \mathbf{emp}\} \\
\qquad\quad \{(\tau/\mathsf{a} \to \tau') = \tau' \wedge \mathbf{emp}\} \\
\qquad\quad \{(\tau/\mathsf{a} \to \tau') = \tau' \wedge (D \mathrel{-\!\!*} D)\} \qquad (*) \\
\qquad\quad \{D \mathrel{-\!\!*} \mathsf{dag}\ (\tau/\mathsf{a} \to \tau')\ (\mathsf{j})\} \\
\qquad\quad \mathsf{i} := \mathsf{j} \\
\qquad\quad \{D \mathrel{-\!\!*} \mathsf{dag}\ (\tau/\mathsf{a} \to \tau')\ (\mathsf{i})\} \\
\qquad \mathbf{else} \\
\qquad\quad \{\mathsf{isatom}(\tau) \wedge \tau \neq \mathsf{a} \wedge \tau = \mathsf{i} \wedge \mathbf{emp}\} \\
\qquad\quad \{(\tau/\mathsf{a} \to \tau') = \tau \wedge \mathsf{isatom}(\tau) \wedge \tau = \mathsf{i}\} \\
\qquad\quad \{(\tau/\mathsf{a} \to \tau') = \tau \wedge \mathsf{dag}\ \tau\ (\mathsf{i})\} \\
\qquad\quad \{\mathsf{dag}\ (\tau/\mathsf{a} \to \tau')\ (\mathsf{i})\} \\
\qquad\quad \mathbf{skip} \\
\qquad\quad \{D \mathrel{-\!\!*} \mathsf{dag}\ (\tau/\mathsf{a} \to \tau')\ (\mathsf{i})\} \qquad (*) \\
\qquad\quad \vdots
\end{array}
$$

—

$\vdots$

else

$\{\exists \tau_0, \tau_1, \mathsf{i}_0, \mathsf{i}_1.\ \tau = (\tau_0 \cdot \tau_1) \wedge (\mathsf{i} \mapsto \mathsf{i}_0, \mathsf{i}_1 * \mathsf{tree}\ \tau_0\ (\mathsf{i}_0) * \mathsf{tree}\ \tau_1\ (\mathsf{i}_1))\}$

$\left.\begin{array}{l}
\mathbf{newvar}\ \mathsf{i}_0, \mathsf{i}_1\ \mathbf{in}\ \big(\mathsf{i}_0 := [\mathsf{i}]\ ;\ \mathsf{i}_1 := [\mathsf{i}+1]\ ; \\
\quad \{\mathsf{tree}\ \tau_0\ (\mathsf{i}_0)\ *\ \mathsf{tree}\ \tau_1\ (\mathsf{i}_1)\} \\
\quad \left.\begin{array}{l} \{\mathsf{tree}\ \tau_0\ (\mathsf{i}_0)\} \\ \mathsf{subst2}(\mathsf{i}_0\ ;\ \mathsf{a}, \mathsf{j})\{\tau_0, \tau'\} \\ \{D \mathrel{-\!\!*} \mathsf{dag}\ (\tau_0/\mathsf{a} \to \tau')\ (\mathsf{i}_0)\} \end{array}\right\} * \mathsf{tree}\ \tau_1\ (\mathsf{i}_1) \\
\quad \left.\begin{array}{l} \{\mathsf{tree}\ \tau_1\ (\mathsf{i}_1)\} \\ \mathsf{subst2}(\mathsf{i}_1\ ;\ \mathsf{a}, \mathsf{j})\{\tau_1, \tau'\} \\ \{D \mathrel{-\!\!*} \mathsf{dag}\ (\tau_1/\mathsf{a} \to \tau')\ (\mathsf{i}_1)\} \end{array}\right\} \\
\qquad\qquad * (D \mathrel{-\!\!*} \mathsf{dag}\ (\tau_0/\mathsf{a} \to \tau')\ (\mathsf{i}_0)) \\
\quad \{(D \mathrel{-\!\!*} \mathsf{dag}\ (\tau_0/\mathsf{a} \to \tau')\ (\mathsf{i}_0))\ * \\
\qquad (D \mathrel{-\!\!*} \mathsf{dag}\ (\tau_1/\mathsf{a} \to \tau')\ (\mathsf{i}_1))\} \\
\quad \{D \mathrel{-\!\!*} (\mathsf{dag}\ (\tau_0/\mathsf{a} \to \tau')\ (\mathsf{i}_0) \wedge \\
\qquad \mathsf{dag}\ (\tau_1/\mathsf{a} \to \tau')\ (\mathsf{i}_1))\} \qquad (*) \\
\qquad\qquad * (\tau = (\tau_0 \cdot \tau_1) \wedge \mathsf{i} \mapsto -, -)
\end{array}\right\} \exists \tau_0, \tau_1$

$[\mathsf{i}] := \mathsf{i}_0\ ;\ [\mathsf{i}+1] := \mathsf{i}_1$

$\{\tau = (\tau_0 \cdot \tau_1) \wedge (\mathsf{i} \mapsto i_0, i_1\ *\ (D \mathrel{-\!\!*}$
$\quad (\mathsf{dag}\ (\tau_0/\mathsf{a} \to \tau')\ (\mathsf{i}_0) \wedge \mathsf{dag}\ (\tau_1/\mathsf{a} \to \tau')\ (\mathsf{i}_1))))\}$

$\{\tau = (\tau_0 \cdot \tau_1) \wedge (D \mathrel{-\!\!*} (\mathsf{i} \mapsto i_0, i_1\ *$ $(*)$
$\quad (\mathsf{dag}\ (\tau_0/\mathsf{a} \to \tau')\ (\mathsf{i}_0) \wedge \mathsf{dag}\ (\tau_1/\mathsf{a} \to \tau')\ (\mathsf{i}_1))))\}$

$\{\tau = (\tau_0 \cdot \tau_1) \wedge$
$\quad (D \mathrel{-\!\!*} \mathsf{dag}\ ((\tau_0/\mathsf{a} \to \tau') \cdot (\tau_1/\mathsf{a} \to \tau'))\ (\mathsf{i}))\}$

$\{D \mathrel{-\!\!*} \mathsf{dag}\ (\tau/\mathsf{a} \to \tau')\ (\mathsf{i})\}.$

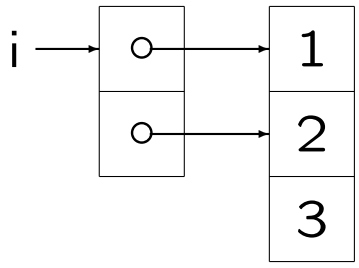
—

Skewed Sharing

Our definition of dag permits *skewed sharing*. For example,

$$\mathsf{dag}\,((1 \cdot 2) \cdot (2 \cdot 3))\,(\mathsf{i})$$

holds when



Skewed sharing is not a problem for the algorithms we have seen so far, which only examine dags while ignoring their sharing structure. But it causes difficulties with algorithms that modify dags or depend upon the sharing structure.

—

A Possible Solution

- We add to the state a mapping ϕ from the domain of the heap to natural numbers, called the *field count*.
- When $x := \mathbf{cons}(e_1, \ldots, e_n)$ sets x to the address a. the field count is extended so that

$$\phi(a) = n \qquad \phi(a+1) = 0 \quad \cdots \quad \phi(a+n-1) = 0.$$

- We introduce the assertion $e \overset{[\widehat{e}]}{\mapsto} e'$, with the meaning

$$\begin{aligned} s, h, \phi \vDash e \overset{[\widehat{e}]}{\mapsto} e' \text{ iff} \\ \mathsf{dom}\, h = \{[\![e]\!]_{\mathsf{exp}} s\} \text{ and } h([\![e]\!]_{\mathsf{exp}} s) = [\![e']\!]_{\mathsf{exp}} s \\ \text{and } \phi([\![e]\!]_{\mathsf{exp}} s) = [\![\widehat{e}]\!]_{\mathsf{exp}} s. \end{aligned}$$

- We also introduce the following abbreviations:

$$e \overset{[\widehat{e}]}{\mapsto} - \overset{\mathsf{def}}{=} \exists x'.\, e \overset{[\widehat{e}]}{\mapsto} x' \quad \text{where } x' \text{ not free in } e \text{ or } \widehat{e}$$

$$e \overset{[\widehat{e}]}{\hookrightarrow} e' \overset{\mathsf{def}}{=} e \overset{[\widehat{e}]}{\mapsto} e' * \mathbf{true}$$

$$e \overset{!}{\mapsto} e_1, \ldots, e_n \overset{\mathsf{def}}{=} e \overset{[n]}{\mapsto} e_1 * e{+}1 \overset{[0]}{\mapsto} e_2 * \cdots * e{+}n{-}1 \overset{[0]}{\mapsto} e_n$$

$$e \overset{!}{\hookrightarrow} e_1, \ldots, e_n \overset{\mathsf{def}}{=} e \overset{[n]}{\hookrightarrow} e_1 * e{+}1 \overset{[0]}{\hookrightarrow} e_2 * \cdots * e{+}n{-}1 \overset{[0]}{\hookrightarrow} e_n$$

$$\text{iff } e \overset{!}{\mapsto} e_1, \ldots, e_n * \mathbf{true}.$$

—

Axiom Schema

$$e \overset{[\mathsf{n}]}{\mapsto} e' \Rightarrow e \mapsto e'$$

$$e \overset{[\mathsf{m}]}{\hookrightarrow} - \wedge e \overset{[\mathsf{n}]}{\hookrightarrow} - \Rightarrow \mathsf{m} = \mathsf{n}$$

$$2 \leq \mathsf{k} \leq \mathsf{n} \wedge e \overset{[\mathsf{n}]}{\hookrightarrow} - \Rightarrow e + \mathsf{k} - 1 \overset{[0]}{\hookrightarrow} -$$

$$e \overset{!}{\hookrightarrow} e_1, \ldots, e_m \wedge e' \overset{!}{\hookrightarrow} e'_1, \ldots, e'_n \wedge e \neq e' \Rightarrow$$
$$e \overset{!}{\mapsto} e_1, \ldots, e_m \ * \ e' \overset{!}{\mapsto} e'_1, \ldots, e'_n \ * \ \mathbf{true}.$$

(The last of these axiom schemas makes it clear that skewed sharing has been prohibited.)

—

Additional Inference Rules

- Allocation: local nonoverwriting form (FCCONSNOL)

$$\frac{}{\{\mathbf{emp}\}\ v := \mathbf{cons}(\overline{e})\ \{v \overset{!}{\mapsto} \overline{e}\}},$$

  where $v \notin \mathsf{FV}(\overline{e})$.

- Mutation: local form (FCMUL)

$$\frac{}{\{e \overset{[\widehat{e}]}{\mapsto} -\}\ [e] := e'\ \{e \overset{[\widehat{e}]}{\mapsto} e'\}}.$$

- Lookup: local nonoverwriting form (FCLKNOL)

$$\frac{}{\{e \overset{[\widehat{e}]}{\mapsto} v''\}\ v := [e]\ \{v = v'' \wedge (e \overset{[\widehat{e}]}{\mapsto} v)\}},$$

  where $v \notin \mathsf{FV}(e, \widehat{e})$.

—

A Problem with Deallocation

If one can deallocate single fields, the use of field counts can be disrupted by deallocating a part of record. For example,

$\mathsf{j} := \mathbf{cons}(1, 2)\ ;\ \mathbf{dispose}\ \mathsf{j} + 1\ ;\ \mathsf{k} := \mathbf{cons}(3, 4)\ ;\ \mathsf{i} := \mathbf{cons}(\mathsf{j}, \mathsf{k})$

could produce skewed sharing if the new record allocated by the second $\mathbf{cons}$ were placed at locations $\mathsf{j}+1$ and $\mathsf{j}+2$. This would make our new axiom schema unsound.

A solution is to replace $\mathbf{dispose}\ e$ with a command $\mathbf{dispose}\ (e, n)$ that disposes of an entire n-field record — and then to require that this record must have been created by an execution of $\mathbf{cons}$ (à la C).

—

New Rules for Deallocation

- The local form (FCDISL)

$$\frac{}{\{e \overset{!}{\mapsto} -^n\}\ \mathbf{dispose}\,(e, n)\ \{\mathbf{emp}\}.}$$

- The global (and backward-reasoning) form (FCDISG)

$$\frac{}{\{(e \overset{!}{\mapsto} -^n)\ *\ r\}\ \mathbf{dispose}\,(e, n)\ \{r\}.}$$

(Here $-^n$ denotes a list of n occurrences of $-$.)

—

Exercise 1

If τ is an S-expression, then $|\tau|$, called the *flattening* of τ, is the sequence defined by:

$$|a| = [a] \quad \text{when } a \text{ is an atom}$$
$$|(t_0 \cdot t_1)| = |\tau_0| \cdot |\tau_1|.$$

Here $[a]$ denotes the sequence whose only element is a, and the "$\cdot$" on the right of the last equation denotes the concatenation of sequences.

Define and prove correct (by an annotated specification of its body) a recursive procedure flatten that mutates a tree denoting an S-expression τ into a singly-linked list segment denoting the flattening of τ. This procedure should not do any allocation or disposal of heap storage. However, since a list segment representing $|\tau|$ contains one more two-cell than a tree representing τ, the procedure should be given as input, in addition to the tree representing τ, a single two-cell, which will become the initial cell of the list segment that is constructed.

More precisely, the procedure should satisfy

$$\{\mathsf{tree}\ \tau\ (\mathsf{i}) \ * \ \mathsf{j} \mapsto -, -\}$$
$$\mathsf{flatten}(; \mathsf{i}, \mathsf{j}, \mathsf{k})$$
$$\{\mathsf{lseg}\ |\tau|\ (\mathsf{j}, \mathsf{k})\}.$$

(Note that flatten must not assign to the variables i, j, or k.)

—

Exercise 2

Show that $\mathsf{tree}\ \tau\ (\mathsf{i})$ and $\exists\tau.\ \mathsf{tree}\ \tau\ (\mathsf{i})$ are precise.

—

Exercise 3

Show that $\mathsf{tree}\ \tau\ (\mathsf{i}) \Rightarrow \mathsf{dag}\ \tau\ (\mathsf{i})$ is valid.

—

Representing S-expressions by Back-Linked Trees

For $\tau \in$ S-exps, we define

$$\mathsf{bktree}\ \tau\ (i, b, t)$$

by structural induction:

$$\mathsf{bktree}\ a\ (i, b, t)\ \text{iff}\ i \mapsto b, t, a, -$$

$$\mathsf{bktree}\ (\tau_1 \cdot \tau_2)\ (i, b, t)\ \text{iff}$$
$$\exists i_1, i_2.\ \ i \mapsto b, t, i_1, i_2\ *\ \mathsf{bktree}\ \tau_1\ (i_1, i, 0)\ *\ \mathsf{bktree}\ \tau_2\ (i_2, i$$

Then

$$\begin{array}{l}
\mathsf{copybktree(i, j)} = \\
\quad \mathbf{if}\ \mathsf{isatom(i)}\ \mathbf{then}\ [\mathsf{j} + 2] := \mathsf{i} \\
\quad \mathbf{else}\ \mathbf{newvar}\ \mathsf{i}', \mathsf{j}'\ \mathbf{in} \\
\quad\quad \big(\mathsf{i}' := [\mathsf{i}]\ ;\ \mathsf{j}' := \mathbf{cons}(\mathsf{j}, 0, 0, 0)\ ;\ \mathsf{copybktree}(\mathsf{i}', \mathsf{j}')\ ;\ [\mathsf{j} + 2] := \\
\quad\quad \mathsf{i}' := [\mathsf{i} + 1]\ ;\ \mathsf{j}' := \mathbf{cons}(\mathsf{j}, 1, 0, 0)\ ;\ \mathsf{copybktree}(\mathsf{i}', \mathsf{j}')\ ;\ [\mathsf{j} + 3]
\end{array}$$

satisfies

$$\begin{array}{l}
\{(\mathsf{p} \wedge \mathsf{dag}\ \tau\ (\mathsf{i}))\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, -, -\} \\
\mathsf{copybktree(i, j)} \\
\{\mathsf{p}\ *\ \mathsf{bktree}\ \tau\ (\mathsf{j}, \mathsf{b}, \mathsf{t})\}
\end{array}$$

since

—

$\{(\mathsf{p} \wedge \mathsf{dag}\,\tau\,(\mathsf{i}))\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, -, -\}$
$\mathbf{if}\ \mathsf{isatom}(\mathsf{i})\ \mathbf{then}$
$\quad\{(\mathsf{p} \wedge \mathsf{isatom}(\tau) \wedge \tau = i)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, -, -\}$
$\quad[\mathsf{j} + 2] := \mathsf{i}$
$\quad\{(\mathsf{p} \wedge \mathsf{isatom}(\tau) \wedge \tau = i)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, \mathsf{i}, -\}$
$\mathbf{else}\ \mathbf{newvar}\ \mathsf{i}', \mathsf{j}'\ \mathbf{in}$
$\quad\big(\{\exists \tau_1, \tau_2, \mathsf{i}_1, \mathsf{i}_2.\ \tau = (\tau_1 \cdot \tau_2) \wedge$
$\qquad\big(\big(p \wedge (\mathsf{i} \mapsto \mathsf{i}_1, \mathsf{i}_2\ *\ (\mathsf{dag}\,\tau_1\,(\mathsf{i}_1) \wedge \mathsf{dag}\,\tau_2\,(\mathsf{i}_2)))\big) * \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, -,$
$\quad\mathsf{i}' := [\mathsf{i}]\ ;$
$\quad\{\exists \tau_1, \tau_2, \mathsf{i}_2.\ \tau = (\tau_1 \cdot \tau_2) \wedge$
$\qquad\big(\big(p \wedge (\mathsf{i} \mapsto \mathsf{i}', \mathsf{i}_2\ *\ (\mathsf{dag}\,\tau_1\,(\mathsf{i}') \wedge \mathsf{dag}\,\tau_2\,(\mathsf{i}_2)))\big) * \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, -, -$
$\quad\mathsf{j}' := \mathbf{cons}(\mathsf{j}, 0, 0, 0)\ ;$
$\quad\{\exists \tau_1, \tau_2, \mathsf{i}_2.\ \tau = (\tau_1 \cdot \tau_2) \wedge$
$\qquad\big(\big(p \wedge (\mathsf{i} \mapsto \mathsf{i}', \mathsf{i}_2\ *\ (\mathsf{dag}\,\tau_1\,(\mathsf{i}') \wedge \mathsf{dag}\,\tau_2\,(\mathsf{i}_2)))\big) * \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, -, -$
$\qquad\quad * \mathsf{j}' \mapsto \mathsf{j}, 0, -, -\big)\}$
$\quad\{\exists \tau_1, \tau_2, \mathsf{i}_2.\ \tau = (\tau_1 \cdot \tau_2) \wedge$
$\qquad\big(\big(p \wedge (\mathsf{i} \mapsto -, \mathsf{i}_2\ *\ \mathsf{dag}\,\tau_2\,(\mathsf{i}_2)) \wedge \mathsf{dag}\,\tau_1\,(\mathsf{i}')\big) * \mathsf{j}' \mapsto \mathsf{j}, 0, -, -$
$\qquad\quad * \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, -, -\big)\}$
$\quad\mathsf{copybktree}(\mathsf{i}', \mathsf{j}')\ ;$
$\quad\{\exists \tau_1, \tau_2, \mathsf{i}_2.\ \tau = (\tau_1 \cdot \tau_2) \wedge$
$\qquad\big(\big(p \wedge (\mathsf{i} \mapsto -, \mathsf{i}_2\ *\ \mathsf{dag}\,\tau_2\,(\mathsf{i}_2))\big) * \mathsf{bktree}\,\tau_1\,(\mathsf{j}', \mathsf{j}, 0)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}$
$\quad[\mathsf{j} + 2] := \mathsf{j}'\ ;$
$\qquad\vdots$
$\{\mathsf{p}\ *\ \mathsf{bktree}\,\tau\,(\mathsf{j}, \mathsf{b}, \mathsf{t})\}.$
—

$\{(\mathsf{p} \wedge \mathsf{dag}\,\tau\,(\mathsf{i}))\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, -, -\}$

$\vdots$

$\{\exists \tau_1, \tau_2, \mathsf{j}_1, \mathsf{i}_2.\ \tau = (\tau_1 \cdot \tau_2) \wedge$

$\big((p \wedge (\mathsf{i} \mapsto -, \mathsf{i}_2\ *\ \mathsf{dag}\,\tau_2\,(\mathsf{i}_2))) * \mathsf{bktree}\,\tau_1\,(\mathsf{j}_1, \mathsf{j}, 0)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}$

$\mathsf{i}' := [\mathsf{i} + 1]$;

$\{\exists \tau_1, \tau_2, \mathsf{j}_1.\ \tau = (\tau_1 \cdot \tau_2) \wedge$

$\big((p \wedge (\mathsf{i} \mapsto -, \mathsf{i}'\ *\ \mathsf{dag}\,\tau_2\,(\mathsf{i}'))) * \mathsf{bktree}\,\tau_1\,(\mathsf{j}_1, \mathsf{j}, 0)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}$

$\mathsf{j}' := \mathbf{cons}(\mathsf{j}, 1, 0, 0)$;

$\{\exists \tau_1, \tau_2, \mathsf{j}_1.\ \tau = (\tau_1 \cdot \tau_2) \wedge$

$\big((p \wedge (\mathsf{i} \mapsto -, \mathsf{i}'\ *\ \mathsf{dag}\,\tau_2\,(\mathsf{i}'))) * \mathsf{bktree}\,\tau_1\,(\mathsf{j}_1, \mathsf{j}, 0)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}$

$*\ \mathsf{j}' \mapsto \mathsf{j}, 1, -, -\big)\}$

$\{\exists \tau_1, \tau_2, \mathsf{j}_1.\ \tau = (\tau_1 \cdot \tau_2) \wedge$

$\big((p \wedge \mathsf{dag}\,\tau_2\,(\mathsf{i}')) * \mathsf{j}' \mapsto \mathsf{j}, 1, -, -$

$*\ \mathsf{bktree}\,\tau_1\,(\mathsf{j}_1, \mathsf{j}, 0)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, \mathsf{j}_1, -\big)\}$

$\mathsf{copybktree}(\mathsf{i}', \mathsf{j}')$;

$\{\exists \tau_1, \tau_2, \mathsf{j}_1.\ \tau = (\tau_1 \cdot \tau_2) \wedge$

$(p\ *\ \mathsf{bktree}\,\tau_2\,(\mathsf{j}', \mathsf{j}, 1)\ *\ \mathsf{bktree}\,\tau_1\,(\mathsf{j}_1, \mathsf{j}, 0)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, \mathsf{j}_1, -)\}$

$[\mathsf{j} + 3] := \mathsf{j}'$

$\{\exists \tau_1, \tau_2, \mathsf{j}_1, \mathsf{j}_2.\ \tau = (\tau_1 \cdot \tau_2) \wedge$

$(p\ *\ \mathsf{bktree}\,\tau_2\,(\mathsf{j}_2, \mathsf{j}, 1)\ *\ \mathsf{bktree}\,\tau_1\,(\mathsf{j}_1, \mathsf{j}, 0)\ *\ \mathsf{j} \mapsto \mathsf{b}, \mathsf{t}, \mathsf{j}_1, \mathsf{j}_2)\}$

$\{\mathsf{p}\ *\ \mathsf{bktree}\,\tau\,(\mathsf{j}, \mathsf{b}, \mathsf{t})\}.$

—

Substitution in Back-Linked Trees

We first note that

$$\begin{array}{l}\mathsf{bktree}\ \tau\ (\mathsf{i},\mathsf{b},\mathsf{t}) \Rightarrow \\ \quad \exists \mathsf{k}.\ \mathsf{i} \hookrightarrow \mathsf{b},\mathsf{t},\mathsf{k},- \\ \qquad \wedge\ (\mathsf{isatom}(\mathsf{k}) \Rightarrow \tau = \mathsf{k} \wedge \mathsf{i} \mapsto \mathsf{b},\mathsf{t},\mathsf{k},-) \\ \qquad \wedge\ (\neg\, \mathsf{isatom}(\mathsf{k}) \Rightarrow \exists \tau_1,\tau_2,\mathsf{i}_2.\ \tau = (\tau_1 \cdot \tau_2) \wedge \\ \qquad\quad (\mathsf{i} \mapsto \mathsf{b},\mathsf{t},\mathsf{k},\mathsf{i}_2\ *\ \mathsf{bktree}\ \tau_1\ (\mathsf{k},\mathsf{i},0)\ *\ \mathsf{bktree}\ \tau_2\ (\mathsf{i}_2,\mathsf{i},1)).\end{array}$$

Then

$$\begin{array}{l}\mathsf{subst3}(\mathsf{a},\mathsf{j},\mathsf{i}) = \\ \quad \mathbf{newvar}\ \mathsf{k}\ \mathbf{in}\ \big(\mathsf{k} := [\mathsf{i}+2]\ ; \\ \qquad \mathbf{if}\ \mathsf{isatom}(\mathsf{k})\ \mathbf{then} \\ \qquad\quad \mathbf{if}\ \mathsf{k} = \mathsf{a}\ \mathbf{then}\ \mathsf{copybktree}(\mathsf{j},\mathsf{i})\ \mathbf{else}\ \mathbf{skip} \\ \qquad \mathbf{else}\ (\mathsf{subst3}(\mathsf{a},\mathsf{j},\mathsf{k})\ ;\ \mathsf{k} := [\mathsf{i}+3]\ ;\ \mathsf{subst3}(\mathsf{a},\mathsf{j},\mathsf{k}))\big)\end{array}$$

satisfies

$$\begin{array}{l}\{\mathsf{bktree}\ \tau\ (\mathsf{i},\mathsf{b},\mathsf{t})\ *\ (\mathsf{p} \wedge \mathsf{dag}\ \tau'\ (\mathsf{j}))\} \\ \mathsf{subst3}(\mathsf{a},\mathsf{j},\mathsf{i}) \\ \{\mathsf{bktree}\ (\tau/\mathsf{a} \rightarrow \tau')\ (\mathsf{i},\mathsf{b},\mathsf{t})\ *\ \mathsf{p}\}\end{array}$$

—

since

$\{\mathsf{bktree}\,\tau\,(\mathsf{i},\mathsf{b},\mathsf{t})\ *\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j}))\}$

$\mathbf{newvar}\ \mathsf{k}\ \mathbf{in}\ \big(\mathsf{k}:=[\mathsf{i}+2]\ ;$

$\quad\{(\mathsf{i}\hookrightarrow\mathsf{b},\mathsf{t},\mathsf{k},-\wedge(\mathsf{isatom}(\mathsf{k})\Rightarrow\tau=\mathsf{k}\wedge\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{k},-)$

$\qquad\wedge(\neg\,\mathsf{isatom}(\mathsf{k})\Rightarrow\exists\tau_1,\tau_2,\mathsf{i}_2.\ \tau=(\tau_1\cdot\tau_2)\wedge$

$\qquad\quad(\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{k},\mathsf{i}_2\ *\ \mathsf{bktree}\,\tau_1\,(\mathsf{k},\mathsf{i},0)\ *\ \mathsf{bktree}\,\tau_2\,(\mathsf{i}_2,\mathsf{i},1))))$

$\qquad*\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j}))\}$

$\quad\mathbf{if}\ \mathsf{isatom}(\mathsf{k})\ \mathbf{then}$

$\qquad\{(\mathsf{isatom}(\tau)\wedge\tau=\mathsf{k}\wedge\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{k},-)\ *\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j}))\}$

$\qquad\mathbf{if}\ \mathsf{k}=\mathsf{a}\ \mathbf{then}$

$\qquad\quad\{(\mathsf{isatom}(\tau)\wedge\tau=\mathsf{a}\wedge\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{k},-)\ *\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j}))\}$

$\qquad\quad\{(\tau/\mathsf{a}\to\tau')=\tau'\wedge(\mathsf{i}\mapsto\mathsf{b},\mathsf{t},-,-\ *\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j})))\}$

$\qquad\quad\mathsf{copybktree}(\mathsf{j},\mathsf{i})$

$\qquad\quad\{(\tau/\mathsf{a}\to\tau')=\tau'\wedge(\mathsf{bktree}\,\tau'\,(\mathsf{i},\mathsf{b},\mathsf{t})\ *\ \mathsf{p})\}$

$\qquad\mathbf{else}$

$\qquad\quad\{(\mathsf{isatom}(\tau)\wedge\tau\neq\mathsf{a}\wedge\tau=\mathsf{k}\wedge\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{k},-)\ *\ \mathsf{p}\}$

$\qquad\quad\{((\tau/\mathsf{a}\to\tau')=\tau\wedge\mathsf{isatom}(\tau)\wedge\tau=\mathsf{k}\wedge\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{k},-)\ *$

$\qquad\quad\mathbf{skip}$

$\qquad\vdots$

$\{\mathsf{bktree}\,(\tau/\mathsf{a}\to\tau')\,(\mathsf{i},\mathsf{b},\mathsf{t})\ *\ \mathsf{p}\}$

—

$\{\mathsf{bktree}\,\tau\,(\mathsf{i},\mathsf{b},\mathsf{t})\ *\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j}))\}$

$\vdots$

$\mathbf{else}$

$\Big(\{\exists\tau_1,\tau_2,\mathsf{i}_2.\ \tau=(\tau_1\cdot\tau_2)\wedge$

$(\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{k},\mathsf{i}_2\ *\ \mathsf{bktree}\,\tau_1\,(\mathsf{k},\mathsf{i},0)\ *\ \mathsf{bktree}\,\tau_2\,(\mathsf{i}_2,\mathsf{i},1)$

$*\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j})\wedge\mathsf{dag}\,\tau'\,(\mathsf{j})))\}$

$\mathsf{subst3}(\mathsf{a},\mathsf{j},\mathsf{k})\ ;$

$\{\exists\tau_1,\tau_2,\mathsf{i}_2.\ \tau=(\tau_1\cdot\tau_2)\wedge$

$(\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{k},\mathsf{i}_2\ *\ \mathsf{bktree}\,(\tau_1/\mathsf{a}\longrightarrow\tau')\,(\mathsf{k},\mathsf{i},0)\ *\ \mathsf{bktree}\,\tau_2\,(\mathsf{i}_2$

$*\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j})))\}$

$\mathsf{k}:=[\mathsf{i}+3]\ ;$

$\{\exists\tau_1,\tau_2,\mathsf{i}_1.\ \tau=(\tau_1\cdot\tau_2)\wedge$

$(\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{i}_1,\mathsf{k}\ *\ \mathsf{bktree}\,(\tau_1/\mathsf{a}\longrightarrow\tau')\,(\mathsf{i}_1,\mathsf{i},0)\ *\ \mathsf{bktree}\,\tau_2\,(\mathsf{k}$

$*\ (\mathsf{p}\wedge\mathsf{dag}\,\tau'\,(\mathsf{j})))\}$

$\mathsf{subst3}(\mathsf{a},\mathsf{j},\mathsf{k})$

$\{\exists\tau_1,\tau_2,\mathsf{i}_1.\ \tau=(\tau_1\cdot\tau_2)\wedge$

$(\mathsf{i}\mapsto\mathsf{b},\mathsf{t},\mathsf{i}_1,\mathsf{k}\ *\ \mathsf{bktree}\,(\tau_1/\mathsf{a}\longrightarrow\tau')\,(\mathsf{i}_1,\mathsf{i},0)$

$*\ \mathsf{bktree}\,(\tau_2/\mathsf{a}\longrightarrow\tau')\,(\mathsf{k},\mathsf{i},1)\ *\ \mathsf{p})\}\Big)\Big)$

$\{\mathsf{bktree}\,(\tau/\mathsf{a}\longrightarrow\tau')\,(\mathsf{i},\mathsf{b},\mathsf{t})\ *\ \mathsf{p}\}$

—