Class Notes for BRICS Minicourse - Fall 2003

SEPARATION LOGIC

1. Introduction

John C. Reynolds
Department of Computer Science
Carnegie Mellon University
and
BRICS
Aarhus University

November 9, 2003

The Problem — An Example

A program for in-place list reversal:

$\mathsf{j} := \mathbf{nil}\ ; \mathbf{while}\ \mathsf{i} \neq \mathbf{nil}\ \mathbf{do}\ (\mathsf{k} := [\mathsf{i}+1]\ ; [\mathsf{i}+1] := \mathsf{j}\ ; \mathsf{j} := \mathsf{i}\ ; \mathsf{i} := \mathsf{k}).$

An inadequate invariant:

$$\exists \alpha, \beta.\ \mathsf{list}\ \alpha\ \mathsf{i} \wedge \mathsf{list}\ \beta\ \mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta,$$

where

$$\mathsf{list}\ \epsilon\ \mathsf{i} \stackrel{\text{def}}{=} \mathsf{i} = \mathbf{nil} \qquad \mathsf{list}(\mathsf{a}{\cdot}\alpha)\ \mathsf{i} \stackrel{\text{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \hookrightarrow \mathsf{a}, \mathsf{j} \wedge \mathsf{list}\ \alpha\ \mathsf{j}.$$

An adequate invariant:

$$\begin{aligned}&(\exists \alpha, \beta.\ \mathsf{list}\ \alpha\ \mathsf{i} \wedge \mathsf{list}\ \beta\ \mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta)\\ &\quad \wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{i},\mathsf{k}) \wedge \mathbf{reachable}(\mathsf{j},\mathsf{k}) \Rightarrow \mathsf{k} = \mathbf{nil}),\end{aligned}$$

where

$$\begin{aligned}\mathbf{reachable}(\mathsf{i},\mathsf{j}) &\stackrel{\text{def}}{=} \exists n \geq 0.\ \mathbf{reachable}_n(\mathsf{i},\mathsf{j})\\ \mathbf{reachable}_0(\mathsf{i},\mathsf{j}) &\stackrel{\text{def}}{=} \mathsf{i} = \mathsf{j}\\ \mathbf{reachable}_{n+1}(\mathsf{i},\mathsf{j}) &\stackrel{\text{def}}{=} \exists \mathsf{a}, \mathsf{k}.\ \mathsf{i} \hookrightarrow \mathsf{a}, \mathsf{k} \wedge \mathbf{reachable}_n(\mathsf{k},\mathsf{j}).\end{aligned}$$

A stronger invariant:

$$\begin{aligned}&(\exists \alpha, \beta.\ \mathsf{list}\ \alpha\ \mathsf{i} \wedge \mathsf{list}\ \beta\ \mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta)\\ &\quad \wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{i},\mathsf{k}) \wedge \mathbf{reachable}(\mathsf{j},\mathsf{k}) \Rightarrow \mathsf{k} = \mathbf{nil})\\ &\quad \wedge \mathsf{list}\ \gamma\ \mathsf{x}\\ &\quad \wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{x},\mathsf{k})\\ &\qquad\qquad \wedge (\mathbf{reachable}(\mathsf{i},\mathsf{k}) \vee \mathbf{reachable}(\mathsf{j},\mathsf{k})) \Rightarrow \mathsf{k} = \mathbf{nil}).\end{aligned}$$

The Solution

Instead of

$$\begin{aligned}&(\exists \alpha, \beta.\ \mathsf{list}\,\alpha\,\mathsf{i} \wedge \mathsf{list}\,\beta\,\mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta)\\ &\quad\wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{i},\mathsf{k}) \wedge \mathbf{reachable}(\mathsf{j},\mathsf{k}) \Rightarrow \mathsf{k} = \mathbf{nil}),\end{aligned}$$

or

$$\begin{aligned}&(\exists \alpha, \beta.\ \mathsf{list}\,\alpha\,\mathsf{i} \wedge \mathsf{list}\,\beta\,\mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta)\\ &\quad\wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{i},\mathsf{k}) \wedge \mathbf{reachable}(\mathsf{j},\mathsf{k}) \Rightarrow \mathsf{k} = \mathbf{nil})\\ &\quad\wedge \mathsf{list}\,\gamma\,\mathsf{x}\\ &\quad\wedge (\forall \mathsf{k}.\ \mathbf{reachable}(\mathsf{x},\mathsf{k})\\ &\qquad\qquad \wedge (\mathbf{reachable}(\mathsf{i},\mathsf{k}) \vee \mathbf{reachable}(\mathsf{j},\mathsf{k})) \Rightarrow \mathsf{k} = \mathbf{nil}),\end{aligned}$$

in separation logic we can write

$$(\exists \alpha, \beta.\ \mathsf{list}\,\alpha\,\mathsf{i} \ * \ \mathsf{list}\,\beta\,\mathsf{j}) \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta,$$

or

$$(\exists \alpha, \beta.\ \mathsf{list}\,\alpha\,\mathsf{i} \ * \ \mathsf{list}\,\beta\,\mathsf{j} \ * \ \mathsf{list}\,\gamma\,\mathsf{x}) \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta).$$

Overview of Separation Logic

- Low-level programming language
  - Extension of simple imperative language
  - Commands for allocating, accessing, mutating, and deallocating data structures
  - Dangling pointer faults (if pointer is dereferenced)
- Program specification and proof
  - Extension of Hoare logic
  - Separating (independent, spatial) conjunction ($*$) and implication ($-\!\!*$)
- Inductive definitions over abstract structures

History

- Distinct Nonrepeating Tree Systems (Burstall 1972)
- Bunched Implication (BI) Logics (O'Hearn and Pym 1999)
- Adding Separating Conjunction to Hoare Logic (Reynolds 1999, with flaws)
- Intuitionistic Separation Logic (Ishtiaq and O'Hearn 2001, Reynolds 2000)
- Classical Separation Logic (Ishtiaq and O'Hearn 2001)
- Adding Address Arithmetic (Reynolds 2001)

States

Without address arithmetic (old version):

$$
\begin{aligned}
\text{Values} &= \text{Integers} \cup \text{Atoms} \cup \text{Addresses} \\
&\quad \text{where Integers, Atoms, and Addresses are disjoint} \\
&\quad \mathbf{nil} \in \text{Atoms} \\
\text{Stores}_V &= V \to \text{Values} \\
\text{Heaps} &= \bigcup_{A \overset{\text{fin}}{\subseteq} \text{Addresses}} (A \to \text{Values}^+) \\
\text{States}_V &= \text{Stores}_V \times \text{Heaps} \\
&\qquad\qquad\qquad \text{where } V \text{ is a finite set of variables.}
\end{aligned}
$$

With address arithmetic (new version):

$$
\begin{aligned}
\text{Values} &= \text{Integers} \\
\text{Atoms} \cup \text{Addresses} &\subseteq \text{Integers} \\
&\quad \text{where Atoms and Addresses are disjoint} \\
\mathbf{nil} &\in \text{Atoms} \\
\text{Stores}_V &= V \to \text{Values} \\
\text{Heaps} &= \bigcup_{A \overset{\text{fin}}{\subseteq} \text{Addresses}} (A \to \text{Values}) \\
\text{States}_V &= \text{Stores}_V \times \text{Heaps} \\
&\quad \text{where } V \text{ is a finite set of variables.}
\end{aligned}
$$

The Programming Language: An Informal View

The simple imperative language:

$:=$ **skip** ; **if** – **then** – **else** – **while** – **do** –

plus:

		Store : $\mathsf{x}\colon 3,\ \mathsf{y}\colon 4$
		Heap : empty
Allocation	$\mathsf{x} := \mathbf{cons}(1, 2)\ ;$	
		Store : $\mathsf{x}\colon 37,\ \mathsf{y}\colon 4$
		Heap : $37\colon 1,\ 38\colon 2$
Lookup	$\mathsf{y} := [\mathsf{x}]\ ;$	
		Store : $\mathsf{x}\colon 37,\ \mathsf{y}\colon 1$
		Heap : $37\colon 1,\ 38\colon 2$
Mutation	$[\mathsf{x} + 1] := 3\ ;$	
		Store : $\mathsf{x}\colon 37,\ \mathsf{y}\colon 1$
		Heap : $37\colon 1,\ 38\colon 3$
Deallocation	$\mathbf{dispose}(\mathsf{x} + 1)$	
		Store : $\mathsf{x}\colon 37,\ \mathsf{y}\colon 1$
		Heap : $37\colon 1$

Note that:

- Expressions depend only upon the store.
  - no side effects or nontermination.
  - **cons** and $[-]$ are parts of commands.
- Allocation is nondeterminate.

Memory Faults

$$
\begin{array}{lll}
 & & \text{Store}: \ \mathsf{x}\!:3,\ \mathsf{y}\!:4 \\
 & & \text{Heap}: \ \text{empty} \\
\text{Allocation} & \mathsf{x} := \mathbf{cons}(1,2)\ ; & \\
 & & \text{Store}: \ \mathsf{x}\!:37,\ \mathsf{y}\!:4 \\
 & & \text{Heap}: \ 37\!:1,\ 38\!:2 \\
\text{Lookup} & \mathsf{y} := [\mathsf{x}]\ ; & \\
 & & \text{Store}: \ \mathsf{x}\!:37,\ \mathsf{y}\!:1 \\
 & & \text{Heap}: \ 37\!:1,\ 38\!:2 \\
\text{Mutation} & [\mathsf{x}+2] := 3\ ; & \\
 & & \mathbf{abort}
\end{array}
$$

Faults can also be caused by out-of-range lookup or deallocation.

Assertions

Standard predicate calculus:

$$\wedge \qquad \vee \qquad \neg \qquad \Rightarrow \qquad \forall \qquad \exists$$

plus:

- **emp**
  The heap is empty.

- $e \mapsto e'$
  The heap contains one cell, at address e with contents e'.

- $p_1 \ * \ p_2$
  The heap can be split into two disjoint parts such that p_1 holds for one part and p_2 holds for the other.

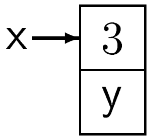
- $p_1 \mathrel{-\!\!*} p_2$
  If the current heap is extended with a disjoint part in which p_1 holds, then p_2 holds for the extended heap.

Some Abbreviations

$$
\begin{aligned}
e \mapsto - &\stackrel{\text{def}}{=} \exists x'.\, e \mapsto x' \quad \text{where } x' \text{ not free in } e \\
e \hookrightarrow e' &\stackrel{\text{def}}{=} e \mapsto e' \ast \mathbf{true} \\
e \mapsto e_1, \ldots, e_n &\stackrel{\text{def}}{=} e \mapsto e_1 \ast \cdots \ast e + n - 1 \mapsto e_n \\
e \hookrightarrow e_1, \ldots, e_n &\stackrel{\text{def}}{=} e \hookrightarrow e_1 \ast \cdots \ast e + n - 1 \hookrightarrow e_n \\
&\text{iff } e \mapsto e_1, \ldots, e_n \ast \mathbf{true}
\end{aligned}
$$

Examples of Separating Conjunction

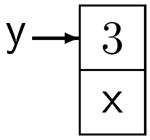
1. $\mathsf{x} \mapsto 3, \mathsf{y}$



   Store : $\mathsf{x}\colon \alpha$, $\mathsf{y}\colon \beta$

   Heap : $\alpha\colon 3$, $\alpha{+}1\colon \beta$

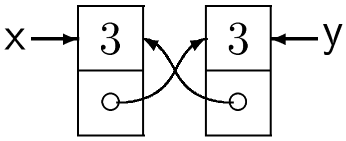
2. $\mathsf{y} \mapsto 3, \mathsf{x}$



   Store : $\mathsf{x}\colon \alpha$, $\mathsf{y}\colon \beta$

   Heap : $\beta\colon 3$, $\beta{+}1\colon \alpha$

3. $\mathsf{x} \mapsto 3, \mathsf{y} \ * \ \mathsf{y} \mapsto 3, \mathsf{x}$

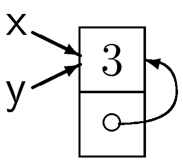


   Store : $\mathsf{x}\colon \alpha$, $\mathsf{y}\colon \beta$

   Heap : $\alpha\colon 3$, $\alpha{+}1\colon \beta$, $\beta\colon 3$, $\beta{+}1\colon \alpha$

   where α, $\alpha + 1$, β, $\beta + 1$ are distinct

4. $\mathsf{x} \mapsto 3, \mathsf{y} \wedge \mathsf{y} \mapsto 3, \mathsf{x}$



   Store : $\mathsf{x}\colon \alpha$, $\mathsf{y}\colon \alpha$

   Heap : $\alpha\colon 3$, $\alpha{+}1\colon \alpha$

5. $\mathsf{x} \hookrightarrow 3, \mathsf{y} \wedge \mathsf{y} \hookrightarrow 3, \mathsf{x}$

   Store : $\mathsf{x}\colon \alpha$, $\mathsf{y}\colon \beta$

   Heap : $\alpha\colon 3$, $\alpha{+}1\colon \beta$, $\beta\colon 3$, $\beta{+}1\colon \alpha$, ...

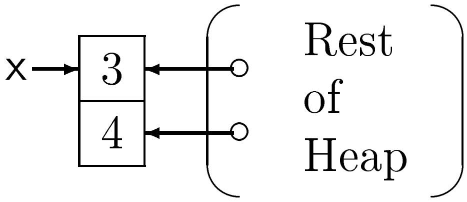
   As in (3) or (4), possibly with additional cells

An Example of Separating Implication

Suppose p holds for

Store : $\mathsf{x}\colon \alpha, \ldots$

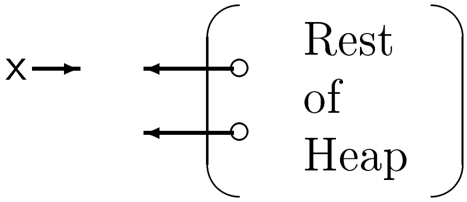
Heap : $\alpha\colon 3, \alpha + 1\colon 4, \ldots$



Then $(\mathsf{x} \mapsto 3, 4) \mathrel{-\!\!*} p$ holds for

Store : $\mathsf{x}\colon \alpha, \ldots$

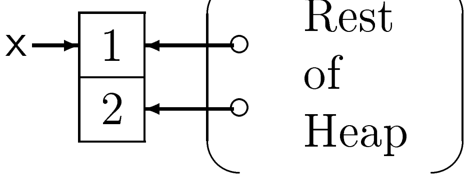
Heap : $\ldots$



and $\mathsf{x} \mapsto 1, 2 \; * \; ((\mathsf{x} \mapsto 3, 4) \mathrel{-\!\!*} p)$ holds for

Store : $\mathsf{x}\colon \alpha, \ldots$

Heap : $\alpha\colon 1, \alpha + 1\colon 2, \ldots$



In particular,

$$\{\mathsf{x} \mapsto 1, 2 \; * \; ((\mathsf{x} \mapsto 3, 4) \mathrel{-\!\!*} p)\} \; [\mathsf{x}] := 3 \,; [\mathsf{x} + 1] := 4 \; \{p\},$$

and more generally,

$$\{\mathsf{x} \mapsto -, - \; * \; ((\mathsf{x} \mapsto 3, 4) \mathrel{-\!\!*} p)\} \; [\mathsf{x}] := 3 \,; [\mathsf{x} + 1] := 4 \; \{p\}.$$

Rules and Axiom Schemata for $*$ and $-\!*$

$$p_1 \ * \ p_2 \Leftrightarrow p_2 \ * \ p_1$$

$$(p_1 \ * \ p_2) \ * \ p_3 \Leftrightarrow p_1 \ * \ (p_2 \ * \ p_3)$$

$$p \ * \ \mathbf{emp} \Leftrightarrow p$$

$$(p_1 \vee p_2) \ * \ q \Leftrightarrow (p_1 \ * \ q) \vee (p_2 \ * \ q)$$

$$(p_1 \wedge p_2) \ * \ q \Rightarrow (p_1 \ * \ q) \wedge (p_2 \ * \ q)$$

$$(\exists x.\ p_1) \ * \ p_2 \Leftrightarrow \exists x.\ (p_1 \ * \ p_2) \quad \text{when } x \text{ not free in } p_2$$

$$(\forall x.\ p_1) \ * \ p_2 \Rightarrow \forall x.\ (p_1 \ * \ p_2) \quad \text{when } x \text{ not free in } p_2$$

$$\frac{p_1 \Rightarrow p_2 \qquad q_1 \Rightarrow q_2}{p_1 \ * \ q_1 \Rightarrow p_2 \ * \ q_2} \quad \text{(monotonicity)}$$

$$\frac{p_1 \ * \ p_2 \Rightarrow p_3}{p_1 \Rightarrow (p_2 -\!* \ p_3)} \quad \text{(currying)} \qquad \frac{p_1 \Rightarrow (p_2 -\!* \ p_3)}{p_1 \ * \ p_2 \Rightarrow p_3.} \quad \text{(decurrying)}$$

Two Unsound Axiom Schemata

$$p \not\Rightarrow p \ast p \qquad \text{(Contraction)}$$

e.g. $p : \mathsf{x} \mapsto 1$

$$p \ast q \not\Rightarrow p \qquad \text{(Weakening)}$$

e.g. $p : \mathsf{x} \mapsto 1$

$q : \mathsf{y} \mapsto 2$

Some Axiom Schemata for $\mapsto$

$$e_1 \mapsto e_1' \wedge e_2 \mapsto e_2' \Leftrightarrow e_1 \mapsto e_1' \wedge e_1 = e_2 \wedge e_1' = e_2'$$

$$e_1 \hookrightarrow e_1' \ast e_2 \hookrightarrow e_2' \Rightarrow e_1 \neq e_2$$

$$\mathbf{emp} \Leftrightarrow \forall x.\ \neg(x \hookrightarrow -).$$

(Regrettably, these are far from complete.)

Specifications

- $\{p\}\ c\ \{q\}$ (partial correctness)

  Starting in any state in which p holds:

  - No execution of c aborts.
  - When some execution of c terminates in a final state, then q holds in the final state.

- $[\,p\,]\ c\ [\,q\,]$ (total correctness)

  Starting in any state in which p holds:

  - No execution of c aborts.
  - Every execution of c terminates.
  - When some execution of c terminates in a final state, then q holds in the final state.

Thus:

• • • Well-specified programs don't go wrong. • • •

— and memory-fault checking is unnecessary.

On the Other Hand

$$
\begin{array}{l}
\{\mathsf{x} \mapsto - \ * \ \mathsf{y} \mapsto -\} \\
\textbf{if } \mathsf{y} = \mathsf{x} + 1 \textbf{ then} \\
\quad \{\mathsf{x} \mapsto -, -\} \\
\quad \textbf{skip} \\
\textbf{else if } \mathsf{x} = \mathsf{y} + 1 \textbf{ then} \\
\quad \{\mathsf{y} \mapsto -, -\} \\
\quad \mathsf{x} := \mathsf{y} \\
\textbf{else} \\
\quad \big(\{\mathsf{x} \mapsto - \ * \ \mathsf{y} \mapsto -\} \\
\quad \textbf{dispose } \mathsf{x}\,; \\
\quad \{\mathsf{y} \mapsto -\} \\
\quad \textbf{dispose } \mathsf{y}\,; \\
\quad \{\textbf{emp}\} \\
\quad \mathsf{x} := \textbf{cons}(1, 2)\big) \\
\{\mathsf{x} \mapsto -, -\}.
\end{array}
$$

Hoare's Inference Rules

The command-specific inference rules of Hoare logic remain sound, as do structural rules such as

- Strengthening Precedent

$$\frac{p \Rightarrow q \qquad \{q\}\ c\ \{r\}}{\{p\}\ c\ \{r\}.}$$

- Weakening Consequent

$$\frac{\{p\}\ c\ \{q\} \qquad q \Rightarrow r}{\{p\}\ c\ \{r\}.}$$

- Existential Quantification (Auxiliary Variable Elimination)

$$\frac{\{p\}\ c\ \{q\}}{\{\exists v.\ p\}\ c\ \{\exists v.\ q\},}$$

  where v is not free in c.

- Conjunction

$$\frac{\{p_1\}\ c\ \{q_1\} \qquad \{p_2\}\ c\ \{q_2\}}{\{p_1 \wedge p_2\}\ c\ \{q_1 \wedge q_2\},}$$

- Substitution

$$\frac{\{p\}\ c\ \{q\}}{(\{p\}\ c\ \{q\})/v_1 \to e_1,\ \ldots\ ,\ v_n \to e_n,}$$

  where $v_1, \ldots, v_n$ 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.

The Failure of the Rule of Constancy

On the other hand,

- Rule of Constancy

$$\frac{\{p\}\ c\ \{q\}}{\{p \wedge r\}\ c\ \{q \wedge r\},}$$

where no variable occurring free in r is modified by c.

is *unsound*, since, for example

$$\frac{\{\mathsf{x} \mapsto -\}\ [\mathsf{x}] := 4\ \{\mathsf{x} \mapsto 4\}}{\{\mathsf{x} \mapsto - \wedge \mathsf{y} \mapsto 3\}\ [\mathsf{x}] := 4\ \{\mathsf{x} \mapsto 4 \wedge \mathsf{y} \mapsto 3\}}$$

fails when $\mathsf{x} = \mathsf{y}$.

Instead, we have the

- Frame Rule (O'Hearn)

$$\frac{\{p\}\ c\ \{q\}}{\{p\ *\ r\}\ c\ \{q\ *\ r\},}$$

where no variable occurring free in r is modified by c.

Local Specifications

- The *footprint* of a command is the variables and the parts of the heap that are actually used by the command.
- A specification of a command is *local* when it mentions only the footprint.
- By using the frame rule, one can move from local to non-local specifications.

Inference Rules for Mutation

- Local

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

- Global

$$\frac{}{\{(e \mapsto -)\ *\ r\}\ [e] := e'\ \{(e \mapsto e')\ *\ r\}.}$$

- Backward Reasoning

$$\frac{}{\{(e \mapsto -)\ *\ ((e \mapsto e') \mathrel{-\!\!*} p)\}\ [e] := e'\ \{p\}.}$$

Inference Rules for Deallocation

- Local
$$\frac{}{\{e \mapsto -\}\ \textbf{dispose}\ e\ \{\textbf{emp}\}.}$$

- Global, Backwards Reasoning
$$\frac{}{\{(e \mapsto -) \ast r\}\ \textbf{dispose}\ e\ \{r\}.}$$

Inference Rules for Noninterfering Allocation

- Local
$$\frac{}{\{\textbf{emp}\}\ v := \textbf{cons}(\overline{e})\ \{v \mapsto \overline{e}\},}$$
where v is not free in $\overline{e} \stackrel{\text{def}}{=} e_1, \ldots, e_n$.

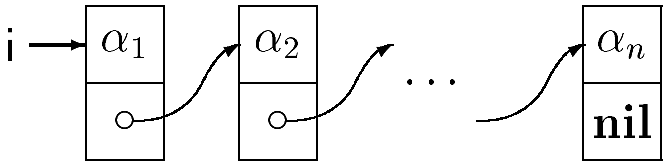
- Global
$$\frac{}{\{r\}\ v := \textbf{cons}(\overline{e})\ \{(v \mapsto \overline{e}) \ast r\},}$$
where v is not free in $\overline{e}$ or r.

An Example of an Annotated Specification

$$
\begin{array}{l}
\{\mathbf{emp}\} \\
\mathsf{x} := \mathbf{cons}(\mathsf{a}, \mathsf{a}) \; ; \\
\{\mathsf{x} \mapsto \mathsf{a}, \mathsf{a}\} \\
\mathsf{y} := \mathbf{cons}(\mathsf{b}, \mathsf{b}) \; ; \\
\{(\mathsf{x} \mapsto \mathsf{a}, \mathsf{a}) \ * \ (\mathsf{y} \mapsto \mathsf{b}, \mathsf{b})\} \\
\{(\mathsf{x} \mapsto \mathsf{a}, -) \ * \ (\mathsf{y} \mapsto \mathsf{b}, -)\} \\
[\mathsf{x} + 1] := \mathsf{y} - \mathsf{x} \; ; \\
\{(\mathsf{x} \mapsto \mathsf{a}, \mathsf{y} - \mathsf{x}) \ * \ (\mathsf{y} \mapsto \mathsf{b}, -)\} \\
[\mathsf{y} + 1] := \mathsf{x} - \mathsf{y} \; ; \\
\{(\mathsf{x} \mapsto \mathsf{a}, \mathsf{y} - \mathsf{x}) \ * \ (\mathsf{y} \mapsto \mathsf{b}, \mathsf{x} - \mathsf{y})\} \\
\{\exists \mathsf{o}. \ (\mathsf{x} \mapsto \mathsf{a}, \mathsf{o}) \ * \ (\mathsf{x} + \mathsf{o} \mapsto \mathsf{b}, \ - \mathsf{o})\}.
\end{array}
$$

Singly-linked Lists

$\mathsf{list}\,\alpha\,\mathsf{i}$:



is defined by

$$\mathsf{list}\,\epsilon\,\mathsf{i} \stackrel{\mathrm{def}}{=} \mathbf{emp} \wedge \mathsf{i} = \mathbf{nil}$$

$$\mathsf{list}\,(\mathsf{a}{\cdot}\alpha)\,\mathsf{i} \stackrel{\mathrm{def}}{=} \exists \mathsf{j}.\ \mathsf{i} \mapsto \mathsf{a}, \mathsf{j} \ * \ \mathsf{list}\,\alpha\,\mathsf{j}.$$

where we write

- ϵ for the empty sequence.
- $\alpha{\cdot}\beta$ for the composition of α followed by β.
- $\alpha^{\dagger}$ for the reflection of α.

Reversing a List

$$
\begin{array}{l}
\{\mathsf{list}\,\alpha_0\,\mathsf{i}\}\\
\{\mathsf{list}\,\alpha_0\,\mathsf{i} \;*\; (\mathbf{emp} \wedge \mathbf{nil} = \mathbf{nil})\}\\
\mathsf{j} := \mathbf{nil}\,;\\
\{\mathsf{list}\,\alpha_0\,\mathsf{i} \;*\; (\mathbf{emp} \wedge \mathsf{j} = \mathbf{nil})\}\\
\{\mathsf{list}\,\alpha_0\,\mathsf{i} \;*\; \mathsf{list}\,\epsilon\,\mathsf{j}\}\\
\{\exists \alpha, \beta.\; (\mathsf{list}\,\alpha\,\mathsf{i} \;*\; \mathsf{list}\,\beta\,\mathsf{j}) \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta\}\\
\mathbf{while}\;\mathsf{i} \neq \mathbf{nil}\;\mathbf{do}\\
\quad \big(\{\exists \mathsf{a}, \alpha, \beta.\; (\mathsf{list}\,(\mathsf{a}{\cdot}\alpha)\,\mathsf{i} \;*\; \mathsf{list}\,\beta\,\mathsf{j}) \wedge \alpha_0^\dagger = (\mathsf{a}{\cdot}\alpha)^\dagger{\cdot}\beta\}\\
\quad\;\; \{\exists \mathsf{a}, \alpha, \beta, \mathsf{k}.\; (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k} \;*\; \mathsf{list}\,\alpha\,\mathsf{k} \;*\; \mathsf{list}\,\beta\,\mathsf{j}) \wedge \alpha_0^\dagger = (\mathsf{a}{\cdot}\alpha)^\dagger{\cdot}\beta\}\\
\quad\;\; \mathsf{k} := [\mathsf{i} + 1]\,;\\
\quad\;\; \{\exists \mathsf{a}, \alpha, \beta.\; (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k} \;*\; \mathsf{list}\,\alpha\,\mathsf{k} \;*\; \mathsf{list}\,\beta\,\mathsf{j}) \wedge \alpha_0^\dagger = (\mathsf{a}{\cdot}\alpha)^\dagger{\cdot}\beta\}\\
\quad\;\; [\mathsf{i} + 1] := \mathsf{j}\,;\\
\quad\;\; \{\exists \mathsf{a}, \alpha, \beta.\; (\mathsf{i} \mapsto \mathsf{a}, \mathsf{j} \;*\; \mathsf{list}\,\alpha\,\mathsf{k} \;*\; \mathsf{list}\,\beta\,\mathsf{j}) \wedge \alpha_0^\dagger = (\mathsf{a}{\cdot}\alpha)^\dagger{\cdot}\beta\}\\
\quad\;\; \{\exists \mathsf{a}, \alpha, \beta.\; (\mathsf{list}\,\alpha\,\mathsf{k} \;*\; \mathsf{list}\,(\mathsf{a}{\cdot}\beta)\,\mathsf{i}) \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\mathsf{a}{\cdot}\beta\}\\
\quad\;\; \{\exists \alpha, \beta.\; (\mathsf{list}\,\alpha\,\mathsf{k} \;*\; \mathsf{list}\,\beta\,\mathsf{i}) \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta\}\\
\quad\;\; \mathsf{j} := \mathsf{i}\,;\,\mathsf{i} := \mathsf{k}\\
\quad\;\; \{\exists \alpha, \beta.\; (\mathsf{list}\,\alpha\,\mathsf{i} \;*\; \mathsf{list}\,\beta\,\mathsf{j}) \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta\}\big)\\
\{\exists \alpha, \beta.\; \mathsf{list}\,\beta\,\mathsf{j} \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta \wedge \alpha = \epsilon\}\\
\{\mathsf{list}\,\alpha_0^\dagger\,\mathsf{j}\}
\end{array}
$$