Verifying Heap-manipulating Recursive Procedures Using Cyclic Proof

Reuben N. S. Rowe

PPLV Group, UCL

Resource Reasoning Meeting
Monday 20th April 2015

Introduction

- Started as an RA in late May 2014
- Boosting Automated Verification Using Cyclic Proof (EPSRC Grant, James Brotherston PI)
    - Developing a cyclic proof theory for program verification
    - Implementing it in the CYCLIST automatic verifier
- My work so far
    - Extending the theoretical framework to verify (a reasonable class of) procedural programs
    - + some implementation

Program Verification with Separation Logic

We use a Hoare-style reasoning system[1,2] where

- assertions are formulas of *separation logic*:

Terms	$t ::= \mathrm{nil} \mid x \mid y \mid \ldots$
Formulas	$F, G ::= \mathrm{emp} \mid x \overset{\mathbf{f}}{\mapsto} \mathbf{t} \mid F * G \mid \ldots$ (classical predicate logic)

  whose models are heaps (with variable stores)

- Hoare triples $\{F\}\ C\ \{G\}$ assert program correctness
- Proof rules allow for symbolic execution and *local* reasoning

$$\frac{\vdash \left\{x \overset{f}{\mapsto} t * F\right\}\ C\ \{G\}}{\vdash \left\{x \overset{f}{\mapsto} t' * F\right\}\ x.f\ :=\ t; C\ \{G\}}\ \text{(fld-write)} \qquad \frac{\vdash \{G\}\ C\ \{H\}}{\vdash \{G * F\}\ C\ \{H * F\}}\ \text{(frame)}$$

[1] P. O'Hearn, J. Reynolds, H. Yang: Local Reasoning about Programs that Alter Data Structures. CSL 2001

[2] J. Berdine, C. Calcagno, P. O'Hearn: Symbolic Execution with Separation Logic. APLAS 2005

Reasoning About Inductive Data Structures

- The logic also makes use of *inductive predicates*

$$x = y \wedge \mathsf{emp} \Rightarrow \mathsf{ls}(x, y) \qquad\qquad x = \mathsf{nil} \wedge \mathsf{emp} \Rightarrow \mathsf{bt}(x)$$

$$\exists z. x \xmapsto{\texttt{nxt}} z * \mathsf{ls}(z, y) \Rightarrow \mathsf{ls}(x, y) \qquad \exists y, z. x \xmapsto{\texttt{l,r}} (y, z) * \mathsf{bt}(y) * \mathsf{bt}(z) \Rightarrow \mathsf{bt}(x)$$

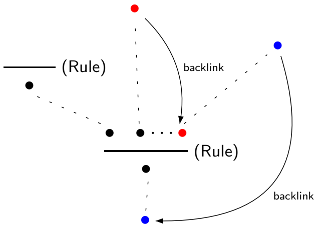
- We incorporate proof rules for unfolding these predicates

$$\frac{\vdash \{\Gamma(t_1 = t_2 \wedge \mathsf{emp})\}\ C\ \{F\} \quad \vdash \left\{\Gamma(\exists z. t_1 \xmapsto{\texttt{nxt}} z * \mathsf{ls}(z, t_2))\right\}\ C\ \{F\}}{\vdash \{\Gamma(\mathsf{ls}(t_1, t_2))\}\ C\ \{F\}}\ \text{(unfold)}$$

which may then allow further symbolic execution rules to fire

- Many automatic verifiers use a fixed set of inductive predicates
- Cyclist works with *general* (i.e. user-defined) predicates

The Cyclic Proof Technique



- In standard proof theory, proofs are derivation *trees*
- In cyclic proof theory, we allow cycles in derivations
    - global soundness: some inductive predicate is unfolded infinitely often along each infinite path

The State of CYCLIST in May 2014

- A framework and implementation for proving:
  - separation logic entailments[3]
  - safety/termination of simple `while` programs[4,5]

$$\begin{array}{rl} \text{Branching Conditions} & B ::= \star \mid t = t \mid t \neq t \\ \text{Program Commands} & C ::= \epsilon \mid x := t; C \mid x := y.f; C \mid x.f := y; C \mid \texttt{new}(\mathbf{f}); C \mid \\ & \quad \texttt{free}(x); C \mid \texttt{if}\ B \{ C \}\ \texttt{else} \{ C \}; C \mid \texttt{while}\ B \{ C \}; C \end{array}$$

- Loops can be symbolically executed in a uniform way

$$\frac{\vdash \{[\![B]\!] \wedge F\}\ C; \texttt{while}\ B \{ C \}; C'\ \{G\} \quad \vdash \left\{[\![\overline{B}]\!] \wedge F\right\}\ C'\ \{G\}}{\vdash \{F\}\ \texttt{while}\ B \{ C \}; C'\ \{G\}}$$

- Loop termination is proved through back-linking

---

3 J. Brotherston: Formalised Inductive Reasoning in the Logic of Bunched Implications. SAS 2007

4 J. Brotherston, R. Bornat, C. Calcagno: Cyclic proofs of program termination in separation logic. POPL 2008

5 J. Brotherston, N. Gorogiannis: Cyclic Abduction of Inductively Defined Safety and Termination Preconditions. SAS 2014

Worked Example

(Deallocating a Linked List)

$\texttt{// pre:}\ \mathsf{ls}(\mathtt{x}, \mathsf{nil})$

$\texttt{while x} \neq \mathsf{nil}\ \{\ \texttt{t} := \texttt{x.nxt};\ \texttt{free}(\texttt{x});\ \texttt{x} := \texttt{t};\ \}$

$\texttt{// post:}\ \mathsf{emp}$

Worked Example

(Deallocating a Linked List)

$$\begin{array}{l} \texttt{// pre: } \mathsf{ls}(\texttt{x}, \mathsf{nil}) \\ \texttt{while x} \neq \mathsf{nil} \; \{ \, \texttt{t} := \texttt{x.nxt}; \, \texttt{free}(\texttt{x}); \, \texttt{x} := \texttt{t}; \} \\ \texttt{// post: } \mathsf{emp} \end{array}$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \left\{ \begin{array}{c} \texttt{x} \neq \mathsf{nil} \\ \wedge\, \texttt{x} \neq \mathsf{nil} \\ \wedge\, \mathsf{emp} \end{array} \right\} \; \texttt{t} := \ldots \; \{\mathsf{emp}\}}\,(\bot) \qquad \dfrac{\dfrac{\dfrac{\vdash \{\mathsf{ls}(\texttt{x}, \mathsf{nil})\} \; \texttt{while} \ldots \; \{\mathsf{emp}\}}{\vdash \{\mathsf{ls}(\texttt{t}, \mathsf{nil})\} \; \texttt{x} := \ldots \; \{\mathsf{emp}\}}}{\vdash \left\{ \begin{array}{c} \texttt{x} \overset{\texttt{nxt}}{\longmapsto} \texttt{t} \\ *\, \mathsf{ls}(\texttt{t}, \mathsf{nil}) \end{array} \right\} \; \texttt{free}(\texttt{x}) \; \{\mathsf{emp}\}}}{\vdash \left\{ \begin{array}{c} \texttt{x} \overset{\texttt{nxt}}{\longmapsto} y \\ *\, \mathsf{ls}(y, \mathsf{nil}) \end{array} \right\} \; \texttt{t} := \ldots \; \{\mathsf{emp}\}}}{\vdash \left\{ \begin{array}{c} \texttt{x} \neq \mathsf{nil} \\ \wedge\, \mathsf{ls}(\texttt{x}, \mathsf{nil}) \end{array} \right\} \; \texttt{t} := \ldots \; \{\mathsf{emp}\}}\,(\text{unfold}) \qquad \vdash \left\{ \begin{array}{c} \texttt{x} = \mathsf{nil} \\ \wedge\, \mathsf{ls}(\texttt{x}, \mathsf{nil}) \end{array} \right\} \; \epsilon \; \{\mathsf{emp}\}}{\vdash \{\mathsf{ls}(\texttt{x}, \mathsf{nil})\} \; \texttt{while} \ldots \; \{\mathsf{emp}\}}$$

Worked Example

(Deallocating a Linked List)

$$\texttt{// pre:}\ \mathsf{ls}(\mathtt{x}, \mathsf{nil})$$

$$\texttt{while}\ \mathtt{x} \neq \mathsf{nil}\ \{\, \mathtt{t} := \mathtt{x.nxt};\ \texttt{free}(\mathtt{x});\ \mathtt{x} := \mathtt{t};\, \}$$

$$\texttt{// post:}\ \mathsf{emp}$$

$$
\dfrac{
\dfrac{
\dfrac{}{\vdash \left\{\begin{array}{c} \mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathsf{emp} \end{array}\right\} \mathtt{t} := \ldots \{\mathsf{emp}\}}(\bot)
\qquad
\dfrac{
\dfrac{
\dfrac{\vdash \{\mathsf{ls}(\mathtt{x}, \mathsf{nil})\}\ \texttt{while} \ldots \{\mathsf{emp}\}}{\vdash \{\mathsf{ls}(\mathtt{t}, \mathsf{nil})\}\ \mathtt{x} := \ldots \{\mathsf{emp}\}}
}{\vdash \left\{\begin{array}{c} \mathtt{x} \xmapsto{\mathtt{nxt}} \mathtt{t} \\ *\ \mathsf{ls}(\mathtt{t}, \mathsf{nil}) \end{array}\right\} \texttt{free}(\mathtt{x})\ \{\mathsf{emp}\}}
}{\vdash \left\{\begin{array}{c} \mathtt{x} \xmapsto{\mathtt{nxt}} y \\ *\ \mathsf{ls}(y, \mathsf{nil}) \end{array}\right\} \mathtt{t} := \ldots \{\mathsf{emp}\}}
}{\vdash \left\{\begin{array}{c} \mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x}, \mathsf{nil}) \end{array}\right\} \mathtt{t} := \ldots \{\mathsf{emp}\}}(\text{unfold})
\qquad
\vdash \left\{\begin{array}{c} \mathtt{x} = \mathsf{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x}, \mathsf{nil}) \end{array}\right\} \epsilon\ \{\mathsf{emp}\}
}{\vdash \{\mathsf{ls}(\mathtt{x}, \mathsf{nil})\}\ \texttt{while} \ldots \{\mathsf{emp}\}}
$$

Worked Example

(Deallocating a Linked List)

// pre: $\mathsf{ls}(\mathtt{x}, \mathsf{nil})$

$\mathtt{while}\ \mathtt{x} \neq \mathsf{nil}\ \{\, \mathtt{t} := \mathtt{x.nxt};\ \mathtt{free}(\mathtt{x});\ \mathtt{x} := \mathtt{t};\, \}$

// post: emp

$$
\dfrac{
\dfrac{
\dfrac{}{\vdash \left\{ \begin{array}{c} \mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathsf{emp} \end{array} \right\} \mathtt{t} := \ldots \{\mathsf{emp}\}}\,(\bot)
\qquad
\dfrac{
\dfrac{
\dfrac{
\vdash \{\mathsf{ls}(\mathtt{x}, \mathsf{nil})\}\ \mathtt{while} \ldots \{\mathsf{emp}\}
}{\vdash \{\mathsf{ls}(\mathtt{t}, \mathsf{nil})\}\ \mathtt{x} := \ldots \{\mathsf{emp}\}}
}{\vdash \left\{ \begin{array}{c} \mathtt{x} \xrightarrow{\mathtt{nxt}} \mathtt{t} \\ *\ \mathsf{ls}(\mathtt{t}, \mathsf{nil}) \end{array} \right\} \mathtt{free}(\mathtt{x})\ \{\mathsf{emp}\}}
}{\vdash \left\{ \begin{array}{c} \mathtt{x} \xrightarrow{\mathtt{nxt}} y \\ *\ \mathsf{ls}(y, \mathsf{nil}) \end{array} \right\} \mathtt{t} := \ldots \{\mathsf{emp}\}}
}{\vdash \left\{ \begin{array}{c} \mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x}, \mathsf{nil}) \end{array} \right\} \mathtt{t} := \ldots \{\mathsf{emp}\}}\,(\text{unfold})
\qquad
\vdash \left\{ \begin{array}{c} \mathtt{x} = \mathsf{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x}, \mathsf{nil}) \end{array} \right\} \epsilon\ \{\mathsf{emp}\}
}{\vdash \{\mathsf{ls}(\mathtt{x}, \mathsf{nil})\}\ \mathtt{while} \ldots \{\mathsf{emp}\}}
$$

Adding Procedures to the Proof System

- Extend the syntax (and operational semantics)

Proc. Decl.	$\texttt{proc}\, p(\mathbf{x})\,\{C\}$	(C must not modify parameters $\mathbf{x}$)
Commands	$C ::= \ldots \mid p(\mathbf{t}); C$	(arguments $\mathbf{t}$ passed by *value*)

- Add new proof rules

$$\text{(proc-unfold)}: \frac{\vdash \{F\}\ C\ \{G\}}{\vdash \{F\}\ p(\mathbf{x})\ \{G\}} \left(\begin{array}{c} \mathit{body}(p) = C,\ \mathit{params}(p) = \mathbf{x} \\ \mathit{locals}(p) \cap (\mathit{fv}(F) \cup \mathit{fv}(G)) = \emptyset \end{array}\right)$$

$$\text{(proc-call)}: \frac{\vdash \{H\}\ p(\mathbf{t})\ \{J\} \quad \vdash \{J * F\}\ C\ \{G\}}{\vdash \{H * F\}\ p(\mathbf{t}); C\ \{G\}} \quad \text{(param-subst)}: \frac{\vdash \{F\}\ p(\mathbf{t})\ \{G\}}{\vdash \{F[t/x]\}\ p(\mathbf{t})[t/x]\ \{G[t/x]\}}$$

- Fits very easily into the cyclic proof framework.
  - Proof rule are ‘standard’ Hoare logic for procedures
  - Symbolic execution rule has built-in framing

Tracing Predicates in Procedure Calls

(Interlude)

$$\text{(proc-call)}: \frac{\vdash \{H\}\ p(\mathbf{t})\ \{J\} \quad \vdash \{J * F\}\ C\ \{G\}}{\vdash \{H * F\}\ p(\mathbf{t});C\ \{G\}}$$

We trace predicates:

Tracing Predicates in Procedure Calls

(Interlude)

$$\text{(proc-call)}: \frac{\vdash \{H\}\ p(\mathbf{t})\ \{J\} \quad \vdash \{J * F\}\ C\ \{G\}}{\vdash \{H * F\}\ p(\mathbf{t}); C\ \{G\}}$$

We trace predicates:

- either through the procedure precondition

Tracing Predicates in Procedure Calls

(Interlude)

$$\text{(proc-call)}: \frac{\vdash \{H\}\ p(\mathbf{t})\ \{J\} \quad \vdash \{J * F\}\ C\ \{G\}}{\vdash \{H * F\}\ p(\mathbf{t}); C\ \{G\}}$$

We trace predicates:

- either through the procedure precondition
- or through the frame (bypassing the procedure)

Worked Example

(Recursively Deallocating a Binary Tree)

```
// pre: bt(x)
delTree(x) {if x ≠ nil { lt := x.l; rt := x.r; free(x); delTree(lt); delTree(rt); }}
// post: emp
```

Worked Example

(Recursively Deallocating a Binary Tree)

```
// pre: bt(x)
delTree(x) {if x ≠ nil { lt := x.l; rt := x.r; free(x); delTree(lt); delTree(rt); }}
// post: emp
```

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{lt})\}\ \mathtt{delTree}(\mathtt{lt})\ \{\mathsf{emp}\}}\,(\text{param})
      \qquad
      \dfrac{
        \dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{rt})\ \{\mathsf{emp}\}}\,(\text{param})
        \qquad
        \dfrac{}{\vdash \{\mathsf{emp} * \mathsf{emp}\}\ \epsilon\ \{\mathsf{emp}\}}
      }{\vdash \{\mathsf{emp} * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{rt}); \ldots\ \{\mathsf{emp}\}}\,(\text{proc-call})
    }{\vdash \{\mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{lt}); \ldots\ \{\mathsf{emp}\}}\,(\text{proc-call})
  }{
    \dfrac{\vdash \left\{\begin{array}{c} \mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \overset{\mathtt{l},\mathtt{r}}{\longmapsto} \mathtt{lt}, \mathtt{rt} * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt}) \end{array}\right\}\ \mathtt{free}(\mathtt{x}); \ldots\ \{\mathsf{emp}\}}{
      \dfrac{\vdash \left\{\begin{array}{c} \mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \overset{\mathtt{l},\mathtt{r}}{\longmapsto} \mathtt{lt}, z * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(z) \end{array}\right\}\ \mathtt{rt}\ := \ \ldots\ \{\mathsf{emp}\}}{
        \vdash \left\{\begin{array}{c} \mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \overset{\mathtt{l},\mathtt{r}}{\longmapsto} y, z * \\ \mathsf{bt}(y) * \mathsf{bt}(z) \end{array}\right\}\ \mathtt{lt}\ := \ \ldots\ \{\mathsf{emp}\}
      }
    }
  }
}{}
$$

$$
\dfrac{
  \dfrac{
    \dfrac{}{\vdash \left\{\begin{array}{c} \mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} = \mathsf{nil} \\ \wedge\ \mathsf{emp} \end{array}\right\}\ \mathtt{lt}\ := \ \ldots\ \{\mathsf{emp}\}}
    \qquad
    \vdash \left\{\begin{array}{c} \mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \overset{\mathtt{l},\mathtt{r}}{\longmapsto} y, z * \\ \mathsf{bt}(y) * \mathsf{bt}(z) \end{array}\right\}\ \mathtt{lt}\ := \ \ldots\ \{\mathsf{emp}\}
  }{\vdash \left\{\begin{array}{c} \mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathsf{bt}(\mathtt{x}) \end{array}\right\}\ \mathtt{lt}\ := \ \ldots\ \{\mathsf{emp}\}}\,(\text{unfold})
  \qquad
  \dfrac{}{\vdash \left\{\begin{array}{c} \mathtt{x} = \mathsf{nil} \\ \wedge\ \mathsf{bt}(\mathtt{x}) \end{array}\right\}\ \epsilon\ \{\mathsf{emp}\}}
}{
  \dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{if} \ldots\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}\,(\text{proc-unfold})
}
$$

Worked Example

(Recursively Deallocating a Binary Tree)

```
// pre: bt(x)
delTree(x) {if x ≠ nil { lt := x.l; rt := x.r; free(x); delTree(lt); delTree(rt); }}
// post: emp
```

$$
\dfrac{
  \dfrac{
    \dfrac{}{\vdash \left\{ \begin{array}{c} x \neq \mathsf{nil} \wedge x = \mathsf{nil} \\ \wedge\ \mathsf{emp} \end{array} \right\}\ \mathtt{lt}\ :=\ \ldots\ \{\mathsf{emp}\}}
    \qquad
    \dfrac{
      \dfrac{
        \dfrac{
          \dfrac{
            \dfrac{\vdash \{\mathsf{bt}(x)\}\ \mathtt{delTree}(x)\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{lt})\}\ \mathtt{delTree}(\mathtt{lt})\ \{\mathsf{emp}\}}\ (\text{param})
            \qquad
            \dfrac{
              \dfrac{\vdash \{\mathsf{bt}(x)\}\ \mathtt{delTree}(x)\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{rt})\ \{\mathsf{emp}\}}\ (\text{param})
              \quad
              \dfrac{}{\vdash \{\mathsf{emp} * \mathsf{emp}\}\ \epsilon\ \{\mathsf{emp}\}}
            }{\vdash \{\mathsf{emp} * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{rt}); \ldots\ \{\mathsf{emp}\}}\ (\text{proc-call})
          }{\vdash \{\mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{lt}); \ldots\ \{\mathsf{emp}\}}\ (\text{proc-call})
        }{\vdash \left\{ \begin{array}{c} x \neq \mathsf{nil} \wedge x \xrightarrow{\mathtt{l,r}} \mathtt{lt}, \mathtt{rt}\ * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt}) \end{array} \right\}\ \mathtt{free}(x); \ldots\ \{\mathsf{emp}\}}
      }{\vdash \left\{ \begin{array}{c} x \neq \mathsf{nil} \wedge x \xrightarrow{\mathtt{l,r}} \mathtt{lt}, z\ * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(z) \end{array} \right\}\ \mathtt{rt}\ :=\ \ldots\ \{\mathsf{emp}\}}
    }{\vdash \left\{ \begin{array}{c} x \neq \mathsf{nil} \wedge x \xrightarrow{\mathtt{l,r}} y, z\ * \\ \mathsf{bt}(y) * \mathsf{bt}(z) \end{array} \right\}\ \mathtt{lt}\ :=\ \ldots\ \{\mathsf{emp}\}}
  }{\vdash \left\{ \begin{array}{c} x \neq \mathsf{nil} \\ \wedge\ \mathsf{bt}(x) \end{array} \right\}\ \mathtt{lt}\ :=\ \ldots\ \{\mathsf{emp}\}}\ (\text{unfold})
  \qquad
  \dfrac{}{\vdash \left\{ \begin{array}{c} x = \mathsf{nil} \\ \wedge\ \mathsf{bt}(x) \end{array} \right\}\ \epsilon\ \{\mathsf{emp}\}}
}{
  \dfrac{\vdash \{\mathsf{bt}(x)\}\ \mathtt{if} \ldots\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(x)\}\ \mathtt{delTree}(x)\ \{\mathsf{emp}\}}\ (\text{proc-unfold})
}
$$

Worked Example

(Recursively Deallocating a Binary Tree)

$$
\begin{array}{l}
\text{// pre: } \mathsf{bt}(\mathtt{x}) \\
\mathtt{delTree}(\mathtt{x})\ \{\mathsf{if}\ \mathtt{x} \neq \mathsf{nil}\ \{\ \mathtt{lt}\ := \mathtt{x.l};\ \mathtt{rt}\ := \mathtt{x.r};\ \mathtt{free}(\mathtt{x});\ \mathtt{delTree}(\mathtt{lt});\ \mathtt{delTree}(\mathtt{rt});\ \}\} \\
\text{// post: } \mathsf{emp}
\end{array}
$$

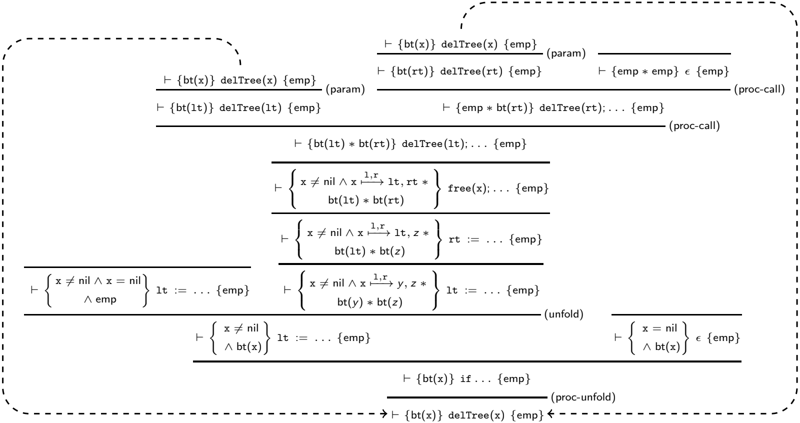
$$
\dfrac{
\dfrac{}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} = \mathsf{nil} \\ \wedge\ \mathsf{emp}\end{array}\right\} \mathtt{lt}\ := \ \ldots\ \{\mathsf{emp}\}}
\quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{lt})\}\ \mathtt{delTree}(\mathtt{lt})\ \{\mathsf{emp}\}}\,(\text{param})
\quad
\dfrac{
\dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{rt})\ \{\mathsf{emp}\}}\,(\text{param})
\quad
\dfrac{}{\vdash \{\mathsf{emp} * \mathsf{emp}\}\ \epsilon\ \{\mathsf{emp}\}}
}{\vdash \{\mathsf{emp} * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{rt}); \ldots\ \{\mathsf{emp}\}}\,(\text{proc-call})
}{\vdash \{\mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{lt}); \ldots\ \{\mathsf{emp}\}}\,(\text{proc-call})
}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l,r}} \mathtt{lt}, \mathtt{rt} * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt})\end{array}\right\} \mathtt{free}(\mathtt{x}); \ldots\ \{\mathsf{emp}\}}
}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l,r}} \mathtt{lt}, z * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(z)\end{array}\right\} \mathtt{rt}\ := \ \ldots\ \{\mathsf{emp}\}}
}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l,r}} y, z * \\ \mathsf{bt}(y) * \mathsf{bt}(z)\end{array}\right\} \mathtt{lt}\ := \ \ldots\ \{\mathsf{emp}\}}
}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathsf{bt}(\mathtt{x})\end{array}\right\} \mathtt{lt}\ := \ \ldots\ \{\mathsf{emp}\}}\,(\text{unfold})
\quad
\dfrac{}{\vdash \left\{\begin{array}{c}\mathtt{x} = \mathsf{nil} \\ \wedge\ \mathsf{bt}(\mathtt{x})\end{array}\right\} \epsilon\ \{\mathsf{emp}\}}
$$

$$
\dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{if} \ldots\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}\,(\text{proc-unfold})
$$

Worked Example

(Recursively Deallocating a Binary Tree)

```
// pre: bt(x)
delTree(x) {if x ≠ nil { lt := x.l; rt := x.r; free(x); delTree(lt); delTree(rt); }}
// post: emp
```



Worked Example

(Recursively Deallocating a Binary Tree)

```
// pre: bt(x)
delTree(x) {if x ≠ nil { lt := x.l; rt := x.r; free(x); delTree(lt); delTree(rt); }}
// post: emp
```

$$\dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{lt})\}\ \mathtt{delTree}(\mathtt{lt})\ \{\mathsf{emp}\}}\,(\text{param}) \qquad \dfrac{\dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{rt})\ \{\mathsf{emp}\}}\,(\text{param}) \qquad \dfrac{}{\vdash \{\mathsf{emp} * \mathsf{emp}\}\ \epsilon\ \{\mathsf{emp}\}}}{\vdash \{\mathsf{emp} * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{rt}); \ldots\ \{\mathsf{emp}\}}\,(\text{proc-call})$$

$$\dfrac{\cdots}{\vdash \{\mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{lt}); \ldots\ \{\mathsf{emp}\}}\,(\text{proc-call})$$

$$\dfrac{\vdash \{\mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt})\}\ \mathtt{delTree}(\mathtt{lt}); \ldots\ \{\mathsf{emp}\}}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l},\mathtt{r}} \mathtt{lt}, \mathtt{rt} * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt})\end{array}\right\}\ \mathtt{free}(\mathtt{x}); \ldots\ \{\mathsf{emp}\}}$$

$$\dfrac{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l},\mathtt{r}} \mathtt{lt}, \mathtt{rt} * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(\mathtt{rt})\end{array}\right\}\ \mathtt{free}(\mathtt{x}); \ldots\ \{\mathsf{emp}\}}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l},\mathtt{r}} \mathtt{lt}, z * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(z)\end{array}\right\}\ \mathtt{rt}\ :=\ \ldots\ \{\mathsf{emp}\}}$$

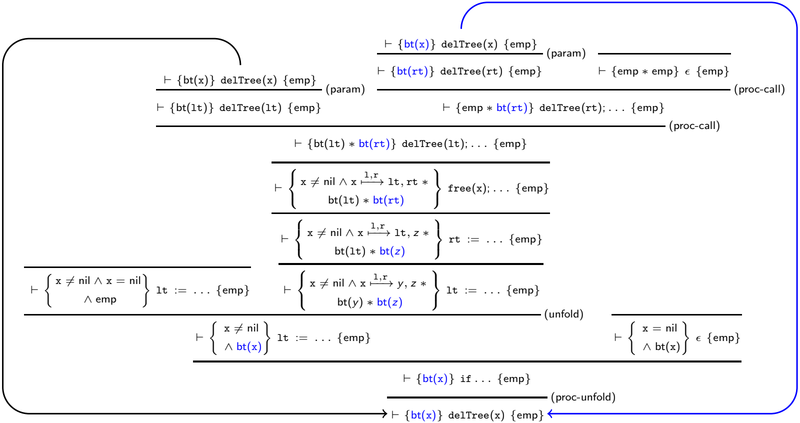
$$\dfrac{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l},\mathtt{r}} \mathtt{lt}, z * \\ \mathsf{bt}(\mathtt{lt}) * \mathsf{bt}(z)\end{array}\right\}\ \mathtt{rt}\ :=\ \ldots\ \{\mathsf{emp}\}}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l},\mathtt{r}} y, z * \\ \mathsf{bt}(y) * \mathsf{bt}(z)\end{array}\right\}\ \mathtt{lt}\ :=\ \ldots\ \{\mathsf{emp}\}}$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} = \mathsf{nil} \\ \wedge\ \mathsf{emp}\end{array}\right\}\ \mathtt{lt}\ :=\ \ldots\ \{\mathsf{emp}\}} \qquad \vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{l},\mathtt{r}} y, z * \\ \mathsf{bt}(y) * \mathsf{bt}(z)\end{array}\right\}\ \mathtt{lt}\ :=\ \ldots\ \{\mathsf{emp}\}}{\vdash \left\{\begin{array}{c}\mathtt{x} \neq \mathsf{nil} \\ \wedge\ \mathsf{bt}(\mathtt{x})\end{array}\right\}\ \mathtt{lt}\ :=\ \ldots\ \{\mathsf{emp}\}}\,(\text{unfold}) \qquad \dfrac{}{\vdash \left\{\begin{array}{c}\mathtt{x} = \mathsf{nil} \\ \wedge\ \mathsf{bt}(\mathtt{x})\end{array}\right\}\ \epsilon\ \{\mathsf{emp}\}}}{\dfrac{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{if} \ldots\ \{\mathsf{emp}\}}{\vdash \{\mathsf{bt}(\mathtt{x})\}\ \mathtt{delTree}(\mathtt{x})\ \{\mathsf{emp}\}}\,(\text{proc-unfold})}$$

Worked Example

(Recursively Deallocating a Binary Tree)

```
// pre: bt(x)
delTree(x) {if x ≠ nil { lt := x.l; rt := x.r; free(x); delTree(lt); delTree(rt); }}
// post: emp
```



How to Account for the Effects of Procedures?

(programs for which we *should* be able to prove termination)

$//\ \texttt{pre:}\ \texttt{x} \xmapsto{\texttt{val}} v * \mathsf{ls}(v, \mathsf{nil})$

$\texttt{proc delHead(x)}\ \{\ \texttt{l := x.val};\ \texttt{if}\ \texttt{l} \neq \mathsf{nil}\ \{\ \texttt{y := l.nxt};\ \texttt{free(l)};\ \texttt{x.val := y};\ \}\ \}$

$//\ \texttt{post:}\ \exists w.\ \texttt{x} \xmapsto{\texttt{val}} w * \mathsf{ls}(w, \mathsf{nil})$

How to Account for the Effects of Procedures?

(programs for which we *should* be able to prove termination)

// pre: $\mathtt{x} \xmapsto{\mathtt{val}} v * \mathsf{ls}(v, \mathsf{nil})$

$\mathtt{proc}\ \mathtt{delHead}(\mathtt{x})\ \{\ \mathtt{l}\ := \ \mathtt{x.val};\ \mathtt{if}\ \mathtt{l} \neq \mathsf{nil}\ \{\ \mathtt{y}\ := \ \mathtt{l.nxt};\ \mathtt{free}(\mathtt{l});\ \mathtt{x.val}\ := \ \mathtt{y};\ \}\ \}$

// post: $\exists w.\, \mathtt{x} \xmapsto{\mathtt{val}} w * \mathsf{ls}(w, \mathsf{nil})$

// pre: $\mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil})$

$\mathtt{while}\ \mathtt{v} \neq \mathsf{nil}\ \{\ \mathtt{delHead}(\mathtt{x});\ \mathtt{v}\ := \ \mathtt{x.val};\ \}$ // terminates!

// post: $\mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil}$

How to Account for the Effects of Procedures?

(programs for which we *should* be able to prove termination)

$// \text{ pre: } \mathtt{x} \xmapsto{\mathtt{val}} v * \mathsf{ls}(v, \mathsf{nil})$
$\mathtt{proc\ delHead(x)}\ \{\ \mathtt{l} := \mathtt{x.val};\ \mathtt{if}\ \mathtt{l} \neq \mathsf{nil}\ \{\ \mathtt{y} := \mathtt{l.nxt};\ \mathtt{free(l)};\ \mathtt{x.val} := \mathtt{y};\ \}\ \}$
$// \text{ post: } \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w * \mathsf{ls}(w, \mathsf{nil})$

$// \text{ pre: } \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil})$
$\mathtt{while}\ \mathtt{v} \neq \mathsf{nil}\ \{\ \mathtt{delHead(x)};\ \mathtt{v} := \mathtt{x.val};\ \}$ $\qquad$ $//\ \mathtt{terminates!}$
$// \text{ post: } \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil}$

$$
\dfrac{
\dfrac{
\dfrac{\text{(proof of procedure)} \atop \vdots}{\vdash \left\{ \begin{matrix} \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} \\ * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{delHead(x)} \left\{ \begin{matrix} \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w \\ * \mathsf{ls}(w, \mathsf{nil}) \end{matrix} \right\}}
\quad
\dfrac{\vdash \left\{ \begin{matrix} \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} \\ * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{while} \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}{\vdash \left\{ \begin{matrix} \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w \\ * \mathsf{ls}(w, \mathsf{nil}) \end{matrix} \right\} \mathtt{v} := \mathtt{x.val}; \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
}{
\vdash \left\{ \begin{matrix} \mathtt{v} \neq \mathsf{nil} \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{delHead(x)}; \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}
}\ \text{(proc-call)}
\atop \vdots
\quad
\dfrac{}{\vdash \left\{ \begin{matrix} \mathtt{v} = \mathsf{nil} \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \epsilon \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
}{
\vdash \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \right\} \mathtt{while} \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}
}
$$

How to Account for the Effects of Procedures?

(programs for which we *should* be able to prove termination)

$// \ \mathtt{pre:}\ \mathtt{x} \xmapsto{\mathtt{val}} v * \mathsf{ls}(v, \mathsf{nil})$

$\mathtt{proc\ delHead(x)}\ \{\ \mathtt{l} := \mathtt{x.val};\ \mathtt{if}\ \mathtt{l} \neq \mathsf{nil}\ \{\ \mathtt{y} := \mathtt{l.nxt};\ \mathtt{free(l)};\ \mathtt{x.val} := \mathtt{y};\ \}\ \}$

$// \ \mathtt{post:}\ \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w * \mathsf{ls}(w, \mathsf{nil})$

$// \ \mathtt{pre:}\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil})$

$\mathtt{while}\ \mathtt{v} \neq \mathsf{nil}\ \{\ \mathtt{delHead(x)};\ \mathtt{v} := \mathtt{x.val};\ \}$ $\qquad //\ \mathtt{terminates!}$

$// \ \mathtt{post:}\ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil}$

$$
\dfrac{
\dfrac{
\dfrac{\text{(proof of procedure)}}{\vdots}
\quad
\vdash \left\{ \begin{matrix} \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} \\ * \, \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{delHead(x)} \left\{ \begin{matrix} \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w \\ * \, \mathsf{ls}(w, \mathsf{nil}) \end{matrix} \right\}
\qquad
\dfrac{\vdash \left\{ \begin{matrix} \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} \\ * \, \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{while} \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
{\vdash \left\{ \begin{matrix} \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w \\ * \, \mathsf{ls}(w, \mathsf{nil}) \end{matrix} \right\} \mathtt{v} := \mathtt{x.val}; \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
}{
\vdash \left\{ \begin{matrix} \mathtt{v} \neq \mathsf{nil} \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{delHead(x)}; \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}
}\ \text{(proc-call)}
\quad \vdots
\qquad
\dfrac{}{\vdash \left\{ \begin{matrix} \mathtt{v} = \mathsf{nil} \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \epsilon \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
}{
\vdash \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \right\} \mathtt{while} \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}
}
$$

How to Account for the Effects of Procedures?

(programs for which we *should* be able to prove termination)

$// \texttt{ pre: } \mathtt{x} \xmapsto{\mathtt{val}} v * \mathsf{ls}(v, \mathsf{nil})$

$\mathtt{proc\ delHead(x)}\ \{\ \mathtt{l}\ := \mathtt{x.val};\ \mathtt{if}\ \mathtt{l} \neq \mathsf{nil}\ \{\ \mathtt{y}\ := \mathtt{l.nxt};\ \mathtt{free(l)};\ \mathtt{x.val}\ := \mathtt{y};\ \}\ \}$

$// \texttt{ post: } \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w * \mathsf{ls}(w, \mathsf{nil})$

$// \texttt{ pre: } \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil})$

$\mathtt{while}\ \mathtt{v} \neq \mathsf{nil}\ \{\ \mathtt{delHead(x)};\ \mathtt{v}\ := \mathtt{x.val};\ \}$ $\qquad //\ \texttt{terminates!}$

$// \texttt{ post: } \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil}$

$$
\dfrac{
\dfrac{
\dfrac{\text{(proof of procedure)} \\ \vdots}{\vdash \left\{ \begin{array}{c} \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} \\ * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{array} \right\} \mathtt{delHead(x)} \left\{ \begin{array}{c} \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w \\ * \mathsf{ls}(w, \mathsf{nil}) \end{array} \right\}}
\quad
\dfrac{\vdash \left\{ \begin{array}{c} \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} \\ * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{array} \right\} \mathtt{while}\ \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}{\vdash \left\{ \begin{array}{c} \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w \\ * \mathsf{ls}(w, \mathsf{nil}) \end{array} \right\} \mathtt{v}\ := \mathtt{x.val};\ \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
}{\vdash \left\{ \begin{array}{c} \mathtt{v} \neq \mathsf{nil}\ \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{array} \right\} \mathtt{delHead(x)};\ \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}} \text{(proc-call)}
\\ \vdots
\quad
\dfrac{}{\vdash \left\{ \begin{array}{c} \mathtt{v} = \mathsf{nil}\ \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \end{array} \right\} \epsilon \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
}{\vdash \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}(\mathtt{v}, \mathsf{nil}) \right\} \mathtt{while}\ \ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
$$

Making Use of Explicit Approximations

Syntax and Semantics

- We need to identify (label) approximants explicitly:

$$\text{Formulas} \qquad F, G ::= \ldots \mid P_\alpha(\mathbf{t}) \mid \ldots$$

  $P_\alpha(\mathbf{t})$ is interpreted in the α^{th} approximant

- We also make use of sets Ω of constraints $\beta < \alpha$
- We combine constraints with formulas (and allow existential quantification over labels):

$$\text{Constrained Formulas} \qquad \phi, \psi ::= \exists \boldsymbol{\alpha} \,.\, \Omega : F$$

  Models for constrained formulas are tuples (ρ, s, h) of label-to-ordinal valuations, variable stores and heaps

Making Use of Explicit Approximations

Proof Rules and Tracing

- Hoare triples $\{\phi\}\ C\ \{\psi\}$ now use constrained formulas
- Predicate unfolding introduces new labels and constraints, e.g.

$$\frac{\vdash \{\exists \boldsymbol{\alpha} \,.\, \Omega : t_1 = t_2 \wedge \mathsf{emp} * F\}\ C\ \{\psi\} \qquad \vdash \left\{ \begin{array}{c} \exists \boldsymbol{\alpha} \cup \{\beta\} \,.\, \Omega \cup \{\beta < \alpha\} : \\ \exists v.\ t_1 \xmapsto{\mathtt{nxt}} v * \mathsf{ls}_\beta(v, t_2) * F \end{array} \right\} C\ \{\psi\}}{\vdash \{\exists \boldsymbol{\alpha} \,.\, \Omega : \mathsf{ls}_\alpha(t_1, t_2) * F\}\ C\ \{\psi\}}\ (\beta \text{ fresh})$$

- We trace label α in conclusions to label β in premises:

$$\frac{\ldots \quad \vdash \{\chi\}\ C_2\ \{\xi\} \quad \ldots}{\vdash \{\phi\}\ C_1\ \{\psi\}} \qquad \begin{array}{l} \beta \in \mathsf{labels}(\chi),\ \chi = \exists \boldsymbol{\beta} \,.\, \Omega_2 : G \\ \\ \alpha \in \mathsf{labels}(\phi),\ \phi = \exists \boldsymbol{\alpha} \,.\, \Omega_1 : F \end{array}$$

  whenever we can infer $\beta \leq \alpha$ from Ω_2
  - If also $\beta < \alpha$ then the trace progresses (cf. rule above)
- We must also be able to rename labels and ‘frame’ constraints

How to Account for the Effects of Procedures

```
// pre: v ≠ nil ∧ x ↦val v ∗ ls_α(v, nil)
proc delHead(x) { l := x.val; if l ≠ nil { y := l.nxt; free(l); x.val := y; } }
// post: ∃β < α : ∃w. x ↦val w ∗ ls_β(w, nil)
```

$$// \text{ pre: } v \neq \mathsf{nil} \wedge \mathtt{x} \xmapsto{\mathtt{val}} v * \mathsf{ls}_\alpha(v, \mathsf{nil})$$

$$\mathtt{proc\ delHead(x)} \{ \mathtt{l} := \mathtt{x.val}; \mathtt{if}\ \mathtt{l} \neq \mathsf{nil} \{ \mathtt{y} := \mathtt{l.nxt}; \mathtt{free(l)}; \mathtt{x.val} := \mathtt{y}; \} \}$$

$$// \text{ post: } \exists \beta < \alpha : \exists w.\ \mathtt{x} \xmapsto{\mathtt{val}} w * \mathsf{ls}_\beta(w, \mathsf{nil})$$

How to Account for the Effects of Procedures

```
// pre: v ≠ nil ∧ x ↦val v * ls_α(v, nil)
proc delHead(x) { l := x.val; if l ≠ nil { y := l.nxt; free(l); x.val := y; } }
// post: ∃β < α : ∃w. x ↦val w * ls_β(w, nil)

// pre: x ↦val v * ls_α(v, nil)
while v ≠ nil { delHead(x); v := x.val; }            // provably terminates!
// post: x ↦val nil
```

$$
\dfrac{
  \dfrac{
    \dfrac{\text{(proof of procedure)}}{\vdots}
    \quad
    \vdash \left\{ \begin{matrix} \mathtt{v} \neq \mathsf{nil} \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} \\ * \, \mathsf{ls}_{\alpha}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{delHead(x)} \left\{ \begin{matrix} \exists \beta < \alpha : \\ \exists w.\, \mathtt{x} \xmapsto{\mathtt{val}} w \\ * \, \mathsf{ls}_{\beta}(w, \mathsf{nil}) \end{matrix} \right\}
    \qquad
    \dfrac{\vdash \left\{ \begin{matrix} \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} \\ * \, \mathsf{ls}_{\alpha}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{while}\ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}{\vdash \left\{ \begin{matrix} \exists \beta < \alpha : \\ \exists w.\, \mathtt{x} \xmapsto{\mathtt{val}} w \\ * \, \mathsf{ls}_{\beta}(w, \mathsf{nil}) \end{matrix} \right\} \mathtt{v := x.val};\ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}} \left(\begin{matrix} \text{relabel} \\ \text{wk-constr} \end{matrix} \right)
  }{
    \dfrac{\vdash \left\{ \begin{matrix} \mathtt{v} \neq \mathsf{nil} \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}_{\alpha}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \mathtt{delHead(x)};\ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}{\vdots}
  } \text{(proc-call)}
  \qquad
  \dfrac{}{\vdash \left\{ \begin{matrix} \mathtt{v} = \mathsf{nil} \wedge \\ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}_{\alpha}(\mathtt{v}, \mathsf{nil}) \end{matrix} \right\} \epsilon \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}}
}{
  \vdash \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathtt{v} * \mathsf{ls}_{\alpha}(\mathtt{v}, \mathsf{nil}) \right\} \mathtt{while}\ldots \left\{ \mathtt{x} \xmapsto{\mathtt{val}} \mathsf{nil} \right\}
}
$$

Conclusions

- Existing cyclic proof framework can be extended to handle procedures in a relatively straightforward way
- But previous implicit tracing strategy must be refined
- A much greater class of programs can now be verified
- Implementation is ongoing work