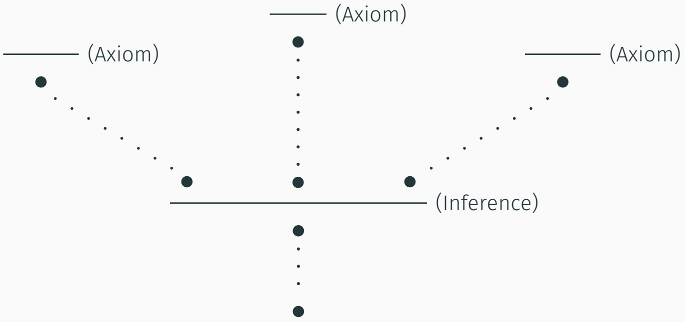
Program Verification Using Cyclic Proof

Reuben N. S. Rowe

University College London
Programming Principles, Logic and Verification Research Group (PPLV)

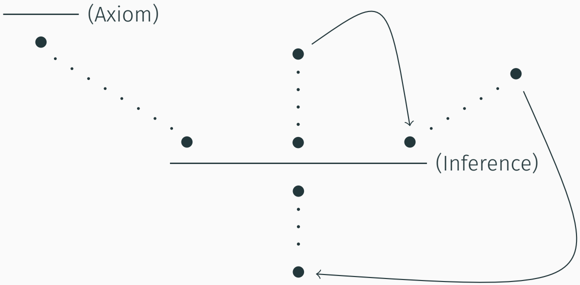
Computer Laboratory Programming Research Group Seminar
Thursday 19th May 2016

Prologue: It's Proof, But Not As We Know It

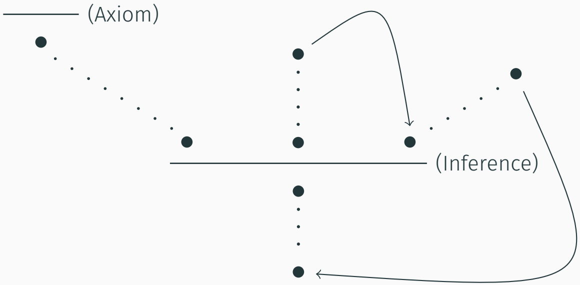


- We are all familiar with proofs as finite trees

Prologue: It's Proof, But Not As We Know It



- We are all familiar with proofs as finite trees
- But what if we allow proofs to be cyclic graphs instead?



- We are all familiar with proofs as finite trees
- But what if we allow proofs to be cyclic graphs instead?
- Cyclic proofs must satisfy a global soundness property

Introduction

- Our research programme has two broad aims:

Introduction

- Our research programme has two broad aims:
    - Develop cyclic proof (meta) theory in a verification setting

Introduction

- Our research programme has two broad aims:
  - Develop cyclic proof (meta) theory in a verification setting
  - Implement the techniques for automatic verification

Introduction

- Our research programme has two broad aims:
  - Develop cyclic proof (meta) theory in a verification setting
  - Implement the techniques for automatic verification

- Why cyclic proof?

Introduction

- Our research programme has two broad aims:
  - Develop cyclic proof (meta) theory in a verification setting
  - Implement the techniques for automatic verification
- Why cyclic proof?
  - It subsumes standard induction

Introduction

- Our research programme has two broad aims:
  - Develop cyclic proof (meta) theory in a verification setting
  - Implement the techniques for automatic verification

- Why cyclic proof?
  - It subsumes standard induction
  - It can help discover inductive hypotheses

Introduction

- Our research programme has two broad aims:
  - Develop cyclic proof (meta) theory in a verification setting
  - Implement the techniques for automatic verification

- Why cyclic proof?
  - It subsumes standard induction
  - It can help discover inductive hypotheses
  - Termination arguments can often be extracted from cyclic proofs

James Brotherston

Alex Simpson

Richard Bornat

Cristiano Calcagno

Dino Distefano

Nikos Gorogiannis

Example: First Order Logic

- Assume signature with zero, successor, and equality
- Allow inductive predicate definitions, e.g.

$$\frac{}{N\,0} \quad \frac{N\,x}{N\,sx} \qquad \frac{}{E\,0} \quad \frac{E\,x}{O\,sx} \quad \frac{O\,x}{E\,sx}$$

Example: First Order Logic

- Assume signature with zero, successor, and equality
- Allow inductive predicate definitions, e.g.

$$\frac{}{N\,0} \quad \frac{N\,x}{N\,sx} \qquad \frac{}{E\,0} \quad \frac{E\,x}{O\,sx} \quad \frac{O\,x}{E\,sx}$$

- These induce unfolding rules for the sequent calculus, e.g.

$$\frac{\Gamma \vdash \Delta, N\,t}{\Gamma \vdash \Delta, N\,st}\ (NR_2) \qquad \frac{\Gamma, t = 0 \vdash \Delta \quad \Gamma, t = sx, N\,x \vdash \Delta}{\Gamma, N\,t \vdash \Delta}\ (\text{Case } N)$$

where x is fresh

A Cyclic Proof of $N\ x \vdash E\ x, O\ x$

$$N\ x \vdash E\ x, O\ x$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$\dfrac{x = 0 \vdash E\,x, O\,x \qquad\qquad x = sy, N\,y \vdash E\,x, O\,x}{N\,x \vdash E\,x, O\,x}\ (\text{Case } N)$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$\dfrac{\dfrac{\vdash E\,0, O\,0}{x = 0 \vdash E\,x, O\,x}\,(=\text{L}) \qquad x = sy, N\,y \vdash E\,x, O\,x}{N\,x \vdash E\,x, O\,x}\,(\text{Case } N)$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$\dfrac{\dfrac{\dfrac{}{\vdash E\,0, O\,0}\,(E\mathrm{R}_1)}{x = 0 \vdash E\,x, O\,x}\,(=\mathrm{L}) \qquad x = sy, N\,y \vdash E\,x, O\,x}{N\,x \vdash E\,x, O\,x}\,(\text{Case } N)$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$
\dfrac{\dfrac{\dfrac{}{\vdash E\,0, O\,0}\,(E\mathrm{R}_1)}{x = 0 \vdash E\,x, O\,x}\,(=\mathrm{L}) \qquad \dfrac{N\,y \vdash E\,sy, O\,sy}{x = sy, N\,y \vdash E\,x, O\,x}\,(=\mathrm{L})}{N\,x \vdash E\,x, O\,x}\,(\text{Case } N)
$$

A Cyclic Proof of $N\ x \vdash E\ x, O\ x$

$$
\dfrac{\dfrac{\dfrac{}{\vdash E\ 0, O\ 0}\ (E\mathrm{R}_1)}{x = 0 \vdash E\ x, O\ x}\ (=\mathrm{L}) \qquad \dfrac{\dfrac{N\ y \vdash E\ y, O\ sy}{N\ y \vdash E\ sy, O\ sy}\ (E\mathrm{R}_2)}{x = sy, N\ y \vdash E\ x, O\ x}\ (=\mathrm{L})}{N\ x \vdash E\ x, O\ x}\ (\mathrm{Case}\ N)
$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$
\dfrac{
  \dfrac{
    \dfrac{}{\vdash E\,0, O\,0}\,(ER_1)
  }{x = 0 \vdash E\,x, O\,x}\,(=\!L)
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{N\,y \vdash E\,y, O\,y}{N\,y \vdash E\,y, O\,sy}\,(OR_1)
      }{N\,y \vdash E\,sy, O\,sy}\,(ER_2)
    }{x = sy, N\,y \vdash E\,x, O\,x}\,(=\!L)
  }{}
}{N\,x \vdash E\,x, O\,x}\,(\text{Case } N)
$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$
\dfrac{\dfrac{\dfrac{}{\vdash E\,0, O\,0}\,(ER_1)}{x = 0 \vdash E\,x, O\,x}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{N\,x \vdash E\,x, O\,x}{N\,y \vdash E\,y, O\,y}\,(\mathrm{Subst})}{N\,y \vdash E\,y, O\,sy}\,(OR_1)}{N\,y \vdash E\,sy, O\,sy}\,(ER_2)}{x = sy, N\,y \vdash E\,x, O\,x}\,(=\mathrm{L})}{N\,x \vdash E\,x, O\,x}\,(\mathrm{Case}\ N)
$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$
\dfrac{
\dfrac{
\dfrac{}{\vdash E\,0, O\,0}\,(E\mathrm{R}_1)
}{x = 0 \vdash E\,x, O\,x}\,(=\mathrm{L})
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{N\,x \vdash E\,x, O\,x}{N\,y \vdash E\,y, O\,y}\,(\mathrm{Subst})
}{N\,y \vdash E\,y, O\,sy}\,(O\mathrm{R}_1)
}{N\,y \vdash E\,sy, O\,sy}\,(E\mathrm{R}_2)
}{x = sy, N\,y \vdash E\,x, O\,x}\,(=\mathrm{L})
}{}
}{N\,x \vdash E\,x, O\,x}\,(\mathrm{Case}\ N)
$$

A Cyclic Proof of $N\ x \vdash E\ x, O\ x$

$$
\dfrac{
\dfrac{\dfrac{}{\vdash E\ 0, O\ 0}\ (ER_1)}{x = 0 \vdash E\ x, O\ x}\ (=\mathrm{L})
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{N\ x \vdash E\ x, O\ x}{N\ y \vdash E\ y, O\ y}\ (\mathrm{Subst})}{N\ y \vdash E\ y, O\ sy}\ (OR_1)}{N\ y \vdash E\ sy, O\ sy}\ (ER_2)}{x = sy, N\ y \vdash E\ x, O\ x}\ (=\mathrm{L})
}{N\ x \vdash E\ x, O\ x}\ (\mathrm{Case}\ N)
$$

- Suppose $N\ x \vdash E\ x, O\ x$ is not valid:

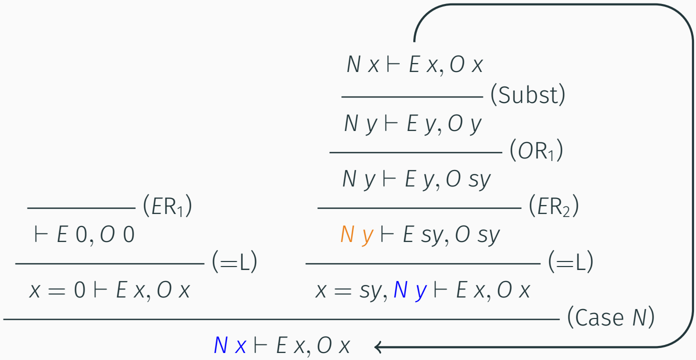
  $[\![X]\!]_{m_1}$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$
\dfrac{
\dfrac{\dfrac{}{\vdash E\,0, O\,0}\,(ER_1)}{x = 0 \vdash E\,x, O\,x}\,(=\mathrm{L})
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{N\,x \vdash E\,x, O\,x}{N\,y \vdash E\,y, O\,y}\,(\mathrm{Subst})}{N\,y \vdash E\,y, O\,sy}\,(OR_1)}{N\,y \vdash E\,sy, O\,sy}\,(ER_2)}{x = sy, {\color{orange}N\,y} \vdash E\,x, O\,x}\,(=\mathrm{L})
}{
{\color{blue}N\,x} \vdash E\,x, O\,x
}\,(\mathrm{Case}\ N)
$$

- Suppose $N\,x \vdash E\,x, O\,x$ is not valid:

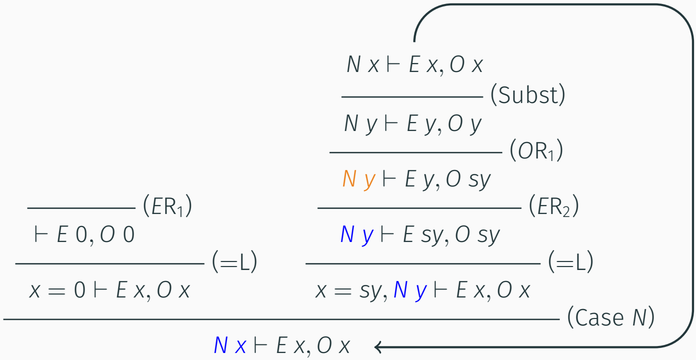
  $[\![x]\!]_{m_1} > {\color{orange}[\![y]\!]_{m_2}}$



- Suppose $N\ x \vdash E\ x, O\ x$ is not valid:

$$[\![x]\!]_{m_1} > [\![y]\!]_{m_2} = [\![y]\!]_{m_3}$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$



- Suppose $N\,x \vdash E\,x, O\,x$ is not valid:

$$[\![x]\!]_{m_1} > [\![y]\!]_{m_2} = [\![y]\!]_{m_3} = [\![y]\!]_{m_4}$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$\cfrac{\cfrac{\cfrac{}{\vdash E\,0, O\,0}\,(ER_1)}{x = 0 \vdash E\,x, O\,x}\,(=\mathrm{L}) \qquad \cfrac{\cfrac{\cfrac{\cfrac{N\,x \vdash E\,x, O\,x}{N\,y \vdash E\,y, O\,y}\,(\text{Subst})}{N\,y \vdash E\,y, O\,sy}\,(OR_1)}{N\,y \vdash E\,sy, O\,sy}\,(ER_2)}{x = sy, N\,y \vdash E\,x, O\,x}\,(=\mathrm{L})}{N\,x \vdash E\,x, O\,x}\,(\text{Case } N)$$

- Suppose $N\,x \vdash E\,x, O\,x$ is not valid:

$$[\![x]\!]_{m_1} > [\![y]\!]_{m_2} = [\![y]\!]_{m_3} = [\![y]\!]_{m_4} = [\![y]\!]_{m_5}$$

A Cyclic Proof of $N\ x \vdash E\ x, O\ x$

$$\dfrac{\dfrac{\dfrac{}{\vdash E\ 0, O\ 0}\,(ER_1)}{x = 0 \vdash E\ x, O\ x}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{N\ x \vdash E\ x, O\ x}{N\ y \vdash E\ y, O\ y}\,(\mathrm{Subst})}{N\ y \vdash E\ y, O\ sy}\,(OR_1)}{N\ y \vdash E\ sy, O\ sy}\,(ER_2)}{x = sy, N\ y \vdash E\ x, O\ x}\,(=\mathrm{L})}{N\ x \vdash E\ x, O\ x}\,(\mathrm{Case}\ N)$$

- Suppose $N\ x \vdash E\ x, O\ x$ is not valid:

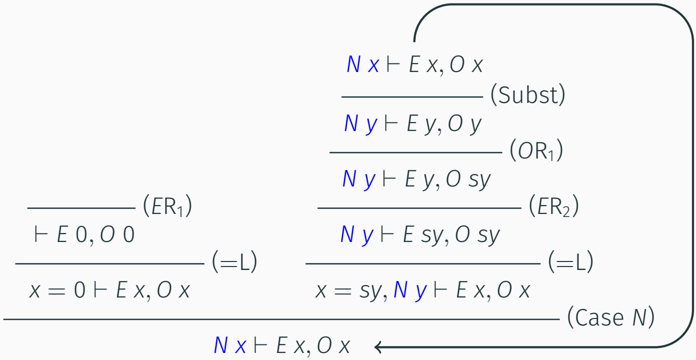
$$[\![x]\!]_{m_1} > [\![y]\!]_{m_2} = [\![y]\!]_{m_3} = [\![y]\!]_{m_4} = [\![y]\!]_{m_5} = [\![x]\!]_{m_6}$$

A Cyclic Proof of $N\,x \vdash E\,x, O\,x$

$$\dfrac{\dfrac{\dfrac{}{\vdash E\,0, O\,0}\,(ER_1)}{x = 0 \vdash E\,x, O\,x}\,(=L) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{N\,x \vdash E\,x, O\,x}{N\,y \vdash E\,y, O\,y}\,(\text{Subst})}{N\,y \vdash E\,y, O\,sy}\,(OR_1)}{N\,y \vdash E\,sy, O\,sy}\,(ER_2)}{x = sy, N\,y \vdash E\,x, O\,x}\,(=L)}{N\,x \vdash E\,x, O\,x}\,(\text{Case } N)$$

- Suppose $N\,x \vdash E\,x, O\,x$ is not valid:

$$[\![x]\!]_{m_1} > [\![y]\!]_{m_2} = [\![y]\!]_{m_3} = [\![y]\!]_{m_4} = [\![y]\!]_{m_5} = [\![x]\!]_{m_6} > [\![y]\!]_{m_7} \dots$$



- Suppose $N\,x \vdash E\,x, O\,x$ is not valid:

$$n_1 > n_2 > n_3 > \ldots$$

Example: Separation Logic

- Separation Logic incorporates formulas for representing heap memory:

Example: Separation Logic

- Separation Logic incorporates formulas for representing heap memory:
  - **emp** denotes the empty heap

Example: Separation Logic

- Separation Logic incorporates formulas for representing heap memory:
  - **emp** denotes the empty heap
  - $x \mapsto \vec{v}$ is the single-cell heap containing values $\vec{v}$ at memory location x

Example: Separation Logic

- Separation Logic incorporates formulas for representing heap memory:
  - **emp** denotes the empty heap
  - $x \mapsto \vec{v}$ is the single-cell heap containing values $\vec{v}$ at memory location x
  - $F * G$ denotes a heap h that can be split into disjoint sub-heaps h_1 and h_2 which model F and G respectively

Example: Separation Logic

- Separation Logic incorporates formulas for representing heap memory:
    - **emp** denotes the empty heap
    - $x \mapsto \vec{v}$ is the single-cell heap containing values $\vec{v}$ at memory location x
    - $F * G$ denotes a heap h that can be split into disjoint sub-heaps h_1 and h_2 which model F and G respectively
- Inductive predicates now represent data-structures, e.g. linked-list segments:

$$\frac{x = y \wedge \mathbf{emp}}{\mathsf{ls}(x, y)} \qquad \frac{x \mapsto z * \mathsf{ls}(z, y)}{\mathsf{ls}(x, y)}$$

A Cyclic Proof of List Segment Concatenation

$$\mathsf{ls}(x, y) * \mathsf{ls}(y, z) \vdash \mathsf{ls}(x, z)$$

A Cyclic Proof of List Segment Concatenation

$$\dfrac{\begin{array}{c}(x = y \land \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z) \\ \vdots \end{array} \qquad x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\ (\text{Case } \mathsf{ls})$$

A Cyclic Proof of List Segment Concatenation

$$\dfrac{\dfrac{\mathsf{emp} * \mathsf{ls}(x, z) \vdash \mathsf{ls}(x, z)}{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y, z) \vdash \mathsf{ls}(x, z)}\ (=\mathrm{L}) \qquad \vdots \qquad x \mapsto v * \mathsf{ls}(v, y) * \mathsf{ls}(y, z) \vdash \mathsf{ls}(x, z)}{\mathsf{ls}(x, y) * \mathsf{ls}(y, z) \vdash \mathsf{ls}(x, z)}\ (\text{Case } \mathsf{ls})$$

A Cyclic Proof of List Segment Concatenation

$$\dfrac{\dfrac{\dfrac{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}{\mathsf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\ (\equiv)}{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\ (=\mathrm{L}) \qquad x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\ (\text{Case } \mathsf{ls})$$

A Cyclic Proof of List Segment Concatenation

$$\dfrac{\dfrac{\dfrac{}{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\;(\mathrm{Id})}{\mathsf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\;(\equiv)}{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\;(=\mathrm{L})$$

$$\dfrac{\vdots \qquad x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\;(\text{Case } \mathsf{ls})$$

A Cyclic Proof of List Segment Concatenation

$$\dfrac{\dfrac{\dfrac{}{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Id})}{\mathsf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\equiv)}{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(=\mathrm{L})$$

$$\vdots$$

$$\dfrac{\vdots \qquad \dfrac{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash x \mapsto v * \mathsf{ls}(v,z)}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathsf{ls}\mathrm{R}_2)}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Case}\ \mathsf{ls})$$

A Cyclic Proof of List Segment Concatenation

$$
\dfrac{
\begin{array}{c}
\dfrac{\dfrac{\dfrac{}{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\ (\mathsf{Id})}{\mathbf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\ (\equiv)}{(x = y \wedge \mathbf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\ (=\mathsf{L}) \\ \vdots
\end{array}
\qquad
\dfrac{\dfrac{x \mapsto v \vdash x \mapsto v \qquad \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(v,z)}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash x \mapsto v * \mathsf{ls}(v,z)}\ (*)}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\ (\mathsf{lsR}_2)
}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\ (\text{Case } \mathsf{ls})
$$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Id})
}{\mathsf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\equiv)
}{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(=\mathrm{L})
\quad
\dfrac{
\dfrac{
\dfrac{}{x \mapsto v \vdash x \mapsto v}\,(\mathrm{Id})
\qquad
\mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(v,z)
}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash x \mapsto v * \mathsf{ls}(v,z)}\,(*)
}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathsf{ls}\mathrm{R}_2)
}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Case}\ \mathsf{ls})
$$

A Cyclic Proof of List Segment Concatenation

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Id})}{\mathsf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\equiv)}{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\dfrac{}{x \mapsto v \vdash x \mapsto v}\,(\mathrm{Id}) \quad \dfrac{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}{\mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(v,z)}\,(\mathrm{Subst})}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash x \mapsto v * \mathsf{ls}(v,z)}\,(*)}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathsf{lsR}_2)}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\text{Case } \mathsf{ls})$$

A Cyclic Proof of List Segment Concatenation

$$\frac{}{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Id})$$

$$\frac{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}{\mathsf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\equiv)$$

$$\frac{\mathsf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(=\mathrm{L})$$

$$\frac{}{x \mapsto v \vdash x \mapsto v}\,(\mathrm{Id})$$

$$\frac{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}{\mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(v,z)}\,(\mathrm{Subst})$$

$$\frac{x \mapsto v \vdash x \mapsto v \qquad \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(v,z)}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash x \mapsto v * \mathsf{ls}(v,z)}\,(*)$$

$$\frac{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash x \mapsto v * \mathsf{ls}(v,z)}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathsf{ls}\mathrm{R}_2)$$

$$\frac{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z) \qquad x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Case}\ \mathsf{ls})$$

A Cyclic Proof of List Segment Concatenation

$$\dfrac{\dfrac{\dfrac{}{\mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Id})}{\mathsf{emp} * \mathsf{ls}(x,z) \vdash \mathsf{ls}(x,z)}\,(\equiv)}{(x = y \wedge \mathsf{emp}) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(=\mathrm{L})$$

$$\dfrac{\dfrac{\dfrac{}{x \mapsto v \vdash x \mapsto v}\,(\mathrm{Id}) \quad \dfrac{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}{\mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(v,z)}\,(\mathrm{Subst})}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash x \mapsto v * \mathsf{ls}(v,z)}\,(*)}{x \mapsto v * \mathsf{ls}(v,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathsf{ls}\mathrm{R}_2)$$

$$\dfrac{\vdots \qquad \vdots}{\mathsf{ls}(x,y) * \mathsf{ls}(y,z) \vdash \mathsf{ls}(x,z)}\,(\mathrm{Case}\ \mathsf{ls})$$

Global Soundness for Cyclic Proof: Elements

- Fix some values that we can trace along paths in the proof
  - In our examples: inductive predicate instances

Global Soundness for Cyclic Proof: Elements

- Fix some values that we can trace along paths in the proof
    - In our examples: inductive predicate instances
- Map (model, trace-value) pairs to elements of a w.f. set

Global Soundness for Cyclic Proof: Elements

- Fix some values that we can trace along paths in the proof
  - In our examples: inductive predicate instances
- Map (model, trace-value) pairs to elements of a w.f. set
  - Inductive definitions induce a monotone operator φ on sets of models

Global Soundness for Cyclic Proof: Elements

- Fix some values that we can trace along paths in the proof
  - In our examples: inductive predicate instances
- Map (model, trace-value) pairs to elements of a w.f. set
  - Inductive definitions induce a monotone operator φ on sets of models
  - Interpret the inductive definitions using the lfp

$$\varphi(\bot) \sqsubseteq \varphi(\varphi(\bot)) \sqsubseteq \ldots \sqsubseteq \varphi^{\omega}(\bot) \sqsubseteq \ldots \sqsubseteq \mu X.\varphi(X)$$

Global Soundness for Cyclic Proof: Elements

- Fix some values that we can trace along paths in the proof
    - In our examples: inductive predicate instances
- Map (model, trace-value) pairs to elements of a w.f. set
    - Inductive definitions induce a monotone operator φ on sets of models
    - Interpret the inductive definitions using the lfp

$$\varphi(\bot) \sqsubseteq \varphi(\varphi(\bot)) \sqsubseteq \ldots \sqsubseteq \varphi^{\omega}(\bot) \sqsubseteq \ldots \sqsubseteq \mu X.\varphi(X)$$

- Map $(m, P\,\vec{t})$ to the least approximation $\varphi^{\alpha}(\bot)$ of P in which m appears

Global Soundness for Cyclic Proof: Elements

- Fix some values that we can trace along paths in the proof
  - In our examples: inductive predicate instances
- Map (model, trace-value) pairs to elements of a w.f. set
  - Inductive definitions induce a monotone operator φ on sets of models
  - Interpret the inductive definitions using the lfp

$$\varphi(\bot) \sqsubseteq \varphi(\varphi(\bot)) \sqsubseteq \ldots \sqsubseteq \varphi^{\omega}(\bot) \sqsubseteq \ldots \sqsubseteq \mu X.\varphi(X)$$

  - Map $(m, P\,\vec{t})$ to the least approximation $\varphi^{\alpha}(\bot)$ of P in which m appears
- Identify the progression points of the proof system, e.g.

$$\frac{x = y \wedge \mathbf{emp} \vdash F \quad x \mapsto v * \mathsf{ls}(v, y) \vdash F}{\mathsf{ls}(x, y) \vdash F} \text{ (Case } \mathsf{ls}\text{)}$$

Global Soundness for Cyclic Proof: General Principle

- Impose global soundness condition on proof graphs:
  - Every infinite path must have an infinitely progressing trace
  - This condition is decidable using Büchi automata

Global Soundness for Cyclic Proof: General Principle

- Impose global soundness condition on proof graphs:
  - Every infinite path must have an infinitely progressing trace
  - This condition is decidable using Büchi automata
- We obtain an infinite descent proof-by-contradiction:

Global Soundness for Cyclic Proof: General Principle

- Impose global soundness condition on proof graphs:
    - Every infinite path must have an infinitely progressing trace
    - This condition is decidable using Büchi automata
- We obtain an infinite descent proof-by-contradiction:
    - Assume the conclusion of the proof is invalid

Global Soundness for Cyclic Proof: General Principle

- Impose global soundness condition on proof graphs:
  - Every infinite path must have an infinitely progressing trace
  - This condition is decidable using Büchi automata
- We obtain an infinite descent proof-by-contradiction:
  - Assume the conclusion of the proof is invalid
  - Local soundness implies an infinite sequence of (counter) models

Global Soundness for Cyclic Proof: General Principle

- Impose global soundness condition on proof graphs:
  - Every infinite path must have an infinitely progressing trace
  - This condition is decidable using Büchi automata
- We obtain an infinite descent proof-by-contradiction:
  - Assume the conclusion of the proof is invalid
  - Local soundness implies an infinite sequence of (counter) models
  - Global soundness then implies an infinite descending chain in a well-founded set

Cyclic Proof vs Explicit Induction

- Explicit induction requires induction hypothesis F up-front

$$\frac{}{N\ 0} \qquad \frac{N\ x}{N\ sx} \qquad \frac{\Gamma \vdash F[0] \quad \Gamma, F[x] \vdash F[sx], \Delta \quad \Gamma, F[t] \vdash \Delta}{\Gamma, N\ t \vdash \Delta}\ (\text{Ind } N)$$

Cyclic Proof vs Explicit Induction

- Explicit induction requires induction hypothesis F up-front

$$\frac{}{N\ 0} \qquad \frac{N\ x}{N\ sx} \qquad \frac{\Gamma \vdash F[0] \quad \Gamma, F[x] \vdash F[sx], \Delta \quad \Gamma, F[t] \vdash \Delta}{\Gamma, N\ t \vdash \Delta}\ (\text{Ind}\ N)$$

- Cyclic proof enables '*discovery*' of induction hypotheses

Cyclic Proof vs Explicit Induction

- Explicit induction requires induction hypothesis F up-front

$$\frac{}{N\ 0} \qquad \frac{N\ x}{N\ sx} \qquad \frac{\Gamma \vdash F[0] \quad \Gamma, F[x] \vdash F[sx], \Delta \quad \Gamma, F[t] \vdash \Delta}{\Gamma, N\ t \vdash \Delta}\ (\text{Ind } N)$$

- Cyclic proof enables '*discovery*' of induction hypotheses
- Complex induction schemes naturally represented by nested and overlapping cycles

Cyclic Proof vs Explicit Induction

- Explicit induction requires induction hypothesis F up-front

$$\frac{}{N\ 0} \qquad \frac{N\ x}{N\ sx} \qquad \frac{\Gamma \vdash F[0] \quad \Gamma, F[x] \vdash F[sx], \Delta \quad \Gamma, F[t] \vdash \Delta}{\Gamma, N\ t \vdash \Delta}\ (\text{Ind}\ N)$$

- Cyclic proof enables '*discovery*' of induction hypotheses
- Complex induction schemes naturally represented by nested and overlapping cycles
- The explicit induction rules are derivable in the cyclic system (cf. Brotherston & Simpson)

Cyclic Proofs vs Infinite Proofs

- Cyclic proofs are the (strict) regular subset of the set of non-well-founded proof trees

Cyclic Proofs vs Infinite Proofs

- Cyclic proofs are the (strict) regular subset of the set of non-well-founded proof trees

- Theorem (Brotherston & Simpson): the full infinite system is cut-free complete

Cyclic Proofs vs Infinite Proofs

- Cyclic proofs are the (strict) regular subset of the set of non-well-founded proof trees
- Theorem (Brotherston & Simpson): the full infinite system is cut-free complete
- Cut is likely not eliminable in the cyclic sub-system

A Simple Imperative Language

$$
\begin{aligned}
&\text{(Terms)} & t &::= \texttt{nil} \mid x \\
&\text{(Boolean Expressions)} & B &::= t\texttt{=}t \mid t\texttt{!=}t \\
&\text{(Programs)} & C &::= \varepsilon && \text{(stop)} \\
& & &\mid\ x\texttt{:=}t\texttt{;}C && \text{(assignment)} \\
& & &\mid\ x\texttt{:=[}y\texttt{];}C \mid \texttt{[}x\texttt{]:=}y\texttt{;}C && \text{(load/store)} \\
& & &\mid\ \texttt{free(}x\texttt{);}C \mid x\texttt{:=new;}C && \text{(de/allocate)} \\
& & &\mid\ \texttt{if}\ B\ \texttt{then}\ C\texttt{;}C && \text{(conditional)} \\
& & &\mid\ \texttt{while}\ B\ \texttt{do}\ C\texttt{;}C && \text{(loop)}
\end{aligned}
$$

A Simple Imperative Language

$$
\begin{array}{rrll}
\text{(Terms)} & t ::= & \texttt{nil} \mid x & \\
\text{(Boolean Expressions)} & B ::= & t\texttt{=}t \mid t\texttt{!=}t & \\
\text{(Programs)} & C ::= & \varepsilon & \text{(stop)} \\
& \mid & x\texttt{:=}t\texttt{;}C & \text{(assignment)} \\
& \mid & x\texttt{:=[}y\texttt{];}C \mid \texttt{[}x\texttt{]:=}y\texttt{;}C & \text{(load/store)} \\
& \mid & \texttt{free(}x\texttt{);}C \mid x\texttt{:=new;}C & \text{(de/allocate)} \\
& \mid & \texttt{if}\ B\ \texttt{then}\ C\texttt{;}C & \text{(conditional)} \\
& \mid & \texttt{while}\ B\ \texttt{do}\ C\texttt{;}C & \text{(loop)}
\end{array}
$$

- The following program deallocates a linked list

```
while x!=nil do y:=[x];free(x);x=y;
```

Program Verification by Symbolic Execution

- We use Hoare logic for proving triples $\{P\}\, C\, \{Q\}$ using Separation Logic as an assertion language

Program Verification by Symbolic Execution

- We use Hoare logic for proving triples $\{P\}\,C\,\{Q\}$ using Separation Logic as an assertion language
- Program commands are executed symbolically by the proof rules, e.g.

$$\text{(load):} \qquad \frac{\{x = v[x'/x] \wedge (P * y \mapsto v)[x'/x]\}\,C\,\{Q\}}{\{P * y \mapsto v\}\,x\texttt{:=[}y\texttt{];}C\,\{Q\}}\,(x' \text{ fresh})$$

Program Verification by Symbolic Execution

- We use Hoare logic for proving triples $\{P\}\, C\, \{Q\}$ using Separation Logic as an assertion language

- Program commands are executed symbolically by the proof rules, e.g.

$$\text{(load):} \quad \frac{\{x = v[x'/x] \wedge (P * y \mapsto v)[x'/x]\}\, C\, \{Q\}}{\{P * y \mapsto v\}\, x\texttt{:=[}y\texttt{];}C\, \{Q\}}\ (x' \text{ fresh})$$

$$\text{(free):} \quad \frac{\{P\}\, C\, \{Q\}}{\{P * x \mapsto v\}\, \texttt{free(}x\texttt{);}C\, \{Q\}}$$

Handling Loops in Cyclic Proofs

- The standard Hoare rule for handling `while` loops:

$$\frac{\{B \wedge P\}\, C_1\, \{P\} \quad \{\neg B \wedge P\}\, C_2\, \{Q\}}{\{P\}\, \texttt{while}\, B\, \texttt{do}\, C_1\, \texttt{;}\, C_2\, \{Q\}}$$

Handling Loops in Cyclic Proofs

- The standard Hoare rule for handling `while` loops:

$$\frac{\{t = z \land B \land P\}\, C_1 \,\{t < z \land P\} \quad \{\neg B \land P\}\, C_2 \,\{Q\}}{\{P\}\, \mathtt{while}\, B\, \mathtt{do}\, C_1 \mathtt{;}\, C_2 \,\{Q\}}$$

  t is the loop variant

Handling Loops in Cyclic Proofs

- The standard Hoare rule for handling `while` loops:

$$\frac{\{t = z \wedge B \wedge P\}\, C_1\, \{t < z \wedge P\} \quad \{\neg B \wedge P\}\, C_2\, \{Q\}}{\{P\}\, \mathtt{while}\, B\, \mathtt{do}\, C_1\mathtt{;}\, C_2\, \{Q\}}$$

  t is the loop variant

- With cyclic proof, it is enough just to unfold loops

$$\frac{\{B \wedge P\}\, C_1\mathtt{;}\mathtt{while}\, B\, \mathtt{do}\, C_1\mathtt{;}\, C_2\, \{Q\} \quad \{\neg B \wedge P\}\, C_2\, \{Q\}}{\{P\}\, \mathtt{while}\, B\, \mathtt{do}\, C_1\mathtt{;}\, C_2\, \{Q\}}$$

Example: Deallocating the Linked List

```
while x!=nil do y:=[x];free(x);x=y;
```

Example: Deallocating the Linked List

$\{\mathsf{ls}(\mathtt{x}, \mathtt{nil})\}$ `while x!=nil do y:=[x];free(x);x=y;`

Example: Deallocating the Linked List

$\{\mathsf{ls}(x, \texttt{nil})\}$ `while x!=nil do y:=[x];free(x);x=y;` $\{\mathsf{emp}\}$

Example: Deallocating the Linked List

$$\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\quad \mathtt{while\,x!=nil\,do\,y:=[x];free(x);x=y;}\quad \{\mathsf{emp}\}$$

$$\dfrac{\left\{\begin{array}{c}\mathtt{x}\neq\mathtt{nil}\\ \wedge\ \mathsf{ls}(\mathtt{x},\mathtt{nil})\end{array}\right\}\ \mathtt{y:=[x];free(x);x=y;while\,x!=nil\,do\,y:=[x];free(x);x=y;}\ \{\mathsf{emp}\} \quad \vdots \quad \left\{\begin{array}{c}\mathtt{x}=\mathtt{nil}\\ \wedge\ \mathsf{ls}(\mathtt{x},\mathtt{nil})\end{array}\right\}\ \varepsilon\ \{\mathsf{emp}\}}{\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\ \mathtt{while}\ \ldots\ \{\mathsf{emp}\}}\ (\mathsf{while})$$

Example: Deallocating the Linked List

$$\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\ \ \mathtt{while\,x\,!=nil\,do\,y:=[x];free(x);x=y;}\ \ \{\mathsf{emp}\}$$

$$\dfrac{\left\{\begin{array}{c}\mathtt{x} \neq \mathtt{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x},\mathtt{nil})\end{array}\right\}\ \mathtt{y:=[x];}\ \ldots\ \{\mathsf{emp}\} \quad \vdots \qquad \left\{\begin{array}{c}\mathtt{x} = \mathtt{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x},\mathtt{nil})\end{array}\right\}\ \varepsilon\ \{\mathsf{emp}\}}{\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\ \mathtt{while}\ \ldots\ \{\mathsf{emp}\}}\ (\mathsf{while})$$

Example: Deallocating the Linked List

$$\{\mathsf{ls}(x, \texttt{nil})\}\ \ \texttt{while x!=nil do y:=[x];free(x);x=y;}\ \ \{\mathsf{emp}\}$$

$$\dfrac{\left\{\begin{array}{c} x \neq \texttt{nil} \\ \wedge\ \mathsf{ls}(x, \texttt{nil}) \end{array}\right\} \texttt{y:=[x];} \ldots \{\texttt{emp}\} \qquad \dfrac{}{\left\{\begin{array}{c} x = \texttt{nil} \\ \wedge\ \mathsf{ls}(x, \texttt{nil}) \end{array}\right\} \varepsilon\ \{\texttt{emp}\}}(\models)}{\{\mathsf{ls}(x, \texttt{nil})\}\ \texttt{while}\ \ldots\ \{\texttt{emp}\}}(\text{while})$$

Example: Deallocating the Linked List

$$\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\quad \mathtt{while\,x!=nil\,do\,y:=[x];free(x);x=y;}\quad \{\mathsf{emp}\}$$

$$\dfrac{\dfrac{\left\{\begin{matrix} \mathtt{x}\neq\mathtt{nil} \\ \wedge\ \mathtt{x}=\mathtt{nil} \\ \wedge\ \mathsf{emp}\end{matrix}\right\}\ \mathtt{y:=[x];}\ \ldots\ \{\mathsf{emp}\} \qquad \left\{\begin{matrix} \mathtt{x}\mapsto v \\ *\ \mathsf{ls}(v,\mathtt{nil})\end{matrix}\right\}\ \mathtt{y:=[x];}\ \ldots\ \{\mathsf{emp}\}}{\left\{\begin{matrix} \mathtt{x}\neq\mathtt{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x},\mathtt{nil})\end{matrix}\right\}\ \mathtt{y:=[x];}\ \ldots\ \{\mathsf{emp}\}}\ (\text{unfold } \mathsf{ls}) \qquad \dfrac{}{\left\{\begin{matrix} \mathtt{x}=\mathtt{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x},\mathtt{nil})\end{matrix}\right\}\ \varepsilon\ \{\mathsf{emp}\}}\ (\models)}{\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\ \mathtt{while}\ \ldots\ \{\mathsf{emp}\}}\ (\text{while})$$

Example: Deallocating the Linked List

$$\{\mathsf{ls}(\texttt{x},\texttt{nil})\}\quad \texttt{while}\,\texttt{x!=nil}\,\texttt{do}\,\texttt{y:=[x];free(x);x=y;}\quad \{\texttt{emp}\}$$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\left\{\begin{array}{c} \texttt{x} \neq \texttt{nil} \\ \wedge\ \texttt{x} = \texttt{nil} \\ \wedge\ \texttt{emp} \end{array}\right\} \texttt{y:=[x];}\ \ldots\ \{\texttt{emp}\}}\,(\bot)
\qquad
\left\{\begin{array}{c} \texttt{x} \mapsto v \\ *\ \mathsf{ls}(v,\texttt{nil}) \end{array}\right\} \texttt{y:=[x];}\ \ldots\ \{\texttt{emp}\}
}{
\left\{\begin{array}{c} \texttt{x} \neq \texttt{nil} \\ \wedge\ \mathsf{ls}(\texttt{x},\texttt{nil}) \end{array}\right\} \texttt{y:=[x];}\ \ldots\ \{\texttt{emp}\}
}\,(\text{unfold } \mathsf{ls})
}{\vdots}
\qquad
\dfrac{}{\left\{\begin{array}{c} \texttt{x} = \texttt{nil} \\ \wedge\ \mathsf{ls}(\texttt{x},\texttt{nil}) \end{array}\right\} \varepsilon\ \{\texttt{emp}\}}\,(\models)
}{
\{\mathsf{ls}(\texttt{x},\texttt{nil})\}\ \texttt{while}\ \ldots\ \{\texttt{emp}\}
}\,(\text{while})
$$

Example: Deallocating the Linked List

$$\{\mathsf{ls}(\mathtt{x}, \mathtt{nil})\} \quad \mathtt{while\ x!=nil\ do\ y:=[x];free(x);x=y;} \quad \{\mathsf{emp}\}$$

$$
\dfrac{
  \dfrac{
    \dfrac{}{\left\{\begin{array}{c} \mathtt{x} \neq \mathtt{nil} \\ \wedge\ \mathtt{x} = \mathtt{nil} \\ \wedge\ \mathsf{emp} \end{array}\right\} \mathtt{y:=[x];} \ldots \{\mathsf{emp}\}}\ (\bot)
    \qquad
    \dfrac{\left\{\begin{array}{c} \mathtt{x} \mapsto y \\ *\ \mathsf{ls}(y, \mathtt{nil}) \end{array}\right\} \mathtt{free(x);} \ldots \{\mathsf{emp}\}}{\left\{\begin{array}{c} \mathtt{x} \mapsto v \\ *\ \mathsf{ls}(v, \mathtt{nil}) \end{array}\right\} \mathtt{y:=[x];} \ldots \{\mathsf{emp}\}}\ (\mathsf{load})
  }{
    \left\{\begin{array}{c} \mathtt{x} \neq \mathtt{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x}, \mathtt{nil}) \end{array}\right\} \mathtt{y:=[x];} \ldots \{\mathsf{emp}\}
  }\ (\text{unfold } \mathsf{ls})
  \quad \vdots
  \qquad
  \dfrac{}{\left\{\begin{array}{c} \mathtt{x} = \mathtt{nil} \\ \wedge\ \mathsf{ls}(\mathtt{x}, \mathtt{nil}) \end{array}\right\} \varepsilon\ \{\mathsf{emp}\}}\ (\models)
}{
  \{\mathsf{ls}(\mathtt{x}, \mathtt{nil})\}\ \mathtt{while} \ldots \{\mathsf{emp}\}
}\ (\text{while})
$$

Example: Deallocating the Linked List

$$\{\mathsf{ls}(\texttt{x},\texttt{nil})\}\quad \texttt{while}\,\texttt{x!=nil}\,\texttt{do}\,\texttt{y:=[x];free(x);x=y;}\quad \{\mathsf{emp}\}$$

$$
\dfrac{
\dfrac{
\dfrac{}{\left\{\begin{array}{c}\texttt{x} \neq \texttt{nil}\\ \wedge\ \texttt{x} = \texttt{nil}\\ \wedge\ \mathsf{emp}\end{array}\right\} \texttt{y:=[x];}\ \ldots\ \{\mathsf{emp}\}}\,(\bot)
\qquad
\dfrac{\dfrac{\{\mathsf{ls}(\texttt{y},\texttt{nil})\}\ \texttt{x=y;}\ \ldots\ \{\mathsf{emp}\}}{\left\{\begin{array}{c}\color{orange}{\texttt{x} \mapsto \texttt{y}}\\ *\ \mathsf{ls}(\texttt{y},\texttt{nil})\end{array}\right\} \texttt{free(x);}\ \ldots\ \{\mathsf{emp}\}}\,(\color{orange}{\text{free}})}{\left\{\begin{array}{c}\texttt{x} \mapsto v\\ *\ \mathsf{ls}(v,\texttt{nil})\end{array}\right\} \texttt{y:=[x];}\ \ldots\ \{\mathsf{emp}\}}\,(\text{load})
}{
\begin{array}{c}\left\{\begin{array}{c}\texttt{x} \neq \texttt{nil}\\ \wedge\ \mathsf{ls}(\texttt{x},\texttt{nil})\end{array}\right\} \texttt{y:=[x];}\ \ldots\ \{\mathsf{emp}\}\\ \vdots\end{array}
}\,(\text{unfold } \mathsf{ls})
\qquad
\dfrac{}{\left\{\begin{array}{c}\texttt{x} = \texttt{nil}\\ \wedge\ \mathsf{ls}(\texttt{x},\texttt{nil})\end{array}\right\} \varepsilon\ \{\mathsf{emp}\}}\,(\models)
}{
\{\mathsf{ls}(\texttt{x},\texttt{nil})\}\ \texttt{while}\ \ldots\ \{\mathsf{emp}\}
}\,(\text{while})
$$

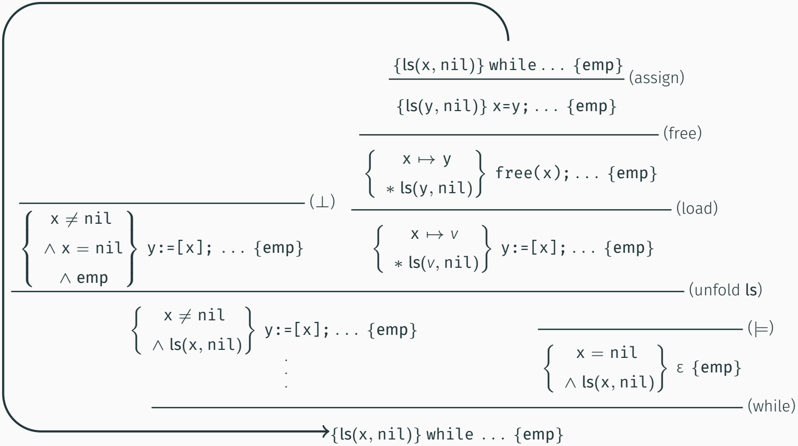
Example: Deallocating the Linked List

$$\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\quad \mathtt{while\,x!=nil\,do\,y:=[x];free(x);x=y;}\quad \{\mathtt{emp}\}$$

$$\dfrac{\dfrac{\dfrac{}{\left\{\begin{array}{c}\mathtt{x}\neq\mathtt{nil}\\ \wedge\ \mathtt{x}=\mathtt{nil}\\ \wedge\ \mathtt{emp}\end{array}\right\}\ \mathtt{y:=[x];}\ \ldots\ \{\mathtt{emp}\}}(\bot)\qquad \dfrac{\dfrac{\dfrac{\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\ \mathtt{while}\ \ldots\ \{\mathtt{emp}\}}{\{\mathsf{ls}(\mathtt{y},\mathtt{nil})\}\ \mathtt{x=y;}\ \ldots\ \{\mathtt{emp}\}}(\text{assign})}{\left\{\begin{array}{c}\mathtt{x}\mapsto\mathtt{y}\\ *\ \mathsf{ls}(\mathtt{y},\mathtt{nil})\end{array}\right\}\ \mathtt{free(x);}\ \ldots\ \{\mathtt{emp}\}}(\text{free})}{\left\{\begin{array}{c}\mathtt{x}\mapsto v\\ *\ \mathsf{ls}(v,\mathtt{nil})\end{array}\right\}\ \mathtt{y:=[x];}\ \ldots\ \{\mathtt{emp}\}}(\text{load})}{\begin{array}{c}\left\{\begin{array}{c}\mathtt{x}\neq\mathtt{nil}\\ \wedge\ \mathsf{ls}(\mathtt{x},\mathtt{nil})\end{array}\right\}\ \mathtt{y:=[x];}\ \ldots\ \{\mathtt{emp}\}\\ \vdots\end{array}}(\text{unfold }\mathsf{ls})\qquad \dfrac{}{\left\{\begin{array}{c}\mathtt{x}=\mathtt{nil}\\ \wedge\ \mathsf{ls}(\mathtt{x},\mathtt{nil})\end{array}\right\}\ \varepsilon\ \{\mathtt{emp}\}}(\models)}{\{\mathsf{ls}(\mathtt{x},\mathtt{nil})\}\ \mathtt{while}\ \ldots\ \{\mathtt{emp}\}}(\text{while})$$

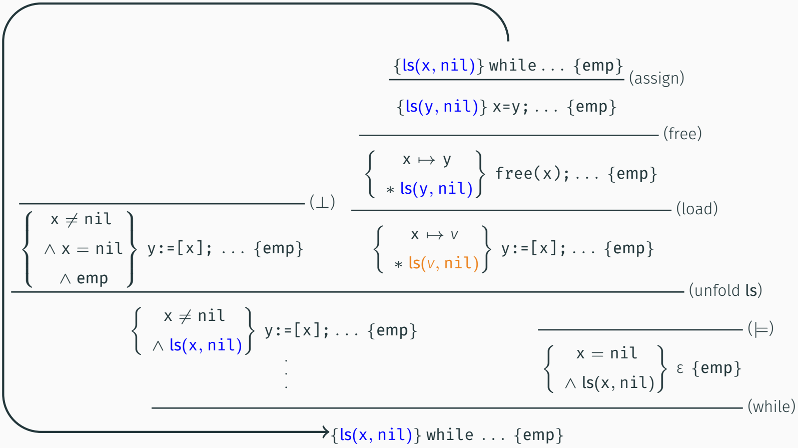
Example: Deallocating the Linked List

{ls(x, nil)} while x!=nil do y:=[x];free(x);x=y; {emp}



Example: Deallocating the Linked List

$\{\mathsf{ls}(\mathtt{x}, \mathtt{nil})\}$ `while x!=nil do y:=[x];free(x);x=y;` $\{\mathsf{emp}\}$



(Procedures)	$\texttt{proc}\, p(\vec{x})\, \{\, C \,\}$
(Programs)	$C ::= \ldots \mid p(\vec{t})\texttt{;}\, C$

Verifying Recursive Procedures

$$\begin{array}{rl} \text{(Procedures)} & \texttt{proc}\, p(\vec{x})\, \{\, C \,\} \\ \text{(Programs)} & C \ ::= \ \ldots \ \mid \ p(\vec{t})\,\texttt{;}\, C \end{array}$$

$$\text{(proc):} \quad \frac{\{P\}\, C\, \{Q\}}{\{P\}\, p(\vec{x})\, \{Q\}}\, (\text{body}(p) = C)$$

Verifying Recursive Procedures

$$\text{(Procedures)} \qquad \texttt{proc}\, p(\vec{x})\,\{\, C \,\}$$

$$\text{(Programs)} \qquad C \ ::= \ \ldots \ | \ p(\vec{t})\,\texttt{;}\, C$$

$$\text{(proc):} \quad \frac{\{P\}\, C\, \{Q\}}{\{P\}\, p(\vec{x})\, \{Q\}}\,(\text{body}(p) = C) \qquad \text{(call):} \quad \frac{\{P\}\, p(\vec{t})\, \{P'\} \quad \{P'\}\, C\, \{Q\}}{\{P\}\, p(\vec{t})\,\texttt{;}\, C\, \{Q\}}$$

Verifying Recursive Procedures

$$\text{(Procedures)} \qquad \mathtt{proc}\, p(\vec{x})\ \{\, C \,\}$$
$$\text{(Programs)} \qquad C \ ::= \ \ldots \mid p(\vec{t})\,;\, C$$

$$\text{(proc):} \quad \frac{\{P\}\, C\, \{Q\}}{\{P\}\, p(\vec{x})\, \{Q\}}\, (\text{body}(p) = C) \qquad \text{(call):} \quad \frac{\{P\}\, p(\vec{t})\, \{P'\} \quad \{P'\}\, C\, \{Q\}}{\{P\}\, p(\vec{t})\,;\, C\, \{Q\}}$$

$$\text{(param):} \quad \frac{\{P\}\, p(\vec{t})\, \{Q\}}{\{P[t/x]\}\, p(\vec{t})[t/x]\, \{Q[t/x]\}}$$

Verifying Recursive Procedures

$$\text{(Procedures)} \qquad \texttt{proc}\, p(\vec{x})\, \{\, C\, \}$$

$$\text{(Programs)} \qquad C \ ::= \ \ldots \mid p(\vec{t});\, C$$

$$\text{(proc): } \frac{\{P\}\, C\, \{Q\}}{\{P\}\, p(\vec{x})\, \{Q\}} (\text{body}(p) = C) \qquad \text{(call): } \frac{\{P\}\, p(\vec{t})\, \{P'\} \quad \{P'\}\, C\, \{Q\}}{\{P\}\, p(\vec{t});\, C\, \{Q\}}$$

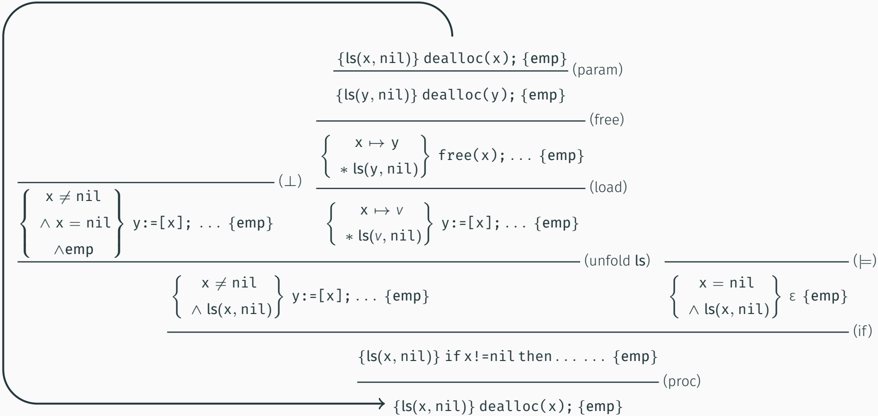
$$\text{(param): } \frac{\{P\}\, p(\vec{t})\, \{Q\}}{\{P[t/x]\}\, p(\vec{t})[t/x]\, \{Q[t/x]\}}$$

- The following procedure recursively deallocates a linked list

```
proc dealloc(x) { if x!=nil then y:=[x]; free(x); dealloc(y); }
```

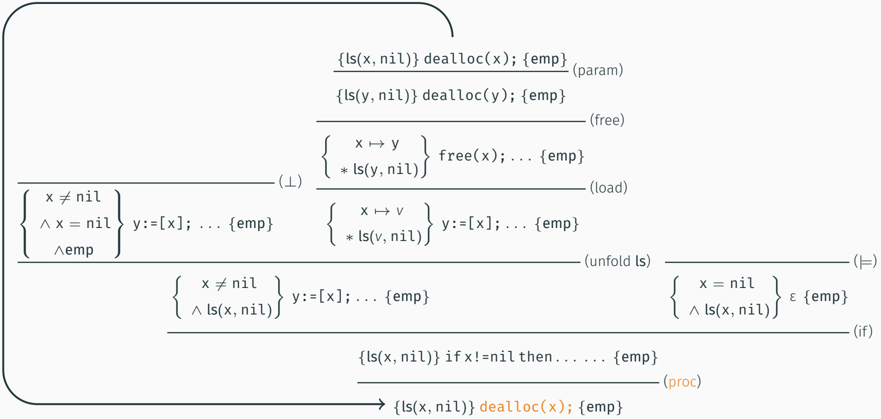
Example: Deallocating the Linked List (Recursively)

```
proc dealloc(x) { if x!=nil then y:=[x]; free(x); dealloc(y); }
```



Example: Deallocating the Linked List (Recursively)

```
proc dealloc(x) { if x!=nil then y:=[x]; free(x); dealloc(y); }
```



Example: Deallocating the Linked List (Recursively)

`proc dealloc(x) { if x!=nil then y:=[x]; free(x); dealloc(y); }`

{ls(x, nil)} dealloc(x); {emp} (param)

{ls(y, nil)} dealloc(y); {emp}

(free)

$\left\{ \begin{array}{c} \texttt{x} \mapsto \texttt{y} \\ * \ \mathsf{ls}(\texttt{y}, \texttt{nil}) \end{array} \right\}$ free(x); ... {emp}

(load)

$\left\{ \begin{array}{c} \texttt{x} \mapsto v \\ * \ \mathsf{ls}(v, \texttt{nil}) \end{array} \right\}$ y:=[x]; ... {emp}

(⊥)

$\left\{ \begin{array}{c} \texttt{x} \neq \texttt{nil} \\ \wedge \ \texttt{x} = \texttt{nil} \\ \wedge \mathsf{emp} \end{array} \right\}$ y:=[x]; ... {emp}

(unfold ls)

$\left\{ \begin{array}{c} \texttt{x} \neq \texttt{nil} \\ \wedge \ \mathsf{ls}(\texttt{x}, \texttt{nil}) \end{array} \right\}$ y:=[x]; ... {emp}

(⊨)

$\left\{ \begin{array}{c} \texttt{x} = \texttt{nil} \\ \wedge \ \mathsf{ls}(\texttt{x}, \texttt{nil}) \end{array} \right\}$ ε {emp}

(if)

{ls(x, nil)} if x!=nil then ... ... {emp}

(proc)

{ls(x, nil)} dealloc(x); {emp}

Example: Deallocating the Linked List (Recursively)

proc dealloc(x) { if x!=nil then y:=[x]; free(x); dealloc(y); }

{ls(x, nil)} dealloc(x); {emp} (param)

{ls(y, nil)} dealloc(y); {emp}

(free)

$\left\{\begin{array}{c} x \mapsto y \\ * \ \mathsf{ls}(y, \mathtt{nil}) \end{array}\right\}$ free(x); . . . {emp}

(⊥) (load)

$\left\{\begin{array}{c} x \neq \mathtt{nil} \\ \wedge\ x = \mathtt{nil} \\ \wedge \mathsf{emp} \end{array}\right\}$ y:=[x]; . . . {emp}

$\left\{\begin{array}{c} x \mapsto v \\ * \ \mathsf{ls}(v, \mathtt{nil}) \end{array}\right\}$ y:=[x]; . . . {emp}

(unfold ls) (⊨)

$\left\{\begin{array}{c} x \neq \mathtt{nil} \\ \wedge\ \mathsf{ls}(x, \mathtt{nil}) \end{array}\right\}$ y:=[x]; . . . {emp}

$\left\{\begin{array}{c} x = \mathtt{nil} \\ \wedge\ \mathsf{ls}(x, \mathtt{nil}) \end{array}\right\}$ ε {emp}

(if)

{ls(x, nil)} if x!=nil then . . . . . . {emp}

(proc)

{ls(x, nil)} dealloc(x); {emp}

A Subtlety with Procedures

```
proc shuffle(x){
  if x!=nil then
    y:=[x]; reverse(y); shuffle(y); [x]:=y;
}
```

A Subtlety with Procedures

```
proc shuffle(x){
  if x!=nil then
    y:=[x]; reverse(y); shuffle(y); [x]:=y;
}
```

$$\dfrac{\dfrac{\dfrac{\vdots}{\{\mathsf{ls}(y,\mathtt{nil})\}\ \mathtt{reverse(y);}\ \{\mathsf{ls}(y,\mathtt{nil})\}}}{\{x \mapsto y * \mathsf{ls}(y,\mathtt{nil})\}\ \mathtt{reverse(y);}\ \{x \mapsto y * \mathsf{ls}(y,\mathtt{nil})\}}\ (\text{frame}) \qquad \dfrac{\vdots}{\{x \mapsto y * \mathsf{ls}(y,\mathtt{nil})\}\ \mathtt{shuffle(y);}\ \ldots\ \{\mathsf{ls}(x,\mathtt{nil})\}}}{\dfrac{\{x \mapsto y * \mathsf{ls}(y,\mathtt{nil})\}\ \mathtt{reverse(y);shuffle(y);}\ \ldots\ \{\mathsf{ls}(x,\mathtt{nil})\}}{\vdots}}$$

A Subtlety with Procedures

```
proc shuffle(x) {
  if x!=nil then
    y:=[x]; reverse(y); shuffle(y); [x]:=y;
}
```

$$\dfrac{\dfrac{\vdots}{\{\mathsf{ls}(y,\texttt{nil})\}\ \texttt{reverse(y);}\ \{\mathsf{ls}(y,\texttt{nil})\}}\ \text{(frame)}}{\{x \mapsto y * \mathsf{ls}(y,\texttt{nil})\}\ \texttt{reverse(y);}\ \{x \mapsto y * \mathsf{ls}(y,\texttt{nil})\}}$$

$$\dfrac{\vdots}{\{x \mapsto y * \mathsf{ls}(y,\texttt{nil})\}\ \texttt{shuffle(y);}\ \ldots\ \{\mathsf{ls}(x,\texttt{nil})\}}$$

$$\dfrac{\{x \mapsto y * \mathsf{ls}(y,\texttt{nil})\}\ \texttt{reverse(y);}\ \{x \mapsto y * \mathsf{ls}(y,\texttt{nil})\} \qquad \{x \mapsto y * \mathsf{ls}(y,\texttt{nil})\}\ \texttt{shuffle(y);}\ \ldots\ \{\mathsf{ls}(x,\texttt{nil})\}}{\{x \mapsto y * \mathsf{ls}(y,\texttt{nil})\}\ \texttt{reverse(y); shuffle(y);}\ \ldots\ \{\mathsf{ls}(x,\texttt{nil})\}}$$

$$\vdots$$

A Subtlety with Procedures

```
proc shuffle(x){
  if x!=nil then
    y:=[x]; reverse(y); shuffle(y); [x]:=y;
}
```

⋮

{ls(y, nil)} reverse(y); {ls(y, nil)}

——————————————————— (frame)

{x ↦ y ∗ ls(y, nil)} reverse(y); {x ↦ y ∗ ls(y, nil)}   ⋮ {x ↦ y ∗ ls(y, nil)} shuffle(y); ... {ls(x, nil)}

———————————————————————————————————

{x ↦ y ∗ ls(y, nil)} reverse(y); shuffle(y); ... {ls(x, nil)}

⋮

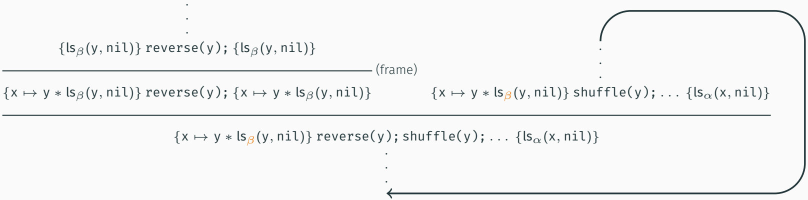
Solution: Explicit Approximation

- We explicitly label predicate instances, e.g. $\mathsf{ls}_{\alpha}(x, y)$
    - indicates which approximation to interpret them in

Solution: Explicit Approximation

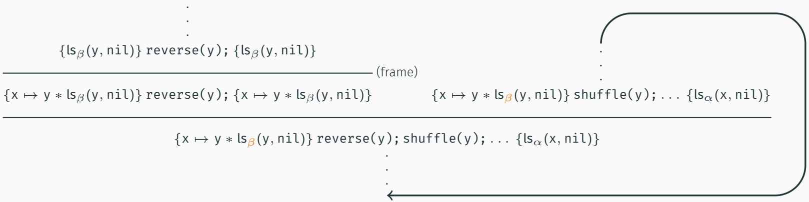
- We explicitly label predicate instances, e.g. $\mathsf{ls}_{\alpha}(x, y)$
  - indicates which approximation to interpret them in

- We now use these labels as the trace values, e.g.



Solution: Explicit Approximation

- We explicitly label predicate instances, e.g. $\mathsf{ls}_{\alpha}(x, y)$
  - indicates which approximation to interpret them in
- We now use these labels as the trace values, e.g.



- We now need constraints on labels when unfolding, e.g.

$$\frac{\Gamma, \beta < \alpha, t = 0 \vdash \Delta \quad \Gamma, \beta < \alpha, t = sx, N_{\beta}\ x \vdash \Delta}{\Gamma, N_{\alpha}\ t \vdash \Delta}\ (\text{Case } N)$$

The Cyclist Verification Tool

- Our verification tool, Cyclist, is implemented in OCaml
- Generic cyclic proof-search procedure using iterated depth-first search
    - Cycles are formed eagerly and discarded if unsound
- The generic proof search is parametric
    - Different proof systems implemented as separate modules

The Cyclist Verification Tool

- Our verification tool, Cyclist, is implemented in OCaml
- Generic cyclic proof-search procedure using iterated depth-first search
    - Cycles are formed eagerly and discarded if unsound
- The generic proof search is parametric
    - Different proof systems implemented as separate modules

github.com/ngorogiannis/cyclist

Performance Results

Program	Time (ms)	LOC	Procs	Nodes	Back-links
list traverse	17	6	1	18	2
tree traverse	24	7	1	26	2
list deallocate	14	7	1	13	1
tree deallocate	21	8	1	24	2
tree reflect	20	9	1	22	2
list rev. deallocate	43	18	1	49	2
list append	28	21	1	34	1
list reverse	122	14	1	34	1
list reverse (tail rec.)	31	18	1	32	1
list reverse (with append)	47	28	2	56	2
list filter	27	16	1	29	1
list partition	31	25	1	40	1
list ackermann	126	17	1	50	3
queue	894	30	3	119	6
functional queue	254	28	3	62	1
shuffle	202	23	2	79	4

Results of Experimental Evaluation on 2.93GHz Intel Core i7-870, 8GB RAM

Concluding Remarks

- Ongoing work: inferring constraints on predicate labels automatically
- Some problems remain hard, of course
  - Generalisation of inductive hypotheses
  - Finding and applying lemmas
  - Synthesizing procedure summaries (see previous point!)

Thank You

Related Work

- Cyclic proofs for FOL with inductive predicates
  (Brotherston & Simpson, LICS 2007)
- Cyclic proofs for Separation Logic with inductive predicates
  (Brotherston, SAS 2007)
- Cyclic proofs verifying simple heap-manipulating **WHILE** language
  (Brotherston, Bornat & Calcagno, POPL 2008)
- Implementations in Isabelle/HOL, then OCaml
  (Brotherston, Distefano, Gorogiannis, CADE 2011/APLAS 2012)
- Abduction of inductive predicates using cyclic proof
  (Brotherston & Gorogiannis, SAS 2014)
- Current Work — cyclic proofs for verifying:
    - procedural heap-manipulating language (Rowe, Brotherston)
    - temporal properties (Tellez Espinosa, Brotherston)