Automatic Cyclic Termination Proofs for Recursive Procedures in Separation Logic

Reuben Rowe, University of Kent, Canterbury
James Brotherston, University College London

CPP, Paris, France
Monday 16th January 2017

Automatically Proving Termination: Challenges

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

Automatically Proving Termination: Challenges

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

recursion

Automatically Proving Termination: Challenges

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

intermediate procedures

recursion

Automatically Proving Termination: Challenges

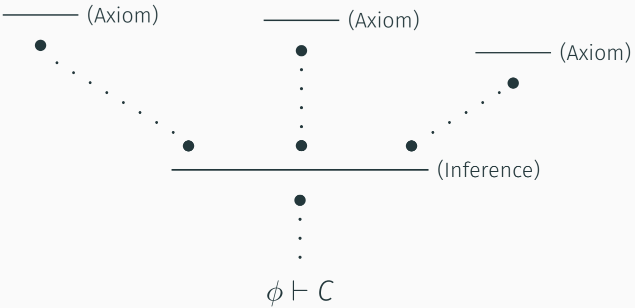
```
proc shuffle(x) {
    if x != nil {
        y := *x;
        reverse(y);
        shuffle(y);
    }
}
```

heap manipulation

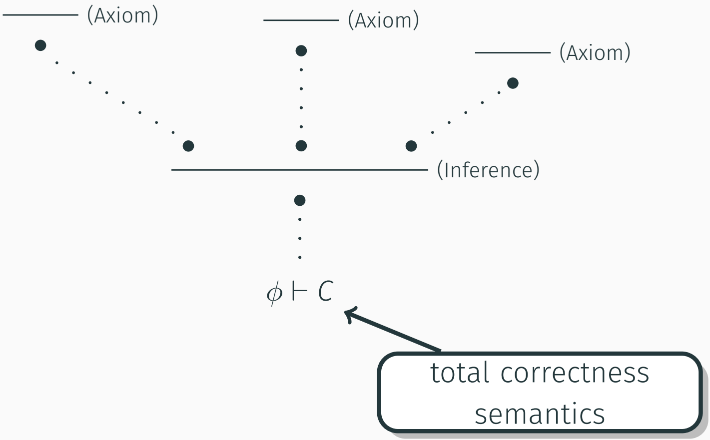
intermediate procedures

recursion

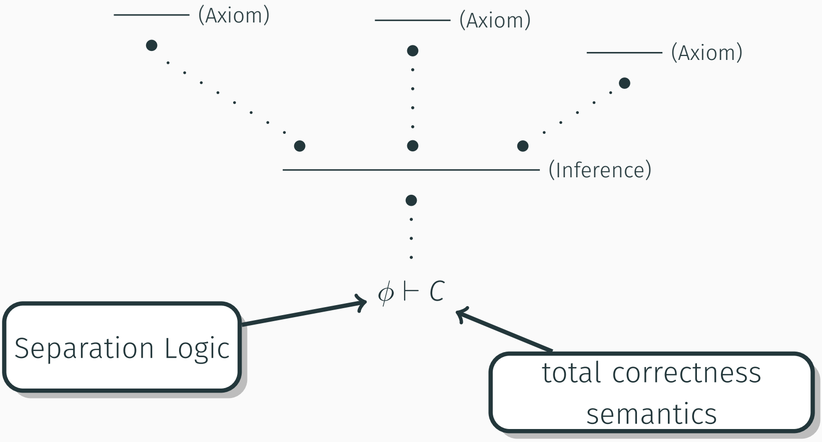
- Following the approach of Brotherston et al. (POPL ’08)



- Following the approach of Brotherston et al. (POPL ’08)

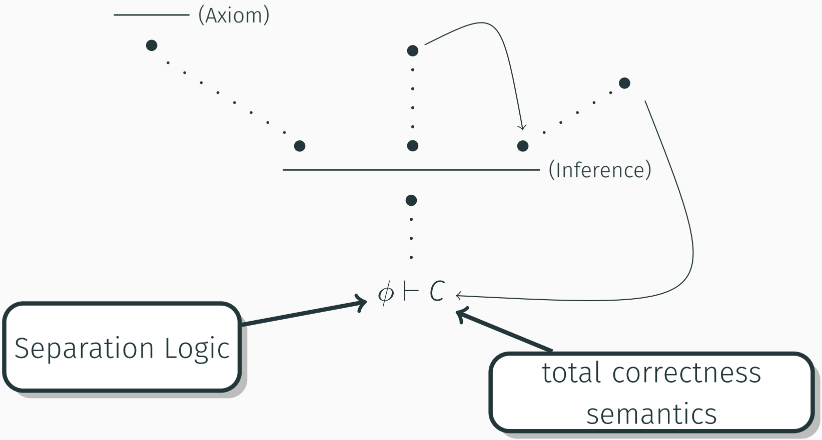


- Following the approach of Brotherston et al. (POPL '08)



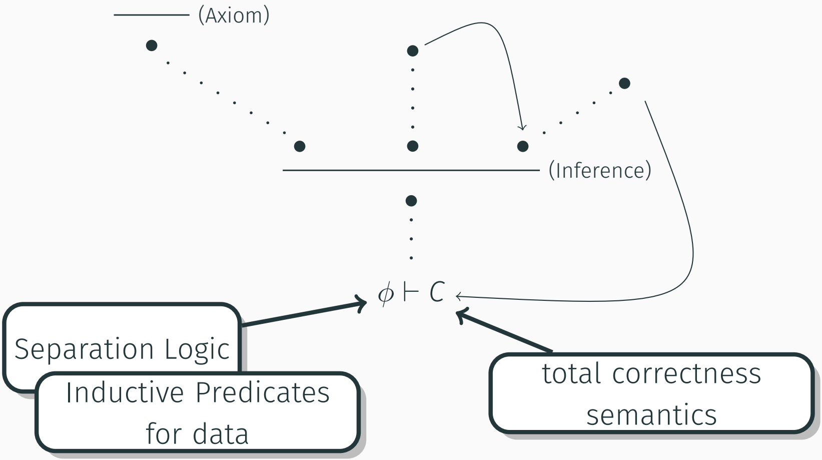
Automatically Proving Termination using Cyclic Proof

- Following the approach of Brotherston et al. (POPL ’08)



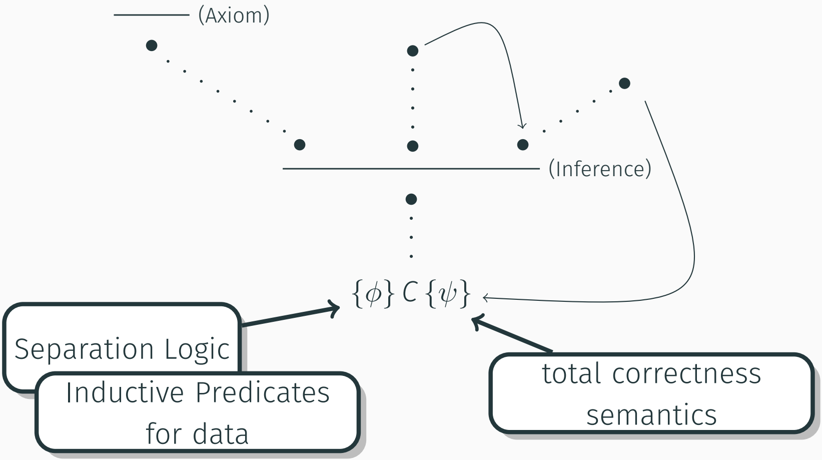
Automatically Proving Termination using Cyclic Proof

- Following the approach of Brotherston et al. (POPL ’08)



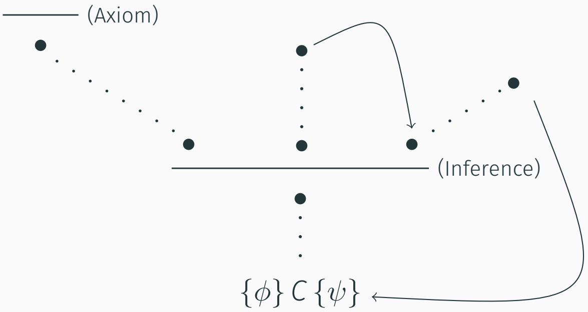
Automatically Proving Termination using Cyclic Proof

- Following the approach of Brotherston et al. (POPL ’08)



Automatically Proving Termination using Cyclic Proof

- Following the approach of Brotherston et al. (POPL '08)



- We use the Cyclist framework for automation/certification

Advantages of Using Cyclic Proof

- Supports *compositional* reasoning

Advantages of Using Cyclic Proof

- Supports *compositional* reasoning
- Naturally encapsulates inductive principles

Advantages of Using Cyclic Proof

- Supports *compositional* reasoning
- Naturally encapsulates inductive principles
- Invariants can be discovered

Advantages of Using Cyclic Proof

- Supports *compositional* reasoning
- Naturally encapsulates inductive principles
- Invariants can be discovered
- Termination measures extracted *automatically*

Ingredients of Our Approach: Separation Logic

- Formulas of SL describe portions of the program memory

Ingredients of Our Approach: Separation Logic

- Formulas of SL describe portions of the program memory
  - **emp** is the empty piece of memory

Ingredients of Our Approach: Separation Logic

- Formulas of SL describe portions of the program memory
  - **emp** is the empty piece of memory
  - $x \mapsto (y_1, \ldots, y_n)$ is a single memory cell referenced by x

Ingredients of Our Approach: Separation Logic

- Formulas of SL describe portions of the program memory
  - **emp** is the empty piece of memory
  - $x \mapsto (y_1, \ldots, y_n)$ is a single memory cell referenced by x
  - $A * B$ is the (separate) conjunction of two domain-disjoint pieces of memory

Ingredients of Our Approach: Separation Logic

- Formulas of SL describe portions of the program memory
  - **emp** is the empty piece of memory
  - $x \mapsto (y_1, \ldots, y_n)$ is a single memory cell referenced by x
  - $A * B$ is the (separate) conjunction of two domain-disjoint pieces of memory

- *Predicates* $P(x_1, \ldots, x_n)$ describe specific structures

  e.g. $\mathsf{lseg}(x, y)$, $\mathsf{list}(z)$

Ingredients of Our Approach: Separation Logic

- Formulas of SL describe portions of the program memory
  - **emp** is the empty piece of memory
  - $x \mapsto (y_1, \ldots, y_n)$ is a single memory cell referenced by x
  - $A * B$ is the (separate) conjunction of two domain-disjoint pieces of memory
- *Predicates* $P(x_1, \ldots, x_n)$ describe specific structures

  e.g. $\mathbf{lseg}(x, y)$, $\mathbf{list}(z)$
- *Symbolic heap* syntax makes reasoning easier

$$x = y \land z \neq \text{nil} \land \mathbf{lseg}(x, y) * \mathbf{list}(z) * v \mapsto w$$

Ingredients of our Approach: Symbolic Execution

$$(\text{free}):\quad \frac{\{\phi\}\, C\, \{\psi\}}{\{\phi * x \mapsto y\}\, \texttt{free}(x);\, C\, \{\psi\}}$$

Ingredients of our Approach: Symbolic Execution

$$\text{(free)}:\quad \frac{\{\phi\}\, C\, \{\psi\}}{\{\phi * x \mapsto y\}\, \texttt{free(}x\texttt{);}\, C\, \{\psi\}}$$

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

Ingredients of our Approach: Symbolic Execution

$$(\text{free}):\quad \frac{\{\phi\}\, C\, \{\psi\}}{\{\phi * x \mapsto y\}\ \texttt{free(}x\texttt{);}\ C\, \{\psi\}}$$

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

$$(\text{proc}):\quad \frac{\{\phi\}\, C\, \{\psi\}}{\{\phi\}\ \text{proc}(\vec{x})\ \{\psi\}}\ (\text{body}(\text{proc}) = C)$$

Ingredients of our Approach: Inductive Predicates

- We support user-defined inductive predicates, e.g.

$$\frac{x = \mathrm{nil} \land \mathbf{emp}}{\mathsf{list}(x)} \qquad \frac{x \mapsto y * \mathsf{list}(y)}{\mathsf{list}(x)}$$

Ingredients of our Approach: Inductive Predicates

- We support user-defined inductive predicates, e.g.

$$\frac{x = \mathrm{nil} \land \mathbf{emp}}{\mathsf{list}(x)} \qquad \frac{x \mapsto y * \mathsf{list}(y)}{\mathsf{list}(x)}$$

- Predicate labels identify termination measures, e.g.

$$\{\mathsf{list}_\alpha(x) * \phi\}\, C\, \{\psi\}$$

Ingredients of our Approach: Inductive Predicates

- We support user-defined inductive predicates, e.g.

$$\frac{x = \mathrm{nil} \wedge \mathbf{emp}}{\mathsf{list}(x)} \qquad \frac{x \mapsto y * \mathsf{list}(y)}{\mathsf{list}(x)}$$

- Predicate labels identify termination measures, e.g.

$$\{\mathsf{list}_\alpha(x) * \phi\}\, C\, \{\psi\}$$

- A logical rule schema allows case split

$$\frac{\{(x = \mathrm{nil} \wedge \mathbf{emp}) * \phi\}\, C\, \{\psi\} \quad \{(\beta < \alpha \wedge x \mapsto y * \mathsf{list}_\beta(x)) * \phi\}\, C\, \{\psi\}}{\{\mathsf{list}_\alpha(x) * \phi\}\, C\, \{\psi\}}$$

Ingredients of our Approach: Inductive Predicates

- We support user-defined inductive predicates, e.g.

$$\frac{x = \mathrm{nil} \wedge \mathbf{emp}}{\mathbf{list}(x)} \qquad \frac{x \mapsto y * \mathbf{list}(y)}{\mathbf{list}(x)}$$

- Predicate labels identify termination measures, e.g.

$$\{\mathbf{list}_{\alpha}(x) * \phi\}\, C\, \{\psi\}$$

- A logical rule schema allows case split

$$\frac{\{(x = \mathrm{nil} \wedge \mathbf{emp}) * \phi\}\, C\, \{\psi\} \quad \{(\beta < \alpha \wedge x \mapsto y * \mathbf{list}_{\beta}(x)) * \phi\}\, C\, \{\psi\}}{\{\mathbf{list}_{\alpha}(x) * \phi\}\, C\, \{\psi\}}$$

Implementation

- We achieve automation by implementing the proof system in Cyclist

Implementation

- We achieve automation by implementing the proof system in Cyclist
    - A generic framework for cyclic proof search

Implementation

- We achieve automation by implementing the proof system in Cyclist
    - A generic framework for cyclic proof search
    - Proof objects can be extracted for certification

Implementation

- We achieve automation by implementing the proof system in CYCLIST
  - A generic framework for cyclic proof search
  - Proof objects can be extracted for certification
- Entailment queries also handled by CYCLIST

Implementation

- We achieve automation by implementing the proof system in Cyclist
    - A generic framework for cyclic proof search
    - Proof objects can be extracted for certification
- Entailment queries also handled by Cyclist
- Procedure calls and backlinks require frame inference

Implementation

- We achieve automation by implementing the proof system in Cyclist
    - A generic framework for cyclic proof search
    - Proof objects can be extracted for certification
- Entailment queries also handled by Cyclist
- Procedure calls and backlinks require frame inference
    - Unfolds predicates and matches atomic spatial assertions

Implementation

- We achieve automation by implementing the proof system in CYCLIST
  - A generic framework for cyclic proof search
  - Proof objects can be extracted for certification
- Entailment queries also handled by CYCLIST
- Procedure calls and backlinks require frame inference
  - Unfolds predicates and matches atomic spatial assertions
  - Requires deciding entailment of sets of constraints $\alpha < \beta$

Implementation

- We achieve automation by implementing the proof system in Cyclist
  - A generic framework for cyclic proof search
  - Proof objects can be extracted for certification
- Entailment queries also handled by Cyclist
- Procedure calls and backlinks require frame inference
  - Unfolds predicates and matches atomic spatial assertions
  - Requires deciding entailment of sets of constraints $\alpha < \beta$
- Currently, we need to provide procedure summaries

A Cyclic Termination Proof for `shuffle`

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

A Cyclic Termination Proof for `shuffle`

proc shuffle(x) $\{\mathsf{list}_\alpha(x)\}$ {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } $\{\mathsf{list}_\alpha(x)\}$

A Cyclic Termination Proof for `shuffle`

proc shuffle(x) $\{\mathsf{list}_\alpha(\mathsf{x})\}$ {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } $\{\mathsf{list}_\alpha(\mathsf{x})\}$

$$\frac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil \{ y:=*x; reverse(y); shuffle(y); *x:=y; \}}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\mathrm{proc})$$

A Cyclic Termination Proof for `shuffle`

$\texttt{proc shuffle(x)}\,\{\mathsf{list}_\alpha(\mathsf{x})\}\,\{$

$\quad\texttt{if x!=nil}\ \{\,\texttt{y:=*x;}\,\texttt{reverse(y);}\,\texttt{shuffle(y);}\,\}\,\}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$

$$\frac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\mathrm{proc})$$

A Cyclic Termination Proof for `shuffle`

proc shuffle(x) $\{\mathsf{list}_\alpha(x)\}$ {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } $\{\mathsf{list}_\alpha(x)\}$

$$\dfrac{\dfrac{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(x)\}\ \texttt{y:=*x; \dots}\ \{\mathsf{list}_\alpha(x)\} \qquad \{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(x)\}\ \epsilon\ \{\mathsf{list}_\alpha(x)\}}{\{\mathsf{list}_\alpha(x)\}\ \texttt{if x!=nil \dots}\ \{\mathsf{list}_\alpha(x)\}}\ (\mathrm{if})}{\{\mathsf{list}_\alpha(x)\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(x)\}}\ (\mathrm{proc})$$

A Cyclic Termination Proof for `shuffle`

proc shuffle(x) $\{\mathsf{list}_\alpha(\mathsf{x})\}$ {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } $\{\mathsf{list}_\alpha(\mathsf{x})\}$

$$\dfrac{\dfrac{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\} \qquad \dfrac{}{\{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\,(\models)}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\,(\text{if})}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\,(\text{proc})$$

A Cyclic Termination Proof for `shuffle`

```
proc shuffle(x) {list_α(x)} {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } {list_α(x)}
```

$$
\dfrac{\dfrac{\dfrac{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{case } \mathbf{list}) \qquad \dfrac{}{\{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\models)}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{if})}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{proc})
$$

A Cyclic Termination Proof for `shuffle`

$$\texttt{proc shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}\ \{$$
$$\quad \texttt{if x!=nil \{ y:=*x; reverse(y); shuffle(y); \} \}}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\beta < \alpha \wedge \mathsf{x} \mapsto y * \mathsf{list}_\beta(y)\}\ \texttt{rev(y); }\ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \texttt{y:=*x; }\ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{load})}{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{y:=*x; }\ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{case }\mathbf{list}) \qquad \dfrac{}{\{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\models)}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil }\ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{if})}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{proc})$$

A Cyclic Termination Proof for `shuffle`

$\texttt{proc shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}\ \{$

$\quad \texttt{if x!=nil \{ y:=*x; reverse(y); shuffle(y); \} \}}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \, \mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\} \texttt{rev(y);} \left\{ \quad \right\} \qquad \{ \qquad \} \ \texttt{shuf(y);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \texttt{rev(y);} \ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\,(\text{seq})}{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\,(\text{load})}{\{x \neq \text{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\,(\text{case } \mathbf{list}) \qquad \dfrac{}{\{x = \text{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\,(\models)}{\dfrac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\,(\text{proc})}\,(\text{if})$$

A Cyclic Termination Proof for shuffle

```
proc shuffle(x) {list_α(x)} {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } {list_α(x)}
```

$\{\mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \{\mathsf{list}_\beta(\mathsf{y})\}$

$\left\{\begin{array}{c}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{array}\right\}\ \mathtt{rev(y);}\ \left\{\quad\right\} \qquad \{\quad\}\ \mathtt{shuf(y);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$ (seq)

$\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}$ (load)

$\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}$ (case **list**)

$\{x \neq \mathrm{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}$

$\overline{\{x = \mathrm{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}$ ($\models$)

(if)

$\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}$ (proc)

$\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$

A Cyclic Termination Proof for `shuffle`

```
proc shuffle(x) {list_α(x)} {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } {list_α(x)}
```

$$\frac{\{\mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \{\mathsf{list}_\beta(\mathsf{y})\}}{\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \,\mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}\ \mathtt{rev(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \,\mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}}\ (\text{frame})$$

$$\frac{\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \,\mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}\ \mathtt{rev(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \,\mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\} \qquad \{\qquad\qquad\}\ \mathtt{shuf(y);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{seq})$$

$$\frac{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{load})$$

$$\frac{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{x \neq \mathrm{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{case } \mathbf{list}) \qquad \frac{}{\{x = \mathrm{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\models)$$

$$\frac{\{x \neq \mathrm{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\} \qquad \{x = \mathrm{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{if})$$

$$\frac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{proc})$$

A Cyclic Termination Proof for `shuffle`

```
proc shuffle(x) {list_α(x)} {
     if x!=nil { y:=*x; reverse(y); shuffle(y); } } {list_α(x)}
```

$$\dfrac{\{\mathsf{list}_\beta(\mathsf{y})\}\ \texttt{rev(y);}\ \{\mathsf{list}_\beta(\mathsf{y})\}}{\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \,\mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}\ \texttt{rev(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \,\mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}}\ (\text{frame})$$

$$\dfrac{\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \,\mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}\ \texttt{rev(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \,\mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\} \qquad \{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \texttt{shuf(y);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \texttt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{seq})$$

$$\dfrac{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \texttt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{load})$$

$$\dfrac{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{case } \mathbf{list})$$

$$\dfrac{}{\{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\models)$$

$$\dfrac{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\} \qquad \{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{if})$$

$$\dfrac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{proc})$$

A Cyclic Termination Proof for `shuffle`

```
proc shuffle(x) {list_α(x)} {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } {list_α(x)}
```

$$\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{shuf(x);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$$

$$\dfrac{\{\mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \{\mathsf{list}_\beta(\mathsf{y})\}}{\left\{\begin{array}{c}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{array}\right\}\ \mathtt{rev(y);}\ \left\{\begin{array}{c}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{array}\right\}}\ (\text{frame})$$

$$\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{shuf(y);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$$

$$\dfrac{\cdots \qquad \cdots}{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{seq})$$

$$\dfrac{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{load})$$

$$\dfrac{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{case } \mathbf{list})$$

$$\dfrac{}{\{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\models)$$

$$\dfrac{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\} \qquad \{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{if})$$

$$\dfrac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{proc})$$

A Cyclic Termination Proof for `shuffle`

`proc shuffle(x)` $\{\mathsf{list}_\alpha(\mathsf{x})\}$ `{`

`if x!=nil { y:=*x; reverse(y); shuffle(y); } }` $\{\mathsf{list}_\alpha(\mathsf{x})\}$

$$\frac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{shuf(x);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}\ (\mathsf{y})\}\ \mathtt{shuf(y);}\ \{\mathsf{list}\ (\mathsf{y})\}}\ \text{(subst)}$$

$$\frac{\{\mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \{\mathsf{list}_\beta(\mathsf{y})\}}{\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}\ \mathtt{rev(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}}\ \text{(frame)}$$

$$\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{shuf(y);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$$

$$\frac{\cdots \qquad \cdots}{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ \text{(seq)}$$

$$\frac{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ \text{(load)}$$

$$\frac{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \mathsf{list}_\beta(v)\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ \text{(case list)}$$

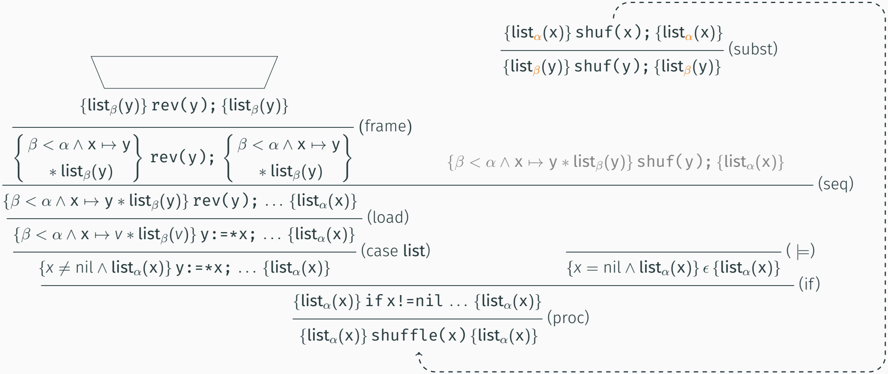
$$\frac{}{\{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\models)$$

$$\frac{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\} \qquad \{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ \text{(if)}$$

$$\frac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ \text{(proc)}$$

A Cyclic Termination Proof for shuffle

proc shuffle(x) $\{\mathsf{list}_\alpha(x)\}$ {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } $\{\mathsf{list}_\alpha(x)\}$



A Cyclic Termination Proof for `shuffle`

$$\texttt{proc shuffle(x)}\ \{\mathsf{list}_\alpha(x)\}\ \{$$
$$\quad \texttt{if x!=nil \{ y:=*x; reverse(y); shuffle(y); \} \}}\ \{\mathsf{list}_\alpha(x)\}$$

$$\dfrac{\{\mathsf{list}_\alpha(x)\}\ \texttt{shuf(x);}\ \{\mathsf{list}_\alpha(x)\}}{\{\mathsf{list}_\beta(y)\}\ \texttt{shuf(y);}\ \{\mathsf{list}_\beta(y)\}}\ (\text{subst})$$

$$\dfrac{\{\mathsf{list}_\beta(y)\}\ \texttt{rev(y);}\ \{\mathsf{list}_\beta(y)\}}{\left\{\begin{matrix}\beta < \alpha \wedge x \mapsto y\\ * \mathsf{list}_\beta(y)\end{matrix}\right\}\ \texttt{rev(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge x \mapsto y\\ * \mathsf{list}_\beta(y)\end{matrix}\right\}}\ (\text{frame})$$

$$\dfrac{\{\mathsf{list}_\beta(y)\}\ \texttt{shuf(y);}\ \{\mathsf{list}_\beta(y)\}}{\left\{\begin{matrix}\beta < \alpha \wedge x \mapsto y\\ * \mathsf{list}_\beta(y)\end{matrix}\right\}\ \texttt{shuf(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge x \mapsto y\\ * \mathsf{list}_\beta(y)\end{matrix}\right\}}\ (\text{frame})$$

$$\{\beta < \alpha \wedge x \mapsto y * \mathsf{list}_\beta(y)\}\ \texttt{shuf(y);}\ \{\mathsf{list}_\alpha(x)\}$$

$$\dfrac{\cdots \qquad \cdots}{\{\beta < \alpha \wedge x \mapsto y * \mathsf{list}_\beta(y)\}\ \texttt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(x)\}}\ (\text{seq})$$

$$\dfrac{\{\beta < \alpha \wedge x \mapsto y * \mathsf{list}_\beta(y)\}\ \texttt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(x)\}}{\{\beta < \alpha \wedge x \mapsto v * \mathsf{list}_\beta(v)\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(x)\}}\ (\text{load})$$

$$\dfrac{\{\beta < \alpha \wedge x \mapsto v * \mathsf{list}_\beta(v)\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(x)\}}{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(x)\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(x)\}}\ (\text{case } \mathbf{list})$$

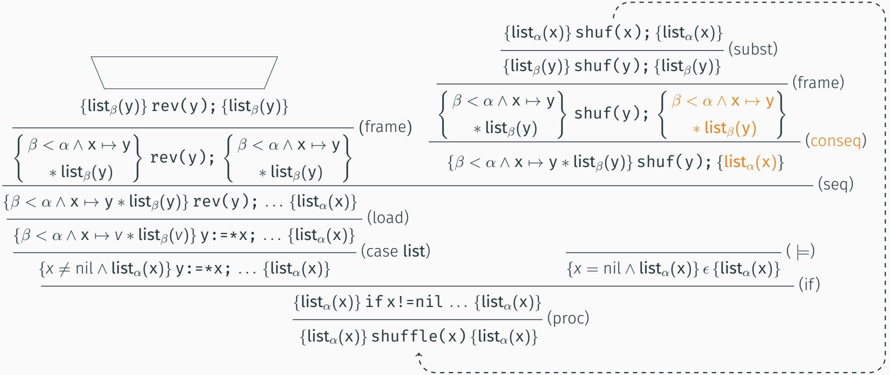
$$\dfrac{}{\{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(x)\}\ \epsilon\ \{\mathsf{list}_\alpha(x)\}}\ (\models)$$

$$\dfrac{\{x \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(x)\}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(x)\} \qquad \{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(x)\}\ \epsilon\ \{\mathsf{list}_\alpha(x)\}}{\{\mathsf{list}_\alpha(x)\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(x)\}}\ (\text{if})$$

$$\dfrac{\{\mathsf{list}_\alpha(x)\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(x)\}}{\{\mathsf{list}_\alpha(x)\}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(x)\}}\ (\text{proc})$$

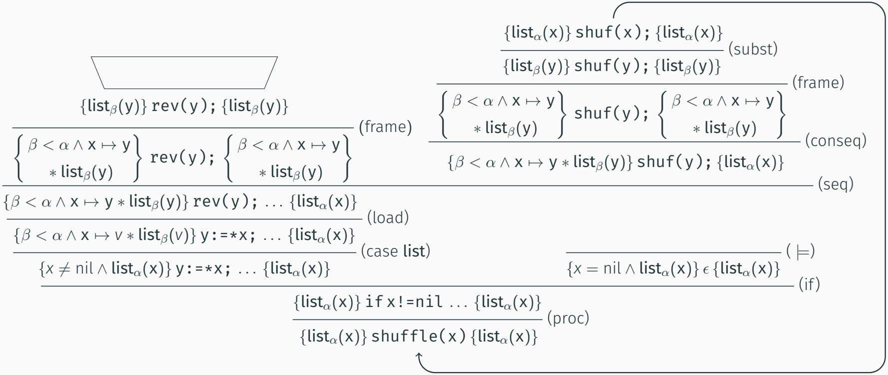
A Cyclic Termination Proof for shuffle

```
proc shuffle(x) {list_α(x)} {
     if x!=nil { y:=*x; reverse(y); shuffle(y); } } {list_α(x)}
```



A Cyclic Termination Proof for `shuffle`

proc shuffle(x) $\{\mathsf{list}_\alpha(\mathsf{x})\}$ {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } $\{\mathsf{list}_\alpha(\mathsf{x})\}$



A Cyclic Termination Proof for `shuffle`

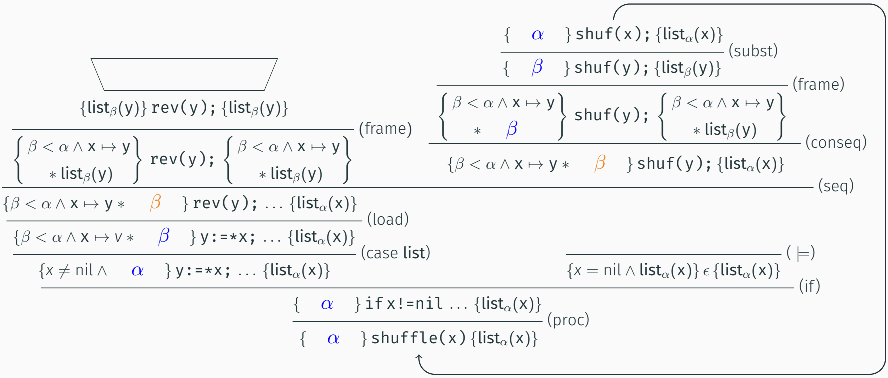
$\texttt{proc shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}\ \{$

$\quad \texttt{if x!=nil \{ y:=*x; reverse(y); shuffle(y); \} \}}\ \{\mathsf{list}_\alpha(\mathsf{x})\}$

$$\dfrac{\dfrac{\dfrac{\{\mathsf{list}_\beta(\mathsf{y})\}\ \texttt{rev(y);}\ \{\mathsf{list}_\beta(\mathsf{y})\}}{\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}\ \texttt{rev(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}}\text{(frame)} \qquad \dfrac{\dfrac{\dfrac{\{\ \alpha\ \}\ \texttt{shuf(x);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\ \beta\ \}\ \texttt{shuf(y);}\ \{\mathsf{list}_\beta(\mathsf{y})\}}\text{(subst)}}{\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \beta\end{matrix}\right\}\ \texttt{shuf(y);}\ \left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}}\text{(frame)}}{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \ \beta\ \}\ \texttt{shuf(y);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\text{(conseq)}}{\dfrac{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \ \beta\ \}\ \texttt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\dfrac{\{\beta < \alpha \wedge \mathsf{x} \mapsto v * \ \beta\ \}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{x \neq \mathsf{nil} \wedge \ \alpha\ \}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\text{(case }\mathsf{list}\text{)}}\text{(load)}}\text{(seq)}$$

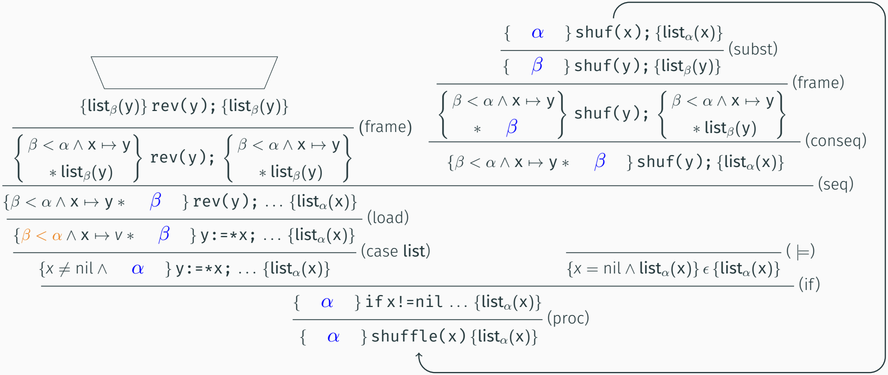
$$\dfrac{\dfrac{\{x \neq \mathsf{nil} \wedge \ \alpha\ \}\ \texttt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\} \qquad \dfrac{}{\{x = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}(\models)}{\{\ \alpha\ \}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\text{(if)}}{\{\ \alpha\ \}\ \texttt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\text{(proc)}$$

proc shuffle(x) $\{\mathsf{list}_\alpha(\mathsf{x})\}$ {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } $\{\mathsf{list}_\alpha(\mathsf{x})\}$



A Cyclic Termination Proof for `shuffle`

proc shuffle(x) $\{\mathsf{list}_\alpha(\mathsf{x})\}$ {
    if x!=nil { y:=*x; reverse(y); shuffle(y); } } $\{\mathsf{list}_\alpha(\mathsf{x})\}$



Some Related Tools

- MUTANT (Berdine et al. '06)
  THOR (Magill et al. '10)

Some Related Tools

- Mutant (Berdine et al. '06)
  Thor (Magill et al. '10)
- Costa (Albert et al. '07)
  Julia (Spoto et al. '10)
  AProVE (Giesl et al. '14)

Some Related Tools

- MUTANT (Berdine et al. '06)
  THOR (Magill et al. '10)
- Costa (Albert et al. '07)
  Julia (Spoto et al. '10)
  AProVE (Giesl et al. '14)
- Verifast (Jacobs et al. '15)

Some Related Tools

- MUTANT (Berdine et al. '06)
  THOR (Magill et al. '10)
- Costa (Albert et al. '07)
  Julia (Spoto et al. '10)
  AProVE (Giesl et al. '14)
- Verifast (Jacobs et al. '15)
- HIPTNT+ (Le, Qin & Chin '15)

Some Related Tools

- MUTANT (Berdine et al. '06)
  THOR (Magill et al. '10)
- Costa (Albert et al. '07)
  Julia (Spoto et al. '10)
  AProVE (Giesl et al. '14)
- Verifast (Jacobs et al. '15)
- HIPTNT+ (Le, Qin & Chin '15)

Empirical Evaluation: Comparison with HipTNT+

Benchmark test	Time (sec) / % Annotated	
	HipTNT+	Cyclist
traverse acyclic linked list	0.31 (25%)	0.02 (33%)
traverse cyclic linked list	0.52 (29%)	0.02 (38%)
append acyclic linked lists	0.36 (25%)	0.03 (10%)
TPDB Shuffle	1.79 (22%)	0.21 (29%)
TPDB Alternate	6.33 (13%)	1.47 (12%)
TPDB UnionFind	4.03 (26%)	1.21 (25%)

Empirical Evaluation: Comparison with AProVE

		Time (seconds)		
Benchmark Suite	Test	AProVE	CYCLIST	(% Annot.)
Costa_Julia_09-Recursive	Ackermann	3.82	0.14	(18%)
	BinarySearchTree	1.41	0.95	(13%)
	BTree	1.77	0.03	(22%)
	List	1.43	1.74	(19%)
Julia_10-Recursive	AckR	3.22	0.14	(18%)
	BTreeR	2.68	0.03	(22%)
	Test8	2.95	0.97	(13%)
AProVE_11_Recursive	CyclicAnalysisRec	2.61	5.21	(27%)
	RotateTree	5.86	0.32	(14%)
	SharingAnalysisRec	2.47	4.72	(16%)
	UnionFind	TIMEOUT	1.21	(25%)
BOG_RTA_11	Alternate	5.47	1.47	(12%)
	AppE	2.19	0.09	(23%)
	BinTreeChanger	3.38	3.33	(20%)
	CAppE	2.04	1.78	(25%)
	ConvertRec	3.72	0.06	(38%)
	DupTreeRec	4.18	0.03	(20%)
	GrowTreeR	3.53	0.05	(20%)
	MirrorBinTreeRec	4.96	0.02	(22%)
	MirrorMultiTreeRec	5.16	0.63	(33%)
	SearchTreeR	2.74	0.34	(14%)
	Shuffle	11.72	0.21	(29%)
	TwoWay	1.94	0.02	(25%)

Future Work

- More expressive contraints for predicate approximations

Future Work

- More expressive contraints for predicate approximations
- Can we *infer* procedure specifications?

Future Work

- More expressive contraints for predicate approximations
- Can we infer procedure specifications?
    - Predicate label annotations

Future Work

- More expressive contraints for predicate approximations
- Can we infer procedure specifications?
    - Predicate label annotations
    - Entire pre-/post-conditions (bi-abduction)

Thank You

`github.com/ngorogiannis/cyclist`