Automatic Cyclic Termination Proofs for Recursive Procedures in Separation Logic

Reuben Rowe and James Brotherston

University College London

TAPAS, Edinburgh, Wednesday 7th September 2016

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

Automatically Proving Termination: Some Solutions

- MUTANT, THOR

Automatically Proving Termination: Some Solutions

- Mutant, Thor
- Julia, Costa, AProVE

Automatically Proving Termination: Some Solutions

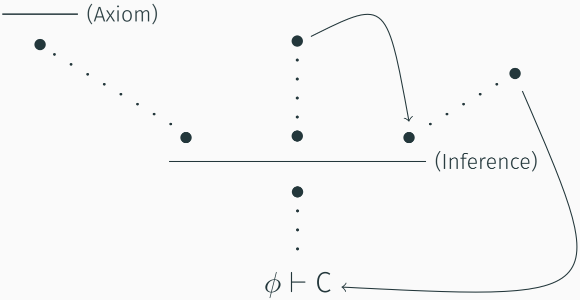
- MUTANT, THOR
- Julia, Costa, AProVE
- Dafny

Automatically Proving Termination: Some Solutions

- Mutant, Thor
- Julia, Costa, AProVE
- Dafny
- HipTNT+

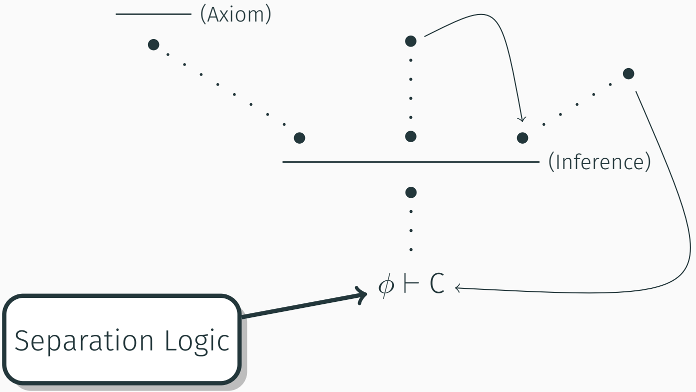
Automatically Proving Termination using Cyclic Proof

- Following the approach of Brotherston Et Al. (POPL '08)



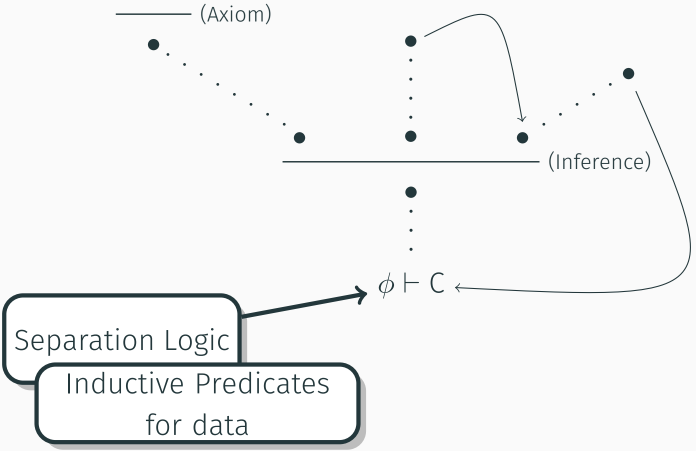
Automatically Proving Termination using Cyclic Proof

- Following the approach of Brotherston Et Al. (POPL ’08)



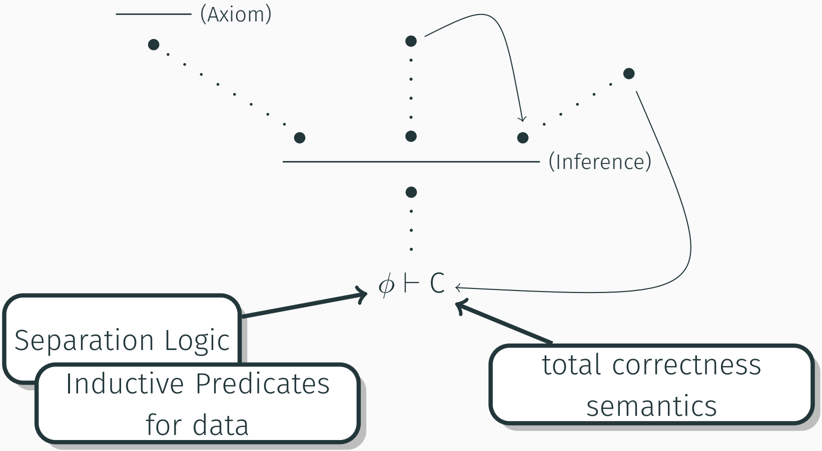
Automatically Proving Termination using Cyclic Proof

- Following the approach of Brotherston Et Al. (POPL '08)



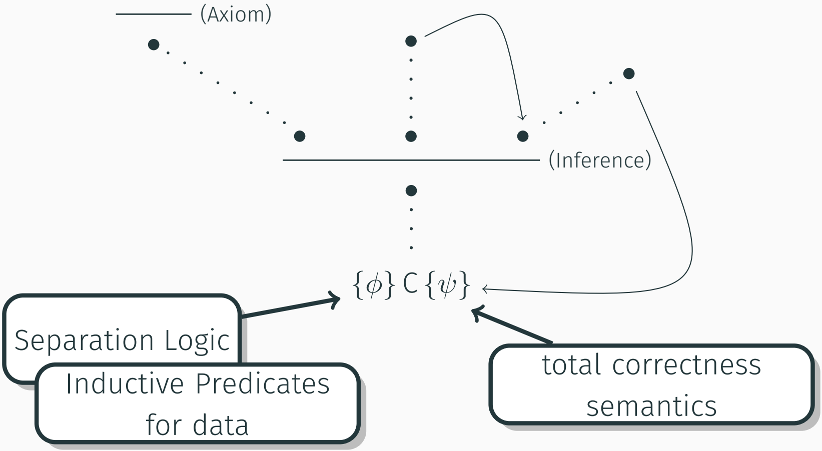
Automatically Proving Termination using Cyclic Proof

- Following the approach of Brotherston Et Al. (POPL '08)



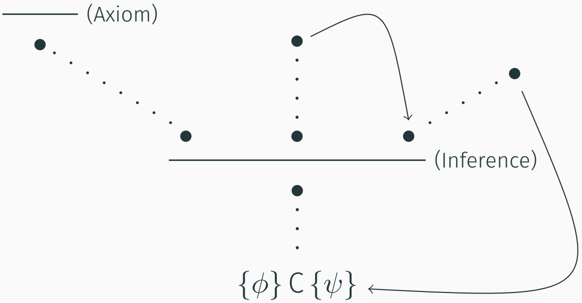
Automatically Proving Termination using Cyclic Proof

- 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

Advantages of Using Cyclic Proof

- Termination measures extracted automatically

Advantages of Using Cyclic Proof

- Termination measures extracted automatically
- Supports compositional reasoning

Advantages of Using Cyclic Proof

- Termination measures extracted automatically
- Supports compositional reasoning
- Naturally encapsulates inductive principles

Ingredients of our Approach: Symbolic Execution

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

Ingredients of our Approach: Symbolic Execution

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

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

Ingredients of our Approach: Symbolic Execution

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

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

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

Ingredients of our Approach: Inductive Predicates

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

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

Ingredients of our Approach: Inductive Predicates

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

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

- Explicit approximations used as termination measures, e.g.

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

Ingredients of our Approach: Inductive Predicates

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

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

- Explicit approximations used as termination measures, e.g.

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

- A logical rule schema allows case split

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

Ingredients of our Approach: Inductive Predicates

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

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

- Explicit approximations used as termination measures, e.g.

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

- A logical rule schema allows case split

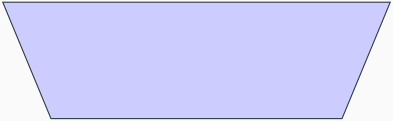
$$\frac{\{(\mathsf{x} = \mathsf{nil} \wedge \mathbf{emp}) * \phi\}\,\mathsf{C}\,\{\psi\} \quad \{(\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathbf{list}_{\beta}(\mathsf{x})) * \phi\}\,\mathsf{C}\,\{\psi\}}{\{\mathbf{list}_{\alpha}(\mathsf{x}) * \phi\}\,\mathsf{C}\,\{\psi\}}$$

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



$$\vdots$$

$$\{\mathsf{list}_\beta(y)\}\ \texttt{reverse(y)}\ \{\mathsf{list}_\beta(y)\}$$

A Cyclic Termination Proof for **shuffle**

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

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

A Cyclic Termination Proof for **shuffle**

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

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

A Cyclic Termination Proof for **shuffle**

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

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

A Cyclic Termination Proof for **shuffle**

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

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

A Cyclic Termination Proof for **shuffle**

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

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

A Cyclic Termination Proof for **shuffle**

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

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

A Cyclic Termination Proof for **shuffle**

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

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

A Cyclic Termination Proof for **shuffle**

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

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

A Cyclic Termination Proof for **shuffle**

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

$$
\cfrac{
  \cfrac{
    \cfrac{
      \cfrac{
        \cfrac{\{\mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \{\mathsf{list}_\beta(\mathsf{y})\}}{\left\{\begin{array}{c}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{array}\right\}\ \mathtt{rev(y);}\ \left\{\begin{array}{c}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{array}\right\}}\ (\text{frame})
        \qquad
        \{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{shuf(y);}\ \{\mathsf{list}_\alpha(\mathsf{x})\}
      }{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}\ \mathtt{rev(y);}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{seq})
    }{\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{v} * \mathsf{list}_\beta(\mathsf{v})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{load})
  }{\{\mathsf{x} \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{y:=*x;}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{case list})
  \qquad
  \cfrac{}{\{\mathsf{x} = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\models)
}{
  \cfrac{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{shuffle(x)}\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ (\text{proc})
}\ (\text{if})
$$

A Cyclic Termination Proof for **shuffle**

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

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

——— (frame)

$\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}$ `rev(y);` $\left\{\begin{matrix}\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} \\ * \ \mathsf{list}_\beta(\mathsf{y})\end{matrix}\right\}$ $\qquad$ $\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}$ `shuf(y);` $\{\mathsf{list}_\alpha(\mathsf{x})\}$

——— (seq)

$\{\beta < \alpha \wedge \mathsf{x} \mapsto \mathsf{y} * \mathsf{list}_\beta(\mathsf{y})\}$ `rev(y); ...` $\{\mathsf{list}_\alpha(\mathsf{x})\}$

——— (load)

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

——— (case **list**) $\qquad$ ——— ($\models$)

$\{\mathsf{x} \neq \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}$ `y:=*x; ...` $\{\mathsf{list}_\alpha(\mathsf{x})\}$ $\qquad$ $\{\mathsf{x} = \mathsf{nil} \wedge \mathsf{list}_\alpha(\mathsf{x})\}\ \epsilon\ \{\mathsf{list}_\alpha(\mathsf{x})\}$

——— (if)

$\{\mathsf{list}_\alpha(\mathsf{x})\}$ `if x!=nil ...` $\{\mathsf{list}_\alpha(\mathsf{x})\}$

——— (proc)

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

A Cyclic Termination Proof for **shuffle**

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

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

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

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

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

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

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

$$\frac{\cdots \quad \cdots}{\{\mathsf{list}_\alpha(\mathsf{x})\}\ \texttt{if x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}}\ \text{(if)}$$

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

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

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

(frame)

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

$$\{\mathsf{list}_\beta(y)\}\ \texttt{shuf(y);}\ \{\mathsf{list}_\beta(y)\}$$

(frame)

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

(conseq)

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

(seq)

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

(load)

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

(case list)

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

($\models$)

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

(if)

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

(proc)

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

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

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

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

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

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

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

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

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

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

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

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

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

(subst)

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

(frame)

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

(conseq)

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

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

(frame)

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

(seq)

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

(load)

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

(case **list**)

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

($\models$)

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

(if)

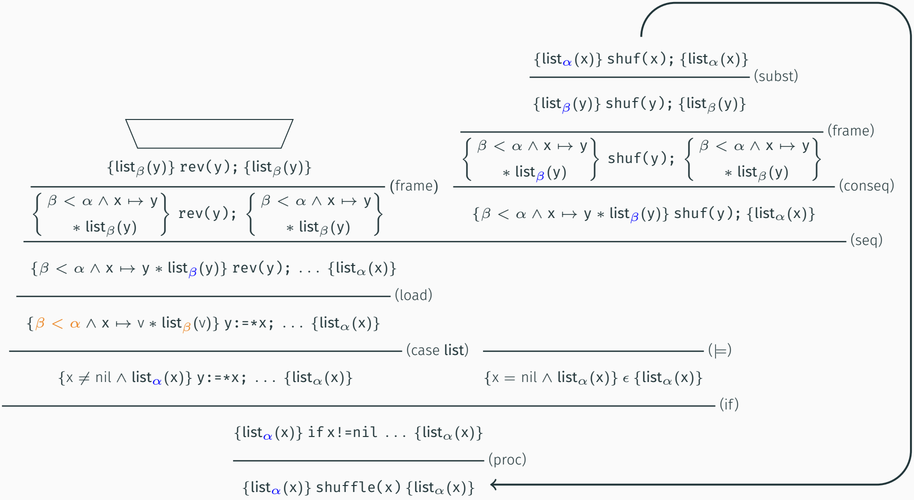
$$\{\mathsf{list}_\alpha(\mathsf{x})\}\ \mathtt{if\ x!=nil}\ \ldots\ \{\mathsf{list}_\alpha(\mathsf{x})\}$$

(proc)

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

A Cyclic Termination Proof for **shuffle**

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



Implementation

- Cyclist is a generic framework for cyclic proof search

Implementation

- Cyclist is a generic framework for cyclic proof search
    - Entailment queries also handled by Cyclist

Implementation

- CYCLIST is a generic framework for cyclic proof search
    - Entailment queries also handled by CYCLIST
- Currently, we need to provide procedure summaries

Implementation

- Cyclist is a generic framework for cyclic proof search
  - Entailment queries also handled by Cyclist
- Currently, we need to provide procedure summaries
- Procedure calls (and backlinks!) require frame inference

Implementation

- Cyclist is a generic framework for cyclic proof search
  - Entailment queries also handled by Cyclist
- Currently, we need to provide procedure summaries
- Procedure calls (and backlinks!) require frame inference
  - Driven by unfolding predicates/matching atomic spatial assertions

Implementation

- Cyclist is a generic framework for cyclic proof search
  - Entailment queries also handled by Cyclist
- Currently, we need to provide procedure summaries
- Procedure calls (and backlinks!) require frame inference
  - Driven by unfolding predicates/matching atomic spatial assertions
  - Requires deciding entailment of sets of constraints $\alpha < \beta$

Empirical Evaluation: Comparison with HipTNT+

Benchmark	Time (seconds)	
	HipTNT+	Cyclist
traverse acyclic linked list	0.31	0.02
traverse cyclic linked list	0.52	0.02
append acyclic linked lists	0.36	0.03
TPDB Shuffle	1.79	0.21
TPDB Alternate	6.33	1.47
TPDB UnionFind	4.03	1.21

Empirical Evaluation: Comparison with AProVE

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

Conclusions & Future Work

Conclusions & Future Work

- Cyclic proof-based termination analysis competes!

Conclusions & Future Work

- Cyclic proof-based termination analysis competes!
- More expressive contraints for predicate approximations

Conclusions & Future Work

- Cyclic proof-based termination analysis competes!
- More expressive contraints for predicate approximations
- Can we infer procedure specifications?

Conclusions & Future Work

- Cyclic proof-based termination analysis competes!
- More expressive contraints for predicate approximations
- Can we infer procedure specifications?
    - Constraints on explicit approximations

Conclusions & Future Work

- Cyclic proof-based termination analysis competes!
- More expressive contraints for predicate approximations
- Can we infer procedure specifications?
  - Constraints on explicit approximations
  - Entire pre-/post-conditions (bi-abduction)

github.com/ngorogiannis/cyclist