Realizability in Cyclic Proof

Extracting Ordering Information for Infinite Descent

Reuben N. S. Rowe ¹  James Brotherston ²

TABLEAUX, Brasília, Brazil, Tuesday 26th September 2017

¹School of Computing, University of Kent, Canterbury, UK

²Department of Computer Science, UCL, London, UK
Motivation: Program Termination

```c
struct ll { int data; ll *next; }

void rev(ll *x) { /* reverses list */ }

void shuffle(ll *x) {
    if ( x != NULL ) {
        ll *y = x -> next;
        rev(y);
        shuffle(y);
    }
}
```
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x, n) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next, n - 1)
void rev(ll *x) { /* reverses list */ }
void shuffle(ll *x) {
    if ( x != NULL ) {
        ll *y = x -> next;
        rev(y);
        shuffle(y);
    }
}
```
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x, n) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next, n - 1)
void rev(ll *x) { /* reverses list */ }
void shuffle(ll *x) { list(x, n) } {
    if ( x != NULL ) {

        ll *y = x -> next;

        rev(y);

        shuffle(y);

    }
} { list(x, n) }
```
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x, n) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next, n − 1)

void rev(ll *x) { /* reverses list */ }
void shuffle(ll *x) { list(x, n) } {
    if ( x != NULL ) {
        { list(x->next, n − 1) }
        ll *y = x -> next;
        { y = x->next ∧ list(y, n − 1) }
        rev(y);
        shuffle(y);
    }
} { list(x, n) }
```
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x, n) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next, n − 1)
void rev(ll *x) { list(x, n) } { ... } { list(x, n) }
void shuffle(ll *x) { list(x, n) } {
  if ( x != NULL ) {
    { list(x->next, n − 1) }
    ll *y = x -> next;
    { y = x->next ∧ list(y, n − 1) }
    rev(y);
    shuffle(y);
  }
} { list(x, n) }
```
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x, n) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next, n - 1)
void rev(ll *x) { list(x, n) } { ... } { list(x, n) }
void shuffle(ll *x) { list(x, n) } {
    if ( x != NULL ) {
        { list(x->next, n - 1) }
        ll *y = x -> next;
        { y = x->next ∧ list(y, n - 1) }
        rev(y);

        shuffle(y);
    }
} { list(x, n) }
```
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x, n) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next, n - 1)

void rev(ll *x) { list(x, n) } { ... } { list(x, n) }

void shuffle(ll *x) { list(x, n) } {
    if ( x != NULL ) {
        { list(x->next, n - 1) }
        ll *y = x -> next;
        { y = x->next ∧ list(y, n - 1) }
        rev(y);
        { y = x->next ∧ list(y, n - 1) }
        shuffle(y);
    }
} { list(x, n) }
```
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x, n) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next, n – 1)
void rev(ll *x) { list(x, n) } { ... } { list(x, n) }
void shuffle(ll *x) { list(x, n) } {
    if ( x != NULL ) {
        { list(x->next, n – 1) }
        ll *y = x -> next;
        { y = x->next ∧ list(y, n – 1) }
        rev(y);
        { y = x->next ∧ list(y, n – 1) }
        shuffle(y);
    }
} { list(x, n) }
```
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x, n) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next, n - 1)
void rev(ll *x) { list(x, n) } { ... } { list(x, n) }
void shuffle(ll *x) { list(x, n) } {
  if ( x != NULL ) {
    { list(x->next, n - 1) }
    ll *y = x -> next;
    { y = x->next ∧ list(y, n - 1) }
    rev(y);
    { y = x->next ∧ list(y, n - 1) }
    shuffle(y);
    { y = x->next ∧ list(y, n - 1) }
  }
} { list(x, n) }
```
Motivation: Program Termination

```
struct ll { int data; ll *next; }
list(x) \iff (n = 0 \land x = \texttt{NULL}) \lor \text{list(x->next)}

void rev(ll *x) { list_{\alpha}(x) } \{ ... \} \{ list_{\alpha}(x) \}
void shuffle(ll *x) { list_{\alpha}(x) } { 
    if ( x \neq \texttt{NULL} ) { 
        \{ list_{\beta}(x->next) \land \beta < \alpha \}
        ll *y = x -> next;
        \{ y = x->next \land list_{\beta}(y) \land \beta < \alpha \}
        rev(y);
        \{ y = x->next \land list_{\beta}(y) \land \beta < \alpha \}
        shuffle(y);
        \{ y = x->next \land list_{\beta}(y) \land \beta < \alpha \}
    }
    \{ list_{\alpha}(x) \}
}
```
Motivation: Program Termination

```c
struct ll {
    int data;
    ll *next;
};

list(x), (n = 0 ∧ x = NULL) \{ list(x) \}

list(x) \Rightarrow (n = 0 ∧ x = NULL) \lor (\{ list(x) \} \{ list(x) \})

void rev(ll *x) {
    list(x)
}

void shuffle(ll *x) {
    list(x)
}
```

Formulas \( \varphi \) (Models)

\[
\begin{align*}
& [\downarrow 1 \downarrow \beta \downarrow \alpha] : \models \varphi(x) \\
& [\downarrow 1 \downarrow \beta \downarrow \alpha] : \models \varphi(y) \\
& [\downarrow 1 \downarrow \beta \downarrow \alpha] : \models \varphi(z)
\end{align*}
\]
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x) \iff (n = 0 \land x = \text{NULL}) \lor \text{list(x->next)}
void rev(ll *x) { list_\alpha(x) \{ ... \} \{ list_\alpha(x) \}
void shuffle(ll *x) { list(x) \}
if (x)
{ list_\beta \{ ll *y = x->next \}
\{ y = rev(y) \}
\{ y = x->next \land \text{list}_\beta(y) \land \beta < \alpha \}
} \{ list_\alpha(x) \}
```

\[ [\cdot]: \text{Formulas} \rightarrow \wp(\text{Models}) \]
\[ [\cdot] \triangleleft [\cdot]_1 \subseteq \ldots \subseteq [\cdot]_\alpha \subseteq [\cdot]_{\alpha+1} \subseteq \ldots \subseteq [\cdot] \]
\[ \forall \alpha. [P(\vec{x})]_\alpha \subseteq [Q(\vec{y})]_\alpha \iff Q(\vec{y}) \leq P(\vec{x}) \]
Motivation: Program Termination

```
struct ll { int data; ll *next; }
list(x) ⇔ (n = 0 ∧ x = NULL) ∨ list(x→next)
void rev(ll *x) { listα(x) } { ... } { listα(x) }
void shuffle(ll *x) { list(y) } {
    if (x != NULL) {
        ll *y = x→next;
        rev(y);
        shuffle(y);
    }
}
```

Intra-procedural analysis produces verification conditions, in the form of entailments, e.g.

$$x \neq NULL ∧ y = x→next ∧ list(y) \Vdash list(x)$$
Motivation: Program Termination

```
struct ll { int data; ll *next; }
list(x) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next)
void rev(ll *x) { listα(x) } { ... } { listα(x) }
void shuffle(ll *x) { list(y) { ... } ...
```

(Axiom)

\[ P(\vec{x}) \implies Q(\vec{y}) \]

(Inference)
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x) ⇔ (n = 0 ∧ x = NULL) ∨ list(x->next)
void rev(ll *x) { listα(x) } { ... } { listα(x) }
void shuffle(ll *x) { list(y) { ...
```

(Axiom)

\[ Q(\vec{y}) ≤? P(\vec{x}) \]

...\( P(\vec{x}) \) ... ⊨ ... \( Q(\vec{y}) \) ...
Motivation: Program Termination

```c
struct ll { int data; ll *next; }
list(x) ↔ (n = 0 ∧ x = NULL) ∨ list(x->next)
void rev(ll *x) { listα(x) } { ... } { listα(x) }
void shuffle(ll *x) { list(y) { ...
```

(Axiom)

\[ Q(\vec{y}) \prec P(\vec{x}) \]

\[ P(\vec{x}) \vdash \ldots Q(\vec{y}) \ldots \]
Overview of Results

We show that:

- Information about semantic inclusions between inductive predicates can be extracted from cyclic proofs of entailments.
Overview of Results

We show that:

• Information about semantic inclusions between inductive predicates can be extracted from cyclic proofs of entailments
  • These inclusions hold when the proof graph satisfies a structural (realizability) condition that we define
We show that:

- Information about semantic inclusions between inductive predicates can be extracted from cyclic proofs of entailments
  - These inclusions hold when the proof graph satisfies a structural (realizability) condition that we define
- The realizability condition is equivalent to a containment between two weighted automata that can be constructed from the proof graph
Overview of Results

We show that:

• Information about semantic inclusions between inductive predicates can be extracted from cyclic proofs of entailments
  • These inclusions hold when the proof graph satisfies a structural (realizability) condition that we define

• The realizability condition is equivalent to a containment between two weighted automata that can be constructed from the proof graph
  • Under certain extra structural conditions, this containment falls within existing decidability results
A Cyclic Proof in LK Sequent Calculus with Equality

\[ \begin{align*}
\Rightarrow & \ \text{N} \ 0 \\
\text{N} \ x & \Rightarrow \ \text{N} \ s \ x \\
\Rightarrow & \ \text{E} \ 0 \\
\text{O} \ x & \Rightarrow \ \text{E} \ s \ x \\
\text{E} \ x & \Rightarrow \ \text{O} \ s \ x
\end{align*} \]

\[ \begin{align*}
\text{Ex} & \vdash \text{N} \ x \\
\text{E} \ z & \vdash \text{N} \ z \\
\frac{}{} & \frac{}{} \Rightarrow \text{(Subst)} \\
\text{E} \ z & \vdash \text{N} \ s \ z \\
\frac{}{} & \frac{}{} \Rightarrow \text{(N R}_2\text{)} \\
y = sz, \text{E} \ z & \vdash \text{N} \ y \\
\frac{}{} & \frac{}{} \Rightarrow \text{(Case O)} \\
\text{O} \ y & \vdash \text{N} \ y \\
\frac{}{} & \frac{}{} \Rightarrow \text{(N R}_2\text{)} \\
x = 0, \text{O} \ y & \vdash \text{N} \ x \\
\frac{}{} & \frac{}{} \Rightarrow \text{(Case E)} \\
\text{Ex} & \vdash \text{N} \ x
\end{align*} \]
A Cyclic Proof in LK Sequent Calculus with Equality

\[ \begin{align*}
\Rightarrow & \quad N \ 0 \\
N \ x & \quad \Rightarrow \quad N \ sx \\
\Rightarrow & \quad E \ 0 \\
O \ x & \quad \Rightarrow \quad E \ sx \\
E \ x & \quad \Rightarrow \quad O \ sx
\end{align*} \]

\[ \begin{align*}
& \quad Ex \vdash N \ x \\
& \quad \quad \quad (\text{Subst}) \\
& \quad Ez \vdash N \ z \\
& \quad \quad \quad (\text{N R}_2) \\
& \quad Ez \vdash N \ sz \\
& \quad \quad \quad (=L) \\
& \quad y = sz, Ez \vdash N \ y \\
& \quad \quad \quad (\text{Case O}) \\
& \quad Oy \vdash N \ y \\
& \quad \quad \quad (\text{N R}_2) \\
& \quad Oy \vdash N \ sy \\
& \quad \quad \quad (=L) \\
& \quad x = sy, Oy \vdash N \ x \\
& \quad \quad \quad (\text{Case E}) \\
& \quad \quad \quad Ex \vdash N \ x
\end{align*} \]
A Cyclic Proof in LK Sequent Calculus with Equality

\[
\begin{align*}
& \vdash N \, 0 \\
& \vdash N \, x \rightarrow N \, s\, x \\
& \vdash E \, 0 \\
& \vdash O \, x \rightarrow E \, s\, x \\
& \vdash E \, x \rightarrow O \, s\, x \\
& \vdash E \, x \rightarrow N \, x \\
& \vdash N \, x \vdash N \, 0 \\
& \vdash N \, x \vdash N \, s\, z \\
& \vdash N \, x \vdash y = s\, z, E \, z \vdash N \, y \\
& \vdash N \, y \vdash N \, s\, y \\
& \vdash O \, y \vdash N \, x \\
& \vdash E \, z \vdash N \, s\, z \\
& \vdash E \, z \vdash y = s\, z, E \, z \vdash N \, y \\
& \vdash E \, z \vdash N \, s\, z \\
& \vdash E \, z \vdash y = s\, z, E \, z \vdash N \, y \\
\end{align*}
\]
A Cyclic Proof in LK Sequent Calculus with Equality

\[
\begin{align*}
E x & \vdash N x \quad (\text{Subst}) \\
E z & \vdash N z \\
E z & \vdash N sz \\
\quad & \vdash N sz \\
y = sz, E z & \vdash N y & \quad (\text{Case O}) \\
O y & \vdash N y \\
O y & \vdash N sy & \quad (\text{Case E}) \\
E x & \vdash N x
\end{align*}
\]

\[
\begin{align*}
\Rightarrow & \quad N 0 \\
N x & \Rightarrow N sx \\
\Rightarrow & \quad E 0 \\
O x & \Rightarrow E sx \\
E x & \Rightarrow O sx
\end{align*}
\]
A Cyclic Proof in LK Sequent Calculus with Equality

\[ \vdash N x \]

\[(\text{Subst})\]

\[ E z \vdash N z \]

\[(\text{N R}_2)\]

\[ E z \vdash N s z \]

\[(=L)\]

\[ y = s z, E z \vdash N y \]

\[(\text{Case O})\]

\[ O y \vdash N y \]

\[(\text{N R}_2)\]

\[ O y \vdash N s y \]

\[(=L)\]

\[ x = s y, O y \vdash N x \]

\[(\text{Case E})\]

\[ E x \vdash N x \]

\[ \Rightarrow N 0 \]

\[ N x \Rightarrow N s x \]

\[ \Rightarrow E 0 \]

\[ O x \Rightarrow E s x \]

\[ E x \Rightarrow O s x \]
A cyclic proof graph is **globally sound** when every infinite path (going from conclusion to premise) is eventually followed by a **trace** of predicate formulas (on the left-hand side of sequents) which **progresses** (through a case-split) **infinitely often**.

A Cyclic Proof in LK Sequent Calculus with Equality

\[
\begin{align*}
\Rightarrow & \quad N \; 0 \\
N \; x & \Rightarrow \; N \; sx \\
\Rightarrow & \quad E \; 0 \\
O \; x & \Rightarrow \; E \; sx \\
E \; x & \Rightarrow \; O \; sx
\end{align*}
\]
A cyclic proof graph is globally sound when every infinite path (going from conclusion to premise) is eventually followed by a trace of predicate formulas (on the left-hand side of sequents) which progresses (through a case-split) infinitely often.
A cyclic proof graph is globally sound when every infinite path (going from conclusion to premise) is eventually followed by a trace of predicate formulas (on the left-hand side of sequents) which progresses (through a case-split) infinitely often.

\[
\begin{align*}
&\Rightarrow \ N \ 0 \\
&N \ x \Rightarrow \ N \ sx \\
&\Rightarrow \ E \ 0 \\
&O \ x \Rightarrow \ E \ sx \\
&E \ x \Rightarrow \ O \ sx
\end{align*}
\]

\[
\begin{align*}
&\vdash \ N \ 0 \\
&\vdash \ N \ 0 \\
&\vdash \ N \ sy \\
&\vdash \ N \ sx
\end{align*}
\]
A cyclic proof graph is *globally sound* when every infinite path (going from conclusion to premise) is eventually followed by a *trace* of predicate formulas (on the left-hand side of sequents) which *progresses* (through a case-split) infinitely often.

\[
\begin{align*}
\Rightarrow & \ N \ 0 \\
N \ x & \Rightarrow \ N \ sx \\
\Rightarrow & \ E \ 0 \\
O \ x & \Rightarrow \ E \ sx \\
E \ x & \Rightarrow \ O \ sx \\
\end{align*}
\]
A cyclic proof graph is **globally sound** when every infinite path (going from conclusion to premise) is eventually followed by a **trace** of predicate formulas (on the left-hand side of sequents) which **progresses** (through a case-split) **infinitely often**.

\[
\begin{align*}
\Rightarrow & \quad \text{N } 0 \\
\text{N } x & \quad \Rightarrow \quad \text{N } sx \\
\Rightarrow & \quad \text{E } 0 \\
\text{O } x & \quad \Rightarrow \quad \text{E } sx \\
\text{E } x & \quad \Rightarrow \quad \text{O } sx \\
\end{align*}
\]

**Case E**

\[
\begin{align*}
\text{O } y & \quad \vdash \quad \text{N } sy \\
\quad & \quad \Rightarrow \quad \text{N } 0 \\
\quad & \quad \vdash \quad \text{N } 0 \\
\quad & \quad \Rightarrow \quad \text{N } sx \\
\quad & \quad \vdash \quad \text{N } sx \\
\quad & \quad \Rightarrow \quad \text{E } 0 \\
\end{align*}
\]

**Case O**

\[
\begin{align*}
\text{y} = \text{sz}, \quad \text{E } z & \quad \vdash \quad \text{N } y \\
\quad & \quad \Rightarrow \quad \text{N } syn \\
\quad & \quad \vdash \quad \text{N } syn \\
\quad & \quad \Rightarrow \quad \text{O } y \\
\end{align*}
\]

**Case (L)**

\[
\begin{align*}
x = \text{sz}, \quad \text{O } y & \quad \vdash \quad \text{N } x \\
\quad & \quad \Rightarrow \quad \text{N } syn \\
\quad & \quad \vdash \quad \text{N } syn \\
\quad & \quad \Rightarrow \quad \text{E } x \\
\end{align*}
\]

**Substitution**

\[
\begin{align*}
\text{Ex} & \quad \vdash \quad \text{N } x \\
\end{align*}
\]
A cyclic proof graph is **globally sound** when every infinite path (going from conclusion to premise) is eventually followed by a **trace** of predicate formulas (on the left-hand side of sequents) which **progresses** (through a case-split) **indefinitely often**

\[
\begin{align*}
\Rightarrow & \ N \ 0 \\
N \ x & \Rightarrow \ N \ sx \\
\Rightarrow & \ E \ 0 \\
O \ x & \Rightarrow \ E \ sx \\
E \ x & \Rightarrow \ O \ sx
\end{align*}
\]
A cyclic proof graph is **globally sound** when every infinite path (going from conclusion to premise) is eventually followed by a **trace** of predicate formulas (on the left-hand side of sequents) which **progresses** (through a case-split) **infinitely often**.

\[
\begin{align*}
& \Rightarrow N 0 \\
& N x \Rightarrow N sx \\
& \Rightarrow E 0 \\
& O x \Rightarrow E sx \\
& E x \Rightarrow O sx
\end{align*}
\]

\[
\begin{align*}
& \frac{}{\frac{N 0}{N x}} \quad (N R_1) \\
& \frac{}{\frac{E z \vdash N z}{N s z}} \quad (N R_2) \\
& \frac{}{\frac{E z \vdash N s z}{(=L)}} \\
& y = sz, E z \vdash N y \quad (Case \ O) \\
& \frac{}{\frac{O y \vdash N y}{(=L)}} \\
& x = 0 \vdash N x \quad (=L) \\
& x = sy, O y \vdash N x \quad (Case \ E) \\
& Ex \vdash N x
\end{align*}
\]
A cyclic proof graph is **globally sound** when every infinite path (going from conclusion to premise) is eventually followed by a **trace** of predicate formulas (on the left-hand side of sequents) which **progresses** (through a case-split) infinitely often.

\[
\begin{align*}
\Rightarrow & \quad N \ 0 \\
N \ x & \Rightarrow \quad N \ sx \\
\Rightarrow & \quad E \ 0 \\
O \ x & \Rightarrow \quad E \ sx \\
E \ x & \Rightarrow \quad O \ sx
\end{align*}
\]

\[
\begin{align*}
\quad & \quad (N \ R_1) \\
\Rightarrow & \quad N \ 0 \\
\quad & \quad (=L) \\
x = 0 & \Rightarrow \quad N \ x \\
\quad & \quad (N \ R_2) \\
O \ y & \Rightarrow \quad N \ sy \\
\quad & \quad (=L) \\
x = sy, \ y & \Rightarrow \quad N \ x \\
\quad & \quad (Case \ E) \\
E \ x & \Rightarrow \quad N \ x
\end{align*}
\]
A cyclic proof graph is **globally sound** when every infinite path (going from conclusion to premise) is eventually followed by a **trace** of predicate formulas (on the left-hand side of sequents) which **progresses** (through a case-split) infinitely often.
A cyclic proof graph is **globally sound** when every infinite path (going from conclusion to premise) is eventually followed by a **trace** of predicate formulas (on the left-hand side of sequents) which **progresses** (through a case-split) infinitely often.

\[ \Rightarrow \text{N 0} \]
\[ \text{N x } \Rightarrow \text{ N sx} \]
\[ \Rightarrow \text{E 0} \]
\[ \text{O x } \Rightarrow \text{ E sx} \]
\[ \text{E x } \Rightarrow \text{ O sx} \]
**Definition (Inductive Definition Set)**

An *inductive definition set* contains productions $P_1 \vec{t}_1, \ldots, P_j \vec{t}_j \Rightarrow P_0 \vec{t}_0$.

**Definition (Characteristic Operators)**

Inductive definition sets $\Phi$ induce *characteristic operators* $\varphi_{\Phi}$ on predicate interpretations $X$ (functions from predicate formulas to sets of models):

$$\varphi_{\Phi}(X)(P \vec{t}\theta) = \{m \mid P_1 \vec{t}_1, \ldots, P_j \vec{t}_j \Rightarrow P \vec{t} \in \Phi, m \in X(P_i \vec{t}_i\theta) \text{ for all } 1 \leq i \leq j\}$$
Definition (Inductive Definition Set)

An *inductive definition set* contains productions $P_1 \overrightarrow{t}_1, \ldots, P_j \overrightarrow{t}_j \Rightarrow P_0 \overrightarrow{t}_0$

Definition (Characteristic Operators)

Inductive definition sets $\Phi$ induce *characteristic operators* $\varphi_\Phi$ on predicate interpretations $X$ (functions from predicate formulas to sets of models):

$$\varphi_\Phi(X)(P \overrightarrow{t} \theta) = \{ m \mid P_1 \overrightarrow{t}_1, \ldots, P_j \overrightarrow{t}_j \Rightarrow P \overrightarrow{t} \in \Phi, \ m \in X(P_i \overrightarrow{t}_i \theta) \text{ for all } 1 \leq i \leq j \}$$

The ordered set of predicate interpretations $(X, \sqsubseteq)$ is a *complete lattice*
**Definition (Inductive Definition Set)**

An *inductive definition set* contains productions $P_1 t_1, \ldots, P_j t_j \Rightarrow P_0 t_0$

**Definition (Characteristic Operators)**

Inductive definition sets $\Phi$ induce *characteristic operators* $\varphi_\Phi$ on predicate interpretations $X$ (functions from predicate formulas to sets of models):

$$\varphi_\Phi(X)(P \bar{t}\theta) = \{m \mid P_1 t_1, \ldots, P_j t_j \Rightarrow \bar{t} \in \Phi, m \in X(P_i t_i\theta) \text{ for all } 1 \leq i \leq j\}$$

The ordered set of predicate interpretations $(X, \sqsubseteq)$ is a **complete lattice**

Characteristic operators $\varphi_\Phi$ are **monotone** wrt $\sqsubseteq$
### Definition (Inductive Definition Set)

An *inductive definition set* contains productions $P_1 \vec{t}_1, \ldots, P_j \vec{t}_j \Rightarrow P_0 \vec{t}_0$.

### Definition (Characteristic Operators)

Inductive definition sets $\Phi$ induce *characteristic operators* $\varphi_{\Phi}$ on predicate interpretations $X$ (functions from predicate formulas to sets of models):

$$\varphi_{\Phi}(X)(P \vec{t} \theta) = \{ m \mid P_1 \vec{t}_1, \ldots, P_j \vec{t}_j \Rightarrow P \vec{t} \in \Phi, m \in X(P_i \vec{t}_i \theta) \text{ for all } 1 \leq i \leq j \}$$

The ordered set of predicate interpretations $(X, \sqsubseteq)$ is a **complete lattice**.

Characteristic operators $\varphi_{\Phi}$ are **monotone** wrt $\sqsubseteq$.

We interpret predicates using the least fixed point, $\llbracket \cdot \rrbracket_{\Phi} \overset{\text{def}}{=} \mu X. \varphi_{\Phi}(X)$

$$X_{\perp} \sqsubseteq \varphi_{\Phi}(X_{\perp}) \sqsubseteq \varphi_{\Phi}(\varphi_{\Phi}(X_{\perp})) \sqsubseteq \ldots \sqsubseteq \varphi_{\Phi}^\alpha(X_{\perp}) \sqsubseteq \ldots \sqsubseteq \mu X. \varphi_{\Phi}(X)$$
## Inductive Predicate Definitions and their Semantics

### Definition (Inductive Definition Set)

An *inductive definition set* contains productions $P_1 \overrightarrow{t_1}, \ldots, P_j \overrightarrow{t_j} \Rightarrow P_0 \overrightarrow{t_0}$

### Definition (Characteristic Operators)

Inductive definition sets $\Phi$ induce *characteristic operators* $\varphi_\Phi$ on predicate interpretations $X$ (functions from predicate formulas to sets of models):

$$\varphi_\Phi(X)(P \overrightarrow{t \theta}) = \{ m \mid P_1 \overrightarrow{t_1}, \ldots, P_j \overrightarrow{t_j} \Rightarrow P \overrightarrow{t} \in \Phi, \ m \in X(P_i \overrightarrow{t_i \theta}) \text{ for all } 1 \leq i \leq j \}$$

The ordered set of predicate interpretations $(\mathcal{X}, \sqsubseteq)$ is a **complete lattice**

Characteristic operators $\varphi_\Phi$ are **monotone** wrt $\sqsubseteq$

We interpret predicates using the least fixed point, $\mathcal{E}_\Phi \overset{\text{def}}{=} \mu X. \varphi_\Phi(X)$

$$\mathcal{E}_0 \subseteq \mathcal{E}_1 \subseteq \mathcal{E}_2 \subseteq \cdots \subseteq \mathcal{E}_\alpha \subseteq \cdots \mathcal{E}_\Phi$$
• Suppose, for contradiction, that the conclusion of the proof is not valid
  • That is, there is a counter-model of the sequent
Cyclic Proof Formalises Infinite Descent

• Suppose, for contradiction, that the conclusion of the proof is not valid
  • That is, there is a counter-model of the sequent

• By **local** soundness of the inference rules, we obtain an infinite sequence of counter-models for some infinite path in the proof
  • Each model can be mapped to an ever smaller approximation $\llbracket P \bar{t} \rrbracket_\alpha^\Phi$ in which it appears
  • These strictly decrease over a case-split

But $(X; \sqsubseteq)$ is a well-ordered set

contradiction!
• Suppose, for contradiction, that the conclusion of the proof is not valid
  • That is, there is a counter-model of the sequent
• By local soundness of the inference rules, we obtain an infinite sequence of counter-models for some infinite path in the proof
  • Each model can be mapped to an ever smaller approximation \([P \vec{t}]_\alpha^\phi\) in which it appears
  • These strictly decrease over a case-split
• By global soundness of the proof, this gives an infinitely descending chain in \((\mathcal{X}, \sqsubseteq)\)
  • But \((\mathcal{X}, \sqsubseteq)\) is a well-ordered set \(\Rightarrow\) contradiction!
Extracting Semantic Orderings from Cyclic Proofs

\[ \frac{\text{Ex} \vdash N \, x}{\text{Subst}} \]
\[ \frac{\text{Ez} \vdash N \, z}{(\text{N} \, \text{R}_2)} \]
\[ \frac{\text{Ez} \vdash N \, sz}{(=\text{L})} \]
\[ y = sz, \text{Ez} \vdash N \, y \quad \text{(Case O)} \]
\[ \frac{\text{Oy} \vdash N \, y}{(=\text{L})} \]
\[ \frac{\text{Oy} \vdash N \, sy}{(=\text{L})} \]
\[ x = sy, \text{Oy} \vdash N \, x \quad \text{(Case E)} \]
\[ \frac{\text{Ex} \vdash N \, 0}{(\text{N} \, \text{R}_1)} \]
\[ \frac{x = 0 \vdash N \, x}{(=\text{L})} \]
\[ \frac{x = 0 \vdash N \, 0}{(\text{N} \, \text{R}_1)} \]
\[ \Rightarrow N \, 0 \]
\[ N \, x \Rightarrow N \, sx \]
\[ \Rightarrow E \, 0 \]
\[ O \, x \Rightarrow E \, sx \]
\[ E \, x \Rightarrow O \, sx \]
The inductive definitions/semantics give immediately, e.g.

\[ \forall m, \alpha : m \in [E \ x]_\alpha \Rightarrow m \in [O \ x]_\alpha \]

and even

\[ \forall m, \alpha : m \in [E \ x]_\alpha \Rightarrow \exists \beta < \alpha . m \in [O \ x]_\beta \]

\[ \Rightarrow N \ 0 \]
\[ N \ x \Rightarrow N \ s \ x \]
\[ \Rightarrow E \ 0 \]
\[ O \ x \Rightarrow E \ s \ x \]
\[ E \ x \Rightarrow O \ s \ x \]

\[ \frac{E x \vdash N \ x}{(N \ R_1)} \]
\[ \frac{\vdash N \ 0}{(=L)} \]
\[ \frac{\vdash N \ x}{(=L)} \]
\[ \frac{E z \vdash N \ y}{(N \ R_2)} \]
\[ \frac{\vdash N \ y}{(=L)} \]
\[ \frac{x = s \ y, O y \vdash N \ x}{(=L)} \]
\[ \frac{x = s \ y, O y \vdash N \ x}{(Case \ E)} \]
The global soundness already gives

$$\forall m : m \in \llbracket Ex \rrbracket \Rightarrow m \in \llbracket Nx \rrbracket$$

but we would also like to know whether

$$\forall m, \alpha : m \in \llbracket Ex \rrbracket_{\alpha} \Rightarrow m \in \llbracket Nx \rrbracket_{\alpha}$$

i.e. $Nx \leq Ex$

$\Rightarrow Nx \Rightarrow Nx 0$

$\Rightarrow Ex \Rightarrow Ex 0$

The proof proceeds as follows:

- \(Ex \vdash Nx\) (Subst)
- \(Ez \vdash Nz\) (N R1)
- \(Ez \vdash Nsz\) (N R2)
- \(y = sz, Ez \vdash Ny\) (=L)
- \(Oy \vdash Ny\) (Case O)
- \(Oy \vdash Nsy\) (Case O)
- \(x = sy, Oy \vdash Nx\) (Case E)
- \(x = 0 \vdash Nx\) (N R1)
- \(y = s, Ex \vdash Ns\) (Subst)
To extract these semantic relationships from cyclic proofs:

- We have to consider traces along the right-hand side of sequents, which are
  - maximally finite
  - matched by some left-hand trace along the same path

- We then count the number of times each trace progresses
  - the left-hand one must progress at least as often as the right-hand one
Extracting Semantic Orderings: Example

\[
\begin{align*}
\Rightarrow & \quad N \ 0 \\
N \ x & \Rightarrow \ N \ sx \\
\Rightarrow & \quad E \ 0 \\
O \ x & \Rightarrow \ E \ sx \\
E \ x & \Rightarrow \ O \ sx
\end{align*}
\]

\[
\begin{align*}
Ex \vdash & \quad N \ x \quad \text{(Subst)} \\
& \quad E z \vdash \quad N \ z \quad \text{(N \ R_2)} \\
& \quad E z \vdash \quad N \ sz \\
& \quad y = sz, \ Ez \vdash \quad N \ y \\
& \quad \quad \quad \quad \quad \text{(Case \ O)} \\
& \quad O y \vdash \quad N \ y \\
& \quad O y \vdash \quad N \ sy \\
& \quad x = sy, \ O y \vdash \quad N \ x \\
& \quad Ex \vdash \quad N \ x
\end{align*}
\]
Extracting Semantic Orderings: Example

\[ \Rightarrow \propto N \ 0 \]
\[ \Rightarrow N \ x \ \Rightarrow \ N \ sx \]
\[ \Rightarrow E \ 0 \]
\[ O \ x \ \Rightarrow \ E \ sx \]
\[ E \ x \ \Rightarrow \ O \ sx \]
\[ N \ ss0 \ \Rightarrow \ O \ sss0 \]

\[ N \ x \ \Rightarrow \ N \ sx \]
\[ N \ ss0 \ \Rightarrow \ N \ ss0 \]
\[ N \ ss0 \ \Rightarrow \ N \ sss0 \]

\[ (Ax) \]
\[ N \ ss0 \ \Rightarrow \ N \ ss0 \]
\[ N \ ss0 \ \Rightarrow \ N \ sss0 \]
\[ y = sss0, N \ ss0 \ \Rightarrow \ N \ y \]
\[ (=L) \]
\[ y = s, E \ z \ \Rightarrow \ N \ y \]
\[ (=L) \]

\[ (N R_1) \]
\[ x = 0 \ \Rightarrow \ N \ x \]
\[ (=L) \]
\[ (N R_2) \]
\[ O \ y \ \Rightarrow \ N \ y \]
\[ (N R_2) \]
\[ O \ y \ \Rightarrow \ N \ sy \]
\[ (=L) \]
\[ x = sy, O \ y \ \Rightarrow \ N \ x \]
\[ (=L) \]

\[ E \ x \ \Rightarrow \ N \ x \]

\[ (Case \ O) \]

\[ (Case \ E) \]
Extracting Semantic Orderings: Example

\[ \Rightarrow N \ 0 \]
\[ N \ x \Rightarrow N \ sx \]
\[ \Rightarrow E \ 0 \]
\[ O \ x \Rightarrow E \ sx \]
\[ E \ x \Rightarrow O \ sx \]
\[ N \ ss0 \Rightarrow O \ sss0 \]
\[ \]
\[ \]
\[ (Ax) \]
\[ N \ ss0 \vdash N \ ss0 \]
\[ (N R_2) \]
\[ N \ ss0 \vdash N \ sss0 \]
\[ (=L) \]
\[ y = ss0, N \ ss0 \vdash N \ y \]
\[ (Subst) \]
\[ E \ z \vdash N \ z \]
\[ (N R_2) \]
\[ E \ z \vdash N \ sz \]
\[ (=L) \]
\[ y = sz, E \ z \vdash N \ y \]
\[ (Case \ O) \]
\[ O \ y \vdash N \ y \]
\[ (N R_2) \]
\[ O \ y \vdash N \ sy \]
\[ (=L) \]
\[ x = sy, O \ y \vdash N \ x \]
\[ (Case \ E) \]
\[ Ex \vdash N \ x \]

8/15
Extracting Semantic Orderings: Example

⇒ N 0
N x ⇒ N sx
⇒ E 0
O x ⇒ E sx
Ex ⇒ O sx
N ss0 ⇒ O sss0

⇒ N 0
⇒ E 0
⇒ E 0
⇒ E 0
⇒ N ss0
⇒ O sss0

(Ax)
N ss0 ⊢ N ss0
(N R₂)
N ss0 ⊢ N sss0
(=L)
y = sss0, N ss0 ⊢ Ny
(Case O)

(Subst)
Ex ⊢ N x
E z ⊢ N z
(N R₂)
E z ⊢ N sz
(=L)
y = sz, Ez ⊢ Ny

(E Subst)
E 0 ⊢ N 0
E x ⊢ N x
(E x)
O y ⊢ N y
(N R₂)
O y ⊢ N sy
(=L)
x = sy, O y ⊢ N x
(Case E)

Ex ⊢ N x
This trace is

- **fully** maximal: the final predicate is introduced by its rule
- **grounded**: the final predicate is derived from a zero premise production

(N.B. \( \forall m : m \in \{N 0\}_1 \))
Extracting Semantic Orderings: Example

\[ \Rightarrow N \ 0 \]
\[ N \ x \Rightarrow N \ sx \]
\[ \Rightarrow E \ 0 \]
\[ O \ x \Rightarrow E \ sx \]
\[ E \ x \Rightarrow O \ sx \]
\[ N \ ss0 \Rightarrow O \ sss0 \]

This trace is partially maximal: the final predicate is the active formula of an axiom.

\[ x = 0 \vdash N \ x \]

\[ E \ x \vdash N \ x \]

\[ N \ sx \vdash N \ ss0 \]

\[ N \ ss0 \vdash N \ sss0 \]

\[ y = ss0, \ N \ ss0 \vdash N \ y \]

\[ y = sz, \ Ez \vdash N \ y \]

\[ O \ y \vdash N \ y \]

\[ O \ y \vdash N \ sy \]

\[ x = sy, \ O \ y \vdash N \ x \]

\[ E \ x \vdash N \ x \]
Extracting Semantic Orderings: Example

\[ \Rightarrow N \ 0 \]
\[ N \ x \Rightarrow N \ s \ x \]
\[ \Rightarrow E \ 0 \]
\[ O \ x \Rightarrow E \ s \ x \]
\[ E \ x \Rightarrow O \ s \ x \]
\[ N \ ss0 \Rightarrow O \ sss0 \]

\[ \vdash N \ 0 \]
\[ N \ x \vdash N \ s \ x \]
\[ \vdash E \ 0 \]
\[ O \ x \vdash E \ s \ x \]
\[ E \ x \vdash O \ s \ x \]
\[ N \ ss0 \vdash O \ sss0 \]

\[ (Ax) \]
\[ N \ s ss0 \vdash N \ ss0 \]
\[ (N \ R_2) \]
\[ N \ ss0 \vdash N \ sss0 \]
\[ (=L) \]
\[ y = ss0, N \ ss0 \vdash N \ y \]
\[ (N \ R_2) \]
\[ N \ ss0 \vdash N \ sss0 \]
\[ (=L) \]
\[ y = ss0, N \ ss0 \vdash N \ y \]

\[ (Case \ O) \]
\[ O \ y \vdash N \ y \]
\[ (N \ R_2) \]
\[ O \ y \vdash N \ sy \]
\[ (=L) \]
\[ x = sy, O \ y \vdash N \ x \]

\[ (Case \ E) \]
\[ E \ x \vdash N \ x \]

\[ (Subst) \]
\[ E \ z \vdash N \ z \]
\[ E \ z \vdash N \ sz \]
\[ (=L) \]
\[ y = sz, E \ z \vdash N \ y \]
Extracting Semantic Orderings: Example

\[
\begin{align*}
\Rightarrow & \quad N \ 0 \\
N \ x & \Rightarrow \quad N \ sx \\
\Rightarrow & \quad E \ 0 \\
O \ x & \Rightarrow \quad E \ sx \\
E \ x & \Rightarrow \quad O \ sx \\
N \ ss0 & \Rightarrow \quad O \ sss0 \\
\end{align*}
\]

\[
\begin{align*}
N \ ss0 & \vdash \ N \ ss0 \\
\quad (Ax) \\
N \ ss0 & \vdash \ N \ sss0 \\
\quad (N \ R_2) \\
y & = \ sss0, \ N \ ss0 & \vdash \ N \ y \\
\quad (=L) \\
y & = \ sz, \ E \ z & \vdash \ N \ y \\
\quad (Case \ O) \\
O \ y & \vdash \ N \ y \\
\quad (N \ R_2) \\
O \ y & \vdash \ N \ sy \\
\quad (=L) \\
x & = \ sy, \ O \ y & \vdash \ N \ x \\
\quad (Case \ E) \\
E \ x & \vdash \ N \ x
\end{align*}
\]
Definition (Realizability Condition)

For every maximal right-hand trace, there must exist a left-hand trace following some prefix of the same path such that:

- either the right-hand trace is grounded, or it is partially maximal with the left-hand trace matching in the length and final predicate
- right unfoldings $\leq$ left unfoldings
Soundness of the Realizability Condition

**Theorem**

Suppose $\mathcal{P}$ is a cyclic proof of $P \vdash Q$ satisfying the realizability condition, then $[P]_\alpha \subseteq [Q]_\alpha$, for all $\alpha$ (i.e. $Q \leq P$)

**Proof.**
Theorem

Suppose $\mathcal{P}$ is a cyclic proof of $P \vdash Q \bar{y}$ satisfying the realizability condition, then $[P \bar{x}]_\alpha \subseteq [Q \bar{y}]_\alpha$, for all $\alpha$ (i.e. $Q \bar{y} \leq P \bar{x}$)

Proof.

Pick a model $m \in [P \bar{x}]_\alpha$ (i.e. $\exists \beta \leq \alpha : m \in [P \bar{x}]_\beta$)

- $m$ corresponds to a maximal right-hand trace in $\mathcal{P}$
- Since $\mathcal{P}$ is a proof $P \bar{x} \vdash Q \bar{y}$ is valid, in particular $m \in [Q \bar{y}]$
- The number of unfoldings in this right-hand trace is an upper bound on the least approximation $[Q \bar{y}]_\gamma$ containing $m$
- The number of unfoldings in any left-hand trace following the same path is a lower bound on the least approximation $[P \bar{x}]_\delta$ containing $m$
- From the realizability condition, we have that $\delta \geq \gamma$
**Definition (Weighted Automata)**

Let $\Sigma$ be an alphabet, and $(V, \oplus, \otimes)$ a semiring of weights. A weighted automaton $\mathcal{A}$ is a tuple $(Q, q_I, F, \gamma)$ consisting of a set $Q$ of states containing an initial state $q_I \in Q$, a set $F \subseteq Q$ of final states, and a weighted transition function $\gamma : (Q \times \Sigma \times Q) \rightarrow V$. 

1. The value of a run of $\mathcal{A}$ is the semiring product of all its transitions.
2. The value of a word is the semiring sum of all runs accepting that word.
3. The quantitative language $L_\mathcal{A}$ is the function $\mathbb{V} : \Sigma^* \rightarrow \mathcal{V}$ computed by $\mathcal{A}$.
Weighted Automata

Definition (Weighted Automata)

Let $\Sigma$ be an alphabet, and $(V, \oplus, \otimes)$ a semiring of weights. A weighted automaton $\mathcal{A}$ is a tuple $(Q, q_I, F, \gamma)$ consisting of a set $Q$ of states containing an initial state $q_I \in Q$, a set $F \subseteq Q$ of final states, and a weighted transition function $\gamma : (Q \times \Sigma \times Q) \rightarrow V$.

1. The value of a run of $\mathcal{A}$ is the semiring product of all its transitions
2. The value of a word is the semiring sum of all runs accepting that word
3. The quantitative language $L_{\mathcal{A}}$ is the function $\Sigma^* \rightarrow V$ computed by $\mathcal{A}$
Weighted Automata

Definition (Weighted Automata)

Let $\Sigma$ be an alphabet, and $(V, \oplus, \otimes)$ a semiring of weights. A weighted automaton $A$ is a tuple $(Q, q_I, F, \gamma)$ consisting of a set $Q$ of states containing an initial state $q_I \in Q$, a set $F \subseteq Q$ of final states, and a weighted transition function $\gamma : (Q \times \Sigma \times Q) \rightarrow V$.

1. The value of a run of $A$ is the semiring product of all its transitions
2. The value of a word is the semiring sum of all runs accepting that word
3. The quantitative language $\mathcal{L}_A$ is the function $\Sigma^* \rightarrow V$ computed by $A$

Definition (Weighted Inclusion)

$\mathcal{L}_1 \leq \mathcal{L}_2$ if and only if for every word $w$ such that $\mathcal{L}_1(w)$ is defined, $\mathcal{L}_2(w)$ is also defined and $\mathcal{L}_1(w) \leq \mathcal{L}_2(w)$
**Definition (Weighted Automata)**

Let $\Sigma$ be an alphabet, and $(V, \oplus, \otimes)$ a semiring of weights. A weighted automaton $A$ is a tuple $(Q, q_I, F, \gamma)$ consisting of a set $Q$ of states containing an initial state $q_I \in Q$, a set $F \subseteq Q$ of final states, and a weighted transition function $\gamma : (Q \times \Sigma \times Q) \to V$.

1. The value of a run of $A$ is the semiring product of all its transitions
2. The value of a word is the semiring sum of all runs accepting that word
3. The quantitative language $L_A$ is the function $\Sigma^* \to V$ computed by $A$

**Definition (Weighted Inclusion)**

$L_1 \leq L_2$ if and only if for every word $w$ such that $L_1(w)$ is defined, $L_2(w)$ is also defined and $L_1(w) \leq L_2(w)$

**Sum** automata are weighted automata over $(\mathbb{N}, +, \text{max})
Weighted Automata: Results

**Definition (Weighted Inclusion)**

\[ \mathcal{L}_1 \leq \mathcal{L}_2 \text{ if and only if for every word } w \text{ such that } \mathcal{L}_1(w) \text{ is defined, } \mathcal{L}_2(w) \text{ is also defined and } \mathcal{L}_1(w) \leq \mathcal{L}_2(w) \]

**Theorem**

*Given two quantitative languages (weighted automata) \( \mathcal{L}_1 \) and \( \mathcal{L}_2 \), it is undecidable whether \( \mathcal{L}_1 \leq \mathcal{L}_2 \) (Krob '94, Almagor Et Al. '11)*
<table>
<thead>
<tr>
<th>Definition (Weighted Inclusion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{L}_1 \leq \mathcal{L}_2$ if and only if for every word $w$ such that $\mathcal{L}_1(w)$ is defined, $\mathcal{L}_2(w)$ is also defined and $\mathcal{L}_1(w) \leq \mathcal{L}_2(w)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given two quantitative languages (weighted automata) $\mathcal{L}_1$ and $\mathcal{L}_2$, it is undecidable whether $\mathcal{L}_1 \leq \mathcal{L}_2$ (Krob '94, Almagor Et Al. '11)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A weighted automaton is called <strong>finite-valued</strong> if there exists a bound on the number of distinct values of accepting runs on any given word</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given two finite-valued weighted automata $\mathcal{A}$ and $\mathcal{B}$, it is decidable whether $\mathcal{L}<em>\mathcal{A} \leq \mathcal{L}</em>\mathcal{B}$ (Filiot, Gentilini &amp; Raskin '14)</td>
</tr>
</tbody>
</table>
Given a cyclic entailment proof $\mathcal{P}$, we can construct two kinds of finite-valued sum automata, $\mathcal{A}_\mathcal{P}[n]$ ($n \in \mathbb{N}$) and $\mathcal{C}_\mathcal{P}$, which count the unfoldings in left- and right-hand traces, respectively:

- The words accepted are paths in the proof from the root sequent.
- The value of a path is the maximum number of unfoldings in the traces along the path.
  - $\mathcal{C}_\mathcal{P}$ only counts traces following the full path.
  - The $\mathcal{A}_\mathcal{P}[n]$ count traces following any prefix of the path.
- Each $\mathcal{A}_\mathcal{P}[n]$ considers only a subset of the paths in the proof.
  - A complete automaton can be constructed but is not, in general, finite-valued.
- $\mathcal{C}_\mathcal{P}$ is grounded when all final states correspond to ground predicate instances.
The construction of the weighted automata allows the following result:

**Theorem**

Let $\mathcal{P}$ be a cyclic entailment proof which is *dynamic* and *balanced*; then $\mathcal{P}$ satisfies the realizability condition if and only if $C_\mathcal{P} \leq A_\mathcal{P}[N]$ and $C_\mathcal{P}$ is grounded (where $N$ is a function of $\mathcal{P}$)

- The properties of *balance* and *dynamism* are additional structural properties of the cycles in $\mathcal{P}$ which ensure completeness of the bound $N$
- The bound $N$ is a function of graph-theoretic quantities relating to the cycles in proofs\(^1\)

---

\(^1\)More details in the paper and technical report!
Conclusions

- We have shown that information about inclusions between the semantics of inductive predicates can be extracted from cyclic proofs of entailments.

- This information can be used to construct ranking functions for programs.

- Our results are formulated abstractly, and so hold for any cyclic proof system whose rules satisfy certain properties (e.g., separation logic).

- We use the term realizability because we extract semantic information from the proofs.
Future Work

- Implement the decision procedure within the cyclic proof-based verification framework CYCLIST
- Evaluate to what extent entailments found ‘in the wild’ satisfy the realizability condition
- Extend the results to better handle cuts in proofs
- Investigate further theoretical questions:
  - are there weaker structural properties of proofs that still admit completeness with the approximate automata
  - If the semantic inclusion $\llbracket P \vec{x} \rrbracket_\alpha \subseteq \llbracket Q \vec{y} \rrbracket_\alpha$ holds, is there a cyclic proof of $P \vec{x} \vdash Q \vec{y}$ satisfying the realizability condition?
Suppose we can deduce from a proof of $\Gamma, P \vec{t} \vdash \Sigma, Q \vec{u}$ that $Q \vec{u} \leq P \vec{t}$

Then we can safely form a well-founded trace across the active formula

$\Gamma, P \vec{t} \vdash \Sigma, Q \vec{u} \quad Q \vec{u}, \Pi \vdash \Delta$

$\quad \Gamma, P \vec{t}, \Pi \vdash \Sigma, \Delta$

This is explicitly forbidden in existing cyclic proof systems, precisely because there is no way to ensure in general that there is an inclusion between $[P \vec{t}]_\alpha$ and $[Q \vec{u}]_\alpha$

Thus, our results can be used to bootstrap and enhance cyclic entailment systems themselves