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

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

[2]Department of Computer Science, UCL, London, UK

Motivation: Program Termination

```
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

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

$$\mathrm{list}(x, n) \;\Leftrightarrow\; (n = 0 \wedge x = \mathtt{NULL}) \;\vee\; \mathrm{list}(x\mathtt{->next}, n - 1)$$

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

void shuffle(ll *x)            {
  if ( x != NULL ) {

    ll *y = x -> next;

    rev(y);

    shuffle(y);

  }
}
```

Motivation: Program Termination

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

$\mathrm{list}(x, n) \;\Leftrightarrow\; (n = 0 \land x = \texttt{NULL}) \;\lor\; \mathrm{list}(x\texttt{->next}, n - 1)$

```
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

```
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

```
struct ll { int data; ll *next; }
$\mathrm{list}(x, n) \Leftrightarrow (n = 0 \land x = \texttt{NULL}) \lor \mathrm{list}(x\texttt{->next}, n - 1)$

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

void shuffle(ll *x) $\{\, \mathrm{list}(\texttt{x}, n) \,\}$ {
   if ( x != NULL ) {
     $\{\, \mathrm{list}(\texttt{x->next}, n - 1) \,\}$
     ll *y = x -> next;
     $\{\, \texttt{y} = \texttt{x->next} \land \mathrm{list}(\texttt{y}, n - 1) \,\}$
     rev(y);

     shuffle(y);

   }
} $\{\, \mathrm{list}(\texttt{x}, n) \,\}$
```

Motivation: Program Termination

```
struct ll { int data; ll *next; }
$\mathrm{list}(x, n) \Leftrightarrow (n = 0 \wedge x = \texttt{NULL}) \vee \mathrm{list}(x\texttt{->next}, n - 1)$
void rev(ll *x) $\{\, \mathrm{list}(x, n) \,\}$ { ... } $\{\, \mathrm{list}(x, n) \,\}$
void shuffle(ll *x) $\{\, \mathrm{list}(\mathtt{x}, n) \,\}$ {
   if ( x != NULL ) {
     $\{\, \mathrm{list}(\mathtt{x\text{->}next}, n - 1) \,\}$
     ll *y = x -> next;
     $\{\, \mathtt{y} = \mathtt{x\text{->}next} \wedge \mathrm{list}(\mathtt{y}, n - 1) \,\}$
     rev(y);

     shuffle(y);

   }
} $\{\, \mathrm{list}(\mathtt{x}, n) \,\}$
```

Motivation: Program Termination

```
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

```
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

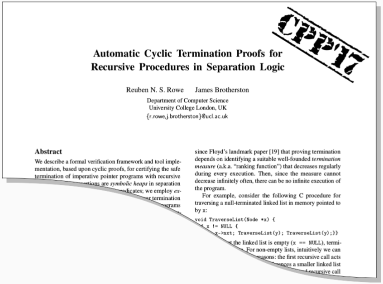
```
struct ll { int data; ll *next; }
```

$$\mathrm{list}(x, n) \Leftrightarrow (n = 0 \wedge x = \texttt{NULL}) \vee \mathrm{list}(x\texttt{->next}, n-1)$$

```
void rev(ll *x) $\{ \mathrm{list}(x, n) \}$ { ... } $\{ \mathrm{list}(x, n) \}$

void shuffle(ll *x) $\{ \mathrm{list}(\texttt{x}, n) \}$ {
   if ( x != NULL ) {
     $\{ \mathrm{list}(\texttt{x->next}, n-1) \}$
     ll *y = x -> next;
     $\{ \texttt{y} = \texttt{x->next} \wedge \mathrm{list}(\texttt{y}, n-1) \}$
     rev(y);
     $\{ \texttt{y} = \texttt{x->next} \wedge \mathrm{list}(\texttt{y}, n-1) \}$
     shuffle(y);
     $\{ \texttt{y} = \texttt{x->next} \wedge \mathrm{list}(\texttt{y}, n-1) \}$
   }
} $\{ \mathrm{list}(\texttt{x}, n) \}$
```

Motivation: Program Termination

```
struct ll { int data; ll *next; }
$\mathsf{list}(x) \Leftrightarrow (n = 0 \wedge x = \texttt{NULL}) \vee \mathsf{list}(x\texttt{->next})$

void rev(ll *x) $\{\,\mathsf{list}_\alpha(x)\,\}$ { ... } $\{\,\mathsf{list}_\alpha(x)\,\}$

void shuffle(ll *x) $\{\,\mathsf{list}_\alpha(\texttt{x})\,\}$ {
   if ( x != NULL ) {
     $\{\,\mathsf{list}_\beta(\texttt{x->next}) \wedge \beta < \alpha\,\}$
     ll *y = x -> next;
     $\{\,\texttt{y} = \texttt{x->next} \wedge \mathsf{list}_\beta(\texttt{y}) \wedge \beta < \alpha\,\}$
     rev(y);
     $\{\,\texttt{y} = \texttt{x->next} \wedge \mathsf{list}_\beta(\texttt{y}) \wedge \beta < \alpha\,\}$
     shuffle(y);
     $\{\,\texttt{y} = \texttt{x->next} \wedge \mathsf{list}_\beta(\texttt{y}) \wedge \beta < \alpha\,\}$
   }
} $\{\,\mathsf{list}_\alpha(\texttt{x})\,\}$
```



CPP'17

Automatic Cyclic Termination Proofs for Recursive Procedures in Separation Logic

Reuben N. S. Rowe James Brotherston

Department of Computer Science
University College London, UK
{r.rowe,j.brotherston}@ucl.ac.uk

Abstract

We describe a formal verification framework and tool implementation, based upon cyclic proofs, for certifying the safe termination of imperative pointer programs with recursive

ertions are *symbolic heaps* in separation

edicates; we employ *ex-*

ur termination

ograms

since Floyd's landmark paper [19] that proving termination depends on identifying a suitable well-founded *termination measure* (a.k.a. "ranking function") that decreases regularly during every execution. Then, since the measure cannot decrease infinitely often, there can be no infinite execution of the program.

For example, consider the following C procedure for traversing a null-terminated linked list in memory pointed to by x:

```
void TraverseList(Node *x) {
 f x != NULL {
    x->nxt; TraverseList(y); TraverseList(y);}}
```

t the linked list is empty (x == NULL), termi-

. For non-empty lists, intuitively we can

easons: the first recursive call acts

ences a smaller linked list

recursive call

Motivation: Program Termination

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

$\mathsf{list}(x) \Leftrightarrow (n = 0 \wedge x = \texttt{NULL}) \vee \mathsf{list}(x\texttt{->next})$

`void rev(ll *x)` $\{\,\mathsf{list}_\alpha(x)\,\}$ `{ ... }` $\{\,\mathsf{list}_\alpha(x)\,\}$

`void shuffle(ll *x)` $\{\,\mathsf{list}_\alpha(x)\,\}$ `{`

`  if ( x`

$\quad\quad\{\,\mathsf{list}_\beta($

`    ll *y`

$\quad\quad\{\,y =$

`    rev(y`

$\quad\quad\{\,y =$

`    shuffle(y);`

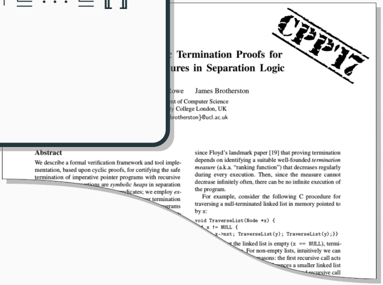
$\quad\quad\{\,y = x\texttt{->next} \wedge \mathsf{list}_\beta(y) \wedge \beta < \alpha\,\}$

`  }`

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

$$[\![\cdot]\!] : \mathsf{Formulas} \to \wp(\mathsf{Models})$$

$$[\![\cdot]\!]_\perp \sqsubseteq [\![\cdot]\!]_1 \sqsubseteq \ldots [\![\cdot]\!]_\alpha \sqsubseteq [\![\cdot]\!]_{\alpha+1} \sqsubseteq \ldots \sqsubseteq [\![\cdot]\!]$$



CPP'17

Termination Proofs for
ures in Separation Logic

owe James Brotherston

nt of Computer Science
y College London, UK
brotherston}@ucl.ac.uk

Abstract

We describe a formal verification framework and tool implementation, based upon cyclic proofs, for certifying the safe termination of imperative pointer programs with recursive … rtions are *symbolic heaps* in separation … edicates; we employ ex- … ur termination … ograms

since Floyd's landmark paper [19] that proving termination depends on identifying a suitable well-founded *termination measure* (a.k.a. "ranking function") that decreases regularly during every execution. Then, since the measure cannot decrease infinitely often, there can be no infinite execution of the program.

For example, consider the following C procedure for traversing a null-terminated linked list in memory pointed to by x:

```
void TraverseList(Node *x) {
 f x != NULL {
   x->nxt; TraverseList(y); TraverseList(y);}}
```

t the linked list is empty (x == NULL), termi- … For non-empty lists, intuitively we can … easons: the first recursive call acts … ences a smaller linked list … recursive call

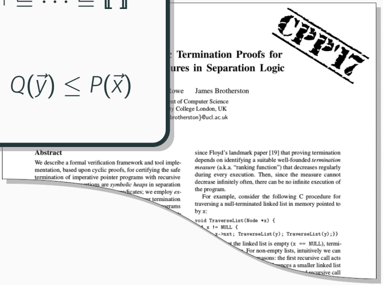
Motivation: Program Termination

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

void rev(ll *x) $\{\, \mathrm{list}_\alpha(x) \,\}$ { ... } $\{\, \mathrm{list}_\alpha(x) \,\}$

void shuffle(ll *x) $\{\, \mathrm{list}_\alpha(x) \,\}$ {
  if ( x
    $\{\, \mathrm{list}_\beta($
    ll *y
    $\{\, y =$
    rev(y
    $\{\, y =$
    shuffle(y);
    $\{\, y = $ x->next $\land\ \mathrm{list}_\beta(y) \land \beta < \alpha \,\}$
  }
} $\{\, \mathrm{list}_\alpha(x) \,\}$
```

$$[\![\cdot]\!] : \texttt{Formulas} \rightarrow \wp(\texttt{Models})$$

$$[\![\cdot]\!]_\perp \sqsubseteq [\![\cdot]\!]_1 \sqsubseteq \ldots [\![\cdot]\!]_\alpha \sqsubseteq [\![\cdot]\!]_{\alpha+1} \sqsubseteq \ldots \sqsubseteq [\![\cdot]\!]$$

$$\forall \alpha \,.\, [\![P(\vec{x})]\!]_\alpha \subseteq [\![Q(\vec{y})]\!]_\alpha \quad \equiv \quad Q(\vec{y}) \leq P(\vec{x})$$



CPP'17

Termination Proofs for
ures in Separation Logic

Rowe James Brotherston
nt of Computer Science
y College London, UK
brotherston}@ucl.ac.uk

Abstract

We describe a formal verification framework and tool implementation, based upon cyclic proofs, for certifying the safe termination of imperative pointer programs with recursive … ertions are *symbolic heaps* in separation … edicates; we employ *ex-* … ur termination … ograms

since Floyd's landmark paper [19] that proving termination depends on identifying a suitable well-founded *termination measure* (a.k.a. "ranking function") that decreases regularly during every execution. Then, since the measure cannot decrease infinitely often, there can be no infinite execution of the program.

For example, consider the following C procedure for traversing a null-terminated linked list in memory pointed to by x:

```
void TraverseList(Node *x) {
 f x != NULL {
   x->nxt; TraverseList(y); TraverseList(y);}}
```

t the linked list is empty (x == NULL), termi- … For non-empty lists, intuitively we can … easons: the first recursive call acts … ences a smaller linked list … recursive call

Motivation: Program Termination

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

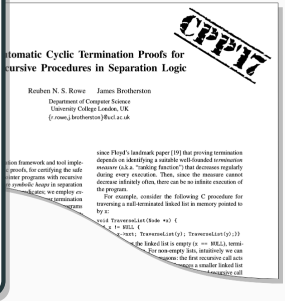
$\mathsf{list}(x) \Leftrightarrow (n = 0 \wedge x = \texttt{NULL}) \vee \mathsf{list}(x\texttt{->next})$

`void rev(ll *x)` $\{\, \mathsf{list}_\alpha(x) \,\}$ `{ ... }` $\{\, \mathsf{list}_\alpha(x) \,\}$

`void shuffle(ll *x)` $\{\, \mathsf{list}_\alpha(x) \,\}$ `{`

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

$$x \neq \texttt{NULL} \wedge y = x\texttt{->next} \wedge \mathsf{list}(y) \vdash \mathsf{list}(x)$$



CPP'17

tomatic Cyclic Termination Proofs for cursive Procedures in Separation Logic

Reuben N. S. Rowe James Brotherston

Department of Computer Science
University College London, UK
{r.rowe,j.brotherston}@ucl.ac.uk

ation framework and tool imple- c proofs, for certifying the safe ointer programs with recursive re *symbolic heaps* in separation edicates; we employ *ex-* ur termination ograms

since Floyd's landmark paper [19] that proving termination depends on identifying a suitable well-founded *termination measure* (a.k.a. "ranking function") that decreases regularly during every execution. Then, since the measure cannot decrease infinitely often, there can be no infinite execution of the program.

For example, consider the following C procedure for traversing a null-terminated linked list in memory pointed to by x:

```
void TraverseList(Node *x) {
 f x != NULL {
    x->nxt; TraverseList(y); TraverseList(y);}}
```

t the linked list is empty (x == NULL), termi- . For non-empty lists, intuitively we can asons: the first recursive call acts nces a smaller linked list recursive call

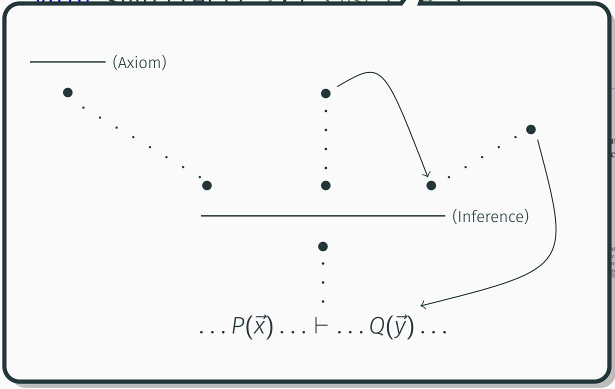
Motivation: Program Termination

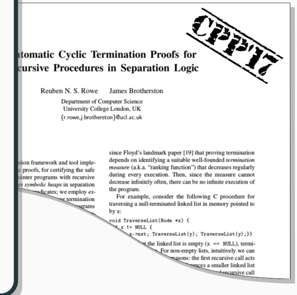
```
struct ll { int data; ll *next; }
```

$\mathsf{list}(x) \;\Leftrightarrow\; (n = 0 \wedge x = \texttt{NULL}) \;\vee\; \mathsf{list}(x\texttt{->next})$

`void rev(ll *x)` $\{\,\mathsf{list}_\alpha(x)\,\}$ `{ ... }` $\{\,\mathsf{list}_\alpha(x)\,\}$

`void shuffle(ll *x)` $\{\,\mathsf{list}_\alpha(x)\,\}$ `{`





CPP'17

tomatic Cyclic Termination Proofs for cursive Procedures in Separation Logic

Reuben N. S. Rowe James Brotherston
Department of Computer Science
University College London, UK
{r.rowe,j.brotherston}@ucl.ac.uk

ation framework and tool implec proofs, for certifying the safe ointer programs with recursive re *symbolic heaps* in separation edicates; we employ *ex*ur termination ograms

since Floyd's landmark paper [19] that proving termination depends on identifying a suitable well-founded *termination measure* (a.k.a. "ranking function") that decreases regularly during every execution. Then, since the measure cannot decrease infinitely often, there can be no infinite execution of the program.

For example, consider the following C procedure for traversing a null-terminated linked list in memory pointed to by x:

```
void TraverseList(Node *x) {
 f x != NULL {
     x->nxt; TraverseList(y); TraverseList(y);}}
```

t the linked list is empty (x == NULL), termi-. For non-empty lists, intuitively we can easons: the first recursive call acts nces a smaller linked list recursive call

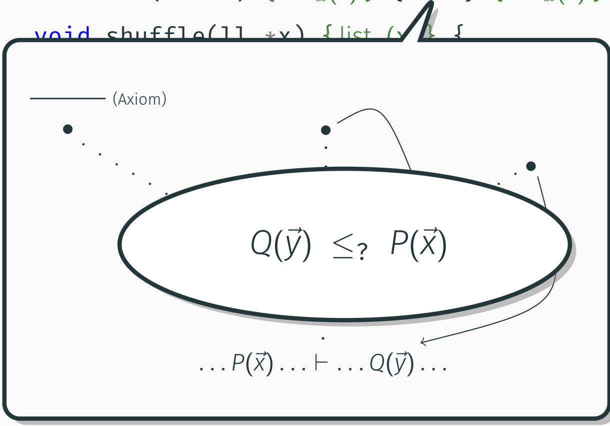
Motivation: Program Termination

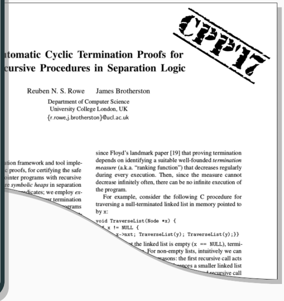
`struct ll { int data; ll *next; }`

$\mathsf{list}(x) \;\Leftrightarrow\; (n = 0 \wedge x = \texttt{NULL}) \;\vee\; \mathsf{list}(x\texttt{->next})$

`void rev(ll *x)` $\{\, \mathsf{list}_\alpha(x) \,\}$ `{ ... }` $\{\, \mathsf{list}_\alpha(x) \,\}$

`void shuffle(ll *x)` $\{\, \mathsf{list}_\alpha(x) \,\}$ `{`


(Axiom)
$Q(\vec{y}) \;\leq_?\; P(\vec{x})$
$\ldots P(\vec{x}) \ldots \vdash \ldots Q(\vec{y}) \ldots$


CPP'17

tomatic Cyclic Termination Proofs for cursive Procedures in Separation Logic

Reuben N. S. Rowe James Brotherston
Department of Computer Science
University College London, UK
{r.rowe,j.brotherston}@ucl.ac.uk

ation framework and tool imple-
c proofs, for certifying the safe
ointer programs with recursive
re *symbolic heaps* in separation
edicates; we employ *ex-*
ur termination
ograms

since Floyd's landmark paper [19] that proving termination depends on identifying a suitable well-founded *termination measure* (a.k.a. "ranking function") that decreases regularly during every execution. Then, since the measure cannot decrease infinitely often, there can be no infinite execution of the program.

For example, consider the following C procedure for traversing a null-terminated linked list in memory pointed to by x:

```
void TraverseList(Node *x) {
 f x != NULL {
   x->nxt; TraverseList(y); TraverseList(y);}}
```

t the linked list is empty (x == NULL), termi-
. For non-empty lists, intuitively we can
easons: the first recursive call acts
ences a smaller linked list
d recursive call

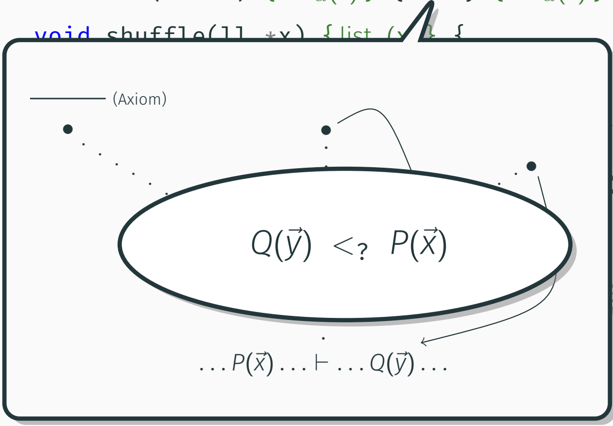
Motivation: Program Termination

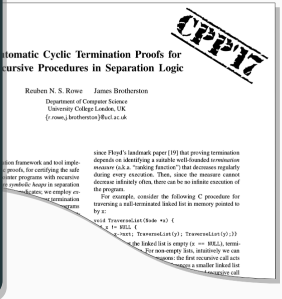
`struct ll { int data; ll *next; }`

$\mathrm{list}(x) \;\Leftrightarrow\; (n = 0 \wedge x = \texttt{NULL}) \;\vee\; \mathrm{list}(x\texttt{->next})$

`void rev(ll *x)` $\{\, \mathrm{list}_\alpha(x) \,\}$ `{ ... }` $\{\, \mathrm{list}_\alpha(x) \,\}$

`void shuffle(ll *x)` $\{\, \mathrm{list}_\alpha(x) \,\}$ `{`


(Axiom)
$Q(\vec{y}) \;<_?\; P(\vec{x})$
$\ldots P(\vec{x}) \ldots \vdash \ldots Q(\vec{y}) \ldots$


tomatic Cyclic Termination Proofs for
cursive Procedures in Separation Logic

CPP'17

Reuben N. S. Rowe James Brotherston
Department of Computer Science
University College London, UK
{r.rowe,j.brotherston}@ucl.ac.uk

tion framework and tool imple-
c proofs, for certifying the safe
ointer programs with recursive
re *symbolic heaps* in separation
edicates; we employ *ex-*
ur termination
ograms

since Floyd's landmark paper [19] that proving termination depends on identifying a suitable well-founded *termination measure* (a.k.a. "ranking function") that decreases regularly during every execution. Then, since the measure cannot decrease infinitely often, there can be no infinite execution of the program.

For example, consider the following C procedure for traversing a null-terminated linked list in memory pointed to by x:

```
void TraverseList(Node *x) {
 x != NULL {
   x->nxt; TraverseList(y); TraverseList(y);}}
```

t the linked list is empty (x == NULL), termi-
. For non-empty lists, intuitively we can
easons: the first recursive call acts
ences a smaller linked list
recursive call

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

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

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

$$\Rightarrow \mathrm{N}\,0$$

$$\mathrm{N}\,x \Rightarrow \mathrm{N}\,\mathrm{s}x$$

$$\Rightarrow \mathrm{E}\,0$$

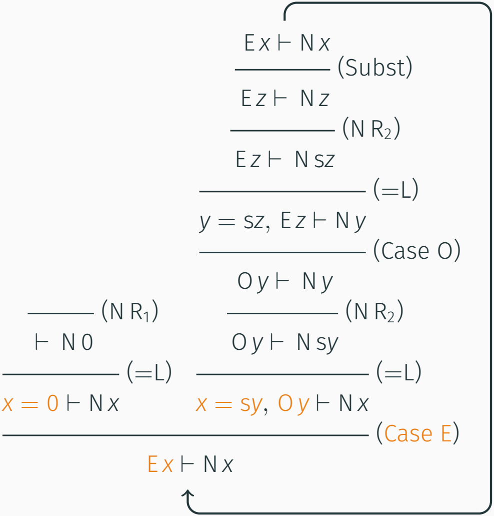
$$\mathrm{O}\,x \Rightarrow \mathrm{E}\,\mathrm{s}x$$

$$\mathrm{E}\,x \Rightarrow \mathrm{O}\,\mathrm{s}x$$

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

A Cyclic Proof in LK Sequent Calculus with Equality

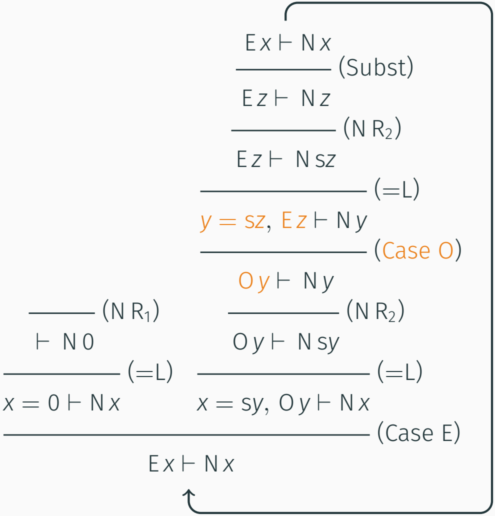
$$\begin{aligned} &\Rightarrow \mathsf{N}\,0 \\ \mathsf{N}\,x &\Rightarrow \mathsf{N}\,\mathsf{s}x \\ &\Rightarrow \mathsf{E}\,0 \\ \mathsf{O}\,x &\Rightarrow \mathsf{E}\,\mathsf{s}x \\ \mathsf{E}\,x &\Rightarrow \mathsf{O}\,\mathsf{s}x \end{aligned}$$



$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}\,0}\,(\mathsf{N}\,\mathsf{R}_1)}{x = 0 \vdash \mathsf{N}\,x}\,(=\mathsf{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{E}\,x \vdash \mathsf{N}\,x}{\mathsf{E}\,z \vdash \mathsf{N}\,z}\,(\text{Subst})}{\mathsf{E}\,z \vdash \mathsf{N}\,\mathsf{s}z}\,(\mathsf{N}\,\mathsf{R}_2)}{y = \mathsf{s}z,\ \mathsf{E}\,z \vdash \mathsf{N}\,y}\,(=\mathsf{L})}{\mathsf{O}\,y \vdash \mathsf{N}\,y}\,(\text{Case O})}{\mathsf{O}\,y \vdash \mathsf{N}\,\mathsf{s}y}\,(\mathsf{N}\,\mathsf{R}_2)}{x = \mathsf{s}y,\ \mathsf{O}\,y \vdash \mathsf{N}\,x}\,(=\mathsf{L})}{\mathsf{E}\,x \vdash \mathsf{N}\,x}\,(\text{Case E})$$

A Cyclic Proof in LK Sequent Calculus with Equality

$$\begin{aligned}
&\Rightarrow \mathsf{N}\,0\\
\mathsf{N}\,x &\Rightarrow \mathsf{N}\,\mathsf{s}x\\
&\Rightarrow \mathsf{E}\,0\\
\mathsf{O}\,x &\Rightarrow \mathsf{E}\,\mathsf{s}x\\
\mathsf{E}\,x &\Rightarrow \mathsf{O}\,\mathsf{s}x
\end{aligned}$$



$$\dfrac{\dfrac{\dfrac{}{\vdash\ \mathsf{N}\,0}\,(\mathsf{N}\,\mathsf{R}_1)}{x = 0 \vdash \mathsf{N}\,x}\,(=\mathsf{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{E}\,x \vdash \mathsf{N}\,x}{\mathsf{E}\,z \vdash\ \mathsf{N}\,z}\,(\text{Subst})}{\mathsf{E}\,z \vdash\ \mathsf{N}\,\mathsf{s}z}\,(\mathsf{N}\,\mathsf{R}_2)}{y = \mathsf{s}z,\ \mathsf{E}\,z \vdash \mathsf{N}\,y}\,(=\mathsf{L})}{\mathsf{O}\,y \vdash\ \mathsf{N}\,y}\,(\text{Case O})}{\mathsf{O}\,y \vdash\ \mathsf{N}\,\mathsf{s}y}\,(\mathsf{N}\,\mathsf{R}_2)}{x = \mathsf{s}y,\ \mathsf{O}\,y \vdash \mathsf{N}\,x}\,(=\mathsf{L})}{\mathsf{E}\,x \vdash \mathsf{N}\,x}\,(\text{Case E})$$

A Cyclic Proof in LK Sequent Calculus with Equality

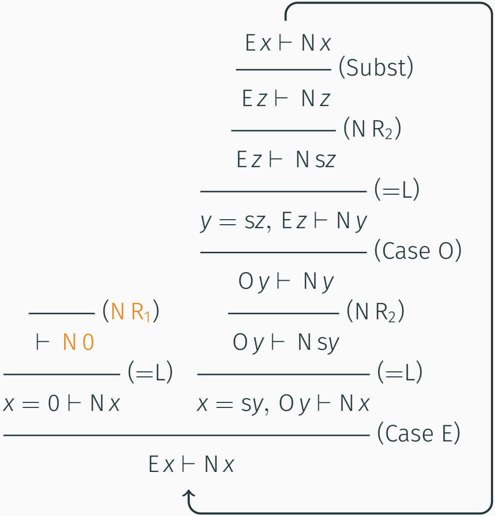
$$\Rightarrow \mathsf{N}\,0$$

$$\mathsf{N}\,x \Rightarrow \mathsf{N}\,\mathsf{s}x$$

$$\Rightarrow \mathsf{E}\,0$$

$$\mathsf{O}\,x \Rightarrow \mathsf{E}\,\mathsf{s}x$$

$$\mathsf{E}\,x \Rightarrow \mathsf{O}\,\mathsf{s}x$$



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

A Cyclic Proof in LK Sequent Calculus with Equality

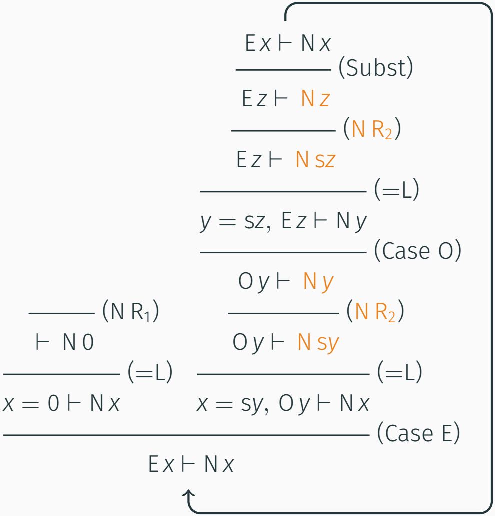
$$\Rightarrow \mathrm{N}\,0$$

$$\mathrm{N}\,x \Rightarrow \mathrm{N}\,\mathsf{s}x$$

$$\Rightarrow \mathrm{E}\,0$$

$$\mathrm{O}\,x \Rightarrow \mathrm{E}\,\mathsf{s}x$$

$$\mathrm{E}\,x \Rightarrow \mathrm{O}\,\mathsf{s}x$$



$\mathrm{E}\,x \vdash \mathrm{N}\,x$

(Subst)

$\mathrm{E}\,z \vdash \mathrm{N}\,z$

$(\mathrm{N}\,\mathrm{R}_2)$

$\mathrm{E}\,z \vdash \mathrm{N}\,\mathsf{s}z$

$(=\mathrm{L})$

$y = \mathsf{s}z,\ \mathrm{E}\,z \vdash \mathrm{N}\,y$

(Case O)

$\mathrm{O}\,y \vdash \mathrm{N}\,y$

$(\mathrm{N}\,\mathrm{R}_2)$

$\mathrm{O}\,y \vdash \mathrm{N}\,\mathsf{s}y$

$(=\mathrm{L})$

$x = \mathsf{s}y,\ \mathrm{O}\,y \vdash \mathrm{N}\,x$

$(\mathrm{N}\,\mathrm{R}_1)$

$\vdash \mathrm{N}\,0$

$(=\mathrm{L})$

$x = 0 \vdash \mathrm{N}\,x$

(Case E)

$\mathrm{E}\,x \vdash \mathrm{N}\,x$

A Cyclic Proof in LK Sequent Calculus with Equality

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

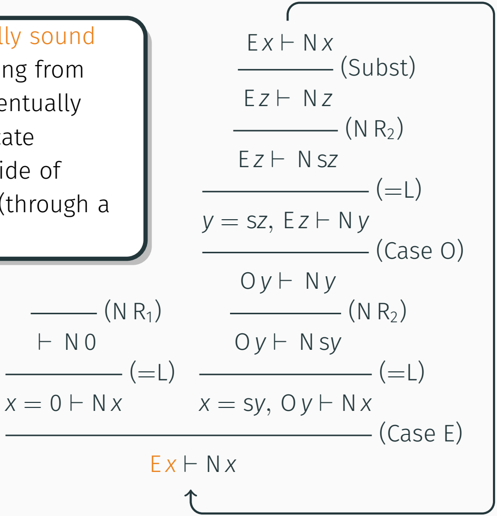
$$\begin{aligned}
&\Rightarrow \mathsf{N}\,0\\
\mathsf{N}\,x &\Rightarrow \mathsf{N}\,\mathsf{s}x\\
&\Rightarrow \mathsf{E}\,0\\
\mathsf{O}\,x &\Rightarrow \mathsf{E}\,\mathsf{s}x\\
\mathsf{E}\,x &\Rightarrow \mathsf{O}\,\mathsf{s}x
\end{aligned}$$

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

A Cyclic Proof in LK Sequent Calculus with Equality

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

$$\begin{aligned} &\Rightarrow \mathsf{N}\,0 \\ \mathsf{N}\,x &\Rightarrow \mathsf{N}\,\mathsf{s}x \\ &\Rightarrow \mathsf{E}\,0 \\ \mathsf{O}\,x &\Rightarrow \mathsf{E}\,\mathsf{s}x \\ \mathsf{E}\,x &\Rightarrow \mathsf{O}\,\mathsf{s}x \end{aligned}$$

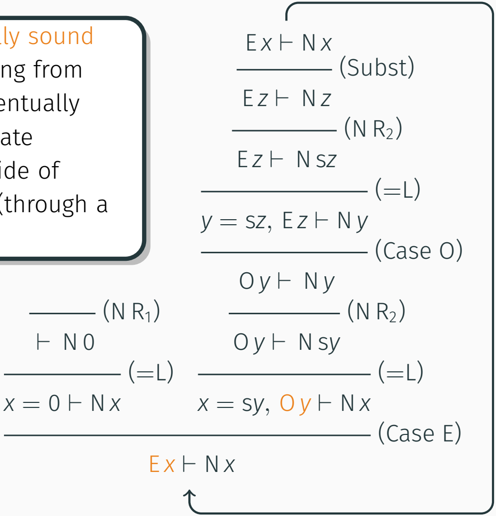


$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}\,0}\,(\mathsf{N}\,\mathrm{R}_1)}{x = 0 \vdash \mathsf{N}\,x}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{E}\,x \vdash \mathsf{N}\,x}{\mathsf{E}\,z \vdash \mathsf{N}\,z}\,(\mathrm{Subst})}{\mathsf{E}\,z \vdash \mathsf{N}\,\mathsf{s}z}\,(\mathsf{N}\,\mathrm{R}_2)}{y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

A Cyclic Proof in LK Sequent Calculus with Equality

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

$$
\begin{aligned}
&\Rightarrow \mathrm{N}\,0\\
\mathrm{N}\,x &\Rightarrow \mathrm{N}\,\mathrm{s}x\\
&\Rightarrow \mathrm{E}\,0\\
\mathrm{O}\,x &\Rightarrow \mathrm{E}\,\mathrm{s}x\\
\mathrm{E}\,x &\Rightarrow \mathrm{O}\,\mathrm{s}x
\end{aligned}
$$

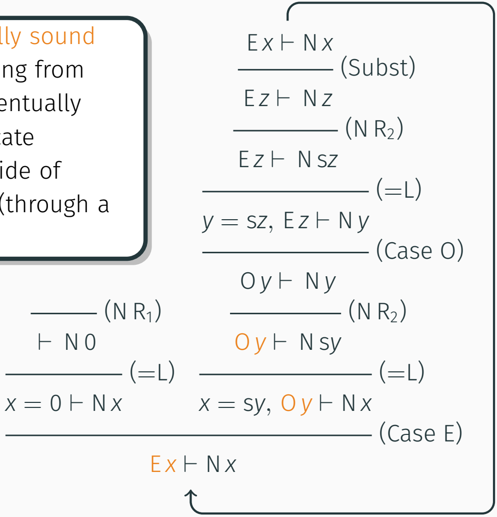


$$
\dfrac{
\dfrac{\dfrac{}{\vdash \mathrm{N}\,0}\,(\mathrm{N\,R}_1)}{x = 0 \vdash \mathrm{N}\,x}\,(=\mathrm{L})
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{E}\,x \vdash \mathrm{N}\,x}{\mathrm{E}\,z \vdash \mathrm{N}\,z}\,(\mathrm{Subst})}{\mathrm{E}\,z \vdash \mathrm{N}\,\mathrm{s}z}\,(\mathrm{N\,R}_2)}{y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

A Cyclic Proof in LK Sequent Calculus with Equality

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

$$\begin{aligned} &\Rightarrow \mathrm{N}\,0 \\ \mathrm{N}\,x &\Rightarrow \mathrm{N}\,\mathrm{s}x \\ &\Rightarrow \mathrm{E}\,0 \\ \mathrm{O}\,x &\Rightarrow \mathrm{E}\,\mathrm{s}x \\ \mathrm{E}\,x &\Rightarrow \mathrm{O}\,\mathrm{s}x \end{aligned}$$



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

A Cyclic Proof in LK Sequent Calculus with Equality

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

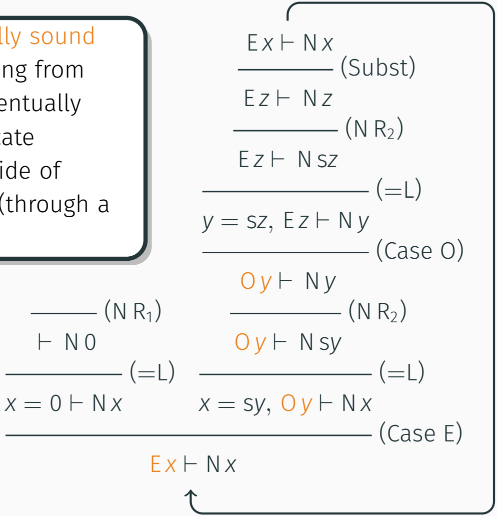
$$\Rightarrow N\,0$$

$$N\,x \Rightarrow N\,sx$$

$$\Rightarrow E\,0$$

$$O\,x \Rightarrow E\,sx$$

$$E\,x \Rightarrow O\,sx$$

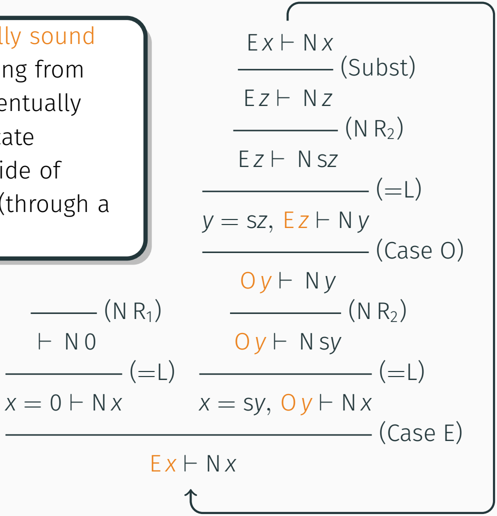


$$\dfrac{\dfrac{\dfrac{}{\vdash N\,0}\,(N\,R_1)}{x = 0 \vdash N\,x}\,(=L) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{E\,x \vdash N\,x}{E\,z \vdash N\,z}\,(\text{Subst})}{E\,z \vdash N\,sz}\,(N\,R_2)}{y = sz,\ E\,z \vdash N\,y}\,(=L)}{O\,y \vdash N\,y}\,(\text{Case O})}{O\,y \vdash N\,sy}\,(N\,R_2)}{x = sy,\ O\,y \vdash N\,x}\,(=L)}{E\,x \vdash N\,x}\,(\text{Case E})$$

A Cyclic Proof in LK Sequent Calculus with Equality

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

$$\begin{array}{rcl} & \Rightarrow & \mathsf{N}\,0 \\ \mathsf{N}\,x & \Rightarrow & \mathsf{N}\,\mathsf{s}x \\ & \Rightarrow & \mathsf{E}\,0 \\ \mathsf{O}\,x & \Rightarrow & \mathsf{E}\,\mathsf{s}x \\ \mathsf{E}\,x & \Rightarrow & \mathsf{O}\,\mathsf{s}x \end{array}$$



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

A Cyclic Proof in LK Sequent Calculus with Equality

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

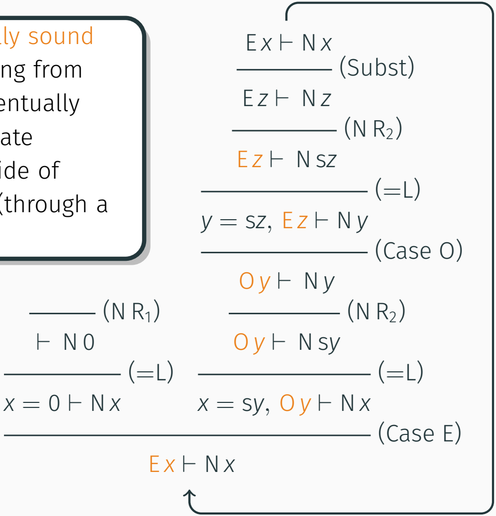
$$\Rightarrow \mathsf{N}\,0$$

$$\mathsf{N}\,x \Rightarrow \mathsf{N}\,sx$$

$$\Rightarrow \mathsf{E}\,0$$

$$\mathsf{O}\,x \Rightarrow \mathsf{E}\,sx$$

$$\mathsf{E}\,x \Rightarrow \mathsf{O}\,sx$$

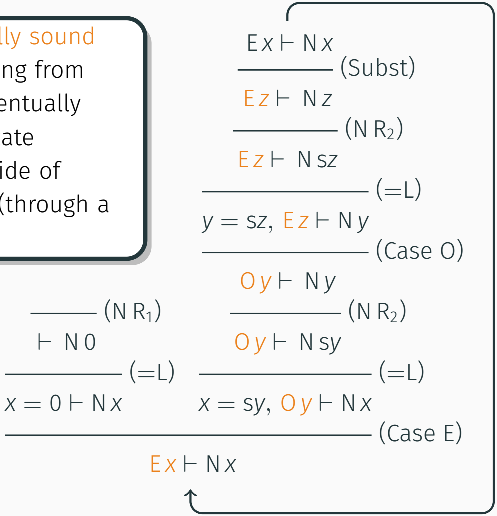


$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}\,0}\,(\mathsf{N}\,R_1)}{x = 0 \vdash \mathsf{N}\,x}\,(=\!L) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{E}\,x \vdash \mathsf{N}\,x}{\mathsf{E}\,z \vdash \mathsf{N}\,z}\,(\text{Subst})}{\mathsf{E}\,z \vdash \mathsf{N}\,sz}\,(\mathsf{N}\,R_2)}{y = sz,\ \mathsf{E}\,z \vdash \mathsf{N}\,y}\,(=\!L)}{\mathsf{O}\,y \vdash \mathsf{N}\,y}\,(\text{Case O})}{\mathsf{O}\,y \vdash \mathsf{N}\,sy}\,(\mathsf{N}\,R_2)}{x = sy,\ \mathsf{O}\,y \vdash \mathsf{N}\,x}\,(=\!L)}{\mathsf{E}\,x \vdash \mathsf{N}\,x}\,(\text{Case E})$$

A Cyclic Proof in LK Sequent Calculus with Equality

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

$$\begin{aligned} &\Rightarrow \mathrm{N}\,0 \\ \mathrm{N}\,x &\Rightarrow \mathrm{N}\,sx \\ &\Rightarrow \mathrm{E}\,0 \\ \mathrm{O}\,x &\Rightarrow \mathrm{E}\,sx \\ \mathrm{E}\,x &\Rightarrow \mathrm{O}\,sx \end{aligned}$$

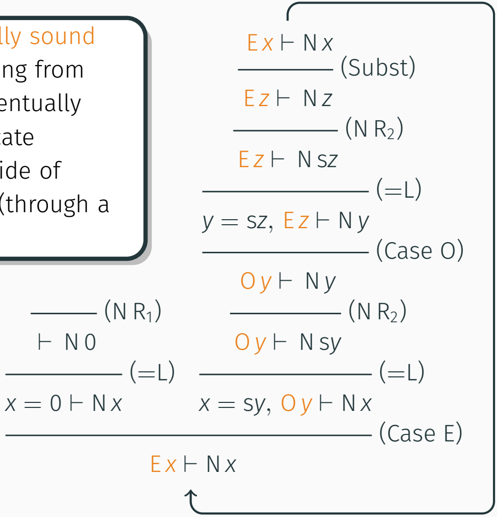


$$\dfrac{\dfrac{\dfrac{}{\vdash \mathrm{N}\,0}\,(\mathrm{N}\,\mathrm{R}_1)}{x = 0 \vdash \mathrm{N}\,x}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{E}\,x \vdash \mathrm{N}\,x}{\mathrm{E}\,z \vdash \mathrm{N}\,z}\,(\mathrm{Subst})}{\mathrm{E}\,z \vdash \mathrm{N}\,sz}\,(\mathrm{N}\,\mathrm{R}_2)}{y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

A Cyclic Proof in LK Sequent Calculus with Equality

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

$$\begin{aligned} &\Rightarrow \mathrm{N}\,0 \\ \mathrm{N}\,x &\Rightarrow \mathrm{N}\,sx \\ &\Rightarrow \mathrm{E}\,0 \\ \mathrm{O}\,x &\Rightarrow \mathrm{E}\,sx \\ \mathrm{E}\,x &\Rightarrow \mathrm{O}\,sx \end{aligned}$$

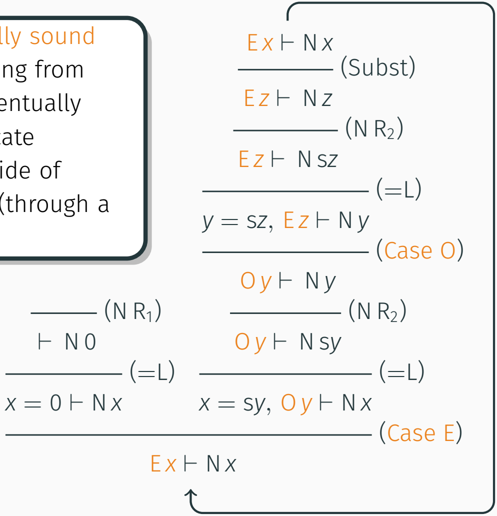


$$\dfrac{\dfrac{\dfrac{}{\vdash \mathrm{N}\,0}\,(\mathrm{N}\,\mathrm{R}_1)}{x = 0 \vdash \mathrm{N}\,x}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathrm{E}\,x \vdash \mathrm{N}\,x}{\mathrm{E}\,z \vdash \mathrm{N}\,z}\,(\mathrm{Subst})}{\mathrm{E}\,z \vdash \mathrm{N}\,sz}\,(\mathrm{N}\,\mathrm{R}_2)}{y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

A Cyclic Proof in LK Sequent Calculus with Equality

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

$$\begin{aligned} &\Rightarrow \mathrm{N}\,0 \\ \mathrm{N}\,x &\Rightarrow \mathrm{N}\,sx \\ &\Rightarrow \mathrm{E}\,0 \\ \mathrm{O}\,x &\Rightarrow \mathrm{E}\,sx \\ \mathrm{E}\,x &\Rightarrow \mathrm{O}\,sx \end{aligned}$$



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

Inductive Predicate Definitions and their Semantics

Definition (Inductive Definition Set)

An *inductive definition set* contains productions $\mathsf{P}_1\,\vec{t_1}, \ldots, \mathsf{P}_j\,\vec{t_j} \Rightarrow \mathsf{P}_0\,\vec{t_0}$

Definition (Characteristic Operators)

Inductive definition sets Φ induce *characteristic operators* φ_Φ on predicate interpretations X (functions from predicate formulas to sets of models):

$$\varphi_\Phi(X)(\mathsf{P}\,\vec{t}\theta) = \{m \mid \mathsf{P}_1\,\vec{t_1}, \ldots, \mathsf{P}_j\,\vec{t_j} \Rightarrow \mathsf{P}\,\vec{t} \in \Phi,\ m \in X(\mathsf{P}_i\,\vec{t_i}\theta) \text{ for all } 1 \leq i \leq j\}$$

Inductive Predicate Definitions and their Semantics

Definition (Inductive Definition Set)

An *inductive definition set* contains productions $\mathrm{P}_1\,\vec{t_1}, \ldots, \mathrm{P}_j\,\vec{t_j} \Rightarrow \mathrm{P}_0\,\vec{t_0}$

Definition (Characteristic Operators)

Inductive definition sets Φ induce *characteristic operators* φ_Φ on predicate interpretations X (functions from predicate formulas to sets of models):

$$\varphi_\Phi(X)(\mathrm{P}\,\vec{t}\theta) = \{m \mid \mathrm{P}_1\,\vec{t_1}, \ldots, \mathrm{P}_j\,\vec{t_j} \Rightarrow \mathrm{P}\,\vec{t} \in \Phi,\; m \in X(\mathrm{P}_i\,\vec{t_i}\theta) \text{ for all } 1 \leq i \leq j\}$$

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

Inductive Predicate Definitions and their Semantics

Definition (Inductive Definition Set)

An *inductive definition set* contains productions $\mathrm{P}_1\, \vec{t_1}, \ldots, \mathrm{P}_j\, \vec{t_j} \Rightarrow \mathrm{P}_0\, \vec{t_0}$

Definition (Characteristic Operators)

Inductive definition sets Φ induce *characteristic operators* φ_Φ on predicate interpretations X (functions from predicate formulas to sets of models):

$$\varphi_\Phi(X)(\mathrm{P}\, \vec{t}\theta) = \{m \mid \mathrm{P}_1\, \vec{t_1}, \ldots, \mathrm{P}_j\, \vec{t_j} \Rightarrow \mathrm{P}\, \vec{t} \in \Phi,\ m \in X(\mathrm{P}_i\, \vec{t_i}\theta) \text{ for all } 1 \leq i \leq j\}$$

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

Characteristic operators φ_Φ are monotone wrt $\sqsubseteq$

Inductive Predicate Definitions and their Semantics

Definition (Inductive Definition Set)

An *inductive definition set* contains productions $\mathsf{P}_1\,\vec{t_1},\ldots,\mathsf{P}_j\,\vec{t_j} \Rightarrow \mathsf{P}_0\,\vec{t_0}$

Definition (Characteristic Operators)

Inductive definition sets Φ induce *characteristic operators* φ_Φ on predicate interpretations X (functions from predicate formulas to sets of models):

$$\varphi_\Phi(X)(\mathsf{P}\,\vec{t}\theta) = \{m \mid \mathsf{P}_1\,\vec{t_1},\ldots,\mathsf{P}_j\,\vec{t_j} \Rightarrow \mathsf{P}\,\vec{t} \in \Phi,\ m \in X(\mathsf{P}_i\,\vec{t_i}\theta) \text{ for all } 1 \leq i \leq j\}$$

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

Characteristic operators φ_Φ are monotone wrt $\sqsubseteq$

We interpret predicates using the least fixed point, $[\![\cdot]\!]_\Phi \overset{def}{=} \mu X.\varphi_\Phi(X)$

$$X_\perp \sqsubseteq \varphi_\Phi(X_\perp) \sqsubseteq \varphi_\Phi(\varphi_\Phi(X_\perp)) \sqsubseteq \ldots \sqsubseteq \varphi_\Phi^\alpha(X_\perp) \sqsubseteq \ldots \sqsubseteq \mu X.\varphi_\Phi(X)$$

Inductive Predicate Definitions and their Semantics

Definition (Inductive Definition Set)

An *inductive definition set* contains productions $\mathrm{P}_1\,\vec{t_1}, \ldots, \mathrm{P}_j\,\vec{t_j} \Rightarrow \mathrm{P}_0\,\vec{t_0}$

Definition (Characteristic Operators)

Inductive definition sets Φ induce *characteristic operators* φ_Φ on predicate interpretations X (functions from predicate formulas to sets of models):

$$\varphi_\Phi(X)(\mathrm{P}\,\vec{t}\theta) = \{m \mid \mathrm{P}_1\,\vec{t_1}, \ldots, \mathrm{P}_j\,\vec{t_j} \Rightarrow \mathrm{P}\,\vec{t} \in \Phi,\ m \in X(\mathrm{P}_i\,\vec{t_i}\theta) \text{ for all } 1 \leq i \leq j\}$$

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

Characteristic operators φ_Φ are monotone wrt $\sqsubseteq$

We interpret predicates using the least fixed point, $[\![\cdot]\!]_\Phi \overset{def}{=} \mu X.\varphi_\Phi(X)$

$$[\![\cdot]\!]_0^\Phi \sqsubseteq [\![\cdot]\!]_1^\Phi \sqsubseteq [\![\cdot]\!]_2^\Phi \sqsubseteq \ldots \sqsubseteq [\![\cdot]\!]_\alpha^\Phi \sqsubseteq \ldots [\![\cdot]\!]^\Phi$$

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

Cyclic Proof Formalises Infinite Descent

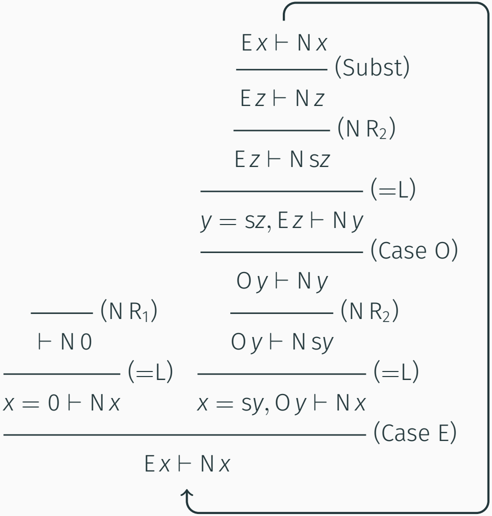
- Suppose, for contradiction, that the conclusion of the proof is not valid
    - That is, there is a counter-model of the sequent
- By local soundness of the inference rules, we obtain an infinite sequence of counter-models for some infinite path in the proof
    - Each model can be mapped to an ever smaller approximation $[\![\mathsf{P}\,\vec{t}\,]\!]^{\Phi}_{\alpha}$ in which it appears
    - These strictly decrease over a case-split

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 $[\![\mathrm{P}\,\vec{t}\,]\!]^{\Phi}_{\alpha}$ 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

$$\begin{aligned} &\Rightarrow N\,0 \\ N\,x &\Rightarrow N\,sx \\ &\Rightarrow E\,0 \\ O\,x &\Rightarrow E\,sx \\ E\,x &\Rightarrow O\,sx \end{aligned}$$



$$\cfrac{\cfrac{\cfrac{}{\vdash N\,0}\,(N\,R_1)}{x = 0 \vdash N\,x}\,(=L) \qquad \cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{E\,x \vdash N\,x}{E\,z \vdash N\,z}\,(\text{Subst})}{E\,z \vdash N\,sz}\,(N\,R_2)}{y = sz, E\,z \vdash N\,y}\,(=L)}{O\,y \vdash N\,y}\,(\text{Case O})}{O\,y \vdash N\,sy}\,(N\,R_2)}{x = sy, O\,y \vdash N\,x}\,(=L)}{E\,x \vdash N\,x}\,(\text{Case E})$$

Extracting Semantic Orderings from Cyclic Proofs

The inductive definitions/semantics give immediately, e.g.

$$\forall m, \alpha : m \in [\![\mathsf{E}\, sx]\!]_\alpha \Rightarrow m \in [\![\mathsf{O}\, x]\!]_\alpha$$

and even

$$\forall m, \alpha : m \in [\![\mathsf{E}\, sx]\!]_\alpha \Rightarrow \exists \beta < \alpha . m \in [\![\mathsf{O}\, x]\!]_\beta$$

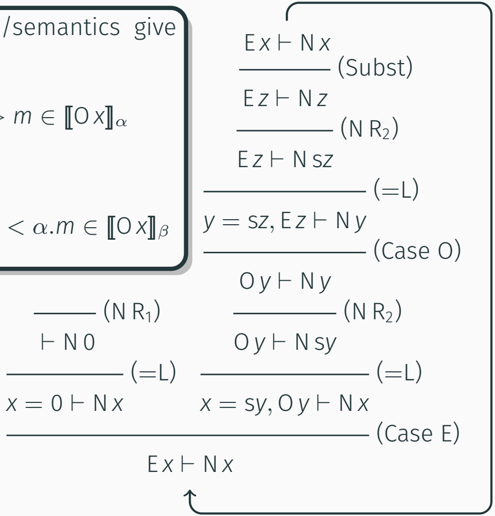
$\Rightarrow \mathsf{N}\,0$

$\mathsf{N}\,x \Rightarrow \mathsf{N}\,sx$

$\Rightarrow \mathsf{E}\,0$

$\mathsf{O}\,x \Rightarrow \mathsf{E}\,sx$

$\mathsf{E}\,x \Rightarrow \mathsf{O}\,sx$



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

Extracting Semantic Orderings from Cyclic Proofs

The global soundness already gives

$$\forall m : m \in [\![\mathsf{E}\,x]\!] \Rightarrow m \in [\![\mathsf{N}\,x]\!]$$

but we would also like to know whether

$$\forall m, \alpha : m \in [\![\mathsf{E}\,x]\!]_\alpha \Rightarrow m \in [\![\mathsf{N}\,x]\!]_\alpha$$

i.e. $\mathsf{N}\,x \leq \mathsf{E}\,x$

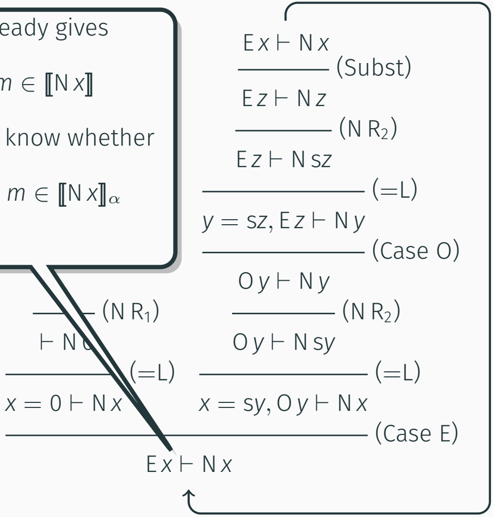
$$\Rightarrow \mathsf{N}\,0$$

$$\mathsf{N}\,x \Rightarrow \mathsf{N}\,\mathsf{s}x$$

$$\Rightarrow \mathsf{E}\,0$$

$$\mathsf{O}\,x \Rightarrow \mathsf{E}\,\mathsf{s}x$$

$$\mathsf{E}\,x \Rightarrow \mathsf{O}\,\mathsf{s}x$$



$\mathsf{E}\,x \vdash \mathsf{N}\,x$

(Subst)

$\mathsf{E}\,z \vdash \mathsf{N}\,z$

$(\mathsf{N}\,\mathsf{R}_2)$

$\mathsf{E}\,z \vdash \mathsf{N}\,\mathsf{s}z$

$(=\mathsf{L})$

$y = \mathsf{s}z, \mathsf{E}\,z \vdash \mathsf{N}\,y$

(Case O)

$\mathsf{O}\,y \vdash \mathsf{N}\,y$

$(\mathsf{N}\,\mathsf{R}_1)$

$(\mathsf{N}\,\mathsf{R}_2)$

$\vdash \mathsf{N}\,0$

$\mathsf{O}\,y \vdash \mathsf{N}\,\mathsf{s}y$

$(=\mathsf{L})$

$(=\mathsf{L})$

$x = 0 \vdash \mathsf{N}\,x$

$x = \mathsf{s}y, \mathsf{O}\,y \vdash \mathsf{N}\,x$

(Case E)

$\mathsf{E}\,x \vdash \mathsf{N}\,x$

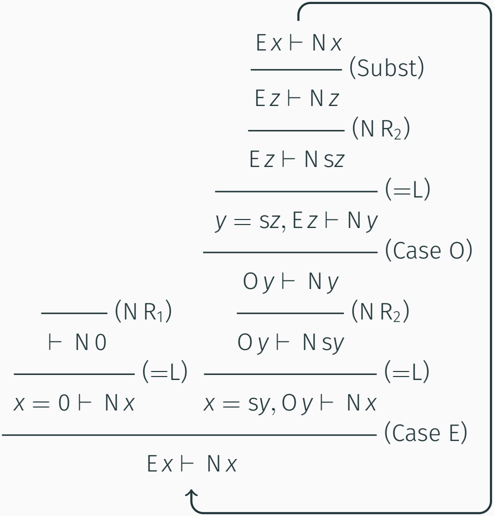
Extracting Semantic Orderings: Basic Ideas

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

$$\begin{aligned} &\Rightarrow \mathsf{N}\,0 \\ \mathsf{N}\,x &\Rightarrow \mathsf{N}\,\mathsf{s}x \\ &\Rightarrow \mathsf{E}\,0 \\ \mathsf{O}\,x &\Rightarrow \mathsf{E}\,\mathsf{s}x \\ \mathsf{E}\,x &\Rightarrow \mathsf{O}\,\mathsf{s}x \end{aligned}$$



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

Extracting Semantic Orderings: Example

$\Rightarrow \mathsf{N}\,0$

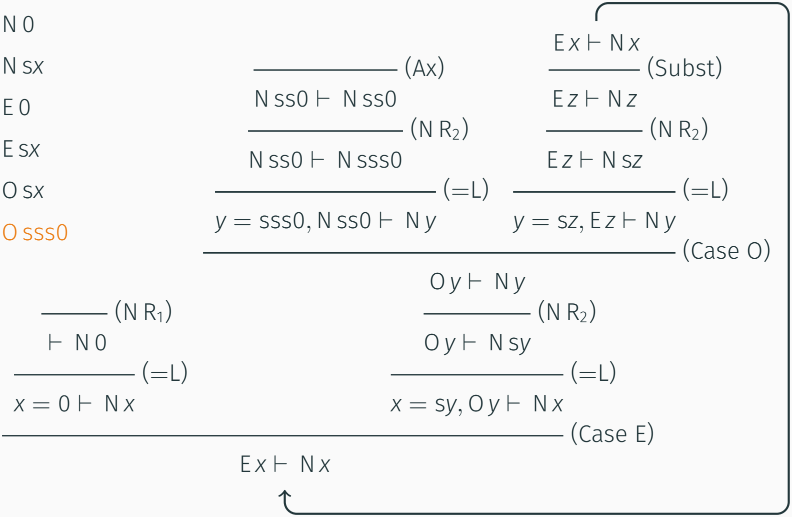
$\mathsf{N}\,x \Rightarrow \mathsf{N}\,sx$

$\Rightarrow \mathsf{E}\,0$

$\mathsf{O}\,x \Rightarrow \mathsf{E}\,sx$

$\mathsf{E}\,x \Rightarrow \mathsf{O}\,sx$

$\mathsf{N}\,ss0 \Rightarrow \mathsf{O}\,sss0$



$\mathsf{E}\,x \vdash \mathsf{N}\,x$

(Ax)

(Subst)

$\mathsf{N}\,ss0 \vdash \mathsf{N}\,ss0$

$\mathsf{E}\,z \vdash \mathsf{N}\,z$

(N R$_2$)

(N R$_2$)

$\mathsf{N}\,ss0 \vdash \mathsf{N}\,sss0$

$\mathsf{E}\,z \vdash \mathsf{N}\,sz$

(=L)

(=L)

$y = sss0, \mathsf{N}\,ss0 \vdash \mathsf{N}\,y$

$y = sz, \mathsf{E}\,z \vdash \mathsf{N}\,y$

(Case O)

$\mathsf{O}\,y \vdash \mathsf{N}\,y$

(N R$_1$)

(N R$_2$)

$\vdash \mathsf{N}\,0$

$\mathsf{O}\,y \vdash \mathsf{N}\,sy$

(=L)

(=L)

$x = 0 \vdash \mathsf{N}\,x$

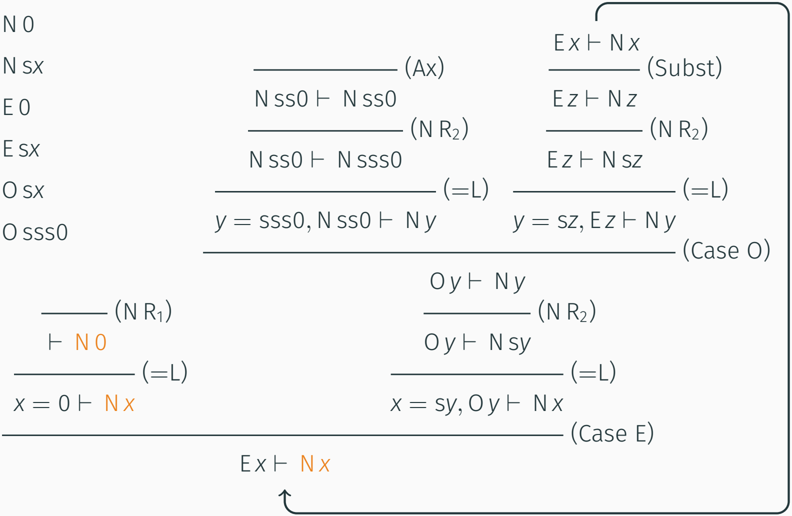
$x = sy, \mathsf{O}\,y \vdash \mathsf{N}\,x$

(Case E)

$\mathsf{E}\,x \vdash \mathsf{N}\,x$

Extracting Semantic Orderings: Example

$$
\begin{aligned}
&\Rightarrow \mathsf{N}\,0\\
\mathsf{N}\,x &\Rightarrow \mathsf{N}\,sx\\
&\Rightarrow \mathsf{E}\,0\\
\mathsf{O}\,x &\Rightarrow \mathsf{E}\,sx\\
\mathsf{E}\,x &\Rightarrow \mathsf{O}\,sx\\
\mathsf{N}\,ss0 &\Rightarrow \mathsf{O}\,sss0
\end{aligned}
$$



$$\dfrac{}{\mathsf{N}\,ss0 \vdash \mathsf{N}\,ss0}\,(\mathrm{Ax})$$

$$\dfrac{\mathsf{N}\,ss0 \vdash \mathsf{N}\,ss0}{\mathsf{N}\,ss0 \vdash \mathsf{N}\,sss0}\,(\mathsf{N}\,\mathrm{R}_2)$$

$$\dfrac{\mathsf{N}\,ss0 \vdash \mathsf{N}\,sss0}{y = sss0, \mathsf{N}\,ss0 \vdash \mathsf{N}\,y}\,(=\mathrm{L})$$

$$\dfrac{\mathsf{E}\,x \vdash \mathsf{N}\,x}{\mathsf{E}\,z \vdash \mathsf{N}\,z}\,(\mathrm{Subst})$$

$$\dfrac{\mathsf{E}\,z \vdash \mathsf{N}\,z}{\mathsf{E}\,z \vdash \mathsf{N}\,sz}\,(\mathsf{N}\,\mathrm{R}_2)$$

$$\dfrac{\mathsf{E}\,z \vdash \mathsf{N}\,sz}{y = sz, \mathsf{E}\,z \vdash \mathsf{N}\,y}\,(=\mathrm{L})$$

$$\dfrac{y = sss0, \mathsf{N}\,ss0 \vdash \mathsf{N}\,y \qquad y = sz, \mathsf{E}\,z \vdash \mathsf{N}\,y}{\mathsf{O}\,y \vdash \mathsf{N}\,y}\,(\mathrm{Case\ O})$$

$$\dfrac{\mathsf{O}\,y \vdash \mathsf{N}\,y}{\mathsf{O}\,y \vdash \mathsf{N}\,sy}\,(\mathsf{N}\,\mathrm{R}_2)$$

$$\dfrac{\mathsf{O}\,y \vdash \mathsf{N}\,sy}{x = sy, \mathsf{O}\,y \vdash \mathsf{N}\,x}\,(=\mathrm{L})$$

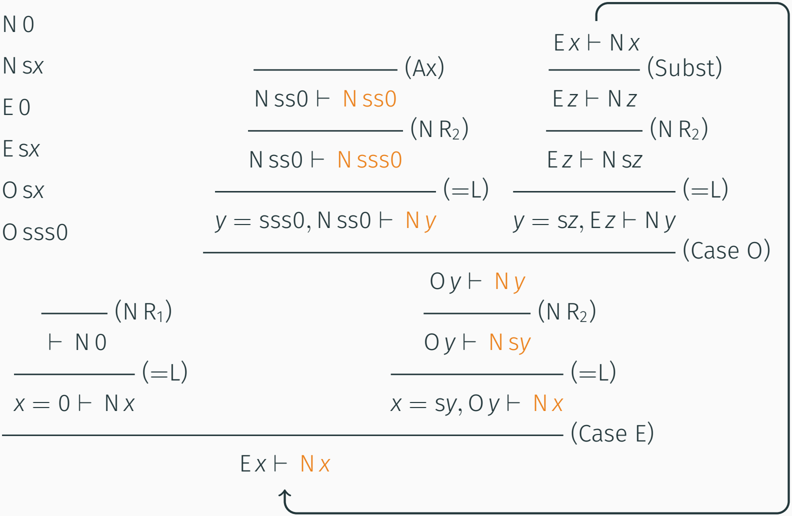
$$\dfrac{}{\vdash \mathsf{N}\,0}\,(\mathsf{N}\,\mathrm{R}_1)$$

$$\dfrac{\vdash \mathsf{N}\,0}{x = 0 \vdash \mathsf{N}\,x}\,(=\mathrm{L})$$

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

Extracting Semantic Orderings: Example

$$\begin{aligned}
&\Rightarrow \mathsf{N}\,0\\
\mathsf{N}\,x &\Rightarrow \mathsf{N}\,\mathsf{s}x\\
&\Rightarrow \mathsf{E}\,0\\
\mathsf{O}\,x &\Rightarrow \mathsf{E}\,\mathsf{s}x\\
\mathsf{E}\,x &\Rightarrow \mathsf{O}\,\mathsf{s}x\\
\mathsf{N}\,\mathsf{ss}0 &\Rightarrow \mathsf{O}\,\mathsf{sss}0
\end{aligned}$$



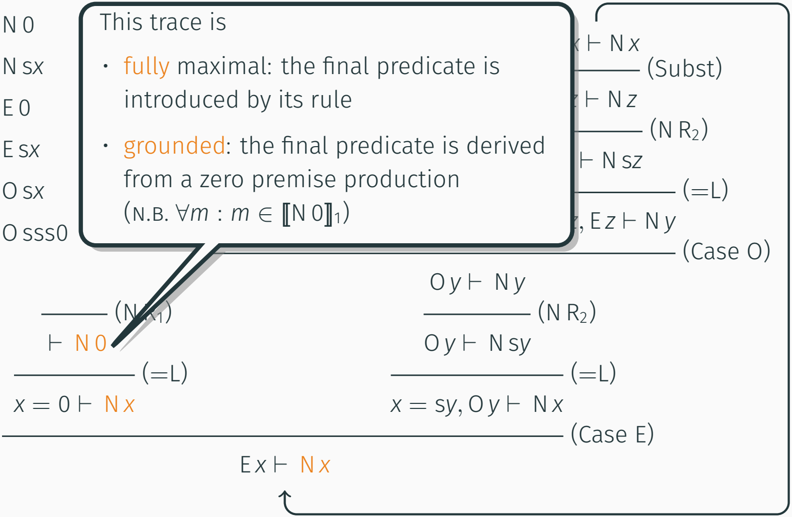
$$\dfrac{\dfrac{\dfrac{}{\vdash\ \mathsf{N}\,0}\,(\mathsf{N}\,\mathsf{R}_1)}{x = 0 \vdash\ \mathsf{N}\,x}\,(=\mathrm{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{N}\,\mathsf{ss}0 \vdash\ \mathsf{N}\,\mathsf{ss}0}\,(\mathrm{Ax})}{\mathsf{N}\,\mathsf{ss}0 \vdash\ \mathsf{N}\,\mathsf{sss}0}\,(\mathsf{N}\,\mathsf{R}_2)}{y = \mathsf{sss}0, \mathsf{N}\,\mathsf{ss}0 \vdash\ \mathsf{N}\,y}\,(=\mathrm{L}) \quad \dfrac{\dfrac{\dfrac{\mathsf{E}\,x \vdash \mathsf{N}\,x}{\mathsf{E}\,z \vdash \mathsf{N}\,z}\,(\mathrm{Subst})}{\mathsf{E}\,z \vdash \mathsf{N}\,\mathsf{s}z}\,(\mathsf{N}\,\mathsf{R}_2)}{y = \mathsf{s}z, \mathsf{E}\,z \vdash \mathsf{N}\,y}\,(=\mathrm{L})}{\mathsf{O}\,y \vdash\ \mathsf{N}\,y}\,(\text{Case O})}{\mathsf{O}\,y \vdash\ \mathsf{N}\,\mathsf{s}y}\,(\mathsf{N}\,\mathsf{R}_2)}{x = \mathsf{s}y, \mathsf{O}\,y \vdash\ \mathsf{N}\,x}\,(=\mathrm{L})}{\mathsf{E}\,x \vdash\ \mathsf{N}\,x}\,(\text{Case E})$$

Extracting Semantic Orderings: Example

$$\Rightarrow \mathrm{N}\,0$$
$$\mathrm{N}\,x \Rightarrow \mathrm{N}\,\mathrm{s}x$$
$$\Rightarrow \mathrm{E}\,0$$
$$\mathrm{O}\,x \Rightarrow \mathrm{E}\,\mathrm{s}x$$
$$\mathrm{E}\,x \Rightarrow \mathrm{O}\,\mathrm{s}x$$
$$\mathrm{N}\,\mathrm{ss}0 \Rightarrow \mathrm{O}\,\mathrm{sss}0$$

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. $\forall m : m \in [\![\mathrm{N}\,0]\!]_1$)



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

Extracting Semantic Orderings: Example

$\Rightarrow \mathsf{N}\,0$

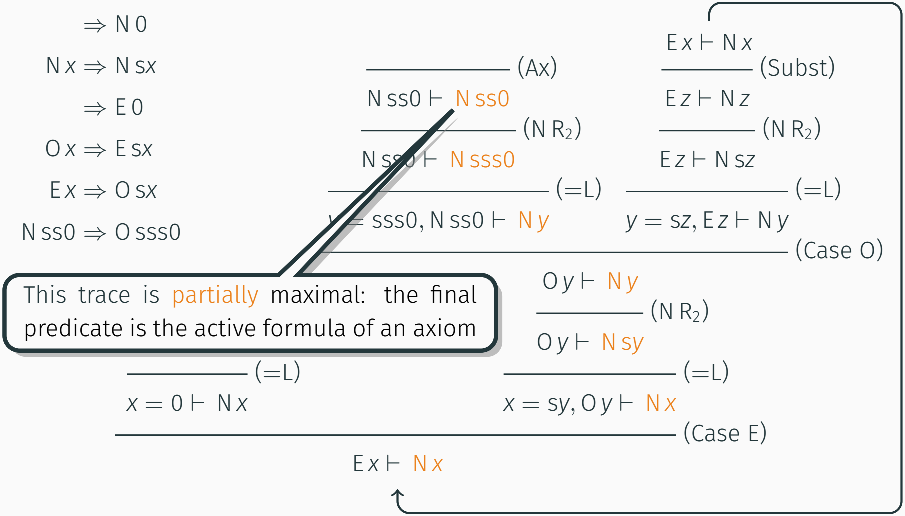
$\mathsf{N}\,x \Rightarrow \mathsf{N}\,\mathsf{s}x$

$\Rightarrow \mathsf{E}\,0$

$\mathsf{O}\,x \Rightarrow \mathsf{E}\,\mathsf{s}x$

$\mathsf{E}\,x \Rightarrow \mathsf{O}\,\mathsf{s}x$

$\mathsf{N}\,\mathsf{ss}0 \Rightarrow \mathsf{O}\,\mathsf{sss}0$



$\mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,\mathsf{ss}0$ (Ax)

$\mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,\mathsf{sss}0$ (N R$_2$)

$y = \mathsf{sss}0, \mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,y$ (=L)

$\mathsf{E}\,x \vdash \mathsf{N}\,x$

$\mathsf{E}\,z \vdash \mathsf{N}\,z$ (Subst)

$\mathsf{E}\,z \vdash \mathsf{N}\,\mathsf{s}z$ (N R$_2$)

$y = \mathsf{s}z, \mathsf{E}\,z \vdash \mathsf{N}\,y$ (=L)

$\mathsf{O}\,y \vdash \mathsf{N}\,y$ (Case O)

$\mathsf{O}\,y \vdash \mathsf{N}\,\mathsf{s}y$ (N R$_2$)

$x = \mathsf{s}y, \mathsf{O}\,y \vdash \mathsf{N}\,x$ (=L)

$x = 0 \vdash \mathsf{N}\,x$ (=L)

$\mathsf{E}\,x \vdash \mathsf{N}\,x$ (Case E)

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

Extracting Semantic Orderings: Example

$$\Rightarrow \mathsf{N}\,0$$

$$\mathsf{N}\,x \Rightarrow \mathsf{N}\,\mathsf{s}x$$

$$\Rightarrow \mathsf{E}\,0$$

$$\mathsf{O}\,x \Rightarrow \mathsf{E}\,\mathsf{s}x$$

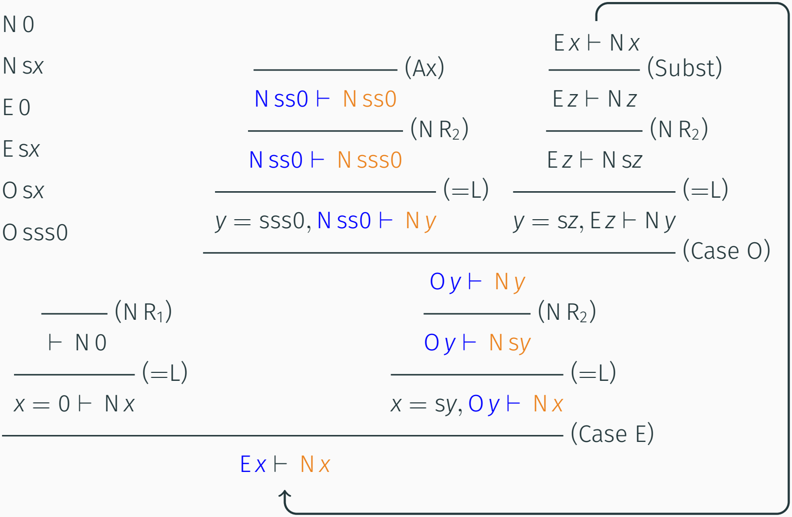
$$\mathsf{E}\,x \Rightarrow \mathsf{O}\,\mathsf{s}x$$

$$\mathsf{N}\,\mathsf{ss}0 \Rightarrow \mathsf{O}\,\mathsf{sss}0$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}\,0}\,(\mathsf{N}\,\mathsf{R}_1)}{x = 0 \vdash \mathsf{N}\,x}\,(=\mathsf{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,\mathsf{ss}0}\,(\mathsf{Ax})}{\mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,\mathsf{sss}0}\,(\mathsf{N}\,\mathsf{R}_2)}{y = \mathsf{sss}0, \mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,y}\,(=\mathsf{L}) \quad \dfrac{\dfrac{\dfrac{\mathsf{E}\,x \vdash \mathsf{N}\,x}{\mathsf{E}\,z \vdash \mathsf{N}\,z}\,(\mathsf{Subst})}{\mathsf{E}\,z \vdash \mathsf{N}\,\mathsf{s}z}\,(\mathsf{N}\,\mathsf{R}_2)}{y = \mathsf{s}z, \mathsf{E}\,z \vdash \mathsf{N}\,y}\,(=\mathsf{L})}{\mathsf{O}\,y \vdash \mathsf{N}\,y}\,(\mathsf{Case}\ \mathsf{O})}{\mathsf{O}\,y \vdash \mathsf{N}\,\mathsf{s}y}\,(\mathsf{N}\,\mathsf{R}_2)}{x = \mathsf{s}y, \mathsf{O}\,y \vdash \mathsf{N}\,x}\,(=\mathsf{L})}{\mathsf{E}\,x \vdash \mathsf{N}\,x}\,(\mathsf{Case}\ \mathsf{E})$$

Extracting Semantic Orderings: Example

$$\begin{aligned} &\Rightarrow \mathsf{N}\,0 \\ \mathsf{N}\,x &\Rightarrow \mathsf{N}\,\mathsf{s}x \\ &\Rightarrow \mathsf{E}\,0 \\ \mathsf{O}\,x &\Rightarrow \mathsf{E}\,\mathsf{s}x \\ \mathsf{E}\,x &\Rightarrow \mathsf{O}\,\mathsf{s}x \\ \mathsf{N}\,\mathsf{ss}0 &\Rightarrow \mathsf{O}\,\mathsf{sss}0 \end{aligned}$$



$$\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{N}\,0}\,(\mathsf{N}\,\mathsf{R}_1)}{x = 0 \vdash \mathsf{N}\,x}\,(=\mathsf{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,\mathsf{ss}0}\,(\mathsf{Ax})}{\mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,\mathsf{sss}0}\,(\mathsf{N}\,\mathsf{R}_2)}{y = \mathsf{sss}0, \mathsf{N}\,\mathsf{ss}0 \vdash \mathsf{N}\,y}\,(=\mathsf{L}) \quad \dfrac{\dfrac{\dfrac{\mathsf{E}\,x \vdash \mathsf{N}\,x}{\mathsf{E}\,z \vdash \mathsf{N}\,z}\,(\mathsf{Subst})}{\mathsf{E}\,z \vdash \mathsf{N}\,\mathsf{s}z}\,(\mathsf{N}\,\mathsf{R}_2)}{y = \mathsf{s}z, \mathsf{E}\,z \vdash \mathsf{N}\,y}\,(=\mathsf{L})}{\mathsf{O}\,y \vdash \mathsf{N}\,y}\,(\mathsf{Case\ O})}{\mathsf{O}\,y \vdash \mathsf{N}\,\mathsf{s}y}\,(\mathsf{N}\,\mathsf{R}_2)}{x = \mathsf{s}y, \mathsf{O}\,y \vdash \mathsf{N}\,x}\,(=\mathsf{L})}{\mathsf{E}\,x \vdash \mathsf{N}\,x}\,(\mathsf{Case\ E})$$

Extracting Semantic Orderings: A Realizability Condition

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 $\mathsf{P}\,\vec{x} \vdash \mathsf{Q}\,\vec{y}$ *satisfying the realizability condition, then* $[\![\mathsf{P}\,\vec{x}]\!]_\alpha \subseteq [\![\mathsf{Q}\,\vec{y}]\!]_\alpha$*, for all* α *(i.e.* $\mathsf{Q}\,\vec{y} \leq \mathsf{P}\,\vec{x}$*)*

Proof.

Soundness of the Realizability Condition

Theorem

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

Proof.

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

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

Weighted Automata

Definition (Weighted Automata)

Let Σ be an alphabet, and $(V, \oplus, \otimes)$ a semiring of weights. A weighted automaton $\mathscr{A}$ is a tuple (Q, q_I, F, γ) 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 $\gamma : (Q \times \Sigma \times Q) \to V$.

Weighted Automata

Definition (Weighted Automata)

Let Σ be an alphabet, and $(V, \oplus, \otimes)$ a semiring of weights. A weighted automaton $\mathscr{A}$ is a tuple (Q, q_I, F, γ) 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 $\mathscr{A}$ 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 $\mathcal{L}_{\mathscr{A}}$ is the function $\Sigma^* \rightharpoonup V$ computed by $\mathscr{A}$

Weighted Automata

Definition (Weighted Automata)

Let Σ be an alphabet, and $(V, \oplus, \otimes)$ a semiring of weights. A weighted automaton $\mathscr{A}$ is a tuple (Q, q_I, F, γ) 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 $\mathscr{A}$ 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 $\mathcal{L}_{\mathscr{A}}$ is the function $\Sigma^* \rightharpoonup V$ computed by $\mathscr{A}$

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 $\mathcal{L}_1(w) \leq \mathcal{L}_2(w)$

Weighted Automata

Definition (Weighted Automata)

Let Σ be an alphabet, and $(V, \oplus, \otimes)$ a semiring of weights. A weighted automaton $\mathscr{A}$ is a tuple (Q, q_I, F, γ) 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 $\mathscr{A}$ 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 $\mathcal{L}_{\mathscr{A}}$ is the function $\Sigma^* \rightharpoonup V$ computed by $\mathscr{A}$

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 $\mathcal{L}_1(w) \leq \mathcal{L}_2(w)$

Sum automata are weighted automata over $(\mathbb{N}, +, \max)$

Weighted Automata: Results

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 $\mathcal{L}_1(w) \leq \mathcal{L}_2(w)$

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)

Weighted Automata: Results

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 $\mathcal{L}_1(w) \leq \mathcal{L}_2(w)$

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)

Definition

A weighted automaton is called finite-valued if there exists a bound on the number of distinct values of accepting runs on any given word

Theorem

Given two finite-valued weighted automata $\mathscr{A}$ and $\mathscr{B}$, it is decidable whether $\mathcal{L}_{\mathscr{A}} \leq \mathcal{L}_{\mathscr{B}}$ (Filiot, Gentilini & Raskin '14)

Weighted Automata from Cyclic Entailment Proofs

Given a cyclic entailment proof $\mathcal{P}$, we can construct two kinds of finite-valued sum automata, $\mathscr{A}_{\mathcal{P}}[n]$ ($n \in \mathbb{N}$) and $\mathscr{C}_{\mathcal{P}}$, 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
    - $\mathscr{C}_{\mathcal{P}}$ only counts traces following the full path
    - the $\mathscr{A}_{\mathcal{P}}[n]$ count traces following any prefix of the path
- Each $\mathscr{A}_{\mathcal{P}}[n]$ considers only a subset of the paths in the proof
    - A complete automaton can be constructed but is not, in general, finite-valued
- $\mathscr{C}_{\mathcal{P}}$ is grounded when all final states correspond to ground predicate instances

Deciding the Realizability Condition

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 $\mathscr{C}_{\mathcal{P}} \leq \mathscr{A}_{\mathcal{P}}[N]$ and $\mathscr{C}_{\mathcal{P}}$ 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 completenss 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 $[\![\mathrm{P}\,\vec{x}]\!]_\alpha \subseteq [\![\mathrm{Q}\,\vec{y}]\!]_\alpha$ holds, is there a cyclic proof of $\mathrm{P}\,\vec{x} \vdash \mathrm{Q}\,\vec{y}$ satisfying the realizability condition?

Bootstrapping Cyclic Entailment Systems

Suppose we can deduce from a proof of $\Gamma, \mathrm{P}\,\vec{t} \vdash \Sigma, \mathrm{Q}\,\vec{u}$ that $\mathrm{Q}\,\vec{u} \leq \mathrm{P}\,\vec{t}$

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

$$\frac{\Gamma, \mathrm{P}\,\vec{t} \vdash \Sigma, \mathrm{Q}\,\vec{u} \qquad \mathrm{Q}\,\vec{u}, \Pi \vdash \Delta}{\Gamma, \mathrm{P}\,\vec{t}, \Pi \vdash \Sigma, \Delta}$$

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 $[\![\mathrm{P}\,\vec{t}]\!]_\alpha$ and $[\![\mathrm{Q}\,\vec{u}]\!]_\alpha$

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