Uniform Inductive Reasoning in Transitive Closure Logic via Infinite Descent

Liron Cohen ¹  Reuben N. S. Rowe ²
SREPLS 10, Birkbeck, University of London
Tuesday 18th September 2018

¹Dept of Computer Science, Cornell University, Ithaca, NY, USA
²School of Computing, University of Kent, Canterbury, UK
Motivation

• Carry out formal inductive reasoning

• Do so automatically (as much as possible)

• Study/compare different ‘styles’ of inductive reasoning
Formalising Inductive Reasoning
Explicit Inductive Definitions

• Use clauses to inductively define predicates:

\[ \phi_1 \land \ldots \land \phi_n \Rightarrow P(\overrightarrow{t}) \]

\[ \vdots \]

\[ \psi_1 \land \ldots \land \psi_m \Rightarrow P(\overrightarrow{t}) \]

• We take the smallest interpretation closed under the rules

<table>
<thead>
<tr>
<th>N</th>
<th>N x</th>
<th>E 0</th>
<th>O x</th>
<th>E x</th>
</tr>
</thead>
<tbody>
<tr>
<td>N 0</td>
<td>N s x</td>
<td>E 0</td>
<td>E s x</td>
<td>O s x</td>
</tr>
</tbody>
</table>

\[[N] = \{0, s0, ss0, \ldots, s^n 0, \ldots\}\]

\[[E] = \{0, ss0, \ldots, s^{2n} 0, \ldots\}\]

\[[O] = \{s0, \ldots, s^{2n+1} 0, \ldots\}\]
Reasoning Using Explicit Induction Principles

- We reason using the corresponding *induction principles*

\[
\Gamma \vdash \text{IND}_Q(F) \quad (\forall Q \text{ mutually recursive with } P) \quad \Gamma, F(\tilde{t}) \vdash \Delta
\]

\[
\Gamma, P\tilde{t} \vdash \Delta
\]

- E.g. the productions for \(N\) give

\[
\Gamma \vdash F(0) \quad \Gamma, F(x) \vdash F(sx) \quad \Gamma, F(t) \vdash \Delta
\]

\[
\Gamma, N\tilde{t} \vdash \Delta
\]
We trace predicate instances through the proof.
At certain points, these progress (i.e., get 'smaller').
Each infinite path must admit some infinite descent.
This global trace condition is an ω-regular property.

\[ P(\vec{t}) \]
\[ Q(\vec{u}) \]
We trace predicate instances through the proof.
At certain points, these progress (i.e., get 'smaller').
Each infinite path must admit some infinite descent.
This global trace condition is an $\omega$-regular property.
i.e., decidable using Büchi automata.
Non-well-founded Proofs: Reasoning by Infinite Descent

We trace predicate instances through the proof
Non-well-founded Proofs: Reasoning by Infinite Descent

We trace predicate instances through the proof.

At certain points, these progress (i.e. get ‘smaller’).
We trace predicate instances through the proof.

- At certain points, these progress (i.e. get ‘smaller’).
- Each infinite path must admit some infinite descent.
We trace predicate instances through the proof

- At certain points, these progress (i.e. get ‘smaller’)
- Each infinite path must admit some infinite descent
- This global trace condition is an $\omega$-regular property
  - i.e. decidable using Büchi automata
An Example Cyclic Proof

⇒ N 0
N x ⇒ N sx
⇒ E 0
O x ⇒ E sx
E x ⇒ O sx

Ex ⊢ N x

=N L
x = 0 ⊢ N x

(Subst)

[N R1]

Ez ⊢ N z

(N R2)

Ez ⊢ N sz

= L

y = sz, Ez ⊢ Ny

(Case O)

O y ⊢ Ny

= L

O y ⊢ N sy

= L

x = sy, O y ⊢ N x

(Case E)

Ex ⊢ N x
An Example Cyclic Proof

⇒ N 0
N x ⇒ N sx
⇒ E 0
O x ⇒ E sx
E x ⇒ O sx

Ex ⊢ Nx (Subst)
Ez ⊢ Nz
Ez ⊢ N sx

⇒ (N R1)
⇒ (N R2)

⇒ (Case O)
⇒ (Case E)
x = 0 ⊢ Nx
x = sy, O y ⊢ N x

Left unfolding rule
An Example Cyclic Proof

\[ N \times \\text{(Subst)} \]
\[ E \times \text{(N R)} \]
\[ E \times \text{(N R)} \]
\[ y = sz, E \times \text{(Case O)} \]
\[ Oy \times \text{(N R)} \]
\[ Oy \times \text{(Case E)} \]

Left unfolding rule

\[ \Rightarrow N \times \]
\[ N \times \Rightarrow N \times \]
\[ \Rightarrow E \times \]
\[ O \times \Rightarrow E \times \]
\[ Ex \Rightarrow O \times \]
An Example Cyclic Proof

Right unfolding rule

\[ \Rightarrow N\,0 \]

\[ N\,x \Rightarrow N\,sx \]

\[ \Rightarrow E\,0 \]

\[ O\,x \Rightarrow E\,sx \]

\[ E\,x \Rightarrow O\,sx \]

\[ \vdash N\,0 \]

\[ (NR_1) \]

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

\[ (=L) \]

\[ (NR_2) \]

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

\[ (Case\,O) \]

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

\[ (=L) \]

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

\[ (Case\,E) \]

\[ Ex \vdash N\,x \]
An Example Cyclic Proof

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

Right unfolding rule

\[ \begin{align*}
Ex \vdash N \ x & \quad (\text{Subst}) \\
Ez \vdash N \ z & \quad (N \ R_2) \\
Ez \vdash N \ sx & \\
& \quad (=L) \\
y = sz, Ez \vdash N \ y & \quad (\text{Case O}) \\
o y \vdash N \ y & \quad (N \ R_2) \\
o y \vdash N \ sy & \quad (=L) \\
x = sy, oy \vdash N \ x & \quad (\text{Case E}) \\
Ex \vdash N \ x
\end{align*} \]
An Example Cyclic Proof

⇒ N 0
N x ⇒ N sx
⇒ E 0
O x ⇒ E sx
E x ⇒ O sx

Ex ⊢ Nx

(Subst)

Nz ⊢ Nz

(N R₂)

Nsz ⊢ Nsz

(=L)

y = sz, Ez ⊢ Ny

(Case O)

Ny ⊢ Ny

(N R₂)

O y ⊢ N sy

(=L)

x = sy, O y ⊢ Nx

(Case E)

Ex ⊢ Nx

Right unfolding rule
An Example Cyclic Proof

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

\[Ex \vdash Nx\]
\[\underline{Es \vdash N sz}\]
\[\underline{Ez \vdash N z}\]
\[\underline{N R_1 \vdash N 0}\]
\[\underline{x = 0 \vdash Nx}\]
\[\underline{N R_2 \vdash N sz}\]
\[\underline{y = sz, Ez \vdash Ny}\]
\[\underline{Ny \vdash N sx}\]
\[\underline{N R_2 \vdash N sy}\]
\[\underline{O y \vdash N y}\]
\[\underline{O y \vdash N sy}\]
\[\underline{x = sy, O y \vdash Nx}\]
\[\underline{Case E}\]

\[\Rightarrow N x\]
Comparing the Two Approaches

For FOL with Martin-Löf style inductive definitions:

[Brotherston & Simpson, 2007]

- Infinitary system sound/complete for standard semantics

- Explicit induction sound/complete for Henkin semantics
Comparing the Two Approaches

For FOL with Martin-Löf style inductive definitions: [Brotherston & Simpson, 2007]

- Infinitary system sound/complete for standard semantics
- Cyclic system subsumes explicit induction

- Explicit induction sound/complete for Henkin semantics
Comparing the Two Approaches

For FOL with Martin-Löf style inductive definitions:

[Brotherston & Simpson, 2007]

• Infinitary system sound/complete for standard semantics

• Cyclic system subsumes explicit induction

  • Equivalent under arithmetic
  • Not equivalent in general (2-Hydra counterexample)

[Berardi & Tatsuta, 2017]

• Explicit induction sound/complete for Henkin semantics
Transitive Closure Logic
Transitive Closure Logic

Transitive Closure (TC) Logic extends FOL with formulas:

- \((RTC_{x,y} \varphi)(s, t)\)
  - \(\varphi\) is a formula
  - \(x\) and \(y\) are distinct variables (which become bound in \(\varphi\))
  - \(s\) and \(t\) are terms

whose intended meaning is an infinite disjunction

\[ s = t \lor \varphi[s/x, t/y] \]
\[ \lor (\exists w_1 . \varphi[s/x, w_1/y] \land \varphi[w_1/x, t/y]) \]
\[ \lor (\exists w_1, w_2 . \varphi[s/x, w_1/y] \land \varphi[w_1/x, w_2/y] \land \varphi[w_2/x, t/y]) \]
\[ \lor \ldots \]
The formal semantics:

- \( M \) is a (standard) first-order model with domain \( D \)
- \( v \) is a valuation of terms in \( M \):

\[
M, v \models (RTC_{x,y} \varphi)(s, t)
\]
The formal semantics:

- $M$ is a (standard) first-order model with domain $D$
- $v$ is a valuation of terms in $M$:

\[ M, v \models (RTC_{x,y} \varphi)(s, t) \iff \exists a_0, \ldots, a_n \in D \]
Transitive Closure Logic: Standard Semantics

The formal semantics:

- $M$ is a (standard) first-order model with domain $D$
- $v$ is a valuation of terms in $M$:

\[
M, v \models (RTC_{x,y} \varphi)(s, t) \iff \\
\exists a_0, \ldots, a_n \in D . v(s) = a_0 \land v(t) = a_n
\]
The formal semantics:

- $M$ is a (standard) first-order model with domain $D$
- $v$ is a valuation of terms in $M$:

$$M, v \models (RTC_{x,y} \varphi)(s, t) \iff \exists a_0, \ldots, a_n \in D \cdot v(s) = a_0 \land v(t) = a_n$$

$$\land M, v[x := a_i, y := a_{i+1}] \models \varphi \quad \text{for all } i < n$$
Example: Arithmetic in TC

Take a signature $\Sigma = \{0, s\} + \text{equality}

$$\text{Nat}(x) \equiv (RTC_{v,w} sv = w)(0, x)$$
Example: Arithmetic in TC

Take a signature $\Sigma = \{0, s\} + \text{equality}$

$$\text{Nat}(x) \equiv (RTC_{v,w} sv = w)(0, x)$$

$$x \leq y \equiv (RTC_{v,w} sv = w)(x, y)$$
Example: Arithmetic in TC

Take a signature $\Sigma = \{0, s\} +$ equality and pairing

$$\text{Nat}(x) \equiv (RTC_{v,w} sv = w)(0, x)$$

$$x \leq y \equiv (RTC_{v,w} sv = w)(x, y)$$

“$x = y + z$” $\equiv$

$$(RTC_{v,w} \exists n_1, n_2 \cdot v = \langle n_1, n_2 \rangle \land w = \langle sn_1, sn_2 \rangle)(\langle 0, y \rangle, \langle z, x \rangle)$$
Example: Arithmetic in TC

Take a signature $\Sigma = \{0, s\} + \text{equality and pairing}$

\[
\text{Nat}(x) \equiv (RTC_{v,w} sv = w)(0, x)
\]

\[
x \leq y \equiv (RTC_{v,w} sv = w)(x, y)
\]

"$x = y + z$" $\equiv$

\[
(RTC_{v,w} \exists n_1, n_2 . v = \langle n_1, n_2 \rangle \land w = \langle sn_1, sn_2 \rangle)(\langle 0, y \rangle, \langle z, x \rangle)
\]
Example: Arithmetic in TC

Take a signature $\Sigma = \{0, s\} + \text{equality and pairing}$

$$\text{Nat}(x) \equiv (RTC_{v,w} sv = w)(0,x)$$

$$x \leq y \equiv (RTC_{v,w} sv = w)(x,y)$$

“$x = y + z$” $\equiv$

$$(RTC_{v,w} \exists n_1, n_2 \cdot v = \langle n_1, n_2 \rangle \land w = \langle sn_1, sn_2 \rangle)(\langle 0, y \rangle, \langle z, x \rangle)$$
Example: Arithmetic in TC

Take a signature $\Sigma = \{0, s\} + \text{equality and pairing}$

$$\text{Nat}(x) \equiv (RTC_{v,w} \; sv = w)(0, x)$$

$$x \leq y \equiv (RTC_{v,w} \; sv = w)(x, y)$$

“$x = y + z$” $\equiv$

$$(RTC_{v,w} \; \exists n_1, n_2 . \; v = \langle n_1, n_2 \rangle \land w = \langle sn_1, sn_2 \rangle)(\langle 0, y \rangle, \langle z, x \rangle)$$
Example: Arithmetic in TC

Take a signature $\Sigma = \{0, s\} + \text{equality and pairing}$

$$\text{Nat}(x) \equiv (RTC_{v, w} sv = w)(0, x)$$

$$x \leq y \equiv (RTC_{v, w} sv = w)(x, y)$$

"$x = y + z$" $\equiv$

$$(RTC_{v, w} \exists n_1, n_2 . v = \langle n_1, n_2 \rangle \land w = \langle sn_1, sn_2 \rangle)(\langle 0, y \rangle, \langle z, x \rangle)$$
Proof Rules for Reasoning in TC

reflexivity

\[
\Gamma \vdash (RTC_{x,y} \varphi)(t, t)
\]

step

\[
\begin{align*}
\Gamma & \vdash \Delta, (RTC_{x,y} \varphi)(s, r) \quad \Gamma \vdash \Delta, \varphi[r/x, t/y] \\
\Gamma & \vdash \Delta, (RTC_{x,y} \varphi)(s, t)
\end{align*}
\]

induction

\[
\begin{align*}
\Gamma & \vdash \Delta, \psi[s/x] \quad \Gamma, \psi(x), \varphi(x, y) \vdash \Delta, \psi[y/x] \quad \Gamma, \psi[t/x] \vdash \Delta \\
\Gamma, (RTC_{x,y} \varphi)(s, t) & \vdash \Delta
\end{align*}
\]

\[x \not\in \text{fv}(\Gamma, \Delta) \text{ and } y \not\in \text{fv}(\Gamma, \Delta, \psi)\]
Proof Rules for Reasoning in TC

reflexivity

\[ \Gamma \vdash (RTC_{x,y} \varphi)(t, t) \]

step

\[ \Gamma \vdash \Delta, (RTC_{x,y} \varphi)(s, r) \quad \Gamma \vdash \Delta, \varphi[r/x, t/y] \]

\[ \Gamma \vdash \Delta, (RTC_{x,y} \varphi)(s, t) \]

induction

\[ \Gamma \vdash \Delta, \psi[s/x] \quad \Gamma, \psi(x), \varphi(x, y) \vdash \Delta, \psi[y/x] \quad \Gamma, \psi[t/x] \vdash \Delta \]

\[ \Gamma, (RTC_{x,y} \varphi)(s, t) \vdash \Delta \]

\( x \notin \text{fv}(\Gamma, \Delta) \) and \( y \notin \text{fv}(\Gamma, \Delta, \psi) \)

case-split

\[ \Gamma, s = t \vdash \Delta \quad \Gamma, (RTC_{x,y} \varphi)(s, z), \varphi[z/x, t/y] \vdash \Delta \]

\[ \Gamma, (RTC_{x,y} \varphi)(s, t) \vdash \Delta \]

(z fresh)
Proof Rules for Reasoning in TC

reflexivity

\[ \Gamma \vdash (RTC_{x,y} \varphi)(t, t) \]

step

\[ \Gamma \vdash \Delta, (RTC_{x,y} \varphi)(s, r) \quad \Gamma \vdash \Delta, \varphi[r/x, t/y] \]

\[ \Gamma \vdash \Delta, (RTC_{x,y} \varphi)(s, t) \]

induction

\[ \Gamma \vdash \Delta, \psi[s/x] \quad \Gamma, \psi(x), \varphi(x, y) \vdash \Delta, \psi[y/x] \quad \Gamma, \psi[t/x] \vdash \Delta \]

\[ \Gamma, (RTC_{x,y} \varphi)(s, t) \vdash \Delta \]

\[ x \notin \text{fv}(\Gamma, \Delta) \text{ and } y \notin \text{fv}(\Gamma, \Delta, \psi) \]

case-split

\[ \Gamma, s = t \vdash \Delta \quad \Gamma, (RTC_{x,y} \varphi)(s, z), \varphi[z/x, t/y] \vdash \Delta \]

\[ \Gamma, (RTC_{x,y} \varphi)(s, t) \vdash \Delta \]
Advantages of TC as a Formal Framework

- It is only a minimal extension of FOL
- It only requires a single, uniform induction principle
- No need to ‘choose’ particular inductive definitions
- Inductive definitions can use arbitrary formulas
- It is a sufficiently expressive logic

Theorem (Avron ’03)

All finitely inductively definable relations\(^\dagger\) are definable in TC.

A. Avron, *Transitive Closure and the Mechanization of Mathematics.*

\(^\dagger\) as formalised in: S. Feferman, *Finitary Inductively Presented Logics*, 1989
Comparing Styles of Induction for TC

- Infinitary system sound/complete for standard semantics

- Explicit induction sound/complete for Henkin semantics
• Infinitary system sound/complete for standard semantics
• Cyclic system subsumes explicit induction
• Explicit induction sound/complete for Henkin semantics
Comparing Styles of Induction for TC

- Infinitary system sound/complete for standard semantics
- Cyclic system subsumes explicit induction
  - Equivalent under arithmetic
- Explicit induction sound/complete for Henkin semantics
Comparing Styles of Induction for TC

- Infinitary system sound/complete for standard semantics
- Cyclic system subsumes explicit induction
  - Equivalent under arithmetic
  - Don’t know if they are inequivalent in general!
    2-Hydra does not work since all inductive definitions available via RTC
- Explicit induction sound/complete for Henkin semantics
Future Work

• open question of equivalence for TC proof systems

• Implementation to support automated reasoning.

• Use TC to better study implicit vs explicit induction.

• Adapt TC for coinductive reasoning?
Thank you!