An Introduction to Cyclic Proofs (part II)

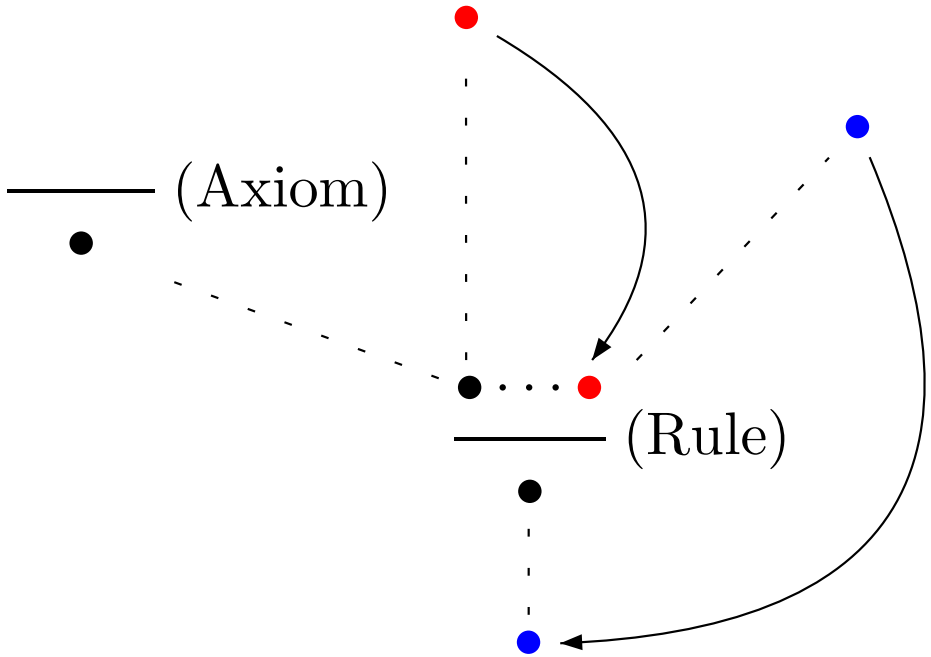
James Brotherston

University College London

PARIS workshop, FLoC, Oxford, 8th July 2018

Cyclic proofs

Cyclic pre-proofs are derivation trees with backlinks:



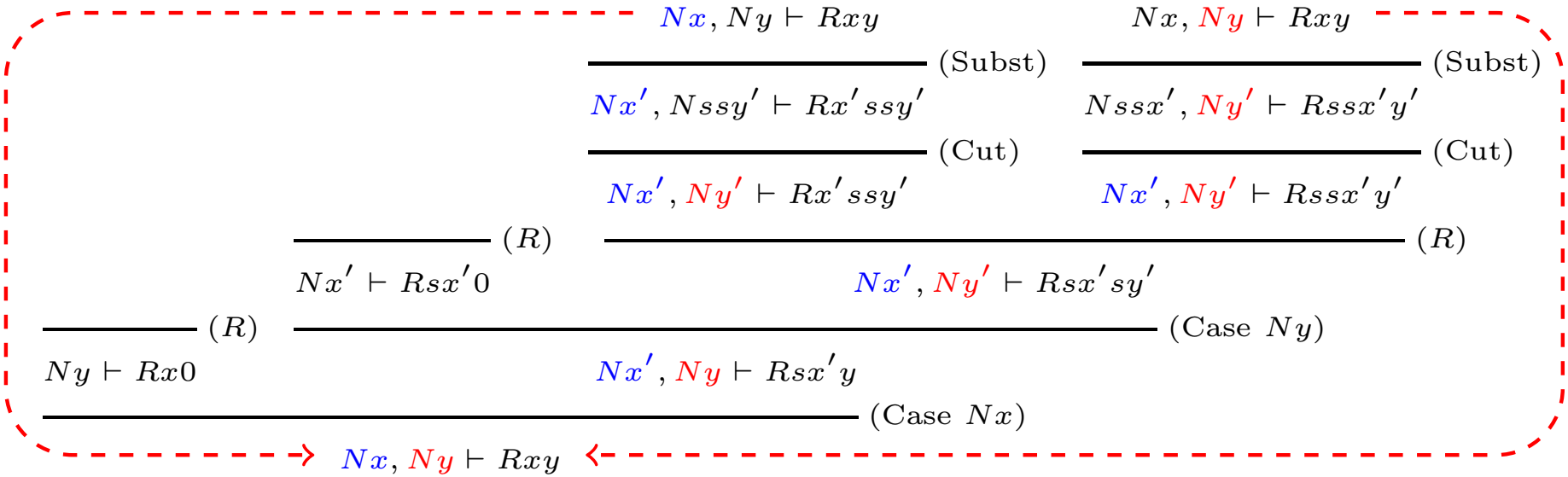
Soundness condition: *A pre-proof is a cyclic proof if, for every infinite path in the proof, there is an infinitely progressing trace along some tail of the path.*

Failure of per-cycle soundness

Consider inductive definitions:

$$\begin{array}{rcl@{\qquad}rcl} & \Rightarrow & N0 & & \Rightarrow & R0y \qquad \Rightarrow Rx0 \\ Nx & \Rightarrow & Nsx & R(ssx,y), R(x,ssy) & \Rightarrow & Rsxsy \end{array}$$

Now $Nx, Ny \vdash Rxy$ is not valid. E.g. $R(s0, ss0)$ fails. But:



The most common question

Infinite descent principle for $\mathbb{N}$:

$$\frac{\neg P(k) \to (\exists k' < k \in \mathbb{N}.\ \neg P(k'))}{\forall n \in \mathbb{N}.\ P(n)}\ (k \text{ arbitrary})$$

Complete induction principle:

$$\frac{(\forall k' < k \in \mathbb{N}.\ P(k')) \to P(k)}{\forall n \in \mathbb{N}.\ P(n)}\ (k \text{ arbitrary})$$

These are obviously interderivable, so aren't cyclic proof and induction proof just the same thing?

The main difficulty is that

- cyclic proof encodes a relatively strong form of infinite descent that is implicit in the structure of the proof (nested cycles, etc.), while

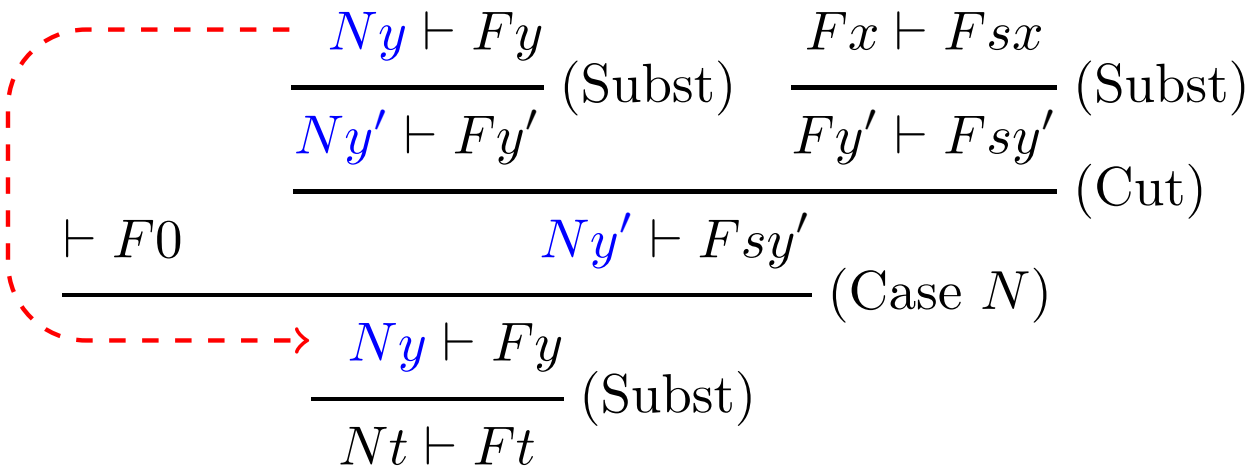
- induction proof often uses a relatively weak form of induction encoded explicitly as a local inference rule. E.g., for N:

$$\frac{\vdash F0 \qquad Fx \vdash Fsx}{Nt \vdash Ft}\,(\text{Ind } N)$$

The equivalence of the two styles, for FOL with ind defns, was a conjecture (Brotherston and Simpson, LICS 2007)

From cyclic to induction proof

Cyclic derivation of N-induction:



This construction generalises to arbitrary inductive definitions.

Theorem

Any sequent provable by induction also has a cyclic proof.

Peano arithmetic using inductive defns

There is an embedding of Peano arithmetic (PA) into an explicit-induction proof system:

- add the first six Peano axioms as closed formulas (on the LHS);
- add formulas Nx for each free variable x;
- relativise all quantifiers over N;
- the Peano induction axiom follows from the induction rule for N.

This means we can formalise PA in a cyclic proof system as well.

An aside on completeness

If we allow proofs to be arbitrary infinite trees rather than cyclic graphs then the system becomes complete (Brotherston and Simpson LICS 2007).

Since we can formalise PA using induction and thus cyclic proof, this gives us a complete system for arithmetic.

However, since true arithmetic is not even semidecidable, there can be no recursive enumeration of the proofs in this system!

Results on cyclic arithmetic

Theorem (Simpson, FoSSaCS 2017)

Cyclic arithmetic is equivalent to Peano arithmetic.

Proof is by formalising the soundness of cyclic arithmetic inside ACA_0 which is conservative over PA.

Theorem (Berardi and Tatsuta, LICS 2017)

Cyclic proof is equivalent to induction proof for any signature that includes Peano arithmetic.

Proof is by explicit conversion, defining a notion of ¡ for all predicates and formalising a version of Ramsay's theorem using explicit induction.

However...

Theorem (Berardi and Tasuta, FoSSaCS 2017)

There is a signature for which cyclic proof is not equivalent to induction proof.

This is essentially because cyclic proof implicitly lets us do things like infinite descent over the max or min of two numbers, concepts which might not be explicitly formalisable in restricted signatures.

Cyclist *theorem prover*

- A generic (logic-independent) theorem prover that supports cyclic proof
- Lead developer Nikos Gorogiannis (Facebook & U. Middlesex)
- Support for inductive definitions
- Automatic checking of cyclic soundness condition (using the Büchi automata construction from yesterday)
- Open source:

`github.com/ngorogiannis/cyclist`

Some CYCLIST *instantiations*

- first-order logic with ind defns
- separation logic with ind defns
- Hoare logic for program termination with recursive procedures (R. Rowe)
- Hoare logic for temporal program properties (G. Tellez Espinosa)

Build your own CYCLIST *instantiation*

To implement your favourite cyclic proof system in CYCLIST you need to provide the following (to `Ocaml` functors):

- a syntax for proof judgements;
- some proof rules for judgements;
- the (progressing) trace pairs associated with each proof rule;
- a matching condition for backlinking;
- (optional) a preferred search strategy.

Why not try it?