An introduction to cyclic proof

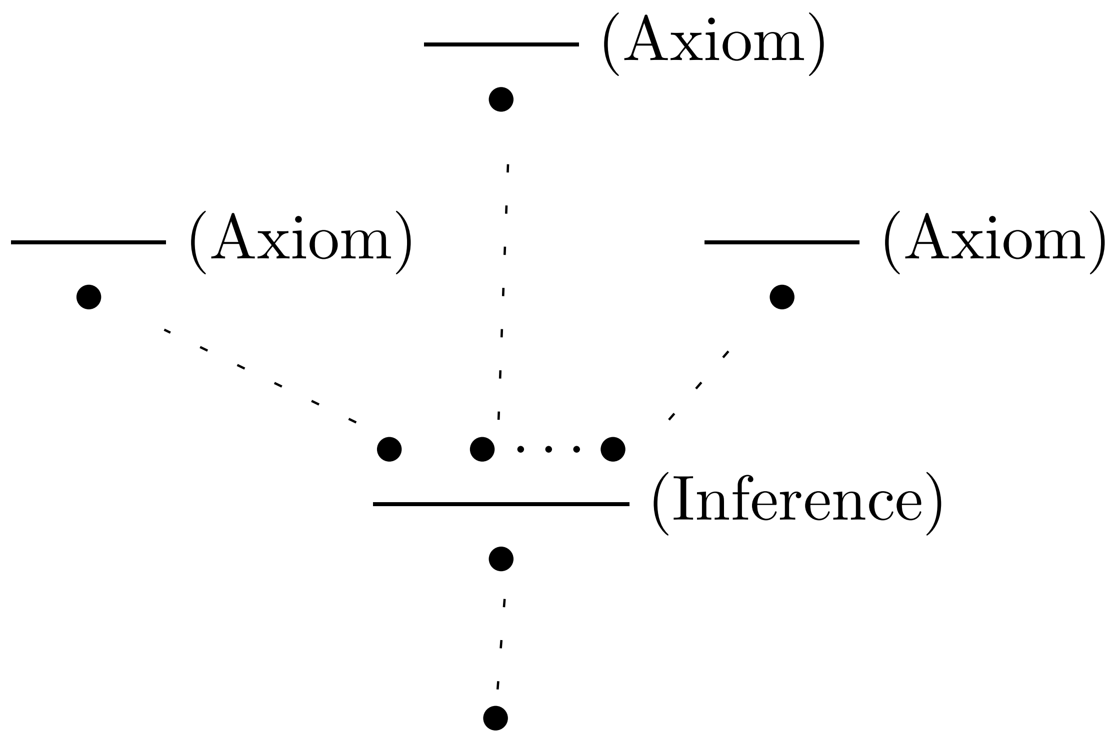
James Brotherston

Imperial College, London

London Theory Day
11 April, 2008

Tree proof vs. cyclic proof (1)

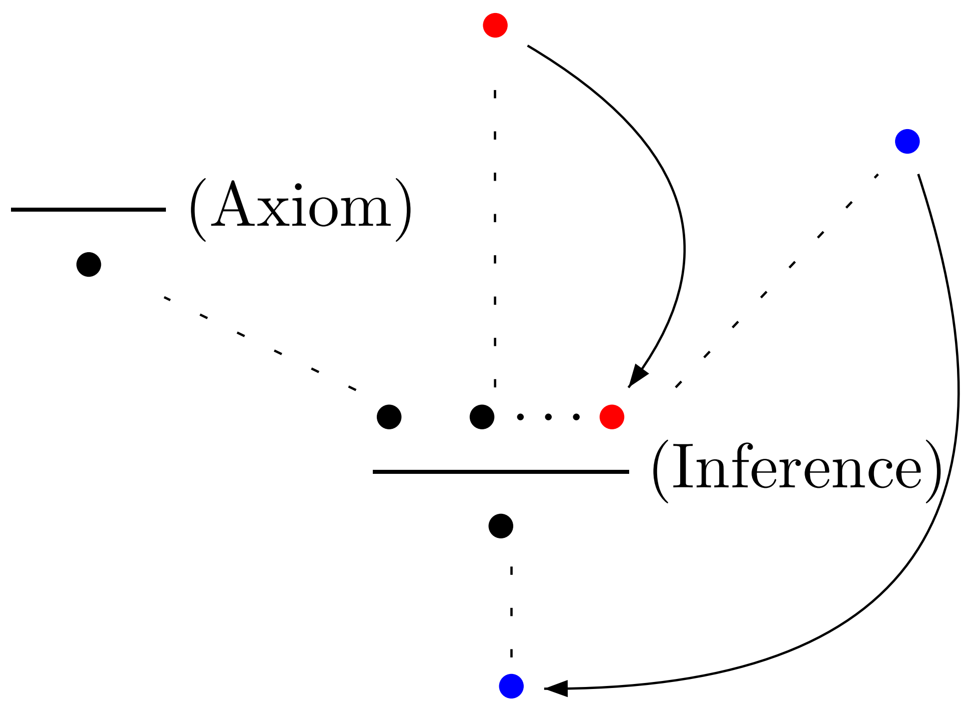
- Usually a proof is a finite tree of sequents (•):



- Soundness of such proofs follows from the local soundness of each inference rule / axiom.

Tree proof vs. cyclic proof (2)

- A cyclic pre-proof is formed from a (partial) derivation by identifying each open subgoal (called a bud) with an identical interior sequent (called its companion):



- Cyclic pre-proofs are not sound in general — we need some extra condition.
- Cyclic proof = pre-proof $\mathcal{P}$ + soundness condition $S(\mathcal{P})$.

Example (cf. Stirling & Bradfield): cyclic proofs of μ-calculus properties of processes

Consider a "clock" process Cl which repeatedly ticks:

$$Cl =_{\text{def}} tick.Cl$$

The μ-calculus formula $\nu X.\,\langle tick\rangle X$ means "the action 'tick' can be performed infinitely often".

$$\dfrac{\dfrac{Cl \vdash \nu X.\,\langle tick\rangle X \quad (\dagger)}{Cl \vdash \langle tick\rangle \nu X.\,\langle tick\rangle X}\,(\langle - \rangle)}{Cl \vdash \nu X.\,\langle tick\rangle X \quad (\dagger)}\,(\nu)$$

This is a cyclic proof since the greatest fixed point ν is unfolded infinitely often on the cycle in the pre-proof.

Inductive definitions in first-order logic

- Consider these inductive definitions of predicates N, E, O:

$$\frac{}{N0} \quad \frac{Nx}{Nsx} \qquad\qquad \frac{}{E0} \quad \frac{Ex}{Osx} \quad \frac{Ox}{Esx}$$

- These definitions give rise to case-split rules, e.g., for N:

$$\frac{\Gamma, t = 0 \vdash \Delta \quad \Gamma, t = sx, Nx \vdash \Delta}{\Gamma, Nt \vdash \Delta} \text{(Case } N\text{)}$$

  where $x \notin FV(\Gamma \cup \Delta \cup \{Nt\})$.

- We call the formula Nx in the right-hand premise a case-descendant of Nt.

Example (1), à la Fermat

$$
\dfrac{\dfrac{\dfrac{}{\vdash E0, O0}\,(ER_1)}{z = 0 \vdash Ez, Oz}\,(=\text{L}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{Nz \vdash Oz, Ez \quad (\dagger)}{Ny \vdash Oy, Ey}\,(\text{Subst})}{Ny \vdash Oy, Osy}\,(OR_1)}{Ny \vdash Esy, Osy}\,(ER_2)}{z = sy, Ny \vdash Ez, Oz}\,(=\text{L})}{Nz \vdash Ez, Oz \quad (\dagger)}\,(\text{Case } N)
$$

- We can view this as a proof by infinite descent.
- If $Nz \vdash Ez, Oz$ was false then we would have:

$$Nz > Ny = Nz' > Ny' = Nz'' > Ny'' \ldots$$

Example (2), generalised infinite descent

$$
\dfrac{
\dfrac{
\dfrac{\dfrac{}{\vdash N0}\,(NR_1)}{x = 0 \vdash Nx}\,(=\text{L})
\qquad
\dfrac{\dfrac{\dfrac{Ox \vdash Nx \quad (\dagger 2)}{Oy \vdash Ny}\,(\text{Subst})}{Oy \vdash Nsy}\,(NR_2)}{x = sy, Oy \vdash Nx}\,(=\text{L})
}{Ex \vdash Nx \quad (\dagger 1)}\,(\text{Case } E)
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{Ex \vdash Nx \quad (\dagger 1)}{Ey \vdash Ny}\,(\text{Subst})}{Ey \vdash Nsy}\,(NR_2)}{x = sy, Ey \vdash Nx}\,(=\text{L})}{Ox \vdash Nx \quad (\dagger 2)}\,(\text{Case } O)
}{Ex \vee Ox \vdash Nx}\,(\vee\text{L})
$$

- Also a cyclic proof since Ox / Ex is unfolded infinitely often along the “figure-of-8” loop in the pre-proof.
- General principle: on every infinite path some inductive definition must be unfolded infinitely often.
- (Formal argument uses approximants of inductive predicates.)

Separation logic

- Separation logic uses extra connectives to reason about heap resource.
- emp denotes the empty heap.
- $F_1 * F_2$ expresses a division of the heap into two parts in which F_1 resp. F_2 hold.
- We can write inductive definitions as normal. E.g. we can define linked list segments $\mathtt{ls}\,x\,y$ by:

$$\frac{\mathsf{emp}}{\mathtt{ls}\,x\,x} \qquad\qquad \frac{x \mapsto x' * \mathtt{ls}\,x'\,y}{\mathtt{ls}\,x\,y}$$

where $\mapsto$ denotes a single-celled heap with domain x and contents x'.

Example (3): list segment concatenation

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{}{\mathtt{ls}\,x\,y \vdash \mathtt{ls}\,x\,y}\,(\mathrm{Id})
    }{\mathsf{emp} * \mathtt{ls}\,x\,y \vdash \mathtt{ls}\,x\,y}\,(\equiv)
  }{(x' = x \wedge \mathsf{emp}) * \mathtt{ls}\,x'\,y \vdash \mathtt{ls}\,x\,y}\,(=\mathrm{L})
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{}{x \mapsto z \vdash x \mapsto z}\,(\mathrm{Id})
      \quad
      \dfrac{(\dagger)\ {\color{blue}\mathtt{ls}\,x\,x'} * \mathtt{ls}\,x'\,y \vdash \mathtt{ls}\,x\,y}{{\color{blue}\mathtt{ls}\,z\,x'} * \mathtt{ls}\,x'\,y \vdash \mathtt{ls}\,z\,y}\,(\mathrm{Subst})
    }{x \mapsto z * {\color{blue}\mathtt{ls}\,z\,x'} * \mathtt{ls}\,x'\,y \vdash x \mapsto z * \mathtt{ls}\,z\,y}\,(*\mathrm{R})
  }{x \mapsto z * {\color{red}\mathtt{ls}\,z\,x'} * \mathtt{ls}\,x'\,y \vdash \mathtt{ls}\,x\,y}\,(\mathtt{ls}R_2)
}{(\dagger)\ {\color{red}\mathtt{ls}\,x\,x'} * \mathtt{ls}\,x'\,y \vdash \mathtt{ls}\,x\,y}\,(\mathrm{Case}\ \mathtt{ls})
$$

Again this is a cyclic proof since $\mathtt{ls}\,x\,x'$ is unfolded infinitely often on the loop in the pre-proof.

A Hoare proof system for termination

- Fix some program (in a simple imperative language):

$$1 : C_1, 2 : C_2, \ldots, n : C_n$$

- We write termination judgements $F \vdash_i \downarrow$ where i is a program label and F is a formula of separation logic.
- Intuitively, $F \vdash_i \downarrow$ means "the program always terminates when started at line i in a state satisfying F".
- As well as logical rules we have symbolic execution rules which capture program commands, e.g.:

$$\frac{Cond \wedge F \vdash_j \downarrow \quad \neg Cond \wedge F \vdash_{i+1} \downarrow}{F \vdash_i \downarrow} \; C_i \equiv \mathtt{if}\, Cond\, \mathtt{goto}\, j$$

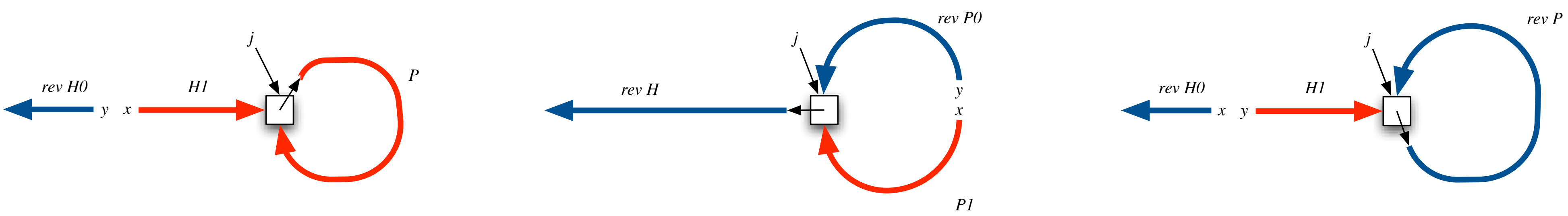
Reversing a "frying-pan" list

- The classical list reverse algorithm is:

1.	$y := \mathsf{nil}$	4.	$x := [x]$	7.	$\mathtt{goto}\,2$
2.	$\mathtt{if}\ x = \mathsf{nil}\ \mathtt{goto}\,8$	5.	$[z] := y$	8.	$\mathtt{stop}$
3.	$z := x$	6.	$y := z$		

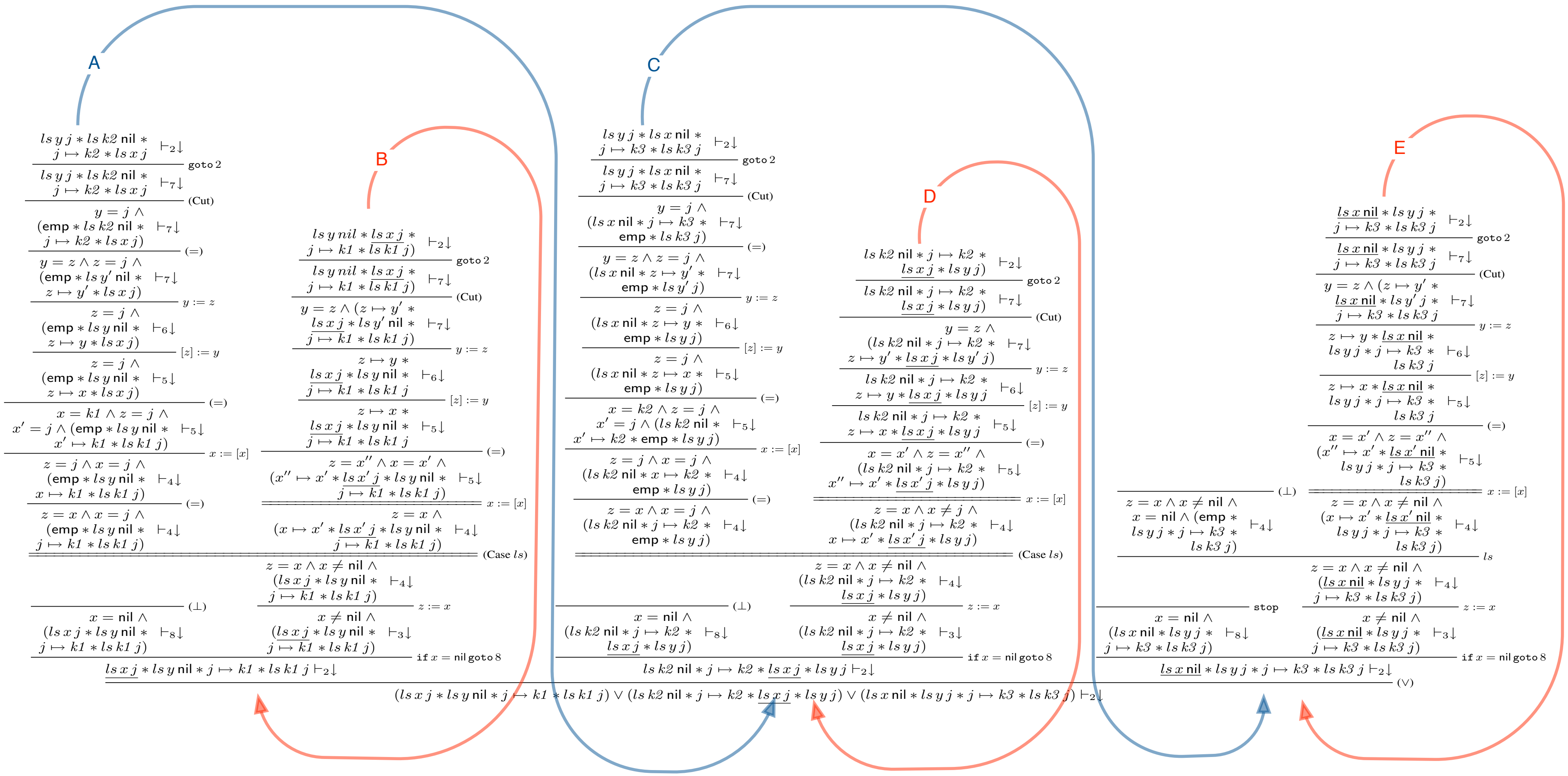
- The invariant for this algorithm given a cyclic list is:

$$\begin{aligned}&\exists k1, k2, k3\cdot\\ &(\mathtt{ls}\,x\,j * \mathtt{ls}\,y\,\mathsf{nil} * j \mapsto k1 * \mathtt{ls}\,k1\,j) \vee\\ &(\mathtt{ls}\,k2\,\mathsf{nil} * j \mapsto k2 * \mathtt{ls}\,x\,j * \mathtt{ls}\,y\,j) \vee\\ &(\mathtt{ls}\,x\,\mathsf{nil} * \mathtt{ls}\,y\,j * j \mapsto k3 * \mathtt{ls}\,k3\,j)\end{aligned}$$



- We want to prove that the invariant implies termination.

Reversing a "frying-pan" list — the cyclic proof



Why not just use induction?

- Cyclic proof typically subsumes proof by explicit induction.
- It allows us to delay the difficult choices in inductive proofs (inductive hypotheses, induction schema).
- Some parts of a proof can be left implicit (e.g., ranking functions for termination).
- It is often theoretically natural (e.g. because the generalisation to infinite trees gives a complete proof system).

Cyclic proof in the future?

- Extension of cyclic proof to work in more advanced program logics.
- Dealing with mixed inductive and coinductive definitions.
- Development as a vehicle for automated theorem proving.

Further reading

- C. Stirling and D. Walker.
  Local model checking in the modal μ-calculus.
  In *Theoretical Computer Science*, 1991.

- Cristoph Sprenger and Mads Dam.
  On the structure of inductive reasoning: circular and tree-shaped proofs in the μ-calculus.
  In *Proceedings of FOSSACS 2003.*

- James Brotherston and Alex Simpson.
  Complete sequent calculi for induction and infinite descent.
  In *Proceedings of LICS 2007.*

- James Brotherston.
  Formalised inductive reasoning in the logic of bunched implications.
  In *Proceedings of SAS 2007.*

- James Brotherston, Richard Bornat and Cristiano Calcagno.
  Cyclic proofs of program termination in separation logic.
  In *Proceedings of POPL 2008.*