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 (●):

```
  (Axiom)
     ●
  (Axiom)  (Axiom)
     ●      ●
  (Inference)
     ●
```

- Soundness of such proofs follows from the local soundness of each inference rule / axiom.
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 \( \mu \)-calculus properties of processes

Consider a “clock” process \( Cl \) which repeatedly ticks:

\[
Cl = \text{def } \text{tick}.Cl
\]

The \( \mu \)-calculus formula \( \nu X. \langle \text{tick} \rangle X \) means “the action ‘tick’ can be performed infinitely often”.

\[
\begin{align*}
Cl \vdash \nu X. \langle \text{tick} \rangle X & \quad (\dagger) \\
Cl \vdash \langle \text{tick} \rangle \nu X. \langle \text{tick} \rangle X & \quad (\langle - \rangle) \\
Cl \vdash \nu X. \langle \text{tick} \rangle X & \quad (\nu)
\end{align*}
\]

This is a cyclic proof since the greatest fixed point \( \nu \) is unfolded infinitely often on the cycle in the pre-proof.
Consider these inductive definitions of predicates $N, E, O$:

<table>
<thead>
<tr>
<th>$N$</th>
<th>$E$</th>
<th>$O$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_0$</td>
<td>$E_0$</td>
<td>$Ox$</td>
</tr>
<tr>
<td>$N_x$</td>
<td>$Ex$</td>
<td>$Osx$</td>
</tr>
<tr>
<td>$Nsx$</td>
<td>$Esx$</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

We call the formula $N_x$ in the right-hand premise a case-descendant of $N_t$. 
Example (1), à la Fermat

\[
\begin{align*}
\vdash & E_0, O_0 \quad (ER_1) \\
\vdash & z = 0 \quad (=L) \\
Nz & \vdash O_z, E_z \quad (†) \\
Ny & \vdash O_y, E_y \quad (OR_1) \\
Ny & \vdash O_y, Osy \quad (ER_2) \\
Ny & \vdash Esy, Osy \\
\vdash & z = sy, Nz \vdash E_z, O_z \quad (=L) \\
\vdash & Nz \vdash E_z, O_z \quad (†) \\
\end{align*}
\]

- We can view this as a proof by infinite descent.
- If \( Nz \vdash E_z, O_z \) was false then we would have:

\[
Nz > Ny = Nz' > Ny' = Nz'' > Ny'' \ldots
\]
Example (2), generalised infinite descent

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 `ls x y` by:

  \[
  \begin{array}{c}
  \text{emp} \\
  \hline
  \text{ls} \ x \ x \\
  \hline
  \text{ls} \ x \ y
  \end{array}
  \quad \text{where} \quad
  \begin{array}{c}
  x \mapsto x' \ast \text{ls} \ x' \ y
  \end{array}
  \]

where $\mapsto$ denotes a single-celled heap with domain $x$ and contents $x'$. 
Example (3): list segment concatenation

Again this is a cyclic proof since $\text{ls } x \ x'$ 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.:

\[
\begin{align*}
\text{Cond} \land F \vdash & j \downarrow \\
\neg \text{Cond} \land F \vdash & i+1 \downarrow \\
\hline
F \vdash i \downarrow & \quad C_i \equiv \text{if Cond goto } j
\end{align*}
\]
Reversing a “frying-pan” list

• The classical list reverse algorithm is:

1. \( y := \text{nil} \)
2. \( \text{if } x = \text{nil } \text{goto } 8 \)
3. \( z := x \)
4. \( x := [x] \)
5. \([z] := y \)
6. \( y := z \)
7. \text{goto } 2
8. \text{stop}

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

\[
\exists k1, k2, k3. \\
(\text{ls } x \text{ j } * \text{ls } y \text{ nil } * \text{j } \mapsto k1 * \text{ls } k1 \text{ j}) \vee \\
(\text{ls } k2 \text{ nil } * \text{j } \mapsto k2 * \text{ls } x \text{ j } * \text{ls } y \text{ j}) \vee \\
(\text{ls } x \text{ nil } * \text{ls } y \text{ j } * \text{j } \mapsto k3 * \text{ls } k3 \text{ j})
\]

• 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 $\mu$-calculus.

Cristoph Sprenger and Mads Dam.
On the structure of inductive reasoning: circular and tree-shaped proofs in the $\mu$-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*.