Cyclic Proofs of Program Termination

Richard Bornat, James Brotherston and Cristiano Calcagno

Simon Docherty

PPLV Cyclic Proof Reading Group

Tuesday 9th December

Pre-Talk Disclaimer

- Unlike the speakers in the previous two sessions, I am *not* presenting my own work today
- The paper is Richard Bornat, James Brotherston, Cristiano Calcalgno. Cyclic Proofs of Program Termination in Separation Logic, POPL 2008.
- Extensive further development by Reuben Rowe.

Pre-Talk Disclaimer

- Unlike the speakers in the previous two sessions, I am *not* presenting my own work today
- The paper is Richard Bornat, James Brotherston, Cristiano Calcalgno. Cyclic Proofs of Program Termination in Separation Logic, POPL 2008.
- Extensive further development by Reuben Rowe.
- .....so if I say something stupid about your work, stop me :)

This Talk

1. A short introduction to BI and Separation Logic;
2. A concrete example of inductive predicates and their relationship to cyclic proof (recall first session!);
3. A novel application of cyclic proof in computer science: program termination proofs.

Goal of the talk

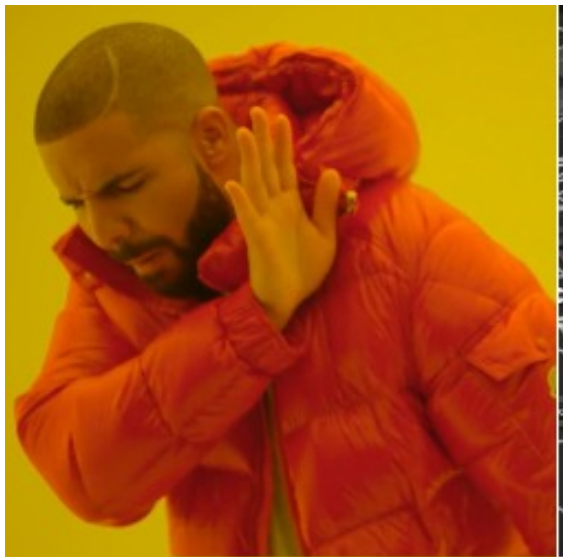
Make paper accessible for you.

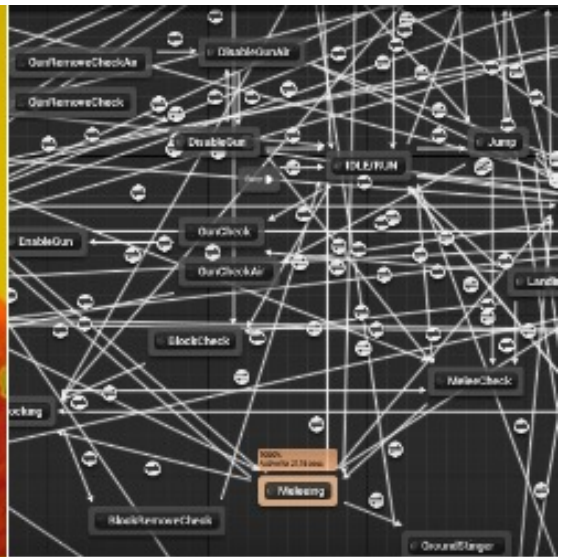
Hoare Logic and Heaps

Program Logics: The Basic Idea

‘Computer programming is an exact science in that all the properties of a program and the consequences of executing it in a given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning.’ Tony Hoare

(Floyd-)Hoare Logic: The Basic Idea





$$\{P\}C\{Q\}$$

Some Rules of Hoare Logic

$$\frac{}{\vdash \{P[E/x]\}x := E\{P\}}$$

Some Rules of Hoare Logic

$$\frac{\vdash \{P \wedge B\} C \{P\}}{\vdash \{P\} \text{while } B \text{ do } C \{\neg B \wedge P\}}$$

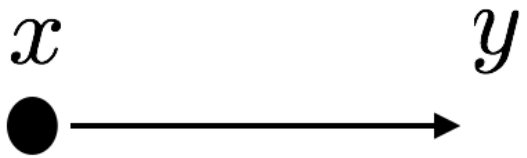
Rule of Constancy

$$\frac{\vdash \{P\}C\{Q\}}{\vdash \{P \wedge R\}C\{Q \wedge R\}}$$

where C does not assign to any free variables in R

Pointers and Heaps

- Pointers: special variables that store a memory address so contents can be referenced.



- The heap is the part of memory set aside for pointer declarations.
- Memory addresses can contain pointers ⇒ build mutable data structures in the heap

The Problem with Pointers

- Memory management in hands of programmers – powerful when used correctly, but cause of numerous bugs.
- Hoare logic doesn't handle them well.

$$\frac{\{\exists z.x \mapsto z\}[x] := 4\{x \mapsto 4\}}{\{y \mapsto 3 \wedge \exists z.x \mapsto z\}[x] := 4\{y \mapsto 3 \wedge x \mapsto 4\}}$$

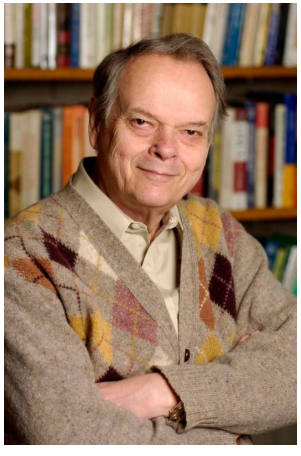
- Legal application of constancy - however invalid conclusion if x and y aliased!
- Clumsy fix: propagate anti-aliasing preconditions $x \neq y$. Scales badly.
- More sophisticated issues arise for mutable data structures (see Reynolds 2002).

Separation Logic

Diagnosing the problem

‘There is a mismatch between simple intuitions about the way that pointer operations work and the complexity of their axiomatic treatments. . . we suggest that the source of this mismatch is the global view of state taken in most formalisms for reasoning about pointers.’

Peter O’Hearn, John Reynolds and Hongseok Yang


From Classical Logic to...?

The Logic of Bunched Implications

Pym & O'Hearn 1999: a substructural logic with:

- Standard connectives $\wedge, \rightarrow, \top$ and resource-sensitive connectives $*, \mathrel{-\!\!*}, \mathrm{Emp}$;
- Kripke semantics on partial commutative monoids of resources $(X, \circ, e)$;
- Proof theory with tree-shaped contexts called bunches (two context formers, one associated to $\wedge$, one to $*$)

$$\langle \mathrm{R} \rightarrow \rangle \quad \frac{\Gamma; \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi} \qquad \langle \mathrm{R} \mathrel{-\!\!*} \rangle \quad \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \mathrel{-\!\!*} \psi}.$$

- Resource sensitivity: the comma context former does not satisfy contraction or weakening $\Rightarrow$ number and kind of formulae joined by commas matters.

A Stack and Heap Model of BI

- A stack s assigns program variables x to values and memory addresses $s : \mathrm{Var} \to \mathrm{Val} \cup \mathrm{Loc}$
- A heap h is an allocation of values to finitely many memory addresses $h : \mathrm{Loc} \rightharpoonup_{fin} \mathrm{Val}$
- If $dom(h) \cap dom(h') = \emptyset$, $h \circ h' \downarrow$ and $h \circ h' = h \sqcup h'$.
- [] is the empty heap (no heap memory allocated).
- $(\mathrm{Stack} \times \mathrm{Heap}, \circ, [])$ is a model of BI.
- This model gives an assertion language for Separation Logic.

Semantics of the Assertion Language

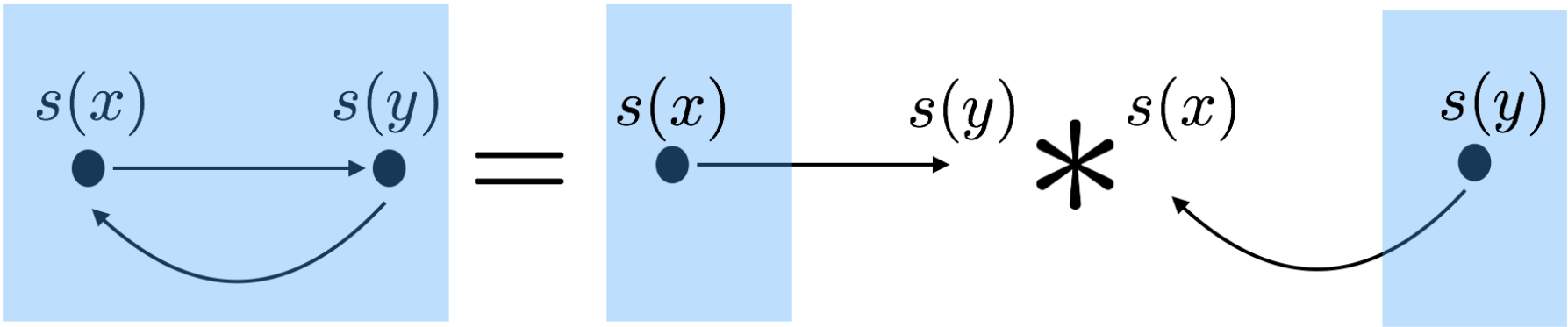
- $s, h \vDash E = E'$ iff $[\![E]\!]s = [\![E']\!]$
- $s, h \vDash \varphi \wedge \psi$ iff $s, h \vDash \varphi$ and $s, h \vDash \psi$
- $s, h \vDash \neg\varphi$ iff $s, h \nvDash \varphi$
- $s, h \vDash \exists x.\varphi(x)$ iff there exists t s.t. $s[x \mapsto t], h \vDash \varphi(t)$

Semantics of the Assertion Language

- $s, h \vDash E = E'$ iff $[\![E]\!]s = [\![E']\!]$
- $s, h \vDash \varphi \wedge \psi$ iff $s, h \vDash \varphi$ and $s, h \vDash \psi$
- $s, h \vDash \neg\varphi$ iff $s, h \nvDash \varphi$
- $s, h \vDash \exists x.\varphi(x)$ iff there exists t s.t. $s[x \mapsto t], h \vDash \varphi(t)$
- $s, h \vDash x \mapsto y$ iff $dom(h) = \{s(x)\}$ and $h(s(x)) = s(y)$
- $s, h \vDash \varphi * \psi$ iff $h = h' \circ h''$, $s, h' \vDash \varphi$ and $s, h'' \vDash \psi$
- $s, h \vDash \varphi \mathrel{-\!\!*} \psi$ iff $h \circ h' \downarrow$ and $s, h' \vDash \varphi$ implies $s, h \circ h' \vDash \psi$
- $s, h \vDash \mathrm{Emp}$ iff $h = []$

The Semantics, Pictorially

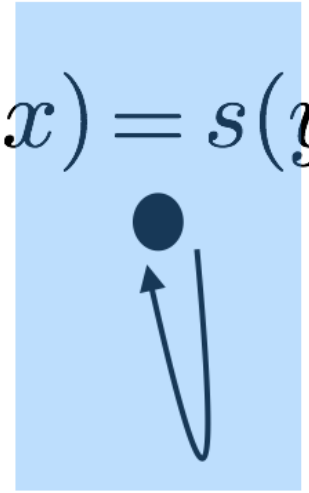
$$s, h \models x \mapsto y * y \mapsto x$$



The Semantics, Pictorially

$$s, h \nvDash x \mapsto y * y \mapsto x$$

$$s(x) = s(y)$$

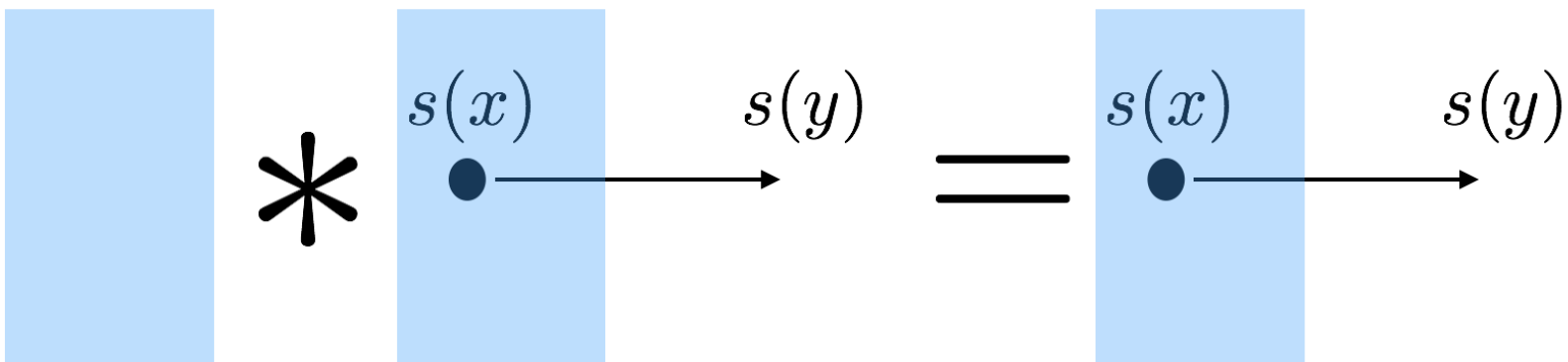


The Semantics, Pictorially

$$s, [] \models \mathrm{Emp}$$

The Semantics, Pictorially

$$s, [] \models x \mapsto y \twoheadrightarrow \neg \mathrm{Emp}$$



Small Axioms

- Axioms dealing with the basic pointer operations

$$\frac{}{\vdash \{\exists x.E \mapsto x\}[E] := F\{E \mapsto F\}}$$

$$\frac{}{\vdash \{\exists x.E \mapsto x\}dispose(E)\{\mathrm{Emp}\}}$$

The Frame Rule

- Separation Logic has a sound version of the rule of constancy, using $*$.

$$\frac{\vdash \{P\}C\{Q\}}{\vdash \{P * R\}C\{Q * R\}}$$

where C does not assign to any free variables in R

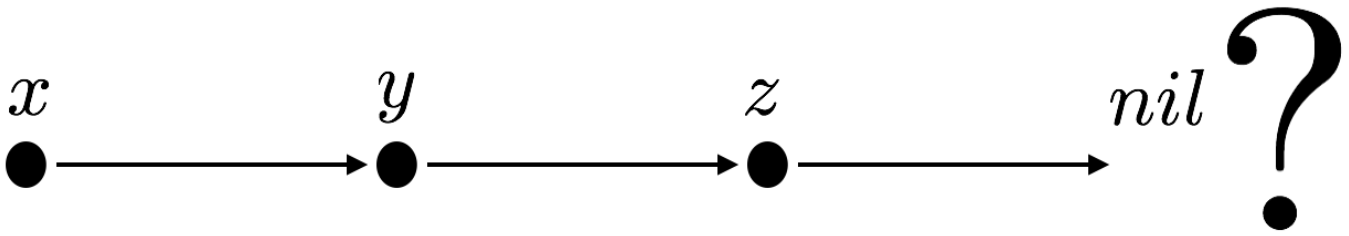
More Features of Separation Logic

- Tight interpretation: $\{P\}C\{Q\}$ valid implies C doesn't touch memory not mentioned in P.
- Compositionality: use frame rule to build procedure proof out of subprocedure proofs;
- Expressive decidable fragments;
- Scalability: all of the above + more (out of scope of talk) = industrial strength automated analysis `http://fbinfer.com`.

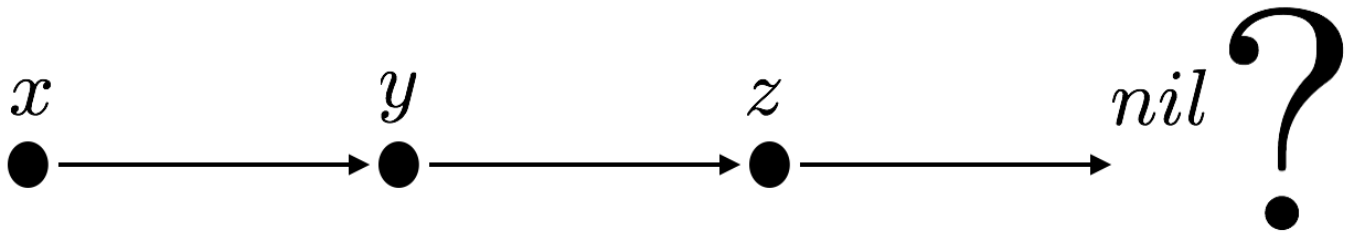
Half-time Summary

Inductive Predicates

What about those mutable data structures?



What about those mutable data structures?



- Most important use of pointers: defining mutable data structures.
- Assertion Language + Inductive Predicates $\Rightarrow$ assertions describing datatypes for mutable data structures.
- Want: $s, h \vDash \mathit{ls}\ x\ \mathit{nil}$ iff h is a nil-terminated linked list from $s(x)$.

Inductive Predicates, Formally

Definition (Inductive Definition)

An inductive definition of an inductive predicate symbol P is a finite set of production rules $C_i(\vec{x}) \Rightarrow Pt_i(\vec{x})$, where C_i is built according to:

$$\begin{aligned} C ::= \quad & Pt(\vec{x}) \mid \hat{\varphi}(\vec{x}) \mid C(\vec{x}) \wedge C(\vec{x}) \mid C(\vec{x}) * C(\vec{x}) \mid \\ & \hat{\varphi}(\vec{x}) \rightarrow C(\vec{x}) \mid \hat{\varphi}(\vec{x}) \mathrel{-\!\!*} C(\vec{x}) \mid \forall x C(\vec{x}) \end{aligned}$$

- $\hat{\varphi}$: any predicate BI formula not involving inductive predicate symbols.
- Simplified slightly here (P can range over all inductive predicate symbols usually)

A Simple Example: List Segments

- Base case: the empty list is considered an empty list segment from any x to x:

$$\mathrm{Emp} \Rightarrow \mathit{ls}\ \mathit{xx}$$

$$x$$

A Simple Example: List Segments

- Base case: the empty list is considered an empty list segment from any x to x:

$$\text{Emp} \Rightarrow ls\ xx$$

$$x$$

- Inductive step: list segment + separately in memory a pointer to its startpoint is a list segment:

$$x \mapsto x' * ls\ x'y \Rightarrow ls\ xy$$

Interpreting an Inductive Predicate

- P an inductive predicate with production rules $C_1(\vec{x_1}) \Rightarrow P\vec{t_1}(\vec{x_1}), \ldots, C_k(\vec{x_k}) \Rightarrow P\vec{t_k}(\vec{x_k})$.

$$\Phi_P(X) = \bigcup_{1 \leq j \leq k} \{(h, [\![\vec{t_j}(\vec{d})]\!]) \mid s[\vec{x_j} \mapsto \vec{d}], h \vDash_{P \mapsto X} C_j(\vec{x_j})\}$$

Interpreting an Inductive Predicate

- P an inductive predicate with production rules $C_1(\vec{x_1}) \Rightarrow P\vec{t_1}(\vec{x_1}), \ldots, C_k(\vec{x_k}) \Rightarrow P\vec{t_k}(\vec{x_k})$.

$$\Phi_P(X) = \bigcup_{1 \leq j \leq k} \{(h, [\![\vec{t_j}(\vec{d})]\!]) \mid s[\vec{x_j} \mapsto \vec{d}], h \vDash_{P \mapsto X} C_j(\vec{x_j})\}$$

- For each production rule $C_j(\vec{x_j}) \Rightarrow P\vec{t_j}(\vec{x_j})$...

Interpreting an Inductive Predicate

- P an inductive predicate with production rules $C_1(\vec{x_1}) \Rightarrow P\vec{t_1}(\vec{x_1}), \ldots, C_k(\vec{x_k}) \Rightarrow P\vec{t_k}(\vec{x_k})$.

$$\Phi_P(X) = \bigcup_{1 \leq j \leq k} \{(h, [\![\vec{t_j}(\vec{d})]\!]) \mid s[\vec{x_j} \mapsto \vec{d}], h \vDash_{P \mapsto X} C_j(\vec{x_j})\}$$

- For each production rule $C_j(\vec{x_j}) \Rightarrow P\vec{t_j}(\vec{x_j})$...
- The tuples: heap h together with evaluated terms from the consequent $P\vec{t_j}(\vec{x_j})$...

Interpreting an Inductive Predicate

- P an inductive predicate with production rules $C_1(\vec{x_1}) \Rightarrow P\vec{t_1}(\vec{x_1}), \ldots, C_k(\vec{x_k}) \Rightarrow P\vec{t_k}(\vec{x_k})$.

$$\Phi_P(X) = \bigcup_{1 \leq j \leq k} \{(h, [\![\vec{t_j}(\vec{d})]\!]) \mid s[\vec{x_j} \mapsto \vec{d}], h \vDash_{P \mapsto X} C_j(\vec{x_j})\}$$

- For each production rule $C_j(\vec{x_j}) \Rightarrow P\vec{t_j}(\vec{x_j})$...
- The tuples: heap h together with evaluated terms from the consequent $P\vec{t_j}(\vec{x_j})$...
- Such that for any stack s that evaluates the terms that way...

Interpreting an Inductive Predicate

- P an inductive predicate with production rules $C_1(\vec{x_1}) \Rightarrow P\vec{t_1}(\vec{x_1}), \ldots, C_k(\vec{x_k}) \Rightarrow P\vec{t_k}(\vec{x_k})$.

$$\Phi_P(X) = \bigcup_{1 \leq j \leq k} \{(h, [\![\vec{t_j}(\vec{d})]\!]) \mid s[\vec{x_j} \mapsto \vec{d}], h \vDash_{P \mapsto X} C_j(\vec{x_j})\}$$

- For each production rule $C_j(\vec{x_j}) \Rightarrow P\vec{t_j}(\vec{x_j})$...
- The tuples: heap h together with evaluated terms from the consequent $P\vec{t_j}(\vec{x_j})$...
- Such that for any stack s that evaluates the terms that way...
- s, h satisfies the antecedent $C_j(\vec{x_j})$ when P is interpreted as X

Interpreting an Inductive Predicate

- P an inductive predicate with production rules $C_1(\vec{x_1}) \Rightarrow P\vec{t_1}(\vec{x_1}), \ldots, C_k(\vec{x_k}) \Rightarrow P\vec{t_k}(\vec{x_k})$.

$$\Phi_P(X) = \bigcup_{1 \leq j \leq k} \{(h, [\![\vec{t_j}(\vec{d})]\!]) \mid s[\vec{x_j} \mapsto \vec{d}], h \vDash_{P \mapsto X} C_j(\vec{x_j})\}$$

- For each production rule $C_j(\vec{x_j}) \Rightarrow P\vec{t_j}(\vec{x_j})$...
- The tuples: heap h together with evaluated terms from the consequent $P\vec{t_j}(\vec{x_j})$...
- Such that for any stack s that evaluates the terms that way...
- s, h satisfies the antecedent $C_j(\vec{x_j})$ when P is interpreted as X

$$\Phi_{ls}(X) = \{([], (v, v) \mid v \in \mathrm{Val}\} \cup \{(h_1 \circ h_2, (v, v')) \mid \exists w \in Val. h_1 = \{(v, w)\} \text{ and } (h_2, (w, v')) \in X\}$$

Least Fixed Point by Approximation

- Φ_P is monotone on the lattice of predicates $\Rightarrow$ has a least fixed point.
- Define lfp interpretation for P by ordinal induction:

$$\Phi_P^0 ::= \Phi_P(\vec{\emptyset})$$
$$\Phi_P^{\alpha+1} ::= \Phi_P(\Phi_P^\alpha)$$
$$\Phi_P^\lambda ::= \bigcup_{\beta<\lambda} \Phi_P^\beta$$

- $[\![P]\!] = \bigcup_\alpha \Phi_P^\alpha$: $s, h \vDash P\vec{t}(\vec{x})$ iff $(h, [\![\vec{t}]\!]) \in [\![P]\!]$.
- Ordinal approximations are why the cyclic system works!

Cyclic Proofs of Program Termination

A Simple Example: Linked-List Traversal

$$
\begin{array}{rl}
\text{Cond} ::= & E = E \mid E \neq E \\
C ::= & x := E \mid x := [E] \mid [E] := E \mid x := \mathit{new}() \mid \mathit{free}(E) \mid \\
& \text{if Cond goto } i \mid \text{stop}
\end{array}
$$

A Simple Example: Linked-List Traversal

$$\begin{aligned} \text{Cond} ::= &\quad E = E \mid E \neq E \\ C ::= &\quad x := E \mid x := [E] \mid [E] := E \mid x := \mathit{new}() \mid \mathit{free}(E) \mid \\ &\quad \text{if Cond goto } i \mid \text{stop} \end{aligned}$$

Consider the following program

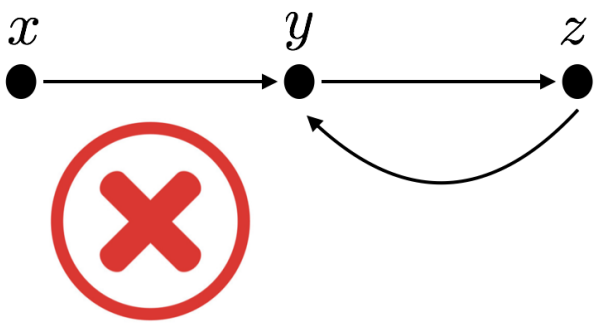
1. if x = nil goto 4; 2. $x := [x]$; 3. goto 1; 4. stop

A Simple Example: Linked-List Traversal

$$\begin{aligned} \text{Cond} ::= \quad & E = E \mid E \neq E \\ C ::= \quad & x := E \mid x := [E] \mid [E] := E \mid x := \mathit{new}() \mid \mathit{free}(E) \mid \\ & \text{if Cond goto } i \mid \text{stop} \end{aligned}$$

Consider the following program

1. if x = nil goto 4; 2. $x := [x]$; 3. goto 1; 4. stop

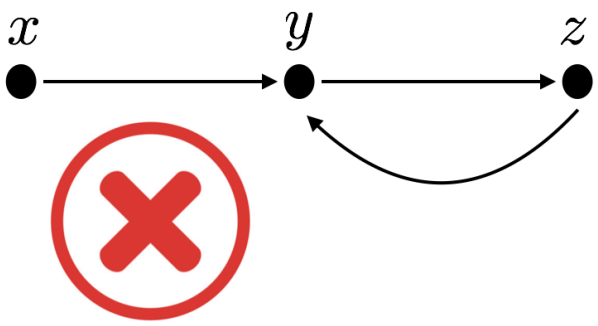


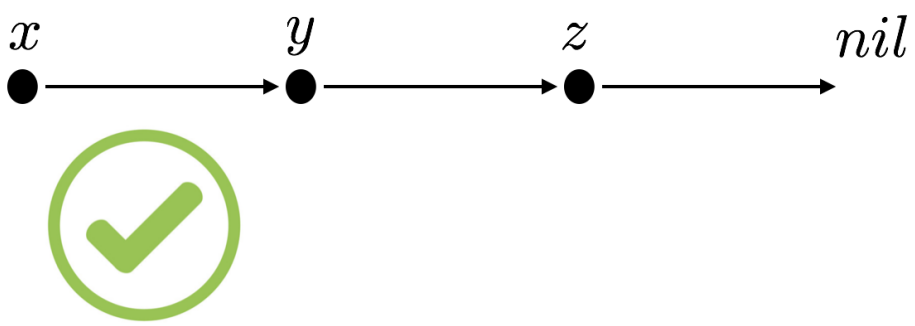
A Simple Example: Linked-List Traversal

$$\begin{aligned} \text{Cond} ::= &\quad E = E \mid E \neq E \\ C ::= &\quad x := E \mid x := [E] \mid [E] := E \mid x := \mathit{new}() \mid \mathit{free}(E) \mid \\ &\quad \text{if Cond goto } i \mid \text{stop} \end{aligned}$$

Consider the following program

1. if x = nil goto 4; 2. $x := [x]$; 3. goto 1; 4. stop





The Key Idea

- Given program C, $(i, s, h) \downarrow$ means C will terminate if state is (s, h) on i-th instruction.
- Previous slide: termination could be deduced from the shape of the heap.

The Key Idea

- Given program C, $(i, s, h) \downarrow$ means C will terminate if state is (s, h) on i-th instruction.
- Previous slide: termination could be deduced from the shape of the heap.
- Idea: given program C prove that formula F is such that $(s, h) \vDash F$ implies C terminates if started in state (s, h).
- How to prove? With a cyclic system!

Road Map

1. Judgements $\Gamma \vdash_i \downarrow$: where bunch $\Gamma \equiv$ assertion about heap;

Road Map

1. Judgements $\Gamma \vdash_i \downarrow$: where bunch $\Gamma \equiv$ assertion about heap;
2. Proof rules symbolically execute program;

Road Map

1. Judgements $\Gamma \vdash_i \downarrow$: where bunch $\Gamma \equiv$ assertion about heap;
2. Proof rules symbolically execute program;
3. Path in proof graph $\equiv$ execution trace

Road Map

1. Judgements $\Gamma \vdash_i \downarrow$: where bunch $\Gamma \equiv$ assertion about heap;
2. Proof rules symbolically execute program;
3. Path in proof graph $\equiv$ execution trace
4. Cycle $\Rightarrow$ program has got back to same instruction with a heap with same shape again;

Road Map

1. Judgements $\Gamma \vdash_i \downarrow$: where bunch $\Gamma \equiv$ assertion about heap;
2. Proof rules symbolically execute program;
3. Path in proof graph $\equiv$ execution trace
4. Cycle $\Rightarrow$ program has got back to same instruction with a heap with same shape again;
5. Unfolding an inductive predicate infinitely often = progressing through heap description = infinite descending chain of approximations $\Rightarrow$ that infinite execution trace is contradictory.

Road Map

1. Judgements $\Gamma \vdash_i \downarrow$: where bunch $\Gamma \equiv$ assertion about heap;
2. Proof rules symbolically execute program;
3. Path in proof graph $\equiv$ execution trace
4. Cycle $\Rightarrow$ program has got back to same instruction with a heap with same shape again;
5. Unfolding an inductive predicate infinitely often = progressing through heap description = infinite descending chain of approximations $\Rightarrow$ that infinite execution trace is contradictory.
6. $\exists$ cyclic proof $\Rightarrow$ there are only finite execution traces $\Rightarrow$ program terminates.

A Snapshot of the System: Execution Rules

$$\frac{Cond; \Gamma \vdash_j \downarrow \qquad \neg Cond; \Gamma \vdash_{i+1} \downarrow}{\Gamma \vdash_i \downarrow} C_i \equiv \text{if Cond goto } j$$

$$\frac{E_0 \mapsto E_1, \Gamma \vdash_{i+1} \downarrow}{E_0 \mapsto t, \Gamma \vdash_i \downarrow} C_i \equiv [E_0] := E_1$$

$$\frac{}{\Gamma \vdash_i \downarrow} C_i \equiv \text{stop}$$

Example of Inductive Predicate Rules

- Brotherston & Simpson: schema for turning production rules into a case-split rule for a cyclic system.
- Occurence of an inductive predicate $\Rightarrow$ unfold into antecedents of production rules.
- For list segment predicate:

$$\frac{\Gamma(t = u; \mathsf{emp}) \vdash_i \downarrow \quad \Gamma(t \mapsto x' * ls\, x'\, u) \vdash_i \downarrow}{\Gamma(ls\, t\, u) \vdash_i \downarrow}\ (\text{Case } ls)$$

Snapshot Continued: Logical Rules

- Logical rules are based on BI's bunched sequent calculus.
- Key difference: multiplicative weakening holds for termination judgements:

$$\frac{\Gamma(\Delta) \vdash_i \downarrow}{\Gamma(\Delta, \Delta') \vdash_i \downarrow}\ \textbf{(WkM)}$$

- Why?

Snapshot Continued: Logical Rules

- Logical rules are based on BI's bunched sequent calculus.
- Key difference: multiplicative weakening holds for termination judgements:

$$\frac{\Gamma(\Delta) \vdash_i \downarrow}{\Gamma(\Delta, \Delta') \vdash_i \downarrow}\ \textbf{(WkM)}$$

- Why?

Proposition (Termination Montonocity)

If $(i, s, h) \downarrow$ *and* $h \circ h'$ *defined, then* $(i, s, h \circ h') \downarrow$ □

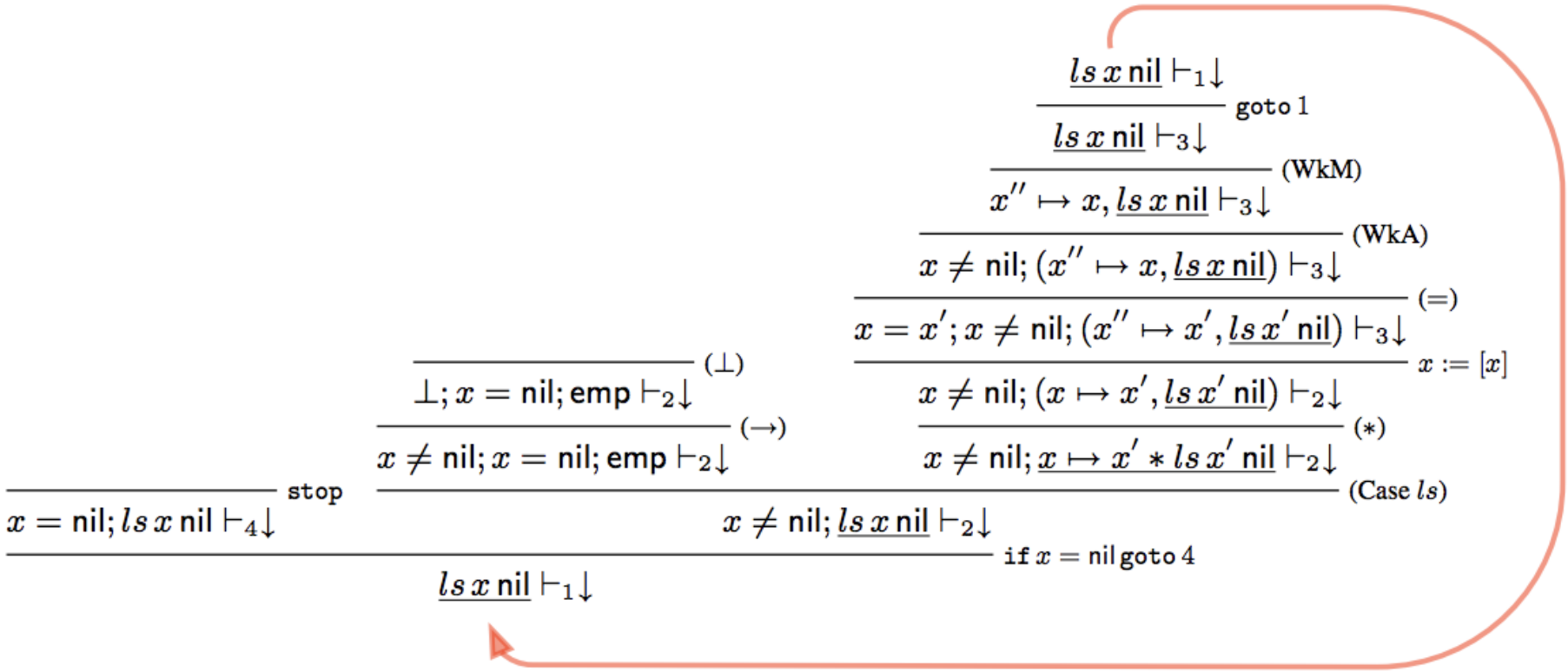
Soundness Condition

- Cycles aren't always sound in cyclic systems: need a correctness condition.
- Trace: sequence of formulae through preproof graph meeting some basic (but lengthy) requirements
- Infinitely progressing trace: infinite trace on which inductive predicates are unfolded infinitely often.

Definition (Proof)

Preproof in which every infinite path is followed by an infinitely progressing trace.

Cyclic Proof of Linked-List Traversal Termination



Why does it work?

- Key property is soundness: if $F \vdash_i \downarrow$ then for all (s, h), $s, h \vDash F$ implies $(i, s, h) \downarrow$.

Why does it work?

- Key property is soundness: if $F \vdash_i \downarrow$ then for all (s, h), $s, h \vDash F$ implies $(i, s, h) \downarrow$.
- Suppose proof of $\Gamma \vdash_i \downarrow$ but exists (s, h) such that $s, h \vDash \Gamma$ and $(i, s, h) \uparrow$.

Why does it work?

- Key property is soundness: if $F \vdash_i \downarrow$ then for all (s, h), $s, h \vDash F$ implies $(i, s, h) \downarrow$.
- Suppose proof of $\Gamma \vdash_i \downarrow$ but exists (s, h) such that $s, h \vDash \Gamma$ and $(i, s, h) \uparrow$.
- Along each infinite path: infinite sequence of (s_i, h_i) invalidating each sequent $\Gamma_i \vdash_j \downarrow$ in trace.

Why does it work?

- Key property is soundness: if $F \vdash_i \downarrow$ then for all (s, h), $s, h \vDash F$ implies $(i, s, h) \downarrow$.
- Suppose proof of $\Gamma \vdash_i \downarrow$ but exists (s, h) such that $s, h \vDash \Gamma$ and $(i, s, h) \uparrow$.
- Along each infinite path: infinite sequence of (s_i, h_i) invalidating each sequent $\Gamma_i \vdash_j \downarrow$ in trace.
- $s, h \vDash_{P \mapsto P^\alpha} \Gamma$ if s, h satisfies Γ when Φ_P^α substituted for $[\![P]\!]$

Why does it work?

- Key property is soundness: if $F \vdash_i \downarrow$ then for all (s, h), $s, h \vDash F$ implies $(i, s, h) \downarrow$.
- Suppose proof of $\Gamma \vdash_i \downarrow$ but exists (s, h) such that $s, h \vDash \Gamma$ and $(i, s, h) \uparrow$.
- Along each infinite path: infinite sequence of (s_i, h_i) invalidating each sequent $\Gamma_i \vdash_j \downarrow$ in trace.
- $s, h \vDash_{P \mapsto P^\alpha} \Gamma$ if s, h satisfies Γ when Φ_P^α substituted for $[\![P]\!]$
- Infinite sequence (α_i): least such α for which $s_i, h_i \vDash_{P \mapsto P^\alpha} \Gamma_i$

Why does it work?

- Key property is soundness: if $F \vdash_i \downarrow$ then for all (s, h), $s, h \vDash F$ implies $(i, s, h) \downarrow$.
- Suppose proof of $\Gamma \vdash_i \downarrow$ but exists (s, h) such that $s, h \vDash \Gamma$ and $(i, s, h) \uparrow$.
- Along each infinite path: infinite sequence of (s_i, h_i) invalidating each sequent $\Gamma_i \vdash_j \downarrow$ in trace.
- $s, h \vDash_{P \mapsto P^\alpha} \Gamma$ if s, h satisfies Γ when Φ_P^α substituted for $[\![P]\!]$
- Infinite sequence (α_i): least such α for which $s_i, h_i \vDash_{P \mapsto P^\alpha} \Gamma_i$

Proposition

(α_i) is an infinite decreasing sequence. At predicate unfolding steps, strictly decreasing. □

Why does it work?

- Key property is soundness: if $F \vdash_i \downarrow$ then for all (s, h), $s, h \vDash F$ implies $(i, s, h) \downarrow$.
- Suppose proof of $\Gamma \vdash_i \downarrow$ but exists (s, h) such that $s, h \vDash \Gamma$ and $(i, s, h) \uparrow$.
- Along each infinite path: infinite sequence of (s_i, h_i) invalidating each sequent $\Gamma_i \vdash_j \downarrow$ in trace.
- $s, h \vDash_{P \mapsto P^\alpha} \Gamma$ if s, h satisfies Γ when Φ_P^α substituted for $[\![P]\!]$
- Infinite sequence (α_i): least such α for which $s_i, h_i \vDash_{P \mapsto P^\alpha} \Gamma_i$

Proposition

(α_i) is an infinite decreasing sequence. At predicate unfolding steps, strictly decreasing. □

- Contradiction!

Pros and Cons

Pros:

- Termination measures handled implicitly by system (versus explicit definition)
- Implementable (and it has been!)
- Pretty cool!

Pros and Cons

Pros:

- Termination measures handled implicitly by system (versus explicit definition)
- Implementable (and it has been!)
- Pretty cool!

Cons:

- No magic: no decision procedure for any class of programs
- Where to find candidate heap description? (addressed in Brotherston and Gorogiannis - Cyclic Abduction of Inductively Defined Safety and Termination Preconditions)
- Need to compile programs into indexed instructions

Where next?

- Linked-list traversal example is v basic – more sophisticated examples in the paper!
- Substantially built upon in Reuben N.S. Rowe and James Brotherston. Automatic Cyclic Termination Proofs for Recursive Procedures in Separation Logic. CPP 2017.
- In that paper: cyclic proofs of total correctness: i.e. termination guarantee and postconditions; implementation.

Conclusions

Conclusions

- Separation Logic = Hoare logic variant with a substructural assertion language describing spatial properties of memory, together with rules for (de)allocation and framing.
- Good qualities: assertion language semantics handles aliasing, facilitates local reasoning, decision procedures for expressive fragments $\Rightarrow$ scalable (`http://fbinfer.com`).
- Natural use of cyclic proof: shape properties defined via inductive predicates $\Rightarrow$ reason about entailment using cyclic systems.
- Novel application of cyclic proof: proofs of termination for programs that are fed the right heap.

Further Reading

- Pym, O'Hearn. The Logic of Bunched Implications. 1999.
- Ishtiaq, O'Hearn. BI as an Assertion Language for Mutable Data Structures. 2001.
- Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures 2001.
- Brotherston. Formalised Inductive Reasoning in the Logic of Bunched Implications. 2007.
- Bornat, Brotherston, Calcagno. Cyclic Proofs of Program Termination in Separation Logic. 2008.
- Rowe, Brotherston. Automatic Cyclic Termination Proofs for Recursive Procedures in Separation Logic. 2017.