Separation Logic
Expressiveness and Copyless Message-Passing

Étienne Lozes

ENS Cachan

july 3rd, 2012
Decades of research on improving programming languages, the introduction of references and mutable data structures, efficient garbage collectors, the abolition of null pointers... and here we are! Separation Logic!

Network Spy in EDF Nuclear Plants

- collects measurements on a network and outputs a synthesis for human supervision.
- low-level software, requires C language
- around 10,000 lines of code.

“semi-critical” software
- should not fault (in particular due to an heap error)
- should run an arbitrarily long time without reboot
- should be validated (EDF buys the code to some other company)
A Question of Leaks

- pipe-lined concurrent program
- threads communicating via FIFO buffers
- respects EDF constraints:
  - buffers and threads are allocated statically
  - no recursive data structure

- problem: program crashed after two years without reboot
  - memory leakage suspected
  - confirmed in ANR Averiles (Arnaud Sangnier, TOPICS tool)
How to Prevent Heap Errors

- restrictions on programs
  - solution adopted by EDF (e.g. forbid lists)
  - not always sensible (e.g. buffers encode lists)
  - program analysis may become simpler... or harder!

- formal proof
  - the best solution if programmers were familiar with it
  - may be hard to prove the program for someone else
  - at least, could quickly become tedious for anyone

- static analysis
  - significant progresses were obtained during the last 5 years
  - incompleteness (30% of proved code = 0% of safe code)
  - press button... but which button?
Formal Proof of Programs

- insert annotations
  precondition, postcondition, loop invariant, ...

- insert ghost code (when needed)

- check Hoare triples \( \{\phi\} p \{\psi\} \)
  - \( p \) does not fault starting from any state satisfying \( \phi \)
  - it terminates in a state satisfying \( \psi \), if it terminates
  - and optionally: terminates, does not deadlock, does not leak memory,...
An Example Proof

cell *revappend(cell *x, cell *y) {
    cell *z;

    while (x != NULL) {
        z = x->next;
        x->next = y;
        y = x;
        x = z;
    }
    return y;
}
An Example Proof

{\text{list}(x, \text{null}) \land \text{list}(y, \text{null})}

\text{cell} *\text{revappend}(\text{cell} *x, \text{cell} *y)\
\begin{align*}
    &\text{cell} *z; \\
    &\{\text{list}(x, \text{null}) \land \text{list}(y, \text{null})\} \\
    &\text{while} (x!=\text{NULL}) \{ \\
    &\quad z = x->\text{next}; \\
    &\quad x->\text{next} = y; \\
    &\quad y = x; \\
    &\quad x = z; \\
    &\} \\
    &\text{return} y;
\end{align*}

\{\text{list}(y, \text{null})\}
An Example Proof with Errors

{list(x, null) \land list(y, null)}
cell *revappend(cell *x, cell *y) {
    cell *z;
    {list(x, null) \land list(y, null)}
    while (x!=NULL) {
        z = x->next;
        x->next = y;
        y = x;
        x = z;
    }
    return y;
}
{list(y, null)}
design choices

- the class of programs
- the program logic
- the degree of automation
design choices
  ▶ the class of programs
  ▶ the program logic
  ▶ the degree of automation

this thesis:
  ▶ concurrent programs
design choices
  ▶ the class of programs
  ▶ the program logic
  ▶ the degree of automation

this thesis:
  ▶ concurrent programs
  ▶ separation logic
design choices

- the class of programs
- the program logic
- the degree of automation

this thesis:

- concurrent programs
- separation logic
- message-passing
design choices

- the class of programs
- the program logic
- the degree of automation

this thesis:

- concurrent programs
- separation logic
- message-passing
- communication errors
Architecture of Provers

annotated code

VC Gen

verification conditions

symbolic execution / WP

entailment problems

theorem prover
Architecture of Provers

- annotated code
  - VC Gen
    - verification conditions
      - symbolic execution / WP
        - entailment problems
          - theorem prover
            - expressiveness issue
            - complexity issue
Architecture of Provers

annotated concurrent code

VC Gen

verification conditions

symbolic execution / WP

expressiveness issue

entailment problems

theorem prover

complexity issue
Architecture of Provers

annotated concurrent code

\[ \text{VC Gen} \]

modularity issue

\[ \text{verification conditions} \]

\[ \text{symbolic execution / WP} \]

expressiveness issue

\[ \text{entailment problems} \]

\[ \text{theorem prover} \]

complexity issue
Architecture of Provers

- annotated concurrent code
- separation logic
- modularity issue
- expressiveness issue
- complexity issue

VC Gen

verification conditions

symbolic execution / WP

entailment problems

theorem prover
Architecture of Provers

- annotated concurrent code
- VC Gen
- verification conditions
- symbolic execution / WP
- entailment problems
- theorem prover
- separation logic
- modularity issue
- expressiveness issue
- complexity issue

proof theory

model theory
Outline

1. contributions to the model theory of SL

2. contributions to the proof theory of SL

3. perspectives
1.
Contributions to the Model Theory of Separation Logic
An Idealized Memory Model

- no pointer arithmetic
- only list structures

- store: $s : \text{Var} \rightarrow \text{Loc} + \mathbb{R}$
- heap: $h : \text{Loc} \rightarrow_{\text{fin}} \text{Loc} \times \mathbb{R}$

\[
s(x) = a \]
\[
h = \{ a \mapsto b, 3.14 \ \ b \mapsto c, 0.57 \} \]
Heap Decomposition

**Definition**

\[ h = h_1 \cdot h_2 \quad \text{if} \quad \begin{cases} h = h_1 \cup h_2, \\ \text{dom } h_1 \cap \text{dom } h_2 = \emptyset \end{cases} \]

**Example**

\[ \{a \mapsto b, 3.14 ; b \mapsto c, 0.57\} = \]

Diagram:

- 3.14 (a)
- 0.57 (b)
- c

Diagram shows the heap decomposition with nodes and mappings.
Heap Decomposition

Definition

\[ h = h_1 \bullet h_2 \quad \text{if} \quad \begin{cases} h = h_1 \cup h_2, \\
\text{dom } h_1 \cap \text{dom } h_2 = \emptyset \end{cases} \]

Example

\[ \{a \mapsto b, 3.14 ; b \mapsto c, 0.57\} = \{a \mapsto b, 3.14\} \]
Heap Decomposition

Definition

\[ h = h_1 \circ h_2 \quad \text{if} \quad \begin{cases} 
  h = h_1 \cup h_2, \\
  \text{dom } h_1 \cap \text{dom } h_2 = \emptyset 
\end{cases} \]

Example

\[
\{ a \mapsto b, 3.14 \ ; \ b \mapsto c, 0.57 \} = \\
\{ a \mapsto b, 3.14 \} \circ \{ b \mapsto c, 0.57 \}
\]
Separating Conjunction [Reynolds, LICS 2002]

\[ \phi \# \psi \]

\[ s, h \models \phi_1 \# \phi_2 \]

if there is \( h_1, h_2 \) s.t.

- \( h = h_1 \bullet h_2 \), and
- \( s, h_1 \models \phi \), and
- \( s, h_2 \models \psi \)
- \( s, h \models \text{emp} \)
Separating Conjunction [Reynolds, LICS 2002]

$s, h \models \phi_1 * \phi_2$
if there is $h_1, h_2$ s.t.

- $h = h_1 \cdot h_2$, and
- $s, h_1 \models \phi$, and
- $s, h_2 \models \psi$
- $s, h \models \text{emp}$
Separating Conjunction [Reynolds, LICS 2002]

\[ \exists \phi \quad \text{s.t.} \quad s, h \models \phi_1 \ast \phi_2 \]

if there is \( h_1, h_2 \) s.t.

- \( h = h_1 \cdot h_2 \), and
- \( s, h_1 \models \phi \), and
- \( s, h_2 \models \psi \)
- \( s, h \models \text{emp} \)
Separating Conjunction \cite{Reynolds, LICS 2002}

\[ \exists s, h \models \phi_1 \ast \phi_2 \]

if there is \( h_1, h_2 \) s.t.

\[ \begin{align*}
& \quad h = h_1 \bullet h_2, \text{ and} \\
& \quad s, h_1 \models \phi, \text{ and} \\
& \quad s, h_2 \models \psi \\
& \quad s, h \models \text{emp}
\end{align*} \]
Separating Conjunction  [Reynolds, LICS 2002]

\[ \exists s, h : \phi_1 \ast \phi_2 \]

if there is \( h_1, h_2 \) s.t.

- \( h = h_1 \bullet h_2 \), and
- \( s, h_1 \models \phi \), and
- \( s, h_2 \models \psi \)
- \( s, h \models \text{emp} \)
Separating Conjunction [Reynolds, LICS 2002]

\[ \exists s, h \models \phi_1 \ast \phi_2 \]
if there is \( h_1, h_2 \) s.t.

- \( h = h_1 \bullet h_2 \), and
- \( s, h_1 \models \phi \), and
- \( s, h_2 \models \psi \)
- \( s, h \models \text{emp} \)

\[ s, h \models \text{emp} \]
if \( h = \emptyset \)
Examples

- expressing non-aliasing
  \[(x \mapsto -) \ast (y \mapsto -) \models x \neq y\]

- valid Hoare triples
  \[
  \{\phi\} x = \text{new()} \{\ast (x \mapsto -) \ast \exists x.\phi\}
  \{\phi \ast (x \mapsto -)\} \text{dispose}(x) \{\phi\}
  \{\text{list}(x, \text{null}) \ast \text{list}(y, \text{null})\} \text{revappend}(x, y) \{\text{list}(y, \text{null})\}
  \]

- still not a calculus of weakest precondition
  \[
  \{??\} x = \text{new()} \{\phi\}
  \{??\} x \rightarrow \text{next} = y \{\phi\}
  \]
Magic Wand \cite{IshtiaqO01}

**Magic Wand** \( (\rightarrow*) \)

\[
s, h \models \phi \rightarrow* \psi \quad \text{if} \\
\forall h', \text{ if } h \bullet h' \text{ is defined} \\
\text{and if } s, h' \models \phi, \\
\text{then } (s, h \bullet h') \models \psi
\]
**Magic Wand** [Ishtiaq & O’Hearn, POPL 2001]

**Magic Wand** (→*)

∀ φ →∗ ψ

\[ s, h \models φ \rightarrow∗ ψ \text{ if } \]

∀ h', if \( h \bullet h' \) is defined

and if \( s, h' \models φ \),

then \( (s, h \bullet h') \models ψ \)
Magic Wand \cite{Ishtiaq:POPL-01}

\textbf{Magic Wand} \((-\star\)

\[
\forall \phi, \phi \rightarrow \star \psi \quad \psi
\]

\[s, h \models \phi \rightarrow \star \psi \quad \text{if}\]

\[\forall h', \text{ if } h \bullet h' \text{ is defined and if } s, h' \models \phi, \text{ then } (s, h \bullet h') \models \psi\]
**Magic Wand** [Ishtiaq & O’Hearn, POPL 2001]

\[\forall s, h \models \phi \rightarrow^* \psi \quad \text{if} \]

\[\forall h', \text{ if } h \cdot h' \text{ is defined and if } s, h' \models \phi, \text{ then } (s, h \cdot h') \models \psi\]

\[A \rightarrow^\circledast B \triangleq \neg (A \rightarrow \neg B)\]
Magic Wand [Ishtiaq & O’Hearn, POPL 2001]

**Magic Wand (−∗)**

\[ \forall s, h \models \phi \dashv \psi \text{ if } \forall h', \text{ if } h \ast h' \text{ is defined and if } s, h' \models \phi, \text{ then } (s, h \ast h') \models \psi \]

**Septraction (−⊗)**

\[ \exists s, h \models \phi \dashv \psi \text{ if } \exists h', h \ast h' \text{ is defined and } s, h' \models \phi, \text{ and } (s, h \ast h') \models \psi \]

\[ A \dashv \otimes B \triangleq \neg (A \dashv \neg B) \]
**Magic Wand** [Ishtiaq & O’Hearn, POPL 2001]

**Magic Wand** ($\to\ast$)

$$\forall s, h \models \phi \to\ast \psi \text{ if }$$

$$\forall h', \text{ if } h \cdot h' \text{ is defined and if } s, h' \models \phi, \text{ then } (s, h \cdot h') \models \psi$$

**Septraction** ($\to\ominus$)

$$\exists s, h \models \phi \to\ominus \psi \text{ if }$$

$$\exists h', \text{ if } h \cdot h' \text{ is defined and } s, h' \models \phi, \text{ and } (s, h \cdot h') \models \psi$$

$$A \to\ominus B \triangleq \neg (A \to \neg B)$$
A First Elimination Result

**Theorem** [Lozes, SPACE 2004]

For every quantifier-free formula of SL, there is an equivalent formula without connectives \{\ast, \rightarrow \ast\}. 
Promises of \(\star\) Elimination

useful for symbolic execution and WP computation

- atomic commands

  examples:
  \[
  \{\forall x. (x \mapsto) \star \phi \}\text{ } x := \text{new()} \{\phi\}
  \]
  \[
  \{(x \mapsto) \star ((x \rightarrow y) \star \phi)\}\text{ } x \rightarrow \text{next} = y \{\phi\}
  \]

- function calls

  assuming \(\{\phi\} f(\vec{x}) \{\psi\}\)

    - \(\{\zeta\} f(\vec{x}) \{\psi \star (\phi \circ \star \zeta)\}\) ("frame inference" problem)
    - \(\{\phi \star (\psi \rightarrow \zeta)\} f(\vec{x}) \{\zeta\}\) ("abduction" problem)

if \(\star\) could be eliminated, these problems would have a simple solution.

**Problem:** the result holds for **quantifier free** formulas only.
What Happened Next

- Yang reported that the result did not extend to quantified formulas for tree-like heaps.

- *symbolic heaps* were adopted ($\sim SL\{\neg, \rightarrow\}$)
  [Berdine, Calcagno & O’Hearn, FSTTCS 2004]

- an “almost complete” symbolic execution for symbolic heap was discovered
  [Berdine, Calcagno & O’Hearn, APLAS 2005]

- $\rightarrow\leftarrow$-elimination showed up once more in RGSEP
  [Calcagno, Parkinson & Vafeiadis, SAS 2007]

- ... and then anyone agreed that SL with magic wand was “extremely complicated”
The Challenging Questions

Question 1

Is there an elimination property for $\to^*$ for SL over lists?

Question 2

If yes, is it effective?

Question 3

Is this logic decidable?
Some Comments

▶ decidability of SL without magic over lists without data
  ▶ FO(*) reduces to MSO(f)
  ▶ MSO(f) is decidable (Rabin’s theorem)

(1) and (2) implie (3)

▶ for similar logics
  ▶ →∗ could be eliminated [Lozes, EXPRESS 04]
  ▶ but not effectively [Conforti& Ghelli, CONCUR 04]

there was a chance that (1) does not implie (2)

→ MSO(f) is strictly more expressive than FO(*)
[Antonopoulos & Dawar, FOSSACS 2009]
Brochenin’s PhD Main Result

**Theorem** [Brochenin, Demri & Lozes, CSL 2008]

$$SL = SO = FO(\neg \ast)$$

consequences:

- $\ast$ cannot be eliminated
- $\ast$ can
- SL is undecidable even on lists without data.

**remark:** “bounded” magic wand is still decidable

[Brochenin, Demri & Lozes, Information and Computation 2011]
Symbolic Heaps

- de facto, the reasonable fragment of SL.

- complexity of the entailment problem
  - initially shown in co-NP [Berdine, Calcagnon & O’Hearn, FSTTCS 2004]
  - recently proved in P [Haase & al, CONCUR 2011]

- still significant improvements are needed
  - restricted to lists (almost)
  - few known about abduction [Gorogiannis, Kanovitch & O’Hearn, SAS 2011]
  - no support for data
  - very few known about permissions
Expressiveness Hierarchy

$SL_{LS} \sim FO_{LS}(\rightarrow) \sim SO_{LS}$

MSO_{LS} \sim FO_{LS}(\star) \sim FO_{LS} \sim \text{Symb.Heaps}

undecidable
non-elem.

PTIME
## Elimination Theorems

<table>
<thead>
<tr>
<th></th>
<th>$SL_{prop}$</th>
<th>$FO_{LS}(\ast)$</th>
<th>$FO_{LS}(\rightarrow)$</th>
<th>$SL_{LS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ast$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$\ast$</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><code>list(.,.)</code></td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\exists$ MSO</td>
<td>no</td>
<td>[AD, 09]</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\exists$ SO</td>
<td>no</td>
<td>[AD, 09]</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

[Lozes, SPACE 04] [Lozes, TCS 05] [BDL, CSL 08] [BDL, I& C 11]

→ **Spatial Graph Logic** [Dawar, Ghelli, and Gardner, I& C 2007]
Dealing with Data [Bansal, Brochenin & Lozes, FOSSACS 2009]

- problem: expressing sortedness or multiset relations, e.g.
  - some reverse function transforms a sorted list into a sorted list
  - some merge function performs multi-set union

- FO is already undecidable over lists with data

- we introduced a decidable fragment that can express sortedness

- but not multiset relations.

at the same time:

- “Composite Shape Logic”
  [Bouajjani, Dragoi, Enea & Sighireanu, CONCUR 2009]

- tool support: CELIA
  [Bouajjani, Dragoi, Enea & Sighireanu, CAV 2011]
Dealing with Time [Brochenin, Demri & Lozes, APAL 09]

extending SL to temporal logic
- expressing progress properties
- expressing memory shapes as “explorations”

our proposals
- LTL with SL as state logic
- LTL+SL with arbitrary nesting of time/space connectives

our contribution: complexity of satisfiability and model-checking
- for list-like heaps
- for pointer arithmetic
2. Contributions to the Proof Theory of Separation Logic
Proof System of SL

**If**

\[
\{ \phi \land b \} \ p \ \{ \psi \} \quad \{ \phi \land \neg b \} \ p' \ \{ \psi \}
\]

\[
\{ \phi \} \quad \text{if } b \text{ then } p \text{ else } p' \ \{ \psi \}
\]

**While**

\[
\{ \phi \land b \} \ p \ \{ \phi \}
\]

\[
\{ \phi \} \quad \text{while } b \text{ do } p \ \{ \phi \land \neg b \}
\]

**Sequential**

\[
\{ \phi \} \ p \ \{ \chi \} \quad \{ \chi \} \ p' \ \{ \psi \}
\]

\[
\{ \phi \} \quad p ; p' \ \{ \psi \}
\]

**Frame**

\[
\{ \phi \} \ p \ \{ \psi \} \quad \text{mv}(p) \cap \text{fv}(\phi_F) = \emptyset
\]

\[
\{ \phi \land \phi_F \} \ p \ \{ \psi \land \phi_F \}
\]

**Consequence**

\[
\phi \vdash \phi' \quad \{ \phi' \} \ p \ \{ \psi' \} \quad \psi' \vdash \psi
\]

\[
\{ \phi \} \ p \ \{ \psi \}
\]

**AVE**

\[
\{ \phi \} \ p \ \{ \psi \} \quad x \notin \text{fv}(p)
\]

\[
\{ \exists x. \phi \} \ p \ \{ \exists x. \psi \}
\]
Soundness and Completeness

**Soundness:** if $\{\phi\} p \{\psi\}$ is derivable, then it is valid.
[Yang, phd thesis, 2001]

Assume now $p$ is a sequential, uni-procedural, program.

**Completeness:** if $\{\phi\} p \{\psi\}$ is valid, then it is derivable.
[Lozes, unpublished]

- this is a consequence of $\text{SL} = \text{SO}$
- infinite disjunctions are not needed.
- frame Rule and AVE rule can be eliminated
Open Problems on Completeness

consider SL for sequential programs with functions
  ▶ frame rule cannot be eliminated
  ▶ but AVE rule?
  ▶ completeness? without information-hiding?
    [O’Hearn, Reynolds & Yang, POPL 04]
    [Birkedal, Torps-Smith & Yang, LICS 05] [Pottier, LICS 08]

consider now concurrent SL without function calls
  ▶ AVE rule cannot be eliminated
  ▶ but frame rule?
  ▶ completeness? without ghost code?
Concurrency and Separation Logic

based on the parallel rule [O’Hearn, CONCUR 2004]

various forms of synchronisation

- conditional critical sections (≈ Hoare monitors)
- locks in the heap
  [Gotsman, phd thesis, 2008]
- non-blocking concurrency and atomicity
  [Vafeiadis, phd thesis, 2007]

- message passing?
Villard’s phD: Modeling Sing#

Singularity: a research project and an operating system.

- *no memory protection*: all processes share the same address space
- memory isolation is verified at compile time (Sing# language)
- processes communicate by message passing
- claimed
  - *race freedom* (e.g. process isolation)
  - *deadlock freedom*
- mentioned that some checks are done at runtime
Sing# Communication Model

- channels are *bidirectional* and *asynchronous*
  channel = pair of FIFO queues

- channel is determined by a pair of *endpoints*
  similar to socket model

- endpoints may be passed through channels
  similar to mobility in π-calculus

- communications are ruled by *communication contracts*
  similar to session types
Our Four Message-Passing Primitives

\[(e, f) = \text{open()}\]  \quad \text{channel allocation}
\[\text{close}(e, f)\]  \quad \text{channel disposal}
\[\text{send}(m, e, x_1, \ldots, x_n)\]  \quad \text{message emission}
\[(x_1, \ldots, x_n) = \text{receive}(m, e)\]  \quad \text{message reception}

\text{where} \; m \; \text{is a “message descriptor”}

Example

\begin{verbatim}
local e,f in
  (e,f)=open();
  send(string,e,x);
  y = receive(string,f);
  close(e,f);
\end{verbatim}

\[\approx y = x;\]  \[\not\approx y = \text{strcpy}(x);\]
What Local Reasoning Misses

\{ x \mapsto \} \\
\text{orphan\_message}(x) \{ \\
  (e,f) := \text{open}(); \\
  \{ x \mapsto *e \mapsto f \mapsto f \mapsto e \} \\
  \text{send}(\text{cell},e,x); \\
  \{ e \mapsto f \mapsto f \mapsto e \} \\
  \text{close}(e,f); \\
\} \\
\{ \text{emp} \}

- channel’s content should be tracked
- not local: may require assume guarantee reasoning
- we can avoid it!
Communication Contracts

- goal: force endpoints to be used \textit{symmetrically}

- solution:
  1. describe protocol as a finite state automaton
  2. check $p$ follows $C$ on one endpoint
  3. check $p$ follows the symmetric of $C$ on the other endpoint
Issues with Sing# Communication Contracts

imprecision in Sing# documentation
  ▶ mentions contract should be deterministic... why?
  ▶ what kind of deadlocks do contracts rule out?

bug reports on two contracts
  ▶ two contracts are not “realizable”, acknowledged problematic
    [Stengel & Bultan, ISTTA 2009]
  ▶ hint: these are the only two contracts with mixed choice

questions left for us
  ▶ what are communication contracts?
  ▶ how do we integrate contracts in SL?
Various Forms of “Deadlocks”

circular_wait() {
  (e, f) := open();
  {
    receive(m1,e); [..]
    ||
    receive(m2,f); [..]
  }
}

head_to_head() {
  (e, f) := open();
  {
    send(m1,e); [..]
    ||
    send(m2,f); [..]
  }
}

orphan_message(x) {
  (e, f) := open();
  send(cell,e,x);
  close(e,f);
}

unspecified_reception(x) {
  (e, f) := open();
  send(string,e,x);
  y=receive(int,f);
}

buffer_overflow(x,y) {
  (e, f) := open();
  // buffer_size = 1 msg
  send(cell,e,x);
  send(cell,e,y);
}
Our Theory of Communication Contracts

two definitions: a syntactic one and a semantic one

▶ $det(C)$ has no mixed choice
▶ $C$ prevents head-to-head deadlocks
aka half-duplex property [Cece & Finkel, CAV 97]

**Theorem** [Lozes & Villard, WS-FM 2011]

1. definitions match
2. they implie absence of unspecified reception
3. they implie realizability
4. unspecified receptions are undecidable in general
5. if moreover in a contract $C$ there are
   ▶ no cycle of ! transitions
   ▶ no cycle of ? transitions
then $C$ prevents orphan messages and ensures buffer boundedness
Embedding Communication Contracts into SL

[Villard, Lozes & Calcagno, APLAS 2009]

the endpoint predicate $e^{ep}(C\{q\}, f)$:

- $e$ is an endpoint
- $e$ is ruled by contract $C$
- $e$ is currently in contract state $q$
- $e$’s peer is $f$

extra rules for message-passing primitives:

- allocation ensures peer endpoints to be in the initial state
- disposal requires peer endpoints to be in a same final state
- communications obey with the contract and update the state
Provable Programs Do Not Go Wrong

Properties induced by local reasoning
- no memory violation
- no data race
- confluent computations

Properties induced by communication contracts
as said before
no unspecified receptions, no orphan messages, guaranteed buffer bound,
no head-to-head “deadlocks”

Tool Heap-Hop [Villard, Lozes & Calcagno, TACAS 2010]

Extension shared contract-obedient endpoints
[Lozes & Villard, ICE 2012]
And Memory Leaks?

we first claimed so...

counter-example [Bono, Messa & Padovani, ESOP 2010]

{emp}
mem_leak_generator(){
   local e,f;
   (e,f) = open(C);
   send(endpoint,e,f);
   send(close_me,e,e);
}
{emp}

▶ what’s wrong: sending f towards itself
▶ forbidding just that is not enough
▶ forbid to send an endpoint in a receive state
   ~ locality condition for $\pi$-calculus [Merro, phd thesis, 2000]
   [Gardner, Laneve & Wischik, I& C 07]
▶ relaxations are possible [Villard, phD thesis 2011]
3. Perspectives
Open Problems in Proof Theory

- completeness of concurrent SL
- role of ghost code
- conciseness of SL with respect to Hoare logic
  - theoretical comparison
  - empirical comparison
- deadlocks
  - completeness of existing proof theories
    [Leino & Müller, ESOP 2010]
  - progress contracts?
Information as Resource

- broadly: use local reasoning for information flow analysis

\[
\{\text{rand}(x)\}
\]
\[
y = \text{random}();
\]
\[
\{\text{rand}(x) \ast \text{rand}(y)\}
\]
\[
\text{send}(x);
\]
\[
\{\text{rand}(y)\}
\]
// y is still ‘‘secret’’

- a first target: completely prove an oblivious transfer
- problem to be solved: how to share information (permissions?)
- other models of information are interesting:
  - symbolic model (applied-π frames)
  - shadow semantics

[Lozes & Villard, CONCUR 2008, LMCS 2010] [JLTV, TOSCA 2010]
[Lozes, FCS-PrivMod 2010]
SL and Session Types

- communication contracts vs session types
  - seems “almost” clear
  - not formalized yet

- typing vs proving

- permissions vs multiparty

- beyond communications
  - objects
  - other synchronization primitives
    e.g. barrier [Hobor, ESOP 2010]
Local Reasoning for MPI

- no shared memory in MPI

- still, ownership transfers in non-blocking primitives

  ```c
  char s[100];
  non_blocking_receive(s);
  // loses ownership of s
  wait();
  // gets back ownership of s
  print(s);
  ```

- communication contracts?
  - no notion of peers
  - multiparty communication contracts could help
Tool Development

limitations of **HEAP-HOP**

- an invented programming language
- simple memory model
- hardly reusable

ongoing: **CHEAP**, a **FRAMA-C** plugin

- proofs of concurrent C code with locks in the heap
- permissions
- collaboration with other plugins (e.g. **CELIA**)
Three Issues on SL Tools for the Next Five Years

1. a common platform for all SL-based tools
   - a first attempt: coreStar
   - far less modular than WHY [Filliâtre & Marche, CAV 2007]
   - always new target languages
     → Javascript [Gardner, Mafeis, Smith, POPL 07]

2. interfacing SL-based proofs with other program logics
   - to the best of my knowledge: nothing similar exists
   - my goal: learn something from interfacing with CELIA

3. develop bug-free tools
   - fault-free (→ Space Inv. bug [Habermehl et al, CAV 2012])
   - sound [Appel & Blazy, TPHOL 2007] [Appel, CPP 2011]
2025?

what will still be there
  - nuclear plants
  - bugs in software

and probably for a long, long time...

what I wish I will say
  - all of my undergraduate students have experienced computer-checked program proofs
  - my phD student wrote a proof of EDF’s network spy in one day and 200 lines.
Thanks!
Merci!
Grazie!