Towards Information as Resource in Separation Logic

Ongoing work

Étienne Lozes[1,2]

LSV, ENS Cachan, CNRS

MOVES, RWTH Aachen University

Example: Communicating Cells

```
x = cons(nil);
y = cons(x);
send(Right, y);
x->next= 3
```

```
z = receive(Left);
dispose(z)
```

Safety: resumes to establish that x and z are non-aliased

Informal proof:

- x and y hold two disjoint cells
- *logically*, left-to-right transfer of the cell y to z (but not x!)
- x and z are thus *non-aliased*.

Example: Communicating Cells (2)

$\{emp\}$	
`x = cons(nil);`	
$\{x \mapsto nil\}$	
`y = cons(x);`	
$\{y \mapsto x * x \mapsto nil\}$	$\{emp\}$
`send(Right, y);`	`z = receive(Left);`
$\{x \mapsto nil\}$	$\{z \mapsto -\}$
`x->next= 3`	`dispose(z)`
$\{x \mapsto 3\}$	$\{emp\}$

Example: Noised Communication

```
new x = random_int();
new y = random_int();
send(Right, x+y)
```

$\|$

```
z = receive(Left)
```

Privacy requirement: $P(x = n_1 | z = n_2) = P(x = n_1)$

Informal proof:

- x and y hold two disjoint informations
- *logically*, left-to-right transfer of the information of y to z
- x and z are thus *independent*.

Example: Noised Communication (2)

$\{\mathsf{emp}\}$	
`new x = random_int();`	
$\{rand(x)\}$	
`new y = random_int();`	
$\{rand(x) * rand(y)\}$	
`send(Right, x+y)`	
$\{rand(x)\}$	
	$\{\mathsf{emp}\}$
	`z = receive(Left)`
	$\{rand(z)\}$

Outline

1. A Model for Message-Passing Probabilistic Programs
2. Independence-Separation Logic
3. Permissions and Sharing
4. Proof of an Oblivious Transfer

Message-Passing Probabilistic Programs

A Toy Programming Language

$$
\begin{array}{lcll}
E & ::= & x \in \mathsf{Var} \mid n \in \mathsf{Int} \mid E + E \mid E - E \mid E(+)E & \text{(int expression)} \\
b & ::= & E = E' \mid b \ \textbf{and} \ b \mid \textbf{not} \ b & \text{(bool expression)} \\
c & ::= & \textbf{new} \ \mathsf{x} = \mathsf{E} \mid \textbf{new} \ \mathsf{x} = \mathsf{random\_int}() & \text{(command)} \\
 & & \mid \textbf{assume}(b) & \\
 & & \mid \textbf{send}(\mathsf{u},\mathsf{m},x_1,\ldots,x_n) & \\
 & & \mid (x_1,\ldots,x_n)\text{=}\textbf{receive}(\mathsf{u},\mathsf{m}) & \\
 & & c;\ c \mid c \| c \mid c + c &
\end{array}
$$

where

- u is a *channel endpoint*
- m is a *message identifier*
- machine integers $[0, \ldots, N-1]$, addition modulo N.

Oblivious Transfer

Initial state

- A database server owns data[0],..,data[N]
- A client owns i

Final state

- The database does not learn i
- The client learns data[i], but nothing more.

The Whole Protocol

```
PROTOCOL () {
AUTHORITY() || CLIENT(i) || SERVER(data)
}
```

The Authority

sends a fresh random array r to the server, and a random machine integer c and r[c] to the client. The common knowledge of the client and the server is thus r[c].

```
AUTHORITY() {
  new r = random_array();
  new c = random_int();
  new rc = r[c];
  send(Serv,msg_r,r)
  send(Cl,msg_rc,c,rc)
}
```

Note: the authority does not learn anything.

The Client

sends i noised with c to the server, then receives data noised with r, and extracts data[i].

```
CLIENT(i) {
  (c,rc) = receive(Aut,msg_rc);
  new s = i+c;
  send(Serv,msg_s,s);
  noise = receive(Serv,msg_noise,noise);
  new result  = noise[i] (+) rc;
}
```

The Server

```
SERVER(data) {
  r = receive(Aut,msg_r);
  s = receive(Cl,msg_s);
  for n=0 to N-1 do
    new noise[n] =
      data[n] (+) r[s-n]
  done;
  send(Cl,msg_noise);
}
```

Desirable Properties

- **adequacy**: if all agents are fair, the client learns data[i]

$$\{\ldots\}\ \text{Client}\,(\,\text{i}\,)\ \{result = data[i] * \ldots\}$$

- **privacy**: the client keeps i secret, and the server keeps data[j] secret for all j≠i.

$$\{rand(i)\}\ \text{Client}\,(\,\text{i}\,)\ \{rand(i) * \ldots\}$$

$$\{rand(data)\}\ \text{Server}(\text{data})\ \{\exists i.rand(data \backslash i) * \ldots\}$$

Objective of this talk: Focus on adequacy : similar to the standard notion of correctness in SL.

Privacy is about *competitive* concurrency, different from the concurrency treated in SL.

Independence-Separation Logic

Randomized States

Deterministic States:

$\Sigma_d = Var \rightharpoonup Int$ (partial functions)

Randomized States

$Rand(\Sigma_d)$: probability distributions for the discrete σ-algebra

e.g. functions $\hat{\sigma} : \Sigma_d \rightarrow [0, 1]$ of the form

$$\sum_{i=1\ldots n} p_i . \delta_{\sigma_i}$$

with $\sum_{i=1\ldots n} p_i = 1$

Separation Algebras

Definition[O'Hearn& al]
A separation algebra is a structure $\Sigma = (\Sigma, \bullet)$ such that: $\bullet$ is a partial composition, associative, commutative, and *cancellative*: if $\sigma_1 \bullet \sigma = \sigma_2 \bullet \sigma$, then $\sigma_1 = \sigma_2$.

Example: $\Sigma_d = \mathit{Var} \rightharpoonup \mathit{Val}$, $\bullet$ is $\uplus$.

Compatibility: $\sigma \perp \sigma'$ if $\sigma \bullet \sigma'$ is defined

Division pre-order: $\sigma \leq \sigma'$ if $\sigma' = \sigma \bullet \sigma''$.

Randomization Functor

Fix $(\Sigma, \bullet)$ a SA.

$Rand(\Sigma, \bullet_r)$ is a SA with

- $\hat{\sigma} \perp_r \hat{\sigma}'$ if $\hat{\sigma} \perp \hat{\sigma}$ almost surely.
- $P(\hat{\sigma} \bullet_r \hat{\sigma}' = \sigma_0) =$

$$\sum_{\sigma \bullet \sigma' = \sigma_0} P(\hat{\sigma} = \sigma).P(\hat{\sigma}' = \sigma')$$

Assertion Language

Syntax

$$A, B, \ldots \quad ::= \quad \text{emp} \mid x \mapsto i \mid e = e' \mid A \vee B \mid A * B \mid \sum_{i \in Int} p_i : x \mapsto i * A_i.$$

Semantics

$\hat{\sigma} \vDash \text{emp}$	if $dom(\sigma) = \emptyset$ almost surely
$\hat{\sigma} \vDash x \mapsto i$	if $dom(\sigma) = \{x\}$ and $\hat{\sigma}(x) = i$ almost surely
$\hat{\sigma} \vDash e = e'$	if $[[e]]\hat{\sigma} = [[e']]\hat{\sigma}$ almost surely
$\hat{\sigma} \models A * B$	if there are $\hat{\sigma_1}, \hat{\sigma_2}.\hat{\sigma_1} \bullet \hat{\sigma_2} = \hat{\sigma}$ & $\hat{\sigma_1} \models A$ & $\hat{\sigma_2} \models B$
$\hat{\sigma} \vDash \sum_{i \in Int} p_i : x \mapsto i * A_i.$	if $P(x = i) = p_i$ & $\hat{\sigma}\vert_{x=i} \vDash x \mapsto i * A_i$

Standard Proof Rules in SL

$$\frac{\{A\}\ c\ \{B\}}{\{A * F\}\ c\ \{B * F\}}\ (Frame) \qquad \frac{\{A\}\ c\ \{B\} \qquad \{A'\}\ c'\ \{B'\}}{\{A * A'\}\ c||c'\ \{B * B'\}}\ (||)$$

$$\frac{\{A\}\ c\ \{B\} \qquad \{A'\}\ c\ \{B'\}}{\{A \vee A'\}\ c\ \{B \vee B'\}}\ (\vee)$$

$$\frac{A' \Rightarrow A \qquad \{A\}\ c\ \{B\} \qquad B \Rightarrow B'}{\{A'\}\ c\ \{B'\}}\ (\Rightarrow)$$

$$\frac{\{A\}\ c\ \{A'\} \qquad \{A'\}\ c'\ \{B\}}{\{A\}\ c; c'\ \{B\}}\ (;) \qquad \frac{\{A\}\ c_1\ \{B\} \qquad \{A\}\ c_2\ \{B\}}{\{A\}\ c_1 + c_2\ \{B\}}\ (+)$$

Proof Rules for Randomness

$$\frac{\text{for all } i \in I \qquad \{x \mapsto i * A_i\}\ p\ \{x \mapsto i * B_i\} \qquad p \Downarrow}{\{\sum_{i \in Int} p_i : x \mapsto i * A_i\}\ p\ \{\sum_{i \in I} p_i : x \rightarrow i * B_i\}}\ (Rand)$$

$$\{b\}\ \text{assume(b)}\ \{b\}$$

$$\{\text{emp}\}\ \text{new x = random\_int()}\ \{rand(x)\}$$

$$\{y \mapsto i * z \mapsto j\}\ \text{new x= E(y,z)}\ \{x \mapsto E(i,j) * y \mapsto i * z \mapsto j\}$$

Random array generators

$\{\text{emp}\}$ rand_array(a) $\{rand(a[0]) * \cdots * rand(a[N-1])\}$

- `new a[0]=rand_int();...; new a[N-1]=rand_int()`
- `new a[0]=rand_int() ||...|| new a[N-1]=rand_int()`

Random access to an array:

$$\dfrac{\dfrac{\dfrac{\{x \mapsto i * a \mapsto -\}\ \text{new y} = \text{a[i]}\ \{x \mapsto i * a \mapsto - * y \mapsto a[i]\}}{\{x \mapsto i * a \mapsto -\}\ \text{assume(x=i);new y} = \text{a[i]}\ \{x \mapsto i * a \mapsto - * y \mapsto a[i]\}}}{\{x \mapsto i * a \mapsto -\}\ \text{new y} = \text{a[x]}\ \{x \mapsto i * a \mapsto - * y \mapsto a[i]\}}}{\{rand(x) * a \mapsto -\}\ \text{new y} = \text{a[x]}\ \{rand(x) * a \mapsto - * y \mapsto a[x]\}}$$

Proof Rules for Message-Passing

Every message identifier m is annotated with a formula I_m.

$$\{I_m(x_1, .., x_n)\}\ \mathsf{send}(\mathsf{u},\ \mathsf{m}, x_1, .., x_n)\ \{\mathsf{emp}\}$$

$$\{\mathsf{emp}\}\ (x_1, .., x_n) = \mathsf{receive}\,(\mathsf{u},\ \mathsf{m})\ \{I_m(x_1, .., x_n)\}$$

Noised Communication, Again

message m(z) $\{footprint = rand(z)\}$

$\{emp\}$	
`new x = random_int();`	
$\{rand(x)\}$	
`new y = random_int();`	
$\{rand(x) * rand(y)\}$	
`new z = x+y;`	
$\{rand(x) * rand(y) * z \mapsto x + y\}$	
$\{rand(x) * rand(z) * y \mapsto -\}$	$\{emp\}$
`send(Right, m, z)`	`z = receive(Left,m)`
$\{rand(x) * y \mapsto -\}$	$\{rand(z)\}$

Permissions and Sharing

First Step of the Proof: Proving the Authority

```
AUTHORITY() {
  // emp
  new r = random_array();
  // rand(r)
  new c = random_int();
  // rand(r)*rand(c)
  new rc = r[c];
  // rand(r)*rand(c)*rc|->r[c]
  send(Serv,msg_r,r)
  // rand(c)*rc|->r[c] ???
  send(Cl,msg_rc,c,rc)
}
```

Problem: information $r[c]$ is *shared*.

Sharing in Separation Logic

A multiple readers scheme:

$$
\begin{array}{ccc}
 & x := new() & ; \\
 & x \rightarrow val = 3 & ; \\
\ldots & \| & \ldots \\
y = x \rightarrow val; & \| & z = y \rightarrow val; \\
\ldots & \| & \ldots
\end{array}
$$

Boyland's Fractional Permissions

Ownership can be shared:

$$x \mapsto 3 = x \overset{0.5}{\mapsto} 3 * x \overset{0.5}{\mapsto} 3$$

Example

$$\{\mathsf{emp}\}$$
$$x := new()$$
$$x \rightarrow val = 3$$
$$\{x \mapsto 3\}$$

$\{x \overset{0.5}{\mapsto} 3\}$	$\{x \overset{0.5}{\mapsto} 3\}$
$\ldots$	$\ldots$
$y = x \rightarrow val$	$z = y \rightarrow val$
$\ldots$	$\ldots$
$\{x \overset{0.5}{\mapsto} 3\}$	$\{x \overset{0.5}{\mapsto} 3\}$

Semantics of Permissions for Deterministic Stores

States
$\Sigma = \mathit{Var} \rightarrow [0,1] \times \mathit{Int}$

Composition
$\sigma \bullet \sigma' : x \mapsto \sigma(x) + \sigma'(x)$ when defined

Formulas:

$$\sigma \models x \overset{p}{\mapsto} i \quad \text{if} \quad \mathit{dom}(\sigma) = \{x\} \text{ and } \sigma(x) = (p, i)$$

Towards a Semantics for Information's Permission

Problem:

- previous construction relies on resources being *localised*
- information is *diffuse*

Two kinds of solutions

- define a specific model for information with permission
- define $Perm(\Sigma)$ for an arbitrary permission algebra
  - the model is the inductive limit of $\Sigma_{i+1} = Perm(Rand(\Sigma_i))$.

A General Construction for Permissions

Coherent sets of states

- $\sigma \frown \sigma'$ if there is σ'' such that $\sigma'' \geq \sigma, \sigma'$
- $E \subseteq \Sigma$ is coherent if it is a $\frown$-clique.

Permission functor

- $f : \Sigma \to [0,1]$ is in $Perm(\Sigma)$ if $f \downarrow$ & $f^{-1}(]0,1])$ is coherent.
- $f_1 \perp_p f_2$ if $f_1(\sigma) + f_2(\sigma) \leq 1$ and $f_1 \bullet_p f_2$ in $Perm(\Sigma)$
- $f_1 \bullet_p f_2 \triangleq \sigma \mapsto \sup\{p : \sigma = \sigma_1 \ldots \sigma_n \ \& \ f_1(\sigma_i) + f_2(\sigma_i) \geq p\}$

Injection $\Sigma \to Perm(\Sigma), \sigma \mapsto \chi_{\downarrow\sigma}$

Exponentiation
for $f \in Perm(\Sigma)$, $f^p \triangleq \sigma \mapsto p.f(\sigma)$

Assertion Language (extended)

Syntax

$$A, B, \ldots \quad ::= \quad \mathsf{emp} \mid x \mapsto i \mid e = e' \\ \mid A \vee B \mid A * B \mid \sum_{i \in Int} p_i : x \overset{p}{\mapsto} i * A_i. \mid A^p$$

Semantics

$$\sigma \models A^p \;\; \text{if there } \sigma' \text{ s.t. } \sigma = \sigma'^p \text{ and } \sigma' \models A$$

Useful macro $know(x) \triangleq rand(x)^{\frac{1}{2}}$

Proof of the Oblivious Transfer Protocol

Proof of the Authority

```
message msg_r (r)
// rand(r\c) * know(r[c])
message msg_rc (c,rc)
// footprint: rand(c) * know(r[c]) * rc|->r[c]

AUTHORITY() // emp
{ new r = random_array();
  new c = random_int();
  new rc = r[c];
// rand(r,c) * rc|->r[c]
  send(Serv,msg_r,r)
// rand(c) * know(r[c]) * rc|->r[c]
  send(Cl,msg_rc,c,r[c])
} // emp
```

Proof of the Client

```
message msg_s (s)
// footprint: know(s)
message msg_noise (noise)
// footprint:
//   know(s,r[c]) * noise|->- * know(data[s-c],r\c)
//   * data[s-c]=noise[s-c] (+) r[c]

CLIENT() // rand(i)
{ (c,rc) = receive(Aut,msg_rc);
// rand(i,c) * know(r[c]) * rc|->r[c]
  new s = i+c;
// rand(i,s) * know(r[c])
//   * rc|-> r[c] * c|->s-i
  send(Serv,msg_s,s);
// rand(i) * know(s,r[c])
//   * rc|-> r[c] * c|->s-i
  noise = receive(Serv,msg_noise);
// rand(i,c,r[c]) * know(data[i],r\c) *s|-> - * noise|-> -
//   * rc|->r[c] * data[i] = noise[i] (+) r[c]
  new result  = noise[i] (+) rc;
} // rand(i,s,r[c]) * know(data[i]) * c|-> - * rc|-> - * noise|-> -
//   * result |-> data[i]
```

Proof of the Server

```
SERVER() // rand(data)
{ r = receive(Aut,msg_r);
  s = receive(Cl,msg_s);
// rand(data,r\c) * know(s,r[c])
  for n=0 to N-1 do
    new noise[n] =
      data[n] (+) r[s-n]
  done;
// rand(data,r\c) * know(r[c],s) * noise|->-
//     * noise[s-c] = data[s-c] (+) r[c]
  send(Cl,msg_noise);
} // Exists i. rand(data\i,r\c) * know(data[i])
```

Conclusion and Future Work

- an axiomatic semantics for probabilistic message-passing
- a resource model for probabilistic states *without sharing*
- tested on a simple oblivious transfer protocol

- soundness of the proof system? relate local/global semantics?
- more examples of concurrent probabilistic programs?
- what is a good, generic construction for permissions?
- is there a simple model for permissions on probabilities?
- is SL relevant for cryptography/competitive concurrency?