Permission-based separation logic for message-passing concurrency

Vladimiro Sassone
University of Southampton

[
joint work with
Julian Rathke (Southampton)
Adrian Francalanza (Malta)
]

14th Jan 09

Overview

- Introduction - starting point O'Hearn/Reynolds
- Good ideas from there and criticism
- Message-passing paradigm
- Deterministic concurrency (as resources)
- Our two-stage approach
- Our language and Effect system
- Subj. Red and Confluence
- Our Logic - satis. (Exists) and seq rules
- Soundness - exists to for all via stability
- Example - parallel mergesort

Introduction

A decent approach to scalable, transparent concurrent programming remains a challenge for computer science. This is in spite of lots of excellent work towards this goal so far.

Introduction

A decent approach to scalable, transparent concurrent programming remains a challenge for computer science. This is in spite of lots of excellent work towards this goal so far.

With the more recent excitement about multi- and many-core systems this is more true than ever.

Introduction

A decent approach to scalable, transparent concurrent programming remains a challenge for computer science. This is in spite of lots of excellent work towards this goal so far.

With the more recent excitement about multi- and many-core systems this is more true than ever.

Part of a solution here will likely be influenced by techniques for reasoning about concurrent programs. These techniques are also currently underdeveloped.

Introduction

A decent approach to scalable, transparent concurrent programming remains a challenge for computer science. This is in spite of lots of excellent work towards this goal so far.

With the more recent excitement about multi- and many-core systems this is more true than ever.

Part of a solution here will likely be influenced by techniques for reasoning about concurrent programs. These techniques are also currently underdeveloped.

Resource access control and locality of reasoning are emerging as important concepts for this latter task. This talk represents our first efforts at understanding these in a message-passing concurrency setting.

Obvious starting point

O'Hearn and Reynolds on "Resources, Concurrency and Local Reasoning"

Promotes the idea of transfer of 'ownership' of resources between threads. Logic rules tailored to this notion of sharing. Some very neat ideas here.

Obvious starting point

O'Hearn and Reynolds on "Resources, Concurrency and Local Reasoning"

Promotes the idea of transfer of 'ownership' of resources between threads. Logic rules tailored to this notion of sharing. Some very neat ideas here.

Addresses shared variable concurrency using conditional critical regions:

```
with r when B do C endwith
```

Mutex on resource r, execution of C is guarded by boolean B.

Obvious starting point

O'Hearn and Reynolds on "Resources, Concurrency and Local Reasoning"

Promotes the idea of transfer of 'ownership' of resources between threads. Logic rules tailored to this notion of sharing. Some very neat ideas here.

Addresses shared variable concurrency using conditional critical regions:

```
with r when B do C endwith
```

Mutex on resource r, execution of C is guarded by boolean B.

$$\frac{\{(P * R) \wedge B\}\ C\ \{Q * R\}}{\{P\}\ \mathsf{with}\ r\ \mathsf{when}\ B\ \mathsf{do}\ C\ \mathsf{endwith}\ \{Q\}}$$

R is the "resource invariant" - all shared state expressed here.

Some example invariants

For semaphore commands, P(s), V(s):

```
R = ((s=0) ∧ emp) ∨ (s=1 ∧ addr-> -)
```

Some example invariants

For semaphore commands, P(s), V(s):

```
R = ((s=0) ∧ emp) ∨ (s=1 ∧ addr-> -)
```

For a shared buffer:

```
R = (¬full ∧ emp) ∨ (full ∧ addr-> -,-)
```

Some example invariants

For semaphore commands, P(s), V(s):

```
R = ((s=0) ∧ emp) ∨ (s=1 ∧ addr-> -)
```

For a shared buffer:

```
R = (¬full ∧ emp) ∨ (full ∧ addr-> -,-)
```

The reasoning about ownership of the resource gets tied together with reasoning about data. The state of the concurrency protocol is codified using particular data values.

Some example invariants

For semaphore commands, P(s),V(s):

```
R = ((s=0) ∧ emp) ∨ (s=1 ∧ addr-> -)
```

For a shared buffer:

```
R = (¬full ∧ emp) ∨ (full ∧ addr-> -,-)
```

The reasoning about ownership of the resource gets tied together with reasoning about data. The state of the concurrency protocol is codified using particular data values.

In many cases, the intended concurrent control flow is clear and inferrable from the code without the need for such codification. e.g. Producer/Consumer problem.

This is partly a problem of the concurrency model used - no signalling!

Message-passing concurrency

We adopt the message-passing concurrency model for various reasons.

Signalling of events and concurrency control are explicit

Inferring the control flow can be easier. This should help with analysis.

Message-passing concurrency

We adopt the message-passing concurrency model for various reasons.

Signalling of events and concurrency control are explicit

Inferring the control flow can be easier. This should help with analysis.

Idea : leverage such inference to isolate the reasoning about (transfer of) ownership of resources from reasoning about data flow.

Step One : Analyse the control flow
Step Two : Prove data manipulation is correct

Our target setting (for starters):

Simple value-passing calculus:

- asynchronous messages
- no shared state
- CCS like

Our target setting (for starters):

Simple value-passing calculus:

- asynchronous messages
- no shared state
- CCS like

Confluent parallel programs:

- functional relationship between inputs and outputs
- parallelism is internal to the program
- Scheduling is unimportant (speed independence)

Our target setting (for starters):

Simple value-passing calculus:

- asynchronous messages
- no shared state
- CCS like

Confluent parallel programs:

- functional relationship between inputs and outputs
- parallelism is internal to the program
- Scheduling is unimportant (speed independence)

Driving example:

- parallel mergesort

Reasoning about message-passing

Traditionally,

Equivalences: $P \approx SPEC$

Temporal/Modal Logics: $P \vDash F$

Hoare-like logics: { state } P { state'}

(Typed) Static analyses: $\Delta \vdash P : eff$

We intend to exploit the latter two approaches by first using and effect system to analyse the concurrency control, and then using local Hoare reasoning to talk about data.

Step 1:
Control Flow

Determinism via resource access control

The main (only) source of non-determinism in message-passing systems is the race for synchronisation on a channel:

$$a?x.P \parallel a?y.Q \parallel a!v$$

Here there is a competition to receive a value on a given channel.

Similarly for

$$a?x.P \parallel a!v \parallel a!w$$

So we could easily delimit deterministic processes by using linear channels.

This would rule out a large class of interesting processes - e.g. Producer/Consumers, or parallel mergesort

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x + y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x + y) \parallel b?z'.a?z.c!z$$

This uses channel 'a' non-linearly yet is deterministic.

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x + y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x + y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x + y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Rather, we consider linear permissions on the resources (channels). These permissions can be passed implicitly by communication from sender to receiver.

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Rather, we consider linear permissions on the resources (channels). These permissions can be passed implicitly by communication from sender to receiver.

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

This uses channel 'a' non-linearly yet is deterministic.

Rather, we consider linear permissions on the resources (channels). These permissions can be passed implicitly by communication from sender to receiver.

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Rather, we consider linear permissions on the resources (channels). These permissions can be passed implicitly by communication from sender to receiver.

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Rather, we consider linear permissions on the resources (channels). These permissions can be passed implicitly by communication from sender to receiver.

$$a\cancel{!}v \parallel d!w \parallel a\cancel{?}x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x + y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Rather, we consider linear permissions on the resources (channels). These permissions can be passed implicitly by communication from sender to receiver.

$$a\cancel{!}v \parallel \cancel{d}!w \parallel a\cancel{?}x.d\cancel{?}y.(b!\langle\rangle \parallel a!x + y) \parallel b?z'.a?z.c!z$$

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Rather, we consider linear permissions on the resources (channels). These permissions can be passed implicitly by communication from sender to receiver.

$$a\cancel{!}v \parallel \cancel{d}!w \parallel a\cancel{?}x.d\cancel{?}y.(\cancel{b!}\langle\rangle \parallel a!x+y) \parallel \cancel{b?}z'.a?z.c!z$$

Determinism again

Consider the following process:

$$a!v \parallel d!w \parallel a?x.d?y.(b!\langle\rangle \parallel a!x+y) \parallel b?z'.a?z.c!z$$

This uses channel ‘a’ non-linearly yet is deterministic.

Rather, we consider linear permissions on the resources (channels). These permissions can be passed implicitly by communication from sender to receiver.

$$a\cancel{!}v \parallel \cancel{d}!w \parallel a\cancel{?}x.d\cancel{?}y.(\cancel{b!}\langle\rangle \parallel a!x+y) \parallel \cancel{b?}z'.a?z.c!z$$

The channel is used linearly at any given snapshot in time.

An analysis of determinism

We check to see whether processes can be assigned linear <u>permissions</u>.

Permissions can be transfered <u>implicitly</u> through communication.

Exactly which permissions will be transfered during each communication is a parameter to the analysis :

Γ is a map from channels to sets of permissions - c |--> {a! , b? }

An analysis of determinism

We check to see whether processes can be as

Permissions can be transfered <u>implicitly</u> throu

Exactly which permissions will be transfered c
is a parameter to the analysis :

Gain permission to send on ‘a’ and receive on ‘b’ after I receive on channel c

Γ is a map from channels to sets of permissions - c |--> {a! , b? }

An analysis of determinism

We check to see whether processes can be assigned linear permissions.

Permissions can be transfered implicitly through communication.

Exactly which permissions will be transfered during each communication is a parameter to the analysis :

Γ is a map from channels to sets of permissions - c |--> {a! , b? }

We think of Γ as representing the intended control flow of the process. It acts as an assertion regarding control which must be checked. This is in accord with O'Hearn's tenet that

"Ownership is in the eye of the asserter"

Our simple calculus

Grammar:

$$e ::= v \mid x \mid e + e \mid \ldots \qquad\qquad v ::= 0 \mid 1 \mid 2$$

$$\begin{array}{lcl} P, Q & ::= & a!e \mid a?x.P \\ & \mid & \mathsf{if}\ e \leq e'\ \mathsf{then}\ P\ \mathsf{else}\ Q \\ & \mid & \mathsf{rec}\ X.P \mid X \\ & \mid & \mathsf{nil} \mid P \parallel Q \mid (\mathsf{new}\ a)P \end{array}$$

With obvious reduction rules including:

$$\frac{e \Downarrow v}{a!e \parallel a?x.P \rightarrow P[v/x]}$$

Effect system

Transfer of ownership

$$\Gamma \vdash P : \epsilon$$

Permissions required by P

Effect system

Transfer of ownership

$$\Gamma \vdash P : \epsilon$$

Permissions required by P

Rule for nil

$$\frac{}{\Gamma \vdash \mathsf{nil} : \emptyset}$$

Effect system

Transfer of ownership

$$\Gamma \vdash P : \epsilon$$

Permissions required by P

Rule for nil

$$\frac{}{\Gamma \vdash \textsf{nil} : \emptyset}$$

Rule for output

$$\frac{}{\Gamma \vdash a!e : \Gamma(a) + a!}$$

Effect system

Transfer of ownership

$$\Gamma \vdash P : \epsilon$$

Permissions required by P

Rule for nil

$$\frac{}{\Gamma \vdash \mathsf{nil} : \emptyset}$$

Rule for output

$$\frac{}{\Gamma \vdash a!e : \Gamma(a) + a!}$$

Rule for input

$$\frac{\Gamma \vdash P : \epsilon}{\Gamma \vdash a?x.P : (\epsilon - \Gamma(a) + a?)}$$

Effect system

Transfer of ownership $\qquad \Gamma \vdash P : \epsilon \qquad$ Permissions required by P

Rule for nil

$$\frac{}{\Gamma \vdash \mathsf{nil} : \emptyset}$$

Rule for output

$$\frac{}{\Gamma \vdash a!e : \Gamma(a) + a!}$$

Rule for input

$$\frac{\Gamma \vdash P : \epsilon}{\Gamma \vdash a?x.P : (\epsilon - \Gamma(a) + a?)}$$

Rule for par

$$\frac{\Gamma \vdash P : \epsilon \qquad \Gamma \vdash Q : \epsilon'}{\Gamma \vdash P \parallel Q : \epsilon \oplus \epsilon'}$$

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$$\Gamma =$$

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{c!}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$$\Gamma =$$

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{s?} {c!}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$$\Gamma =$$

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{s?, c!}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$$\Gamma =$$

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{s?, c!}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$\Gamma =$ p → { s? , c! , }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{s?, c!}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$\Gamma =$ p $\rightarrow$ { s? , c! , }

c $\rightarrow$ { s? , p! , }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{s?, c!}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$\Gamma =$ p → { s? , c! , p! }

c → { s? , p! , }

Example

$$Pr = \textsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{s?, c!}

$$Con = \textsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$\Gamma =$ p → { s? , c! , p! }

c → { s? , p! , c! }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{s?,c!, p!}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$\Gamma =$ p $\rightarrow$ { s? , c! , p! }

c $\rightarrow$ { s? , p! , c! }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{p?}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$\Gamma =$ p → { s? , c! , p! }

c → { s? , p! , c! }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{p?} {p?}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$$\Gamma = \begin{array}{l} p \rightarrow \{\, s?\,, c!\,,\ p! \quad \} \\ c \rightarrow \{\, s?\,, p!\,,\ c! \quad \} \end{array}$$

Example

$$Pr = \textsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$

{p?}

$$Con = \textsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$\Gamma =$ p → { s? , c! , p! }

c → { s? , p! , c! }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X)) \qquad : \{p?\}$$

{p?}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$

$\Gamma =$ p → { s? , c! , p! }

c → { s? , p! , c! }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$: {p?}

{p?}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$: {c?}

$\Gamma =$ p → { s? , c! , p! }

c → { s? , p! , c! }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$: {p?}

{p?}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$: {c?}

$\Gamma =$ p → { s? , c! , p! }

c → { s? , p! , c! }

$$Start = p!\langle\rangle$$

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$: {p?}

{p?}

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$: {c?}

$$\Gamma =$$

p → { s? , c! , p! }

c → { s? , p! , c! }

$$Start = p!\langle\rangle$$: { p! , c! , s? }

Example

$$Pr = \mathsf{rec}\ X.(p?().s?().(c!\langle\rangle \parallel X))$$: {p?}

{p?}

$$\bigoplus$$

$$Con = \mathsf{rec}\ X.(c?().s?().(p!\langle\rangle \parallel X))$$: {c?}

$\Gamma =$ p → { s? , c! , p! }

c → { s? , p! , c! }

$$\bigoplus$$

$$Start = p!\langle\rangle$$: { p! , c! , s? }

Properties of effect system

Proposition (subject reduction) :

$$\Gamma \vdash P : \epsilon \text{ and } P \rightarrow Q \text{ implies } \Gamma \vdash Q : \epsilon' \text{ and } \epsilon' \subseteq \epsilon$$

Properties of effect system

Proposition (subject reduction) :

$$\Gamma \vdash P : \epsilon \text{ and } P \rightarrow Q \text{ implies } \Gamma \vdash Q : \epsilon' \text{ and } \epsilon' \subseteq \epsilon$$

Proposition (confluence) :

$$\Gamma \vdash P : \epsilon \text{ and } P \rightarrow P_1 \text{ and } P \rightarrow P_2 \text{ implies}$$
$$P_1 \equiv P_2 \text{ or } P_1 \rightarrow P_3 \text{ and } P_2 \rightarrow P_3 \text{ for some } P_3$$

Step 2:
Proof System

Logic

$$\phi, \varphi ::= \mathsf{emp} \mid a\langle v\rangle \mid \phi \rhd \phi \mid \phi * \phi$$

The a<v> formulas represent outputs on the wire - analogous to memory locations

Satisfaction is defined by:

$$\begin{array}{lll}
\Gamma \triangleright P \models \mathsf{emp} & \text{iff} & \textit{always} \\
\Gamma \triangleright P \models a\langle v\rangle & \text{iff} & P \equiv a!e \parallel P' \text{ where } e \Downarrow v \\
\Gamma \triangleright P \models \phi \rhd \psi & \text{iff} & \forall Q. \Gamma \triangleright Q \models \phi, \Gamma \vdash Q \perp P \text{ implies } \Gamma \triangleright P \parallel Q \mid\models \psi \\
\Gamma \triangleright P \models \phi_1 * \phi_2 & \text{iff} & \exists P_1, P_2. P \equiv P_1 \parallel P_2 \text{ and } \Gamma \triangleright P_1 \models \phi_1, \Gamma \triangleright P_2 \models \phi_2 \\
\Gamma \triangleright P \mid\models \phi & \text{iff} & \exists Q. P \rightarrow^* Q, \Gamma \triangleright Q \models \phi
\end{array}$$

Logic

$$\phi, \varphi ::= \mathsf{emp} \mid a\langle v\rangle \mid \phi \rhd \phi \mid \phi * \phi$$

The a<v> formulas represent outputs on the wire - analogous to memory locations

Satisfaction is defined by:

$$
\begin{array}{lll}
\Gamma \triangleright P \models \mathsf{emp} & \text{iff} & \textit{always} \\
\Gamma \triangleright P \models a\langle v\rangle & \text{iff} & P \equiv a!e \parallel P' \text{ where } e \Downarrow v \\
\Gamma \triangleright P \models \phi \rhd \psi & \text{iff} & \forall Q.\Gamma \triangleright Q \models \phi, \Gamma \vdash Q \perp P \text{ implies } \Gamma \triangleright P \parallel Q \Vdash \psi \\
\Gamma \triangleright P \models \phi_1 * \phi_2 & \text{iff} & \exists P_1, P_2.P \equiv P_1 \parallel P_2 \text{ and } \Gamma \triangleright P_1 \models \phi_1, \Gamma \triangleright P_2 \models \phi_2 \\
\Gamma \triangleright P \Vdash \phi & \text{iff} & \exists Q.P \rightarrow^* Q, \Gamma \triangleright Q \models \phi
\end{array}
$$

Separate permissions needed

Logic

$$\phi, \varphi ::= \mathsf{emp} \mid a\langle v\rangle \mid \phi \triangleright \phi \mid \phi * \phi$$

The a<v> formulas represent outputs on the wire - analogous to memory locations

Satisfaction is defined by:

$$\begin{array}{lll}
\Gamma \triangleright P \models \mathsf{emp} & \text{iff} & \textit{always} \\
\Gamma \triangleright P \models a\langle v\rangle & \text{iff} & P \equiv a!e \parallel P' \text{ where } e \Downarrow v \\
\Gamma \triangleright P \models \phi \triangleright \psi & \text{iff} & \forall Q. \Gamma \triangleright Q \models \phi, \Gamma \vdash Q \perp P \text{ implies } \Gamma \triangleright P \parallel Q \models\!\models \psi \\
\Gamma \triangleright P \models \phi_1 * \phi_2 & \text{iff} & \exists P_1, P_2. P \equiv P_1 \parallel P_2 \text{ and } \Gamma \triangleright P_1 \models \phi_1, \Gamma \triangleright P_2 \models \phi_2 \\
\Gamma \triangleright P \models\!\models \phi & \text{iff} & \exists Q. P \rightarrow^* Q, \Gamma \triangleright Q \models \phi
\end{array}$$

Separate permissions needed

Sequent rules

Sequents are interpreted as follows:

$$\Gamma \models \{\phi\}P\{\psi\} \stackrel{def}{=} \Gamma \vdash Q \perp P, \;\; \Gamma \triangleright Q \models \phi$$
$$\text{implies } \Gamma \triangleright P \parallel Q \models \psi$$

Sequent rules

Sequents are interpreted as follows:

$$\Gamma \models \{\phi\}P\{\psi\} \stackrel{def}{=} \Gamma \vdash Q \perp P, \;\; \Gamma \triangleright Q \models \phi$$
$$\text{implies } \Gamma \triangleright P \parallel Q \| \models \psi$$

Some rules:

Sequent rules

Sequents are interpreted as follows:

$$\Gamma \models \{\phi\}P\{\psi\} \stackrel{def}{=} \Gamma \vdash Q \perp P, \;\; \Gamma \triangleright Q \models \phi$$
$$\text{implies } \Gamma \triangleright P \parallel Q \| \models \psi$$

Some rules:

Wire $\dfrac{}{\Gamma \vdash \{\phi\}\ P\ \{\phi\}}$

Sequent rules

Sequents are interpreted as follows:

$$\Gamma \models \{\phi\}P\{\psi\} \stackrel{def}{=} \Gamma \vdash Q \perp P, \;\; \Gamma \triangleright Q \models \phi$$
$$\text{implies } \Gamma \triangleright P \parallel Q \| \models \psi$$

Some rules:

Wire $$\frac{}{\Gamma \vdash \{\phi\}\ P\ \{\phi\}}$$

Par $$\frac{\Gamma \vdash \{\phi_1\}\ P\ \{\psi_1\} \qquad \Gamma \vdash \{\phi_2\}\ Q\ \{\psi_2\}}{\Gamma \vdash \{\phi_1 * \phi_2\}\ P \parallel Q\ \{\psi_1 * \psi_2\}}$$

More sequent rules

Cut

$$\frac{\Gamma \vdash \{\phi\}\ P\ \{\chi\} \qquad \Gamma \vdash \{\chi\}\ Q\ \{\psi\}}{\Gamma \vdash \{\phi\}\ P \parallel Q\ \{\psi\}}$$

More sequent rules

Cut
$$\frac{\Gamma \vdash \{\phi\}\ P\ \{\chi\} \qquad \Gamma \vdash \{\chi\}\ Q\ \{\psi\}}{\Gamma \vdash \{\phi\}\ P \parallel Q\ \{\psi\}}$$

In-R
$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi\}\ a?x.P\ \{a\langle v\rangle \rhd \psi\}}$$

More sequent rules

Cut
$$\frac{\Gamma \vdash \{\phi\}\ P\ \{\chi\} \qquad \Gamma \vdash \{\chi\}\ Q\ \{\psi\}}{\Gamma \vdash \{\phi\}\ P \parallel Q\ \{\psi\}}$$

In-R
$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi\}\ a?x.P\ \{a\langle v\rangle \triangleright \psi\}}$$

$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi * a\langle v\rangle\}\ a?x.P\ \{\psi\}}$$
In-L

More sequent rules

Cut
$$\frac{\Gamma \vdash \{\phi\}\ P\ \{\chi\} \qquad \Gamma \vdash \{\chi\}\ Q\ \{\psi\}}{\Gamma \vdash \{\phi\}\ P \parallel Q\ \{\psi\}}$$

In-R
$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi\}\ a?x.P\ \{a\langle v\rangle \triangleright \psi\}}$$

In-L
$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi * a\langle v\rangle\}\ a?x.P\ \{\psi\}}$$

Out-R
$$\frac{e \Downarrow v}{\Gamma \vdash \{\mathsf{emp}\}\ a!e\ \{a\langle v\rangle\}}$$

More sequent rules

Cut
$$\frac{\Gamma \vdash \{\phi\}\ P\ \{\chi\} \qquad \Gamma \vdash \{\chi\}\ Q\ \{\psi\}}{\Gamma \vdash \{\phi\}\ P \parallel Q\ \{\psi\}}$$

In-R
$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi\}\ a?x.P\ \{a\langle v\rangle \rhd \psi\}}$$

$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi * a\langle v\rangle\}\ a?x.P\ \{\psi\}}$$
In-L

Out-R
$$\frac{e \Downarrow v}{\Gamma \vdash \{\mathsf{emp}\}\ a!e\ \{a\langle v\rangle\}}$$

$$\frac{e \Downarrow v}{\Gamma \vdash \{a\langle v\rangle \rhd \phi\}\ a!e\ \{\phi\}}$$
Out-L

More sequent rules

Cut
$$\frac{\Gamma \vdash \{\phi\}\ P\ \{\chi\} \qquad \Gamma \vdash \{\chi\}\ Q\ \{\psi\}}{\Gamma \vdash \{\phi\}\ P \parallel Q\ \{\psi\}}$$

In-R
$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi\}\ a?x.P\ \{a\langle v\rangle \rhd \psi\}}$$

In-L
$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi * a\langle v\rangle\}\ a?x.P\ \{\psi\}}$$

Out-R
$$\frac{e \Downarrow v}{\Gamma \vdash \{\mathsf{emp}\}\ a!e\ \{a\langle v\rangle\}}$$

Out-L
$$\frac{e \Downarrow v}{\Gamma \vdash \{a\langle v\rangle \rhd \phi\}\ a!e\ \{\phi\}}$$

No nasty side-conditions on Par, Cut etc.

More sequent rules

Cut
$$\frac{\Gamma \vdash \{\phi\}\ P\ \{\chi\} \qquad \Gamma \vdash \{\chi\}\ Q\ \{\psi\}}{\Gamma \vdash \{\phi\}\ P \parallel Q\ \{\psi\}}$$

In-R
$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi\}\ a?x.P\ \{a\langle v\rangle \rhd \psi\}}$$

$$\frac{\Gamma \vdash \{\phi\}\ P[v/x]\ \{\psi\}}{\Gamma \vdash \{\phi * a\langle v\rangle\}\ a?x.P\ \{\psi\}}$$
In-L

Out-R
$$\frac{e \Downarrow v}{\Gamma \vdash \{\mathsf{emp}\}\ a!e\ \{a\langle v\rangle\}}$$

$$\frac{e \Downarrow v}{\Gamma \vdash \{a\langle v\rangle \rhd \phi\}\ a!e\ \{\phi\}}$$
Out-L

No nasty side-conditions on Par, Cut etc.

The actual form of sequents is $\Gamma, b \vdash \{\phi\}\ P\ \{\phi\}$ with b boolean

Soundness

Theorem:

$$\Gamma \vdash \{\phi\}\ P\ \{\psi\} \text{ implies } \Gamma \models \{\phi\}\ P\ \{\psi\}$$

Straightforward to prove - because of the path existential interpretation of formulas

Soundness

Theorem:

$$\Gamma \vdash \{\phi\}\ P\ \{\psi\} \text{ implies } \Gamma \models \{\phi\}\ P\ \{\psi\}$$

Straightforward to prove - because of the path existential interpretation of formulas

This interpretation is not conventional and fairly weak - but recall that we are working with a class of confluent processes. One path is much the same as any other.

We make use of this fact to lift soundness to an ‘all paths’ interpretation.

Satisfaction preservation

We'd like to prove

Proposition:

$$\Gamma \triangleright P \Vert\!\models \phi, \quad P \rightarrow^* Q \text{ implies } \Gamma \triangleright Q \Vert\!\models \phi$$

But unfortunately, this is not true for all formulas:

$$\phi = a\langle v\rangle * (a\langle v\rangle \triangleright b\langle v\rangle) \qquad P = (a!v \parallel a?x.b!x)$$

$$P \rightarrow b!v \not\models \phi$$

Satisfaction preservation

We'd like to prove

Proposition:

$$\Gamma \triangleright P \mid\!\models \phi, \quad P \rightarrow^* Q \text{ implies } \Gamma \triangleright Q \mid\!\models \phi$$

But unfortunately, this is not true for all formulas:

$$\phi = a\langle v\rangle * (a\langle v\rangle \triangleright b\langle v\rangle) \qquad P = (a!v \parallel a?x.b!x)$$

$$P \rightarrow b!v \not\models \phi$$

However, if we rule out adjoints and check that P does not require 'input-permissions' specified by Φ, then the above proposition holds.

Example - parallel mergesort

$$\mathbf{srt}(X)_{i,j} \triangleq \begin{cases} X! & i = j \\ (\mathsf{new}\, c_1, c_2) \begin{pmatrix} \mathbf{srt}(c_1)_{i,m} \| \mathbf{srt}(c_2)_{m+1,j} \\ \| c_1?.c_2?.\mathbf{mrg}(X)_{i,m+1,j} \end{pmatrix} & i < j,\; m = \frac{j-i}{2} \end{cases} \qquad \mathbf{shft}(X)_{i,j} \triangleq \begin{cases} X! & i = j \\ a_{j-1}?x. \begin{pmatrix} \mathbf{shft}(X)_{i,j-1} \\ \| a_j!x \end{pmatrix} & i < j \end{cases}$$

$$\mathbf{mrg}(X)_{i,m,j} \triangleq \begin{cases} X! & i = m \vee m = j+1 \\ a_i?x.a_m?y.\mathsf{if}\ x \leq y\ \mathsf{then}\ \left(a_i!x \| a_m!y \| \mathbf{mrg}(X)_{i+1,m,j}\right) & \\ \qquad \mathsf{else}\ \left(a_i!y. \| (\mathsf{new}\, b) \begin{pmatrix} \mathbf{shft}(b)_{i+1,m} \| \\ b?. \begin{pmatrix} a_{i+1}!x \\ \| \mathbf{mrg}(X)_{i+1,m+1,j} \end{pmatrix} \end{pmatrix}\right) & i < m \leq j \end{cases}$$

The arrays are represented using a collection of outputs:

$$a_1!v_1 \parallel a_2!v_2 \parallel \cdots \parallel a_n!v_n$$

Example specified

$$A_i^j\langle\vec{v}_i^{\,j}\rangle \triangleq \begin{cases}\mathbf{emp} & i > j\\ a_i\langle v_i\rangle * A_{i+1}^j\langle\vec{v}_{i+1}^{\,j}\rangle & i \le j\end{cases} \qquad \vec{u}_i^{\,j} - v \stackrel{\text{def}}{=} \begin{cases}\vec{u}_i^{\,m-1}\vec{u}_{m+1}^{\,j} & \text{if } \vec{u}_i^{\,j} = \vec{u}_i^{\,m-1} v \vec{u}_{m+1}^{\,j}\\ \text{undefined} & \text{otherwise}\end{cases}$$

$$v \le \vec{u}_i^{\,j} \stackrel{\text{def}}{=} \forall i \le k \le j.\ v \le u_k \qquad v \in \vec{u}_i^{\,j} \stackrel{\text{def}}{=} \exists i \le m \le j.\ \vec{u}_i^{\,j} = \vec{u}_i^{\,m-1} v \vec{u}_{m+1}^{\,j}$$

$$\vec{v}_i^{\,j} \doteq \vec{u}_i^{\,j} \stackrel{\text{def}}{=} v_i \in \vec{u}_i^{\,j} \ \wedge\ \vec{v}_{i+1}^{\,j} \doteq (\vec{u}_i^{\,j} - v) \qquad \mathbf{srt}(\vec{v}_i^{\,j}) \stackrel{\text{def}}{=} v_i \le \vec{v}_{i+1}^{\,j} \ \wedge\ \mathbf{srt}(\vec{v}_{i+1}^{\,j})$$

We can derive the sequent by building a proof tree by induction over the indices:

$$\left\{A_i^j\langle\vec{v}_i^{\,j}\rangle\right\} \qquad \mathbf{Srt}(b)_{i,j} \qquad \left\{b\langle\rangle * A_i^j\langle\vec{u}_i^{\,j}\rangle\right\}$$

where

$$\underline{(\vec{v}_i^{\,j} \doteq \vec{w}_i^{\,j}) \wedge (\vec{w}_i^{\,j} \doteq \vec{u}_i^{\,j}) \wedge \mathbf{srt}(\vec{w}_i^{\,m-1}) \wedge \mathbf{srt}(\vec{w}_m^{\,j}) \wedge \mathbf{srt}(\vec{u}_i^{\,j})}$$

Other related work

Hennessy Milner Logic for CCS (Stirling)

Hennessy Milner Logic for the Pi-calculus (Dam)

Modal logics for Typed Pi-calculus (Berger+)

Compositional proof systems - path blow-up

Need to study this further

Rely-Guarantee Separation Logic (Feng+, Vafeiadis+)

Permission accounting in Separation Logic (Bornat+)

Could be very useful model for extensions

Conclusions

We proposed a two-step analysis for local reasoning about message-passing concurrency:

- Effect analysis of deterministic flow
- Local Hoare logic for reasoning about data

Plenty more work to do to strengthen the class of programs we address

- controlled introduction of interference and racy-programs
- name-passing (a la pi-calculus)
- lots more examples

Thank You