DimSum 

A Decentralized Approach to Multi-language Semantics and Verification

Michael Sammler, *Simon Spies, Youngju Song, Emanuele D'Osualdo, Robbert Krebbers, Deepak Garg, and Derek Dreyer*

POPL'23

January 18, 2023

Dimsum icon by Icons8

How can we **reason modularly** about **multi-language programs**?

Example

```
void main() {
   char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
   memmove(x + 1, x + 0, 2);         // x = {1, 1, 2}
   print(x[1]); print(x[2]);
}
```

Example

System call

```
void main() {
    char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
    memmove(x + 1, x + 0, 2);         // x = {1, 1, 2}
    print(x[1]); print(x[2]);
}
```

Example

System call

```
void main() {
    char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
    memmove(x + 1, x + 0, 2);         // x = {1, 1, 2}
    print(x[1]); print(x[2]);
}
```

Library print

print : mov $\mathbf{x8}$, $\mathbf{PRINT}$; syscall; ret

Example

System call

```
void main() {
    char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
    memmove(x + 1, x + 0, 2);         // x = {1, 1, 2}
    print(x[1]); print(x[2]);
}
```

Key aspects

#1 Extend **C** with system calls via **print** library

Library print

print : mov $\mathbf{x8}$, $\mathbf{PRINT}$; syscall; ret

Example

Key aspects

#1 Extend **C** with system calls via **print** library

```
void main() {
   char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
   memmove(x + 1, x + 0, 2);         // x = {1, 1, 2}
   print(x[1]); print(x[2]);
}
```

Library memmove

```
void memmove(char *d, char *s, int n) {
  if (locle(d, s)) { return memcpy(d, s, n, 1); }
  else { return memcpy(d+n-1, s+n-1, n, -1); } }
void memcpy(char *d, char *s, int n, int o) {
  if (0 < n) { *d = *s; memcpy(d+o, s+o, n-1, o) } }
```

Library print

print : mov x8, PRINT; syscall; ret

Example

```
void main() {
    char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
    memmove(x + 1, x + 0, 2);          // x = {1, 1, 2}
    print(x[1]); print(x[2]);
}
```

Key aspects

#1 Extend **C** with system calls via **print** library

Address comparison

Library memmove

```
void memmove(char *d, char *s, int n) {
  if (locle(d, s)) { return memcpy(d, s, n, 1); }
  else { return memcpy(d+n-1, s+n-1, n, -1); } }
void memcpy(char *d, char *s, int n, int o) {
  if (0 < n) { *d = *s; memcpy(d+o, s+o, n-1, o) } }
```

Library print

print : mov x8, PRINT; syscall; ret

Example

```
void main() {
   char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
   memmove(x + 1, x + 0, 2);          // x = {1, 1, 2}
   print(x[1]); print(x[2]);
}
```

Key aspects

#1 Extend **C** with system calls via **print** library

Address comparison

Library memmove

```
void memmove(char *d, char *s, int n) {
  if (locle(d, s)) { return memcpy(d, s, n, 1); }
  else { return memcpy(d+n-1, s+n-1, n, -1); } }
void memcpy(char *d, char *s, int n, int o) {
  if (0 < n) { *d = *s; memcpy(d+o, s+o, n-1, o) } }
```

Library locle

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

Library print

print : mov $\mathbf{x8}$, $\mathbf{PRINT}$; syscall; ret

Example

```
void main() {
    char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
    memmove(x + 1, x + 0, 2);         // x = {1, 1, 2}
    print(x[1]); print(x[2]);
}
```

Key aspects

#1 Extend **C** with system calls via **print** library

#2 Use **Asm**'s concrete memory model to provide address comparison to **C** via **locle**

Address comparison

Library memmove

```
void memmove(char *d, char *s, int n) {
  if (locle(d, s)) { return memcpy(d, s, n, 1); }
  else { return memcpy(d+n-1, s+n-1, n, -1); } }
void memcpy(char *d, char *s, int n, int o) {
  if (0 < n) { *d = *s; memcpy(d+o, s+o, n-1, o) } }
```

Library locle

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

Library print

print : mov $\mathbf{x8}$, $\mathbf{PRINT}$; syscall; ret

Example

```
void main() {
    char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
    memmove(x + 1, x + 0, 2);         // x = {1, 1, 2}
    print(x[1]); print(x[2]);
}
```

Key aspects

#1 Extend **C** with system calls via **print** library

#2 Use **Asm**'s concrete memory model to provide address comparison to **C** via **locle**

#3 Reason about **memcpy** independent of **Asm**

Library memmove

```
void memmove(char *d, char *s, int n) {
  if (locle(d, s)) { return memcpy(d, s, n, 1); }
  else { return memcpy(d+n-1, s+n-1, n, -1); } }
void memcpy(char *d, char *s, int n, int o) {
  if (0 < n) { *d = *s; memcpy(d+o, s+o, n-1, o) } }
```

Library locle

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

Library print

print : mov $\mathbf{x8}$, $\mathbf{PRINT}$; syscall; ret

Existing approaches

Existing approaches

1. Pilsner [Neis et al. 2015]**, …**

*All target-level (**Asm**) code is specified using source-level (**C**) code.*

Existing approaches

1. Pilsner [Neis et al. 2015]**, …**

*All target-level (**Asm**) code is specified using source-level (**C**) code.*

Fixes the source language as specification language: disallows **print** and **locle**

Existing approaches

1. Pilsner [Neis et al. 2015], **…**

*All target-level (**Asm**) code is specified using source-level (**C**) code.*

2. CompCert-based approaches
[Stewart et al. 2015, …]

Link all languages via a common interaction protocol.

Fixes the source language as specification language: disallows **print** and **locle**

Existing approaches

1. Pilsner [Neis et al. 2015], **…**

*All target-level (**Asm**) code is specified using source-level (**C**) code.*

2. CompCert-based approaches
[Stewart et al. 2015, …]

Link all languages via a common interaction protocol.

Fixes the source language as specification language: disallows **print** and **locle**

Fixes (abstract) memory model: disallows **locle**

Existing approaches

1. Pilsner [Neis et al. 2015], **…**

*All target-level (**Asm**) code is specified using source-level (**C**) code.*

2. CompCert-based approaches
[Stewart et al. 2015, …]

Link all languages via a common interaction protocol.

3. Syntactic multi-languages
[Ahmed and Blume 2011, …]

Embed all languages into one large multi-language.

Fixes the source language as specification language: disallows **print** and **locle**

Fixes (abstract) memory model: disallows **locle**

Existing approaches

1. Pilsner [Neis et al. 2015], **…**

*All target-level (**Asm**) code is specified using source-level (**C**) code.*

2. CompCert-based approaches [Stewart et al. 2015, …]

Link all languages via a common interaction protocol.

3. Syntactic multi-languages [Ahmed and Blume 2011, …]

Embed all languages into one large multi-language.

Fixes the source language as specification language: disallows **print** and **locle**

Fixes (abstract) memory model: disallows **locle**

Fixes the set of languages: requires reasoning about **Asm** context for **memcpy**

Desiderata for multi-language reasoning

Desiderata for multi-language reasoning

#1 No fixed source language as specification language

Can link with target code that is not representable in the source

Desiderata for multi-language reasoning

#1 No fixed source language as specification language

Can link with target code that is not representable in the source

#2 No fixed memory model

Supports languages with different memory models

Desiderata for multi-language reasoning

#1 No fixed source language as specification language
Can link with target code that is not representable in the source

#2 No fixed memory model
Supports languages with different memory models

#3 No fixed set of languages
Allows language-local reasoning

Desiderata for multi-language reasoning

#1 No fixed source language as specification language
Can link with target code that is not representable in the source

#2 No fixed memory model
Supports languages with different memory models

#3 No fixed set of languages
Allows language-local reasoning

#4 No fixed notion of linking

Desiderata for multi-language reasoning

#1 No fixed source language as specification language

Can link with target code that is not representable in the source

#2 No fixed memory model

Supports languages with different memory models

#3 No fixed set of languages

Allows language-local reasoning

DimSum

#4 No fixed notion of linking

Dimsum icon by Icons8

DimSum

Decentralized **Multi-language Reasoning**

Dimsum icon by Icons8

#1 No fixed source / spec. language

#2 No fixed memory model

#3 No fixed set of languages

#4 No fixed notion of linking

DimSum

Decentralized **Multi-language Reasoning**

[Dimsum](#) icon by [Icons8](#)

#1 No fixed source / spec. language

#2 No fixed memory model

#3 No fixed set of languages

#4 No fixed notion of linking

DimSum

Decentralized **Multi-language Reasoning**

Combining ideas from

process algebra, Kripke relations, angelic non-determinism, …

Dimsum icon by Icons8

#1 No fixed source / spec. language

#2 No fixed memory model

#3 No fixed set of languages

#4 No fixed notion of linking

DimSum

Decentralized **Multi-language Reasoning**

Combining ideas from

process algebra, Kripke relations, angelic non-determinism, …

Instantiations

Rec: *C-like language*

Asm: *assembly language*

Spec: *specification language*

Dimsum icon by Icons8

#1 No fixed source / spec. language

#2 No fixed memory model

#3 No fixed set of languages

#4 No fixed notion of linking

DimSum

***Decentralized* Multi-language Reasoning**

Combining ideas from
process algebra, Kripke relations, angelic non-determinism, …

Instantiations	**Evaluation**
Rec: *C-like language*	$\downarrow R$: Compiler from *Rec* to *Asm*
Asm: *assembly language*	locle : pointer comparison
Spec: *specification language*	$\oplus_{coro}$: coroutines

Dimsum icon by Icons8

#1 No fixed source / spec. language

#2 No fixed memory model

#3 No fixed set of languages

#4 No fixed notion of linking

DimSum

Decentralized Multi-language Reasoning

Combining ideas from

process algebra, Kripke relations, angelic non-determinism, …

Instantiations

Rec: *C-like language*

Asm: *assembly language*

Spec: *specification language*

Evaluation

$\downarrow R$: Compiler from *Rec* to *Asm*

locle : pointer comparison

$\oplus_{\text{coro}}$: coroutines

#1 No fixed source / spec. language

#2 No fixed memory model

#3 No fixed set of languages

#4 No fixed notion of linking

DimSum

Decentralized Multi-language Reasoning

Combining ideas from

process algebra, Kripke relations, angelic non-determinism, …

Instantiations

Rec: *C-like language*

Asm: *assembly language*

Spec: *specification language*

Evaluation

$\downarrow_R$: Compiler from *Rec* to *Asm*

$\boxed{\text{locle : pointer comparison}}$

$\oplus_{coro}$: coroutines

Dimsum icon by Icons8

Example: onetwo

```
void main() {
   char x[3]; x[0] = 1; x[1] = 2;    // x = {1, 2, *}
   memmove(x + 1, x + 0, 2);         // x = {1, 1, 2}
   print(x[1]); print(x[2]);
}
```

Library memmove

```
void memmove(char *d, char *s, int n) {
  if (locle(d, s)) { return memcpy(d, s, n, 1); }
  else { return memcpy(d+n-1, s+n-1, n, -1); } }
void memcpy(char *d, char *s, int n, int o) {
  if (0 < n) { *d = *s; memcpy(d+o, s+o, n-1, o) } }
```

Library $\mathbf{locle}$

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

Library $\mathbf{print}$

print : mov $\mathbf{x8}$, $\mathbf{PRINT}$; syscall; ret

Specification

$$\textbf{onetwo} \quad {}_{\mathsf{a}}\!\preceq_{\mathsf{s}} \quad \textbf{onetwo}_{\mathsf{spec}}$$

Specification

$$\mathbf{onetwo} \quad {}_{a}\!\preceq_{s} \quad \mathbf{onetwo}_{\mathsf{spec}}$$

$$\uparrow$$

$$\downarrow \mathsf{main} \cup_a \downarrow \mathsf{memmove}$$

$$\cup_a \mathbf{locle} \cup_a \mathbf{print}$$

Specification

Rec to Asm compilation

$\downarrow R$

$$\mathbf{onetwo} \quad {}_{a}{\lesssim}_{s} \quad \mathbf{onetwo}_{\text{spec}}$$

$$\downarrow \text{main} \cup_a \downarrow \text{memmove}$$

$$\cup_a \mathbf{locle} \cup_a \mathbf{print}$$

Specification

Rec to Asm compilation

$\downarrow R$

onetwo $\;{}_{a}{\preceq}_{s}\;$ **onetwo**$_{spec}$

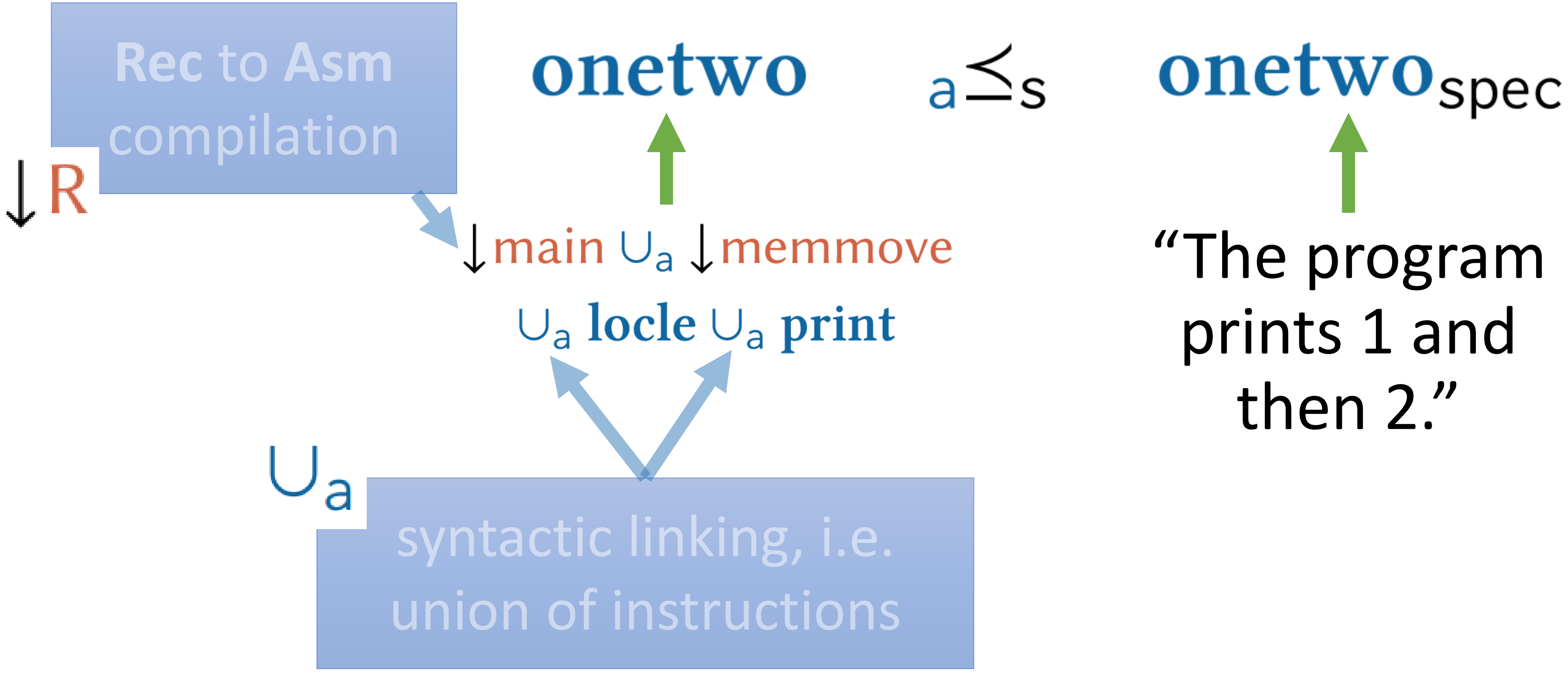
$\downarrow$main $\cup_a$ $\downarrow$memmove

$\cup_a$ **locle** $\cup_a$ **print**

$\cup_a$

syntactic linking, i.e. union of instructions

Specification



Specification

?

How to define refinement *in a decentralized fashion*?

$$\textbf{onetwo} \quad {}_{a}\!\preceq_{s} \textbf{onetwo}_{\text{spec}}$$

defined as

?

Specification

?

How to define refinement *in a decentralized fashion*?

$$\mathbf{onetwo} \;\; {}_{\mathsf{a}}{\preceq}_{\mathsf{s}} \; \mathbf{onetwo}_{\mathsf{spec}}$$

defined as

$$[\![\mathbf{onetwo}]\!]_{\mathsf{a}} \preceq \mathbf{onetwo}_{\mathsf{spec}}$$

!

Use *labeled transition systems* as semantic domain with *interaction via events*!

Specification

?

How to define refinement *in a decentralized fashion*?

$$\mathbf{onetwo} \;\; {}_{a}{\precsim}_{s} \; \mathbf{onetwo}_{\mathsf{spec}}$$

defined as

$$[\![\mathbf{onetwo}]\!]_{a} \precsim \mathbf{onetwo}_{\mathsf{spec}}$$

!

Use *labeled transition systems* as semantic domain with *interaction via events*!

$$\bullet \xrightarrow{\boxed{\text{Syscall!(PRINT, 1)}}} \bullet \xrightarrow{\boxed{\text{Syscall!(PRINT, 2)}}} \bullet$$

syntactic program to semantic LTS (i.e., module)

Specification

?

How to define refinement *in a decentralized fashion*?

$$\mathbf{onetwo} \ {}_{a}{\preceq}_{s}\ \mathbf{onetwo}_{\mathsf{spec}}$$

defined as

$$[\![\mathbf{onetwo}]\!]_{a} \preceq \mathbf{onetwo}_{\mathsf{spec}}$$

“are in simulation”

!

Use *labeled transition systems* as semantic domain with *interaction via events*!

Proof outline

$$
\begin{aligned}
[\![\text{onetwo}]\!]_a &= [\![{\downarrow}\text{main} \cup_a {\downarrow}\text{memmove} \cup_a \text{locle} \cup_a \text{print}]\!]_a \\
&\le [\![{\downarrow}\text{main}]\!]_a \oplus_a [\![{\downarrow}\text{memmove}]\!]_a \oplus_a [\![\text{locle}]\!]_a \oplus_a [\![\text{print}]\!]_a \\
&\le \lceil [\![\text{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\text{locle}]\!]_a \oplus_a [\![\text{print}]\!]_a \\
&\le \lceil [\![\text{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \text{locle}_{\text{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \text{print}_{\text{spec}} \\
&\le \lceil [\![\text{main} \cup_r \text{memmove}]\!]_r \oplus_r \text{locle}_{\text{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \text{print}_{\text{spec}} \\
&\le \lceil \text{main}_{\text{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \text{print}_{\text{spec}} \\
&\le \text{onetwo}_{\text{spec}}
\end{aligned}
$$

Proof outline

$$[\![\mathbf{onetwo}]\!]_a = [\![\downarrow \mathrm{main} \cup_a \downarrow \mathrm{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a$$

1. Modularize proof via semantic linking $\oplus_a$
2. Translate between languages via semantic wrapping $\lceil \cdot \rceil_{r \rightleftharpoons a}$
3. Language-local verification in **Rec**

$$\leq \mathbf{onetwo}_{\mathrm{spec}}$$

Proof outline

$$[\![\mathbf{onetwo}]\!]_a = [\![\downarrow\mathrm{main} \quad \cup_a \ \downarrow\mathrm{memmove} \quad \cup_a \ \mathbf{locle} \quad \cup_a \ \mathbf{print}]\!]_a$$

1. Modularize proof via semantic linking $\oplus_a$
2. Translate between languages via semantic wrapping $\lceil\cdot\rceil_{r\rightleftharpoons a}$
3. Language-local verification in **Rec**

$$\leq \mathbf{onetwo}_{\mathrm{spec}}$$

Syntactic vs. semantic linking

$$
\begin{aligned}
[\![\textbf{onetwo}]\!]_a &= [\![{\downarrow}\mathrm{main} \quad \cup_a \; {\downarrow}\mathrm{memmove} \quad \cup_a \; \textbf{locle} \quad \cup_a \; \textbf{print}]\!]_a \\
&\preceq [\![{\downarrow}\mathrm{main}]\!]_a \quad \oplus_a [\![{\downarrow}\mathrm{memmove}]\!]_a \quad \oplus_a [\![\textbf{locle}]\!]_a \quad \oplus_a [\![\textbf{print}]\!]_a
\end{aligned}
$$

Syntactic vs. semantic linking

$\cup_a$ syntactic linking, i.e. union of instructions

$$
\begin{aligned}
[\![\textbf{onetwo}]\!]_a &= [\![{\downarrow}\mathrm{main} \quad \cup_a \; {\downarrow}\mathrm{memmove} \quad \cup_a \; \textbf{locle} \quad \cup_a \; \textbf{print}]\!]_a \\
&\lesssim [\![{\downarrow}\mathrm{main}]\!]_a \quad \oplus_a [\![{\downarrow}\mathrm{memmove}]\!]_a \quad \oplus_a [\![\textbf{locle}]\!]_a \quad \oplus_a [\![\textbf{print}]\!]_a
\end{aligned}
$$

Syntactic vs. semantic linking

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow\mathrm{main} \cup_a \downarrow\mathrm{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\preceq [\![\downarrow\mathrm{main}]\!]_a \oplus_a [\![\downarrow\mathrm{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a
\end{aligned}
$$

$\cup_a$ syntactic linking, i.e. union of instructions

$\oplus_a$ semantic linking, i.e. synchronization on events

Syntactic vs. semantic linking

$\cup_a$ syntactic linking, i.e. union of instructions

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![{\downarrow}\mathrm{main} \;\cup_a\; {\downarrow}\mathrm{memmove} \;\cup_a\; \mathbf{locle} \;\cup_a\; \mathbf{print}]\!]_a \\
&\lesssim [\![{\downarrow}\mathrm{main}]\!]_a \;\oplus_a\; [\![{\downarrow}\mathrm{memmove}]\!]_a \;\oplus_a\; [\![\mathbf{locle}]\!]_a \;\oplus_a\; [\![\mathbf{print}]\!]_a
\end{aligned}
$$

$\oplus_a$ semantic linking, i.e. synchronization on events

$\oplus_a$ matches outgoing **Jump!** with incoming **Jump?**

$$
\begin{aligned}
\mathbf{Events} \ni \mathbf{e} ::= {} & \mathbf{Jump!}(\mathbf{r}, \mathbf{m}) \mid \mathbf{Jump?}(\mathbf{r}, \mathbf{m}) \\
\mid {} & \mathbf{Syscall!}(\mathbf{v}_1, \mathbf{v}_2, \mathbf{m}) \mid \mathbf{SyscallRet?}(\mathbf{v}, \mathbf{m})
\end{aligned}
$$

Syntactic vs. semantic linking

$\cup_a$ syntactic linking, i.e. union of instructions

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow\mathrm{main} \quad \cup_a \ \downarrow\mathrm{memmove} \quad \cup_a \ \mathbf{locle} \quad \cup_a \ \mathbf{print}]\!]_a \\
&\leq [\![\downarrow\mathrm{main}]\!]_a \quad \oplus_a [\![\downarrow\mathrm{memmove}]\!]_a \quad \oplus_a [\![\mathbf{locle}]\!]_a \quad \oplus_a [\![\mathbf{print}]\!]_a
\end{aligned}
$$

$\oplus_a$ semantic linking, i.e. synchronization on events

Key property: Horizontal compositionality

ASM-LINK-HORIZONTAL

$$
\frac{\mathbf{M}_1 \leq \mathbf{M}_1' \qquad \mathbf{M}_2 \leq \mathbf{M}_2'}{\mathbf{M}_1 \oplus_a \mathbf{M}_2 \leq \mathbf{M}_1' \oplus_a \mathbf{M}_2'}
$$

$\oplus_a$ enables *modular reasoning* using $\leq$.

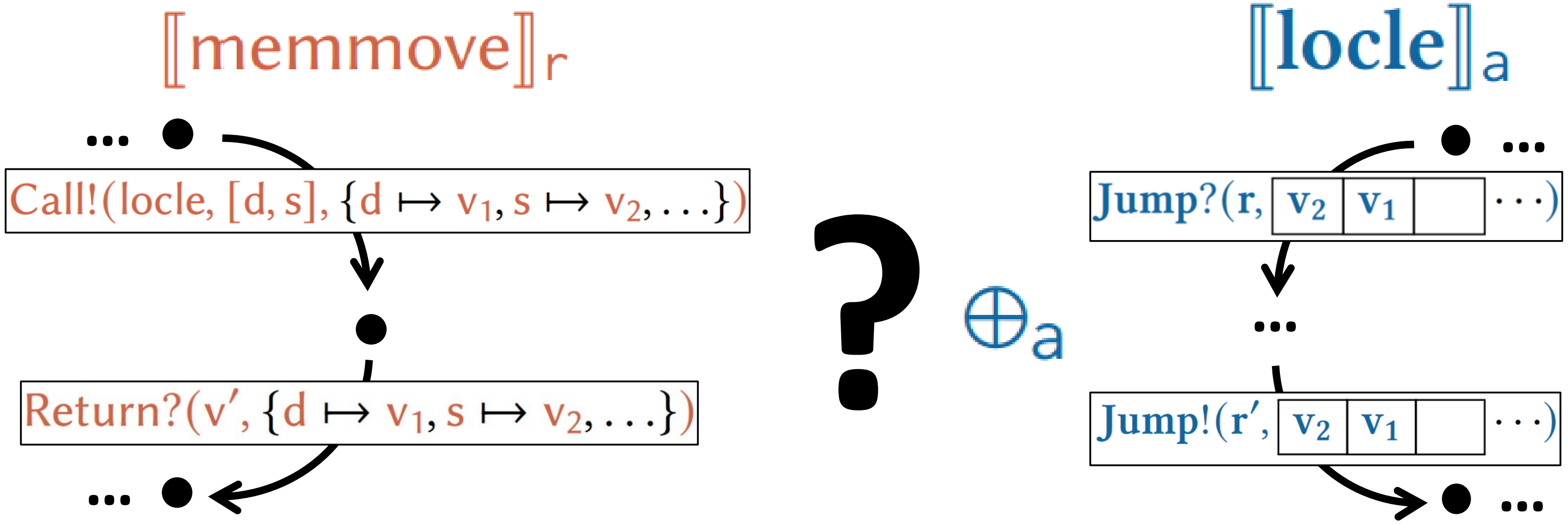
Proof outline

$$[\![\mathbf{onetwo}]\!]_a = [\![\downarrow\mathrm{main} \;\cup_a\; \downarrow\mathrm{memmove} \;\cup_a\; \mathbf{locle} \;\cup_a\; \mathbf{print}]\!]_a$$

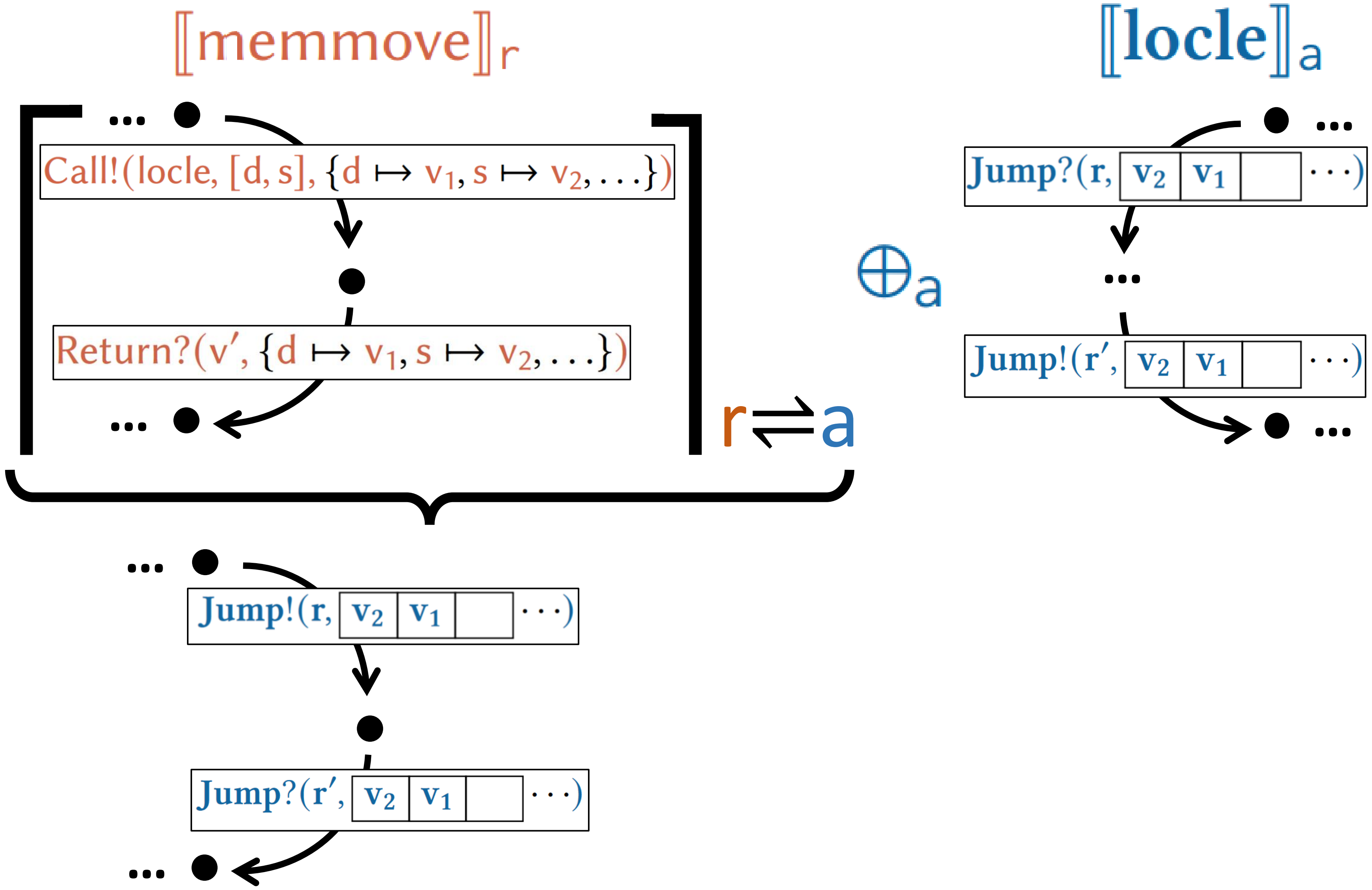
1. Modularize proof via semantic linking $\oplus_a$
2. Translate between languages via semantic wrapping $\lceil\cdot\rceil_{r\rightleftharpoons a}$
3. Language-local verification in **Rec**

$$\leq \mathbf{onetwo}_{\mathrm{spec}}$$

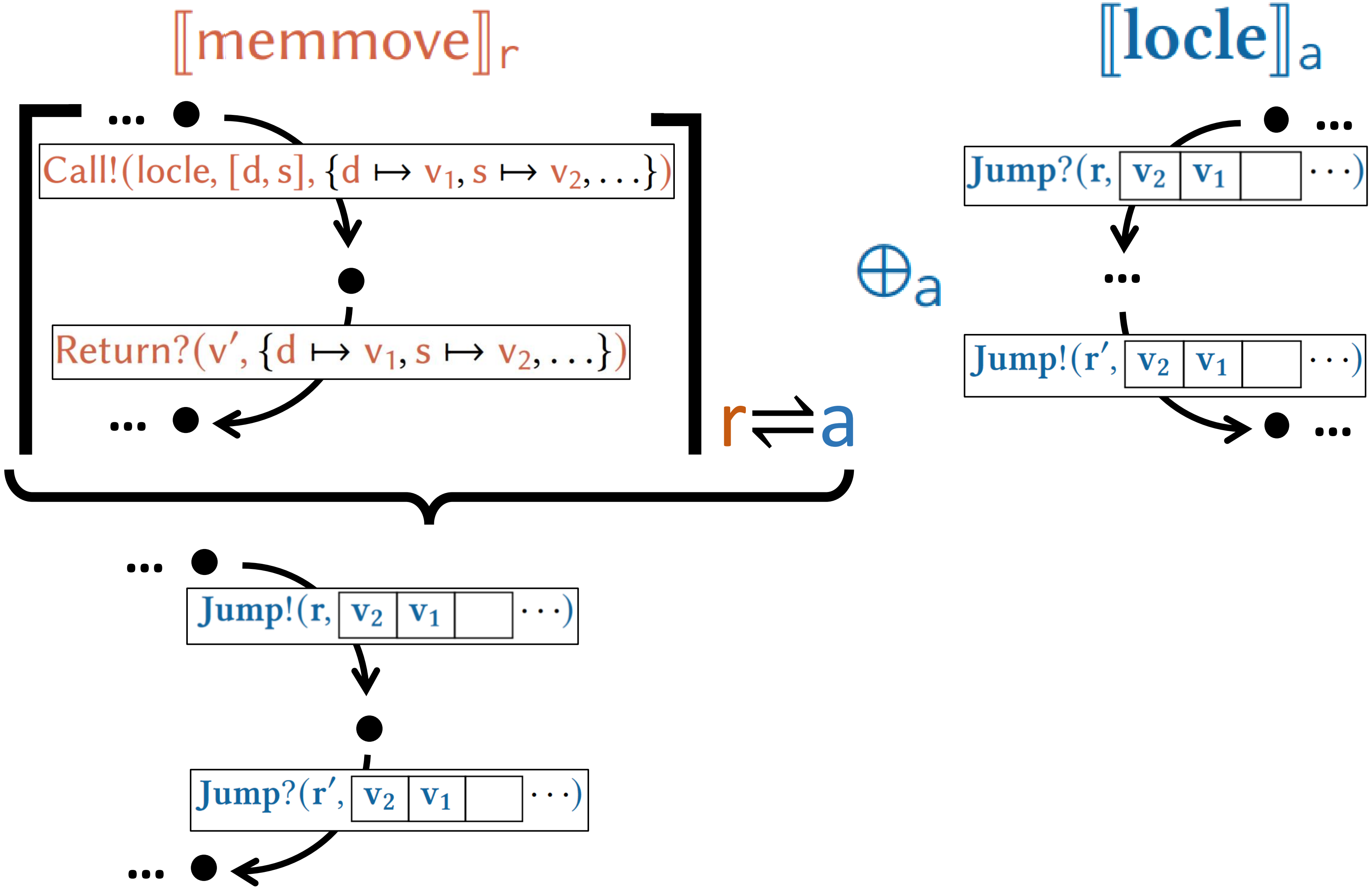
Translating between languages: wrapper $\lceil \cdot \rceil_{r \rightleftharpoons a}$



Translating between languages: wrapper $\lceil \cdot \rceil_{r \rightleftharpoons a}$

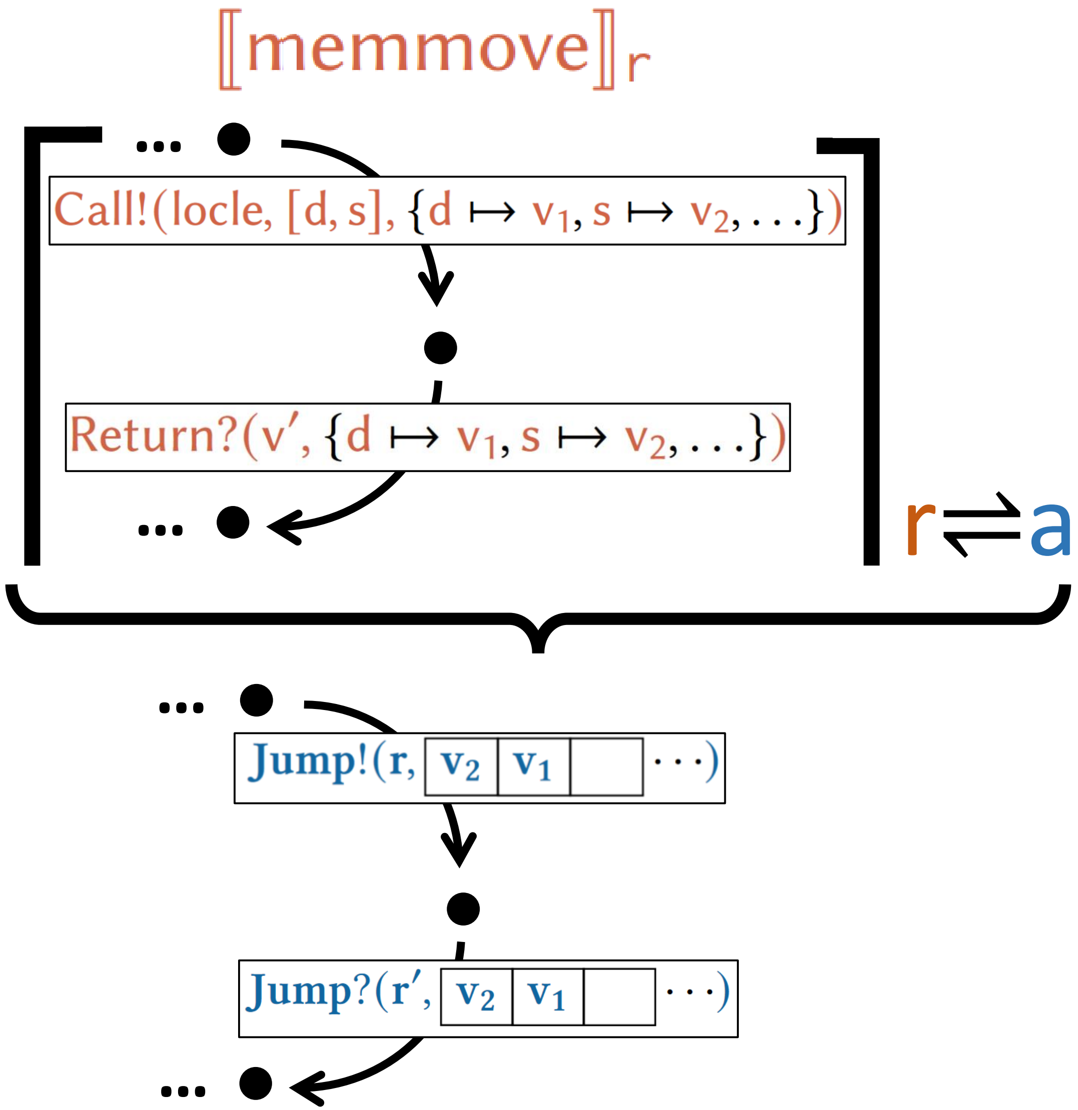


Translating between languages: wrapper $\lceil\cdot\rceil_{r\rightleftharpoons a}$



Desideratum #2: $\lceil\cdot\rceil_{r\rightleftharpoons a}$ enables interoperation between language and memory models

Translating between languages: wrapper $\lceil\cdot\rceil_{r \rightleftharpoons a}$



Key technical idea:
rely-guarantee protocol via *demonic* and *angelic* non-determinism

Translating between languages: wrapper $\lceil \cdot \rceil_{r \rightleftharpoons a}$

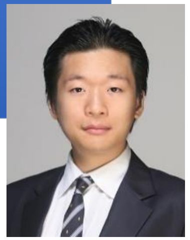
$[\![\text{memmove}]\!]_r$

$$\left\lceil \begin{array}{l} \ldots \bullet \\ \boxed{\text{Call!}(\text{locle}, [d, s], \{d \mapsto v_1, s \mapsto v_2, \ldots\})} \\ \bullet \\ \boxed{\text{Return?}(v', \{d \mapsto v_1, s \mapsto v_2, \ldots\})} \\ \ldots \bullet \end{array} \right\rceil_{r \rightleftharpoons a}$$

$$\begin{array}{l} \ldots \bullet \\ \boxed{\text{Jump!}(r, \boxed{v_2}\boxed{v_1}\boxed{} \cdots)} \\ \bullet \\ \boxed{\text{Jump?}(r', \boxed{v_2}\boxed{v_1}\boxed{} \cdots)} \\ \ldots \bullet \end{array}$$

Key technical idea:
rely-guarantee protocol via
demonic and *angelic*
non-determinism

inspired by *Conditional Contextual Refinement* [Song et al., POPL'23]



Compiler correctness

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![{\downarrow}\mathsf{main} &&\cup_a {\downarrow}\mathsf{memmove} &&\cup_a \mathbf{locle} &&\cup_a \mathbf{print}]\!]_a \\
&\leq [\![{\downarrow}\mathsf{main}]\!]_a &&\oplus_a [\![{\downarrow}\mathsf{memmove}]\!]_a &&\oplus_a [\![\mathbf{locle}]\!]_a &&\oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} &&\oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} &&\oplus_a [\![\mathbf{locle}]\!]_a &&\oplus_a [\![\mathbf{print}]\!]_a
\end{aligned}
$$

Compiler correctness

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow \mathrm{main} \quad \cup_a \ \downarrow \mathrm{memmove} \quad \cup_a \ \mathbf{locle} \quad \cup_a \ \mathbf{print}]\!]_a \\
&\leq [\![\downarrow \mathrm{main}]\!]_a \oplus_a [\![\downarrow \mathrm{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathrm{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a
\end{aligned}
$$

COMPILER-CORRECT

$$
[\![\downarrow R]\!]_a \leq \lceil [\![R]\!]_r \rceil_{r \rightleftharpoons a}
$$

Compiler correctness

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![{\downarrow}\mathsf{main} && \cup_a\ {\downarrow}\mathsf{memmove} && \cup_a\ \mathbf{locle} && \cup_a\ \mathbf{print}]\!]_a \\
&\leq [\![{\downarrow}\mathsf{main}]\!]_a && \oplus_a [\![{\downarrow}\mathsf{memmove}]\!]_a && \oplus_a [\![\mathbf{locle}]\!]_a && \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} && \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} && \oplus_a [\![\mathbf{locle}]\!]_a && \oplus_a [\![\mathbf{print}]\!]_a
\end{aligned}
$$

syntactically translated

COMPILER-CORRECT

$$[\![{\downarrow}R]\!]_a \leq \lceil [\![R]\!]_r \rceil_{r \rightleftharpoons a}$$

Compiler correctness

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![{\downarrow}\mathsf{main} \cup_a {\downarrow}\mathsf{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\leq [\![{\downarrow}\mathsf{main}]\!]_a \oplus_a [\![{\downarrow}\mathsf{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a
\end{aligned}
$$

syntactically translated

COMPILER-CORRECT

$$[\![{\downarrow}R]\!]_a \leq \lceil [\![R]\!]_r \rceil_{r \rightleftharpoons a}$$

semantically translated

Abstracting Asm to Rec transition system

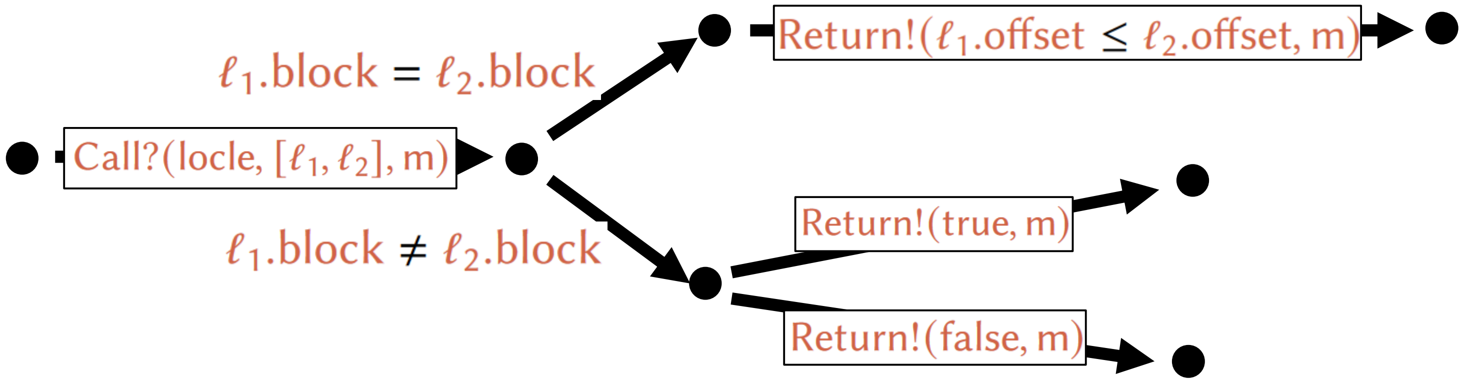
$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow \mathsf{main} \cup_a \downarrow \mathsf{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\le [\![\downarrow \mathsf{main}]\!]_a \oplus_a [\![\downarrow \mathsf{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\le \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\le \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \mathsf{locle}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}}
\end{aligned}
$$

$$\textsc{locle-correct}\quad [\![\mathbf{locle}]\!]_a \le \lceil \mathsf{locle}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a}$$

Abstracting Asm to Rec transition system

$$
\begin{aligned}
[\![\mathrm{onetwo}]\!]_a &= [\![{\downarrow}\mathrm{main} \cup_a {\downarrow}\mathrm{memmove} \cup_a \mathrm{locle} \cup_a \mathrm{print}]\!]_a \\
&\leq [\![{\downarrow}\mathrm{main}]\!]_a \oplus_a [\![{\downarrow}\mathrm{memmove}]\!]_a \oplus_a [\![\mathrm{locle}]\!]_a \oplus_a [\![\mathrm{print}]\!]_a \\
&\leq \lceil [\![\mathrm{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathrm{locle}]\!]_a \oplus_a [\![\mathrm{print}]\!]_a \\
&\leq \lceil [\![\mathrm{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \mathrm{locle}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathrm{print}_{\mathrm{spec}}
\end{aligned}
$$

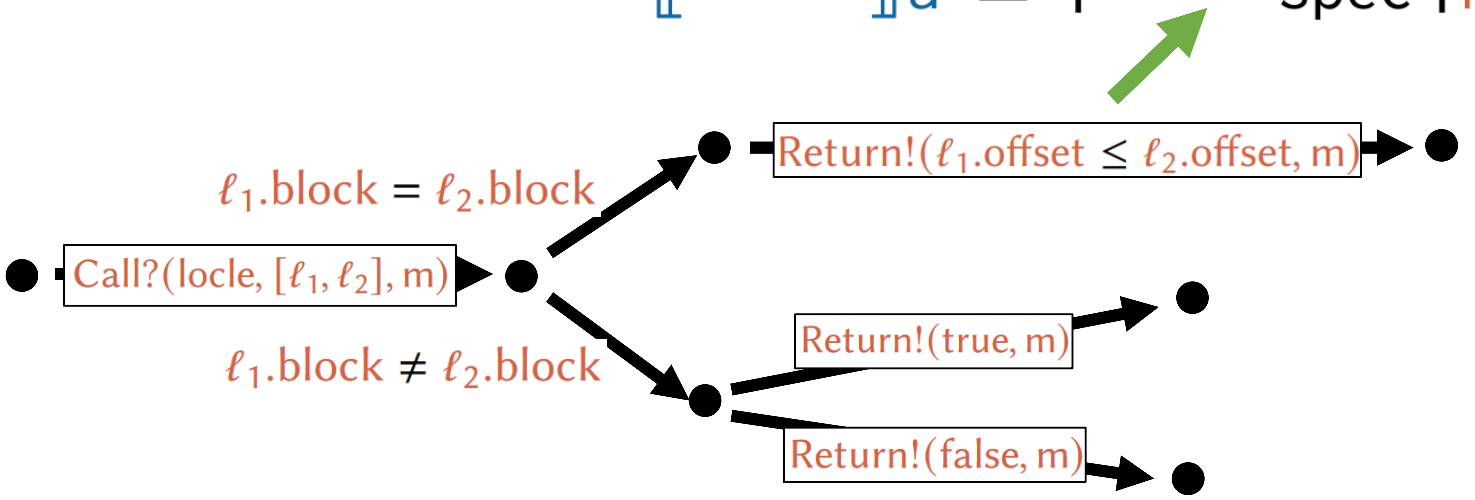
LOCLE-CORRECT $[\![\mathrm{locle}]\!]_a \leq \lceil \mathrm{locle}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a}$



Abstracting Asm to Rec transition system

$$
\begin{aligned}
\llbracket \mathrm{onetwo} \rrbracket_a &= \llbracket \downarrow \mathrm{main} \cup_a \downarrow \mathrm{memmove} \cup_a \mathrm{locle} \cup_a \mathrm{print} \rrbracket_a \\
&\precsim \llbracket \downarrow \mathrm{main} \rrbracket_a \oplus_a \llbracket \downarrow \mathrm{memmove} \rrbracket_a \oplus_a \llbracket \mathrm{locle} \rrbracket_a \oplus_a \llbracket \mathrm{print} \rrbracket_a \\
&\precsim \lceil \llbracket \mathrm{main} \rrbracket_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \llbracket \mathrm{memmove} \rrbracket_r \rceil_{r \rightleftharpoons a} \oplus_a \llbracket \mathrm{locle} \rrbracket_a \oplus_a \llbracket \mathrm{print} \rrbracket_a \\
&\precsim \lceil \llbracket \mathrm{main} \rrbracket_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \llbracket \mathrm{memmove} \rrbracket_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \mathrm{locle}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathrm{print}_{\mathrm{spec}}
\end{aligned}
$$

LOCLE-CORRECT $\llbracket \mathrm{locle} \rrbracket_a \precsim \lceil \mathrm{locle}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a}$



Desideratum #1

No syntactic **Rec** program required!

Proof outline

$$[\![\textbf{onetwo}]\!]_a = [\![\downarrow \text{main} \cup_a \downarrow \text{memmove} \cup_a \textbf{locle} \cup_a \textbf{print}]\!]_a$$

1. Modularize proof via semantic linking $\oplus_a$
2. Translate between languages via semantic wrapping $\lceil \cdot \rceil_{r \rightleftharpoons a}$
3. Language-local verification in **Rec**

$$\leq \textbf{onetwo}_{\text{spec}}$$

Bundling Rec modules

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow \mathsf{main} \cup_a \downarrow \mathsf{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\preceq [\![\downarrow \mathsf{main}]\!]_a \oplus_a [\![\downarrow \mathsf{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\preceq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\preceq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \mathsf{locle}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \lceil [\![\mathsf{main} \cup_r \mathsf{memmove}]\!]_r \oplus_r \mathsf{locle}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}}
\end{aligned}
$$

Bundling Rec modules

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![{\downarrow}\mathrm{main} \cup_a {\downarrow}\mathrm{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\leq [\![{\downarrow}\mathrm{main}]\!]_a \oplus_a [\![{\downarrow}\mathrm{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathrm{main}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathrm{main}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil \mathrm{locle}_{\mathrm{spec}} \rceil_{r\rightleftharpoons a} \oplus_a \mathbf{print}_{\mathrm{spec}} \\
&\leq \lceil [\![\mathrm{main} \cup_r \mathrm{memmove}]\!]_r \oplus_r \mathrm{locle}_{\mathrm{spec}} \rceil_{r\rightleftharpoons a} \oplus_a \mathbf{print}_{\mathrm{spec}}
\end{aligned}
$$

REC-TO-ASM-LINK

$$\lceil M_1 \rceil_{r\rightleftharpoons a} \oplus_a \lceil M_2 \rceil_{r\rightleftharpoons a} \leq \lceil M_1 \oplus_r M_2 \rceil_{r\rightleftharpoons a}$$

Rec-level reasoning

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow \mathsf{main} \cup_a \downarrow \mathsf{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\leq [\![\downarrow \mathsf{main}]\!]_a \oplus_a [\![\downarrow \mathsf{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \mathsf{locle}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\leq \lceil [\![\mathsf{main} \cup_r \mathsf{memmove}]\!]_r \oplus_r \mathsf{locle}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\leq \lceil \mathsf{main}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}}
\end{aligned}
$$

Rec-level reasoning

$$
\begin{aligned}
[\![\mathrm{onetwo}]\!]_a &= [\![{\downarrow}\mathrm{main} \cup_a {\downarrow}\mathrm{memmove} \cup_a \mathrm{locle} \cup_a \mathrm{print}]\!]_a \\
&\preceq [\![{\downarrow}\mathrm{main}]\!]_a \oplus_a [\![{\downarrow}\mathrm{memmove}]\!]_a \oplus_a [\![\mathrm{locle}]\!]_a \oplus_a [\![\mathrm{print}]\!]_a \\
&\preceq \lceil [\![\mathrm{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathrm{locle}]\!]_a \oplus_a [\![\mathrm{print}]\!]_a \\
&\preceq \lceil [\![\mathrm{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \mathrm{locle}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathrm{print}_{\mathrm{spec}} \\
&\preceq \lceil [\![\mathrm{main} \cup_r \mathrm{memmove}]\!]_r \oplus_r \mathrm{locle}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathrm{print}_{\mathrm{spec}} \\
&\preceq \lceil \mathrm{main}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathrm{print}_{\mathrm{spec}}
\end{aligned}
$$

$$
[\![\mathrm{main} \cup_r \mathrm{memmove}]\!]_r \oplus_r \mathrm{locle}_{\mathrm{spec}} \preceq \mathrm{main}_{\mathrm{spec}}
$$

Rec-level reasoning

Desideratum #3:
Language-local reasoning
(independent of **Asm**)

$$
\begin{aligned}
[\![\mathrm{onetwo}]\!]_a &= [\![\downarrow \mathrm{main} \quad \cup_a \; \downarrow \qquad\qquad \cup_a \; \mathrm{print}]\!]_a \\
&\preceq [\![\downarrow \mathrm{main}]\!]_a \quad \oplus_a [\![\downarrow \qquad\qquad \oplus_a [\![\mathrm{print}]\!]_a \\
&\preceq \lceil [\![\mathrm{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathrm{locle}]\!]_a \oplus_a [\![\mathrm{print}]\!]_a \\
&\preceq \lceil [\![\mathrm{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathrm{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \mathrm{locle}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathrm{print}_{\mathrm{spec}} \\
&\preceq \lceil [\![\mathrm{main} \cup_r \mathrm{memmove}]\!]_r \oplus_r \mathrm{locle}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathrm{print}_{\mathrm{spec}} \\
&\preceq \lceil \mathrm{main}_{\mathrm{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathrm{print}_{\mathrm{spec}}
\end{aligned}
$$

$$
[\![\mathrm{main} \cup_r \mathrm{memmove}]\!]_r \oplus_r \mathrm{locle}_{\mathrm{spec}} \preceq \mathrm{main}_{\mathrm{spec}}
$$

Rec-level reasoning

Desideratum #3:
Language-local reasoning (independent of **Asm**)

via

$$\textsc{Rec-Wrapper-Compat}\quad \frac{M \preceq M'}{\lceil M \rceil_{r\rightleftharpoons a} \preceq \lceil M' \rceil_{r\rightleftharpoons a}}$$

$$\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow \mathsf{main} \quad \cup_a \ \downarrow \\
&\preceq [\![\downarrow \mathsf{main}]\!]_a \quad \oplus_a [\![\downarrow \quad \oplus_a [\![\mathbf{print}]\!]_a \\
&\preceq \lceil [\![\mathsf{main}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\preceq \lceil [\![\mathsf{main}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil \mathsf{locle}_{\mathsf{spec}} \rceil_{r\rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \lceil [\![\mathsf{main} \cup_r \mathsf{memmove}]\!]_r \oplus_r \mathsf{locle}_{\mathsf{spec}} \rceil_{r\rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \lceil \mathsf{main}_{\mathsf{spec}} \rceil_{r\rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}}
\end{aligned}$$

$$[\![\mathsf{main} \cup_r \mathsf{memmove}]\!]_r \oplus_r \mathsf{locle}_{\mathsf{spec}} \preceq \mathsf{main}_{\mathsf{spec}}$$

Reasoning with specifications

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![{\downarrow}\mathsf{main} \cup_a {\downarrow}\mathsf{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\preceq [\![{\downarrow}\mathsf{main}]\!]_a \oplus_a [\![{\downarrow}\mathsf{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\preceq \lceil [\![\mathsf{main}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\preceq \lceil [\![\mathsf{main}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r\rightleftharpoons a} \oplus_a \lceil \mathsf{locle}_{\mathsf{spec}} \rceil_{r\rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \lceil [\![\mathsf{main} \cup_r \mathsf{memmove}]\!]_r \oplus_r \mathsf{locle}_{\mathsf{spec}} \rceil_{r\rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \lceil \mathsf{main}_{\mathsf{spec}} \rceil_{r\rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \mathbf{onetwo}_{\mathsf{spec}}
\end{aligned}
$$

Complete verification

$$
\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow \mathsf{main} \cup_a \downarrow \mathsf{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\preceq [\![\downarrow \mathsf{main}]\!]_a \oplus_a [\![\downarrow \mathsf{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\preceq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\preceq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil \mathsf{locle}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \lceil [\![\mathsf{main} \cup_r \mathsf{memmove}]\!]_r \oplus_r \mathsf{locle}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \lceil \mathsf{main}_{\mathsf{spec}} \rceil_{r \rightleftharpoons a} \oplus_a \mathbf{print}_{\mathsf{spec}} \\
&\preceq \mathbf{onetwo}_{\mathsf{spec}}
\end{aligned}
$$

Recap: Desiderata for multi-language reasoning

Recap: Desiderata for multi-language reasoning

#1 No fixed source language as specification language

Labeled transition systems as semantic domain

Recap: Desiderata for multi-language reasoning

#1 No fixed source language as specification language

Labeled transition systems as semantic domain

#2 No fixed memory model

Wrappers like $\lceil \cdot \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}}$ translate between memory models

Recap: Desiderata for multi-language reasoning

#1 No fixed source language as specification language

Labeled transition systems as semantic domain

#2 No fixed memory model

Wrappers like $\lceil \cdot \rceil_{r \rightleftharpoons a}$ translate between memory models

#3 No fixed set of languages

Language-local reasoning via compatibility with refinement

Recap: Desiderata for multi-language reasoning

#1 No fixed source language as specification language

Labeled transition systems as semantic domain

#2 No fixed memory model

Wrappers like $\lceil \cdot \rceil_{r \rightleftharpoons a}$ translate between memory models

#3 No fixed set of languages

Language-local reasoning via compatibility with refinement

#4 No fixed notion of linking

DimSum allows defining custom linking operators

Recap: Desiderata for multi-language reasoning

DimSum

https://plv.mpi-sws.org/dimsum

#1 No fixed source language as specification language

Labeled transition systems as semantic domain

#2 No fixed memory model

Wrappers like $\lceil \cdot \rceil_{r \rightleftharpoons a}$ translate between memory models

#3 No fixed set of languages

Language-local reasoning via compatibility with refinement

#4 No fixed notion of linking

DimSum allows defining custom linking operators https://plv.mpi-sws.org/dimsum

Questions?

In the paper

Wrappers via demonic and angelic non-determinism

$$\lceil \cdot \rceil_{r \rightleftharpoons a}$$

Verification of $\downarrow R$

$$[\![R]\!]_r \geq [\![R_{SSA}]\!]_r \qquad [\![R_{SSA}]\!]_r \geq [\![R_{lin}]\!]_r \qquad \lceil [\![R_{lin}]\!]_r \rceil_{r \rightleftharpoons r} \geq [\![R_{opt}]\!]_r \qquad \lceil [\![R_{opt}]\!]_r \rceil_{r \rightleftharpoons a} \geq [\![\downarrow R]\!]_a$$

$$\boxed{\begin{array}{c}R\\(\mathrm{Rec})\end{array}} \xrightarrow{\mathit{SSA}} \boxed{\begin{array}{c}R_{SSA}\\(\mathrm{Rec})\end{array}} \xrightarrow{\mathit{Linearize}} \boxed{\begin{array}{c}R_{lin}\\(\mathrm{LinearRec})\end{array}} \xrightarrow{\mathit{Mem2Reg}} \boxed{\begin{array}{c}R_{opt}\\(\mathrm{LinearRec})\end{array}} \xrightarrow{\mathit{Codegen}} \boxed{\begin{array}{c}\downarrow R\\(\mathrm{Asm})\end{array}}$$

Language-generic combinators

$$M_1 \times M_2 \qquad M_1 \setminus M_2$$

$$\lceil M \rceil_X \qquad M_1 \oplus_X M_2$$

Operational semantics for dual non-determinism

$$\rightarrow \; \in \mathcal{P}(S \times \mathrm{option}(E) \times \mathcal{P}(S))$$

Coroutines

$$[\![\mathrm{main}]\!]_r \oplus_{coro} [\![\mathrm{stream}]\!]_r$$

#1 No fixed source / spec. language

#2 No fixed memory model

#3 No fixed set of languages

#4 No fixed notion of linking

DimSum

Questions?

https://plv.mpi-sws.org/dimsum

Decentralized Multi-language Reasoning

Combining ideas from

process algebra, *wrappers*, *Kripke relations*, and *angelic non-determinism*

Instantiations

Rec: *C-like language*

Asm: *assembly language*

Spec: *specification language*

Evaluation

$\downarrow R$: Compiler from *Rec* to *Asm*

locle : pointer comparison

$\oplus_{\mathrm{coro}}$: coroutines

Dimsum icon by Icons8

Backup slides

What next?

Other language features

e.g. closures, concurrency, …

What next?

Other language features

e.g. closures, concurrency, …

Combine with existing verification tools

e.g. RefinedC, Islaris, …

What next?

Other language features

e.g. closures, concurrency, …

Combine with existing verification tools

e.g. RefinedC, Islaris, …

Investigate meta-level properties

e.g. boundary cancellation, …

What next?

Other language features
e.g. closures, concurrency, …

Combine with existing verification tools
e.g. RefinedC, Islaris, …

Investigate meta-level properties
e.g. boundary cancellation, …

…

Events

Asm

$$\mathrm{Events} \ni e ::= \mathrm{Jump!}(r, m) \mid \mathrm{Jump?}(r, m) \mid \mathrm{Syscall!}(v_1, v_2, m) \mid \mathrm{SyscallRet?}(v, m)$$

Rec

$$\mathrm{Events} \ni e ::= \mathrm{Call!}(f, \overline{v}, m) \mid \mathrm{Call?}(f, \overline{v}, m) \mid \mathrm{Return!}(\overline{v}, m) \mid \mathrm{Return?}(\overline{v}, m)$$

Events

unstructured jumps vs. call and return

Asm

$$\mathrm{Events} \ni e ::= \mathrm{Jump!}(r, m) \mid \mathrm{Jump?}(r, m)$$
$$\mid \mathrm{Syscall!}(v_1, v_2, m) \mid \mathrm{SyscallRet?}(v, m)$$

Rec

$$\mathrm{Events} \ni e ::= \mathrm{Call!}(f, \overline{v}, m) \mid \mathrm{Call?}(f, \overline{v}, m)$$
$$\mid \mathrm{Return!}(\overline{v}, m) \mid \mathrm{Return?}(\overline{v}, m)$$

Events

unstructured jumps vs. call and return

Asm

$$\text{Events} \ni e ::= \mathrm{Jump!}(r, m) \mid \mathrm{Jump?}(r, m) \mid \mathrm{Syscall!}(v_1, v_2, m) \mid \mathrm{SyscallRet?}(v, m)$$

system calls

Rec

$$\text{Events} \ni e ::= \mathrm{Call!}(f, \overline{v}, m) \mid \mathrm{Call?}(f, \overline{v}, m) \mid \mathrm{Return!}(\overline{v}, m) \mid \mathrm{Return?}(\overline{v}, m)$$

Events

unstructured jumps vs. call and return

Asm

$$\text{Events} \ni e ::= \text{Jump!}(r, m) \mid \text{Jump?}(r, m)$$
$$\mid \text{Syscall!}(v_1, v_2, m) \mid \text{SyscallRet?}(v, m)$$

system calls

$$\text{Val} \ni v \triangleq \mathbb{Z}$$

Rec

$$\text{Events} \ni e ::= \text{Call!}(f, \overline{v}, m) \mid \text{Call?}(f, \overline{v}, m)$$
$$\mid \text{Return!}(\overline{v}, m) \mid \text{Return?}(\overline{v}, m)$$

$$\text{Val} \ni v ::= z : \mathbb{Z} \mid b : \mathbb{B} \mid \ell : \text{Loc}$$

Events

unstructured jumps vs. call and return

Asm

$$\mathrm{Events} \ni e ::= \mathrm{Jump!}(r, m) \mid \mathrm{Jump?}(r, m)$$
$$\mid \mathrm{Syscall!}(v_1, v_2, m) \mid \mathrm{SyscallRet?}(v, m)$$

system calls

Rec

$$\mathrm{Events} \ni e ::= \mathrm{Call!}(f, \overline{v}, m) \mid \mathrm{Call?}(f, \overline{v}, m)$$
$$\mid \mathrm{Return!}(\overline{v}, m) \mid \mathrm{Return?}(\overline{v}, m)$$

unstructured vs. structured values

$$\mathrm{Val} \ni v \triangleq \mathbb{Z}$$

$$\mathrm{Val} \ni v ::= z : \mathbb{Z} \mid b : \mathbb{B} \mid \ell : \mathrm{Loc}$$

Events

unstructured jumps vs. call and return

Asm

$$\mathrm{Events} \ni e ::= \mathrm{Jump!}(r, m) \mid \mathrm{Jump?}(r, m)$$
$$\mid \mathrm{Syscall!}(v_1, v_2, m) \mid \mathrm{SyscallRet?}(v, m)$$

system calls

Rec

$$\mathrm{Events} \ni e ::= \mathrm{Call!}(f, \overline{v}, m) \mid \mathrm{Call?}(f, \overline{v}, m)$$
$$\mid \mathrm{Return!}(\overline{v}, m) \mid \mathrm{Return?}(\overline{v}, m)$$

unstructured vs. structured values

$$\mathrm{Val} \ni v \triangleq \mathbb{Z}$$

$$\mathrm{Val} \ni v ::= z : \mathbb{Z} \mid b : \mathbb{B} \mid \ell : \mathrm{Loc}$$

$$\mathrm{Registers} \ni r \triangleq \mathrm{RegisterName} \to \mathrm{Val}$$
$$\mathrm{Memory} \ni m \triangleq \mathbb{Z} \xrightarrow{\mathrm{fin}} \mathrm{Val} \cup \{\#\}$$

$$\mathrm{Memory} \ni m \triangleq \mathrm{Loc} \xrightarrow{\mathrm{fin}} \mathrm{Val}$$
$$\mathrm{Loc} \ni \ell ::= \{\mathrm{blockid} : \mathrm{Id},\ \mathrm{offset} : \mathbb{Z}\}$$

Events

unstructured jumps vs. call and return

Asm

$$\text{Events} \ni e ::= \text{Jump!}(r, m) \mid \text{Jump?}(r, m) \mid \text{Syscall!}(v_1, v_2, m) \mid \text{SyscallRet?}(v, m)$$

system calls

$$\text{Val} \ni v \triangleq \mathbb{Z}$$

$$\text{Registers} \ni r \triangleq \text{RegisterName} \to \text{Val}$$

$$\text{Memory} \ni m \triangleq \mathbb{Z} \xrightarrow{\text{fin}} \text{Val} \cup \{\#\}$$

Rec

$$\text{Events} \ni e ::= \text{Call!}(f, \overline{v}, m) \mid \text{Call?}(f, \overline{v}, m) \mid \text{Return!}(\overline{v}, m) \mid \text{Return?}(\overline{v}, m)$$

$$\text{Val} \ni v ::= z : \mathbb{Z} \mid b : \mathbb{B} \mid \ell : \text{Loc}$$

$$\text{Memory} \ni m \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{Val}$$

$$\text{Loc} \ni \ell ::= \{\text{blockid} : \text{Id}, \text{offset} : \mathbb{Z}\}$$

unstructured vs. structured values

flat vs. block-based memory

Specification

$$\mathrm{Spec}(E) \ni p ::=_{\mathrm{coind}} \mathsf{any} \mid \mathsf{vis}(e); p \mid \mathsf{assume}(\phi); p \mid \exists x : T; p(x) \mid \cdots$$

Specification

$$\mathrm{Spec}(E) \ni p ::=_{\mathsf{coind}} \mathsf{any} \mid \mathsf{vis}(e); p \mid \mathsf{assume}(\phi); p \mid \exists x : T; p(x) \mid \cdots$$

$$\begin{aligned}\mathbf{onetwo}_{\mathsf{spec}} \triangleq\ & \exists \mathbf{r}, \mathbf{m}_0; \mathsf{vis}(\mathbf{Jump?}(\mathbf{r}, \mathbf{m}_0)); \mathsf{assume}(\mathbf{r}(\mathbf{pc}) = a_{\mathsf{main}} \land \mathbf{has\_stack}(\mathbf{r}(\mathbf{sp}), \mathbf{m}_0));\\ & \exists \mathbf{m}_1; \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, 1, \mathbf{m}_1)); \exists \mathbf{m}_2; \mathsf{vis}(\mathbf{SyscallRet?}(*, \mathbf{m}_2)); \mathsf{assume}(\mathbf{m}_2 = \mathbf{m}_1);\\ & \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, 2, *)); \mathsf{vis}(\mathbf{SyscallRet?}(*, *)); \mathsf{any}\end{aligned}$$

Specification

$$\mathsf{Spec}(E) \ni p ::=_{\mathsf{coind}} \mathsf{any} \mid \mathsf{vis}(e); p \mid \mathsf{assume}(\phi); p \mid \exists x : T; p(x) \mid \cdots$$

≈ “Assuming the environment calls the main function, …”

$$\begin{aligned}\mathbf{onetwo}_{\mathsf{spec}} \triangleq\ & \exists \mathbf{r}, \mathbf{m}_0; \mathsf{vis}(\mathbf{Jump?}(\mathbf{r}, \mathbf{m}_0)); \mathsf{assume}(\mathbf{r}(\mathbf{pc}) = a_{\mathsf{main}} \land \mathbf{has\_stack}(\mathbf{r}(\mathbf{sp}), \mathbf{m}_0));\\ & \exists \mathbf{m}_1; \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, \mathbf{1}, \mathbf{m}_1)); \exists \mathbf{m}_2; \mathsf{vis}(\mathbf{SyscallRet?}(*, \mathbf{m}_2)); \mathsf{assume}(\mathbf{m}_2 = \mathbf{m}_1);\\ & \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, \mathbf{2}, *)); \mathsf{vis}(\mathbf{SyscallRet?}(*, *)); \mathsf{any}\end{aligned}$$

Specification

$$\mathrm{Spec}(E) \ni p ::=_{\mathrm{coind}} \mathsf{any} \mid \mathsf{vis}(e); p \mid \mathsf{assume}(\phi); p \mid \exists x : T; p(x) \mid \cdots$$

≈ "Assuming the environment calls the main function, …"

$$\begin{aligned}\mathbf{onetwo}_{\mathsf{spec}} \triangleq\ & \exists \mathbf{r}, \mathbf{m}_0; \mathsf{vis}(\mathbf{Jump?}(\mathbf{r}, \mathbf{m}_0)); \mathsf{assume}(\mathbf{r}(\mathbf{pc}) = a_{\mathsf{main}} \wedge \mathbf{has\_stack}(\mathbf{r}(\mathbf{sp}), \mathbf{m}_0));\\ & \exists \mathbf{m}_1; \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, 1, \mathbf{m}_1)); \exists \mathbf{m}_2; \mathsf{vis}(\mathbf{SyscallRet?}(*, \mathbf{m}_2)); \mathsf{assume}(\mathbf{m}_2 = \mathbf{m}_1);\\ & \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, 2, *)); \mathsf{vis}(\mathbf{SyscallRet?}(*, *)); \mathsf{any}\end{aligned}$$

≈ "… the program prints 1 and then 2."

Assembly verification $\textsc{print-correct}\ [\![\mathbf{print}]\!]_{\mathsf{a}} \preceq [\![\mathbf{print}_{\mathsf{spec}}]\!]_{\mathsf{s}}$

Library print

$$\mathsf{print} : \mathsf{mov}\ \mathbf{x8},\ \ \mathbf{PRINT}; \mathsf{syscall}; \mathsf{ret}$$

$$\preceq$$

$$\begin{aligned}\mathbf{print}_{\mathsf{spec}} \triangleq_{\mathsf{coind}}\ & \exists \mathbf{r}, \mathbf{m}; \mathsf{vis}(\mathbf{Jump?}(\mathbf{r}, \mathbf{m})); \mathsf{assume}(\mathbf{r}(\mathbf{pc}) = a_{\mathsf{print}}); \\ & \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, \mathbf{r}(\mathbf{x0}), \mathbf{m})); \exists \mathbf{v}, \mathbf{m}'; \mathsf{vis}(\mathbf{SyscallRet?}(\mathbf{v}, \mathbf{m}')); \\ & \mathsf{vis}(\mathbf{Jump!}(\mathbf{r}[\mathbf{pc} \mapsto \mathbf{r}(\mathbf{x30})][\mathbf{x0} \mapsto \mathbf{v}][\mathbf{x8} \mapsto *], \mathbf{m}')); \mathbf{print}_{\mathsf{spec}}\end{aligned}$$

Assembly verification

PRINT-CORRECT $[\![\mathbf{print}]\!]_{\mathsf{a}} \preceq [\![\mathbf{print}_{\mathsf{spec}}]\!]_{\mathsf{s}}$

Library print

$\mathsf{print} : \mathsf{mov}\ \mathbf{x8},\ \mathbf{PRINT}; \mathsf{syscall}; \mathsf{ret}$

$\preceq$

When the environment jumps to print, …

$$\begin{aligned}\mathbf{print}_{\mathsf{spec}} \triangleq_{\mathsf{coind}}\ & \exists \mathbf{r}, \mathbf{m}; \mathsf{vis}(\mathbf{Jump?}(\mathbf{r}, \mathbf{m})); \mathsf{assume}(\mathbf{r}(\mathbf{pc}) = a_{\mathsf{print}}); \\ & \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, \mathbf{r}(\mathbf{x0}), \mathbf{m})); \exists \mathbf{v}, \mathbf{m}'; \mathsf{vis}(\mathbf{SyscallRet?}(\mathbf{v}, \mathbf{m}')); \\ & \mathsf{vis}(\mathbf{Jump!}(\mathbf{r}[\mathbf{pc} \mapsto \mathbf{r}(\mathbf{x30})][\mathbf{x0} \mapsto \mathbf{v}][\mathbf{x8} \mapsto *], \mathbf{m}')); \mathbf{print}_{\mathsf{spec}}\end{aligned}$$

Assembly verification

$\text{PRINT-CORRECT} \; [\![\mathbf{print}]\!]_a \precsim [\![\mathbf{print}_{\mathrm{spec}}]\!]_s$

Library print print : mov x8, PRINT; syscall; ret

$\precsim$

When the environment jumps to print, …

$$\begin{aligned}\mathbf{print}_{\mathrm{spec}} \triangleq_{\mathrm{coind}}\; & \exists \mathbf{r}, \mathbf{m};\, \mathsf{vis}(\mathbf{Jump?}(\mathbf{r}, \mathbf{m}));\, \mathsf{assume}(\mathbf{r}(\mathbf{pc}) = a_{\mathrm{print}});\\ & \mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, \mathbf{r}(\mathbf{x0}), \mathbf{m}));\, \exists \mathbf{v}, \mathbf{m}';\, \mathsf{vis}(\mathbf{SyscallRet?}(\mathbf{v}, \mathbf{m}'));\\ & \mathsf{vis}(\mathbf{Jump!}(\mathbf{r}[\mathbf{pc} \mapsto \mathbf{r}(\mathbf{x30})][\mathbf{x0} \mapsto \mathbf{v}][\mathbf{x8} \mapsto *], \mathbf{m}'));\, \mathbf{print}_{\mathrm{spec}}\end{aligned}$$

… invoke PRINT syscall with argument r(x0) …

Assembly verification

PRINT-CORRECT $[\![\mathbf{print}]\!]_{\mathsf{a}} \preceq [\![\mathbf{print}_{\mathsf{spec}}]\!]_{\mathsf{s}}$

Library print

print : mov x8, PRINT; syscall; ret

$\preceq$

When the environment jumps to print, …

$$\mathbf{print}_{\mathsf{spec}} \triangleq_{\mathsf{coind}} \exists \mathbf{r}, \mathbf{m}; \mathsf{vis}(\mathbf{Jump?}(\mathbf{r}, \mathbf{m})); \mathsf{assume}(\mathbf{r}(\mathbf{pc}) = a_{\mathsf{print}});$$
$$\mathsf{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, \mathbf{r}(\mathbf{x0}), \mathbf{m})); \exists \mathbf{v}, \mathbf{m}'; \mathsf{vis}(\mathbf{SyscallRet?}(\mathbf{v}, \mathbf{m}'));$$
$$\mathsf{vis}(\mathbf{Jump!}(\mathbf{r}[\mathbf{pc} \mapsto \mathbf{r}(\mathbf{x30})][\mathbf{x0} \mapsto \mathbf{v}][\mathbf{x8} \mapsto *], \mathbf{m}')); \mathbf{print}_{\mathsf{spec}}$$

… invoke PRINT syscall with argument r(x0) …

…, and jump to return address r(x30).

Assembly verification

PRINT-CORRECT $[\![\mathbf{print}]\!]_a \preceq [\![\mathbf{print}_{\text{spec}}]\!]_s$

register with syscall id

Library **print**

print : mov x8, PRINT; syscall; ret

$\preceq$

When the environment jumps to print, …

$$\mathbf{print}_{\text{spec}} \triangleq_{\text{coind}} \exists \mathbf{r}, \mathbf{m}; \text{vis}(\mathbf{Jump?}(\mathbf{r}, \mathbf{m})); \text{assume}(\mathbf{r}(\mathbf{pc}) = a_{\text{print}});$$
$$\text{vis}(\mathbf{Syscall!}(\mathbf{PRINT}, \mathbf{r}(\mathbf{x0}), \mathbf{m})); \exists \mathbf{v}, \mathbf{m}'; \text{vis}(\mathbf{SyscallRet?}(\mathbf{v}, \mathbf{m}'));$$
$$\text{vis}(\mathbf{Jump!}(\mathbf{r}[\mathbf{pc} \mapsto \mathbf{r}(\mathbf{x30})][\mathbf{x0} \mapsto \mathbf{v}][\mathbf{x8} \mapsto *], \mathbf{m}')); \mathbf{print}_{\text{spec}}$$

… invoke PRINT syscall with argument r(x0) …

…, and jump to return address r(x30).

Assembly verification

$[\![\mathbf{locle}]\!]_{\mathsf{a}} \leq \lceil [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}} \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}}$

Library locle

$\mathsf{locle} : \mathsf{sle}\ \mathbf{x0},\ \mathbf{x0},\ \mathbf{x1};\mathsf{ret}$

Assembly verification

$$[\![\mathbf{locle}]\!]_{a} \preceq \lceil [\![\mathrm{locle}_{spec}]\!]_{s} \rceil_{r \rightleftharpoons a}$$

"set if less or equal"

Library **locle**

$$\mathsf{locle} : \mathsf{sle}\ \mathbf{x0},\ \ \mathbf{x0},\ \ \mathbf{x1};\ \mathsf{ret}$$

Assembly verification

$$[\![\mathbf{locle}]\!]_a \precsim \lceil [\![\mathrm{locle}_{spec}]\!]_s \rceil_{r \rightleftharpoons a}$$

"set if less or equal"

Library locle

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

$$\precsim$$

$$\left\lceil \begin{array}{l} \mathrm{locle}_{spec} \triangleq_{coind} \exists f, \overline{v}, m; \mathrm{vis}(\mathrm{Call?}(f, \overline{v}, m)); \mathrm{assume}(f = \mathrm{locle}); \mathrm{assume}(\overline{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\mathrm{blockid} = \ell_2.\mathrm{blockid} \text{ then } \mathrm{vis}(\mathrm{Return!}(\ell_1.\mathrm{offset} \leq \ell_2.\mathrm{offset}, m)); \mathrm{locle}_{spec} \\ \quad \text{else } \exists b; \mathrm{vis}(\mathrm{Return!}(b, m)); \mathrm{locle}_{spec} \end{array} \right\rceil_{r \rightleftharpoons a}$$

Assembly verification

$[\![\mathbf{locle}]\!]_a \precsim \lceil [\![\text{locle}_{spec}]\!]_s \rceil_{r \rightleftharpoons a}$

"set if less or equal"

Library locle

$\text{locle} : \text{sle } \mathbf{x0},\ \mathbf{x0},\ \mathbf{x1}; \text{ret}$

"When the environment calls locle with ℓ_1 and ℓ_2, ... $\precsim$

$$\left\lceil \begin{array}{l} \text{locle}_{spec} \triangleq_{coind} \exists f, \overline{v}, m; \text{vis}(\text{Call?}(f, \overline{v}, m)); \text{assume}(f = \text{locle}); \text{assume}(\overline{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\text{blockid} = \ell_2.\text{blockid} \text{ then vis}(\text{Return!}(\ell_1.\text{offset} \le \ell_2.\text{offset}, m)); \text{locle}_{spec} \\ \quad \text{else } \exists b; \text{vis}(\text{Return!}(b, m)); \text{locle}_{spec} \end{array} \right\rceil_{r \rightleftharpoons a}$$

Assembly verification

$[\![\mathbf{locle}]\!]_a \precsim \lceil [\![\mathrm{locle}_{spec}]\!]_s \rceil_{r \rightleftharpoons a}$

"set if less or equal"

Library **locle**

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

"When the environment calls locle with ℓ_1 and ℓ_2, … $\precsim$

$$\left\lceil \begin{array}{l} \mathrm{locle_{spec}} \triangleq_{\mathrm{coind}} \exists \mathrm{f}, \overline{\mathrm{v}}, \mathrm{m}; \mathrm{vis}(\mathrm{Call?}(\mathrm{f}, \overline{\mathrm{v}}, \mathrm{m})); \mathrm{assume}(\mathrm{f} = \mathrm{locle}); \mathrm{assume}(\overline{\mathrm{v}} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\mathrm{blockid} = \ell_2.\mathrm{blockid} \text{ then } \mathrm{vis}(\mathrm{Return!}(\ell_1.\mathrm{offset} \leq \ell_2.\mathrm{offset}, \mathrm{m})); \mathrm{locle_{spec}} \\ \quad \text{else } \exists \mathrm{b}; \mathrm{vis}(\mathrm{Return!}(\mathrm{b}, \mathrm{m})); \mathrm{locle_{spec}} \end{array} \right\rceil_{r \rightleftharpoons a}$$

Assembly verification

$[\![\mathbf{locle}]\!]_a \precsim \lceil [\![\mathrm{locle}_{\mathrm{spec}}]\!]_s \rceil_{r \rightleftharpoons a}$

"set if less or equal"

Library locle

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

"When the environment calls locle with ℓ_1 and ℓ_2, … $\precsim$

$$\left\lceil \begin{array}{l} \mathrm{locle}_{\mathrm{spec}} \triangleq_{\mathrm{coind}} \exists f, \overline{v}, m; \mathrm{vis}(\mathrm{Call?}(f, \overline{v}, m)); \mathrm{assume}(f = \mathrm{locle}); \mathrm{assume}(\overline{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\mathrm{blockid} = \ell_2.\mathrm{blockid} \text{ then } \mathrm{vis}(\mathrm{Return!}(\ell_1.\mathrm{offset} \le \ell_2.\mathrm{offset}, m)); \mathrm{locle}_{\mathrm{spec}} \\ \quad \text{else } \exists b; \mathrm{vis}(\mathrm{Return!}(b, m)); \mathrm{locle}_{\mathrm{spec}} \end{array} \right\rceil_{r \rightleftharpoons a}$$

Assembly verification

$$[\![\mathbf{locle}]\!]_{\mathsf{a}} \precsim \lceil [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}} \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}}$$

"set if less or equal"

Library locle

$$\mathsf{locle} : \mathsf{sle}\ \mathbf{x0},\ \mathbf{x0},\ \mathbf{x1}; \mathsf{ret}$$

"When the environment calls locle with ℓ_1 and ℓ_2, … $\precsim$

$$\left\lceil \begin{array}{l} \mathsf{locle}_{\mathsf{spec}} \triangleq_{\mathsf{coind}} \exists \mathsf{f}, \overline{\mathsf{v}}, \mathsf{m}; \mathsf{vis}(\mathsf{Call?}(\mathsf{f}, \overline{\mathsf{v}}, \mathsf{m})); \mathsf{assume}(\mathsf{f} = \mathsf{locle}); \mathsf{assume}(\overline{\mathsf{v}}\ \mathsf{is}\ [\ell_1, \ell_2]); \\ \qquad \mathsf{if}\ \ell_1.\mathsf{blockid} = \ell_2.\mathsf{blockid}\ \mathsf{then}\ \mathsf{vis}(\mathsf{Return!}(\ell_1.\mathsf{offset} \leq \ell_2.\mathsf{offset}, \mathsf{m})); \mathsf{locle}_{\mathsf{spec}} \\ \qquad \mathsf{else}\ \exists \mathsf{b}; \mathsf{vis}(\mathsf{Return!}(\mathsf{b}, \mathsf{m})); \mathsf{locle}_{\mathsf{spec}} \end{array} \right\rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}}$$

Assembly verification

$[\![\mathbf{locle}]\!]_a \precsim \lceil [\![\text{locle}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a}$

"set if less or equal"

Library locle locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

"When the environment calls locle with ℓ_1 and ℓ_2 , … $\precsim$

$$\left\lceil \begin{array}{l} \text{locle}_{\text{spec}} \triangleq_{\text{coind}} \exists f, \overline{v}, m; \text{vis}(\text{Call?}(f, \overline{v}, m)); \text{assume}(f = \text{locle}); \text{assume}(\overline{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\text{blockid} = \ell_2.\text{blockid} \text{ then vis}(\text{Return!}(\ell_1.\text{offset} \leq \ell_2.\text{offset}, m)); \text{locle}_{\text{spec}} \\ \quad \text{else } \exists b; \text{vis}(\text{Return!}(b, m)); \text{locle}_{\text{spec}} \end{array} \right\rceil_{r \rightleftharpoons a}$$

… if the blocks are equal,
compare the offsets, …

Assembly verification

$[\![\text{locle}]\!]_a \precsim \lceil [\![\text{locle}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a}$

"set if less or equal"

Library locle

$\text{locle} : \text{sle } \mathbf{x0},\ \mathbf{x0},\ \mathbf{x1}; \text{ret}$

"When the environment calls locle with ℓ_1 and ℓ_2, … $\precsim$

$$\left\lceil \begin{array}{l} \text{locle}_{\text{spec}} \triangleq_{\text{coind}} \exists f, \overline{v}, m; \text{vis}(\text{Call?}(f, \overline{v}, m)); \text{assume}(f = \text{locle}); \text{assume}(\overline{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\text{blockid} = \ell_2.\text{blockid} \text{ then vis}(\text{Return!}(\ell_1.\text{offset} \leq \ell_2.\text{offset}, m)); \text{locle}_{\text{spec}} \\ \quad \text{else } \exists b; \text{vis}(\text{Return!}(b, m)); \text{locle}_{\text{spec}} \end{array} \right\rceil_{r \rightleftharpoons a}$$

… if the blocks are equal,
compare the offsets, …

Assembly verification

$$[\![\mathbf{locle}]\!]_a \precsim \lceil [\![\text{locle}_{spec}]\!]_s \rceil_{r \rightleftharpoons a}$$

"set if less or equal"

Library locle locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

"When the environment calls locle with ℓ_1 and ℓ_2, … $\precsim$

$$\left\lceil \begin{array}{l} \text{locle}_{spec} \triangleq_{coind} \exists f, \overline{v}, m; \text{vis}(\text{Call?}(f, \overline{v}, m)); \text{assume}(f = \text{locle}); \text{assume}(\overline{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\text{blockid} = \ell_2.\text{blockid} \text{ then vis}(\text{Return!}(\ell_1.\text{offset} \leq \ell_2.\text{offset}, m)); \text{locle}_{spec} \\ \quad \text{else } \exists b; \text{vis}(\text{Return!}(b, m)); \text{locle}_{spec} \end{array} \right\rceil_{r \rightleftharpoons a}$$

… if the blocks are equal,

compare the offsets, …

Assembly verification

$[\![\mathbf{locle}]\!]_a \precsim \lceil [\![\mathrm{locle}_{spec}]\!]_s \rceil_{r \rightleftharpoons a}$

"set if less or equal"

Library locle

$$\text{locle} : \text{sle } \mathbf{x0},\ \mathbf{x0},\ \mathbf{x1}; \text{ret}$$

"When the environment calls locle with ℓ_1 and ℓ_2, … $\precsim$

$$\left\lceil \begin{array}{l} \mathrm{locle}_{spec} \triangleq_{coind} \exists f, \bar{v}, m; \mathrm{vis}(\mathrm{Call?}(f, \bar{v}, m)); \mathrm{assume}(f = \mathrm{locle}); \mathrm{assume}(\bar{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\mathrm{blockid} = \ell_2.\mathrm{blockid} \text{ then } \mathrm{vis}(\mathrm{Return!}(\ell_1.\mathrm{offset} \leq \ell_2.\mathrm{offset}, m)); \mathrm{locle}_{spec} \\ \quad \text{else } \exists b; \mathrm{vis}(\mathrm{Return!}(b, m)); \mathrm{locle}_{spec} \end{array} \right\rceil_{r \rightleftharpoons a}$$

… if the blocks are equal,

compare the offsets, …

… otherwise return a

non-deterministic Boolean."

Assembly verification

$[\![\mathbf{locle}]\!]_{a} \precsim \lceil [\![\mathrm{locle}_{spec}]\!]_{s} \rceil_{r \rightleftharpoons a}$

"set if less or equal"

Library $\mathbf{locle}$

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

"When the environment calls locle with ℓ_1 and ℓ_2, … $\precsim$

$$\left\lceil \begin{array}{l} \mathrm{locle}_{spec} \triangleq_{coind} \exists f, \overline{v}, m; \mathrm{vis}(\mathrm{Call?}(f, \overline{v}, m)); \mathrm{assume}(f = \mathrm{locle}); \mathrm{assume}(\overline{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\mathrm{blockid} = \ell_2.\mathrm{blockid} \text{ then } \mathrm{vis}(\mathrm{Return!}(\ell_1.\mathrm{offset} \leq \ell_2.\mathrm{offset}, m)); \mathrm{locle}_{spec} \\ \quad \text{else } \exists b; \mathrm{vis}(\mathrm{Return!}(b, m)); \mathrm{locle}_{spec} \end{array} \right\rceil_{r \rightleftharpoons a}$$

… if the blocks are equal, compare the offsets, …

… otherwise return a non-deterministic Boolean."

Assembly verification

$$[\![\mathbf{locle}]\!]_a \precsim \lceil [\![\mathrm{locle_{spec}}]\!]_s \rceil_{r \rightleftarrows a}$$

"set if less or equal"

Library locle	
	locle : sle x0, x0, x1; ret

"When the environment calls locle with ℓ_1 and ℓ_2, … $\precsim$

$$\left\lceil \begin{array}{l} \mathrm{locle_{spec}} \triangleq_{\mathrm{coind}} \exists f, \overline{v}, m; \mathrm{vis}(\mathrm{Call?}(f, \overline{v}, m)); \mathrm{assume}(f = \mathrm{locle}); \mathrm{assume}(\overline{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\mathrm{blockid} = \ell_2.\mathrm{blockid} \text{ then } \mathrm{vis}(\mathrm{Return!}(\ell_1.\mathrm{offset} \le \ell_2.\mathrm{offset}, m)); \mathrm{locle_{spec}} \\ \quad \text{else } \exists b; \mathrm{vis}(\mathrm{Return!}(b, m)); \mathrm{locle_{spec}} \end{array} \right\rceil_{r \rightleftarrows a}$$

… if the blocks are equal, compare the offsets, …

… otherwise return a non-deterministic Boolean."

Assembly verification

$[\![\mathbf{locle}]\!]_a \lesssim \lceil [\![\text{locle}_{spec}]\!]_s \rceil_{r \rightleftharpoons a}$

"set if less or equal"

Library locle

locle : sle $\mathbf{x0}$, $\mathbf{x0}$, $\mathbf{x1}$; ret

"When the environment calls locle with ℓ_1 and ℓ_2, … $\lesssim$

$$\left\lceil \begin{array}{l} \text{locle}_{spec} \triangleq_{coind} \exists f, \bar{v}, m; \text{vis}(\text{Call?}(f, \bar{v}, m)); \text{assume}(f = \text{locle}); \text{assume}(\bar{v} \text{ is } [\ell_1, \ell_2]); \\ \quad \text{if } \ell_1.\text{blockid} = \ell_2.\text{blockid} \text{ then vis}(\text{Return!}(\ell_1.\text{offset} \le \ell_2.\text{offset}, m)); \text{locle}_{spec} \\ \quad \text{else } \exists b; \text{vis}(\text{Return!}(b, m)); \text{locle}_{spec} \end{array} \right\rceil_{r \rightleftharpoons a}$$

… if the blocks are equal,

compare the offsets, …

… otherwise return a non-deterministic Boolean."

No syntactic **Rec** program required!

$\lceil\cdot\rceil_{r\rightleftharpoons a}$ wrapper

$$[\![\mathrm{memmove}]\!]_r \;\underset{\mathrm{Return?}(v', m')}{\overset{\mathrm{Call!}(\mathrm{locle}, [d, s], m)}{\rightleftarrows}}\; \lceil\cdot\rceil_{r\rightleftharpoons a} \;\underset{\mathbf{Jump?}(\mathbf{r}', \mathbf{m}')}{\overset{\mathbf{Jump!}(\mathbf{r}, \mathbf{m})}{\rightleftarrows}}\; \oplus_a [\![\mathbf{locle}]\!]_a$$

$\lceil\cdot\rceil_{\mathsf{r}\rightleftharpoons\mathsf{a}}$ wrapper

$$\lceil\cdot\rceil_{\mathsf{r}\rightleftharpoons\mathsf{a}}$$

$$[\![\mathrm{memmove}]\!]_{\mathsf{r}} \;\underset{\mathrm{Return?}(v', m')}{\overset{\mathrm{Call!}(\mathrm{locle}, [d, s], m)}{\rightleftarrows}}\; \Big|\; \underset{\mathbf{Jump?}(\mathbf{r}', \mathbf{m}')}{\overset{\mathbf{Jump!}(\mathbf{r}, \mathbf{m})}{\rightleftarrows}} \;\oplus_{\mathsf{a}}\; [\![\mathbf{locle}]\!]_{\mathsf{a}}$$

$\mathsf{fn}\ \mathsf{memmove}(d, s, n) \triangleq$

$\quad \mathsf{if}\ \mathsf{locle}(d, s)\ \mathsf{then}\ \mathsf{memcpy}(d, s, n, 1)\ \mathsf{else}\ \mathsf{memcpy}(d+n-1, s+n-1, n, -1)$

$$\ulcorner \cdot \urcorner_{r \rightleftharpoons a} : [\![\mathrm{memmove}]\!]_r \underset{\mathrm{Return?}(v', m')}{\overset{\mathrm{Call!}(\mathrm{locle}, [d, s], m)}{\rightleftarrows}} \underset{\mathrm{Jump?}(r', m')}{\overset{\mathbf{Jump!}(\mathbf{r}, \mathbf{m})}{\rightleftarrows}} \oplus_a [\![\mathbf{locle}]\!]_a$$

Translating values:

$$z \sim_w z \qquad b \sim_w (\text{if } b \text{ then } 1 \text{ else } 0) \qquad \ell \sim_w w(\ell.\mathrm{blockid}) + \ell.\mathrm{offset}$$

$$\ulcorner\cdot\urcorner_{r \rightleftarrows a} : [\![\text{memmove}]\!]_r \xrightarrow[\text{Return?}(v', m')]{\text{Call!}(\text{locle}, [d, s], m)} \;\Big|\; \xrightarrow[\text{Jump?}(r', m')]{\text{Jump!}(r, m)} \oplus_a [\![\text{locle}]\!]_a$$

Translating values:

$$z \sim_w z \qquad b \sim_w (\text{if } b \text{ then } 1 \text{ else } 0) \qquad \ell \sim_w w(\ell.\text{blockid}) + \ell.\text{offset}$$

Kripke world: map from block ids to base addresses

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\text{memmove}]\!]_r \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \Big| \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \oplus_a [\![\mathbf{locle}]\!]_a$$

Translating values:

$$z \sim_w z \qquad b \sim_w (\text{if } b \text{ then } 1 \text{ else } 0) \qquad \ell \sim_w w(\ell.\text{blockid}) + \ell.\text{offset}$$

Kripke world: map from block ids to base addresses

Translating events:

$$\text{Call!}(f, \overline{v}, m) \rightharpoonup_w \mathbf{Jump!}(\mathbf{r}, \mathbf{m}) \triangleq \mathbf{r}(\mathbf{pc}) = a_f \land \overline{v} \sim_w \mathbf{r}(\mathbf{x0} \ldots \mathbf{x8}) \land m \sim_w \mathbf{m}$$

$$\lceil \cdot \rceil_{r \rightleftarrows a} : [\![\mathrm{memmove}]\!]_r \overset{\mathrm{Call!}(\mathrm{locle}, [d, s], m)}{\underset{\mathrm{Return?}(v', m')}{\rightleftarrows}} \;\Big|\; \overset{\mathbf{Jump!}(\mathbf{r}, \mathbf{m})}{\underset{\mathbf{Jump?}(\mathbf{r}', \mathbf{m}')}{\rightleftarrows}} \oplus_a [\![\mathbf{locle}]\!]_a$$

Translating values:

$$z \sim_w z \qquad b \sim_w (\text{if } b \text{ then } 1 \text{ else } 0) \qquad \ell \sim_w w(\ell.\mathrm{blockid}) + \ell.\mathrm{offset}$$

Kripke world: map from block ids to base addresses

Translating events:

$$\mathrm{Call!}(f, \overline{v}, m) \rightharpoonup_w \mathbf{Jump!}(\mathbf{r}, \mathbf{m}) \triangleq \mathbf{r}(\mathbf{pc}) = a_f \land \overline{v} \sim_w \mathbf{r}(\mathbf{x0} \ldots \mathbf{x8}) \land m \sim_w \mathbf{m}$$

PC contains address of the function

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\text{memmove}]\!]_r \xrightarrow{\text{Call!(locle, [d, s], m)}} \xrightarrow{\text{Jump!(r, m)}} \oplus_a [\![\text{locle}]\!]_a$$

$$\xleftarrow[\text{Return?(v', m')}]{} \xleftarrow[\text{Jump?(r', m')}]{}$$

Translating values:

$$z \sim_w z \qquad b \sim_w (\text{if } b \text{ then } 1 \text{ else } 0) \qquad \ell \sim_w w(\ell.\text{blockid}) + \ell.\text{offset}$$

Kripke world: map from block ids to base addresses

Translating events:

$$\text{Call!}(f, \overline{v}, m) \rightharpoonup_w \text{Jump!}(r, m) \triangleq r(pc) = a_f \wedge \overline{v} \sim_w r(x0 \ldots x8) \wedge m \sim_w m$$

PC contains address of the function

Rec arguments are related to Asm argument registers

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\mathrm{memmove}]\!]_r \;\; \underset{\mathrm{Return?}(v', m')}{\overset{\mathrm{Call!}(\mathrm{locle}, [d, s], m)}{\rightleftarrows}} \;\Big|\; \underset{\mathrm{Jump?}(r', m')}{\overset{\mathbf{Jump!}(\mathbf{r}, \mathbf{m})}{\rightleftarrows}} \;\; \oplus_a [\![\mathbf{locle}]\!]_a$$

Translating values:

$$z \sim_w z \qquad b \sim_w (\text{if } b \text{ then } 1 \text{ else } 0) \qquad \ell \sim_w w(\ell.\mathrm{blockid}) + \ell.\mathrm{offset}$$

Kripke world: map from block ids to base addresses

Translating events:

$$\mathrm{Call!}(\mathrm{f}, \overline{\mathrm{v}}, \mathrm{m}) \rightharpoonup_w \mathbf{Jump!}(\mathbf{r}, \mathbf{m}) \triangleq \mathbf{r}(\mathbf{pc}) = a_{\mathrm{f}} \land \overline{\mathrm{v}} \sim_w \mathbf{r}(\mathbf{x0} \ldots \mathbf{x8}) \land \mathrm{m} \sim_w \mathbf{m}$$

PC contains address of the function

Rec arguments are related to Asm argument registers

memories are related

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\text{memmove}]\!]_r \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \Big| \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \oplus_a [\![\textbf{locle}]\!]_a$$

$$\begin{aligned}
[\![\textbf{onetwo}]\!]_a &= [\![\downarrow \text{main} \cup_a \downarrow \text{memmove} \cup_a \textbf{locle} \cup_a \textbf{print}]\!]_a \\
&\le [\![\downarrow \text{main}]\!]_a \oplus_a [\![\downarrow \text{memmove}]\!]_a \oplus_a [\![\textbf{locle}]\!]_a \oplus_a [\![\textbf{print}]\!]_a \\
&\le \lceil [\![\text{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\textbf{locle}]\!]_a \oplus_a [\![\textbf{print}]\!]_a \\
&\le \lceil [\![\text{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\text{locle}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\textbf{print}_{\text{spec}}]\!]_s \\
&\le \lceil [\![\text{main}]\!]_r \oplus_r [\![\text{memmove}]\!]_r \oplus_r [\![\text{locle}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\textbf{print}_{\text{spec}}]\!]_s \\
&\le \lceil [\![\text{main} \cup_r \text{memmove}]\!]_r \oplus_r [\![\text{locle}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\textbf{print}_{\text{spec}}]\!]_s \\
&\le \lceil [\![\text{main}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\textbf{print}_{\text{spec}}]\!]_s \\
&\le [\![\textbf{onetwo}_{\text{spec}}]\!]_s
\end{aligned}$$

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\text{memmove}]\!]_r \; \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \;\Big|\; \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \quad \oplus_a [\![\text{locle}]\!]_a$$

In $[\![\downarrow \text{memmove}]\!]_a \leq \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a}$, consider locle returning 0:

$$[\![\downarrow \text{memmove}]\!]_a \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\text{memmove}]\!]_r \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \Big| \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \oplus_a [\![\text{locle}]\!]_a$$

In $[\![\downarrow \text{memmove}]\!]_a \preceq \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a}$, consider locle returning 0:

$$[\![\downarrow \text{memmove}]\!]_a \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

$$\preceq$$

$$[\![\text{memmove}]\!]_r \xleftarrow{\text{Return?}(?, m)} \Big| \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

$$\lceil\cdot\rceil_{r\rightleftharpoons a} : [\![\mathrm{memmove}]\!]_r \underset{\mathrm{Return?}(v', m')}{\overset{\mathrm{Call!}(\mathrm{locle}, [d, s], m)}{\rightleftarrows}} \Big| \underset{\mathrm{Jump?}(r', m')}{\overset{\mathrm{Jump!}(r, m)}{\rightleftarrows}} \oplus_a [\![\mathrm{locle}]\!]_a$$

In $[\![\downarrow\mathrm{memmove}]\!]_a \precsim \lceil[\![\mathrm{memmove}]\!]_r\rceil_{r\rightleftharpoons a}$, consider locle returning 0:

$$[\![\downarrow\mathrm{memmove}]\!]_a \xleftarrow{\mathrm{Jump?}(r(x0) = 0, \ldots, m)}$$

$$\precsim$$

$$[\![\mathrm{memmove}]\!]_r \xleftarrow{\mathrm{Return?}(?, m)} \Big| \xleftarrow{\mathrm{Jump?}(r(x0) = 0, \ldots, m)}$$

Which value should $\lceil\cdot\rceil_{r\rightleftharpoons a}$ pick?

$\mathrm{false} \sim_w 0$

$0 \sim_w 0$

$\ell_0 \sim_w 0$

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\text{memmove}]\!]_r \;\; \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \;\Big\vert\; \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \;\; \oplus_a [\![\text{locle}]\!]_a$$

In $[\![\downarrow \text{memmove}]\!]_a \precsim \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a}$, consider locle returning 0:

$$[\![\downarrow \text{memmove}]\!]_a \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

$$\precsim$$

$$[\![\text{memmove}]\!]_r \xleftarrow{\text{Return?}(?, m)} \Big\vert \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

$$\text{false} \sim_w 0$$

$$0 \sim_w 0$$

$$\ell_0 \sim_w 0$$

Which value should $\lceil \cdot \rceil_{r \rightleftharpoons a}$ pick?

Choose non-deterministically!

$$\lceil \cdot \rceil_{r \rightleftarrows a} : [\![\text{memmove}]\!]_r \; \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \;\Big|\; \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \; \oplus_a [\![\text{locle}]\!]_a$$

$$[\![\text{memmove}]\!]_r \xleftarrow{\text{Return?}(?, m)} \;\Big|\; \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

Demonic non-determinism

"$\exists v.\ v \sim_w r(x0) \wedge \ldots$"

$\text{false} \sim_w 0 \quad 0 \sim_w 0 \quad \ell_0 \sim_w 0$

$$\lceil \cdot \rceil_{r \rightleftarrows a} : [\![\text{memmove}]\!]_r \;\underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}}\;\Big|\; \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \;\oplus_a [\![\text{locle}]\!]_a$$

$$[\![\text{memmove}]\!]_r \xleftarrow{\text{Return?}(?, m)} \Big| \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

Demonic non-determinism

"$\exists v.\ v \sim_w r(x0) \wedge \ldots$"

$\text{false} \sim_w 0 \quad 0 \sim_w 0 \quad \ell_0 \sim_w 0$

?

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\text{memmove}]\!]_r \; \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \;\Big|\; \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \; \oplus_a [\![\textbf{locle}]\!]_a$$

$$[\![\text{memmove}]\!]_r \xleftarrow{\text{Return?}(?, m)} \Big| \xleftarrow{\textbf{Jump?}(\textbf{r}(\textbf{x0}) = \textbf{0}, \ldots, \textbf{m})}$$

Demonic non-determinism

"$\exists v.\ v \sim_w \textbf{r}(\textbf{x0}) \wedge \ldots$"

$\text{false} \sim_w \textbf{0} \quad 0 \sim_w \textbf{0} \quad \ell_0 \sim_w \textbf{0}$

?

$$\lceil \cdot \rceil_{r \rightleftarrows a} : [\![\mathrm{memmove}]\!]_r \; \underset{\mathrm{Return?}(v', m')}{\overset{\mathrm{Call!}(\mathrm{locle}, [d, s], m)}{\rightleftarrows}} \;\Big|\; \underset{\mathrm{Jump?}(\mathbf{r}', \mathbf{m}')}{\overset{\mathrm{Jump!}(\mathbf{r}, \mathbf{m})}{\rightleftarrows}} \; \oplus_a [\![\mathbf{locle}]\!]_a$$

$$[\![\mathrm{memmove}]\!]_r \; \xleftarrow{\mathrm{Return?}(?, m)} \;\Big|\; \xleftarrow{\mathrm{Jump?}(\mathbf{r}(\mathbf{x0}) = \mathbf{0}, \ldots, \mathbf{m})}$$

Demonic non-determinism

"$\exists v.\ v \sim_w \mathbf{r}(\mathbf{x0}) \wedge \ldots$"

$\mathrm{false} \sim_w \mathbf{0} \quad 0 \sim_w \mathbf{0} \quad \ell_0 \sim_w \mathbf{0}$

?

Angelic non-determinism

"$\forall v.\ v \sim_w \mathbf{r}(\mathbf{x0}) \Rightarrow \ldots$"

$\mathrm{false} \sim_w \mathbf{0} \quad 0 \sim_w \mathbf{0} \quad \ell_0 \sim_w \mathbf{0}$

$$\lceil \cdot \rceil_{r \rightleftarrows a} : [\![\text{memmove}]\!]_r \; \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \;\Big\vert\; \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \; \oplus_a [\![\text{locle}]\!]_a$$

$$[\![\text{memmove}]\!]_r \xleftarrow{\text{Return?}(?, m)} \;\Big\vert\; \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

Demonic non-determinism

Angelic non-determinism

"$\exists v.\ v \sim_w r(x0) \land \ldots$"

"$\forall v.\ v \sim_w r(x0) \Rightarrow \ldots$"

$\text{false} \sim_w 0 \quad 0 \sim_w 0 \quad \ell_0 \sim_w 0$

?

$\text{false} \sim_w 0 \quad 0 \sim_w 0 \quad \ell_0 \sim_w 0$

$$\ulcorner \cdot \urcorner_{r \rightleftarrows a} : [\![\text{memmove}]\!]_r \underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}} \;\vdots\; \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \quad \oplus_a [\![\text{locle}]\!]_a$$

$$[\![\text{memmove}]\!]_r \xleftarrow{\text{Return?}(?, m)} \;\vdots\; \xleftarrow{\text{Jump?}(r(x0) = 0, \ldots, m)}$$

Demonic non-determinism

“$\exists v.\ v \sim_w r(x0) \wedge \ldots$”

$\text{false} \sim_w 0 \quad 0 \sim_w 0 \quad \ell_0 \sim_w 0$

?

Angelic non-determinism

“$\forall v.\ v \sim_w r(x0) \Rightarrow \ldots$”

$\text{false} \sim_w 0 \quad 0 \sim_w 0 \quad \ell_0 \sim_w 0$

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\text{memmove}]\!]_r \;\underset{\text{Return?}(v', m')}{\overset{\text{Call!}(\text{locle}, [d, s], m)}{\rightleftarrows}}\; \underset{\text{Jump?}(r', m')}{\overset{\text{Jump!}(r, m)}{\rightleftarrows}} \;\oplus_a [\![\text{locle}]\!]_a$$

$$\begin{aligned}
[\![\text{onetwo}]\!]_a &= [\![{\downarrow}\text{main} \cup_a {\downarrow}\text{memmove} \cup_a \text{locle} \cup_a \text{print}]\!]_a \\
&\leq [\![{\downarrow}\text{main}]\!]_a \oplus_a [\![{\downarrow}\text{memmove}]\!]_a \oplus_a [\![\text{locle}]\!]_a \oplus_a [\![\text{print}]\!]_a \\
&\leq \lceil [\![\text{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\text{locle}]\!]_a \oplus_a [\![\text{print}]\!]_a \\
&\leq \lceil [\![\text{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\text{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\text{locle}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\text{print}_{\text{spec}}]\!]_s \\
&\leq \lceil [\![\text{main}]\!]_r \oplus_r [\![\text{memmove}]\!]_r \oplus_r [\![\text{locle}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\text{print}_{\text{spec}}]\!]_s \\
&\leq \lceil [\![\text{main} \cup_r \text{memmove}]\!]_r \oplus_r [\![\text{locle}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\text{print}_{\text{spec}}]\!]_s \\
&\leq \lceil [\![\text{main}_{\text{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\text{print}_{\text{spec}}]\!]_s \\
&\leq [\![\text{onetwo}_{\text{spec}}]\!]_s
\end{aligned}$$

$$\lceil \cdot \rceil_{r \rightleftharpoons a} : [\![\mathsf{memmove}]\!]_r \;\; \underset{\mathrm{Return?}(v', m')}{\overset{\mathrm{Call!}(\mathrm{locle}, [d, s], m)}{\rightleftarrows}} \;\; \underset{\mathbf{Jump?}(\mathbf{r}', \mathbf{m}')}{\overset{\mathbf{Jump!}(\mathbf{r}, \mathbf{m})}{\rightleftarrows}} \;\; \oplus_a [\![\mathbf{locle}]\!]_a$$

$$\begin{aligned}
[\![\mathbf{onetwo}]\!]_a &= [\![\downarrow \mathsf{main} \cup_a \downarrow \mathsf{memmove} \cup_a \mathbf{locle} \cup_a \mathbf{print}]\!]_a \\
&\leq [\![\downarrow \mathsf{main}]\!]_a \oplus_a [\![\downarrow \mathsf{memmove}]\!]_a \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{locle}]\!]_a \oplus_a [\![\mathbf{print}]\!]_a \\
&\leq \lceil [\![\mathsf{main}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{locle}_{\mathsf{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{print}_{\mathsf{spec}}]\!]_s \\
&\leq \lceil [\![\mathsf{main}]\!]_r \oplus_r [\![\mathsf{memmove}]\!]_r \oplus_r [\![\mathsf{locle}_{\mathsf{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{print}_{\mathsf{spec}}]\!]_s \\
&\leq \lceil [\![\mathsf{main} \cup_r \mathsf{memmove}]\!]_r \oplus_r [\![\mathsf{locle}_{\mathsf{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{print}_{\mathsf{spec}}]\!]_s \\
&\leq \lceil [\![\mathsf{main}_{\mathsf{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \oplus_a [\![\mathbf{print}_{\mathsf{spec}}]\!]_s \\
&\leq [\![\mathbf{onetwo}_{\mathsf{spec}}]\!]_s
\end{aligned}$$

Resolving the angelic choice

Consider $\lceil [\![\mathrm{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathrm{locle}_{\mathsf{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \leq \lceil [\![\mathrm{memmove}]\!]_r \oplus_r [\![\mathrm{locle}_{\mathsf{spec}}]\!]_s \rceil_{r \rightleftharpoons a}$

Resolving the angelic choice

Consider $\lceil [\![\mathsf{memmove}]\!]_{\mathsf{r}} \rceil_{\mathsf{r} \rightleftarrows \mathsf{a}} \oplus_{\mathsf{a}} \lceil [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}} \rceil_{\mathsf{r} \rightleftarrows \mathsf{a}} \preceq \lceil [\![\mathsf{memmove}]\!]_{\mathsf{r}} \oplus_{\mathsf{r}} [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}} \rceil_{\mathsf{r} \rightleftarrows \mathsf{a}}$

false , 0, or ℓ_0 ?

$$[\![\mathsf{memmove}]\!]_{\mathsf{r}} \xleftarrow{\mathrm{Return?}(?,\, \mathrm{m})} \;\vdots\; \xleftarrow{\mathbf{Jump?}(\mathbf{r}(\mathbf{x0}) = \mathbf{0}, \ldots, \mathbf{m})} \;\vdots\; \xleftarrow{\mathrm{Return?}(\mathrm{false},\, \mathrm{m})} [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}}$$

Resolving the angelic choice

Consider $\lceil [\![\mathsf{memmove}]\!]_r \rceil_{r \rightleftharpoons a} \oplus_a \lceil [\![\mathsf{locle}_{\mathsf{spec}}]\!]_s \rceil_{r \rightleftharpoons a} \preceq \lceil [\![\mathsf{memmove}]\!]_r \oplus_r [\![\mathsf{locle}_{\mathsf{spec}}]\!]_s \rceil_{r \rightleftharpoons a}$

false , 0, or ℓ_0 ?

$$[\![\mathsf{memmove}]\!]_r \xleftarrow{\mathrm{Return?}(?,\, m)} \;\vdots\; \xleftarrow{\mathbf{Jump?}(\mathbf{r}(\mathbf{x0}) = \mathbf{0}, \ldots, \mathbf{m})} \;\vdots\; \xleftarrow{\mathrm{Return?}(\mathrm{false},\, m)} [\![\mathsf{locle}_{\mathsf{spec}}]\!]_s$$

$$\preceq$$

$$[\![\mathsf{memmove}]\!]_r \xleftarrow{\mathrm{Return?}(\mathrm{false},\, m)} [\![\mathsf{locle}_{\mathsf{spec}}]\!]_s$$

Resolving the angelic choice

Consider $\lceil [\![\mathsf{memmove}]\!]_{\mathsf{r}} \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}} \oplus_{\mathsf{a}} \lceil [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}} \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}} \preceq \lceil [\![\mathsf{memmove}]\!]_{\mathsf{r}} \oplus_{\mathsf{r}} [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}} \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}}$

false , 0, or ℓ_0 ?

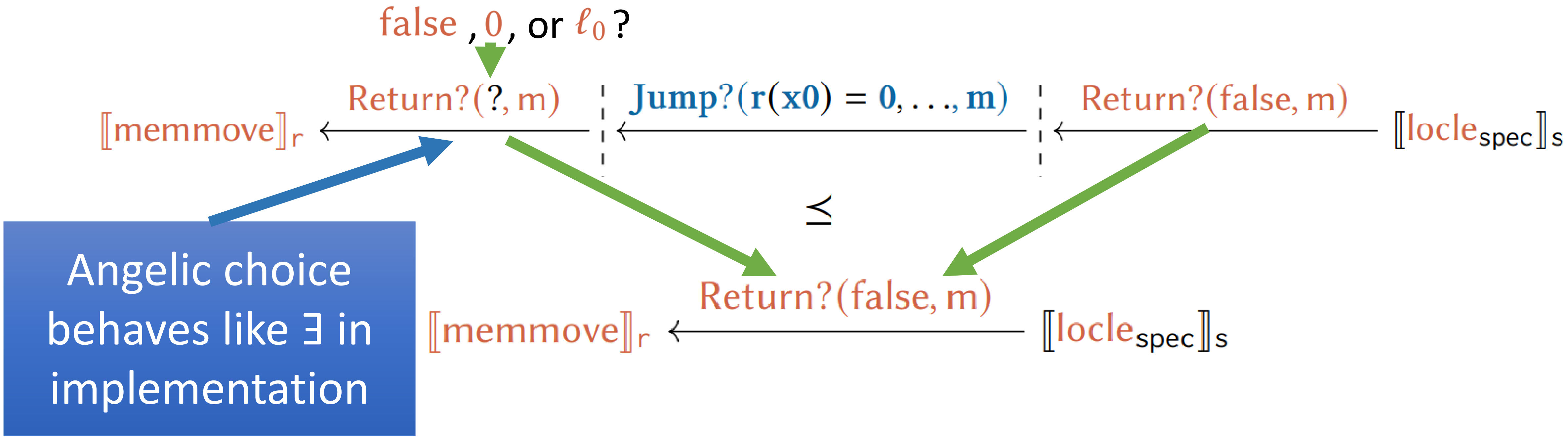
$$[\![\mathsf{memmove}]\!]_{\mathsf{r}} \xleftarrow{\mathsf{Return?}(?,\, \mathsf{m})} \;\vdots\; \xleftarrow{\mathbf{Jump?}(\mathbf{r}(\mathbf{x0}) = \mathbf{0}, \ldots, \mathbf{m})} \;\vdots\; \xleftarrow{\mathsf{Return?}(\mathsf{false},\, \mathsf{m})} [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}}$$

$$\preceq$$

$$[\![\mathsf{memmove}]\!]_{\mathsf{r}} \xleftarrow{\mathsf{Return?}(\mathsf{false},\, \mathsf{m})} [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}}$$

Resolving the angelic choice

Consider $\lceil [\![\mathsf{memmove}]\!]_{\mathsf{r}} \rceil_{\mathsf{r}\rightleftharpoons\mathsf{a}} \oplus_{\mathsf{a}} \lceil [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}} \rceil_{\mathsf{r}\rightleftharpoons\mathsf{a}} \preceq \lceil [\![\mathsf{memmove}]\!]_{\mathsf{r}} \oplus_{\mathsf{r}} [\![\mathsf{locle}_{\mathsf{spec}}]\!]_{\mathsf{s}} \rceil_{\mathsf{r}\rightleftharpoons\mathsf{a}}$



Non-determinism summary

implementation $\preceq$ specification

Angelic *non-determinism*	***Demonic*** *non-determinism*
$\forall$ in specification	$\exists$ in specification
$\exists$ in implementation	$\forall$ in implementation
Assumption about the environment *Rely*	Guarantee to the environment *Guarantee*

Program main	fn $\mathrm{main}() \triangleq$ let $x := \mathrm{yield}(0)$ in $\mathrm{print}(x);$ let $x := \mathrm{yield}(0)$ in $\mathrm{print}(x); \mathrm{yield}(0)$
Library stream	fn $\mathrm{stream}(n) \triangleq \mathrm{yield}(n); \mathrm{stream}(n+1);$
Library yield	yield : ... save and restore registers, and switch stack ...

CORO-LINK

$$[\![\mathbf{yield}]\!]_{\mathsf{a}} \oplus_{\mathsf{a}} \lceil M_1 \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}} \oplus_{\mathsf{a}} \lceil M_2 \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}} \leq \lceil M_1 \oplus_{\mathsf{coro}} M_2 \rceil_{\mathsf{r} \rightleftharpoons \mathsf{a}}$$

Non-determinism in Spec

demonic angelic

$$\mathsf{Spec}(E) \ni p ::=_{\mathsf{coind}} \mathsf{vis}(e); p \mid \exists x : T; p(x) \mid \forall x : T; p(x) \qquad (e \in E)$$

$$\textsc{Sim-Ex-R}\ \frac{\exists y \in T.\ M \leq [\![p(y)]\!]_{\mathsf{s}}}{M \leq [\![\exists x : T; p(x)]\!]_{\mathsf{s}}} \qquad \textsc{Sim-Ex-L}\ \frac{\forall y \in T.\ [\![p(y)]\!]_{\mathsf{s}} \leq M}{[\![\exists x : T; p(x)]\!]_{\mathsf{s}} \leq M}$$

$$\textsc{Sim-All-R}\ \frac{\forall y \in T.\ M \leq [\![p(y)]\!]_{\mathsf{s}}}{M \leq [\![\forall x : T; p(x)]\!]_{\mathsf{s}}} \qquad \textsc{Sim-All-L}\ \frac{\exists y \in T.\ [\![p(y)]\!]_{\mathsf{s}} \leq M}{[\![\forall x : T; p(x)]\!]_{\mathsf{s}} \leq M}$$

Non-determinism in Spec

demonic angelic

$$\mathsf{Spec}(E) \ni p ::=_{\mathsf{coind}} \mathsf{vis}(e); p \mid \exists x : T; p(x) \mid \forall x : T; p(x) \qquad (e \in E)$$

$$\textsc{sim-ex-r}\quad \frac{\exists y \in T.\, M \le [\![p(y)]\!]_{\mathsf{s}}}{M \le [\![\exists x : T; p(x)]\!]_{\mathsf{s}}} \qquad \textsc{sim-ex-l}\quad \frac{\forall y \in T.\, [\![p(y)]\!]_{\mathsf{s}} \le M}{[\![\exists x : T; p(x)]\!]_{\mathsf{s}} \le M}$$

$$\textsc{sim-all-r}\quad \frac{\forall y \in T.\, M \le [\![p(y)]\!]_{\mathsf{s}}}{M \le [\![\forall x : T; p(x)]\!]_{\mathsf{s}}} \qquad \textsc{sim-all-l}\quad \frac{\exists y \in T.\, [\![p(y)]\!]_{\mathsf{s}} \le M}{[\![\forall x : T; p(x)]\!]_{\mathsf{s}} \le M}$$

Non-determinism in Spec

demonic angelic

$$\mathrm{Spec}(E) \ni p ::=_{\mathrm{coind}} \mathsf{vis}(e);p \mid \exists x : T; p(x) \mid \forall x : T; p(x) \qquad (e \in E)$$

demonic:

SIM-EX-R
$$\frac{\exists y \in T.\, M \preceq [\![p(y)]\!]_{\mathsf{s}}}{M \preceq [\![\exists x : T; p(x)]\!]_{\mathsf{s}}}$$

SIM-EX-L
$$\frac{\forall y \in T.\, [\![p(y)]\!]_{\mathsf{s}} \preceq M}{[\![\exists x : T; p(x)]\!]_{\mathsf{s}} \preceq M}$$

angelic:

SIM-ALL-R
$$\frac{\forall y \in T.\, M \preceq [\![p(y)]\!]_{\mathsf{s}}}{M \preceq [\![\forall x : T; p(x)]\!]_{\mathsf{s}}}$$

SIM-ALL-L
$$\frac{\exists y \in T.\, [\![p(y)]\!]_{\mathsf{s}} \preceq M}{[\![\forall x : T; p(x)]\!]_{\mathsf{s}} \preceq M}$$

∀ and ∃ behave like the logical quantifiers

Operational semantics for angelic non-determinism

Definition of modules:

set of states

$$M = (S, \rightarrow, \sigma^0)$$

$$\rightarrow \in \mathcal{P}(S \times \mathsf{option}(E) \times \mathcal{P}(S))$$

$$(\exists x : T; p(x)) \xrightarrow{\tau}_{\mathsf{s}} \{p(y)\} \text{ (for } y \in T) \qquad (\forall x : T; p(x)) \xrightarrow{\tau}_{\mathsf{s}} \{p(y) \mid y \in T\}$$

$$(\mathsf{vis}(e); p) \xrightarrow{e}_{\mathsf{s}} \{p\}$$

Operational semantics for angelic non-determinism

Definition of modules:

set of states   initial state

$$M = (S, \to, \sigma^0)$$

$$\to \in \mathcal{P}(S \times \text{option}(E) \times \mathcal{P}(S))$$

$$(\exists x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y)\} \text{ (for } y \in T) \qquad (\forall x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y) \mid y \in T\}$$

$$(\mathrm{vis}(e); p) \xrightarrow{e}_{\mathrm{s}} \{p\}$$

Operational semantics for angelic non-determinism

Definition of modules:

set of states   initial state

$$M = (\underset{\text{set of states}}{S}, \rightarrow, \underset{\text{initial state}}{\sigma^0})$$

$$\rightarrow \in \mathcal{P}(S \times \mathsf{option}(E) \times \mathcal{P}(S))$$

$$(\exists x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y)\} \text{ (for } y \in T) \qquad (\forall x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y) \mid y \in T\}$$

$$(\mathsf{vis}(e); p) \xrightarrow{e}_{\mathrm{s}} \{p\}$$

Operational semantics for angelic non-determinism

Definition of modules:

set of states

initial state

$$M = (S, \rightarrow, \sigma^0)$$

transition relation

$$\rightarrow \in \mathcal{P}(S \times \mathrm{option}(E) \times \mathcal{P}(S))$$

demonic

$$(\exists x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y)\} \text{ (for } y \in T)$$

$$(\forall x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y) \mid y \in T\}$$

$$(\mathrm{vis}(e); p) \xrightarrow{e}_{\mathrm{s}} \{p\}$$

Operational semantics for angelic non-determinism

Definition of modules:

set of states

initial state

$$M = (S, \rightarrow, \sigma^0)$$

transition relation

$$\rightarrow \in \mathcal{P}(S \times \mathsf{option}(E) \times \mathcal{P}(S))$$

demonic

angelic

$$(\exists x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y)\} \text{ (for } y \in T)$$

$$(\forall x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y) \mid y \in T\}$$

$$(\mathsf{vis}(e); p) \xrightarrow{e}_{\mathrm{s}} \{p\}$$

Operational semantics for angelic non-determinism

Definition of modules:

set of states

initial state

$$M = (S, \to, \sigma^0)$$

transition relation

$$\to \in \mathcal{P}(S \times \text{option}(E) \times \mathcal{P}(S))$$

demonic

angelic

$$(\exists x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y)\} \text{ (for } y \in T) \qquad (\forall x : T; p(x)) \xrightarrow{\tau}_{\mathrm{s}} \{p(y) \mid y \in T\}$$

$$(\text{vis}(e); p) \xrightarrow{e}_{\mathrm{s}} \{p\}$$

Refinement / Simulation

$$M_1 \preceq M_2 \triangleq (M_1, \sigma^0_{M_1}) \preceq_{\text{co}} (M_2, \sigma^0_{M_2})$$

$(M_1, \sigma_1) \preceq_{\text{co}} (M_2, \sigma_2) \triangleq_{\text{coind}}$

For each demonic choice in M_1

$$\forall e, \Sigma_1.\, \sigma_1 \xrightarrow{e}_{M_1} \Sigma_1 \Rightarrow \exists \Sigma_2.\, \sigma_2 \xrightarrow{e}{}^{*}_{M_2} \Sigma_2 \wedge$$

$$\forall \sigma_2' \in \Sigma_2.\, \exists \sigma_1' \in \Sigma_1.\, (M_1, \sigma_1') \preceq_{\text{co}} (M_2, \sigma_2')$$

Refinement / Simulation

$$M_1 \preceq M_2 \triangleq (M_1, \sigma^0_{M_1}) \preceq_{\text{co}} (M_2, \sigma^0_{M_2})$$

$(M_1, \sigma_1) \preceq_{\text{co}} (M_2, \sigma_2) \triangleq_{\text{coind}}$

For each demonic choice in M_1 exists a demonic choice in M_2, s.t.

$$\forall e, \Sigma_1 .\, \sigma_1 \xrightarrow{e}_{M_1} \Sigma_1 \Rightarrow \exists \Sigma_2 .\, \sigma_2 \xrightarrow{e}{}^{*}_{M_2} \Sigma_2 \wedge$$

$$\forall \sigma_2' \in \Sigma_2 .\, \exists \sigma_1' \in \Sigma_1 .\, (M_1, \sigma_1') \preceq_{\text{co}} (M_2, \sigma_2')$$

Refinement / Simulation

$$M_1 \preceq M_2 \triangleq (M_1, \sigma^0_{M_1}) \preceq_{\mathrm{co}} (M_2, \sigma^0_{M_2})$$

$(M_1, \sigma_1) \preceq_{\mathrm{co}} (M_2, \sigma_2) \triangleq_{\mathrm{coind}}$

For each demonic choice in M_1 *exists a demonic choice in M_2, s.t.*

$$\forall e, \Sigma_1.\, \sigma_1 \xrightarrow{e}_{M_1} \Sigma_1 \Rightarrow \exists \Sigma_2.\, \sigma_2 \xrightarrow{e}{}^{*}_{M_2} \Sigma_2 \wedge$$

$$\forall \sigma_2' \in \Sigma_2.\, \exists \sigma_1' \in \Sigma_1.\, (M_1, \sigma_1') \preceq_{\mathrm{co}} (M_2, \sigma_2')$$

for each angelic choice in M_2 *exists an angelic choice in M_1.*

Refinement / Simulation

$$M_1 \preceq M_2 \triangleq (M_1, \sigma^0_{M_1}) \preceq_{\text{co}} (M_2, \sigma^0_{M_2})$$

$$(M_1, \sigma_1) \preceq_{\text{co}} (M_2, \sigma_2) \triangleq_{\text{coind}}$$

For each demonic choice in M_1 *exists a demonic choice in M_2, s.t.*

$$\forall e, \Sigma_1 .\, \sigma_1 \xrightarrow{e}_{M_1} \Sigma_1 \Rightarrow \exists \Sigma_2 .\, \sigma_2 \xrightarrow{e}{}^{*}_{M_2} \Sigma_2 \wedge$$

$$\forall \sigma_2' \in \Sigma_2 .\, \exists \sigma_1' \in \Sigma_1 .\, (M_1, \sigma_1') \preceq_{\text{co}} (M_2, \sigma_2')$$

for each angelic choice in M_2 *exists an angelic choice in M_1.*

Refinement / Simulation

$$M_1 \preceq M_2 \triangleq (M_1, \sigma^0_{M_1}) \preceq_{co} (M_2, \sigma^0_{M_2})$$

$(M_1, \sigma_1) \preceq_{co} (M_2, \sigma_2) \triangleq_{\text{coind}}$

For each demonic choice in M_1 *exists a demonic choice in M_2, s.t.*

$$\forall e, \Sigma_1 .\, \sigma_1 \xrightarrow{e}_{M_1} \Sigma_1 \Rightarrow \exists \Sigma_2 .\, \sigma_2 \xrightarrow{e}{}^{*}_{M_2} \Sigma_2 \wedge$$

$$\forall \sigma_2' \in \Sigma_2 .\, \exists \sigma_1' \in \Sigma_1 .\, (M_1, \sigma_1') \preceq_{co} (M_2, \sigma_2')$$

for each angelic choice in M_2 *exists an angelic choice in M_1.*