Pushing the Frame Rule into Abstract Interpretation

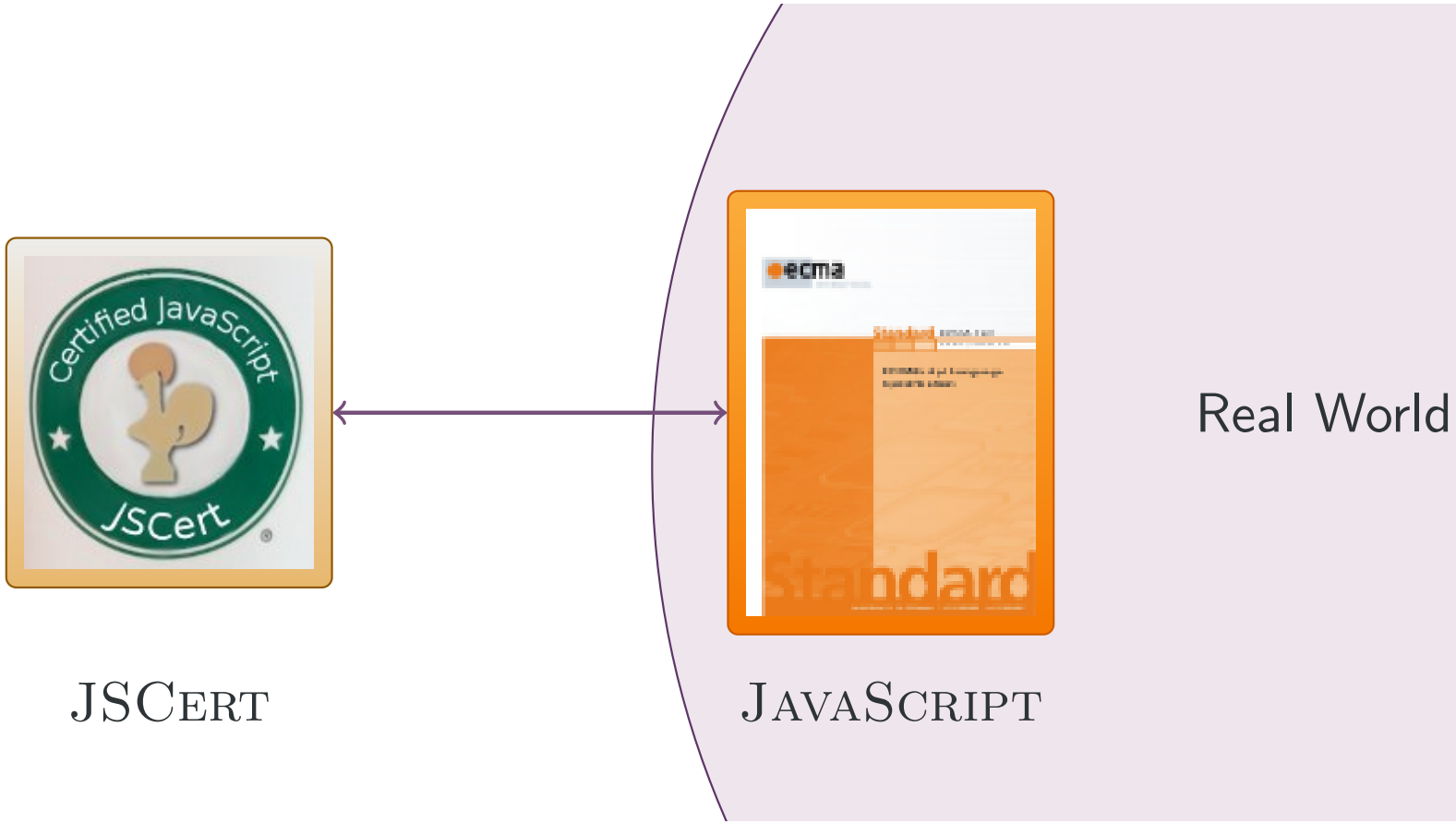
Martin Bodin /maʁtɛ̃ bodɛ̃/
Thomas Jensen Alan Schmitt

Inria

30th of May

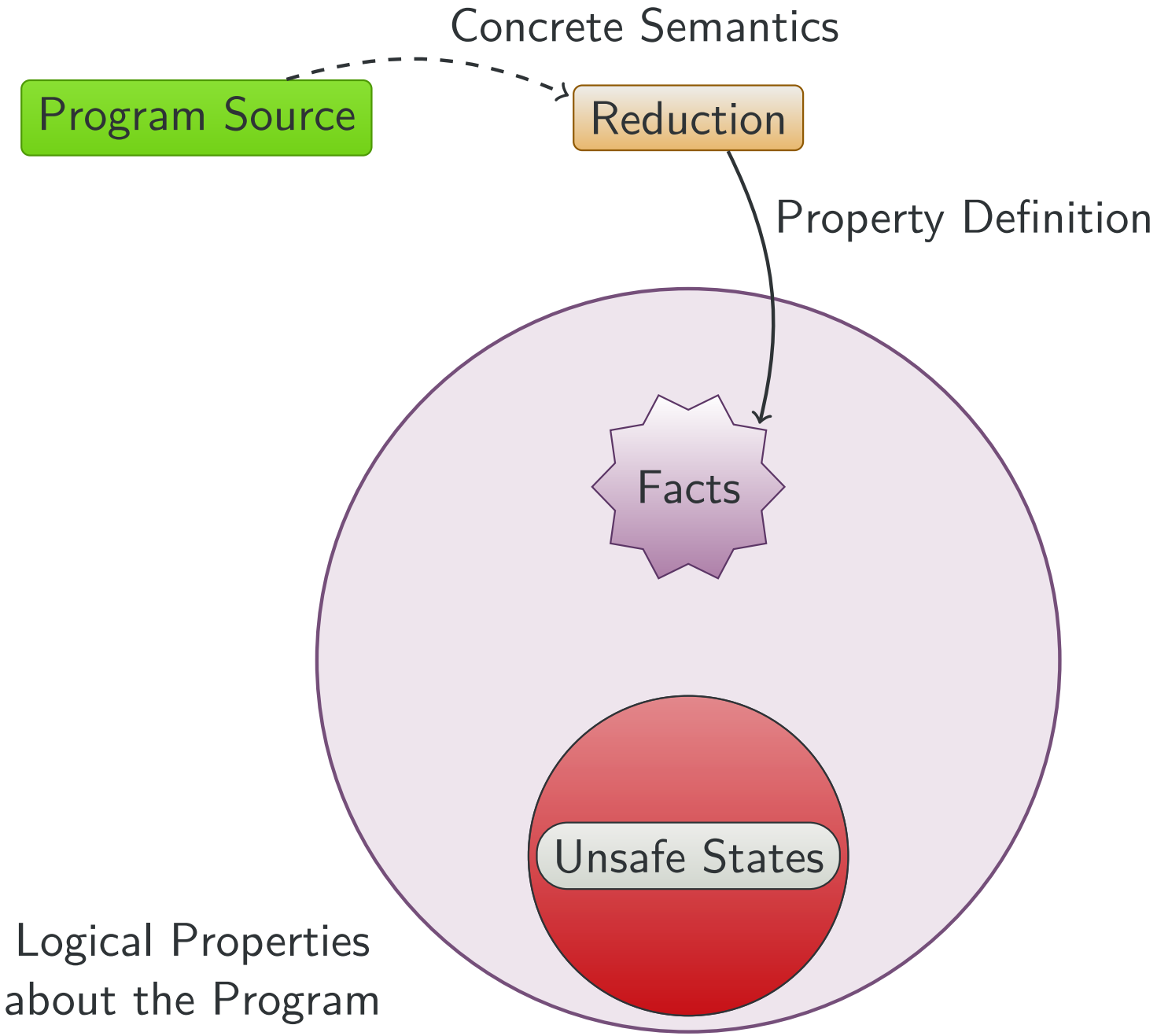
Séminaire Gallium

JavaScript's Semantics

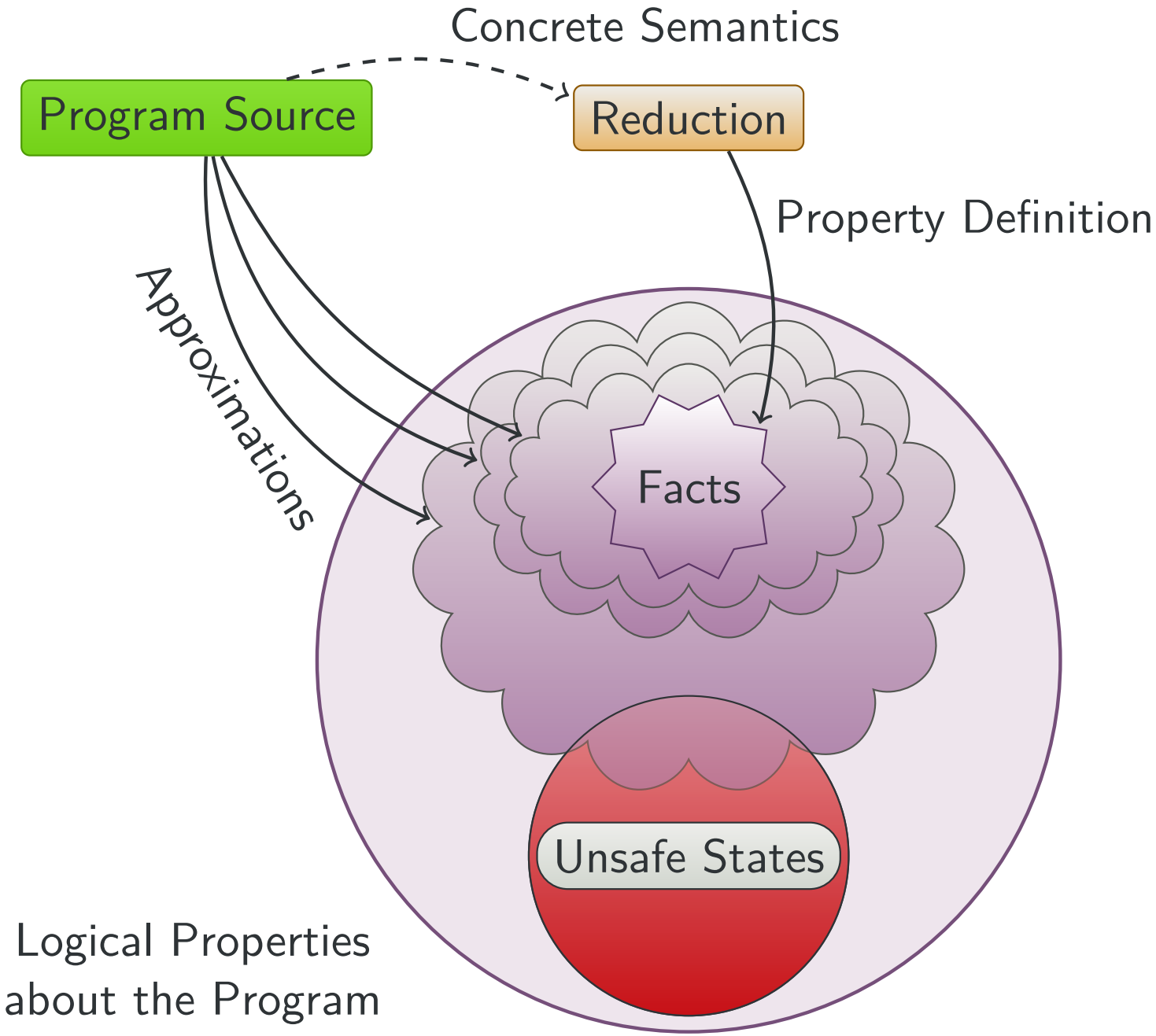


- More than 900 reduction rules.
- How to define a certified analyser of JavaScript?

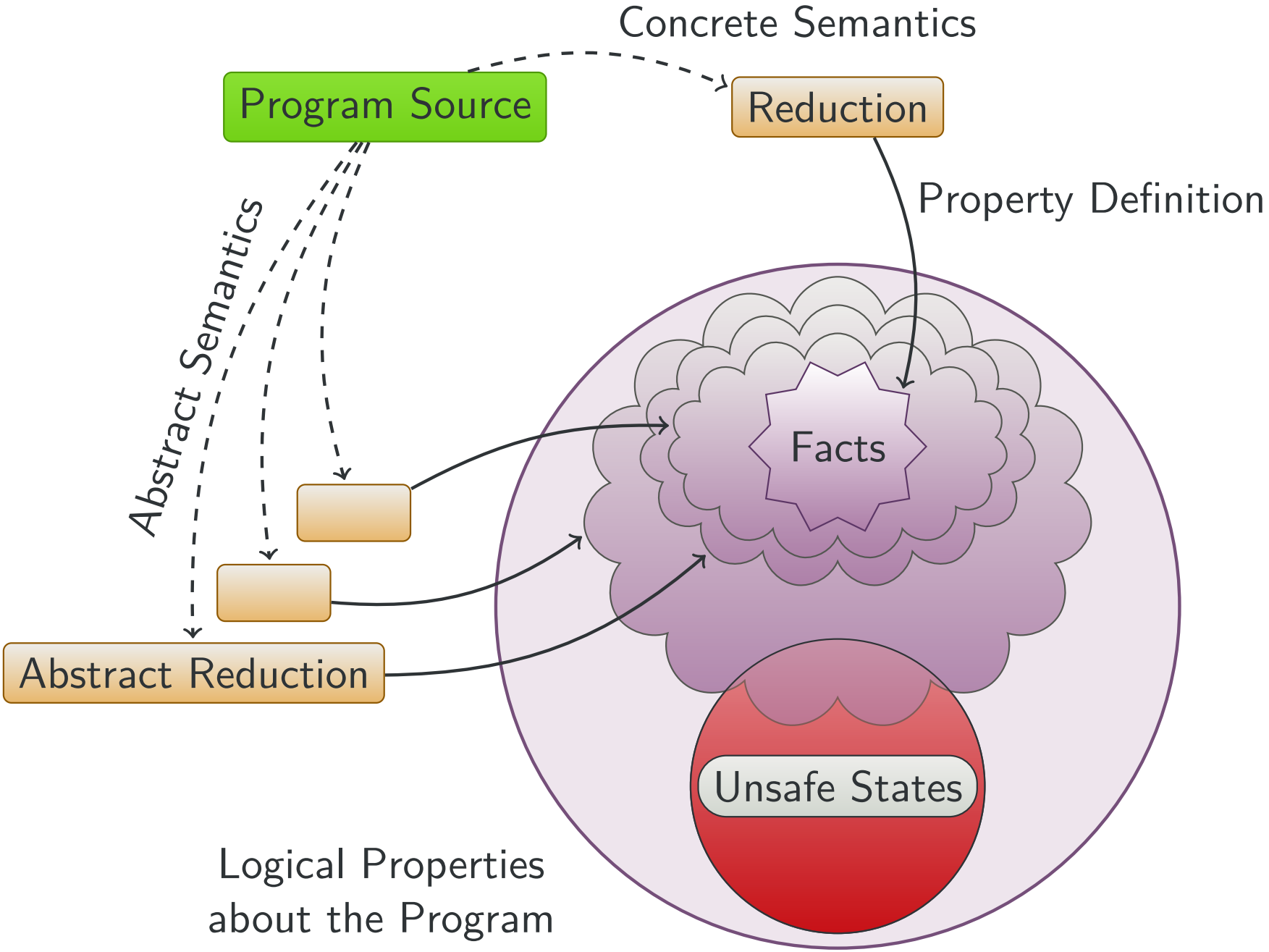
Abstract Interpretation



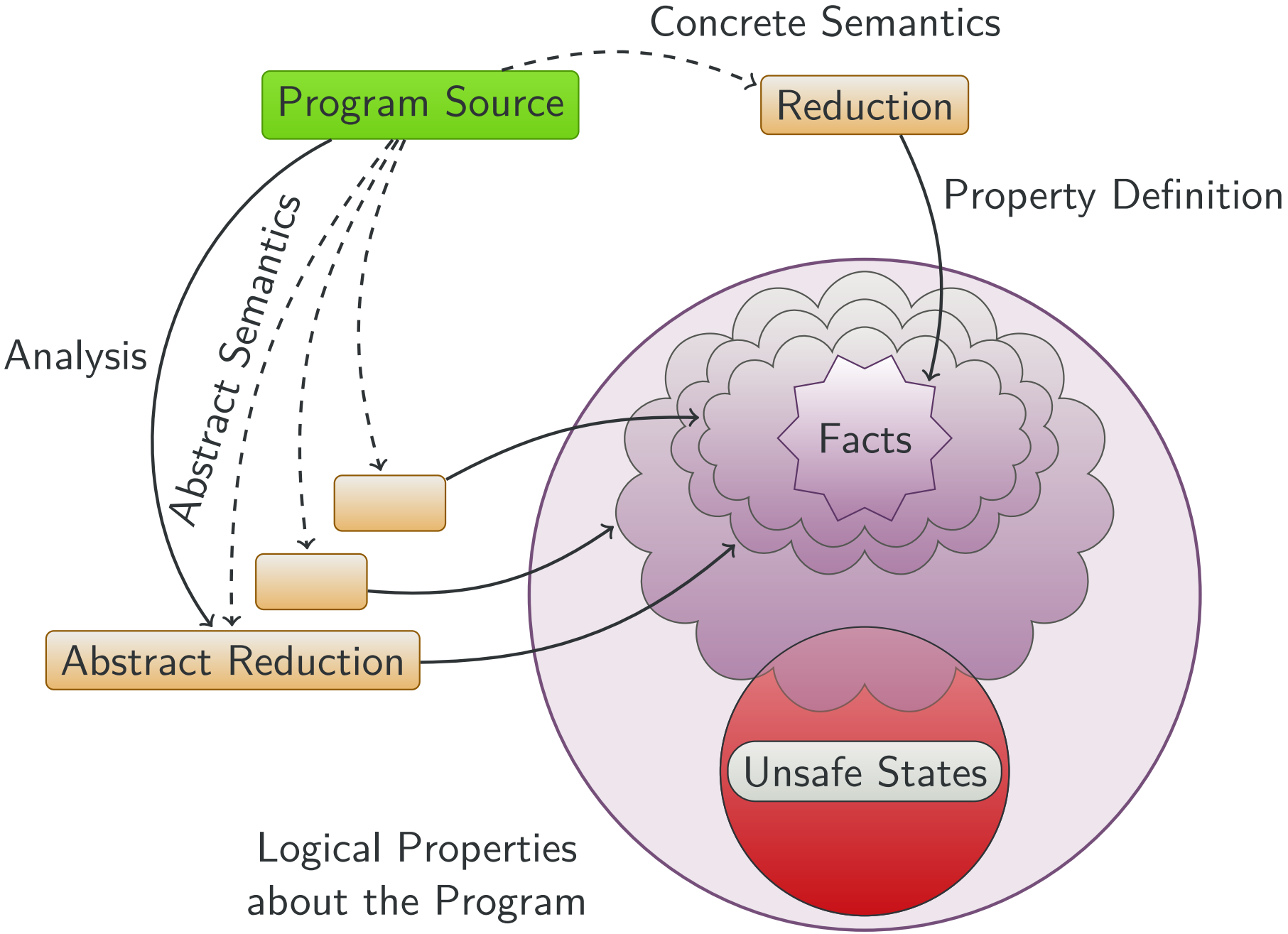
Abstract Interpretation



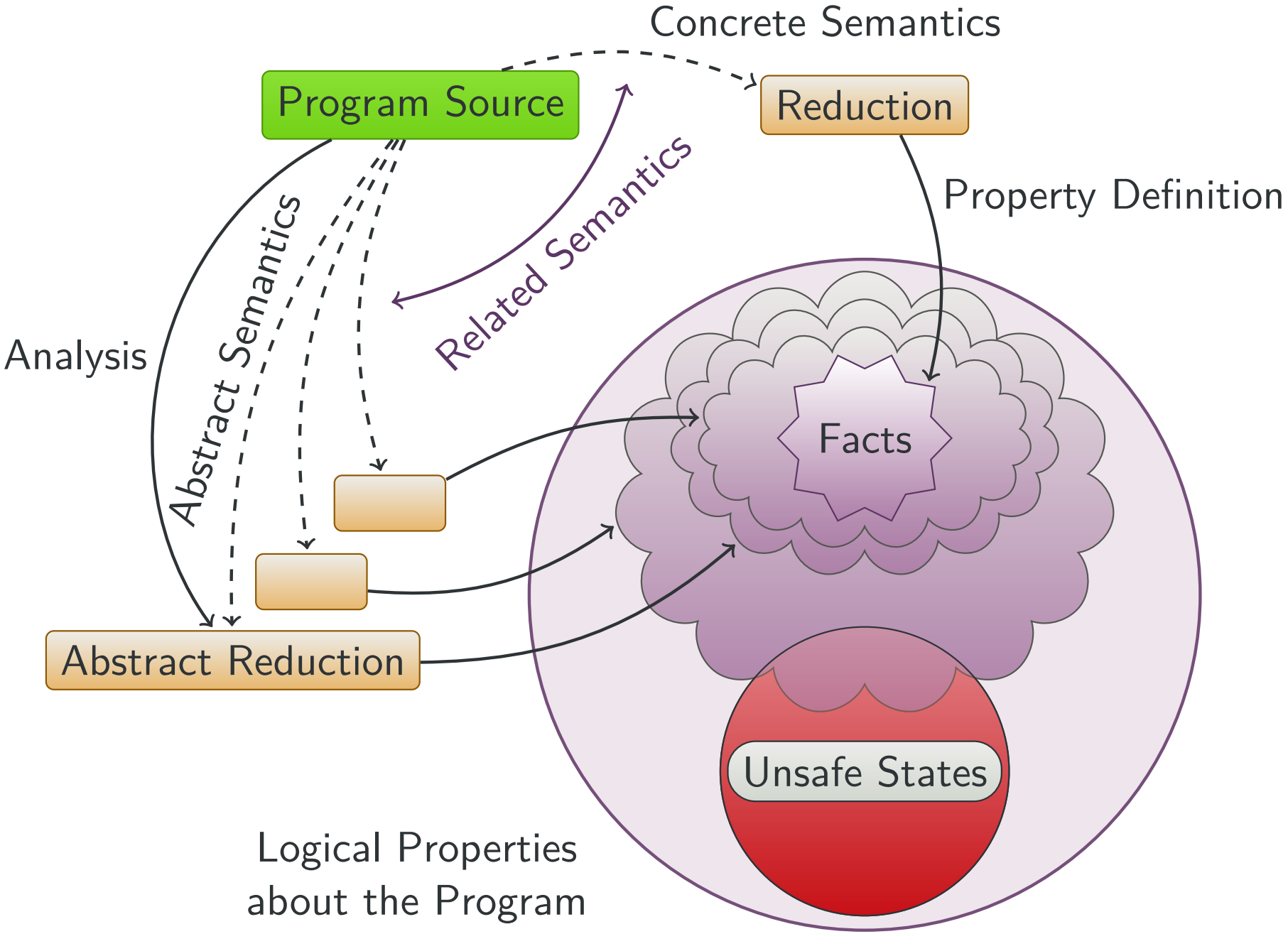
Abstract Interpretation



Abstract Interpretation



Abstract Interpretation



Correctness of an Abstract Semantics

The abstract semantics misses no concrete behaviour

$$\begin{aligned} & \sigma^\sharp, t \Downarrow^\sharp r^\sharp \\ \wedge\ & \sigma, t \Downarrow r \\ \wedge\ & \sigma \in \gamma\left(\sigma^\sharp\right) \\ & \quad \Longrightarrow\ r \in \gamma\left(r^\sharp\right) \end{aligned}$$

- σ, semantic context,
- r, results,
- γ, concretisation functions,
- derivation conclusions of the form $\sigma, t \Downarrow r$.

Building an Abstract Semantics (Traditional Way)

Concrete Semantics

Abstract Semantics

$$\textsc{IfTrue}\ \frac{E, s_1 \Downarrow E'}{E, \mathit{if}\ (\mathit{true})\ s_1\, s_2 \Downarrow E'}$$

$$\textsc{IfFalse}\ \frac{E, s_2 \Downarrow E'}{E, \mathit{if}\ (\mathit{false})\ s_1\, s_2 \Downarrow E'}$$

Building an Abstract Semantics (Traditional Way)

Concrete Semantics

IfTrue

$$\frac{E, s_1 \Downarrow E'}{E, \mathit{if}\ (\mathit{true})\ s_1\, s_2 \Downarrow E'}$$

IfFalse

$$\frac{E, s_2 \Downarrow E'}{E, \mathit{if}\ (\mathit{false})\ s_1\, s_2 \Downarrow E'}$$

Abstract Semantics

IfTrue

$$\frac{E^\sharp, s_1 \Downarrow^\sharp {E'}^\sharp}{E^\sharp, \mathit{if}\ \left(\mathit{true}^\sharp\right)\ s_1\, s_2 \Downarrow^\sharp {E'}^\sharp}$$

IfFalse

$$\frac{E^\sharp, s_2 \Downarrow^\sharp {E'}^\sharp}{E^\sharp, \mathit{if}\ \left(\mathit{false}^\sharp\right)\ s_1\, s_2 \Downarrow^\sharp {E'}^\sharp}$$

IfTop

$$\frac{E^\sharp, s_1 \Downarrow^\sharp {E'_1}^\sharp \qquad E^\sharp, s_2 \Downarrow^\sharp {E'_2}^\sharp}{E^\sharp, \mathit{if}\ (\top)\ s_1\, s_2 \Downarrow^\sharp {E'_1}^\sharp \sqcup {E'_2}^\sharp}$$

IfBot

$$\frac{}{E^\sharp, \mathit{if}\ (\bot)\ s_1\, s_2 \Downarrow^\sharp \bot}$$

Building an Abstract Semantics (Traditional Way)

Concrete Semantics

$$\textsc{While}\quad \frac{E, \mathit{if}\ (e)\ (s;\ \mathit{while}\ (e)\ s)\ \mathit{skip} \Downarrow E'}{E, \mathit{while}\ (e)\ s \Downarrow E'}$$

Abstract Semantics

$$\textsc{Weaken}\quad \frac{E_1^\sharp \sqsubseteq E_2^\sharp \qquad E_2^\sharp, t \Downarrow^\sharp E_3^\sharp \qquad E_3^\sharp \sqsubseteq E_4^\sharp}{E_1^\sharp, t \Downarrow^\sharp E_4^\sharp}$$

$$\textsc{While}\quad \frac{E^\sharp, s \Downarrow^\sharp E^\sharp}{E^\sharp, \mathit{while}\ (e)\ s \Downarrow^\sharp E^\sharp}$$

Building an Abstract Semantics (Traditional Way)

Concrete Semantics

IfTrue

$$\frac{E, s_1 \Downarrow E'}{E, \mathit{if}\ (\mathit{true})\ s_1\, s_2 \Downarrow E'}$$

IfFalse

$$\frac{E, s_2 \Downarrow E'}{E, \mathit{if}\ (\mathit{false})\ s_1\, s_2 \Downarrow E'}$$

Abstract Semantics

IfTrue

$$\frac{E^\sharp, s_1 \Downarrow^\sharp {E'}^\sharp}{E^\sharp, \mathit{if}\ \left(\mathit{true}^\sharp\right)\ s_1\, s_2 \Downarrow^\sharp {E'}^\sharp}$$

IfFalse

$$\frac{E^\sharp, s_2 \Downarrow^\sharp {E'}^\sharp}{E^\sharp, \mathit{if}\ \left(\mathit{false}^\sharp\right)\ s_1\, s_2 \Downarrow^\sharp {E'}^\sharp}$$

IfTop

$$\frac{E^\sharp, s_1 \Downarrow^\sharp {E'_1}^\sharp \qquad E^\sharp, s_2 \Downarrow^\sharp {E'_2}^\sharp}{E^\sharp, \mathit{if}\ (\top)\ s_1\, s_2 \Downarrow^\sharp {E'_1}^\sharp \sqcup {E'_2}^\sharp}$$

IfBot

$$\frac{}{E^\sharp, \mathit{if}\ (\bot)\ s_1\, s_2 \Downarrow^\sharp \bot}$$

Building an Abstract Semantics (Traditional Way)

Concrete Semantics

IfTrue

$$\frac{E, s_1 \Downarrow E'}{E, \mathit{if}\ (\mathit{true})\ s_1\ s_2 \Downarrow E'}$$

$$\frac{E, s_2 \Downarrow E'}{E, \mathit{if}\ (\mathit{false})\ s_1\ s_2 \Downarrow E'}$$

Abstract Semantics

IfTrue

$$\frac{E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\ (\mathit{true}^\sharp)\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}$$

IfFalse

$$\frac{E^\sharp, s_2 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\ (\mathit{false}^\sharp)\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}$$

IfTop

$$\frac{E^\sharp, s_1 \Downarrow^\sharp E_1'^\sharp \qquad E^\sharp, s_2 \Downarrow^\sharp E_2'^\sharp}{E^\sharp, \mathit{if}\ (\top)\ s_1\ s_2 \Downarrow^\sharp E_1'^\sharp \sqcup E_2'^\sharp}$$

IfBot

$$\frac{}{E^\sharp, \mathit{if}\ (\bot)\ s_1\ s_2 \Downarrow^\sharp \bot}$$

Definitions and proofs very long

Mostly systematic... but exponential on the size of the concrete semantics!

- There has to be a better solution.

Our Solution: Abstract Each Rule Locally

Concrete Semantics

Abstract Semantics

$$\textsc{IfTrue}\ \frac{E, s_1 \Downarrow E'}{E, \mathit{if}\ (\mathit{true})\ s_1\, s_2 \Downarrow E'} \quad \longleftrightarrow \quad \textsc{IfTrue}\ \frac{E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\ (\mathit{true}^\sharp)\ s_1\, s_2 \Downarrow^\sharp E'^\sharp}$$

$$\textsc{IfFalse}\ \frac{E, s_2 \Downarrow E'}{E, \mathit{if}\ (\mathit{false})\ s_1\, s_2 \Downarrow E'} \quad \longleftrightarrow \quad \textsc{IfFalse}\ \frac{E^\sharp, s_2 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\ (\mathit{false}^\sharp)\ s_1\, s_2 \Downarrow^\sharp E'^\sharp}$$

Our Solution: Abstract Each Rule Locally

Concrete Semantics

Abstract Semantics

$$\textsc{IfTrue}\ \frac{E, s_1 \Downarrow E'}{E, \mathit{if}\ (\mathit{true})\ s_1\, s_2 \Downarrow E'} \qquad \textsc{IfTrue}\ \frac{E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\ (\mathit{true}^\sharp)\ s_1\, s_2 \Downarrow^\sharp E'^\sharp}$$

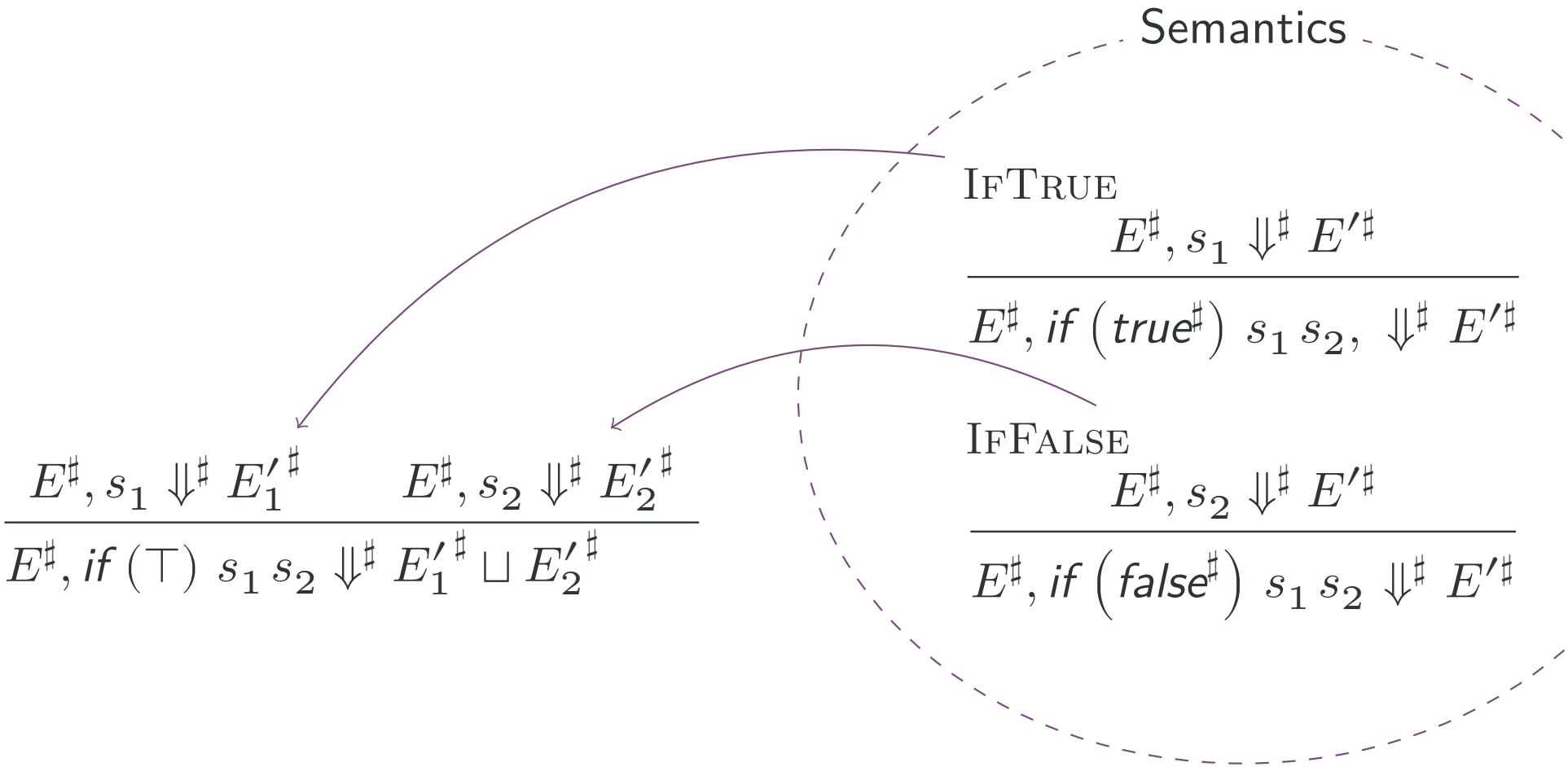
$$\textsc{IfFalse}\ \frac{E, s_2 \Downarrow E'}{E, \mathit{if}\ (\mathit{false})\ s_1\, s_2 \Downarrow E'} \qquad \textsc{IfFalse}\ \frac{E^\sharp, s_2 \Downarrow^\sharp E'^\sharp}{E^\sharp, \mathit{if}\ (\mathit{false}^\sharp)\ s_1\, s_2 \Downarrow^\sharp E'^\sharp}$$

- Side conditions $cond_n^\sharp$,
- Transfer functions.

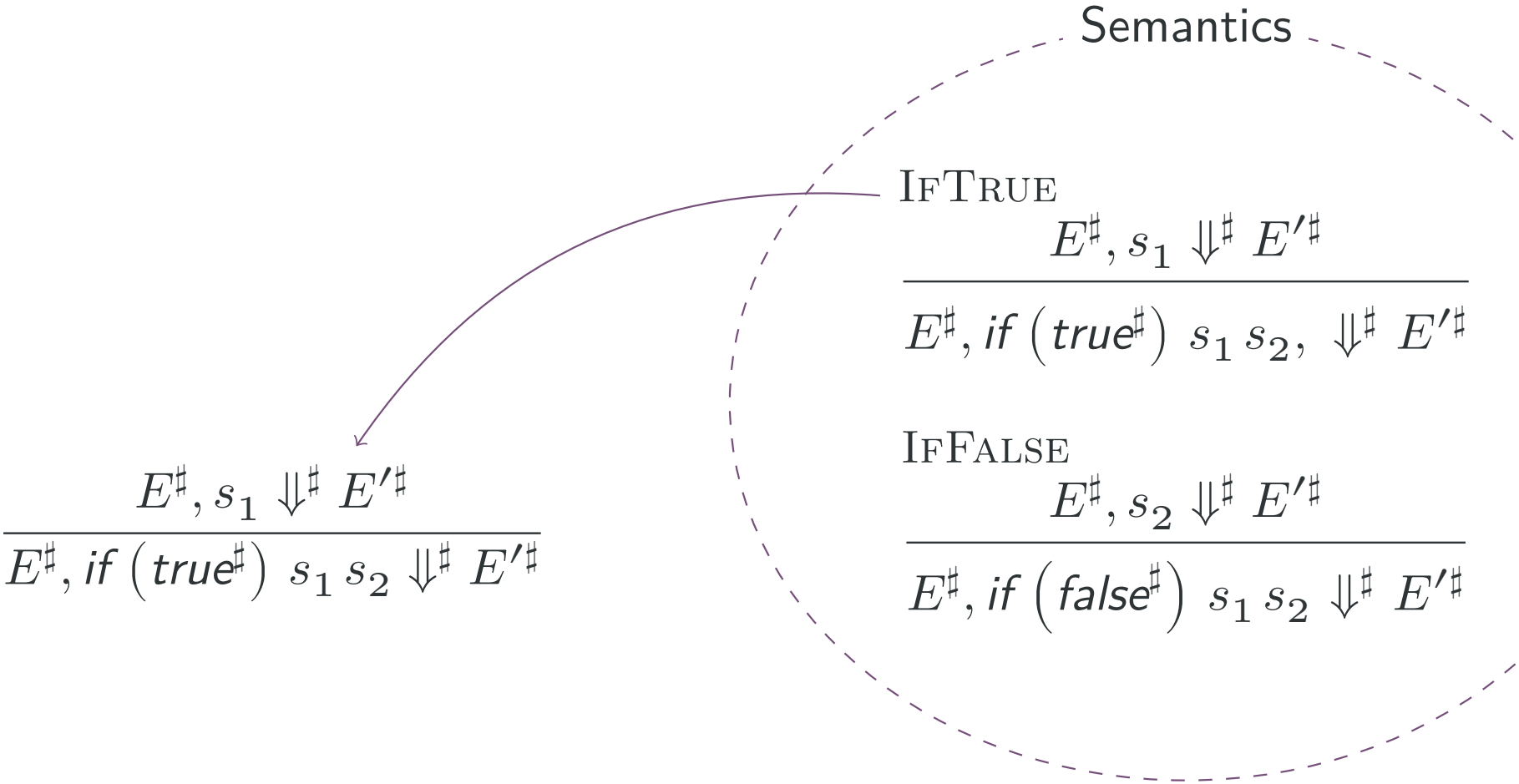
These rules are not correct as-is

- We have to change the way we *derive* an abstract semantics from abstract rules.
- At each step, we have to apply *all* the rules which apply.

Apply all the rules which apply



Apply all the rules which apply



Apply all the rules which apply, formally

- $apply^\sharp_n\left(\Downarrow^\sharp_0\right)$ applies Rule n, using Relation $\Downarrow^\sharp_0$ as preconditions.
  ▸ $apply^\sharp$
- Immediate consequence $\mathcal{F}^\sharp$.

$$\mathcal{F}^\sharp\left(\Downarrow^\sharp_0\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \middle| \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond^\sharp_n\left(\sigma^\sharp\right) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in apply^\sharp_n\left(\Downarrow^\sharp_0\right) \end{array} \right\}$$

Apply all the rules which apply, formally

- $apply^\sharp_n\left(\Downarrow^\sharp_0\right)$ applies Rule n, using Relation $\Downarrow^\sharp_0$ as preconditions.
  ▸ $apply^\sharp$
- Immediate consequence $\mathcal{F}^\sharp$.

$$\mathcal{F}^\sharp\left(\Downarrow^\sharp_0\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \middle| \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond^\sharp_n\left(\sigma^\sharp\right) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in apply^\sharp_n\left(\Downarrow^\sharp_0\right) \end{array} \right\}$$

- $\mathcal{F}^\sharp\left(\emptyset\right)$: all the axioms's conclusions.
- Iterating $\mathcal{F}^\sharp$ k times: all the conclusions of derivations whose depth is less than k.
  - Can abstract any concrete derivation up to depth k.

Not all programs are boundable

- We need a fixed point of $\mathcal{F}^\sharp$, but which one?

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \middle| \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

The least fixed point

To use it, we need to bound the size of the concrete derivation.

The greatest fixed point

To use it, we need to find an invariant.

Not all programs are boundable

- We need a fixed point of $\mathcal{F}^\sharp$, but which one?

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

Why do we even hesitate?

All the fixed points are correct, but one allows to prove more.

- We consider the greatest fixed point $\Downarrow^\sharp$ of $\mathcal{F}^\sharp$.

Derivable rule

$$\textsc{While}\ \frac{E^\sharp, s \Downarrow^\sharp E^\sharp}{E^\sharp, \mathit{while}\ (e)\ s \Downarrow^\sharp E^\sharp}$$

Weakening

$$\textsc{Weaken}\ \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

- The $\textsc{Weaken}$ rule is not sound in a coinductive definition.

$$\dfrac{\dfrac{\dfrac{\vdots}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}\ \textsc{Weaken}}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}\ \textsc{Weaken}$$

Weakening

$$\textsc{Weaken}\quad \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, \sigma^\sharp \sqsubseteq \sigma'^\sharp \wedge r'^\sharp \sqsubseteq r^\sharp \\ \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

Weakening

$$\textsc{Weaken}\quad \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \middle| \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow (\sigma^\sharp, t, r^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \middle| \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, \sigma^\sharp \sqsubseteq \sigma'^\sharp \wedge r'^\sharp \sqsubseteq r^\sharp \\ \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \middle| \begin{array}{c} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, glue_n\left(\sigma'^\sharp, r'^\sharp, \sigma^\sharp, r^\sharp\right) \\ \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

Glue

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \;\middle|\; \begin{array}{l} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \quad \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, glue_{\,n}\left(\sigma'^\sharp, r'^\sharp, \sigma^\sharp, r^\sharp\right) \\ \quad \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

- *glue* is an inductively defined predicate.

$$\textsc{Filter}(n)\ \frac{\sigma_{|cond_n}, t \Downarrow^\sharp r}{\sigma, t \Downarrow^\sharp r} \qquad \textsc{Weaken}\ \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

Summing up

Differences between $\Downarrow$ and $\Downarrow^\sharp$

Concrete Semantics $\Downarrow$	Abstract Semantics $\Downarrow^\sharp$
At each step, apply *one* rule that applies	At each step, apply *all* the rules that apply
Inductive interpretation of the rules $\Downarrow = lfp\,(\mathcal{F})$	Co-inductive interpretation of the rules $\Downarrow^\sharp = gfp\,(\mathcal{F}^\sharp)$
	Some glue (approximations)

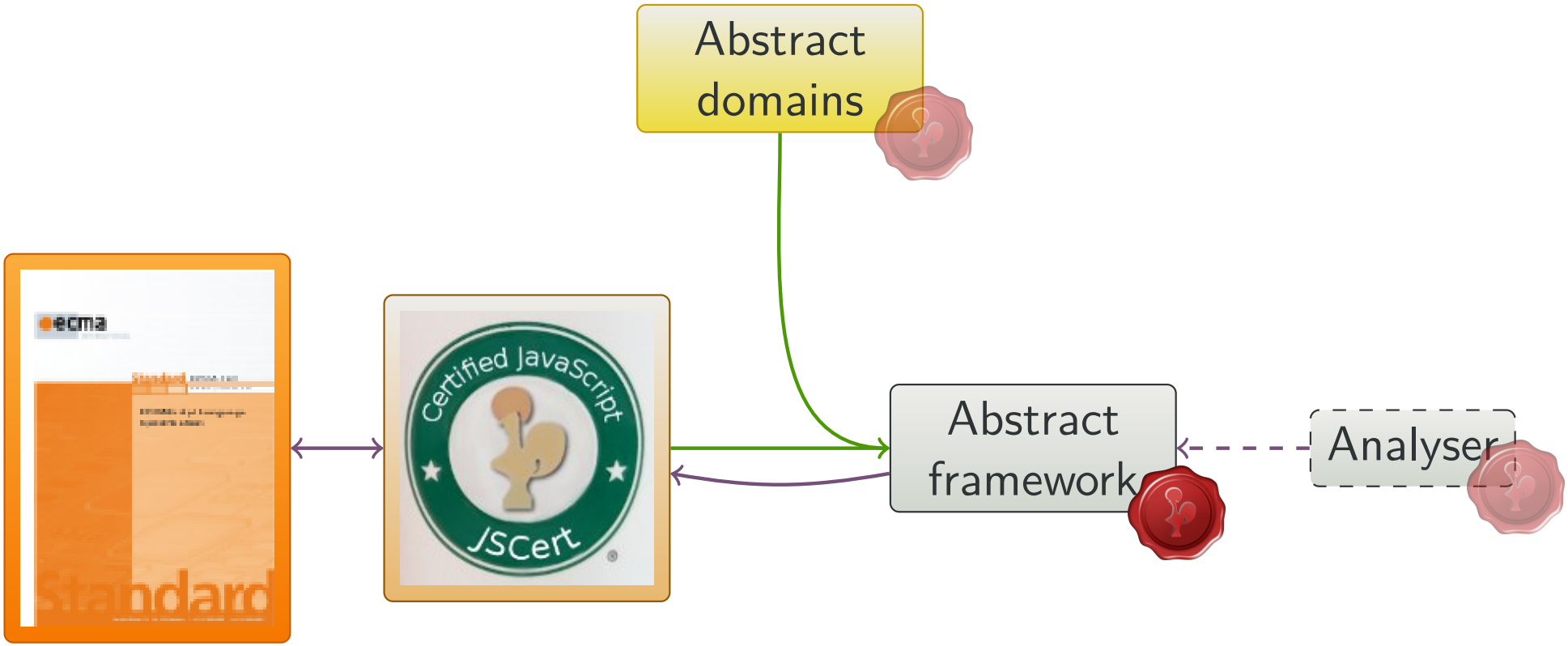
$$\frac{E_0^\sharp, s_1 \Downarrow^\sharp E_1^\sharp \quad \uparrow \textsc{IfTrue} \qquad E_0^\sharp, s_2 \Downarrow^\sharp E_2^\sharp \quad \uparrow \textsc{IfFalse}}{E_0^\sharp, \mathit{if}\ (e)\ s_1\, s_2 \Downarrow^\sharp E_1^\sharp \sqcup E_2^\sharp}$$

1. Abstract Interpretation

2. Separation Logic

3. Adding Summary Nodes

Global Situation



JavaScript JSCert

JavaScript 's Memory Model

- No pointer arithmetic!
- Locations $l^i \in \mathit{Loc} \subseteq \mathit{Val}$.
- A heap $H : \mathit{Loc} \rightharpoonup \mathfrak{F} \rightharpoonup \mathit{Val}$.

Field names

- Allocating new locations.
- Adding/removing fields on the fly.
- Checking whether a field exists.

Interlinked Extensible Records

Separation Logic

$$\phi ::= \mathit{emp} \mid \phi_1 \star \phi_2 \mid l \mapsto \{o\}$$
$$o ::= \mathtt{f} : v^\sharp, o \mid \_ : v^\sharp$$

The lattice of abstract values

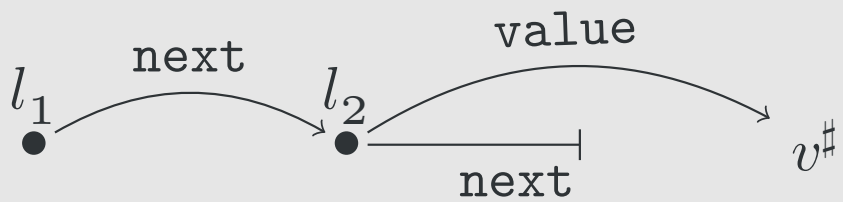
$$+, \pm, -, \ldots \in v^\sharp$$
$$l \in v^\sharp$$
$$\mathit{nil} \in v^\sharp$$
$$\boxtimes \in v^\sharp$$

Because of JavaScript's dynamism, values of different kinds can be mixed: $+ \sqcup l \sqcup \boxtimes$.

Separation Logic

$$\phi ::= \mathit{emp} \mid \phi_1 \star \phi_2 \mid l \mapsto \{o\}$$
$$o ::= \mathtt{f} : v^\sharp, o \mid \_ : v^\sharp$$

$l_1 \mapsto \{\mathtt{next} : l_2, \_ : \boxtimes\} \star l_2 \mapsto \{\mathtt{next} : \mathit{nil}, \mathtt{value} : +, \_ : \boxtimes\}$



$$\phi ::= \mathit{emp} \mid \phi_1 \star \phi_2 \mid l \mapsto \{o\}$$
$$o ::= \mathtt{f} : v^\sharp, o \mid \_ : v^\sharp$$

Write

$$\frac{}{l \mapsto \{\mathtt{f} : \_, \ldots\}, l.\mathtt{f} := v^\sharp \Downarrow^\sharp l \mapsto \{\mathtt{f} : v^\sharp, \ldots\}}$$

Delete

$$\frac{}{l \mapsto \{\mathtt{f} : \_, \ldots\}, \mathit{delete}\, l.\mathtt{f} \Downarrow^\sharp l \mapsto \{\mathtt{f} : \boxtimes, \ldots\}}$$

The Frame Rule

$$\textsc{Frame}\ \frac{\phi, s \Downarrow^{\sharp} \phi'}{\phi \star \phi_c, s \Downarrow^{\sharp} \phi' \star \phi_c}$$

- Everything not explicitly changed is unchanged.
- Allows to focus during function analyses.

Good news

We can use the glue.

The Frame Rule

$$\textsc{Frame}\quad \frac{\phi, s \Downarrow^\sharp \phi'}{\phi \star \phi_c, s \Downarrow^\sharp \phi' \star \phi_c}$$

$$\textsc{Weaken}\quad \frac{\sigma^\sharp \sqsubseteq \sigma'^\sharp \qquad \sigma'^\sharp, t \Downarrow^\sharp r'^\sharp \qquad r'^\sharp \sqsubseteq r^\sharp}{\sigma^\sharp, t \Downarrow^\sharp r^\sharp}$$

- No longer possible to change ϕ to ϕ' if the interface changed, even though $\gamma(\phi) = \gamma(\phi')$.

Keep track of the interface using a *membrane*

$$(l_a \to l_1, l_b \to l_2 \,|\, l_1 \mapsto \{\mathtt{f} : l_2, \_ : \boxtimes\})$$
$$= (l_a \to l_2, l_b \to l_3 \,|\, l_2 \mapsto \{\mathtt{f} : l_3, \_ : \boxtimes\})$$

- Membranes M are maps from "outer" locations to "inner" locations.
- They scope the formulae.
- They also track new locations:

$$\textsc{Alloc}\quad \frac{}{(M \,|\, \mathit{emp})\,, \mathit{alloc}\,() \Downarrow^\sharp (M, \nu l \,|\, l \mapsto \{\_ : \boxtimes\}\,, l)}$$

Extended Formulae

$$
\begin{aligned}
\Phi &::= (M|\phi) \\
M &::= M, M \mid l \to l' \mid \nu l \\
\phi &::= \mathit{emp} \mid \phi_1 \star \phi_2 \mid l \mapsto \{o\} \\
o &::= \mathtt{f} : v^\sharp, o \mid \_ : v^\sharp
\end{aligned}
$$

with some well-formedness conditions.

Extended Formulae

$$
\begin{aligned}
\Phi &::= (M|\phi) \\
M &::= M, M \mid l \to l' \mid \nu l \\
\phi &::= \mathit{emp} \mid \phi_1 \star \phi_2 \mid l \mapsto \{o\} \\
o &::= \mathtt{f} : v^\sharp, o \mid \_ : v^\sharp
\end{aligned}
$$

with some well-formedness conditions.

Two frame operators

- $\circledcirc$ and $\circledast$ update a formula Φ to some context.
- They can fail to compute a new formula.

$$\textsc{Frame-}\circledcirc \quad \frac{\Phi, t \Downarrow \Phi'}{M \circledcirc \Phi, t \Downarrow M \circledcirc \Phi'}$$

$$\textsc{Frame-}\circledast \quad \frac{\Phi, t \Downarrow \Phi'}{\phi \circledast \Phi, t \Downarrow \phi \circledast \Phi'}$$

How does the frame rule work?

$$\textsc{Frame-}\circledast \quad \frac{\Phi, t \Downarrow \Phi'}{\phi \circledast \Phi, t \Downarrow \phi \circledast \Phi'}$$

A semantic property to prove on transfer functions

- Concrete derivations have to be rewritable, by removing unused contexts H_c.

$$\frac{\dots}{H_c \bullet H, t \Downarrow H_c \bullet H'} \dashrightarrow \frac{\dots}{H, t \Downarrow H'}$$

▸ In Coq

- This is strongly linked with the notion of *footprints* in separation logic.
- Coq not finished yet.

1. Abstract Interpretation

2. Separation Logic

3. Adding Summary Nodes

What about loops?

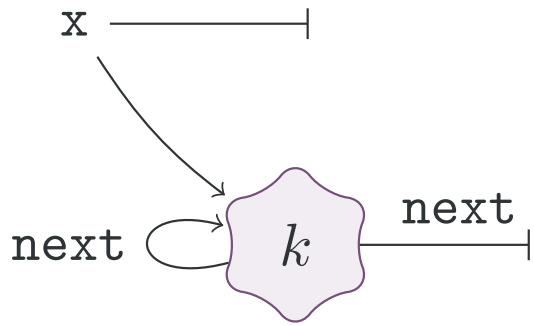
- A simple abstraction: summary nodes.

```
x := nil;
while (e) (
    t := alloc();
    t.next := x;
    x := t
)
```

What about loops?

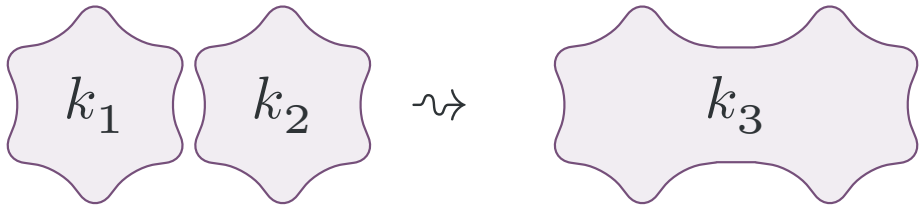
- A simple abstraction: summary nodes.

```
x := nil;
while (e) (
    t := alloc()^k;
    t.next := x;
    x := t
)
```



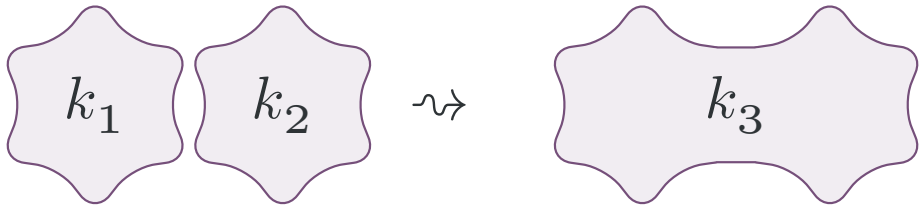
$k \mapsto \{\texttt{next} : k \sqcup \mathit{nil}, \_ : \boxtimes\}$

Summary nodes change interfaces



$$k_1 \mapsto \{\texttt{f} : +, \_ : \boxtimes\} \star k_2 \mapsto \{\texttt{f} : +, \_ : \boxtimes\}$$
$$\rightsquigarrow k_3 \mapsto \{\texttt{f} : +, \_ : \boxtimes\}$$

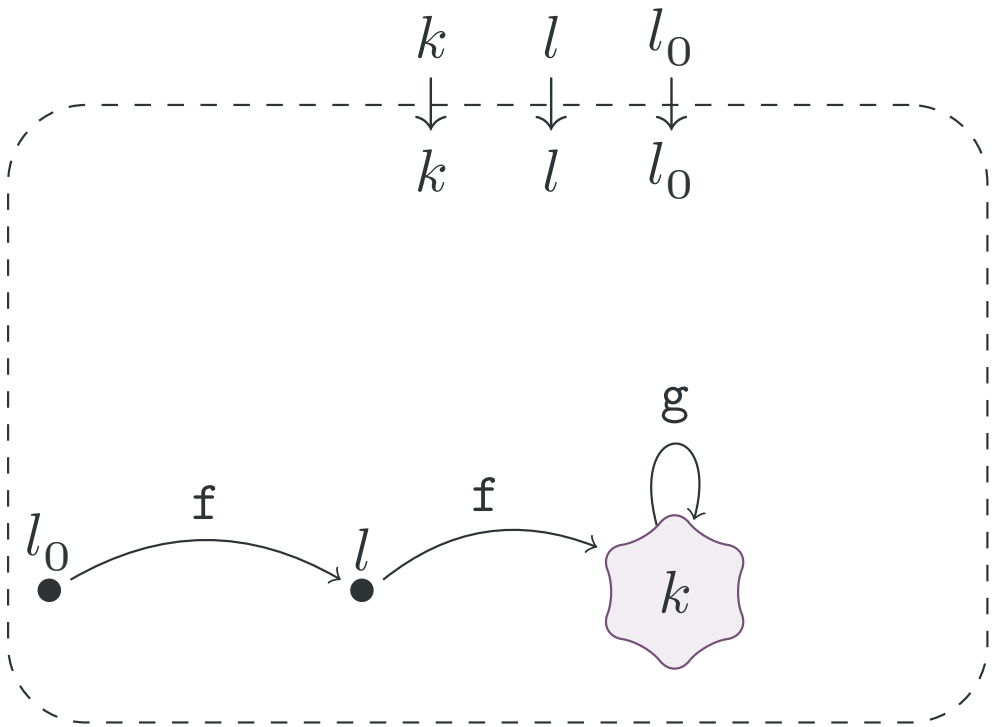
Summary nodes change interfaces



$$(k_a \to k_1, k_b \to k_2 \,|\, k_1 \mapsto \{\texttt{f} : +, \_ : \boxtimes\} \star k_2 \mapsto \{\texttt{f} : +, \_ : \boxtimes\})$$
$$\rightsquigarrow (k_a \to k_3, k_b \to k_3 \,|\, k_3 \mapsto \{\texttt{f} : +, \_ : \boxtimes\})$$

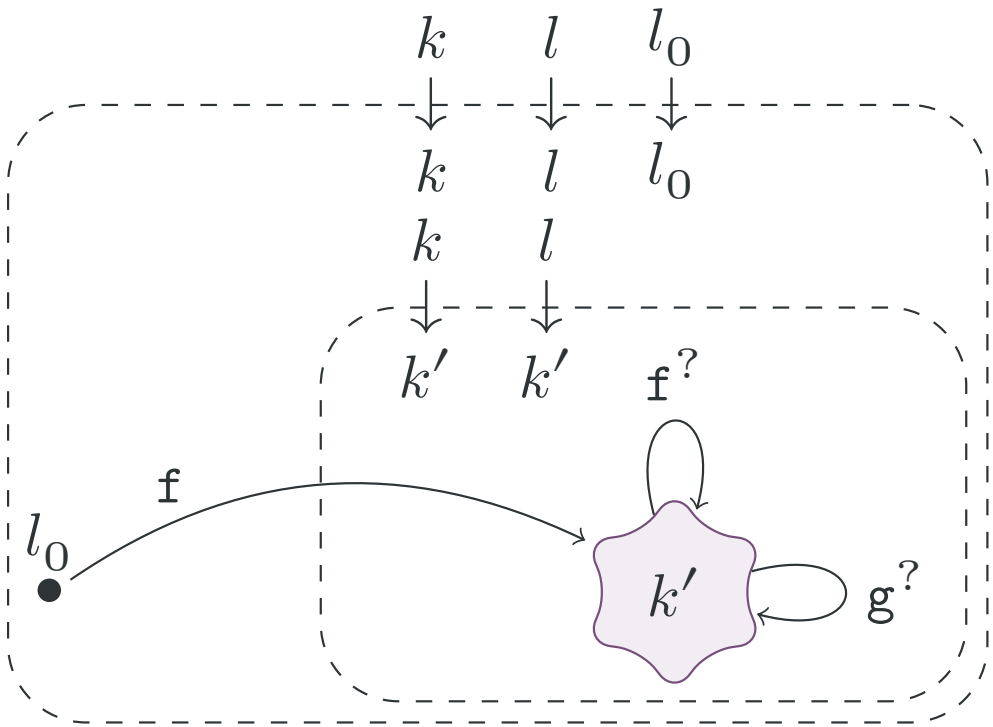
An example of membrane manipulation

$$(k \to k, l \to l, l_0 \to l_0 \,|\, l_0 \mapsto \{\mathtt{f} : l, \_ : \boxtimes\}$$
$$\star\, l \mapsto \{\mathtt{f} : k, \_ : \boxtimes\} \star k \mapsto \{\mathtt{g} : k, \_ : \boxtimes\})$$



An example of membrane manipulation

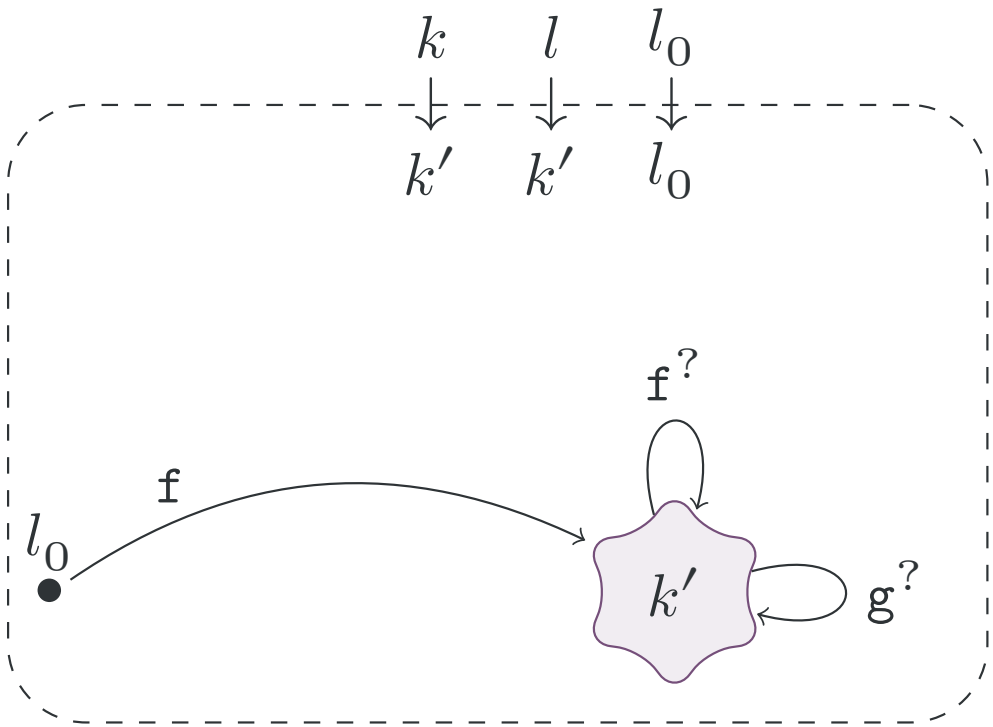
$$(k \to k, l \to l, l_0 \to l_0 \,|\, l_0 \mapsto \{\mathtt{f} : l,\ \_ : \boxtimes\} \star l \mapsto \{\mathtt{f} : k,\ \_ : \boxtimes\} \star k \mapsto \{\mathtt{g} : k,\ \_ : \boxtimes\})$$



$$(k \to k, l \to l) \circledcirc l_0 \mapsto \{\mathtt{f} : l,\ \_ : \boxtimes\} \circledcirc \circledast (k \to k', l \to k' \,|\, k' \mapsto \{\mathtt{f} : k' \sqcup \boxtimes,\ \mathtt{g} : k' \sqcup \boxtimes,\ \_ : \boxtimes\})$$

An example of membrane manipulation

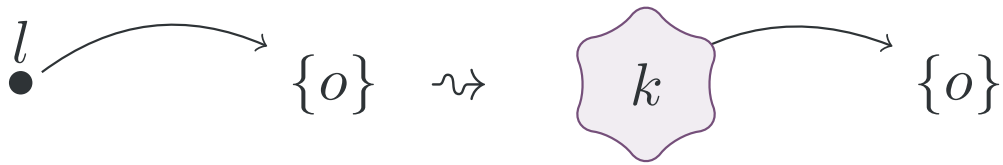
$$(k \to k, l \to l, l_0 \to l_0 \,|\, l_0 \mapsto \{\mathtt{f} : l,\ \_ : \boxtimes\} \star l \mapsto \{\mathtt{f} : k,\ \_ : \boxtimes\} \star k \mapsto \{\mathtt{g} : k,\ \_ : \boxtimes\})$$



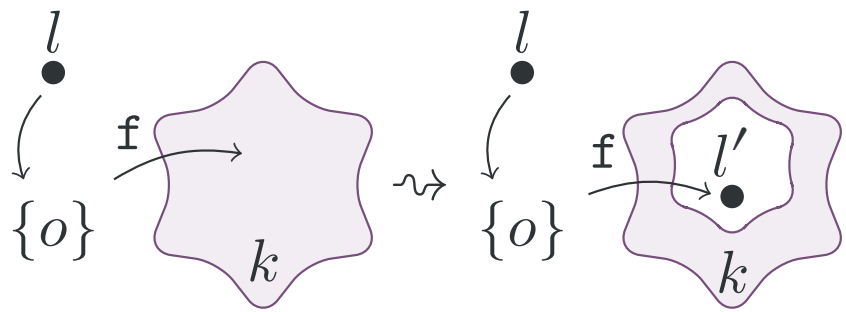
$$(k \to k', l \to k' \,|\, l_0 \mapsto \{\mathtt{f} : k',\ \_ : \boxtimes\} \star k' \mapsto \{\mathtt{f} : k' \sqcup \boxtimes,\ \mathtt{g} : k' \sqcup \boxtimes,\ \_ : \boxtimes$$

Other membrane manipulations

- A precise location l gets approximated into a summary node k,



- Summary node materialisation,



- Filter summary nodes...

Conclusion and future works

- A framework for certified abstract semantics.
- Compatible with both abstract interpretation and separation logic.

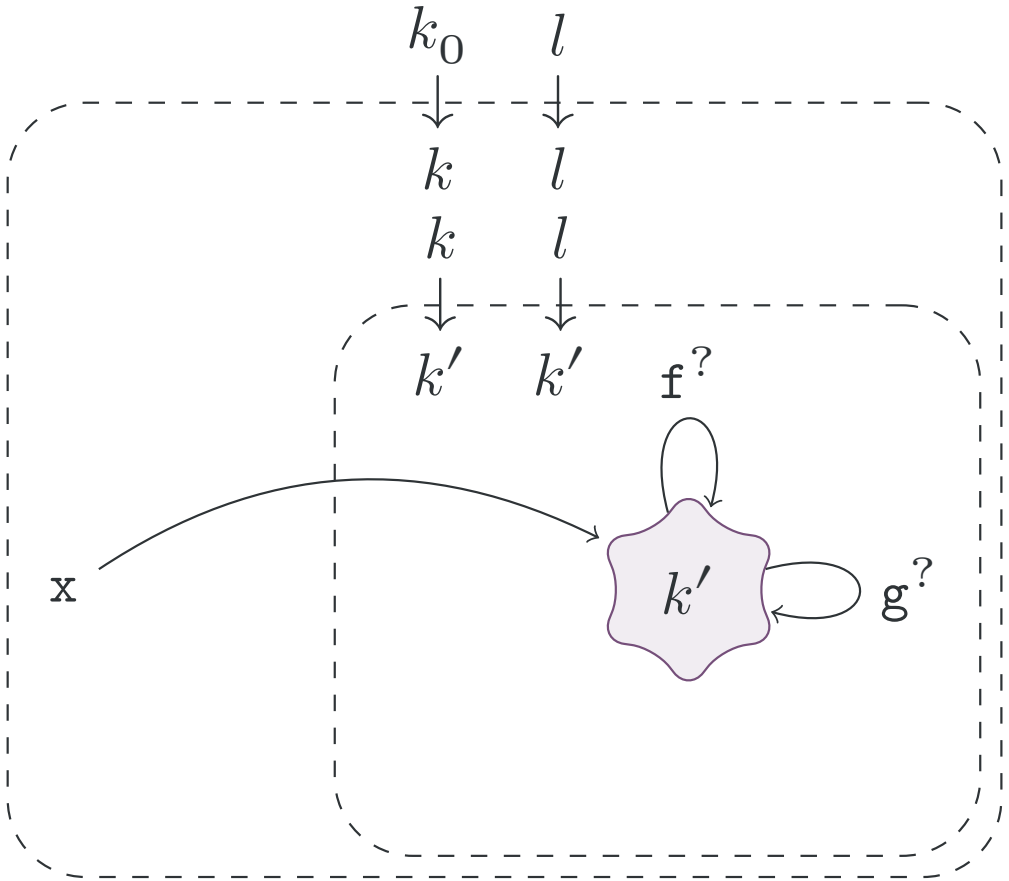
Next steps

- Coq in progress.
- Apply to JavaScript.

Thank you for listening

$$(k \to k \,|\, \mathtt{x} \doteq l)$$

$$\circledast\,(k \to k', l \to k' \,\big|\, k' \mapsto \{\mathtt{f} : k' \sqcup \boxtimes, \mathtt{g} : k' \sqcup \boxtimes, \_ : \boxtimes\})$$



Do you have questions?

1. Abstract Interpretation

2. Separation Logic

3. Adding Summary Nodes

Planches pour questions

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

Syntaxe

$$
\begin{array}{llll}
s ::= & \mathit{skip} & \mid s_1;\ s_2 & \mid \mathit{if}\, e\, s_1\, s_2 \\
& \mid \mathit{while}\, e\, s & \mid \mathit{throw} & \mid \mathtt{x} := e \\
& \mid e_1.\mathtt{f} := e_2 & \mid \mathit{delete}\, e.\mathtt{f} &
\end{array}
$$

$$
\begin{array}{llll}
e ::= & n \in \mathbb{Z} & \mid\ ? & \mid \mathtt{x} \in \mathit{Var} \qquad \mid \mathit{nil} \\
& \mid \mathit{alloc}() & \mid e.\mathtt{f} & \mid \mathtt{f}\ \mathtt{in}\ e \qquad \mid \neg\ e \\
& \mid = e_1\ e_2 & \mid \bowtie\ e_1\ e_2 & (\bowtie \in \{>, +, -\})
\end{array}
$$

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

Syntax With Membranes

$$\phi ::= \mathit{emp} \mid \phi_1 \star \phi_2 \mid \mathtt{x} \doteq v^\sharp \mid h \mapsto \{o\}$$
$$h ::= l \mid k$$

$$o ::= \mathtt{f} : v^\sharp, o \mid \_ : v^\sharp$$

$$m \in \mathfrak{M} ::= h \to h_1 + \ldots + h_n \mid \nu h$$

$$\Phi ::= (M \mid \phi) \qquad M \in \mathcal{P}_f(\mathfrak{M})$$

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

~~The~~ One Problem of JavaScript

An Introduction to Separation Logic, by John C. Reynolds

The soundness of the frame rule is surprisingly sensitive to the semantics of our programming language. Suppose, for example, we changed the behavior of deallocation, so that, instead of causing a memory fault, `dispose x` behaved like `skip` when the value of `x` was not in the domain of the heap. Then $\{emp\}\texttt{dispose x}\{emp\}$ would be valid, and the frame rule could be used to infer $\{emp \star \texttt{x} \doteq 10\}\texttt{dispose x}\{emp \star \texttt{x} \doteq 10\}$. Then, since *emp* is a neutral element for $\star$, we would have $\{\texttt{x} \doteq 10\}\texttt{dispose x}\{\texttt{x} \doteq 10\}$, which is patently false.

- JavaScript's `delete` does exactly this.

~~The~~ One Problem of JavaScript

An Introduction to Separation Logic, by John C. Reynolds

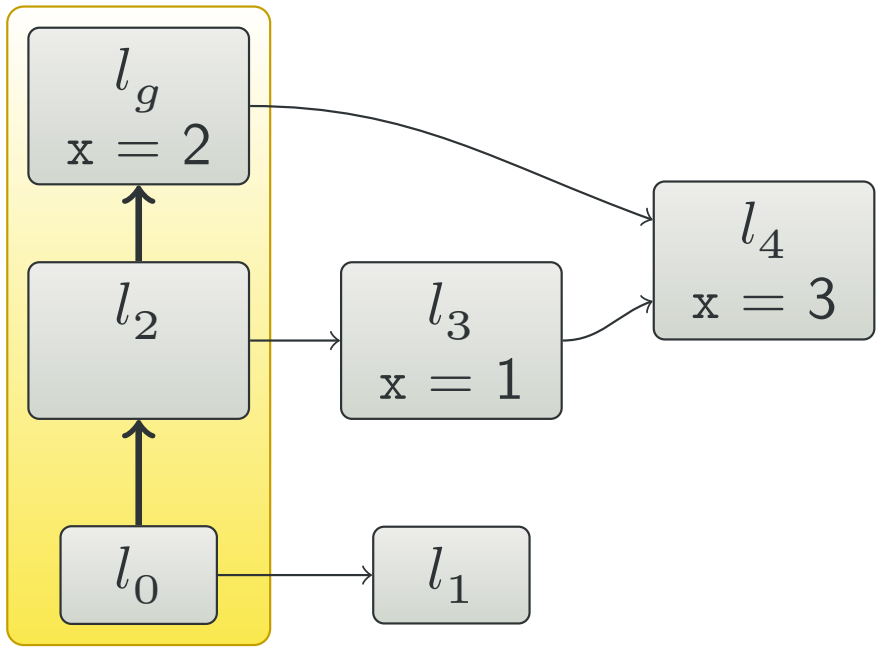
$$\dfrac{\dfrac{\dfrac{}{\{emp\}\texttt{dispose x}\{emp\}}\ \textsc{AlreadyDisposed}}{\{emp \star \texttt{x} \doteq 10\}\texttt{dispose x}\{emp \star \texttt{x} \doteq 10\}}\ \textsc{Frame}}{\{\texttt{x} \doteq 10\}\texttt{dispose x}\{\texttt{x} \doteq 10\}}\ \textsc{Rewrite}$$

- JavaScript's `delete` does exactly this.

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

About the Dark Matter "⊠"

Why do we need it?



lexical environment

It is very important to track the absence of properties in objects.

About the Dark Matter "$\boxtimes$"

Can you tell the difference between these formulae?

- emp
- $True$
- $l \mapsto \{\mathtt{f} : \boxtimes, \_ : \boxtimes\}$
- $l \mapsto \{\mathtt{f} : \boxtimes, \_ : \boxtimes\} \star \phi$
- $l \mapsto \{\mathtt{f} : \bot, \_ : \boxtimes\}$

About the Dark Matter "$\boxtimes$"

Can you tell the difference between these formulae?

- *emp*
- *True*
- $l \mapsto \{\mathtt{f} : \boxtimes, \_ : \boxtimes\}$
- $l \mapsto \{\mathtt{f} : \boxtimes, \_ : \boxtimes\} \star \phi$
- $l \mapsto \{\mathtt{f} : \bot, \_ : \boxtimes\} = \mathit{False}$

$$\gamma\,(\mathit{emp}) = \gamma\,(l \mapsto \{\mathtt{f} : \boxtimes, \_ : \boxtimes\})$$

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

CPP'15 in a Nutshell: Abstract Rules

$$\textsc{IfTrue}\ \frac{E, s_1 \Downarrow E'}{(v, E)\,, \mathit{if}\ s_1\ s_2 \Downarrow E'}\ v \in \mathbb{Z}^\star \qquad \textsc{IfFalse}\ \frac{E, s_2 \Downarrow E'}{(v, E)\,, \mathit{if}\ s_1\ s_2 \Downarrow E'}\ v \in \{0\}$$

$$\textsc{IfTrue}\ \frac{E^\sharp, s_1 \Downarrow^\sharp E'^\sharp}{\left(v^\sharp, E^\sharp\right), \mathit{if}\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}\ \gamma\left(v^\sharp\right) \cap \mathbb{Z}^\star \neq \emptyset \qquad \textsc{IfFalse}\ \frac{E^\sharp, s_2 \Downarrow^\sharp E'^\sharp}{\left(v^\sharp, E^\sharp\right), \mathit{if}\ s_1\ s_2 \Downarrow^\sharp E'^\sharp}\ \gamma\left(v^\sharp\right) \cap \{0\} \neq \emptyset$$

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

Updating the Frame Rule

The $\star$ for membraned formulae $\Phi = (M \mid \phi)$

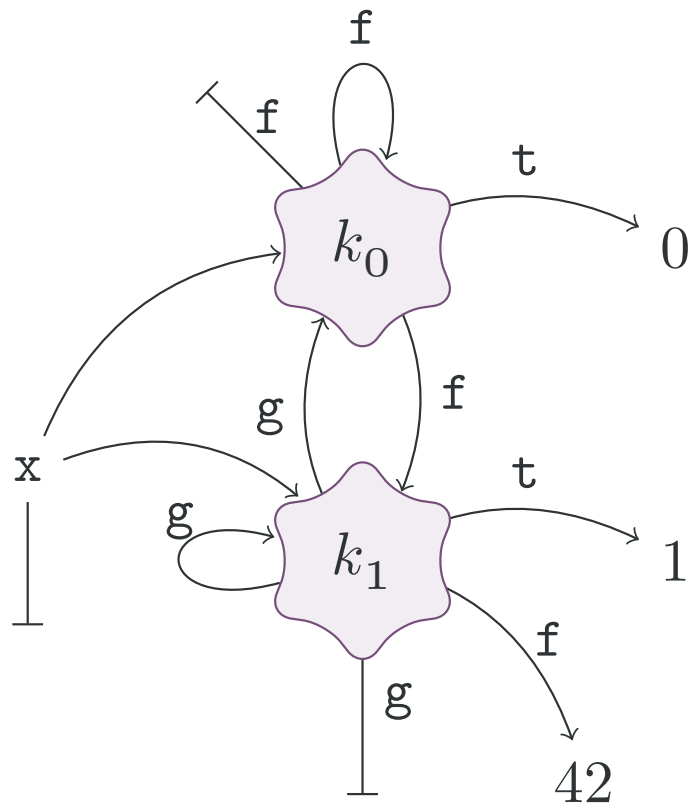
$$(M \mid \phi) \circledast (M_c \mid \phi_c) = \langle\!\langle\, (M_c \mid (M \mid \phi) \star \phi_c) \,\rangle\!\rangle$$

$$\textsc{Frame}\ \frac{\Phi, s \Downarrow^{\sharp} \Phi'}{\Phi \circledast \Phi_c, s \Downarrow^{\sharp} \Phi' \circledast \Phi_c}$$

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

Example

```
x := nil;
while?(
    t := x;
    x := alloc();
    if?(
        x.t := 0;
        x.f := t
    )(
        x.t := 1;
        x.g := t;
        x.f := 42
    )
);
while x ≠ nil(
    if x.t (x := x.g)(x := x.f)
)
```



We can express a loop invariant without adding new constructions!

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

Definitions of $\mathcal{F}$ and $aapply^\sharp$

$$\mathcal{F}^\sharp\left(\Downarrow_0^\sharp\right) = \left\{ (\sigma^\sharp, t, r^\sharp) \middle| \begin{array}{l} \forall n.\, t = term\,(n) \Rightarrow cond_n^\sharp\left(\sigma^\sharp\right) \\ \quad \Rightarrow \exists \sigma'^\sharp, r'^\sharp.\, glue_n\left(\sigma'^\sharp, r'^\sharp, \sigma^\sharp, r^\sharp\right) \\ \quad \wedge (\sigma'^\sharp, t, r'^\sharp) \in apply_n^\sharp\left(\Downarrow_0^\sharp\right) \end{array} \right\}$$

$$\begin{array}{l} apply_n^\sharp\,(\Downarrow_0) := \\ \left| \begin{array}{l} \textit{match } rule^\sharp\,(n) \textit{ with} \\ \quad | \quad Ax\left(ax^\sharp\right) \Rightarrow \left\{ \left(\sigma^\sharp, term\,(n)\,, r^\sharp\right) \middle| ax^\sharp\left(\sigma^\sharp\right) = r^\sharp \right\} \\ \quad | \quad R_1\left(up^\sharp\right) \Rightarrow \left\{ \left(\sigma^\sharp, term\,(i)\,, r^\sharp\right) \middle| \begin{array}{r} up^\sharp\left(\sigma^\sharp\right) = \sigma'^\sharp \\ \wedge \sigma'^\sharp, term_1 \Downarrow_0 r^\sharp \end{array} \right\} \\ \quad | \quad R_2\left(up^\sharp, next^\sharp\right) \Rightarrow \left\{ \left(\sigma^\sharp, term\,(n)\, r^\sharp\right) \middle| \begin{array}{r} up^\sharp\left(\sigma^\sharp\right) = \sigma'^\sharp \\ \wedge \sigma'^\sharp, term_1\,(n) \Downarrow_0 r_1^\sharp \\ \wedge next^\sharp\left(\sigma^\sharp, r_1^\sharp\right) = \sigma''^\sharp \\ \wedge \sigma''^\sharp, term_2\,(n) \Downarrow_0 r^\sharp \end{array} \right\} \end{array} \right. \end{array}$$

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

Correctness of the glue in Coq

```
1  forall n asigma' ar' asigma ar,
2    glue n asigma' ar' asigma ar ->
3    (forall sigma' r' (A : apply sem n sigma' r'),
4      gst asigma' sigma' ->
5      cond n sigma' ->
6      apply_depth A < k ->
7      gres ar' r') ->
8    forall sigma r (A : apply sem n sigma r),
9      gst asigma sigma ->
10     cond n sigma ->
11     apply_depth A < k ->
12     gres ar r.
```

◂ Back

- ▸ CPP'15
- ▸ Syntax
- ▸ Membrane syntax
- ▸ Dark matter
- ▸ Why the Dark matter?
- ▸ Abstract rules
- ▸ A more complex example
- ▸ $apply^\sharp$
- ▸ Glue's correctness

1. Abstract Interpretation

2. Separation Logic

3. Adding Summary Nodes