Modular Semantics and Metatheory for LLVM IR

Dissertation Defense

University of Pennsylvania

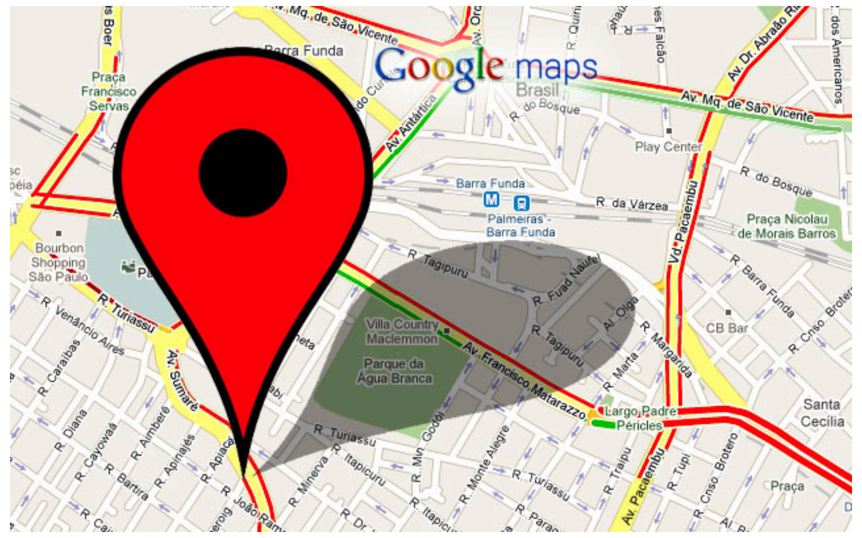
Irene Yoon
11/27/2023

Software correctness

A need for reliable, high-assurance software

<u>Levels of assurance</u>

printers



navigation system

space vehicles

text editors

train schedule display

radiation therapy

nuclear power

Low (low consequence) High (very safety-critical)

Formal verification

Mathematically proved absence of bugs

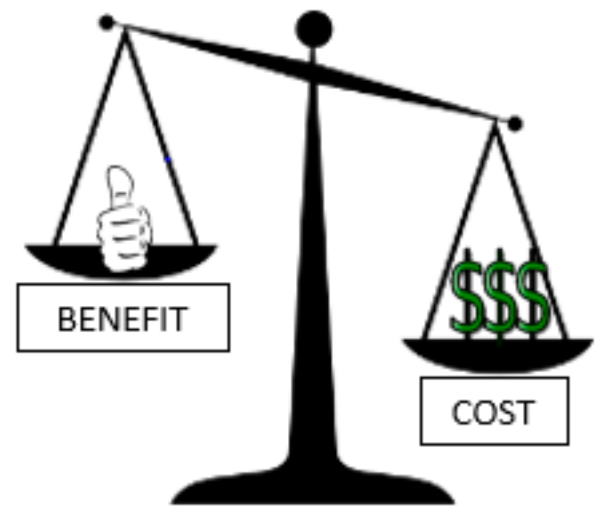
- Interactive theorem proving (e.g. Coq, Isabelle/HOL, LEAN)

(1) Formal specifications: mathematical specification about the behavior of a program

(2) Certified software: can extract a certified program from the proof of formal specification

- Success stories: CompCert C Compiler [Leroy et al.], sel4 OS kernel [Klein et al.]
  - CSmith Random testing finds bug in eleven C compilers [Regehr et al 2011], except for CompCert: errors only found in unverified parts

The cost-benefit of formal verification



- Notoriously labor- and expertise- intensive
- Alternatives: lightweight verification (lower cost, lower assurance)
  - property-based testing, refinement types, model checking, etc.

For certain high-assurance, complex software (such as compilers), formal verification is necessary to guarantee the absence of bugs

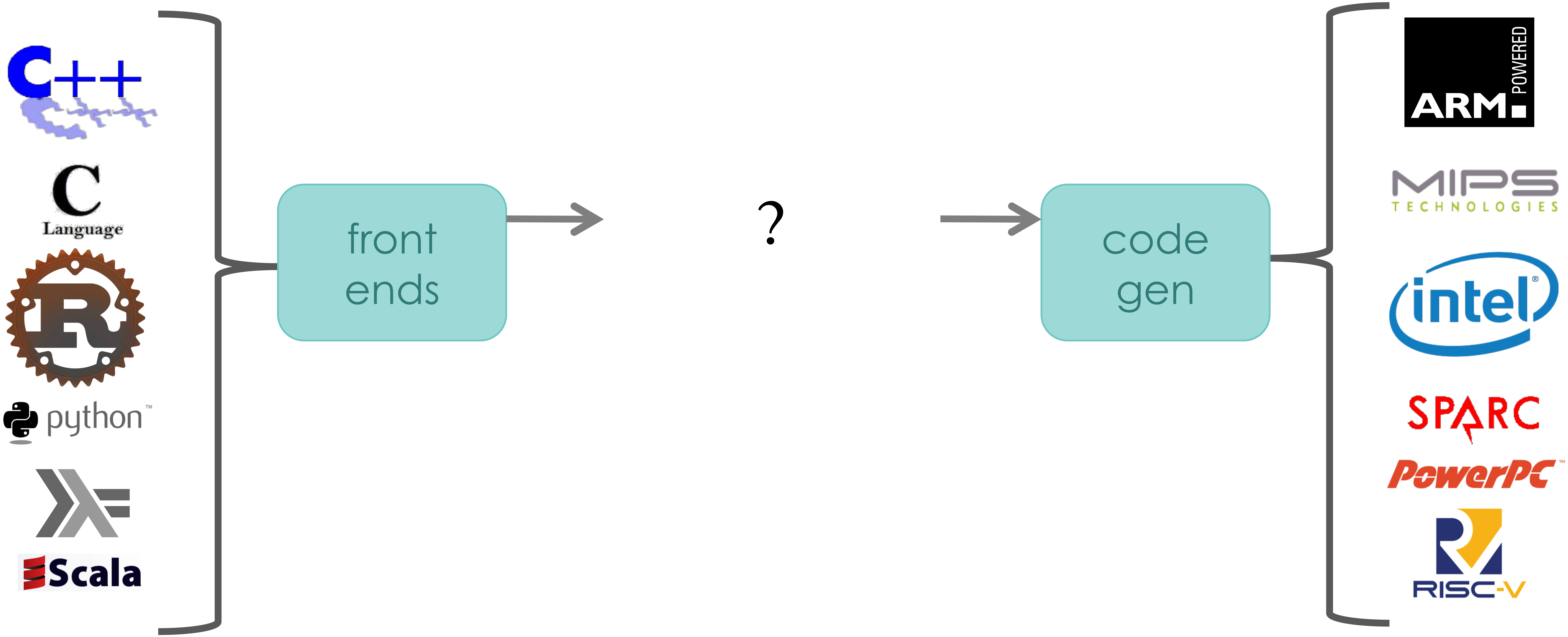
(1) What should we verify?

Target an infrastructure that is <u>common ground</u> for as many software as possible

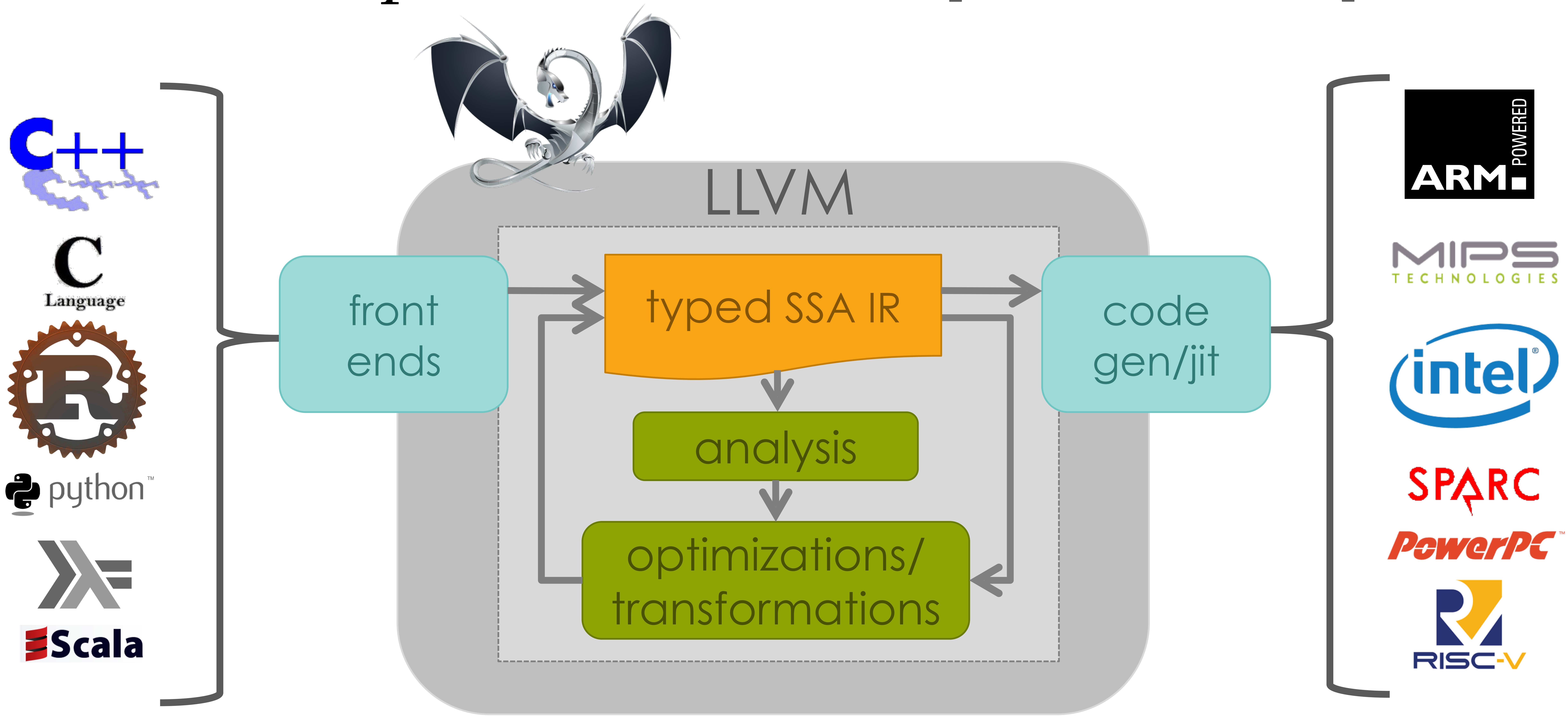
(2) How do we verify it?

<u>Modularity</u> is good: specify and verify a system piece-by-piece, reuse when possible!

A modular and reusable infrastructure for compiler pipelines



LLVM Compiler Infrastructure [Lattner et al.]



Modular semantics and metatheory for LLVM IR

Two key components of formally verified software

(1) Formal semantics

Characterizes the meaning of the constructs of the programming language in which the software is written

(2) Program logic

Formal logic for expressing and proving a program specification

<u>Modular semantics and metatheory for LLVM IR</u>

Part I. Semantics
Part II. General meta-theory
Part III. Program logic

Bird's eye view

Contributions and overview

Part I. Semantics

VIR, A modular and executable semantics for LLVM IR

[Zakowski, Beck, **Yoon**, Zaichuk, Zaliva, Zdancewic] ICFP 2021

Part II. Metatheory

eqmR, Formal reasoning about layered monadic interpreters

[**Yoon**, Zakowski, Zdancewic] ICFP 2022

Part III. Program Logic

Velliris, A relational separation logic for LLVM IR

[**Yoon**, Spies, Gäher, Song, Dreyer, Zdancewic] In submission, 2023

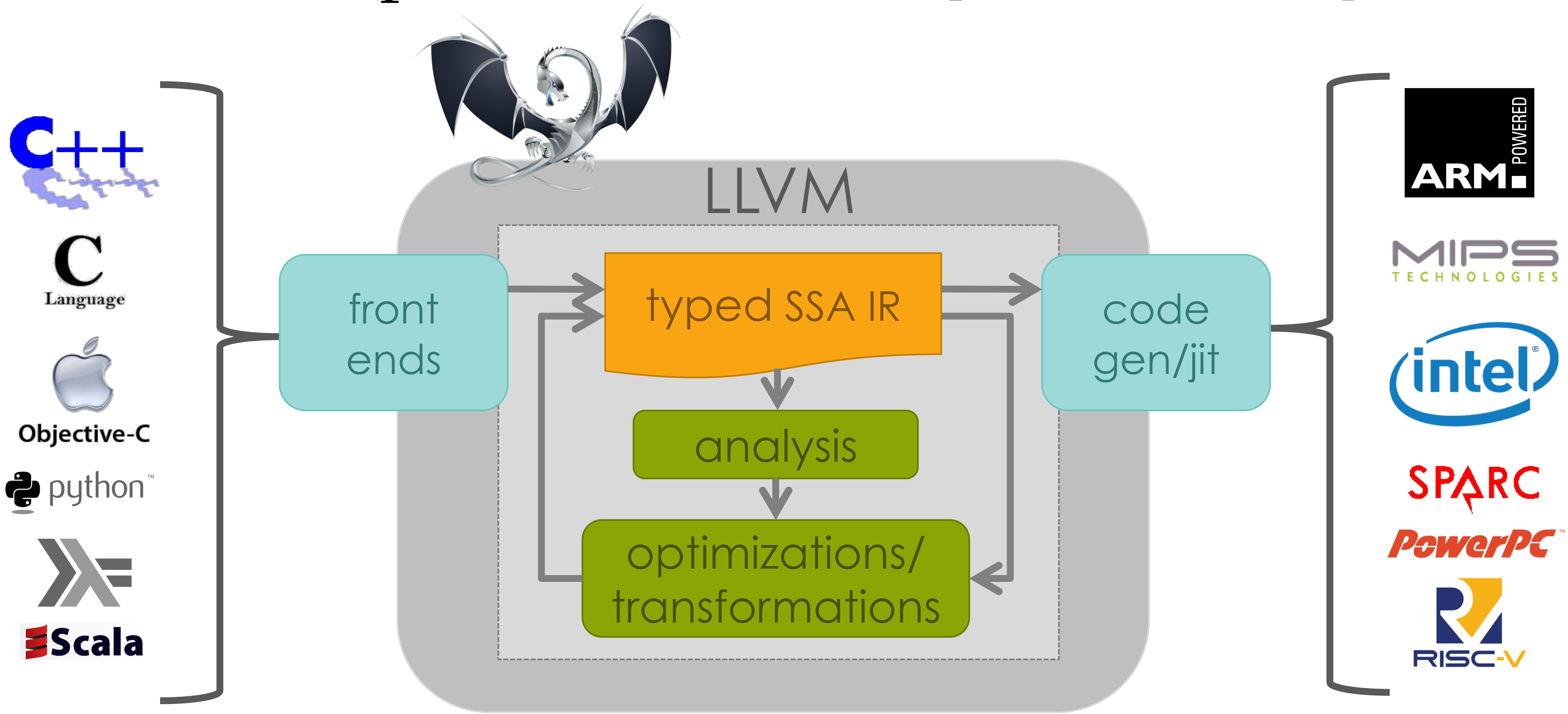
* : all results mechanized in the Coq Proof Assistant

Part I: A Modular Semantics for LLVM IR

joint work with

Zakowski, Beck, Zaichuk, Zaliva, Zdancewic

LLVM Compiler Infrastructure [Lattner et al.]



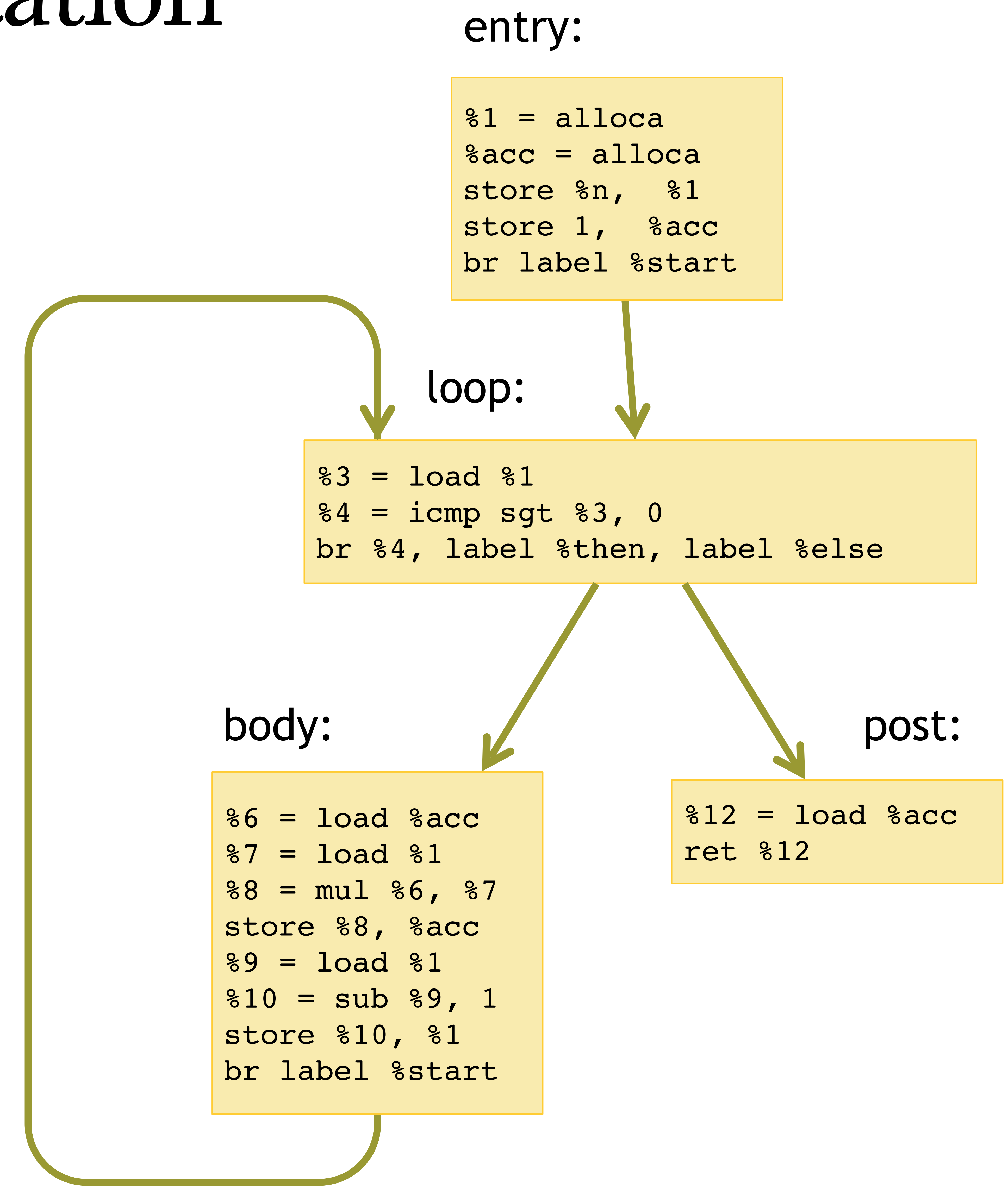
LLVM Intermediate Representation

- LLVM IR
  - Control-flow Graphs:
    - Labeled blocks
    - Straight-line Code
    - Block Terminators
    - Static Single Assignment Form (phi-nodes)
- Types:
  - i64 $\Rightarrow$ 64-bit integers
  - i64* $\Rightarrow$ pointer

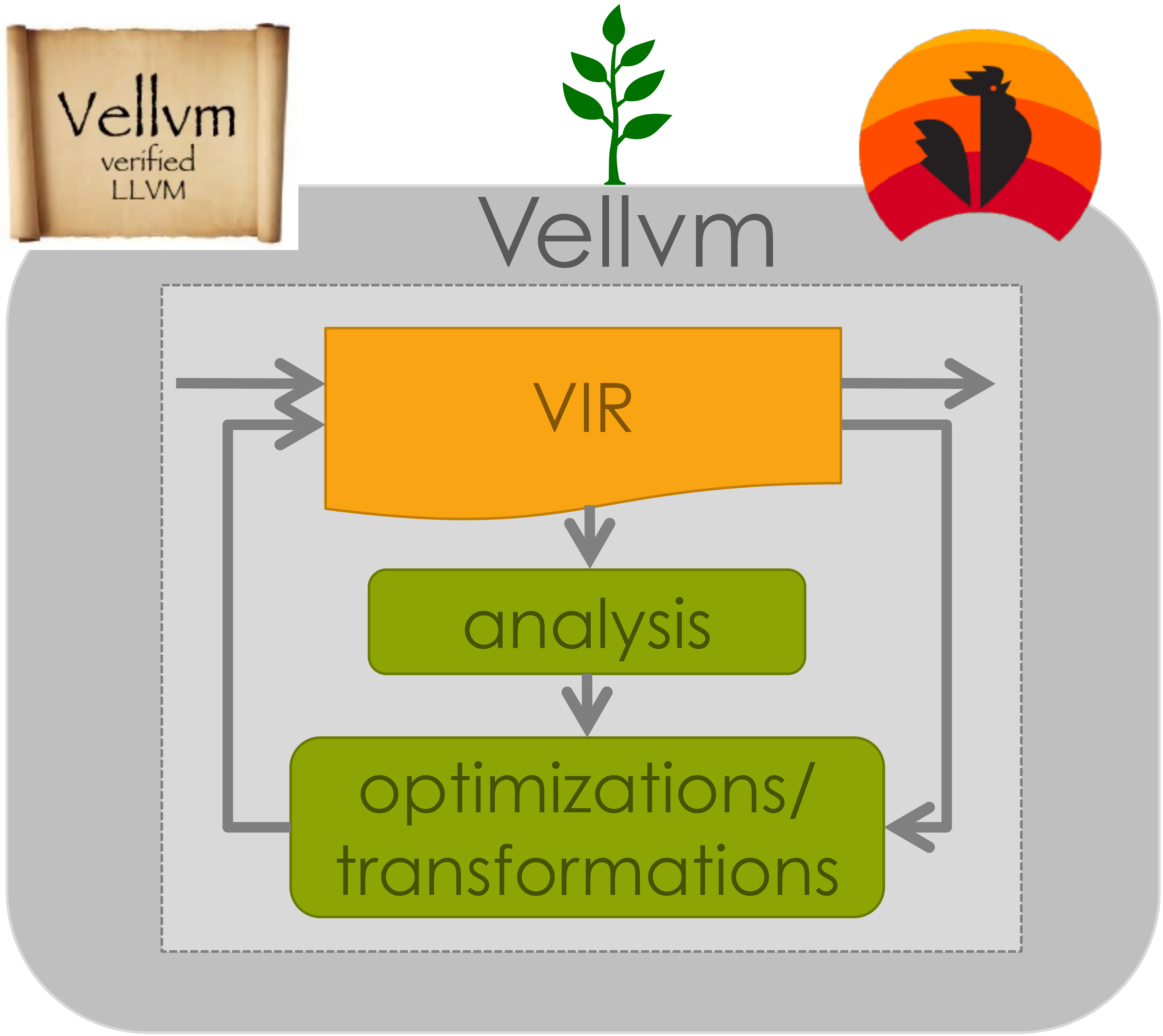
SSA $\approx$ functional program [Appel 1998]

+

- Undefined values / poison
- Effects
  - structured heap load/store
  - system calls (I/O)
- Types & Memory Layout
  - structured, recursive types
  - type-directed projection



The Vellvm Project ("Vellvm 1.0")



[Zhao and Zdancewic - CPP 2012]
Verified computation of dominators

[Zhao et al. - POPL 2012]
Formal semantics of IR + verified SoftBound

[Zhao et al. - POPL 2013]
Verification of (v)mem2reg!

https://github.com/vellvm/vellvm-legacy

A success, but monolithic

$$\boxed{G \vdash pc, mem \to pc', mem'}$$

Vellvm 2.0: A redesign of Vellvm

A Coq formal semantics for a large, sequential fragment of LLVM IR coming with:

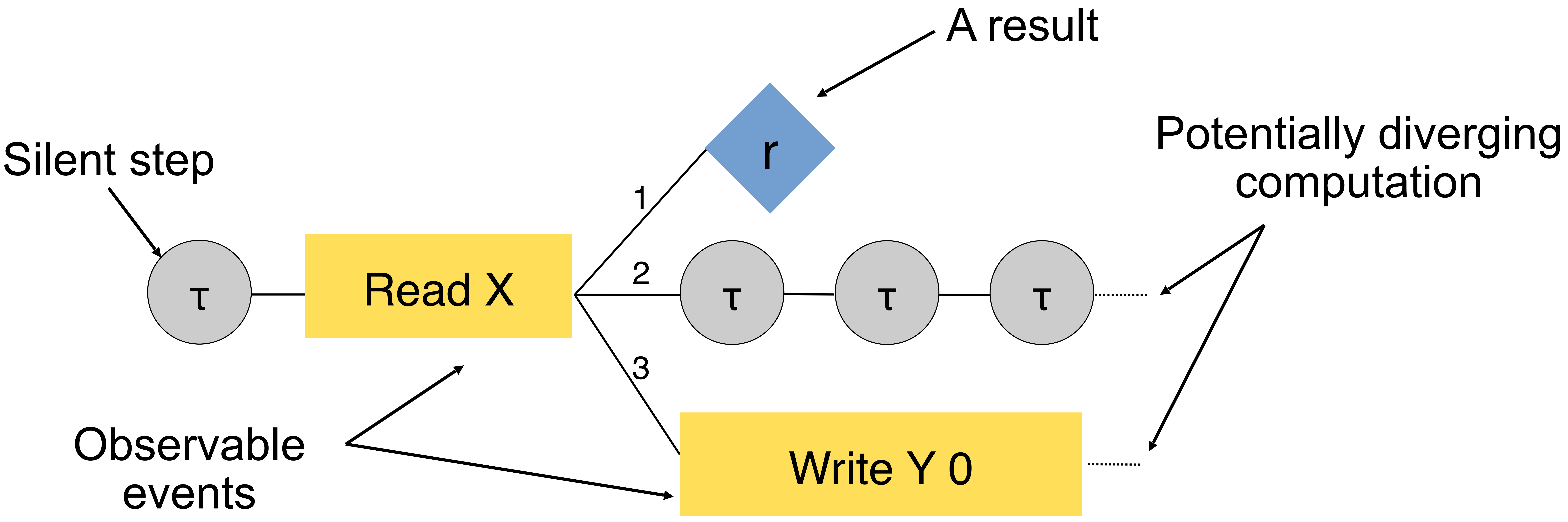
VIR: an Interaction Tree [Xia et al.] based semantics for LLVM IR

- a certified interpreter
- modularity (extensible events) and compositionality (denotational semantics)
- a rich equational theory
- an equational style to refinement proofs

("Vellvm, revamped")

Interaction Trees

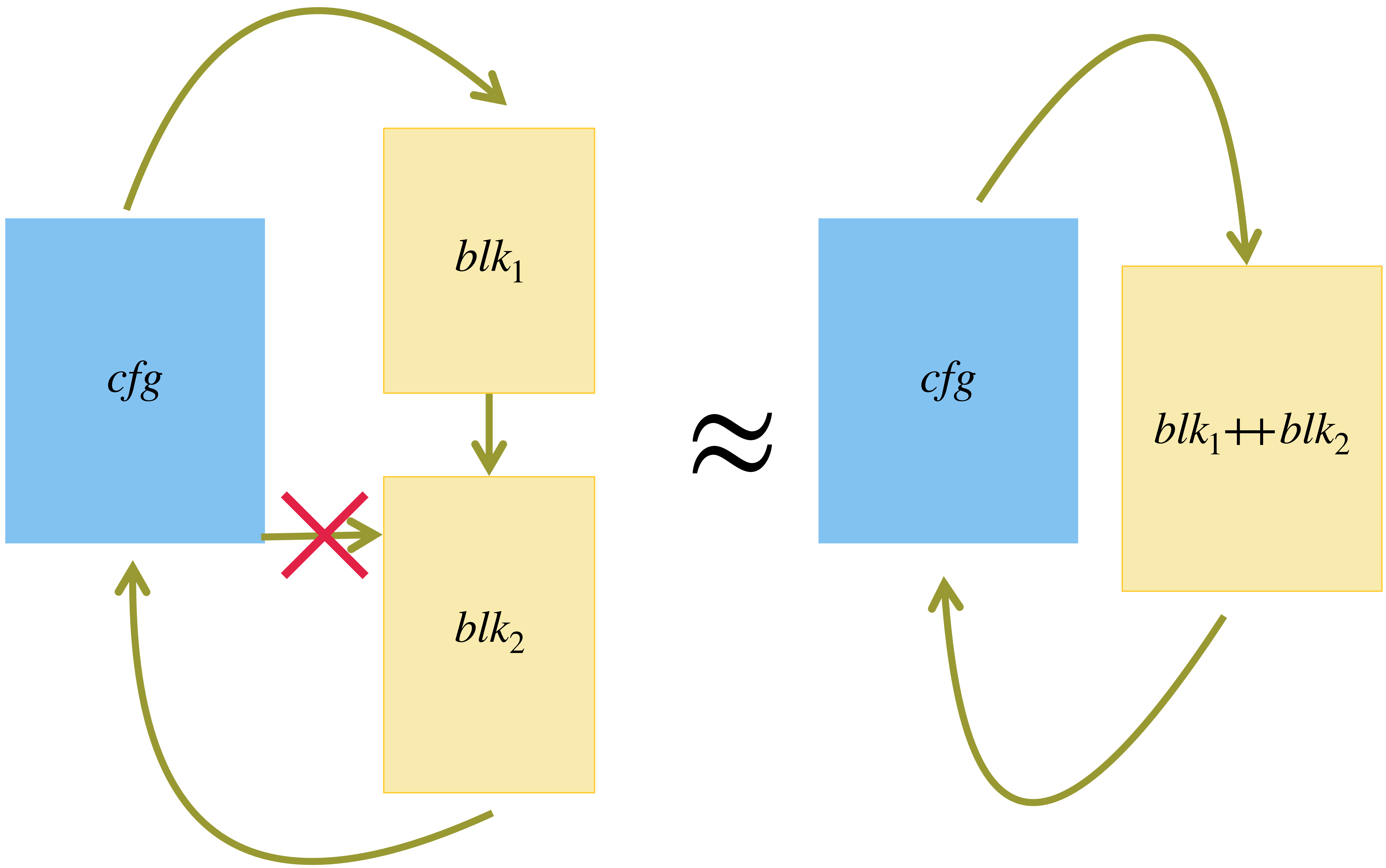
Modular and executable semantics



- Event-based semantics (modular)
- Denotational semantics (compositional)
- Certified interpreter via extraction (executable)

Benefits of Interaction-Tree based reasoning

Reasoning about control-flow



- Proof of block-merging optimization
- Reasoning about composing control-flow operators is simple
- Benefit

  Proof involves reasoning only about control flow, not other side-effects (e.g. state, exception..)

Reference Interpreter: Executability

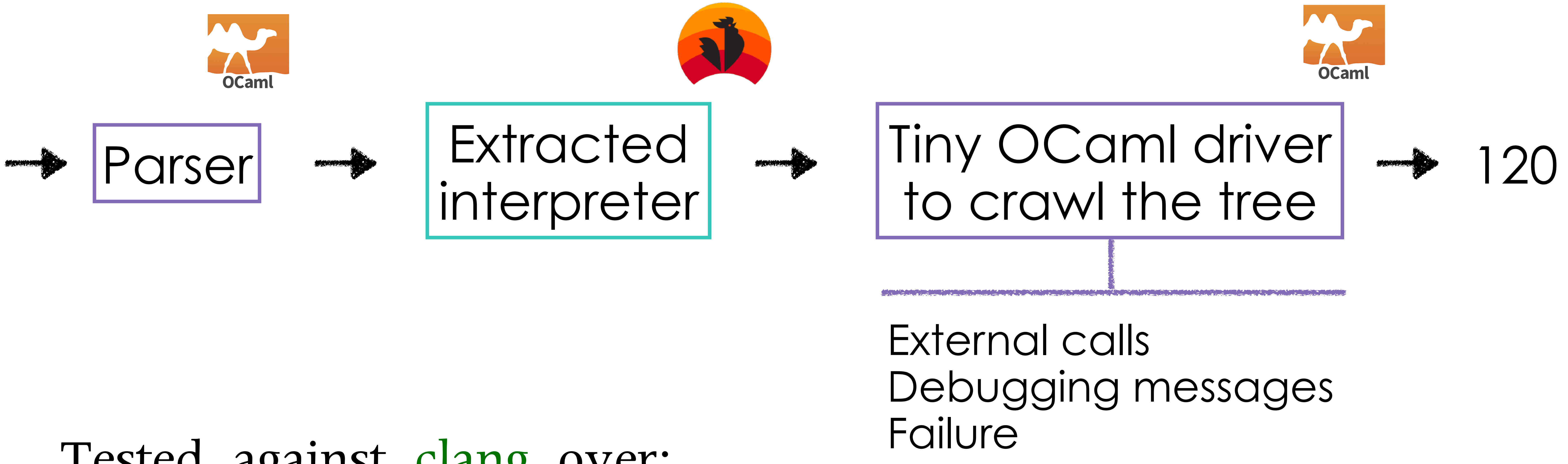
```
define i64 @factorial(i64 %n) {
  %1 = alloca i64
  %acc = alloca i64
  store i64 %n, i64* %1
  store i64 1, i64* %acc
  br label %start

start:
  %2 = load i64, i64* %1
  %3 = icmp sgt i64 %2, 0
  br i1 %3, label %then, label %end

then:
  %4 = load i64, i64* %acc
  %5 = load i64, i64* %1
  %6 = mul i64 %4, %5
  store i64 %6, i64* %acc
  %7 = load i64, i64* %1
  %8 = sub i64 %7, 1
  store i64 %8, i64* %1
  br label %start

end:
  %9 = load i64, i64* %acc
  ret i64 %9
}

define i64 @main(i64 %argc, i8** %arcv) {
  %1 = alloca i64
  store i64 0, i64* %1
  %2 = call i64 @factorial(i64 5)
  ret i64 %2
}
```



Tested against clang over:

- A collection of unit tests
- A handful of significant programs from the HELIX frontend
- Experiments over randomly generated programs using QuickChick

Part II: A Layered Equational Framework

joint work with
Zakowski, Zdancewic

Scaling up monadic interpreters

VIR

$\downarrow$ *structural representation*

Level 0	`itree VellvmE` $\mathcal{V}_u$
	$\downarrow$ ***intrinsics***
Level 1	`itree` E_0 $\mathcal{V}_u$
	$\downarrow$ ***global environment***
Level 2	$\texttt{stateT}_{Env_G}$ `(itree` E_1`)` $\mathcal{V}_u$
	$\downarrow$ ***local environment***
Level 3	$\texttt{stateT}_{Env_L * Env_G}$ `(itree` E_2`)` $\mathcal{V}_u$
	$\downarrow$ ***memory model***
Level 4	$\texttt{stateT}_{Mem * Env_L * Env_G}$ `(itree` E_3`)` $\mathcal{V}_u$

propositional model

$\texttt{stateT}_{Mem * Env_L * Env_G}$ `(itree` E_4`)` $\mathcal{V}_u \to \mathbb{P}$

$\downarrow$ model $\texttt{undef}_\tau$

$\texttt{stateT}_{Mem * Env_L * Env_G}$ `(itree` E_5`)` $\mathcal{V}_u \to \mathbb{P}$

$\ni$

executable interpreter

$\texttt{stateT}_{Mem * Env_L * Env_G}$ `(itree` E_4`)` $\mathcal{V}_u$

$\downarrow$ interpret $\texttt{undef}_\tau = 0_\tau$

$\texttt{stateT}_{Mem * Env_L * Env_G}$ `(itree` E_5`)` $\mathcal{V}_u$

Free monads and monadic interpreters

Using monadic interpreters to model programming languages

```
let x = 1 in
y := 100
…
```

```
itree (MemoryE +' LocalE) Nat
```

Layered equivalences

Lifting equivalences and structural laws across interpretation

$\equiv$ *syntactic equivalence*

$\approx$ *free-monadic equivalence*

$\approx^{\text{heap}}$

equivalence up to resulting environment

$\approx^{\text{heap, env}}$

- There exists certain properties that is specific to the interpretation (e.g. effect-specific laws about local environment, heap)
- However, there is "redundant" theory for structural properties that is preserved throughout interpretation
- eqmR: formalization of metatheory which is preserved throughout a generic notion of monadic interpretation, i.e. monad laws, iterative laws, lifting relations across interpretation

In practice

Imp2Asm compiler correctness

```
OLD PROOF:
  intros.
  unfold interp_asm, interp_map.
       cbn.
  repeat rewrite interp_bind.
  repeat rewrite interp_state_bind.
```

Same structural rule, Separate proof obligation for each layer

```
  repeat rewrite bind_bind.
  eapply eutt_clo_bind; [
       reflexivity | ..].
  intros. rewrite H.
  destruct u2 as [g' [l' x]].
  reflexivity.
```

Redundant boilerplate

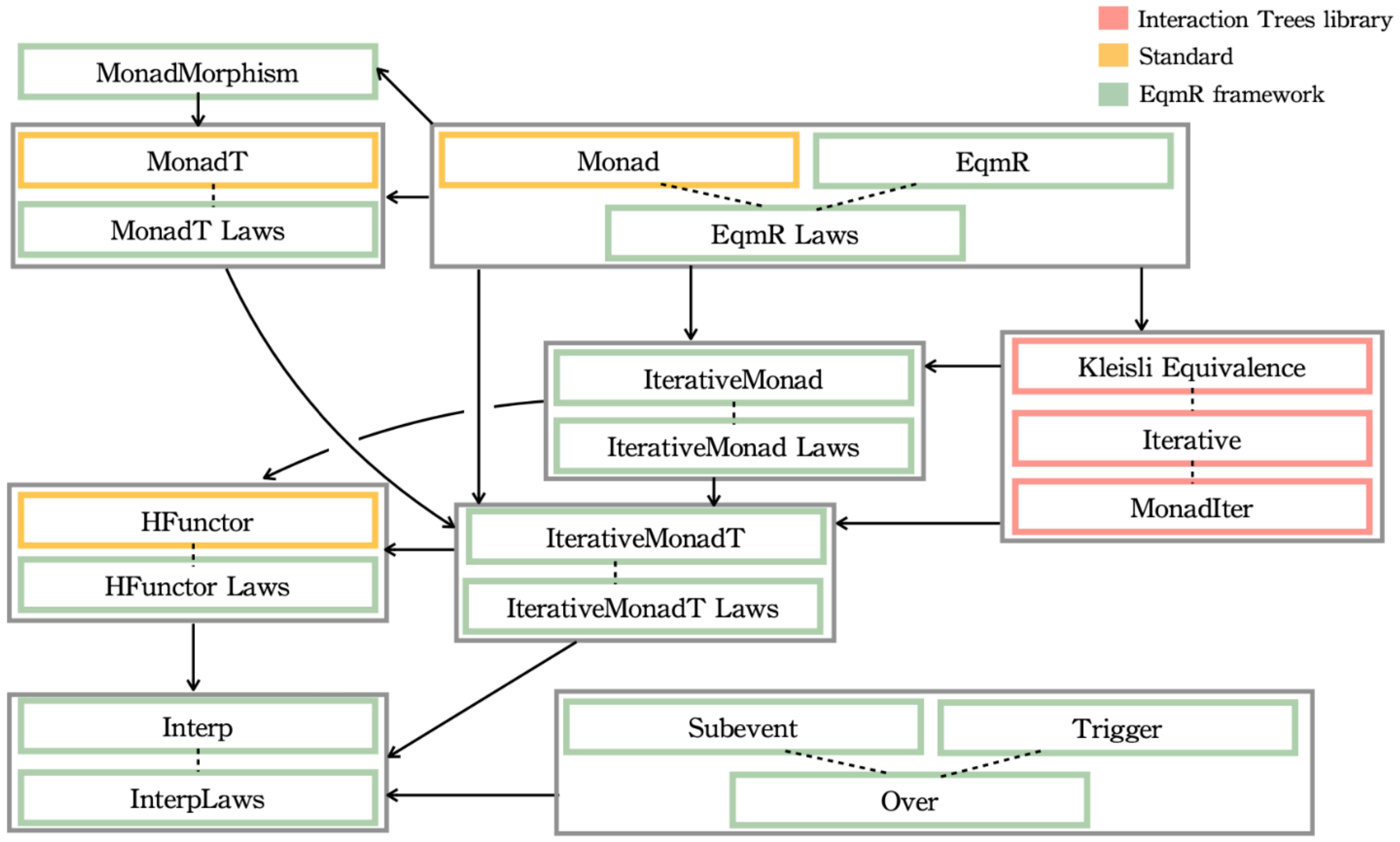
```
NEW PROOF:
  intros; unfold interp_asm.
  do 2 ibind.
```

Apply the structural rules which are preserved throughout interpretation!

Contributions

Extensible metatheory for extensible effects

- Metatheory to reduce boilerplate in formal reasoning about layered interpreters
- Relational Hoare reasoning and lifting of equivalences across interpretation
    - Generalization of automatic injection of handlers
    - Interpretable monads respecting theory of iteration
- Coq library extending InteractionTree framework
- Case study (Imp2Asm compiler correctness)



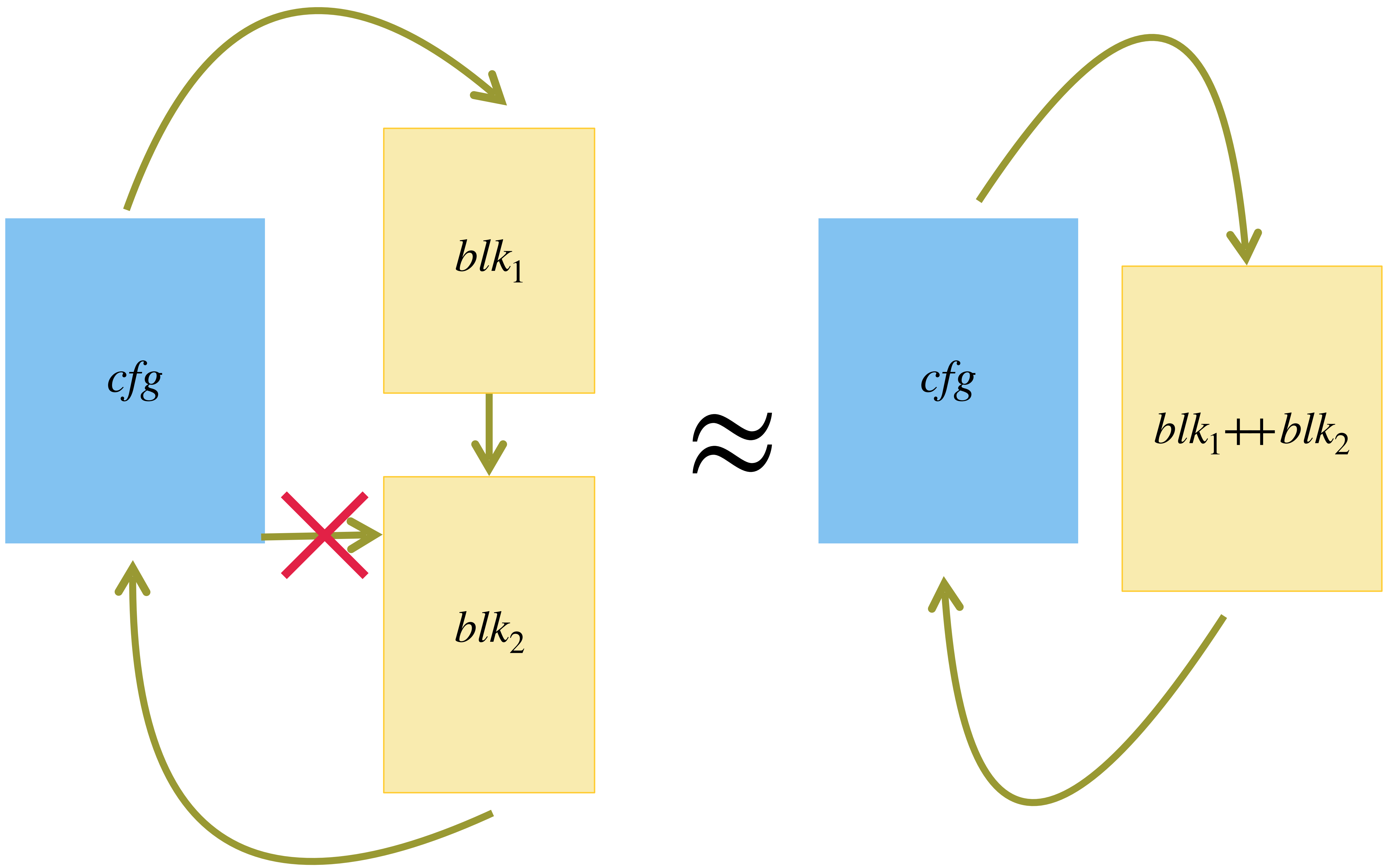
Part III: A Relational Separation Logic Framework

joint work with

Spies, Gäher, Song, Dreyer, Zdancewic

Benefits of Interaction-Tree based reasoning

Reasoning about control-flow



- Proof of block-merging optimization
- Reasoning about composing control-flow operators is simple
- Benefit

  Proof involves reasoning only about control flow, not other side-effects (e.g. state, exception..)

The need for a state-aware program logic

Stateful reasoning in VIR

- Relational reasoning on ITree-based semantics

$$e_t \approx_R e_s$$

Two programs e_t and e_s
(1) Both terminate and satisfy the postcondition R over the result of the computation, OR
(2) Both diverge in simulation with each other

- Stateful Hoare-style reasoning

Given a stateful interpretation function $[\![-]\!]$: `itree (E +' F) A → stateT S (itree F) A`

$$\{\mathscr{P}\}e_t \approx e_s\{\mathscr{Q}\} := \forall \sigma_t, \sigma_s \,.\, \mathscr{P}(\sigma_t, \sigma_s) \Rightarrow [\![e_t]\!]\sigma_t \ \approx_{\mathscr{Q}}\ [\![e_s]\!]\sigma_s$$

- Localize reasoning about state using separation logic

Separation Logic and Iris

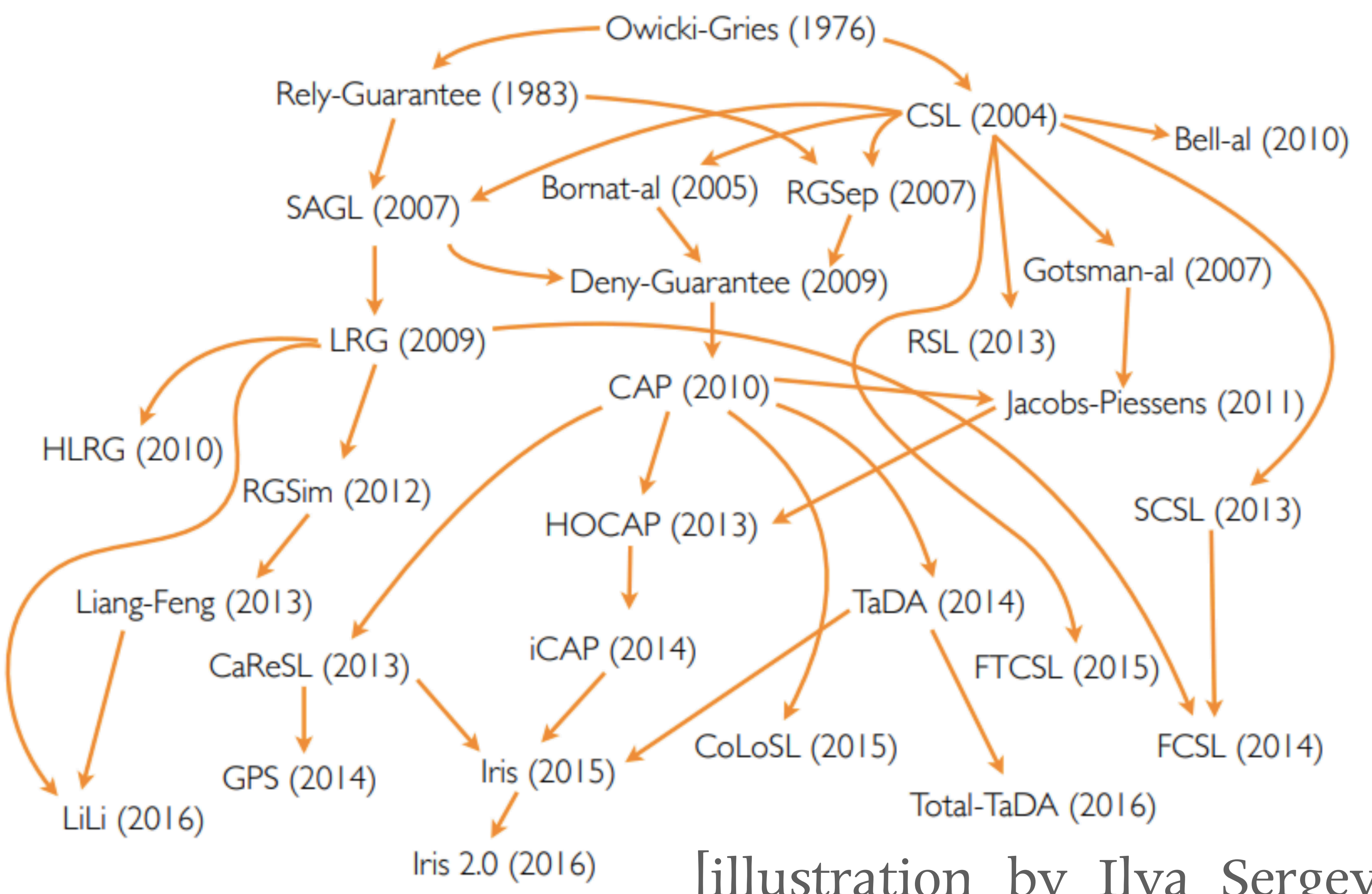
Local stateful reasoning for all!

- Separation logic [O. Hearn et al.]

$$P * Q \qquad \frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}$$

- Iris [Jung et al.]: a higher-order concurrent separation logic framework
    - Highly reusable and influential in consolidating variants of separation logics
    - Used for various other realistic semantics (RustBelt, RefinedC, Iris-WASM, etc).

<u>The genealogy of separation logics</u>



[illustration by Ilya Sergey]

Solution: Marry benefits of VIR and Iris

Semantics*

Program logic

Vellvm
verified
LLVM

VIR + Iris = Velliris

Relational separation logic for LLVM IR!

Related

Simuliris: A Separation Logic Framework for Verifying Concurrent Program Optimizations [Gäher et al.]

Modular Denotational Semantics for Effects with Guarded Interaction Trees [Frumin et al.]

Motivating example: loop invariant code motion

```c
void increment(int *n);
int get_int (int *x) {
  int *n; int i = 0; n = &i;
  while (*n < *x)  { increment(n); }
  return *n;
}
```

```c
void increment(int *n);
int get_int_opt (int *x) {
  int *n; int i = 0;
  n = &i; int y = *x;
  while (*n < y) { increment(n); }
  return *n;
}
```

- LLVM optimizations (1) reorder (or modify/remove) memory-related instructions, and (2) often make certain assumptions about external calls while doing so
- By adding an annotation at the generated LLVM IR for the C code above, one can specify that the function only accesses memory through its arguments

```
declare i32 @increment(i32*) argmemonly
```

" function can only affect memory accessible by the arguments passed on to the function "

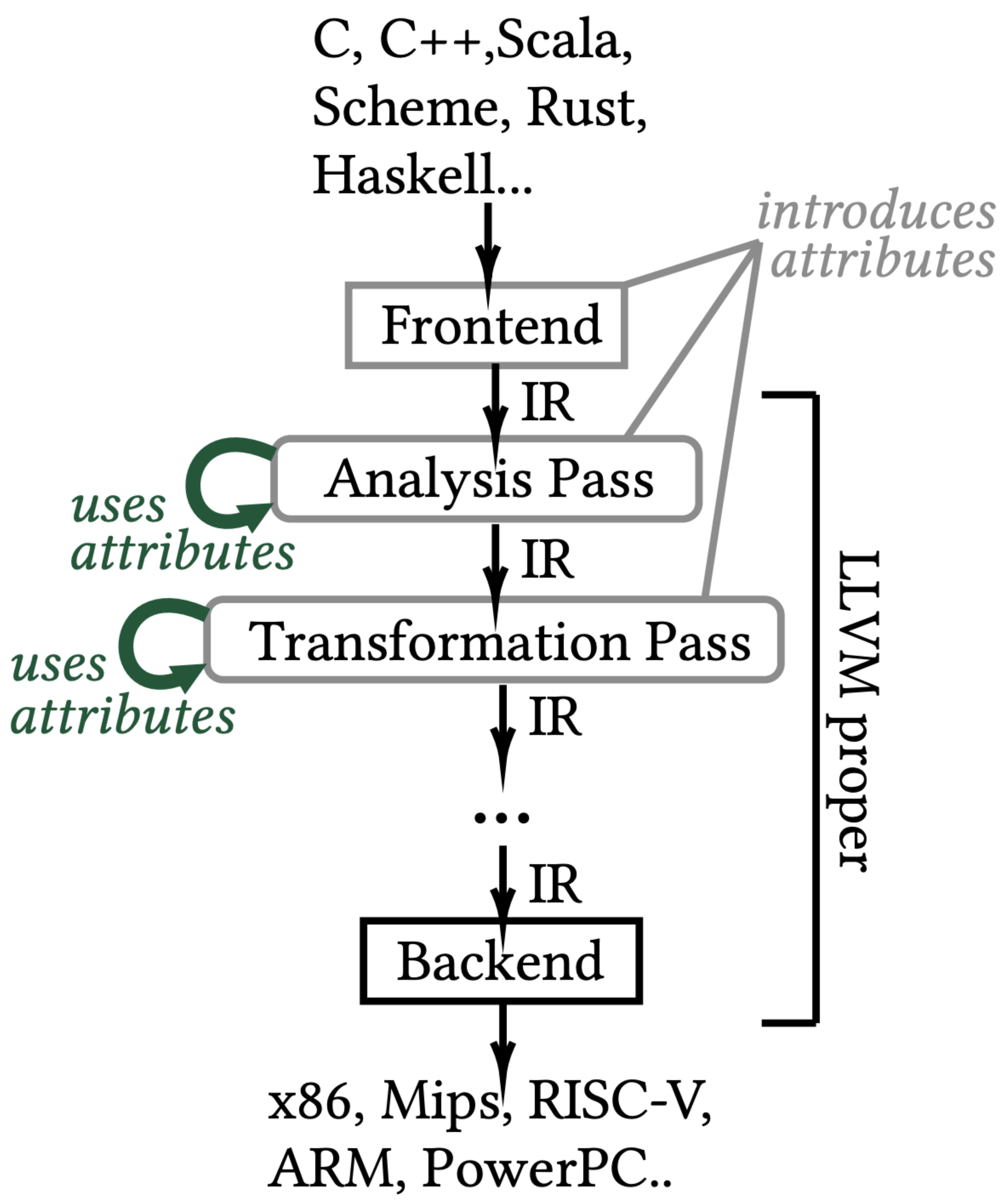
Memory attributes in LLVM IR

- LLVM optimization and analysis passes often use memory attributes, lightweight specifications about how a function may affect memory

```
define void @f(i32*) readonly argmemonly { ... }
```

" function f only reads from arguments passed on to the function "

- Logical interpretation of memory attributes using permission-based ownership
- Can reason about reordering across calls and transformations that take advantage of memory attributes



(Typical) Recipe to using Iris

(1) Ingredient: a small-step semantics

Given a small-step semantics, a Hoare triple can be derived via the typical weakest precondition model* of Iris

(technically, a Banach guarded fixpoint)*

(2) Ingredient: an abstract view on state (ghost theory) using separation logic resources

Iris has a notion of resource algebras and generic constructions of resource algebras suitable for read-only map, permission-based ownership, etc.

Building an Iris framework for VIR

Typical recipe

(1) Ingredient: a small-step semantics

Given a small-step semantics, a Hoare triple can be derived via the typical weakest precondition model* of Iris

(technically, a Banach guarded fixpoint)*

(2) Ingredient: an abstract view on state (ghost theory) using separation logic resources

Building an Iris framework for VIR

Typical recipe

(1) Ingredient: a small-step semantics

Given a small-step semantics, a Hoare triple can be derived via the typical weakest precondition model* of Iris

(technically, a Banach guarded fixpoint)*

(2) Ingredient: an abstract view on state (ghost theory) using separation logic resources

What we have (and need)

(1) Ingredient: ITree-based semantics

A new weakest precondition model* of Iris for stateful ITrees

(technically, a Knaster-Tarski mixed fixpoint)*

(2) Ingredient: A ghost theory for VIR resources

Contributions

Velliris: A relational separation logic framework for LLVM IR

- A relational, coinductive weakest precondition model of Iris which supports a monadic semantics based on the Interaction Trees framework
- A relational separation logic and ghost theory for VIR resources
- Logical interpretation for memory-relevant attributes
- Formalization and proof of contextual refinement
  - Reasoning principles over iteration and mutual recursion
- Case study: collection of simple examples and proof of simple loop invariant code motion algorithm

Summary overview

Part I

VIR, A modular and executable semantics for LLVM IR

Part II

eqmR, Formal reasoning about layered monadic interpreters

Part III

Velliris, A relational separation logic for LLVM IR

Future Work

- Connecting to a back-end (or Rust/C front-end)
- Extensive case study (e.g. verification of realistic optimization algorithm)
- Support for concurrency & relaxed memory model

Thank you!

* : all results mechanized in the Coq Proof Assistant

Extra slides

Quick excerpt of logical relation

(full definition on the thesis..)

$$\mathsf{Inv}^{(i_t, i_s)}_{(C, A_t, A_s)} \triangleq \exists args_t, args_s.\ \mathsf{FrameRes}^{\mathsf{tgt}}_{i_t}(A_t, \overrightarrow{\pi_1}\ args_t) * \mathsf{FrameRes}^{\mathsf{src}}_{i_s}(A_s, \overrightarrow{\pi_1}\ args_s) * \mathsf{checkout}(C) *$$

$$\Big(\mathop{\ast}_{(l_t, v_t);(l_s, v_s) \in args_t; args_s} \langle \mathsf{l_t} := v_t \rangle^{\mathsf{tgt}}_{i_t} * \langle \mathsf{l_s} := v_s \rangle^{\mathsf{src}}_{i_s} * l_t = l_s * \mathcal{V}_{\mathsf{U}}(v_t, v_s)\Big) *$$

$$\Big(\mathop{\ast}_{v_t; v_s \in A_t, A_s} \mathcal{V}_{\mathsf{U}}(v_t, v_s) * (v_t, v_s) \notin C\Big)$$

$$e_t \ \preceq^{exp}_{\log\ (A_t, A_s, C)} \ e_s \triangleq \forall i_t, i_s.\ \mathsf{Inv}^{(i_t, i_s)}_{(C, A_t, A_s)} \mathrel{-\!\!*} [\![e_t]\!]^{\uparrow \mathsf{exp}}_{\mathsf{expr}} \preceq [\![e_s]\!]^{\uparrow \mathsf{exp}}_{\mathsf{expr}} \ \{\lambda v_t, v_s.\, \mathcal{V}_{\mathsf{U}}(v_t, v_s) * \mathsf{Inv}^{(i_t, i_s)}_{(C, A_t, A_s)}\}$$

$$\iota_t \ \preceq^{instr}_{\log\ (A_t, A_s, C)} \ \iota_s \triangleq \forall i_t, i_s.\ \mathsf{Inv}^{(i_t, i_s)}_{(C, A_t, A_s)} \mathrel{-\!\!*} [\![\iota_t]\!]^{\uparrow \mathsf{instr}}_{\mathsf{instr}} \preceq [\![\iota_s]\!]^{\uparrow \mathsf{instr}}_{\mathsf{instr}} \ \{\exists A'_t, A'_s.\mathsf{Inv}^{(i_t, i_s)}_{(C, A_t @ A'_t, A_s @ A'_s)}\}$$

$$o_t \ \preceq^{ocfg}_{\log\ (A_t, A_s, C)} \ o_s \triangleq \forall i_t, i_s.\ \mathsf{Inv}^{(i_t, i_s)}_{(C, A_t, A_s)} \mathrel{-\!\!*} [\![o_t]\!]^{\uparrow \mathsf{instr}}_{\mathsf{ocfg}} \preceq [\![o_s]\!]^{\uparrow \mathsf{instr}}_{\mathsf{ocfg}}$$

$$\{\lambda v_t, v_s.\ ((\exists id_t, id_s. v_t = \mathtt{inl}\ id_t * v_s = \mathtt{inl}\ id_s * id_t = id_s) \vee$$

$$(\exists b_t, b_s. v_t = \mathtt{inr}\ b_t * v_s = \mathtt{inr}\ b_s * \mathcal{V}_{\mathsf{U}}(b_t, b_s)) * \exists A'_t, A'_s.\mathsf{Inv}^{(i_t, i_s)}_{(C, A_t @ A'_t, A_s @ A'_s)}\}$$

Contextual refinement

(full definition in the document)

Logical relations, continued.

$$f_t \ \preceq^{fun}_{\mathsf{log}\ (A_t, A_s, C)} f_s \triangleq \forall i_t, i_s.\ (|i_s| > 0 \rightarrow |i_t| > 0) \twoheadrightarrow \ast\ \mathsf{Frame_{tgt}}\ i_t \twoheadrightarrow\ast\ \mathsf{Frame_{src}}\ i_s \twoheadrightarrow\ast\ \mathsf{checkout}(C) \twoheadrightarrow\ast$$
$$(\bigstar_{v_t, v_s \in args_t, args_s} \mathcal{V}_{\mathsf{U}}(v_t, v_s)) \twoheadrightarrow\ast [\![f_t]\!]^{\uparrow \mathsf{instr}}_{\mathsf{fun}} \preceq [\![f_s]\!]^{\uparrow \mathsf{instr}}_{\mathsf{fun}} \{\lambda v_t, v_s.\ \mathcal{V}_{\mathsf{U}}(v_t, v_s) \ast \mathsf{Frame_{tgt}}\ i_t \ast \mathsf{Frame_{src}}\ i_s \ast \mathsf{checkout}(C)\}$$

$$F_t \ \preceq^{funs}_{\mathsf{log}\ (A_t, A_s, C)} F_s \triangleq \forall i, i_t, i_s, a_t, f_t, a_s, f_s. (|i_s| > 0 \rightarrow |i_t| > 0) \twoheadrightarrow\ast F_t[i] = (a_t, f_t) \twoheadrightarrow\ast F_s[i] = (a_s, f_s) \twoheadrightarrow\ast$$
$$\mathcal{V}_{\mathsf{Dyn}}(a_t, a_s) \twoheadrightarrow\ast \mathsf{Frame_{tgt}}\ i_t \twoheadrightarrow\ast \mathsf{Frame_{src}}\ i_s \twoheadrightarrow\ast \mathsf{checkout}(C) \twoheadrightarrow\ast$$
$$(\bigstar_{v_t, v_s \in args_t, args_s} \mathcal{V}_{\mathsf{U}}(v_t, v_s)) \twoheadrightarrow\ast [\![f_t]\!]^{\uparrow \mathsf{instr}}_{\mathsf{fun}} \preceq [\![f_s]\!]^{\uparrow \mathsf{instr}}_{\mathsf{fun}} \{\lambda v_t, v_s.\ \mathcal{V}_{\mathsf{U}}(v_t, v_s) \ast \mathsf{Frame_{tgt}}\ i_t \ast \mathsf{Frame_{src}}\ i_s \ast \mathsf{checkout}(C)\})$$

Contextual refinement

Definition 6.1 (Contextual refinement). $e_t \sqsubseteq_{\mathsf{ctx}} e_s := \forall C, \mathsf{wf}\ C \wedge \mathsf{wf}_{prog}([\![C[e_t]]\!]\sigma_t)([\![C[e_s]]\!]\sigma_s) \Rightarrow ([\![C[e_t]]\!]\sigma_t) \approx_{\mathcal{V}\downarrow} ([\![C[e_s]]\!]\sigma_s)$.

THEOREM 6.2 (CONTEXTUAL ADEQUACY). *Given* $e_t \preceq^{fun}_{\mathsf{log}} e_s$, *then* $e_t \sqsubseteq_{\mathsf{ctx}} e_s$.