Interaction Trees and Verified Compilation

Paolo Torrini and Benjamin Gregoire

INRIA Sophia-Antipolis

Rocqshop'25, Reykjavik 27.09.2025

Semantics

Well-established: operational semantics (CompCert, CakeML)

– weak wrt compositionality

– small-step (CompCert): not directly executable

– big-step (CakeML): need clocks to deal with divergence

Newer: Interaction Trees (ITrees)

– denotational: compositional wrt the language syntax, executable

– coinductive (OK with divergence)

– computations represented as trees, interpreted as monads

– ITrees as free monads: side effects, modularly

– facilitate coinductive reasoning:
  – relying on PACO (parameterized coinduction)
  – supporting effectively equational reasoning

Interaction Trees (simplified)

```
CoInductive itree (E: Type → Type) (V: Type) :=
      Ret (v: V) | Tau (t: itree E V)
     | Vis (A: Type) (e: E A) (k: A → itree E V).
```

Event handler (basic one, no dependencies, no transfomers):

$$h_i \ : \ \forall \{E\} \ V, \ E_i \ V \to \text{itree } E \ V$$

Monadic interpreter (folding the handler on the tree):

$$\text{Intr}_{h_i} \ : \ \forall \{E\} \ V, \ \text{itree } (E_i +' E) \ V \to \text{itree } E \ V$$

Layered interpretation:

$$t : \text{itree } (E_2 +' E_1 +' E_0) \ V \implies \text{Intr}_{h_1} \circ \text{Intr}_{h_2} \ t \ : \ \text{itree } \mathsf{E}_0 \ V$$

Jasmin

`https://github.com/jasmin-lang/jasmin`

- Low-level language for criptographic applications
- formalized in Rocq: semantics and verified compiler
- old verification using unclocked inductive big-step semantics: terminating programs only
- lifting the restriction using ITrees
- front-end made of ca 20 passes (incl. constant propagation, dead code elimination, inlining, stack allocation)
- concrete memory model

Compiler verification

- p_1, p_2 programs (resp. source and target), with $p_2 := \mathrm{Comp}\ p_1$
- $\lceil p \rceil_s$: itree $E\ S$ executable semantic interpretation of p in $s : S$
- R: here a relation between source and target states
  ($?R$ between optional ones, to take divergence into account)

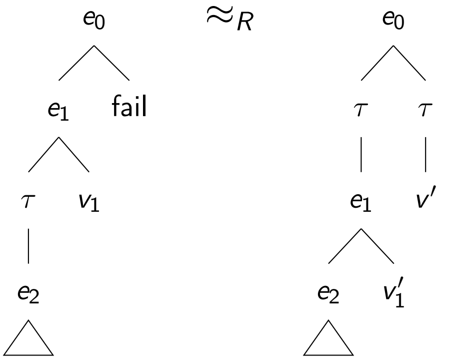
$$\begin{tikzcd} s_1 \arrow[r, dash, "R"] \arrow[d, "\lceil p_1 \rceil"] & s_2 \arrow[d, "\lceil p_2 \rceil"] \\ ?s_1' \arrow[r, dash, "?R"] & ?s_2' \end{tikzcd}$$

$$\forall s_1 s_2,\ s_1\ R\ s_2 \to \lceil p_1 \rceil_{s_1} \approx_R \lceil p_2 \rceil_{s_2}$$

- Determistic semantics: no substantial difference between forward and backward simulation
- yet difference between forward and backward reasoning (resp. inductively on the source or on the target)

Equivalence up-to-tau

R: in general, relation between values (possibly of different types)



assuming :

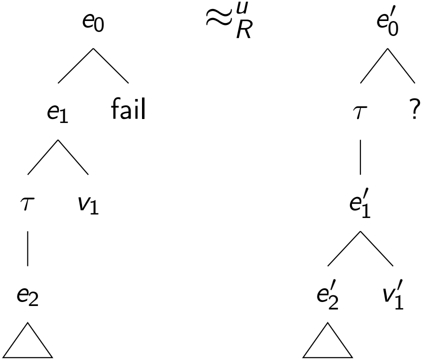
– $v_i \; R \; v'_i$

– $\forall v, k_i \; v \approx_R k'_i \; v$

Relaxing equivalence: heterogeneous events and cutoffs

Φ: precondition between events (possibly of different types)

Ψ: postcondition between event answers



assuming :

- $v_i \ R \ v'_i$
- $e_i \ \Phi \ e'_i$
- $\forall v \ v', (e_i, v) \ \Psi \ (e'_1, v')$
  $\rightarrow k_i \ v \approx^u_R k'_i \ v'$
- fail is a left cutoff

Equivalence up-to-cutoff

Coinductive-inductive definition, with cutoff Boolean predicates (C^l, C^r).

$$\frac{v_1 \; R \; v_2}{\mathrm{Ret}(v_1) \overset{u}{\approx} \mathrm{Ret}(v_2)}\ \mathrm{Ret} \qquad \frac{t_1 \overset{u}{\approx} t_2}{\mathrm{Tau}(t_1) \overset{u}{\approx} \mathrm{Tau}(t_2)}\ \mathrm{Tau}$$

$$\frac{e_1 \; \Phi \; e_2 \qquad \forall v_1 \; v_2.\; (e_1, v_1) \; \Psi \; (e_2, v_2) \implies k_1(v_1) \overset{u}{\approx} k_2(v_2)}{\mathrm{Vis}(e_1, k_1) \overset{u}{\approx} \mathrm{Vis}(e_2, k_2)}\ \mathrm{Vis}$$

$$\frac{C^l(e_1)}{\mathrm{Vis}(e_1, k_1) \overset{u}{\approx} t_2}\ \mathrm{Cut}_l \qquad \frac{C^r(e_2)}{t_1 \overset{u}{\approx} \mathrm{Vis}(e_2, k_2)}\ \mathrm{Cut}_r$$

$$\frac{t_1 \overset{u}{\approx} t_2}{\mathrm{Tau}(t_1) \overset{u}{\approx} t_2}\ \mathrm{Tau}_l \qquad \frac{t_1 \overset{u}{\approx} t_2}{t_1 \overset{u}{\approx} \mathrm{Tau}(t_2)}\ \mathrm{Tau}_r$$

Jasmin semantics

Failure interpreter (using ExecT as error monad transformer):

$$\mathsf{Intr}_{h_F}\ :\ \forall\{E\}\ V,\ \mathsf{itree}\ (F +' E)\ V \to \mathsf{ExecT}\ (\mathsf{itree}\ E)\ V$$

Recursive call interpreter (depending on F):

$$\mathsf{Intr}_{h_{Rec}} :\ \forall\{E\}\ `\{F \sqsubseteq E\}\ V,\ \mathsf{itree}\ (\mathsf{Rec} +' E)\ V \to \mathsf{itree}\ E\ V$$

Modular interpretation:

$$t : \mathsf{itree}\ (\mathsf{Rec} +' F +' E)\ V \implies \mathsf{Intr}_{h_F} \circ \mathsf{Intr}_{h_{Rec}}\ t\ :\ \mathsf{itree}\ \mathrm{E}\ (\mathsf{Exec}\ V)$$

Semantics excerpts

```
Definition while_round IS (c1 c2: list instr) (e : expr) (s : S)
  : itree E (S + S) := ...

Definition while_loop IS (c1 c2: list instr) (e: expr) (s : S)
  : itree E S := ITree.iter (while_round c1 c2 e) s.

Fixpoint instr_sem (p : prog) (i : instr) (s : S)
  : itree E S := match i with ...
    | Cwhile c1 e c2 ⇒
       while_loop instr_sem c1 c2 e s
    | Ccall xs fn args ⇒
       vargs <- eval_exprs args s;;
       fs <- trigger (fun_call fn (mk_fun_state vargs s)) ;;
       opt_update_state xs fs s
    end.
```

Modular verification

Compiler correctness:

$$\forall s_i s_j,\ s_i\ R\ s_j \rightarrow \lceil p_i \rceil_{s_i} \approx^{u}_{R \Phi \Psi C_F C_\emptyset} \lceil p_j \rceil_{s_j}$$

In general:

- correctness of individual compiler passes proved inductively on the syntax of the source program
- single-pass proofs composed together by transitivity

$$\frac{\vdash_\Gamma \lceil p_0 \rceil_{s_o} \approx^{u}_{R_1 \Phi_1 \Psi_1 C_F C_\emptyset} \lceil p_1 \rceil_{s_1} \quad \vdash_\Gamma \lceil p_1 \rceil_{s_1} \approx^{u}_{R_2 \Phi_2 \Psi_2 C_F C_\emptyset} \lceil p_2 \rceil_{s_2}}{\vdash_\Gamma \lceil p_0 \rceil_{s_0} \approx^{u}_{(R_1 \circ R_2)(\Phi_1 \circ \Phi_2)(\Psi_1 \circ \Psi_2) C_F C_\emptyset} \lceil p_2 \rceil_{s_2}}$$

- use of relational Hoare logic

Further work: generalized transitivity

$$e_0 \; (\Phi_1 \circ \Phi_2) \; e_2 \; := \; \exists \{V_1\} \; (e_1 : E_1 \; V_1), \; e_0 \; \Phi_1 \; e_1 \wedge \; e_1 \; \Phi \; e_2$$

$$\begin{aligned}
&(e_0, v_0) \; (\Psi_1 \circ \Psi_2) \; (e_2, v_2) \; := \; \forall \{V_1\} \; (e_1 : E_1 \; V_1), \\
&\quad e_0 \; \Phi_1 \; e_1 \wedge e_1 \; \Phi_2 \; e_2 \rightarrow \\
&\quad \exists v_1, \; (e_0, v_0) \; \Psi_1 \; (e_1, v_1) \; \wedge \; (e_1, v_1) \; \Psi_2 \; (o_2, v_2)
\end{aligned}$$

$$\frac{\begin{array}{ll}
\vdash_\Gamma \forall e, \; \neg(C_1^r \; e \wedge C_2^l \; e) & \\
\vdash_\Gamma \forall e e', \; e \; \Phi_1 \; e' \wedge C_2^l \; e' \rightarrow C_1^l \; e & \\
\vdash_\Gamma \forall e e', \; e \; \Phi_2 \; e' \wedge C_1^r \; e \rightarrow C_2^r \; e' & \\
\vdash_\Gamma t \approx^u_{R_1 \Phi_1 \Psi_1 C_1^l C_1^r} t' & \vdash_\Gamma t' \approx^u_{R_2 \Phi_2 \Psi_2 C_2^l C_2^r} t''
\end{array}}{\vdash_\Gamma t \approx^u_{(R_1 \circ R_2)(\Phi_1 \circ \Phi_2)(\Psi_1 \circ \Psi_2) C_1^l C_2^r} t''}$$

Layering

Generalized layering with monad transformers (MT):

$$h_i \ : \ \forall \{E\} \ V, \ E_i \ V \to MT_i \ (\text{itree } E) \ V$$

$$\text{Intr}_{h_i} \ : \ \forall \{F\} \ \{E\} \ V, \ F \ (\text{itree } (E_i +' E)) \ V \to \ F \ (MT_i \ (\text{itree } E)) \ V$$

$$t : \text{itree } (E_2 +' E_1 +' E_0) \ R \implies \text{Intr}_{h_1} \circ \text{Intr}_{h_2} \ t \ : \ MT_2 \ (MT_1 \ (\text{itree } \mathrm{E}_0)) \ R$$

Some problems (in our experience):

- disjoint union modulo AC (minor snags)
- matching interpreters with MT (not generalized)
- universe inconsistencies popping up (panic)

Conclusions and future work

- Done: verified the Jasmin compiler front-end
- probabilistic semantics
- To do: verify the backend
  Jasmin FE $\Longrightarrow$ Linear $\Longrightarrow$ ASM
  Various backends: x86, ARM, RISC5
- compare with fuel-based inductive techniques and step-indexing
- integration with safety analysis

`https://github.com/jasmin-lang/jasmin`