Guarantees for Resource-Bounded Computations

MRG team, incl. Olha Shkaravska

Institut of Informatics, LMU

Munich, Germany

Motivation

1st of September: MOBIUS,
“Mobility, Ubiquity and Security”

15 participants, incl. IoC, Tallinn

Aim of MOBIUS: to develop the technology for establishing trust and security of global computers, using Proof-Carrying Code (PCC) paradigm

Mobile Resource Guarantees (MRG) is one of the predecessors of MOBIUS.

It develops a PCC paradigm for resource bounded computations.

Structure of this talk

What is this talk about:
MRG and what from its experience may be used in future.

- MRG - a framework for ensuring heap space safety of programs
- Break?
- Specifications of programs and their proofs in more detail

Security of Mobile Code

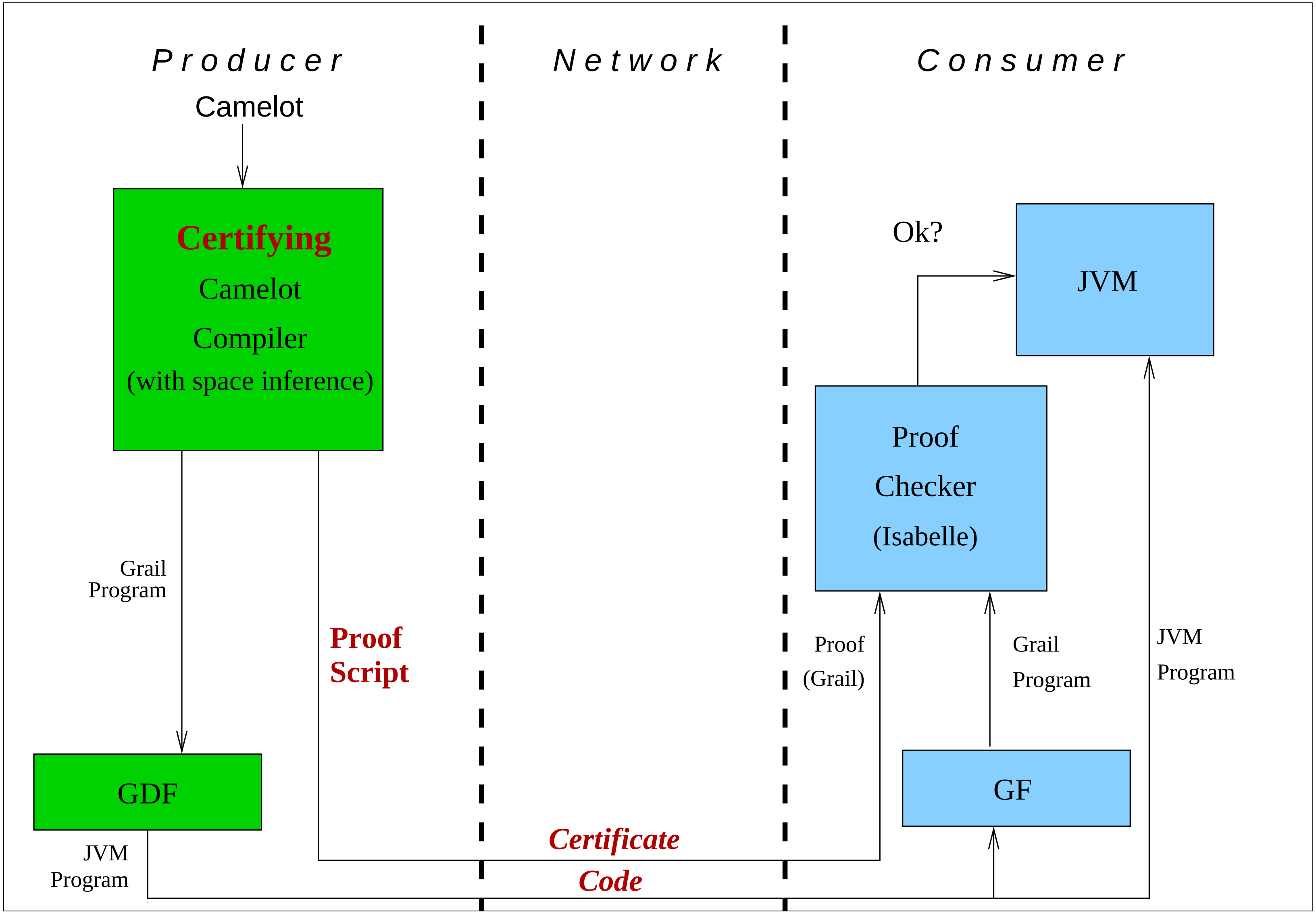
Examples of mobile code: Java-applets on the Internet, applications for smart cards, ...

Alarm: alien on the device!!!

We download the code if it is secure

- What do we mean by security in MRG?
  A program runs inside (quantitatively) restricted memory
- How to ensure security?
  - Sandboxing - too restrictive
  - Signed applets - too bureaucratic: many questions and permissions
  - PCC: mobile code is supported with the proof of its safety

PCC framework for MRG



High-level Analysis for Heap Consumption

We can infer linear heap-consumption bounds for Camelot programs:

- Given

$$f : \mathit{List}(\mathtt{Int}) \longrightarrow \mathit{List}(\mathtt{Int})$$

- Obtain a notated, with numbers, signature

$$f : \mathit{List}(\mathtt{Int},\, k),\, n \longrightarrow \mathit{List}(\mathtt{Int},\, k'),\, n'$$

For example

$\mathtt{copy} : \mathit{List}(\mathtt{Int},\, 1),\, 0 \longrightarrow \mathit{List}(\mathtt{Int},\, 0),\, 0$

or

$\mathtt{cons} : \mathtt{Int},\, \mathit{List}(\mathtt{Int},\, 0),\, 1 \longrightarrow \mathit{List}(\mathtt{Int},\, 0),\, 0$

High-level Analysis for Heap Consumption

$$f:\ \mathit{List}(\mathtt{Int},\, k),\, n \;\longrightarrow\; \mathit{List}(\mathtt{Int},\, k'),\, n'$$

- with $|x|$ be the length of an input list x
- with (at least) $k|x| + n$ of free heap units available
- the body of f terminates with a value v

then there will be (at least) $k'|v| + n'$ free heap units available after evaluation.

$$\begin{array}{l}\mathtt{append}:\ \mathit{List}(\mathtt{Int},\, 0),\, \mathit{List}(\mathtt{Int},\, 0),\, 0 \;\longrightarrow\; \mathit{List}(\mathtt{Int},\, 0),\, 0\\ \mathtt{append\_cp}:\ \mathit{List}(\mathtt{Int},\, 1),\, \mathit{List}(\mathtt{Int},\, 0),\, 0 \;\longrightarrow\; \mathit{List}(\mathtt{Int},\, 0),\, 0\end{array}$$

where `append_cp` appends 2nd arg. to the copy of the 1st one.

A derived assertion for Grail

Bounds for a given program are to be proved on the level of compiled code, i. e. Grail

To prove a statement about a code one needs

- to formalise semantics of its basic operations and structured expressions via a partial correctness assertion of the form $E, h \vdash e \rightsquigarrow h', v$
- to formalise the meaning of the statement itself
  - to define the region occupied by a list, a tree
  - to define virtual cost number, which is for $\mathtt{List}(\mathtt{Int}, 3)$ of length k is equal to $3k$.

... lots of work

A derived assertion for Grail

A resource statement for a compiled code =

The soundness for a for a high-level typing judgment (mod. compilation)

$$e : \llbracket U,\ n,\ \Gamma\ \blacktriangleright\ T,\ m \rrbracket.$$

If e terminates on a given environment E and a heap h, then when

- E satisfies the context Γ;
- the used by e variables are in U;
- n extra free heap units (+ the virtual costs of U defined by Γ) are available before evaluation

the expression e terminates with

- an output value of type T;
- m extra free heap units (+ the virtual cost of a value) are after evaluation.

Derived rules for Grail

How to prove such assertions?

We have a basic logic, that is a set of weakest conditions for Grail constructions, mirroring operational semantics

- straightforward proofs, i.e. syntactically driven application of the rules of the basic logic
  Naive! Eventually we obtain a huge HOL-predicate over heaps and environments with right-hand-side existensials!
- proofs with derived rules mirroring high-level typing rules

Derived rules for Grail

Example: the let-rules

$$\frac{\Gamma,\ n\ \vdash\ e_1\ :\ T_0,\ l \qquad (\Gamma,\ x : T_0),\ l\ \vdash\ e_2\ :\ T,\ m}{\Gamma,\ n\ \vdash\ \texttt{let x} = e_1\ \texttt{in}\ e_2\ :\ T,\ m}\ \texttt{Camelot} - \texttt{Let}$$

$$\Downarrow$$

$$\frac{e_1 :\ [\![U_1,\ n,\ \Gamma\ \blacktriangleright\ T_0,\ l]\!] \qquad e_2 :\ [\![U_2,\ l,\ (\Gamma,\ x : T_0)\ \blacktriangleright\ T,\ m]\!]}{\texttt{let x} = e_1\ \texttt{in}\ e_2 :\ [\![U_1 \uplus (U_2 \setminus \{x\}),\ n,\ \Gamma\ \blacktriangleright\ T,\ m]\!]}\ \texttt{Grail} - \texttt{Let}$$

Derived rules for Grail

Restriction:
current judgments are designed
for linear usage of variables:

- soundness is proved for a linear `let`-rule
- for that one has to prohibit sharing of arguments

although the high-level analysis is sound for a weaker (semantical) condition of benign sharing.
We cannot prove that

$$\texttt{let h} = \texttt{length x in cons}(\texttt{h}, \texttt{x}) :$$
$$[\![\{x\},\ 1,\ \mathit{List}(0)\ \blacktriangleright\ \mathit{List}(0),\ 0]\!]$$

Derived rules for Grail

- correctness of the analyser assumes benign sharing of variables in let-rule, i.e. no reachable from e_2 cells get deallocated by e_1
- to get derived assertions for proving bounds one needs to approximate benign sharing statically
- one way of approximation is linear let-rule, which is rather restrictive
- the other way is to involve usage aspects
- there is even more deep analysis, involving layered sharing...

How to prove derived assertions in a smart way?

Observations:

- numerical part of a resource aware assertion may be separated from sharing-managing assertion
- a pure resource aware assertion and various sharing-managing assertions have similar structure, so do their proofs!

It does make sense to design generic rules!

- we find a condition, say `LET`, that implies a rule `Let`
- to prove `Let` for a given assertion show that it satisfies `LET` and then instantiate the generic `Let` with the given assertion
- `LET` must have a special property allowing to combine assertions without duplicating work (later ...)

Tired?

Break ...

Typing judgment + Semantics = Derived Assertion

Does our type system work as we mean?

Derived assertions help to answer this question

$$\begin{tikzcd} \Gamma \vdash e : T \arrow[d, Rightarrow, "\textit{meaning}"] \\ e : \lambda\, E\, h\, h'\, v.\, \texttt{Spec}(E,\, h,\, h',\, v) \end{tikzcd}$$

Γ may be decorated

where e satisfies a partial correctness assertion of the form

$$E,\, h \vdash e \leadsto h',\, v$$

Example

$$\Gamma \vdash e : \texttt{List}(\texttt{Bool})$$

Let $\Gamma(x) = \texttt{List}(\texttt{Bool})$ for all $x \in \texttt{Dom}(\Gamma)$,
an inductive data structure `list` is defined in a heap.
If

- $E(x) \models^{h}_{\Gamma(x)} l$, i.e. “$l$ is a well-formed list”
- Extra-property holds: l is acyclic

then there exists a well-formed list l', s.t.

- $v \models^{h'}_{\texttt{List}(\texttt{Bool})} l'$
- l' is acyclic

A Generic Assertion

A precondition is a model relation and some property

$$\begin{aligned} Pre_{\Gamma}^{E,\,h}(X) \equiv\ & \Gamma \models^{E,\,h} X_1 \wedge \\ & Property(X) \end{aligned}$$

A postcondition states an existence of a model for the output with some property

$$\begin{aligned} Post_{T}^{h',\,v}(X) \equiv\ & \exists Y.\, v \models_{T}^{h'} Y_1 \wedge \\ & Property_{T}(X,\, Y) \end{aligned}$$

A Generic Assertion

$$\begin{tikzcd}
\Gamma \vdash e : T \arrow[d, Rightarrow, "\textit{meaning}"] \\
e : \lambda E\, h\, h'\, v.\, \forall X. Pre_{\Gamma}^{E,\, h}(X) \longrightarrow Post_{T}^{h',\, v}(X)
\end{tikzcd}$$

Γ may be decorated

Definition

$$Pre \Rightarrow Post \equiv \lambda E\, h\, h'\, v.\, \forall X. \quad Pre^{E,\, h}(X) \longrightarrow Post^{h',\, v}(X)$$

A Generic Assertion

We will consider inference rules for assertions of the form

$$e : \mathit{Pre} \ \Rightarrow \ \mathit{Post}$$

regardless if they mirror some type judgment or not.

A semantic mapping of a typing judgment onto an assertion of this form motivates our interest to such assertions, but they give just a partial case of inference systems for $e : \mathit{Pre} \ \Rightarrow \ \mathit{Post}$.

In the case of a typing judgment like $\Gamma \ \vdash \ e \ : \ T$ one defines corresponding parametric pre- and postconditions, Pre_Γ and Post_T, and instantiate with them a generic assertion $e : \mathit{Pre}_\Gamma \ \Rightarrow \ \mathit{Post}_T$

We are looking for Higher-Order Soundness Predicates

We want to justify generic proof rules, like

$$\frac{\begin{array}{c} e_1 : \; Pre_1 \;\Rightarrow\; Post_1 \\ e_2 : \; Pre_2 \;\Rightarrow\; Post_2 \end{array}}{\texttt{let x =} e_1 \;\texttt{in}\; e_2 : \; Pre \;\Rightarrow\; Post} \; \texttt{Let}$$

that is, to find a predicate

$$\lambda \, Pre \; Pre_1 \; Post_1 \; x \; Pre_2 \; Post_2 \; Post. \; \texttt{LET}$$

s. t. for all $e_1, \; e_2, \; x$ one has

$$\texttt{LET}(Pre, \; Pre_1, \; Post_1, \; x, \; Pre_2, \; Post_2, \; Post) \;\Rightarrow\; \texttt{Let}$$

and similarly for other rules.

We are looking for Higher-Order Soundness Predicates

Combinations of Type Systems

- independent type systems
  $e : Pre_1 \Rightarrow Post_1 \wedge Pre_2 \Rightarrow Post_2$, that is the proofs for $e : Pre_1 \Rightarrow Post_1$ and $e : Pre_2 \Rightarrow Post_2$ are obtained separately,
- interleaving type systems
  $e : Pre_1 \wedge Pre_2 \Rightarrow Post_1 \wedge Post_2$

The First case: obviously, we need just two collection of soundness predicates, for both systems.

The Second case: ...

We are looking for Higher-Order Soundness Predicates

$$\frac{\begin{array}{c} e_1 : \; Pre_{11} \wedge Pre_{21} \;\Rightarrow\; Post_{11} \wedge Post_{21} \\ e_2 : \; Pre_{12} \wedge Pre_{22} \;\Rightarrow\; Post_{12} \wedge Post_{22} \end{array}}{\texttt{let x} = e_1 \;\texttt{in}\; e_2 : \; Pre_1 \wedge Pre_2 \;\Rightarrow\; Post_1 \wedge Post_2}\;\texttt{Let}$$

Shall we prove the following (almost) from scratch?

$$\texttt{LET}\big(Pre_1 \wedge Pre_2,\; Pre_{11} \wedge Pre_{21},\; Post_{11} \wedge Post_{21},\; x,\; Pre_{12} \wedge Pre_{22},\; Post_{12} \wedge Post_{22},\; Post_1 \wedge Post_2\big)$$

We are looking for Higher-Order Soundness Predicates

Properties like

$$\left.\begin{array}{l}\texttt{LET}(Pre_1,\ Pre_{11},\ Post_{11},\ x,\ Pre_{12},\ Post_{12},\ Post_1)\\ \texttt{LET}(Pre_2,\ Pre_{21},\ Post_{21},\ x,\ Pre_{22},\ Post_{22},\ Post_2)\end{array}\right\}\Rightarrow$$

$$\texttt{LET}(Pre_1 \wedge Pre_2,\ Pre_{11} \wedge Pre_{21},\ Post_{11} \wedge Post_{21},\ x,\\ Pre_{12} \wedge Pre_{22},\ Post_{12} \wedge Post_{22},\ Post_1 \wedge Post_2)$$

allow to re-use soundness statements for both systems:

no need in proofs for their combinations!

How do we prove a let-rule

$$\frac{\begin{array}{ll} e_1 : \lambda\, E\, h\, h'\, v.\, \forall\, X. Pre_1^{E,\,h}(X) \longrightarrow Post_1^{h',\,v}(X) & (1) \\ e_2 : \lambda\, E\, h\, h'\, v.\, \forall\, X. Pre_2^{E,\,h}(X) \longrightarrow Post_2^{h',\,v}(X) & (2) \\ \multicolumn{2}{c}{\text{lemma}_1,\ \text{lemma}_2,\ \text{lemma}_3} \end{array}}{\texttt{let x} = e_1\ \texttt{in}\ e_2 :\ \lambda\, E\, h\, h'\, v.\, \forall\, X. Pre^{E,\,h}(X) \longrightarrow Post^{h',\,v}(X)}$$

Fix E, h, h', v and X.

$$\begin{tikzcd}
\exists Y. Pre_1^{E,\,h}(Y) \arrow[r, "(1)"] & Post_1^{h_0,\,v_0}(Y) \arrow[d, "\text{lemma}_2"'] & Post^{h',\,v}(X) \\
Pre^{E,\,h}(X) \arrow[u, "\text{lemma}_1"] \arrow[r, "\text{lemma}_2"] & \exists Z. Pre_2^{E[x:=v_0],\,h_0}(Z) \arrow[r, "(2)"] & Post_2^{h',\,v}(Z) \arrow[u, "\text{lemma}_3"']
\end{tikzcd}$$

How do we prove a let-rule

$$\lambda\ Pre\ \ Pre_1.\ \texttt{lemma}_1(Pre,\ \ Pre_1) \equiv$$
$$\forall\, E\, h.\ \forall\, X.\ Pre^{E,\, h}(X) \longrightarrow \exists\, Y. Pre_1^{E,\, h}(Y)$$

$$\lambda\ Pre\ \ Pre_1\ \ Post_1\ x\ Pre_2.\texttt{lemma}_2(Pre,\ \ Pre_1,\ \ Post_1,\ \ x,\ \ Pre_2)$$
$$\equiv$$
$$\forall\, E\, h\, h_0\, v_0.\ \forall\, X\, Y.\ Pre^{E,\, h}(X) \longrightarrow$$
$$Pre_1^{E,\, h}(Y) \longrightarrow$$
$$Post_1^{h_0,\, v}(Y) \longrightarrow$$
$$\exists\, Z. Pre_2^{E[x:=v],\, h_0}(Z)$$

How do we prove a let-rule

$$\lambda\, Pre\ \ Pre_1\ \ Post_1\ \ x\ \ Pre_2\ \ Post_2\ \ Post.$$
$$\texttt{lemma}_3(Pre,\ Pre_1,\ Post_1,\ x,\ Pre_2,\ Post_2,\ Post)$$

$$\equiv$$

$$\begin{gathered}
\forall\, E\, h\, h_0\, v_0\, h'\, v.\ \forall\, X\, Y\, Z.\ Pre^{E,\,h}(X) \longrightarrow \\
Pre_1^{E,\,h}(Y) \longrightarrow \\
Post_1^{h_0,\,v}(Y) \longrightarrow \\
Pre_2^{E[x:=v],\,h_0}(Z) \longrightarrow \\
Post_2^{h',\,v}(Z) \longrightarrow \\
Post^{h',\,v}(X)
\end{gathered}$$

LET

$$\lambda\, Pre\ Pre_1\ Post_1\ x\ Pre_2\ Post_2\ Post.\ \mathtt{LET}$$

$$\equiv$$

$$\mathtt{lemma}_1(Pre,\ Pre_1) \wedge$$

$$\mathtt{lemma}_2(Pre,\ Pre_1,\ Post_1,\ x,\ Pre_2) \wedge$$

$$\mathtt{lemma}_3(Pre,\ Pre_1,\ Post_1,\ x,\ Pre_2,\ Post_2,\ Post)$$

$$\left.\begin{array}{l} \mathtt{LET}(Pre_1,\ Pre_{11},\ Post_{11},\ x,\ Pre_{12},\ Post_{12},\ Post_1) \\ \mathtt{LET}(Pre_2,\ Pre_{21},\ Post_{21},\ x,\ Pre_{22},\ Post_{22},\ Post_2) \end{array}\right\} \Rightarrow$$

$$\mathtt{LET}(Pre_1 \wedge Pre_2,\ Pre_{11} \wedge Pre_{21},\ Post_{11} \wedge Post_{21},\ x,$$
$$Pre_{12} \wedge Pre_{22},\ Post_{12} \wedge Post_{22},\ Post_1 \wedge Post_2)$$

is satisfied as well

A non-decent let-rule

$$\frac{\begin{array}{c} e_1 : \; Pre_1 \;\Rightarrow\; Post_1 \\ e_2 : \; Pre_2 \;\Rightarrow\; Post_2 \\ e_1 : \; \mathcal{A} \end{array}}{\texttt{let x =}e_1\;\texttt{in}\;e_2 : \; Pre \;\Rightarrow\; Post}\;\texttt{Let}$$

$$\begin{array}{c} \texttt{lemma}_2(Pre,\; Pre_1,\; Post_1,\; \mathcal{A},\; x,\; Pre_2) \equiv \forall\, E\, h\, h_0\, v_0.\, \forall\, X\, Y. \\ Pre^{E,\, h}(X) \longrightarrow \\ Pre_1^{E,\, h}(Y) \longrightarrow \\ Post_1^{h_0,\, v}(Y) \longrightarrow \\ \mathcal{A}(E,\, h,\, h_0,\, v_0) \longrightarrow \\ \exists\, Z. Pre_2^{E[x:=v],\, h_0}(Z) \end{array}$$

Similarly with $\texttt{lemma}_3$.

non-decent LET

$$\lambda\, Pre\ Pre_1\ Post_1\ x\ Pre_2\ Post_2\ Post.\ \texttt{LET}$$

$$\equiv$$

$$\texttt{lemma}_1(Pre,\ Pre_1) \wedge$$
$$\texttt{lemma}_2(Pre,\ Pre_1,\ Post_1,\ \mathcal{A},\ x,\ Pre_2) \wedge$$
$$\texttt{lemma}_3(Pre,\ Pre_1,\ Post_1,\ \mathcal{A},\ x,\ Pre_2,\ Post_2,\ Post)$$

Interleaving

We want to get a decent let-rule by approximating $\mathcal{A}$ with another type system $(Pre^A,\ Post^A)$.

Define

$\mathtt{Approximate}(Pre,\ \mathcal{A}) \equiv \forall\, E\, h\, h_0\, v_0 . Pre(E,\ h) \longrightarrow \mathcal{A}(E,\ h,\ h_0,\ v_0)$.

$$\underbrace{\begin{array}{l}\mathtt{LET}(Pre_1,\ Pre_{11},\ Post_{11},\ \mathcal{A},\ x,\ Pre_{12},\ Post_{12},\ Post_1)\\ \mathtt{LET}(Pre_2,\ Pre_{21},\ Post_{21},\ x,\ Pre_{22},\ Post_{22},\ Post_2)\\ \mathtt{Approximate}(Pre_2,\ \mathcal{A})\end{array}}_{\Downarrow}$$

$$\frac{\begin{array}{c}e_1 :\ Pre_{11} \wedge Pre_{21}\ \Rightarrow\ Post_{11} \wedge Post_{21}\\ e_2 :\ Pre_{12} \wedge Pre_{22}\ \Rightarrow\ Post_{12} \wedge Post_{22}\end{array}}{\mathtt{let\ x} = e_1\ \mathtt{in}\ e_2 :\ Pre_1 \wedge Pre_2\ \Rightarrow\ Post_1 \wedge Post_2}$$

Non-decent Let-rule becomes a decent one: no non-typeable assertion

because

$$\left.\begin{array}{l}\texttt{LET}(Pre_1,\ Pre_{11},\ Post_{11},\ \mathcal{A},\ x,\ Pre_{12},\ Post_{12},\ Post_1)\\ \texttt{LET}(Pre_2,\ Pre_{21},\ Post_{21},\ x,\ Pre_{22},\ Post_{22},\ Post_2)\\ \texttt{Approximate}(Pre_2,\ \mathcal{A})\end{array}\right\}\Rightarrow$$

$$\texttt{LET}(Pre_1 \wedge Pre_2,\ Pre_{11} \wedge Pre_{21},\ Post_{11} \wedge Post_{21},\ x,\ Pre_{12} \wedge Pre_{22}$$

Conclusions and Future Work

(Some of) MRG's achievements:

- Type inference system for linear heap space bounds,
- The system of derived assertions for bytecode logic,
- MRG-architecture: PCC framework

Future Work:

- Advanced type inference system for nonlinear heap usage bounds.
- Generic derived assertions are to be instantiated with different assertions. In particular, with the "pure" resource aware assertion and sharing-managing ones,
- PCC framework with Isabelle theorem prover is huge and at present may be used for off-device verification. Proof checking on the device itself is an extremely challenging goal!