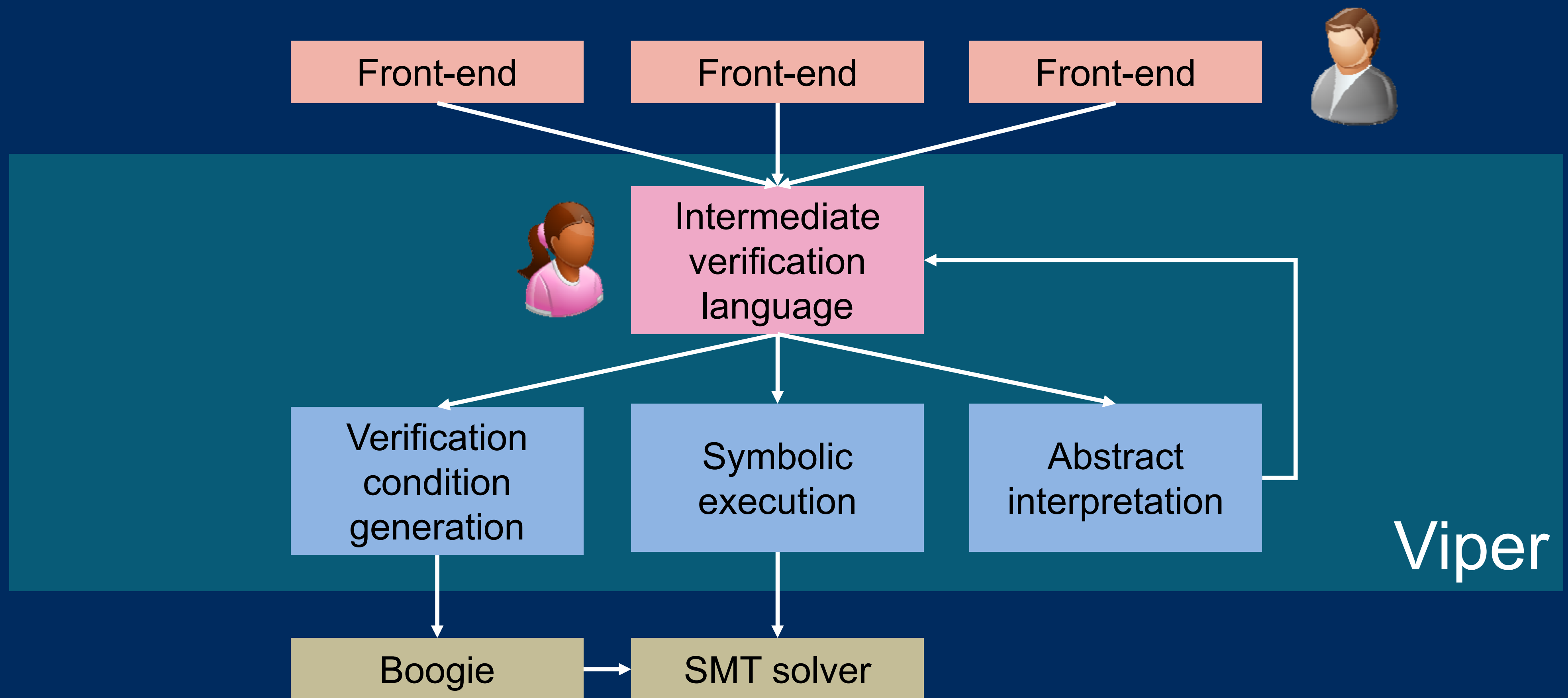
Prototyping
Relaxed Separation Logic in Viper

Peter Müller

ETH zürich


Front-end
Front-end
Front-end
Intermediate verification language
Verification condition generation
Symbolic execution
Abstract interpretation
Viper
Boogie
SMT solver

Separation Logic

- Heap properties are specified via points-to assertions

$$x.f \rightarrow v$$

- Each heap access to x.f requires permission to x.f

$$\{\ x.f \rightarrow _\ \}\ \ x.f := v\ \ \{\ x.f \rightarrow v\ \}$$

Separating Conjunction

- Composition of heaps is described using separating conjunction

$$P * R$$

- Frame rule

$$\frac{\{P\}\ S\ \{Q\}}{\{P * R\}\ S\ \{Q * R\}}$$

- Parallel composition

$$\frac{\{P_1\}\ S_1\ \{Q_1\} \qquad \{P_2\}\ S_2\ \{Q_2\}}{\{P_1 * P_2\}\ S_1 \,||\, S_2\ \{Q_1 * Q_2\}}$$

Permission Transfer

$$\frac{\dfrac{\{P\}\ \textbf{method}\ m\ \{Q\}}{\{P\}\ e.m()\ \{Q\}}}{\{P * R\}\ e.m()\ \{Q * R\}}$$

$$\frac{\dfrac{\{P_1\}\ S_1\ \{Q_1\} \qquad \{P_2\}\ S_2\ \{Q_2\}}{\{P_1 * P_2\}\ S_1 \,||\, S_2\ \{Q_1 * Q_2\}}}{\{P_1 * P_2 * R\}\ S_1 \,||\, S_2\ \{Q_1 * Q_2 * R\}}$$

Exhale and Inhale

exhale P

- Assert all logical constraints in P
- Check and remove permissions described by P

inhale Q

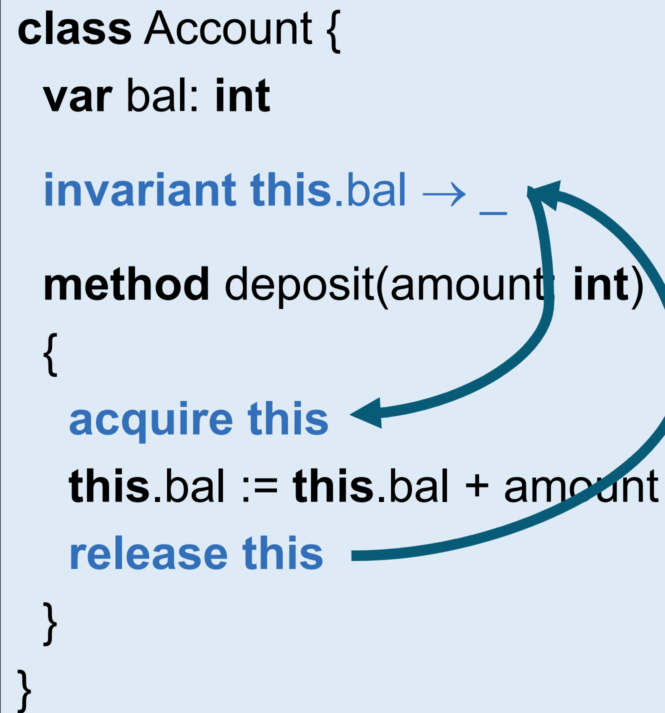
- Obtain permissions described by Q
- Assume all logical constraints in Q

$$\frac{\dfrac{\{P\}\ \textbf{method}\ m\ \{Q\}}{\{P\}\ e.m()\ \{Q\}}}{\{P * R\}\ e.m()\ \{Q * R\}}$$

exhale P
inhale Q

- Analogs of **assert** and **assume**

Encoding Monitors




```
inhale this.bal → _
this.bal := this.bal + amount
exhale this.bal → _
```

Weak Memory

- Modern hardware often does not provide sequentially consistent shared memory
- Weak memory permits behaviors that are not possible under sequential consistency
- However, data-race free programs have only sequentially consistent behaviors

```
        a = 0;
        b = 0;

a = 1;       ||    b = 1;
print(b);    ||    print(a);
```

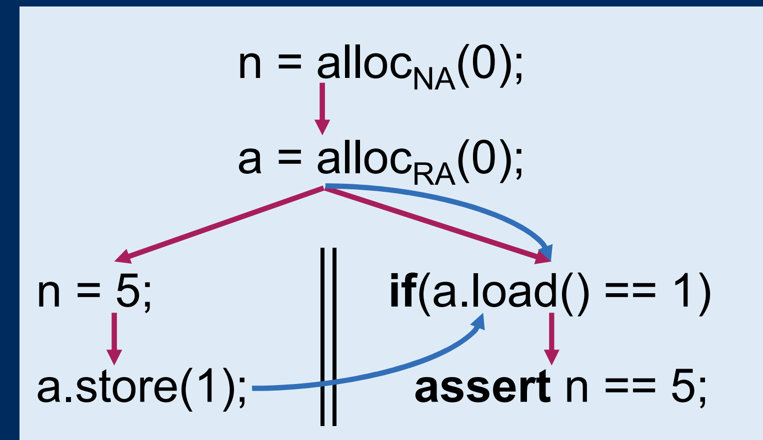
Possible results:
under SC: 10, 01, 11
under WM: also 00

C11

- The C11 memory model provides several kinds of variables

- Non-atomic variables
  - Data races are errors

- Atomic variables with release-write and acquire-read
  - Writes and reads are synchronized



- Relaxed separation logic (RSL) supports some features of the C11 memory model

Non-Atomic Variables

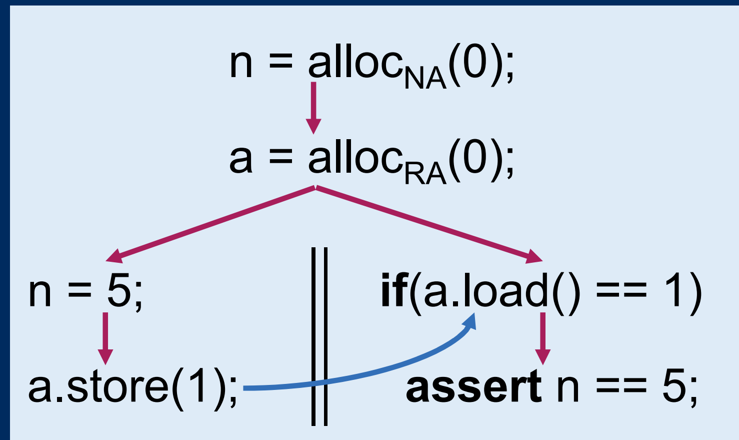
- Permissions prevent data races on non-atomic variables

$$\{ \text{true} \}\ x = \text{alloc}_{NA}(v)\ \{ x \rightarrow v \}$$

$$\{ x \rightarrow _ \}\ {}^{*}x = v\ \{ x \rightarrow v \}$$

$$\{ x \rightarrow V \}\ t = {}^{*}x\ \{ x \rightarrow V \ast t = V \}$$

Release-Acquire

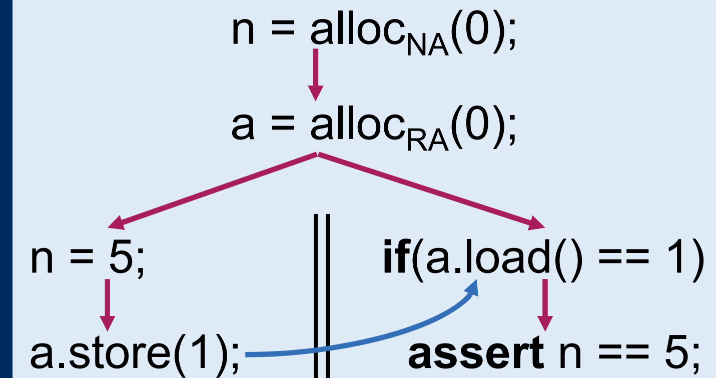


- Races on atomic variables are permitted
- Release-acquire can be seen as message passing
- Messages may transfer permissions to non-atomic variables

Location Invariants

- Location invariant Q(v) specifies an assertion that holds when the location has value v
- Acquire-read of value v transfers permissions of Q(v) from atomic variable to thread
- Release-write of value v transfers permissions of Q(v) from thread to atomic variable

$$Q(v) \equiv \begin{cases} n \to 5 & \text{if } v = 1 \\ \text{true} & \text{otherwise} \end{cases}$$



Proof Rules

- Choose location invariant when allocating an atomic location

$$\{ Q(v) \} \;\; x = alloc_{RA}(v) \;\; \{ Rel_Q(x) \ast Acq_Q(x) \}$$

- Release-write gives up permissions

$$\{ Rel_Q(x) \ast Q(v) \} \;\; x.store(v) \;\; \{ Rel_Q(x) \}$$

- Acquire-read gains permissions

$$\{ Acq_Q(x) \} \;\; t = x.load() \;\; \{ Q(t) \ast Acq_Q(x) \}$$

Proof Rules

$$\{ \text{Acq}_Q(x) \}\ \ t = x.\text{load}()\ \ \{ Q(t) * \text{Acq}_{Q[t := \text{true}]}(x) \}$$

$$Q(v) \equiv \begin{cases} n \rightarrow 5 & \text{if } v = 1 \\ \text{true} & \text{otherwise} \end{cases}$$

```
x = a.load();
y = a.load();
if(x == y)
  assert false;
```

- Reading the same value more than once would duplicate permissions

Proof Outline

$\{ \text{true} \}$

n = $\text{alloc}_{NA}(0)$;

$\{ n \rightarrow 0 \}$

a = $\text{alloc}_{RA}(0)$;

$\{ n \rightarrow 0 \ast \text{Rel}_Q(a) \ast \text{Acq}_Q(a) \}$

$\{ n \rightarrow 0 \ast \text{Rel}_Q(a) \}$

n = 5;

$\{ n \rightarrow 5 \ast \text{Rel}_Q(a) \}$

a.store(1);

$\{ \text{Rel}_Q(a) \}$

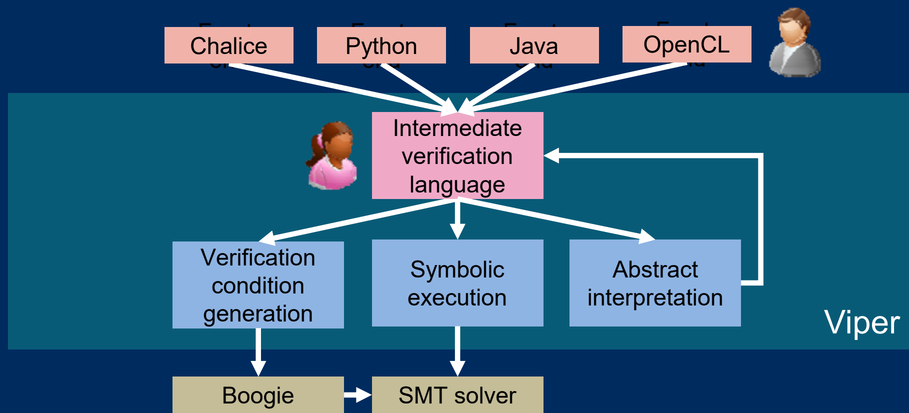
$\{ \text{Acq}_Q(a) \}$

if(a.load() == 1)

$\{ Q(1) \ast \text{Acq}_Q(a) \}$

assert n == 5;

$$Q(v) \equiv \begin{cases} n \rightarrow 5 & \text{if } v = 1 \\ \text{true} & \text{otherwise} \end{cases}$$

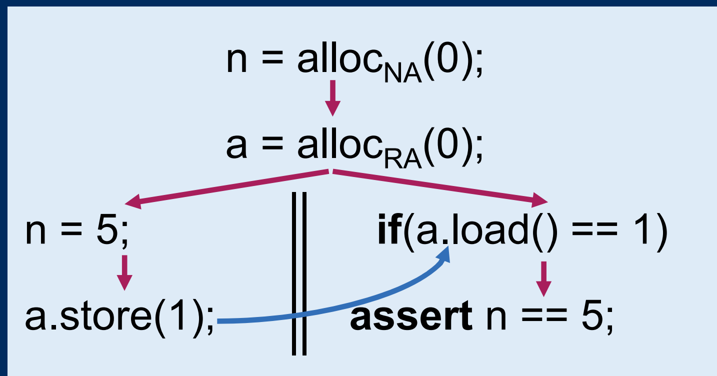


Intermediate languages enable prototyping and reuse of infrastructure

$$\frac{\{ P \}\ S\ \{ Q \}}{\{ P * R \}\ S\ \{ Q * R \}}$$

$$\frac{\{ P_1 \}\ S_1\ \{ Q_1 \} \qquad \{ P_2 \}\ S_2\ \{ Q_2 \}}{\{ P_1 * P_2 \}\ S_1 \parallel S_2\ \{ Q_1 * Q_2 \}}$$

Permissions enable framing and reasoning about concurrency



RSL proof rules can be encoded into Viper

viper.ethz.ch