Extracting a Data Flow Analyser in Constructive Logic

David Cachera, Thomas Jensen, David Pichardie and Vlad Rusu

APPSEM'04, Tallinn

IRISA
institut de recherche en informatique et systèmes aléatoires

école normale supérieure de Cachan
ENS BRETAGNE

Static program analysis

The goals of static program analysis

- To prove properties about the run-time behaviour of a program
- In a fully automatic way
- Without actually executing this program

Static program analysis

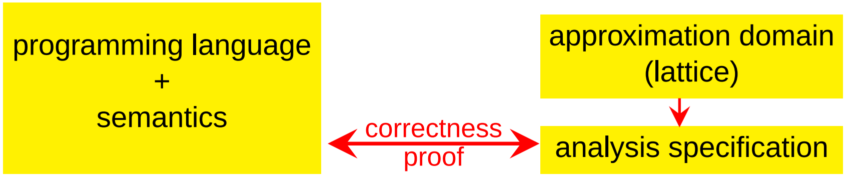
The goals of static program analysis

- To prove properties about the run-time behaviour of a program
- In a fully automatic way
- Without actually executing this program

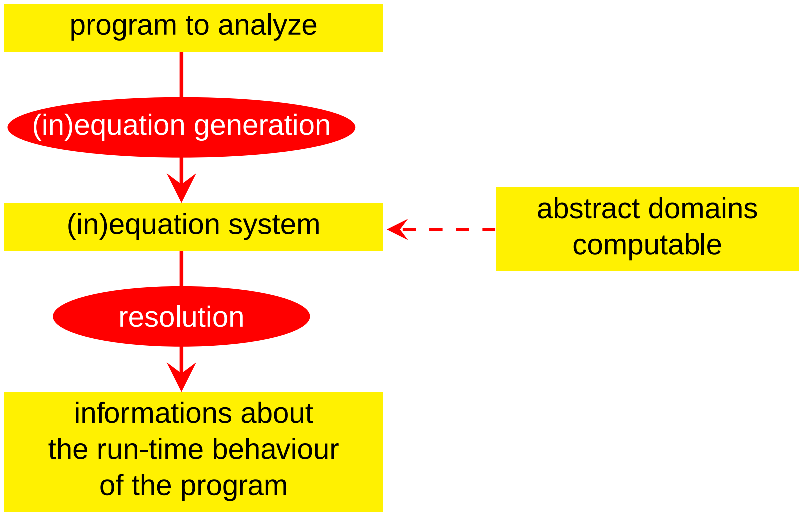
Solid foundations for designing an analyser

- Formalization and correctness proof by abstract interpretation
- Resolution of constraints on lattices by iteration and symbolic computation

Formalization



Resolution


program to analyze
(in)equation generation
(in)equation system
abstract domains computable
resolution
informations about the run-time behaviour of the program

So what’s the problem ?

Formalization part

$$\ddot{\alpha}\llbracket P\rrbracket(\mathrm{Post}\llbracket \mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}\rrbracket)$$

$=$ $\wr$def. (110) of $\ddot{\alpha}\llbracket P\rrbracket\wr$

$$\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{Post}\llbracket \mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}\rrbracket \circ \ddot{\gamma}\llbracket P\rrbracket$$

$=$ $\wr$def. (103) of Post$\wr$

$$\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{post}[\tau^\star\llbracket \mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}\rrbracket] \circ \ddot{\gamma}\llbracket P\rrbracket$$

$=$ $\wr$big step operational semantics (93)$\wr$

$$\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{post}\,[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{B}) \circ \tau^\star\llbracket S_t\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{t}) \cup (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{\bar{B}}) \circ \tau^\star\llbracket S_f\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{f})] \circ \ddot{\gamma}\llbracket P\rrbracket$$

$=$ $\wr$Galois connection (98) so that post preserves joins$\wr$

$$\ddot{\alpha}\llbracket P\rrbracket \circ (\mathrm{post}\,[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{B}) \circ \tau^\star\llbracket S_t\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{t})] \mathbin{\dot{\sqcup}} \mathrm{post}[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{\bar{B}}) \circ \tau^\star\llbracket S_f\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{f})]) \circ \ddot{\gamma}\llbracket P\rrbracket$$

$=$ $\wr$Galois connection (106) so that $\ddot{\alpha}\llbracket P\rrbracket$ preserves joins$\wr$

$$(\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{post}[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{B}) \circ \tau^\star\llbracket S_t\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{t})] \circ \ddot{\gamma}\llbracket P\rrbracket) \mathbin{\ddot{\sqcup}} (\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{post}[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{\bar{B}}) \circ \tau^\star\llbracket S_f\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{f})] \circ \ddot{\gamma}\llbracket P\rrbracket)$$

$\ddot{\sqsubseteq}$ $\wr$lemma (5.3) and similar one for the **else** branch$\wr$

```
λJ • let J^{t'} = λl ∈ in_P⟦P⟧•(l = at_P⟦S_t⟧ ? J_{at_P⟦S_t⟧} ⊔̇ Abexp⟦B⟧(J_ℓ) ¿ J_l) in        (120)
       let J^{t''} = APost⟦S_t⟧(J^{t'}) in
         λl ∈ in_P⟦P⟧•(l = ℓ' ? J^{t''}_{ℓ'} ⊔̇ J^{t''}_{after_P⟦S_t⟧} ¿ J^{t''}_l)
     ⊔̈
     let J^{f'} = λl ∈ in_P⟦P⟧•(l = at_P⟦S_f⟧ ? J_{at_P⟦S_f⟧} ⊔̇ Abexp⟦T(¬B)⟧(J_ℓ) ¿ J_l) in
       let J^{f''} = APost⟦S_f⟧(J^{f'}) in
         λl ∈ in_P⟦P⟧•(l = ℓ' ? J^{f''}_{ℓ'} ⊔̇ J^{f''}_{after_P⟦S_f⟧} ¿ J^{f''}_l)
```

$=$ $\wr$by grouping similar terms$\wr$

```
λJ • let J^{t'} = λl ∈ in_P⟦P⟧•(l = at_P⟦S_t⟧ ? J_{at_P⟦S_t⟧} ⊔̇ Abexp⟦B⟧(J_ℓ) ¿ J_l)
     and J^{f'} = λl ∈ in_P⟦P⟧•(l = at_P⟦S_f⟧ ? J_{at_P⟦S_f⟧} ⊔̇ Abexp⟦T(¬B)⟧(J_ℓ) ¿ J_l) in
       let J^{t''} = APost⟦S_t⟧(J^{t'})
       and J^{f''} = APost⟦S_f⟧(J^{f'}) in
         λl ∈ in_P⟦P⟧•(l = ℓ' ? J^{t''}_{ℓ'} ⊔̇ J^{t''}_{after_P⟦S_t⟧} ⊔̇ J^{f''}_{ℓ'} ⊔̇ J^{f''}_{after_P⟦S_f⟧} ¿ J^{t''}_l ⊔̇ J^{f''}_l)
```

$=$ $\wr$by locality (113) and labelling scheme (59) so that in particular $J^{t''}_{\ell'} = J^{t'}_{\ell'} = J^{t}_{\ell'} = J^{f}_{\ell'} = J^{f'}_{\ell'} = J^{f''}_{\ell'}$ and $\mathrm{APost}\llbracket S_t\rrbracket$ and $\mathrm{APost}\llbracket S_f\rrbracket$ do not interfere$\wr$

Formalization part

$$\ddot{\alpha}\llbracket P\rrbracket(\mathrm{Post}\llbracket \mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}\rrbracket)$$

$= \quad$ ⦃def. (110) of $\ddot{\alpha}\llbracket P\rrbracket$⦄

$$\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{Post}\llbracket \mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}\rrbracket \circ \ddot{\gamma}\llbracket P\rrbracket$$

$= \quad$ ⦃def. (103) of Post⦄

$$\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{post}[\tau^{\star}\llbracket \mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}\rrbracket] \circ \ddot{\gamma}\llbracket P\rrbracket$$

$= \quad$ ⦃big step operational semantics (93)⦄

$$\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{post}\,[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{B}) \circ \tau^{\star}\llbracket S_t\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{t}) \cup (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{\bar{B}}) \circ \tau^{\star}\llbracket S_f\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{f})] \circ \ddot{\gamma}\llbracket P\rrbracket$$

$= \quad$ ⦃Galois connection (98) so that post preserves joins⦄

$$\ddot{\alpha}\llbracket P\rrbracket \circ (\mathrm{post}\,[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{B}) \circ \tau^{\star}\llbracket S_t\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{t})] \mathbin{\dot{\cup}} \mathrm{post}[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{\bar{B}}) \circ \tau^{\star}\llbracket S_f\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{f})]) \circ \ddot{\gamma}\llbracket P\rrbracket$$

$= \quad$ ⦃Galois connection (106) so that $\ddot{\alpha}\llbracket P\rrbracket$ preserves joins⦄

$$(\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{post}[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{B}) \circ \tau^{\star}\llbracket S_t\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{t})] \circ \ddot{\gamma}\llbracket P\rrbracket) \mathbin{\ddot{\cup}} (\ddot{\alpha}\llbracket P\rrbracket \circ \mathrm{post}[(1_{\Sigma\llbracket P\rrbracket} \cup \tau^{\bar{B}}) \circ \tau^{\star}\llbracket S_f\rrbracket \circ (1_{\Sigma\llbracket P\rrbracket} \cup \tau^{f})] \circ \ddot{\gamma}\llbracket P\rrbracket)$$

$\ddot{\sqsubseteq} \quad$ ⦃lemma (5.3) and similar one for the **else** branch⦄

$$\begin{aligned}
&\lambda J\cdot \mathrm{let}\ J^{t'} = \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\cdot(l = \mathrm{at}_P\llbracket S_t\rrbracket\ ?\ J_{\mathrm{at}_P\llbracket S_t\rrbracket} \mathbin{\dot{\sqcup}} \mathrm{Abexp}\llbracket B\rrbracket(J_\ell)\ \text{¿}\ J_l)\ \mathrm{in} && (120)\\
&\quad \mathrm{let}\ J^{t''} = \mathrm{APost}\llbracket S_t\rrbracket(J^{t'})\ \mathrm{in}\\
&\quad\quad \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\cdot(l = \ell'\ ?\ J^{t''}_{\ell'} \mathbin{\dot{\sqcup}} J^{t''}_{\mathrm{after}_P\llbracket S_t\rrbracket}\ \text{¿}\ J^{t''}_l)\\
&\quad \ddot{\sqcup}\\
&\quad \mathrm{let}\ J^{f'} = \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\cdot(l = \mathrm{at}_P\llbracket S_f\rrbracket\ ?\ J_{\mathrm{at}_P\llbracket S_f\rrbracket} \mathbin{\dot{\sqcup}} \mathrm{Abexp}\llbracket T(\neg B)\rrbracket(J_\ell)\ \text{¿}\ J_l)\ \mathrm{in}\\
&\quad\quad \mathrm{let}\ J^{f''} = \mathrm{APost}\llbracket S_f\rrbracket(J^{f'})\ \mathrm{in}\\
&\quad\quad\quad \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\cdot(l = \ell'\ ?\ J^{f''}_{\ell'} \mathbin{\dot{\sqcup}} J^{f''}_{\mathrm{after}_P\llbracket S_f\rrbracket}\ \text{¿}\ J^{f''}_l)
\end{aligned}$$

$= \quad$ ⦃by grouping similar terms⦄

$$\begin{aligned}
&\lambda J\cdot \mathrm{let}\ J^{t'} = \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\cdot(l = \mathrm{at}_P\llbracket S_t\rrbracket\ ?\ J_{\mathrm{at}_P\llbracket S_t\rrbracket} \mathbin{\dot{\sqcup}} \mathrm{Abexp}\llbracket B\rrbracket(J_\ell)\ \text{¿}\ J_l)\\
&\quad \mathrm{and}\ J^{f'} = \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\cdot(l = \mathrm{at}_P\llbracket S_f\rrbracket\ ?\ J_{\mathrm{at}_P\llbracket S_f\rrbracket} \mathbin{\dot{\sqcup}} \mathrm{Abexp}\llbracket T(\neg B)\rrbracket(J_\ell)\ \text{¿}\ J_l)\ \mathrm{in}\\
&\quad\quad \mathrm{let}\ J^{t''} = \mathrm{APost}\llbracket S_t\rrbracket(J^{t'})\\
&\quad\quad \mathrm{and}\ J^{f''} = \mathrm{APost}\llbracket S_f\rrbracket(J^{f'})\ \mathrm{in}\\
&\quad\quad\quad \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\cdot(l = \ell'\ ?\ J^{t''}_{\ell'} \mathbin{\dot{\sqcup}} J^{t''}_{\mathrm{after}_P\llbracket S_t\rrbracket} \mathbin{\dot{\sqcup}} J^{f''}_{\ell'} \mathbin{\dot{\sqcup}} J^{f''}_{\mathrm{after}_P\llbracket S_f\rrbracket}\ \text{¿}\ J^{t''}_l \mathbin{\dot{\sqcup}} J^{f''}_l)
\end{aligned}$$

$= \quad$ ⦃by locality (113) and labelling scheme (59) so that in particular $J^{t''}_{\ell'} = J^{t'}_{\ell'} = J^{t}_{\ell'} = J^{f}_{\ell'} = J^{f'}_{\ell'} = J^{f''}_{\ell'}$ and $\mathrm{APost}\llbracket S_t\rrbracket$ and $\mathrm{APost}\llbracket S_f\rrbracket$ do not interfere⦄

Implementation part

```
int main(int argc, char **argv)
{
  int i, j, t, parent_a, parent_b;
  int **swap, **newpop, **oldpop;
  double *fit, *normfit;

  get_options(argc, argv, options, help_string);
  srandom(seed);
  read_specs(specs);
  size += (size / 2 * 2 != size);
  newpop = xmalloc(sizeof(int *) * size);
  oldpop = xmalloc(sizeof(int *) * size);
  fit = xmalloc(sizeof(double) * size);
  normfit = xmalloc(sizeof(double) * size);
  for(i = 0; i < size; i++) {
    newpop[i] = xmalloc(sizeof(int) * len);
    oldpop[i] = xmalloc(sizeof(int) * len);
    for(j = 0; j < len * 2; j++)
      random_solution(oldpop[i]);
  }
  for(t = 0; t < gens; t++) {
    compute_fitness(oldpop, fit, normfit);
    dump_stats(t, oldpop, fit);
    for(i = 0; i < size; i += 2) {
      parent_a = select_one(normfit);
      parent_b = select_one(normfit);
      reproduce(oldpop, newpop, parent_a, parent_b, i);
    }
    swap = newpop; newpop = oldpop; oldpop = swap;
  }
  exit(0);
}
```

Formalization part

$\dot{\ddot{\alpha}}[\![P]\!](\text{Post}[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!])$

$=$ ⟅def. (110) of $\dot{\ddot{\alpha}}[\![P]\!]$⟆

$\ddot{\alpha}[\![P]\!] \circ \text{Post}[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!] \circ \ddot{\gamma}[\![P]\!]$

$=$ ⟅def. (103) of Post⟆

$\ddot{\alpha}[\![P]\!] \circ \text{post}[\tau^\star[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!]] \circ \ddot{\gamma}[\![P]\!]$

$=$ ⟅big step operational semantics (93)⟆

$\ddot{\alpha}[\![P]\!] \circ \text{post}\,[(1_{\Sigma[\![P]\!]} \cup \tau^{B}) \circ \tau^\star[\![S_t]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^{t}) \cup (1_{\Sigma[\![P]\!]} \cup \tau^{\bar{B}}) \circ \tau^\star[\![S_f]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^{f})] \circ \ddot{\gamma}[\![P]\!]$

$=$ ⟅Galois connection (98) so that post preserves joins⟆

$\ddot{\alpha}[\![P]\!] \circ (\text{post}\,[(1_{\Sigma[\![P]\!]} \cup \tau^{B}) \circ \tau^\star[\![S_t]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^{t})] \mathbin{\dot{\cup}} \text{post}[(1_{\Sigma[\![P]\!]} \cup \tau^{\bar{B}}) \circ \tau^\star[\![S_f]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^{f})]) \circ \ddot{\gamma}[\![P]\!]$

$=$

$(\ddot{\alpha}[$

pos

$\dot{\sqsubseteq}$ ⟅I

$\lambda J \cdot \text{let } J^{t'} = \lambda l \in \text{in}_P[\![P]\!] \cdot (l = \text{at}_P[\![S_t]\!] \;?\; J_{\text{at}_P[\![S_t]\!]} \mathbin{\dot{\sqcup}} \text{Abexp}[\![B]\!](J_\ell) \;¿\; J_l) \text{ in}$ (120)

$\quad \text{let } J^{t''} = \text{APost}[\![S_t]\!](J^{t'}) \text{ in}$

$\quad\quad \lambda l \in \text{in}_P[\![P]\!] \cdot (l = \ell' \;?\; J^{t''}_{\ell'} \mathbin{\dot{\sqcup}} J^{t''}_{\text{after}_P[\![S_t]\!]} \;¿\; J^{t''}_l)$

Ü

$\text{let } J^{f'} = \lambda l \in \text{in}_P[\![P]\!] \cdot (l = \text{at}_P[\![S_f]\!] \;?\; J_{\text{at}_P[\![S_f]\!]} \mathbin{\dot{\sqcup}} \text{Abexp}[\![T(\neg B)]\!](J_\ell) \;¿\; J_l) \text{ in}$

$\quad \text{let } J^{f''} = \text{APost}[\![S_f]\!](J^{f'}) \text{ in}$

$\quad\quad \lambda l \in \text{in}_P[\![P]\!] \cdot (l = \ell' \;?\; J^{f''}_{\ell'} \mathbin{\dot{\sqcup}} J^{f''}_{\text{after}_P[\![S_f]\!]} \;¿\; J^{f''}_l)$

$=$ ⟅by grouping similar terms⟆

$\lambda J \cdot \text{let } J^{t'} = \lambda l \in \text{in}_P[\![P]\!] \cdot (l = \text{at}_P[\![S_t]\!] \;?\; J_{\text{at}_P[\![S_t]\!]} \mathbin{\dot{\sqcup}} \text{Abexp}[\![B]\!](J_\ell) \;¿\; J_l)$

$\quad \text{and } J^{f'} = \lambda l \in \text{in}_P[\![P]\!] \cdot (l = \text{at}_P[\![S_f]\!] \;?\; J_{\text{at}_P[\![S_f]\!]} \mathbin{\dot{\sqcup}} \text{Abexp}[\![T(\neg B)]\!](J_\ell) \;¿\; J_l) \text{ in}$

$\quad\quad \text{let } J^{t''} = \text{APost}[\![S_t]\!](J^{t'})$

$\quad\quad \text{and } J^{f''} = \text{APost}[\![S_f]\!](J^{f'}) \text{ in}$

$\quad\quad\quad \lambda l \in \text{in}_P[\![P]\!] \cdot (l = \ell' \;?\; J^{t''}_{\ell'} \mathbin{\dot{\sqcup}} J^{t''}_{\text{after}_P[\![S_t]\!]} \mathbin{\dot{\sqcup}} J^{f''}_{\ell'} \mathbin{\dot{\sqcup}} J^{f''}_{\text{after}_P[\![S_f]\!]} \;¿\; J^{t''}_l \mathbin{\dot{\sqcup}} J^{f''}_l)$

$=$ ⟅by locality (113) and labelling scheme (59) so that in particular $J^{t''}_{\ell'} = J^{t'}_{\ell'} = J^{t}_{\ell'} = J^{f}_{\ell'} = J^{f'}_{\ell'} = J^{f''}_{\ell'}$ and $\text{APost}[\![S_t]\!]$ and $\text{APost}[\![S_f]\!]$ do not interfere⟆

©P. Cousot

Implementation part

```
int main(int argc, char **argv)
{
  int i, j, t, parent_a, parent_b;
  int **swap, **newpop, **oldpop;
  double *fit, *normfit;

  get_options(argc, argv, options, help_string);
  srandom(seed);
  read_specs(specs);
  size += (size / 2 * 2 != size);
  newpop = xmalloc(sizeof(int *) * size);
```

```
    oldpop[i] = xmalloc(sizeof(int) * len);
    for(j = 0; j < len * 2; j++)
      random_solution(oldpop[i]);
  }
  for(t = 0; t < gens; t++) {
    compute_fitness(oldpop, fit, normfit);
    dump_stats(t, oldpop, fit);
    for(i = 0; i < size; i += 2) {
      parent_a = select_one(normfit);
      parent_b = select_one(normfit);
      reproduce(oldpop, newpop, parent_a, parent_b, i);
    }
    swap = newpop; newpop = oldpop; oldpop = swap;
  }
  exit(0);
}
```

Do both parts talk about the same ?

Static Analysis for real-life languages

Example of real-life language : bytecode JavaCard

- 180 instructions
- Real need of static analysis to verify properties about security, memory management, ...

For this kind of languages,

- Abstract domains can be complex
- Correctness proofs become long and tiresome
- Implementation and maintenance of the analyser become a software engineering task

In this work

We propose a technique based on the Coq proof assistant

- To specify a static analysis,
- To prove its correctness wrt. the semantics of the language,
- To extract a static analyser from the proof of existence of a correct program analysis result

Program-as-proofs paradigm:

Write a function f which verifies a specification P $\forall x,\ P(x, f(x))$ $\Longleftrightarrow$ Make a constructive proof of $\forall x,\ \exists y,\ P(x, y)$

Outline

- Motivation
- A Static Analysis for Carmel
- Building a certified static analyser
- Conclusion

Outline

- Motivation
- A Static Analysis for Carmel
- Building a certified static analyser
- Conclusion

Case study : a static analysis for Carmel

We follow the analysis proposed by René Rydhof Hansen[1]

- Carmel : an intermediate representation of Java Card byte code
- Construction of a certified data flow analyser for Carmel

[1] René Rydhof Hansen. Flow Logic for Carmel. SECSAFE-IMM-001, 2002

Syntax of Carmel

Instruction $::=$

$$\begin{array}{l}\texttt{nop}\end{array}$$

$$\left.\begin{array}{l}\texttt{push}\ c\\ \texttt{pop}\\ \texttt{numop}\ op\end{array}\right\}\text{stack manipulation}$$

$$\left.\begin{array}{l}\texttt{load}\ x\\ \texttt{store}\ x\end{array}\right\}\text{local variables manipulation}$$

$$\left.\begin{array}{l}\texttt{if}\ pc\\ \texttt{goto}\ pc\end{array}\right\}\text{jump}$$

$$\left.\begin{array}{l}\texttt{new}\ cl\\ \texttt{putfield}\ f\\ \texttt{getfield}\ f\end{array}\right\}\text{heap manipulation}$$

$$\left.\begin{array}{l}\texttt{invokevirtual}\ m_{id}\\ \texttt{return}\end{array}\right\}\text{method call and return}$$

Semantic domains

$$
\begin{array}{rcll}
\mathsf{Val} & ::= & \mathsf{num}\ n & n \in \mathbb{N} \\
 & & \mathsf{ref}\ r & r \in \mathsf{Reference} \\
 & & \mathsf{null} & \\
\mathsf{Stack} & = & \mathsf{Val}^* & \\
\mathsf{LocalVar} & = & \mathsf{Var} \rightarrow \mathsf{Val} & \\
\mathsf{Frame} & = & \mathsf{PointProg} \times \mathsf{NameMethod} & \\
 & & & \times \mathsf{LocalVar} \times \mathsf{Stack} \\
\mathsf{CallStack} & = & \mathsf{Frame}^* & \\
\mathsf{Object} & = & \mathsf{FieldName} \rightarrow \mathsf{Val} & \\
\mathsf{Heap} & = & \mathsf{Reference} \rightarrow \mathsf{Object}_{\perp} & \\
\mathsf{State} & = & \mathsf{Heap} \times \mathsf{CallStack} &
\end{array}
$$

Example :

$$(H, \langle m, pc, L, v :: S \rangle :: SF)$$

Dynamic semantics

Operational semantics with rules like

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{push}\ c}{(H, \langle m, pc, L, S\rangle :: SF) \Rightarrow (H, \langle m, pc+1, L, c :: S\rangle :: SF)}$$

$$\frac{\begin{array}{c}\text{instructionAt}_P(m, pc) = \texttt{invokevirtual}\ m_{id} \\ \begin{array}{rcl} m' & = & \text{methodLookup}(m_{id}, h(loc)) \\ f' & = & \langle m', 1, V, \varepsilon\rangle \\ f'' & = & \langle m, pc, l, s\rangle \end{array}\end{array}}{(h, \langle m, pc, l, loc :: V :: s\rangle :: sf) \Rightarrow (h, f' :: f'' :: sf)}$$

A Static Analysis for Carmel

We want to calculate an approximation $\left(\hat{H}, \hat{L}, \hat{S}\right)$ on the domain

$$\begin{aligned}\widehat{\text{State}} = \widehat{\text{Heap}} \quad & \times \quad \left(\text{NameMethod} \times \text{PointProg} \rightarrow \widehat{\text{LocalVar}}\right) \\ & \times \quad \left(\text{NameMethod} \times \text{PointProg} \rightarrow \widehat{\text{Stack}}\right)\end{aligned}$$

- An approximation for all reachable heaps
- For each program points, an approximation of the operand stack and the local variables
- An object is abstracted to its class
- Numeric values are abstracted using Killdall’s Constant Propagation domain

Analysis specification

Each instruction impose constraints on $\left(\hat{H}, \hat{L}, \hat{S}\right)$.

Example

```
0 :   push 1
1 :   push 2
2 :   store 0
3 :   load 0
4 :   numop mult
5 :   goto 1
```

Analysis specification

Each instruction impose constraints on $\left(\hat{H}, \hat{L}, \hat{S}\right)$.

Example

$\widehat{\mathrm{nil}} \sqsubseteq \hat{S}(m, 0)$ $\top \sqsubseteq \hat{L}(m, 0)$

```
0 :   push 1
1 :   push 2
2 :   store 0
3 :   load 0
4 :   numop mult
5 :   goto 1
```

Analysis specification

Each instruction impose constraints on $\left(\hat{H}, \hat{L}, \hat{S}\right)$.

Example

```
0 :   push 1
1 :   push 2
2 :   store 0
3 :   load 0
4 :   numop mult
5 :   goto 1
```

$$\widehat{\text{nil}} \sqsubseteq \hat{S}(m, 0) \qquad \top \sqsubseteq \hat{L}(m, 0)$$

$$\widehat{\text{push}}(\hat{1}, \hat{S}(m, 0)) \sqsubseteq \hat{S}(m, 1) \qquad \hat{L}(m, 0) \sqsubseteq \hat{L}(m, 1)$$

Analysis specification

Each instruction impose constraints on $\left(\hat{H}, \hat{L}, \hat{S}\right)$.

Example

	$\widehat{\text{nil}} \sqsubseteq \hat{S}(m,0)$	$\top \sqsubseteq \hat{L}(m,0)$
0 : `push 1`	$\widehat{\text{push}}(\hat{1}, \hat{S}(m,0)) \sqsubseteq \hat{S}(m,1)$	$\hat{L}(m,0) \sqsubseteq \hat{L}(m,1)$
1 : `push 2`	$\widehat{\text{push}}(\hat{2}, \hat{S}(m,1)) \sqsubseteq \hat{S}(m,2)$	$\hat{L}(m,1) \sqsubseteq \hat{L}(m,2)$
2 : `store 0`	$\widehat{\text{pop}}(\hat{S}(m,2)) \sqsubseteq \hat{S}(m,3)$	$\hat{L}(m,2)[0 \mapsto \widehat{\text{top}}(\hat{S}(m,2))] \sqsubseteq \hat{L}(m,3)$
3 : `load 0`	$\widehat{\text{push}}(\hat{L}(m,3)[0], \hat{S}(m,3)) \sqsubseteq \hat{S}(m,4)$	$\hat{L}(m,3) \sqsubseteq \hat{L}(m,4)$
4 : `numop mult`	$\ldots$	$\hat{L}(m,4) \sqsubseteq \hat{L}(m,5)$
5 : `goto 1`	$\hat{S}(m,5) \sqsubseteq \hat{S}(m,1)$	$\hat{L}(m,5) \sqsubseteq \hat{L}(m,1)$

Analysis solution

The smallest value which verifies all constraints.

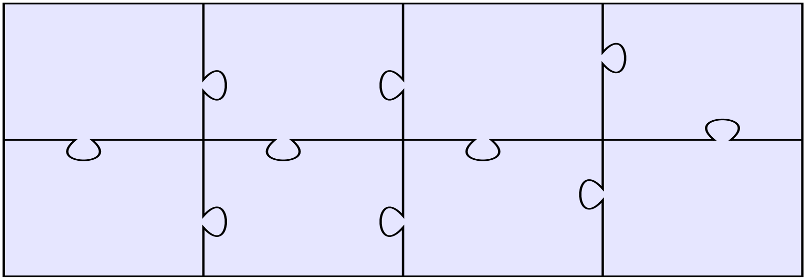
Example

		$\widehat{\text{nil}}$	$[0 \mapsto \top;\ 1 \mapsto \top]$
0 :	`push 1`		
		$< \hat{2} >$	$[0 \mapsto \hat{1};\ 1 \mapsto \top]$
1 :	`push 2`		
		$< \hat{1} :: \hat{2} >$	$[0 \mapsto \hat{1};\ 1 \mapsto \top]$
2 :	`store 0`		
		$< \hat{2} >$	$[0 \mapsto \hat{1};\ 1 \mapsto \top]$
3 :	`load 0`		
		$< \hat{1} :: \hat{2} >$	$[0 \mapsto \hat{1};\ 1 \mapsto \top]$
4 :	`numop mult`		
		$< \hat{2} >$	$[0 \mapsto \hat{1};\ 1 \mapsto \top]$
5 :	`goto 1`		
		$\perp$	$\perp$

Outline

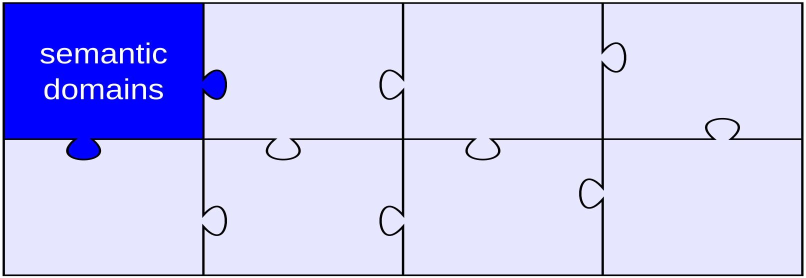
- Motivation
- A Static Analysis for Carmel
- Building a certified static analyser
- Conclusion

Building a certified static analyser



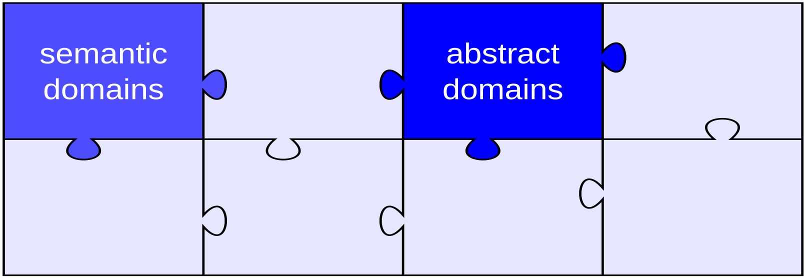
- A puzzle with 8 pieces,
- Each piece interacts with its neighbors

Building a certified static analyser



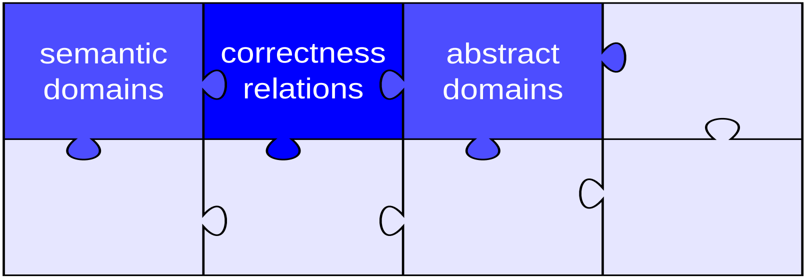
- Each semantic domain is modeled with a type
- Following exactly the definitions already seen in a previous slide

Building a certified static analyser



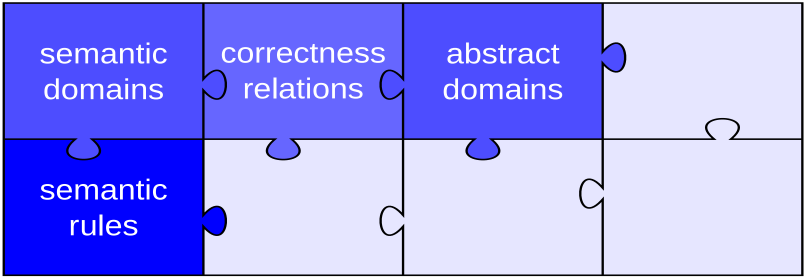
- Each semantic domain is in relation with an abstract domain
- an abstract domain is a lattice (formalization of lattices in Coq to follow...)

Building a certified static analyser



- A relation $\sim$ between State and $\widehat{\text{State}}$
- $s \sim \widehat{\Sigma}$ interprets as "$\widehat{\Sigma}$ is a correct approximation of s"
- $\sim$ must be monotone :
  $\forall s \in \text{State}, \forall \widehat{\Sigma}_1, \widehat{\Sigma}_2 \in \widehat{\text{State}},$
  $\quad$ if $s \sim \widehat{\Sigma}_1$ and $\widehat{\Sigma}_1 \sqsubseteq \widehat{\Sigma}_2$ then $s \sim \widehat{\Sigma}_2$

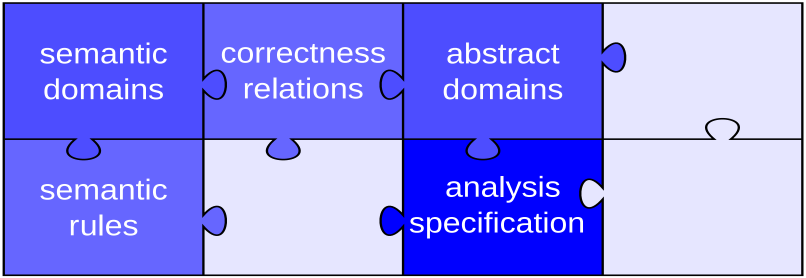
Building a certified static analyser



- The transition relation $\cdot \Rightarrow \cdot$ is defined using Coq inductive types
- Collecting semantics :
  $[\![P]\!] = \{s \mid \exists s_0 \text{ an initial state}, \text{with } s_0 \Rightarrow^* s\}$

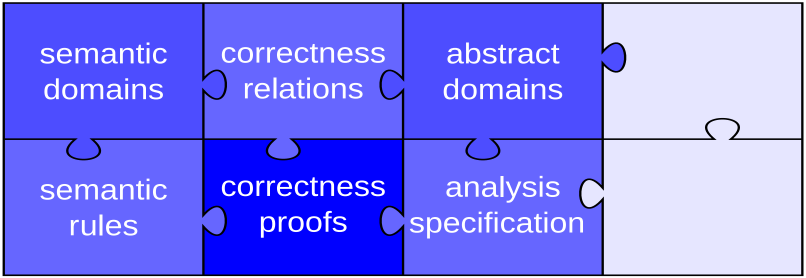
We want to compute a correct approximation of $[\![P]\!]$

Building a certified static analyser



- we define a predicate $P \vdash \widehat{\Sigma}$ which imposes a set of constraints on an abstract state $\widehat{\Sigma}$

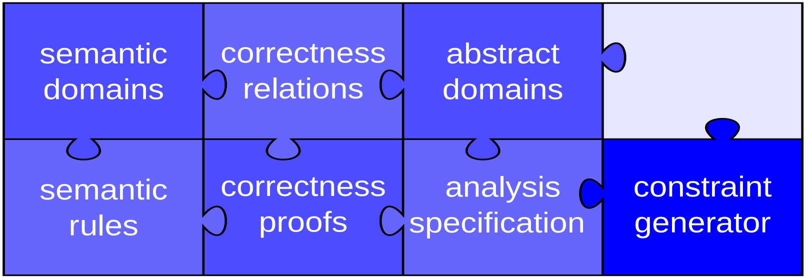
Building a certified static analyser



$$\forall P : \text{Program},\ \forall \widehat{\Sigma} : \widehat{\text{State}},\quad P \vdash \widehat{\Sigma} \Rightarrow [\![P]\!] \sim \widehat{\Sigma}$$

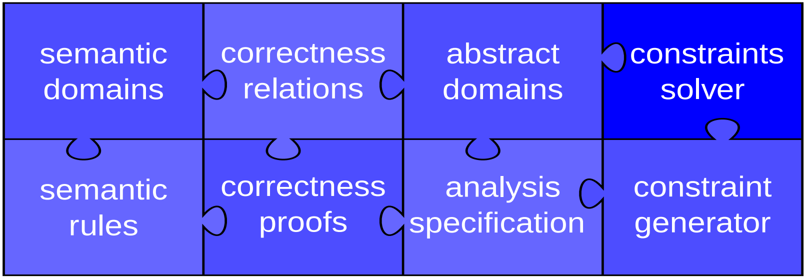
- One case by instruction
- With a special treatment for the `invokevirtual`/`return` instructions

Building a certified static analyser



- Collects all constraint of a given program

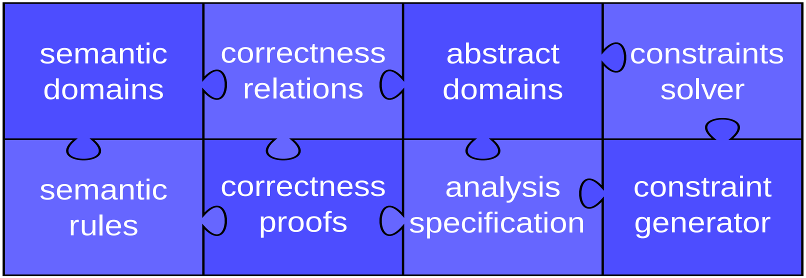
Building a certified static analyser



$$\forall P : \text{Program},\ \exists \widehat{\Sigma} : \widehat{\text{State}},\quad P \vdash \widehat{\Sigma}$$

- In fact, a stronger result : there exists a smallest solution

Building a certified static analyser



Final result

$$\left.\begin{array}{l}\forall P,\ \forall\widehat{\Sigma},\ P\vdash\widehat{\Sigma}\Rightarrow[\![P]\!]\sim\widehat{\Sigma}\\ \forall P,\ \exists\widehat{\Sigma},\ P\vdash\widehat{\Sigma}\end{array}\right\}\quad \forall P,\ \exists\widehat{\Sigma},\ [\![P]\!]\sim\widehat{\Sigma}$$

Abstract domains

The `lattice` type is a big structure :

```
Record Lattice [A:Set] :  Type := {
  eq :  A → A → Prop;

  order :  A → A → Prop;

  join :  A → A → A;

  eq_dec :  A → A → bool;

  bottom :  A;

  top :  A;


}.
```

Abstract domains

The `lattice` type is a big structure :

```
Record Lattice [A:Set] :  Type := {
  eq :  A → A → Prop;
    eq_prop :  ...; // eq  is an equivalence relation
  order :  A → A → Prop;
    order_prop :  ...; // order  is an order relation
  join :  A → A → A;
    join_prop :  ...; // join  is a correct binary least upper bound
  eq_dec :  A → A → bool;
    eq_dec_prop :  ...; // eq_dec  is a correct equality test
  bottom :  A;
    bottom_prop :  ...; // bottom  is the smallest element
  top :  A;
    top_prop :  ...; // top  is the biggest element
    acc_prop :  ...; // ⊐  is well founded (ascending chain condition)
}.
```

Abstract domains

The `lattice` type is a big structure :

```
Record Lattice [A:Set] : Type := {
  eq : A → A → Prop;
    eq_prop : ...; // eq  is an equivalence relation
  order : A → A → Prop;
    order_prop : ...; // order  is an order relation
  join : A → A → A;
    join_prop : ...; // join  is a correct binary least upper bound
  eq_dec : A → A → bool;
    eq_dec_prop : ...; // eq_dec  is a correct equality test
  bottom : A;
    bottom_prop : ...; // bottom  is the smallest element
  top : A;
    top_prop : ...; // top  is the biggest element
    acc_prop : ...; // ⊐  is well founded (ascending chain condition)
}.
```

A lattice library

- Two base lattices
  - Flat lattice of constants
  - Lattice of sets over a finite subset of integer
- Four functions to combine lattices
  - Product of lattice
  - Sum of lattice
  - Arrays whose elements live in a lattice and whose size is bounded (efficient functional structure)
  - List whose elements live in a lattice

This modular construction saves a considerable amount of time and effort.

For this analysis : (array (array (list (finiteSet $+$ constants))))
$\times$ (array (array (array (finiteSet $+$ constants))))
$\times$ (array (array (finiteSet $+$ constants)))

Constraint solver

1. A generic fixed point solver

$$\forall L : (\texttt{lattice}\ A),\ \forall f : A \to A,\ f \text{ monotone},$$
$$\exists x : A,\ x \text{ is the least fixed point of } f$$

   proof : the sequence $\bot, f(\bot), f^2(\bot), \ldots$ stabilizes on the least fixed point

2. We use it to solve the functional constraints, using the fact that

$$\begin{array}{c} x \text{ is the least solution of} \\ f_1(x) \sqsubseteq x, \ldots, f_n(x) \sqsubseteq x \end{array} \iff \begin{array}{c} x \text{ is the least fixed point of} \\ \hat{f}_1 \circ \cdots \circ \hat{f}_n \end{array}$$

   with $\hat{f}(x) = f(x) \sqcup x$

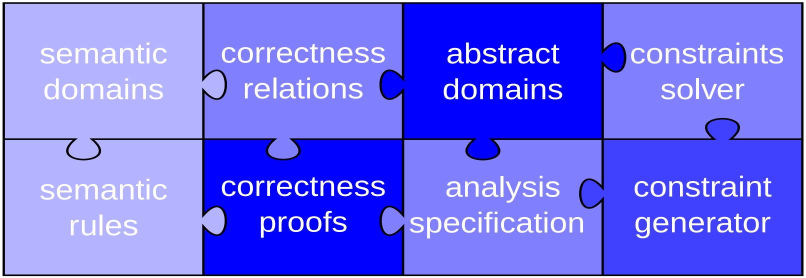
3. Combined with the constraint generator, we obtain

$$\forall P,\ \exists \widehat{\Sigma},\ P \vdash \widehat{\Sigma}$$

Outline

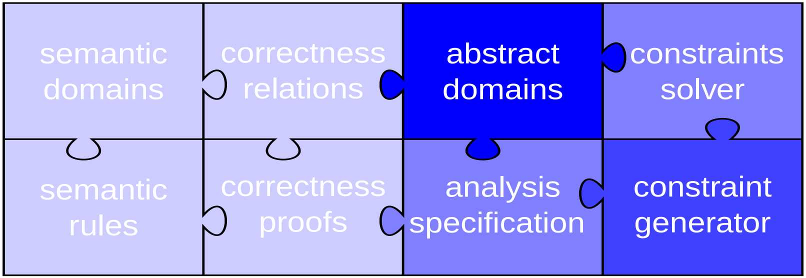
- Motivation
- A Static Analysis for Carmel
- Building a certified static analyser
- Conclusion

Where is the proof effort ?



- Most technical part in the lattice library
- Correctness part does not require specific competences in Coq
- A majority of proof a reusable to develop others analysis

Where is the programming effort ?



- The extraction mechanism only keeps the computational content of proofs
- The corresponding parts require a high attention to obtain an efficient analyser

Conclusion on the work

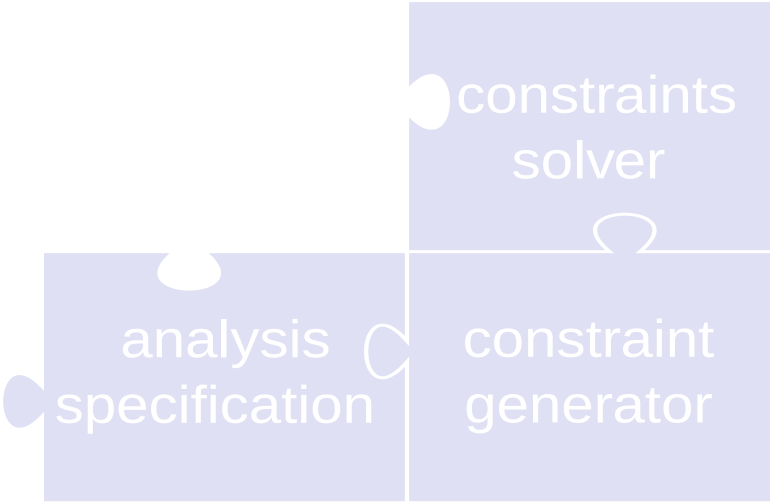
- We proposed a technique based on the Coq proof assistant
  - To develop a certified static analyser
  - To extract a correct analyser in Ocaml
- We illustrated this technique with a data flow analysis for the Carmel language
  - 10000 lines of Coq converted in 2000 lines of OCaml
  - With a reasonable efficiency of the analyser :
    - About 1 minute to analyse 1000 lines of Carmel byte code

Further works

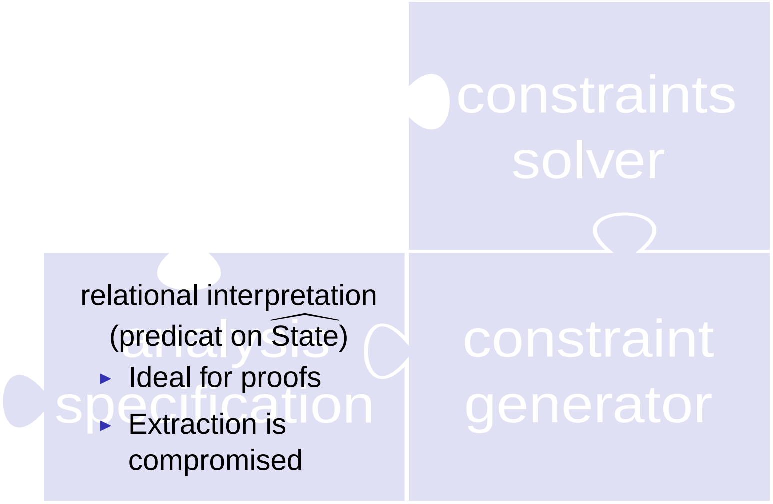
- Construction of an efficient certified work-set based program instead of the actual naive resolution
  - This program must be independent of the abstract domains
- Lattice of infinite height like intervals
- Automatization of the correctness proof ?
  - So as to quickly extend the number of language instructions
- A more extensive use of the abstract interpretation formalism
  - We must find a compromise between the reusability possibilities and the technical efforts in Coq
- Application of this technique to others languages

?

Constraint representation



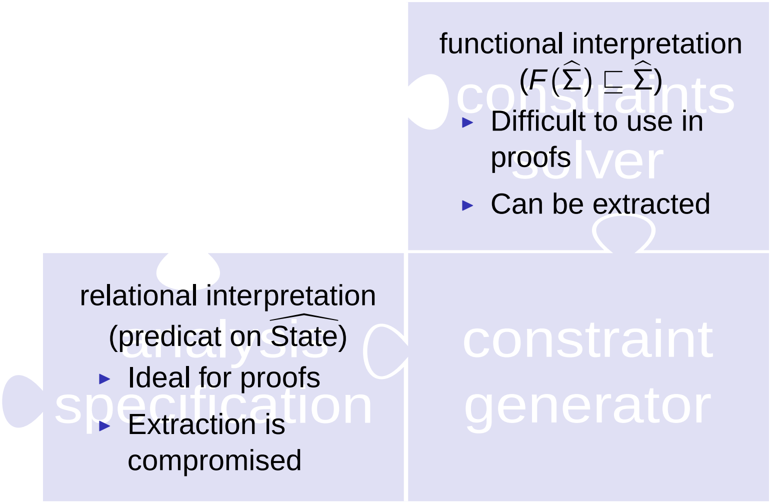
Constraint representation



relational interpretation
(predicat on $\widehat{\text{State}}$)

- Ideal for proofs
- Extraction is compromised

Constraint representation



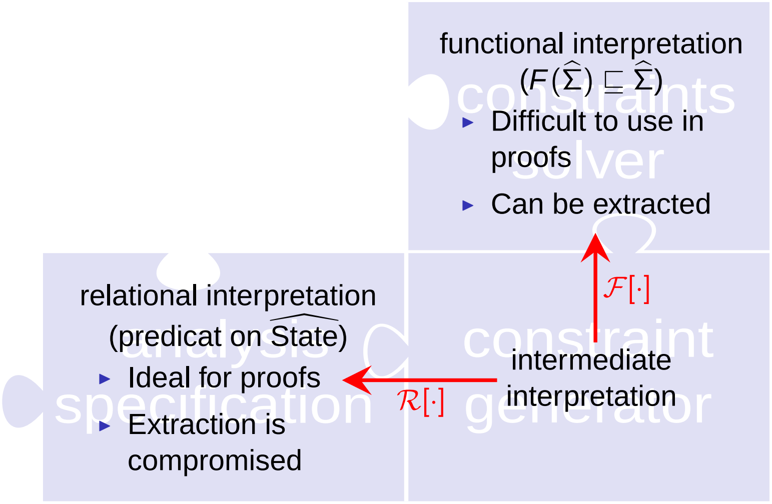
functional interpretation
$(F(\widehat{\Sigma}) \sqsubseteq \widehat{\Sigma})$

- Difficult to use in proofs
- Can be extracted

relational interpretation
(predicat on $\widehat{\text{State}}$)

- Ideal for proofs
- Extraction is compromised

Constraint representation



functional interpretation
$(F(\widehat{\Sigma}) \sqsubseteq \widehat{\Sigma})$

- Difficult to use in proofs
- Can be extracted

intermediate interpretation

relational interpretation
(predicat on $\widehat{\text{State}}$)

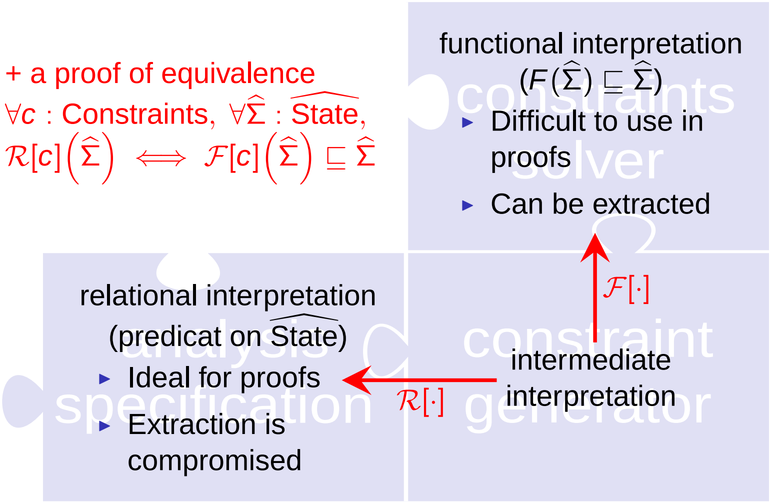
- Ideal for proofs
- Extraction is compromised

Constraint representation

+ a proof of equivalence

$\forall c : \text{Constraints}, \ \forall \widehat{\Sigma} : \widehat{\text{State}},$

$\mathcal{R}[c]\left(\widehat{\Sigma}\right) \iff \mathcal{F}[c]\left(\widehat{\Sigma}\right) \sqsubseteq \widehat{\Sigma}$



functional interpretation
($F(\widehat{\Sigma}) \sqsubseteq \widehat{\Sigma}$)

- Difficult to use in proofs
- Can be extracted

relational interpretation
(predicat on $\widehat{\text{State}}$)

- Ideal for proofs
- Extraction is compromised

intermediate interpretation

Correctness proof

$$\forall P : \text{Program},\ \forall \widehat{\Sigma} : \widehat{\text{State}},\quad P \vdash \widehat{\Sigma} \Longrightarrow [\![P]\!] \sim \widehat{\Sigma}$$

We prove it by well-founded induction on the length of the program execution.
Induction step:

- for all instructions I, except `return`
  $$\begin{array}{l} \forall P,\ \forall \widehat{\Sigma} : \widehat{\text{State}},\ P \vdash \widehat{\Sigma} \Longrightarrow \\ \qquad \forall s_1 : \text{State},\ s_1 \sim \widehat{\Sigma} \Longrightarrow \\ \qquad\qquad \forall s_2 : \text{State},\ [s_1 \Rightarrow_I s_2] \Longrightarrow s_2 \sim \widehat{\Sigma} \end{array}$$
- for the return instruction
  $$\begin{array}{l} \forall P,\ \forall \widehat{\Sigma} : \widehat{\text{State}},\ P \vdash \widehat{\Sigma} \Longrightarrow \\ \qquad \forall s_1 : \text{State},\ \left(\forall s,\ s \Rightarrow^* s_1 \Longrightarrow s \sim \widehat{\Sigma}\right) \Longrightarrow \\ \qquad\qquad \forall s_2 : \text{State},\ [s_1 \Rightarrow_{\texttt{return}} s_2] \Longrightarrow s_2 \sim \widehat{\Sigma} \end{array}$$