Extracting a Certified Static Analyser in Constructive Logic: Application to Proof Carrying Code

David Pichardie

CR2 INRIA - Lande project

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

- Abstract Interpretation gives a guideline
  - to formalise analyses
  - to prove their soundness with respect to the semantics of the programming language
- Resolution of constraints on lattices by iteration and symbolic computation

So what's the problem ?

Proof

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

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

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

$$= \quad \wr\text{def. (103) of Post}\wr$$

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

$$= \quad \wr\text{big step operational semantics (93)}\wr$$

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

$$= \quad \wr\text{Galois connection (98) so that post preserves joins}\wr$$

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

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

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

$$\dot{\ddot{\sqsubseteq}} \quad \wr\text{lemma (5.3) and similar one for the } \texttt{else} \text{ branch}\wr$$

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

$$= \quad \wr\text{by grouping similar terms}\wr$$

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

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

Proof

$$
\begin{array}{ll}
 & \dot{\ddot{\alpha}}\llbracket P\rrbracket(\mathrm{Post}\llbracket \mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}\rrbracket) \\
= & \quad \wr\text{def. (110) of } \dot{\ddot{\alpha}}\llbracket P\rrbracket \int \\
 & \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 \wr\text{def. (103) of Post}\int \\
 & \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 \wr\text{big step operational semantics (93)}\int \\
 & \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 \wr\text{Galois connection (98) so that post preserves joins}\int \\
 & \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 \wr\text{Galois connection (106) so that } \ddot{\alpha}\llbracket P\rrbracket \text{ preserves joins}\int \\
 & (\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{\dot{\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) \\
\dot{\ddot{\sqsubseteq}} & \quad \wr\text{lemma (5.3) and similar one for the } \mathtt{else} \text{ branch}\int \\
 & \lambda J\bullet \mathrm{let}\ J^{t'} = \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\bullet(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)\ ¿\ J_l)\ \mathrm{in} \qquad (120) \\
 & \qquad \mathrm{let}\ J^{t''} = \mathrm{APost}\llbracket S_t\rrbracket(J^{t'})\ \mathrm{in} \\
 & \qquad\quad \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\bullet(l = \ell'\ ?\ J^{t''}_{\ell'} \mathbin{\dot{\sqcup}} J^{t''}_{\mathrm{after}_P\llbracket S_t\rrbracket}\ ¿\ J^{t''}_l) \\
 & \quad \ddot{\sqcup} \\
 & \quad \mathrm{let}\ J^{f'} = \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\bullet(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)\ ¿\ J_l)\ \mathrm{in} \\
 & \qquad \mathrm{let}\ J^{f''} = \mathrm{APost}\llbracket S_f\rrbracket(J^{f'})\ \mathrm{in} \\
 & \qquad\quad \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\bullet(l = \ell'\ ?\ J^{f''}_{\ell'} \mathbin{\dot{\sqcup}} J^{f''}_{\mathrm{after}_P\llbracket S_f\rrbracket}\ ¿\ J^{f''}_l) \\
= & \quad \wr\text{by grouping similar terms}\int \\
 & \lambda J\bullet \mathrm{let}\ \ J^{t'} = \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\bullet(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)\ ¿\ J_l) \\
 & \quad \mathrm{and}\ J^{f'} = \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\bullet(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)\ ¿\ J_l)\ \mathrm{in} \\
 & \qquad \mathrm{let}\ \ J^{t''} = \mathrm{APost}\llbracket S_t\rrbracket(J^{t'}) \\
 & \qquad \mathrm{and}\ J^{f''} = \mathrm{APost}\llbracket S_f\rrbracket(J^{f'})\ \mathrm{in} \\
 & \qquad\quad \lambda l \in \mathrm{in}_P\llbracket P\rrbracket\bullet(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}\ ¿\ J^{t''}_l \mathbin{\dot{\sqcup}} J^{f''}_l) \\
= & \quad \wr\text{by locality (113) and labelling scheme (59) so that in particular } J^{t''}_{\ell'} = J^{t'}_{\ell'} = J^{t}_{\ell'} = J^{f}_{\ell'} \\
 & \quad = J^{f'}_{\ell'} = J^{f''}_{\ell'} \text{ and APost}\llbracket S_t\rrbracket \text{ and APost}\llbracket S_f\rrbracket \text{ do not interfere}\int
\end{array}
$$

Implementation

```
matrix_t* _matrix_alloc_int(const int mr, const int nc)
{
  matrix_t* mat = (matrix_t*)malloc(sizeof(matrix_t));
  mat->nbrows = mat->_maxrows = mr;
  mat->nbcolumns = nc;
  mat->_sorted = s;
  if (mr*nc>0){
    int i;
    pkint_t* q;
    mat->_pinit = _vector_alloc_int(mr*nc);
    mat->p = (pkint_t**)malloc(mr * sizeof(pkint_t*));
    q = mat->_pinit;
    for (i=0;i<mr;i++){
      mat->p[i]=q;
      q=q+nc;
    }}
  return mat;
}

void backsubstitute(matrix_t* con, int rank)
{
  int i,j,k;
  for (k=rank-1; k>=0; k--) {
    j = pk_cherni_intp[k];
    for (i=0; i<k; i++) {
      if (pkint_sgn(con->p[i][j]))
        matrix_combine_rows(con,i,k,i,j);
    }
    for (i=k+1; i<con->nbrows; i++) {
      if (pkint_sgn(con->p[i][j]))
        matrix_combine_rows(con,i,k,i,j);
    }}
}
```

Proof

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

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

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

$=$ ≀def. (103) of Post≀

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

$=$ ≀big step operational semantics (93)≀

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

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

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

$=$ ≀Galois connection (106) so that $\ddot{\alpha}[\![P]\!]$ preserves joins≀

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

$\dot{\ddot{\sqsubseteq}}$ ≀len

$\lambda J \cdot$ let

let

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

$\ddot{\sqcup}$

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

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

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

$=$ ≀by grouping similar terms≀

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

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

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

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

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

$=$ ≀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 APost$[\![S_t]\!]$ and APost$[\![S_f]\!]$ do not interfere≀

Implementation

```
matrix_t* _matrix_alloc_int(const int mr, const int nc)
{
  matrix_t* mat = (matrix_t*)malloc(sizeof(matrix_t));
  mat->nbrows = mat->_maxrows = mr;
  mat->nbcolumns = nc;
  mat->_sorted = s;
  if (mr*nc>0){
    int i;
    pkint_t* q;
    mat->_pinit = _vector_alloc_int(mr*nc);
    mat->p = (pkint_t**)malloc(mr * sizeof(pkint_t*));
    q = mat->_pinit;
    for (i=0;i<mr;i++){
      mat->p[i]=q;
```

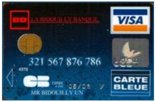
```
void backsubstitute(matrix_t* con, int rank)
{
  int i,j,k;
  for (k=rank-1; k>=0; k--) {
    j = pk_cherni_intp[k];
    for (i=0; i<k; i++) {
      if (pkint_sgn(con->p[i][j]))
        matrix_combine_rows(con,i,k,i,j);
    }
    for (i=k+1; i<con->nbrows; i++) {
      if (pkint_sgn(con->p[i][j]))
        matrix_combine_rows(con,i,k,i,j);
    }}
}
```


Do the two parts talk about the same?

Certified static analyses

A *certified static analysis* is an analysis whose implementation has been formally proved correct using a proof assistant.

Kernel $\longrightarrow$ Proof assistant $\longrightarrow$ Static analysis $\longrightarrow$



- proof assistant : Coq
  - we benefit from the extraction mechanism to prove executable analyser
- proof technique : abstract interpretation
  - general enough to handle a broad range of static analysis
- applications to static analysis of bytecode programs
  - to go beyond the state of the art about Sun's bytecode verifier

Contributions

- We have isolated a fragment of Abstract Interpretation adequate for proving analysis soundness in the constructive logic of Coq
- We have programmed certified generic fixpoint solvers
- We have developed a lattice library allowing an easy construction of complex terminations proofs
- Several cases studies for bytecode Java
  1. A control flow analysis for a representative subset of JavaCard
  2. A memory usage analysis for a representative subset of JavaCard
  3. An interval analysis for an imperative fragment of JavaCard with dynamic arrays (see PCC applications)
- Application to proof carrying code

Outline

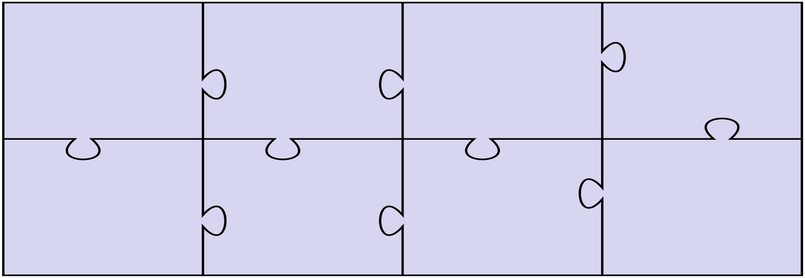
Outline

Outline

Outline

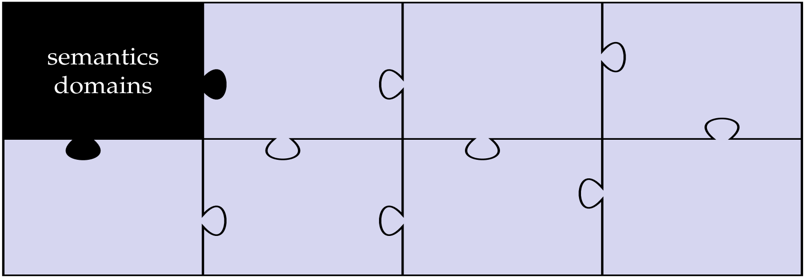
Outline

Building a certified static analyser



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

Building a certified static analyser



Example: JVM states

frame

call stack

$$\langle\!\langle h, \langle m, pc, l, v :: s\rangle, sf\rangle\!\rangle$$

heap

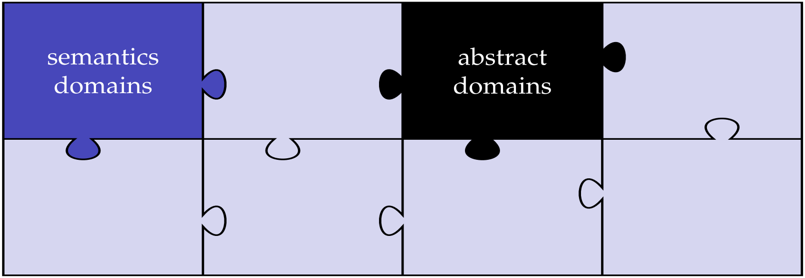
method

program point

operand stack

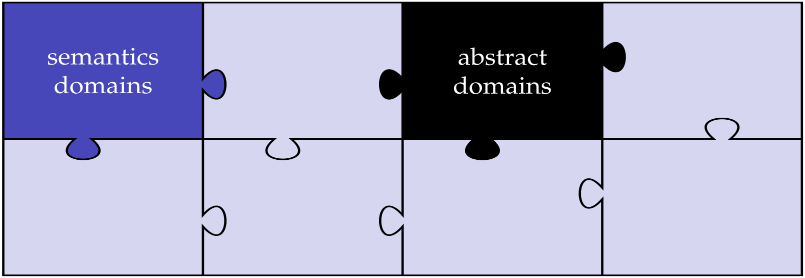
local variables

Building a certified static analyser



- Each semantic sub-domain has its abstract counterpart
- An abstract domain is a lattice $\left(\mathcal{D}^\sharp, =, \sqsubseteq, \bot, \sqcup, \sqcap\right)$ without infinite strictly increasing chains $x_0 \sqsubset x_1 \sqsubset \cdots \sqsubset \cdots$
- First difficult point: how can we quickly develop big lattice structures in Coq ?

Building a certified static analyser



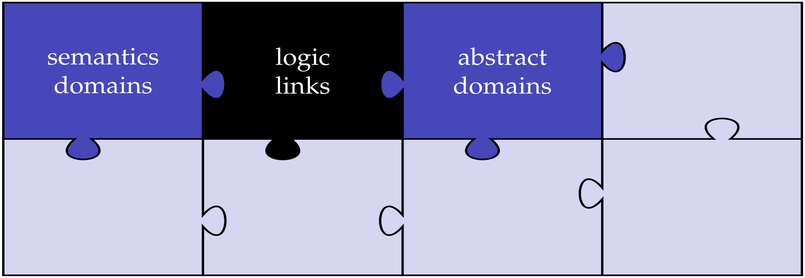
- Each semantic sub-domain has its abstract counterpart
- An abstract domain is a lattice $\left(\mathcal{D}^\sharp, =, \sqsubseteq, \bot, \sqcup, \sqcap\right)$ without infinite strictly increasing chains $x_0 \sqsubset x_1 \sqsubset \cdots \sqsubset \cdots$
- First difficult point: how can we quickly develop big lattice structures in Coq ?
  - generic lattice library

Building lattices in Coq

We propose a technique based on the Coq module system (inspired by the ML module system)

- Lattice requirements are collected in a module contract
- Various functors are proposed in order to build lattices by composition of others
  - Base lattices: signs, congruences, constants, intervals, finite set (implemented with trees)
  - Functors : product, disjoint and lifted sums, lists, functional arrays (implemented with trees)
  - For each functor the most challenging proofs deals with the preservation of the termination criterion.
- The library deals as well with widening/narrowing

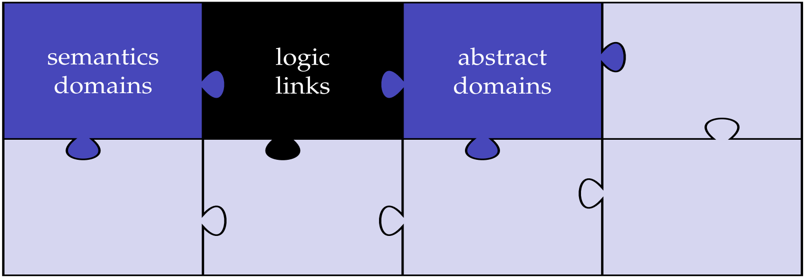
Building a certified static analyser



- Each abstract value represents a property on concrete values
- This correspondence is formalised by a monotone concretisation function

$$\gamma : (\mathcal{D}^\sharp, \sqsubseteq) \longrightarrow_m (\wp(\mathcal{D}), \subseteq)$$

Building a certified static analyser

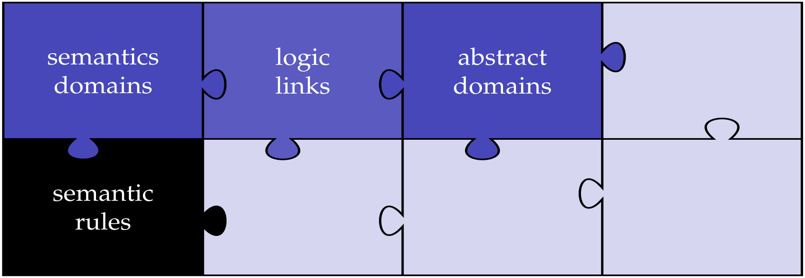


- Each abstract value represents a property on concrete values
- This correspondence is formalised by a monotone concretisation function

$$\gamma : (\mathcal{D}^\sharp, \sqsubseteq) \longrightarrow_m (\wp(\mathcal{D}), \subseteq)$$

$x \subseteq \gamma(x^\sharp)$ means "$x^\sharp$ is a correct approximation of x"

Building a certified static analyser



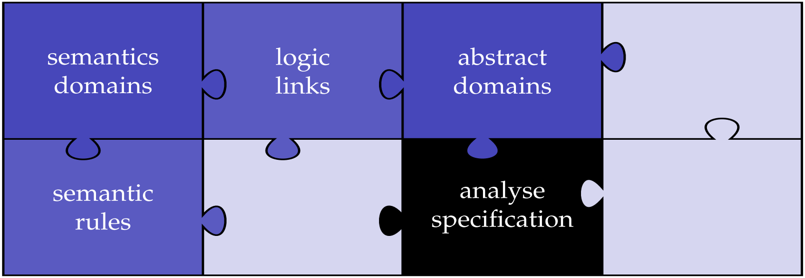
- operational semantics $\cdot \rightarrow_P \cdot$ between states
- collecting semantics: $[\![P]\!] = \{\, s \mid \exists s_0 \in S_{\mathrm{init}},\ s_0 \rightarrow_P^* s \,\}$
- we want to compute a correct approximation of $[\![P]\!]$
  - a sound invariant $s^\sharp$ on the reachable states : $[\![P]\!] \subseteq \gamma(s^\sharp)$

Example : JVM operational semantics

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{push}\ c}{\langle\!\langle h, \langle m, pc, l, s\rangle, sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m, pc+1, l, c :: s\rangle, sf\rangle\!\rangle}$$

$$\frac{\begin{array}{c}\text{instructionAt}_P(m, pc) = \texttt{invokevirtual}\ m_{id} \\ \begin{array}{rcl} m' & = & \text{methodLookup}(m_{id}, h(loc)) \\ V & = & v_1 :: \cdots :: v_{\text{nbArguments}(m_{id})}\end{array}\end{array}}{\langle\!\langle h, \langle m, pc, l, loc :: V :: s\rangle, sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m', 1, V, \varepsilon\rangle, \langle m, pc, l, s\rangle :: sf\rangle\!\rangle}$$

Building a certified static analyser



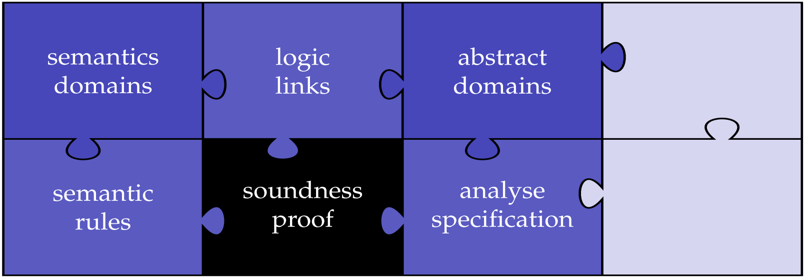
- the analysis is specified as a solution of a post fixpoint problem

$$F_P^\sharp(s^\sharp) \sqsubseteq^\sharp s^\sharp$$

- after partitioning : constraint system

$$\begin{cases} f_1^\sharp(s_1^\sharp,\ldots,s_n^\sharp) & \sqsubseteq^\sharp & s_{i_1}^\sharp \\ & \cdots & \\ f_n^\sharp(s_1^\sharp,\ldots,s_n^\sharp) & \sqsubseteq^\sharp & s_{i_n}^\sharp \end{cases}$$

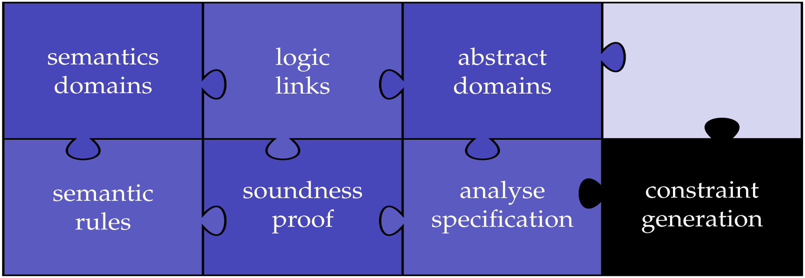
Building a certified static analyser



$$\forall P,\ \forall s^\sharp,\quad F_P^\sharp(s^\sharp) \sqsubseteq^\sharp s^\sharp \ \Rightarrow\ [\![P]\!] \subseteq \gamma(s^\sharp)$$

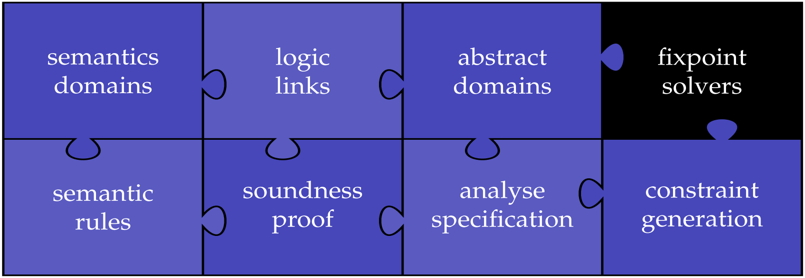
- easy proof, but tedious
- one proof by instruction: a long work for real langages

Building a certified static analyser



- collects all constraints in a program
- generic tool

Building a certified static analyser



$$\forall P,\ \exists s^\sharp,\quad F_P^\sharp(s^\sharp) \sqsubseteq s^\sharp$$

Two techniques of iterative computation

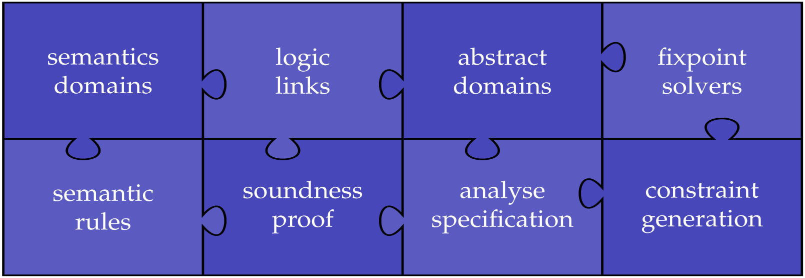
- traditional least (post)-fixpoint computation

$$\bot \rightarrow F_P^\sharp(\bot) \rightarrow {F_P^\sharp}^2(\bot) \rightarrow \cdots \mathrm{lfp}(F_P^\sharp)$$

- post-fixpoint computation by widening/narrowing with chaotic iterations

In the two cases, a generic tool

Building a certified static analyser



Final result

$$\left.\begin{array}{l}\forall P,\ \forall s^\sharp,\ F^\sharp_P(s^\sharp) \sqsubseteq s^\sharp \Rightarrow [\![P]\!] \subseteq \gamma(s^\sharp) \\ \forall P,\ \exists s^\sharp,\ F^\sharp_P(s^\sharp) \sqsubseteq s^\sharp\end{array}\right\} \quad \forall P,\ \exists s^\sharp,\ [\![P]\!] \subseteq \gamma(s^\sharp)$$

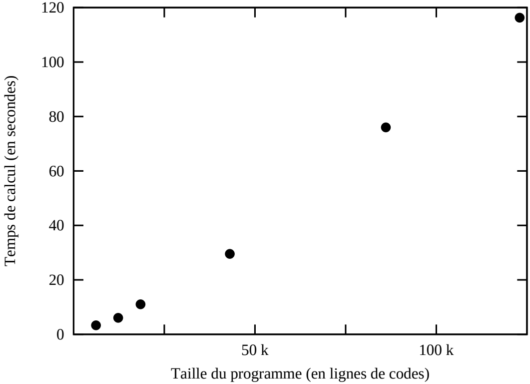
In Coq : `analyse : ∀ p:program, { s:abstate | sem(P) ⊆ gamma(P,s) }`

In Caml : `analyse : program → abstate`

Efficiency of the extracted program

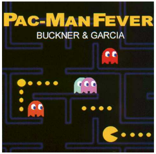
Experiments on the memory usage analysis

$$\mathcal{D}^\sharp = (\mathbb{M} \to \mathcal{P}(\mathbb{M})) \times \mathcal{P}(\mathbb{M}) \times (\mathbb{M} \to \mathbb{P} \to \mathcal{P}(\mathbb{P})) \times \mathcal{P}(\mathbb{M})$$



Outline

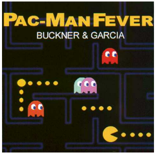
Mobile code dilemmas...



Mobile code dilemmas...

Untrusted code

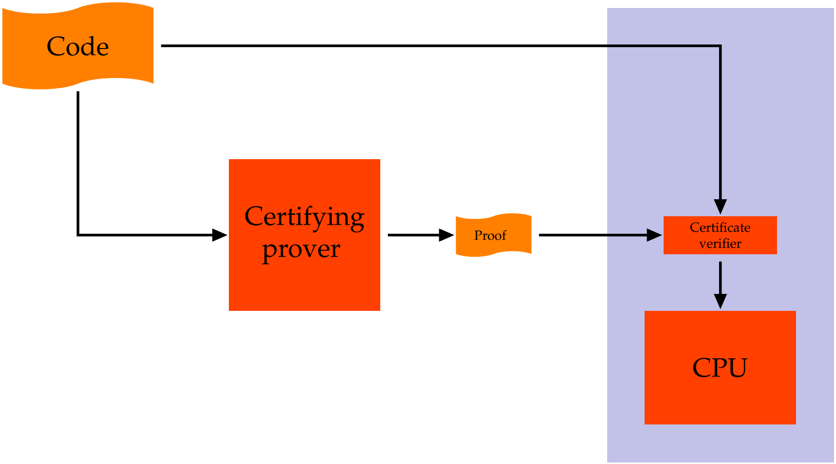
Host system



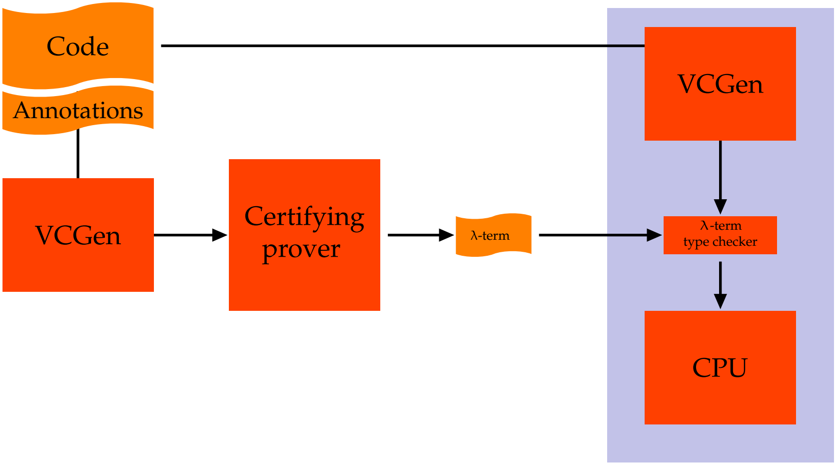
Safe ?

- The untrusted code may cause damages on the system
  - intern structure corruption
- The untrusted code may use too many resources
  - CPU, memory, SMS...

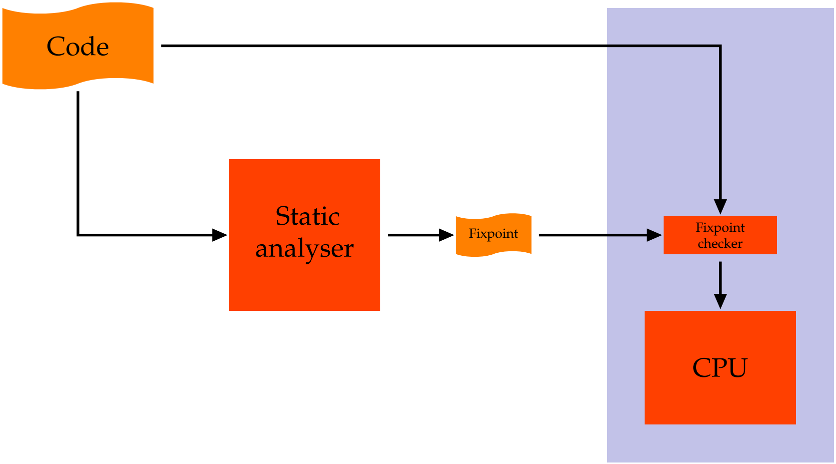
Proof carrying code



Proof carrying code: standard framework



Proof carrying code by abstract interpretation



Proof carrying code by abstract interpretation

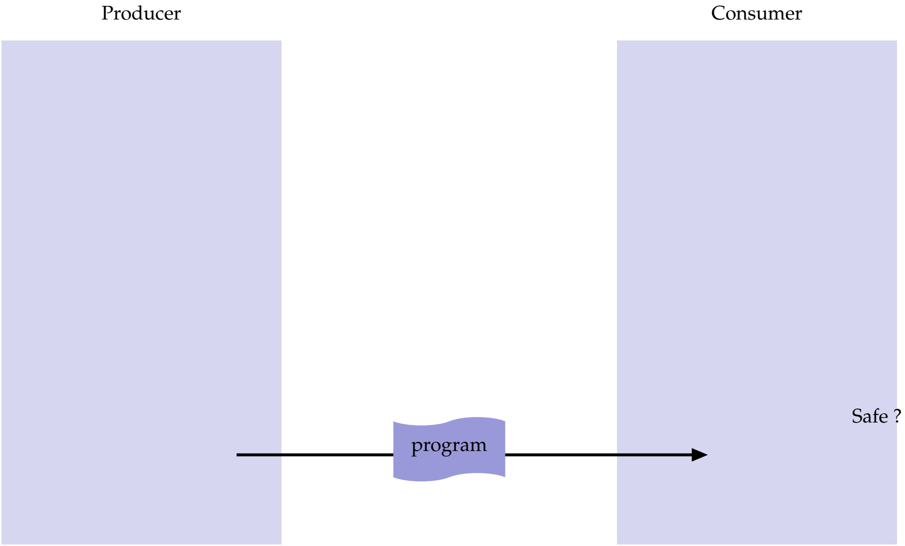
PCC requirements:

- the certificate must be small
- the verifier must be efficient
- soundness of the certifying prover is not critical
- soundness of the verifier is critical

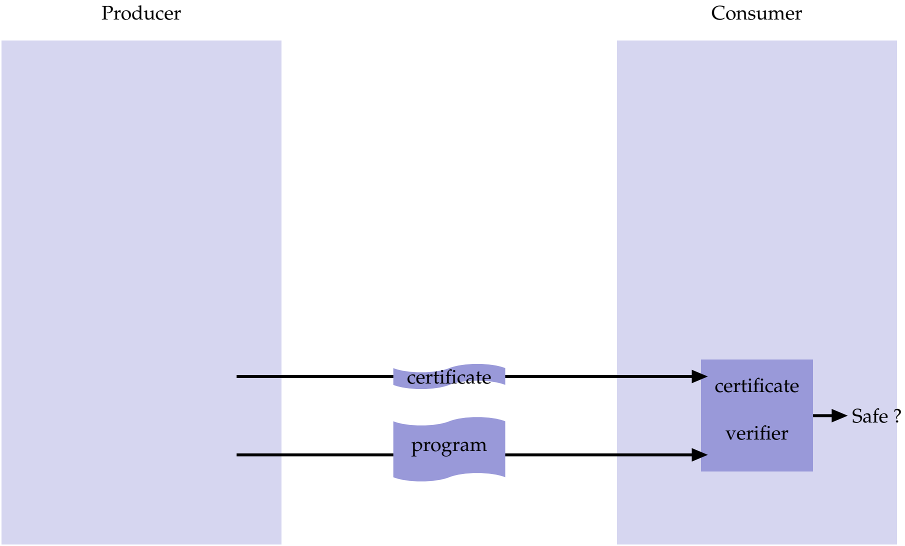
Our works on certified static analysis allows the consumer to check semantics soundness of the verifier

- certified proof carrying code = certified static analysis without fixpoint solver

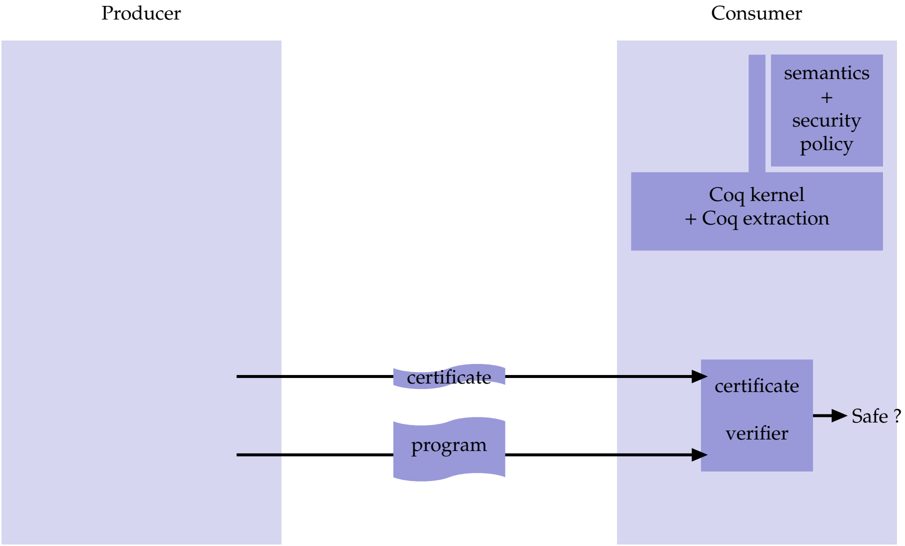
Proof carrying code by certified abstract interpretation



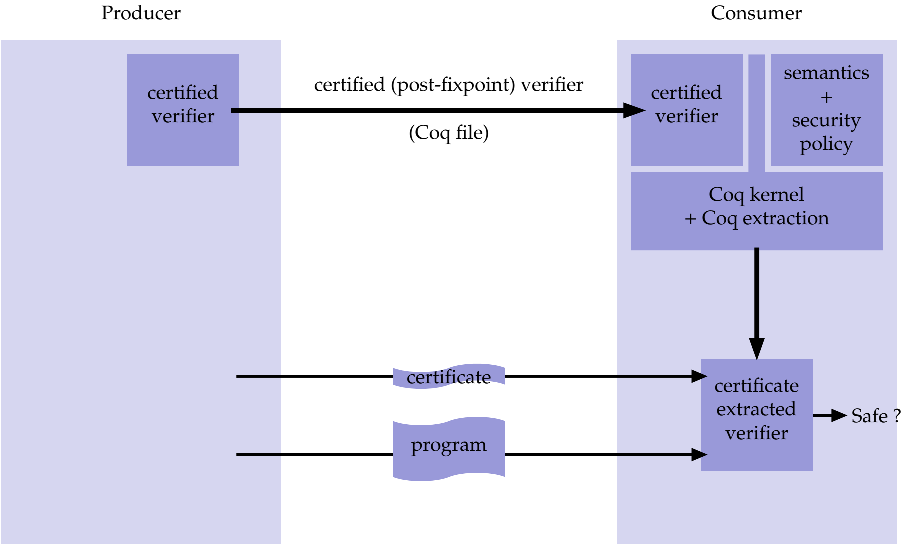
Proof carrying code by certified abstract interpretation



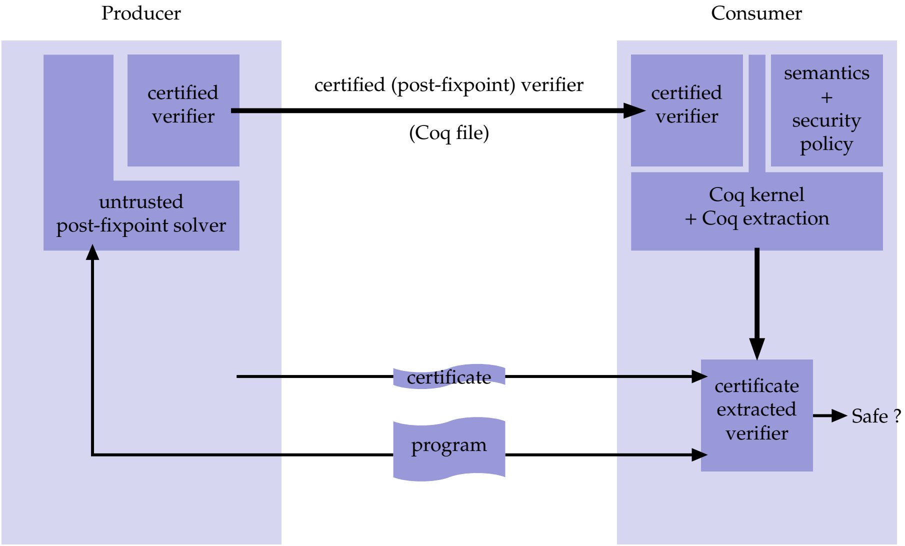
Proof carrying code by certified abstract interpretation



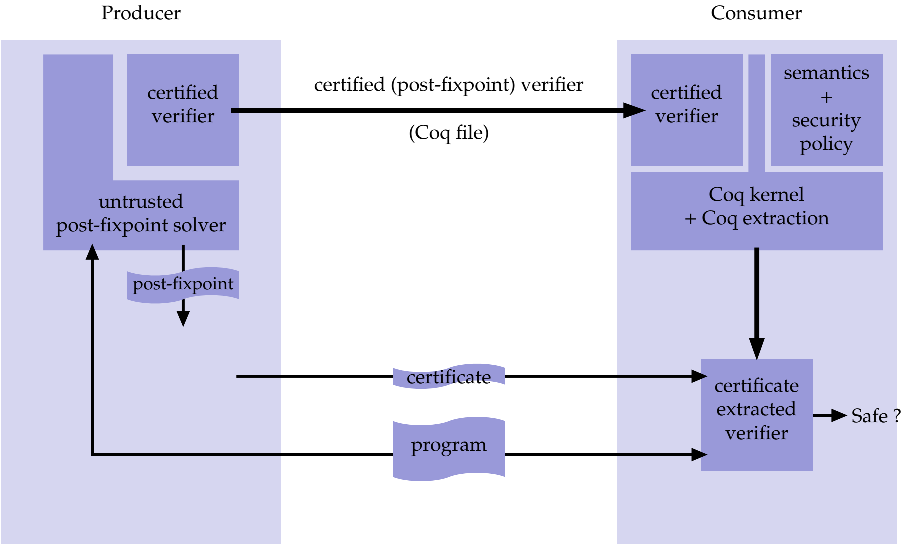
Proof carrying code by certified abstract interpretation



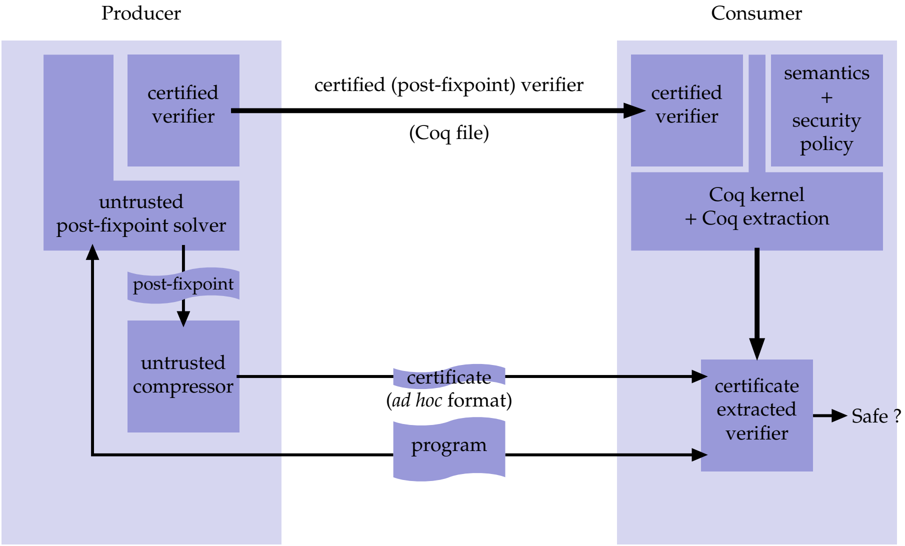
Proof carrying code by certified abstract interpretation



Proof carrying code by certified abstract interpretation



Proof carrying code by certified abstract interpretation



First case study

- imperative fragment of bytecode Java + dynamically allocated arrays
- we verify arrays access (no out of bounds errors) by intervals approximation
- experiments on various algorithms[1]:

Program	.java size	.class size	certificate size	checking time
BubbleSort	440	528	32	0.015
HeapSort	1044	858	63	0.050
QuickSort	1078	965	124	0.060
ConvolutionProduct	378	542	52	0.010
FloydWharshall	417	596	134	0.020
PolynomProduct	509	604	87	0.010

certificate/programme ~ 10%, verification time <0.1s

[1]Size file in bytes, time in secs

Outline

Perspectives

Increasing the power of certified static analyses

- symbolic manipulation
- relational abstract domains (octagons, polyhedra)

Proof-carrying code

- fixpoint compression (see ESOP'07)
- we want to demonstrate the scalability of our architecture

Pre-analyses for others verification tools

- automatically detect some unthrowable exceptions
- alias analysis
- race detection