Verification of Imperative Programs through Transformation of Constraint Logic Programs

Emanuele De Angelis[1], Fabio Fioravanti[1],
Alberto Pettorossi[2], and Maurizio Proietti[3]

[1]University of Chieti-Pescara 'G. d'Annunzio'
[2]University of Rome 'Tor Vergata'
[3]CNR - Istituto di Analisi dei Sistemi ed Informatica, Rome

VPT @CAV2013
Saint Petersburg, Russia, 13 July 2013

Outline

- Constraint Logic Programming as a metalanguage for representing
  - the imperative program (integer and array variables)
  - the semantics of the imperative language (i.e. the interpreter)
  - the properties to be verified (not only reachability)
- Verification Method based on CLP Program Transformation
  - Semantics-preserving unfold/fold rules (and strategies)
  - Remove the interpreter by specialization
  - Propagate the initial or error properties
  - Iterate
- The verification method at work
  - Array Maximum
    - theory of arrays
  - Greatest Common Divisor
    - specifications given by recursive CLP clauses

Outline

- Constraint Logic Programming as a metalanguage for representing
  - the imperative program (integer and array variables)
  - the semantics of the imperative language (i.e. the interpreter)
  - the properties to be verified (not only reachability)
- Verification Method based on CLP Program Transformation
  - Semantics-preserving unfold/fold rules (and strategies)
  - Remove the interpreter by specialization
  - Propagate the initial or error properties
  - Iterate
- The verification method at work
  - Array Maximum
    - theory of arrays
  - Greatest Common Divisor
    - specifications given by recursive CLP clauses

Outline

- Constraint Logic Programming as a metalanguage for representing
  - the imperative program (integer and array variables)
  - the semantics of the imperative language (i.e. the interpreter)
  - the properties to be verified (not only reachability)
- Verification Method based on CLP Program Transformation
  - Semantics-preserving unfold/fold rules (and strategies)
  - Remove the interpreter by specialization
  - Propagate the initial or error properties
  - Iterate
- The verification method at work
  - Array Maximum
    - theory of arrays
  - Greatest Common Divisor
    - specifications given by recursive CLP clauses

Program (Partial) Correctness

The ArrayMax Program

```
while(i < n) {
    if(max < a[i])
       max=a[i];
    i=i+1;
  }
```

Initial and error properties

$\varphi_{init}(i,n,a,max) \equiv$
$\quad i=0 \wedge n=dim(a) \wedge n\geq 1 \wedge max=a[i]$

$\varphi_{error}(n,a,max) \equiv$
$\quad \exists k\ (0\leq k<n\ \wedge\ a[k]>max)$

Definition (Partial Correctness)

A program P is correct w.r.t. φ_{init} and φ_{error} if
from any configuration satisfying φ_{init}
no final configuration satisfying φ_{error} can be reached.

Otherwise, program P is incorrect.

Program (Partial) Correctness

The ArrayMax Program

```
while(i < n) {
    if(max < a[i])
       max=a[i];
    i=i+1;
  }
```

Initial and error properties

$\varphi_{init}(i,n,a,max) \equiv$
$\quad i = 0 \wedge n = dim(a) \wedge n \geq 1 \wedge max = a[i]$

$\varphi_{error}(n,a,max) \equiv$
$\quad \exists k\ (0 \leq k < n\ \wedge\ a[k] > max)$

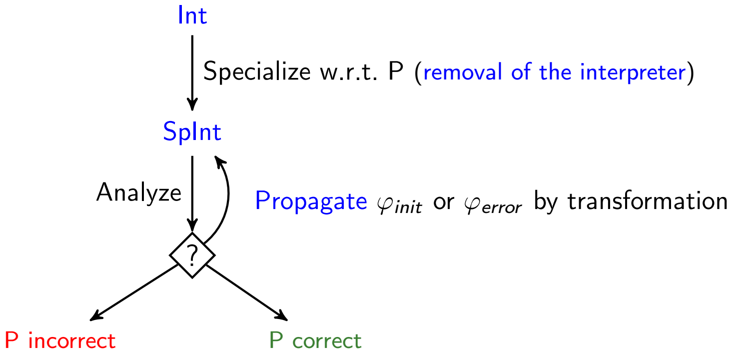
Definition (Partial Correctness)

A program P is correct w.r.t. φ_{init} and φ_{error} if
from any configuration satisfying φ_{init}
no final configuration satisfying φ_{error} can be reached.

Otherwise, program P is incorrect.

Verification Framework

- P is the (CLP encoding of the) imperative program to be verified
- Int encodes the semantics of the language, and
- φ_{init} and φ_{error} are the initial and error properties



CLP Encoding of imperative programs

Program ArrayMax

ℓ_0: `while(i < n) {`
ℓ_1: `    if(max < a[i])`
ℓ_2: `        max=a[i];`
ℓ_3: `    i=i+1;`
ℓ_h: `}`

CLP encoding of program ArrayMax

$\mathtt{at}(\ell_0, \mathtt{ite}(\mathtt{less}(\mathtt{int}(\mathtt{i}), \mathtt{int}(\mathtt{n})), \ell_1, \ell_\mathrm{h})).$
$\mathtt{at}(\ell_1, \mathtt{ite}(\mathtt{less}(\mathtt{int}(\mathtt{max}), \mathtt{read}(\mathtt{array}(\mathtt{a}), \mathtt{int}(\mathtt{i}))), \ell_2, \ell_3)).$
$\mathtt{at}(\ell_2, \mathtt{asgn}(\mathtt{int}(\mathtt{max}), \mathtt{read}(\mathtt{array}(\mathtt{a}), \mathtt{int}(\mathtt{i})))).$
$\mathtt{at}(\ell_3, \mathtt{asgn}(\mathtt{int}(\mathtt{i}), \mathtt{plus}(\mathtt{int}(\mathtt{i}), 1))).$
$\mathtt{at}(\ell_4, \mathtt{goto}(\ell_0)).$
$\mathtt{at}(\ell_\mathrm{h}, \mathtt{halt}).$

CLP Encoding of imperative programs

Program ArrayMax

ℓ_0: `while(i < n) {`
ℓ_1: `    if(max < a[i])`
ℓ_2: `        max=a[i];`
ℓ_3: `    i=i+1;`
ℓ_h: `}`

CLP encoding of program ArrayMax

$\mathtt{at}(\ell_0, \mathtt{ite}(\mathtt{less}(\mathtt{int}(\mathtt{i}), \mathtt{int}(\mathtt{n})), \ell_1, \ell_\mathrm{h})).$

$\mathtt{at}(\ell_1, \mathtt{ite}(\mathtt{less}(\mathtt{int}(\mathtt{max}), \mathtt{read}(\mathtt{array}(\mathtt{a}), \mathtt{int}(\mathtt{i}))), \ell_2, \ell_3)).$

$\mathtt{at}(\ell_2, \mathtt{asgn}(\mathtt{int}(\mathtt{max}), \mathtt{read}(\mathtt{array}(\mathtt{a}), \mathtt{int}(\mathtt{i})))).$

$\mathtt{at}(\ell_3, \mathtt{asgn}(\mathtt{int}(\mathtt{i}), \mathtt{plus}(\mathtt{int}(\mathtt{i}), 1))).$

$\mathtt{at}(\ell_4, \mathtt{goto}(\ell_0)).$

$\mathtt{at}(\ell_\mathrm{h}, \mathtt{halt}).$

Encoding the interpreter of the imperative language (1)

- a set of configurations: $\texttt{cf}(\texttt{C},\texttt{S})$ (○)

  A configuration is made out of :

  - a command C to be executed
  - an environment S: a list of [variable, value] pairs

    for instance: $[[\texttt{int}(\texttt{x}), \texttt{11}],\ [\texttt{int}(\texttt{y}), \texttt{7}]]$

- a transition relation: $\texttt{tr}(\texttt{cf}(\texttt{C},\texttt{S}),\ \texttt{cf}(\texttt{C1},\texttt{S1}))$ (→)

  (i.e., the operational semantics)

Encoding the interpreter of the imperative language (1)

- a set of configurations: $\texttt{cf}(\texttt{C},\texttt{S})$ ($\bigcirc$)

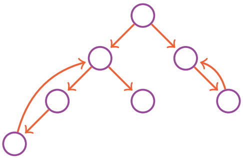
  A configuration is made out of :

  - a command C to be executed
  - an environment S: a list of [variable, value] pairs

    for instance: $[[\texttt{int}(\texttt{x}),11],\ [\texttt{int}(\texttt{y}),7]]$

- a transition relation: $\texttt{tr}(\texttt{cf}(\texttt{C},\texttt{S}),\ \texttt{cf}(\texttt{C1},\texttt{S1}))$ ($\rightarrow$)

  (i.e., the operational semantics)



`Id = Expr;`	`tr(cf(L,asgn(Id,Expr),S),cf(C,S1)) :-` `aeval(Expr,S,V), update(Id,V,S,S1), nextlab(L,C).`

`if (Expr) {`
`goto L1;`
`} else`
`goto L2;`
`}`
	`tr(cf(ite(Expr,L1,L2),S),cf(C,S)) :-`
	`beval(Expr,S), at(L1,C).`
	`tr(cf(ite(Expr,L1,L2),S),cf(C,S)) :-`
	`beval(not(Expr),S), at(L2,C).`

`goto L;`	`tr(cf(goto(L),S),cf(C,S)) :- at(L,C).`

CLP Encoding
Initial and error configurations

Initial configuration

$$\texttt{initConf}(\texttt{cf}(\overline{\texttt{cmd}(\texttt{i},\texttt{C}),} \\ [[\texttt{int}(\texttt{i}),\texttt{I}],[\texttt{int}(\texttt{n}),\texttt{N}],[\texttt{array}(\texttt{a}),(\texttt{A},\texttt{N})],[\texttt{int}(\texttt{max}),\texttt{Max}]])) \\ \texttt{:-}\ \texttt{at}(\texttt{i},\texttt{C}),\texttt{phiInit}(\texttt{I},\texttt{N},\texttt{A},\texttt{Max}).$$

$$\texttt{phiInit}(\texttt{I},\texttt{N},\texttt{A},\texttt{Max})\ \texttt{:-}\ \texttt{I}=0,\ \texttt{N}\geq 1,\ \texttt{read}((\texttt{A},\texttt{N}),\texttt{I},\texttt{Max}).$$

Error configuration

$$\texttt{errorConf}(\texttt{cf}(\overline{\texttt{cmd}(\texttt{h},\texttt{C}),} \\ [[\texttt{int}(\texttt{i}),\texttt{I}],[\texttt{int}(\texttt{n}),\texttt{N}],[\texttt{array}(\texttt{a}),(\texttt{A},\texttt{N})],[\texttt{int}(\texttt{max}),\texttt{Max}]])) \\ \texttt{:-}\ \texttt{at}(\texttt{h},\texttt{C}),\texttt{phiError}(\texttt{N},\texttt{A},\texttt{Max}).$$

$$\texttt{phiError}(\texttt{N},\texttt{A},\texttt{Max})\ \texttt{:-}\ \texttt{K}\geq 0,\ \texttt{N}>\texttt{K},\ \texttt{Z}>\texttt{Max},\ \texttt{read}((\texttt{A},\texttt{N}),\texttt{K},\texttt{Z}).$$

CLP Encoding
Initial and error configurations

Initial configuration

```
initConf(cf(cmd(i, C),
    [[int(i), I], [int(n), N], [array(a), (A, N)], [int(max), Max]]))
  :- at(i, C), phiInit(I, N, A, Max).

phiInit(I, N, A, Max) :- I = 0,  N ≥ 1,  read((A, N), I, Max).
```

Error configuration

```
errorConf(cf(cmd(h, C),
    [[int(i), I], [int(n), N], [array(a), (A, N)], [int(max), Max]]))
  :- at(h, C), phiError(N, A, Max).

phiError(N, A, Max) :- K ≥ 0,  N > K,  Z > Max,  read((A, N), K, Z).
```

CLP Encoding

The interpreter (Int)

```
incorrect :- initConf(A), reach(A).
reach(A) :- tr(A,B), reach(B).
reach(A) :- errorConf(A).
```

+ clauses for `tr` (i.e., the interpreter of the imp. language)

+ clauses for `at` (i.e., the given program P)

+ clauses for `initConf` and `errorConf`
  (i.e., the initial and error configurations)

Theorem (Correctness of the CLP encoding)

Program P is correct iff the atom `incorrect` *does not belong to the least model M*(Int) *of the CLP program Int.*

CLP Encoding

The interpreter (Int)

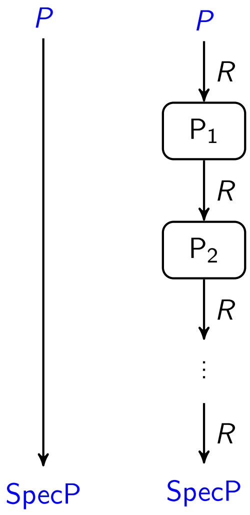
```
incorrect :- initConf(A), reach(A).
reach(A) :- tr(A,B), reach(B).
reach(A) :- errorConf(A).
```

+ clauses for `tr` (i.e., the interpreter of the imp. language)
+ clauses for `at` (i.e., the given program P)
+ clauses for `initConf` and `errorConf`
  (i.e., the initial and error configurations)

Theorem (Correctness of the CLP encoding)

Program P is correct iff the atom `incorrect` *does not belong to the least model M*(Int) *of the CLP program Int.*

Unfold/Fold Program Transformation



- transformation rules:

  $R \in \{$ Conjunctive Definition ,
  Unfolding,
  Conjunctive Folding ,
  Goal Replacement,
  Clause Removal $\}$

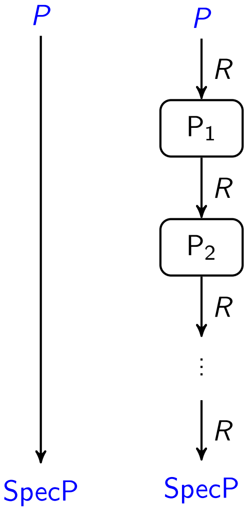
- transformation rules preserve the semantics:

  incorrect $\in M(\text{P})$ *iff* incorrect $\in M(\text{SpecP})$

- transformation strategy :

  $($ Unf ; Goal Repl ; Clause Rem ; Def ; Fold $)^*$

Unfold/Fold Program Transformation



- transformation rules:

  $R \in \{$ Conjunctive Definition ,
  Unfolding,
  Conjunctive Folding ,
  Goal Replacement,
  Clause Removal $\}$

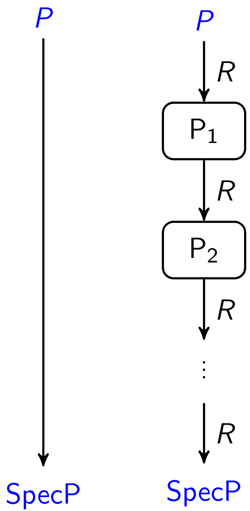
- transformation rules preserve the semantics:

  `incorrect` $\in M(\mathrm{P})$ *iff* `incorrect` $\in M(\mathrm{SpecP})$

- transformation strategy :

  (Unf ; Goal Repl ; Clause Rem ; Def ; Fold) *

Unfold/Fold Program Transformation



- transformation rules:

  $R \in \{$ Conjunctive Definition ,
  Unfolding,
  Conjunctive Folding ,
  Goal Replacement,
  Clause Removal $\}$

- transformation rules preserve the semantics:

  `incorrect` $\in M(\text{P})$ *iff* `incorrect` $\in M(\text{SpecP})$

- transformation strategy :

  (Unf ; Goal Repl ; Clause Rem ; Def ; Fold) *

Rules for Specializing CLP Programs

R1. Conjunctive definition : $\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{G}$ where $\mathtt{G} \equiv \mathtt{A_1} \land \ldots \land \mathtt{A_n}$

R2. Unfolding : $\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{L} \land \underline{\mathtt{A}} \land \mathtt{R}$

$$\underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_1} \land \mathtt{A_1}}, \ldots, \underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_m} \land \mathtt{A_m}}$$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{d_1} \land \mathtt{L} \land \mathtt{A_1} \land \mathtt{R}, \ldots, \mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{d_m} \land \mathtt{L} \land \mathtt{A_m} \land \mathtt{R}$$

R3. Conjunctive Folding : $\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{L} \land \underline{\mathtt{G}} \land \mathtt{R}$

$\underline{\mathtt{newq(X)}} \leftarrow \mathtt{d} \land \underline{\mathtt{G}}$ and $\mathtt{c} \rightarrow \mathtt{d}$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{L} \land \underline{\mathtt{newq(X)}} \land \mathtt{R}$$

R4. Clause Removal : clauses with unsatisfiable constraint or subsumed

---

R5. Goal Replacement : if $M(P \cup \mathcal{T}) \models \forall(\mathtt{c_1} \land \mathtt{G_1} \leftrightarrow \mathtt{c_2} \land \mathtt{G_2})$

then replace $\mathtt{H} \leftarrow \mathtt{c} \land \underline{\mathtt{c_1}} \land \mathtt{L} \land \underline{\mathtt{G_1}} \land \mathtt{R}$ with $\mathtt{H} \leftarrow \mathtt{c} \land \underline{\mathtt{c_2}} \land \mathtt{L} \land \underline{\mathtt{G_2}} \land \mathtt{R}$

Rules for Specializing CLP Programs

R1. Conjunctive definition : $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{G}$ where $\mathtt{G} \equiv \mathtt{A_1} \wedge \ldots \wedge \mathtt{A_n}$

R2. Unfolding : $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{A}} \wedge \mathtt{R}$

$$\underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_1} \wedge \mathtt{A_1}}, \ldots, \underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_m} \wedge \mathtt{A_m}}$$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{d_1} \wedge \mathtt{L} \wedge \mathtt{A_1} \wedge \mathtt{R}, \ldots, \mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{d_m} \wedge \mathtt{L} \wedge \mathtt{A_m} \wedge \mathtt{R}$$

R3. Conjunctive Folding : $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{G}} \wedge \mathtt{R}$

$\underline{\mathtt{newq(X)}} \leftarrow \mathtt{d} \wedge \underline{\mathtt{G}}$ and $\mathtt{c} \rightarrow \mathtt{d}$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{newq(X)}} \wedge \mathtt{R}$$

R4. Clause Removal : clauses with unsatisfiable constraint or subsumed

---

R5. Goal Replacement : if $M(P \cup \mathcal{T}) \models \forall (\mathtt{c_1} \wedge \mathtt{G_1} \leftrightarrow \mathtt{c_2} \wedge \mathtt{G_2})$

then replace $\mathtt{H} \leftarrow \mathtt{c} \wedge \underline{\mathtt{c_1}} \wedge \mathtt{L} \wedge \underline{\mathtt{G_1}} \wedge \mathtt{R}$ with $\mathtt{H} \leftarrow \mathtt{c} \wedge \underline{\mathtt{c_2}} \wedge \mathtt{L} \wedge \underline{\mathtt{G_2}} \wedge \mathtt{R}$

Rules for Specializing CLP Programs

R1. Conjunctive definition : $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{G}$ where $\mathtt{G} \equiv \mathtt{A_1} \wedge \ldots \wedge \mathtt{A_n}$

R2. Unfolding : $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{A}} \wedge \mathtt{R}$

$$\underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_1} \wedge \mathtt{A_1}}, \ldots, \underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_m} \wedge \mathtt{A_m}}$$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{d_1} \wedge \mathtt{L} \wedge \mathtt{A_1} \wedge \mathtt{R}, \ldots, \mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{d_m} \wedge \mathtt{L} \wedge \mathtt{A_m} \wedge \mathtt{R}$$

R3. Conjunctive Folding : $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{G}} \wedge \mathtt{R}$

$$\underline{\mathtt{newq(X)}} \leftarrow \mathtt{d} \wedge \underline{\mathtt{G}} \quad \text{and} \quad \mathtt{c} \rightarrow \mathtt{d}$$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{newq(X)}} \wedge \mathtt{R}$$

R4. Clause Removal : clauses with unsatisfiable constraint or subsumed

---

R5. Goal Replacement : if $M(P \cup \mathcal{T}) \models \forall(\mathtt{c_1} \wedge \mathtt{G_1} \leftrightarrow \mathtt{c_2} \wedge \mathtt{G_2})$

then replace $\mathtt{H} \leftarrow \mathtt{c} \wedge \underline{\mathtt{c_1}} \wedge \mathtt{L} \wedge \underline{\mathtt{G_1}} \wedge \mathtt{R}$ with $\mathtt{H} \leftarrow \mathtt{c} \wedge \underline{\mathtt{c_2}} \wedge \mathtt{L} \wedge \underline{\mathtt{G_2}} \wedge \mathtt{R}$

Rules for Specializing CLP Programs

R1. Conjunctive definition: $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{G}$ where $\mathtt{G} \equiv \mathtt{A_1} \wedge \ldots \wedge \mathtt{A_n}$

R2. Unfolding: $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{A}} \wedge \mathtt{R}$

$$\underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_1} \wedge \mathtt{A_1}}, \ldots, \underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_m} \wedge \mathtt{A_m}}$$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{d_1} \wedge \mathtt{L} \wedge \mathtt{A_1} \wedge \mathtt{R}, \ldots, \mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{d_m} \wedge \mathtt{L} \wedge \mathtt{A_m} \wedge \mathtt{R}$$

R3. Conjunctive Folding: $\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{G}} \wedge \mathtt{R}$

$$\underline{\mathtt{newq(X)}} \leftarrow \mathtt{d} \wedge \underline{\mathtt{G}} \quad \text{and} \quad \mathtt{c} \rightarrow \mathtt{d}$$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \wedge \mathtt{L} \wedge \underline{\mathtt{newq(X)}} \wedge \mathtt{R}$$

R4. Clause Removal: clauses with unsatisfiable constraint or subsumed

---

R5. Goal Replacement: if $M(P \cup \mathcal{T}) \models \forall (\mathtt{c_1} \wedge \mathtt{G_1} \leftrightarrow \mathtt{c_2} \wedge \mathtt{G_2})$

then replace $\mathtt{H} \leftarrow \mathtt{c} \wedge \underline{\mathtt{c_1}} \wedge \mathtt{L} \wedge \underline{\mathtt{G_1}} \wedge \mathtt{R}$ with $\mathtt{H} \leftarrow \mathtt{c} \wedge \underline{\mathtt{c_2}} \wedge \mathtt{L} \wedge \underline{\mathtt{G_2}} \wedge \mathtt{R}$

Rules for Specializing CLP Programs

R1. Conjunctive definition : $\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{G}$ where $\mathtt{G} \equiv \mathtt{A_1} \land \ldots \land \mathtt{A_n}$

R2. Unfolding : $\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{L} \land \underline{\mathtt{A}} \land \mathtt{R}$

$$\underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_1} \land \mathtt{A_1}}, \ldots, \underline{\mathtt{A}} \leftarrow \underline{\mathtt{d_m} \land \mathtt{A_m}}$$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{d_1} \land \mathtt{L} \land \mathtt{A_1} \land \mathtt{R}, \ldots, \mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{d_m} \land \mathtt{L} \land \mathtt{A_m} \land \mathtt{R}$$

R3. Conjunctive Folding : $\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{L} \land \underline{\mathtt{G}} \land \mathtt{R}$

$\underline{\mathtt{newq(X)}} \leftarrow \mathtt{d} \land \underline{\mathtt{G}}$ and $\mathtt{c} \rightarrow \mathtt{d}$

yields

$$\mathtt{newp(X)} \leftarrow \mathtt{c} \land \mathtt{L} \land \underline{\mathtt{newq(X)}} \land \mathtt{R}$$

R4. Clause Removal : clauses with unsatisfiable constraint or subsumed

---

R5. Goal Replacement : if $M(P \cup \mathcal{T}) \models \forall(\mathtt{c_1} \land \mathtt{G_1} \leftrightarrow \mathtt{c_2} \land \mathtt{G_2})$

then replace $\mathtt{H} \leftarrow \mathtt{c} \land \underline{\mathtt{c_1}} \land \mathtt{L} \land \underline{\mathtt{G_1}} \land \mathtt{R}$ with $\mathtt{H} \leftarrow \mathtt{c} \land \underline{\mathtt{c_2}} \land \mathtt{L} \land \underline{\mathtt{G_2}} \land \mathtt{R}$

Transformation strategy

Transform(P)

SpecP $= \emptyset$;
Def $= \{$`incorrect :- init(A), reach(A)`$\}$;
while $\exists q \in$ Def **do**
  Cls = Unfold(q);
  Cls = Goal Replacement(Cls);
  Cls = Clause Removal(Cls);
  Def = (Def $-$ $\{q\}$) $\cup$ Define(Cls);
  SpecP = SpecP $\cup$ Fold(Cls, Def);
od

Theorem (Correctness of the Transformation Strategy)

`incorrect` $\in M$(P) *iff* `incorrect` $\in M$(SpecP)

Transformation strategy

Transform(P)

SpecP $= \emptyset$;
Def $= \{$`incorrect :- init(A), reach(A)`$\}$;
while $\exists q \in$ Def **do**
  Cls $=$ Unfold(q);
  Cls $=$ Goal Replacement(Cls);
  Cls $=$ Clause Removal(Cls);
  Def $=$ (Def $- \{q\}) \cup$ Define(Cls);
  SpecP $=$ SpecP $\cup$ Fold(Cls, Def);
od

Theorem (Correctness of the Transformation Strategy)

`incorrect` $\in M(\mathrm{P})$ *iff* `incorrect` $\in M(\mathrm{SpecP})$

Removal of the Interpreter

Compile away the interpreter, i.e., remove all references to:

- `tr` (i.e., the operational semantics of the imperative language)
- `at` (i.e., the encoding of P)

The Specialized Interpreter (SpInt) for ArrayMax

```
incorrect :- I=0, N≥1, read((A,N),I,Max), new1(I,N,A,Max).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M>Max, read((A,N),I,M),
                   new1(I1,N,A,M).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M≤Max, read((A,N),I,M),
                   new1(I1,N,A,Max).
new1(I,N,A,Max) :- I≥N, K≥0, N>K, Z>Max, read((A,N),K,Z).
```

Removal of the Interpreter

Compile away the interpreter, i.e., remove all references to:

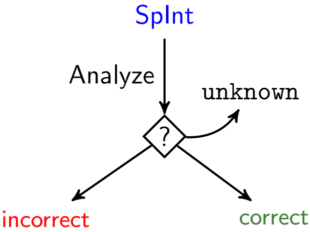
- tr (i.e., the operational semantics of the imperative language)
- at (i.e., the encoding of P)

The Specialized Interpreter (SpInt) for ArrayMax

```
incorrect :- I=0, N≥1, read((A,N),I,Max), new1(I,N,A,Max).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M>Max, read((A,N),I,M),
                   new1(I1,N,A,M).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M≤Max, read((A,N),I,M),
                   new1(I1,N,A,Max).
new1(I,N,A,Max) :- I≥N, K≥0, N>K, Z>Max, read((A,N),K,Z).
```

Checking correctness of programs

- P is correct iff incorrect $\notin M(\text{SpInt})$,
- Checking whether or not incorrect belongs to $M(\text{SpInt})$ is undecidable,
- We need a lightweight analysis SpInt to check the correctness of P:

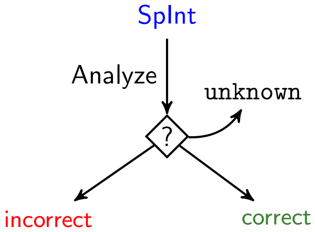


Syntactic analysis

- no constrained fact implies $M(\text{SpInt}) = \emptyset$,
- a fact incorrect. implies incorrectness
- very efficient
- achieve precision by iteration

Checking correctness of programs

- P is correct iff $\texttt{incorrect} \notin M(\mathsf{SpInt})$,
- Checking whether or not $\texttt{incorrect}$ belongs to $M(\mathsf{SpInt})$ is undecidable,
- We need a lightweight analysis SpInt to check the correctness of P:



Syntactic analysis

- no constrained fact implies $M(\mathsf{SpInt}) = \emptyset$,
- a fact `incorrect.` implies incorrectness
- very efficient
- achieve precision by iteration

Analyze SpInt

The output of Specialize, i.e., SpInt

```
incorrect :- I=0, N≥1, read((A,N),I,Max), new1(I,N,A,Max).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M>Max, read((A,N),I,M),
                   new1(I1,N,A,M).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M≤Max, read((A,N),I,M),
                   new1(I1,N,A,Max).
new1(I,N,A,Max) :- I≥N, K≥0, N>K, Z>Max, read((A,N),K,Z).
```

- We can't syntactically check whether incorrect holds or not.
- We need additional transformations.
  - Transform the specialized program into a new transition system
  - Reverse the transition system
  - Iterate the transformation process

Analyze SpInt

The output of Specialize, i.e., SpInt

```
incorrect :- I=0, N≥1, read((A,N),I,Max), new1(I,N,A,Max).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M>Max, read((A,N),I,M),
                   new1(I1,N,A,M).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M≤Max, read((A,N),I,M),
                   new1(I1,N,A,Max).
new1(I,N,A,Max) :- I≥N, K≥0, N>K, Z>Max, read((A,N),K,Z).
```

- We can't syntactically check whether `incorrect` holds or not.
- We need additional transformations.
  - Transform the specialized program into a new transition system
  - Reverse the transition system
  - Iterate the transformation process

From transformed programs to transition systems

The output of Specialize, i.e., SpInt

$$
\begin{array}{l}
\texttt{incorrect :- } I=0,\ N\geq 1,\ read((A,N),I,Max),\ new1(I,N,A,Max).\\
new1(I,N,A,Max)\ \texttt{:-}\ I1=I+1,\ I<N,\ I\geq 0,\ M>Max, read((A,N),I,M),\\
\qquad new1(I1,N,A,M).\\
new1(I,N,A,Max)\ \texttt{:-}\ I1=I+1,\ I<N,\ I\geq 0,\ M\leq Max, read((A,N),I,M),\\
\qquad new1(I1,N,A,Max).\\
new1(I,N,A,Max)\ \texttt{:-}\ I\geq N,\ K\geq 0,\ N>K,\ Z>Max,\ read((A,N),K,Z).
\end{array}
$$

can be viewed as a transition system:

$$
\begin{array}{l}
initial(\ (new1,I,N,A,Max)\ )\ \texttt{:-}\ I=0,\ N\geq 1,\ read((A,N),I,Max).\\
tr((new1,I,N,A,Max),(new1,I,N,A,M))\ \texttt{:-}\\
\qquad I1=I+1,\ I<N,\ I\geq 0,\ M>Max, read((A,N),I,M).\\
tr((new1,I,N,A,Max),(new1,I1,N,A,Max))\ \texttt{:-}\\
\qquad I1=I+1,\ I<N,\ I\geq 0,\ M\leq Max, read((A,N),I,M).\\
error((new1,I,N,A,Max))\ \texttt{:-}\\
\qquad I\geq N,\ K\geq 0,\ N>K,\ Z>Max,\ read((A,N),K,Z).
\end{array}
$$

From transformed programs to transition systems

The output of Specialize, i.e., SpInt

```
incorrect :- I=0, N≥1, read((A,N),I,Max), new1(I,N,A,Max).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M>Max, read((A,N),I,M),
                   new1(I1,N,A,M).
new1(I,N,A,Max) :- I1=I+1, I<N, I≥0, M≤Max, read((A,N),I,M),
                   new1(I1,N,A,Max).
new1(I,N,A,Max) :- I≥N, K≥0, N>K, Z>Max, read((A,N),K,Z).
```

can be viewed as a transition system:

```
initial( (new1,I,N,A,Max) ) :- I=0, N≥1, read((A,N),I,Max).
tr((new1,I,N,A,Max),(new1,I,N,A,M)) :-
                  I1=I+1, I<N, I≥0, M>Max, read((A,N),I,M).
tr((new1,I,N,A,Max),(new1,I1,N,A,Max)) :-
                  I1=I+1, I<N, I≥0, M≤Max, read((A,N),I,M).
error((new1,I,N,A,Max)) :-
                  I≥N, K≥0, N>K, Z>Max, read((A,N),K,Z).
```

Program Reversal

SpInt

```
incorrect :- initial(A), reach(A).
reach(A) :- tr(A,B), reach(B).
reach(A) :- error(A).
```

By specializing SpInt w.r.t. `incorrect`, we propagate the constraints of the initial property φ_{init}.

RevSpInt

```
incorrect :- error(A), revreach(A).
revreach(B) :- tr(A,B), revreach(A).
revreach(A) :- initial(A).
```

By specializing RevSpInt w.r.t. `incorrect`, we propagate the constraints of the error property φ_{error}.

Theorem (Correctness of Program Reversal)

`incorrect` $\in M(\text{SpInt})$ *iff* `incorrect` $\in M(\text{RevSpInt})$

Program Reversal

SpInt

```
incorrect :- initial(A), reach(A).
reach(A) :- tr(A,B), reach(B).
reach(A) :- error(A).
```

By specializing SpInt w.r.t. `incorrect`, we propagate the constraints of the initial property φ_{init}.

RevSpInt

```
incorrect :- error(A), revreach(A).
revreach(B) :- tr(A,B), revreach(A).
revreach(A) :- initial(A).
```

By specializing RevSpInt w.r.t. `incorrect`, we propagate the constraints of the error property φ_{error}.

Theorem (Correctness of Program Reversal)

`incorrect` $\in M(\text{SpInt})$ *iff* `incorrect` $\in M(\text{RevSpInt})$

Program Reversal

SpInt

```
incorrect :- initial(A), reach(A).
reach(A) :- tr(A,B), reach(B).
reach(A) :- error(A).
```

By specializing SpInt w.r.t. `incorrect`, we propagate the constraints of the initial property φ_{init}.

RevSpInt

```
incorrect :- error(A), revreach(A).
revreach(B) :- tr(A,B), revreach(A).
revreach(A) :- initial(A).
```

By specializing RevSpInt w.r.t. `incorrect`, we propagate the constraints of the error property φ_{error}.

Theorem (Correctness of Program Reversal)

`incorrect` $\in M(\text{SpInt})$ *iff* `incorrect` $\in M(\text{RevSpInt})$

Goal Replacement

We iterate the transformation from RevSpInt.
After some unfoldings we get the following clause:

$$\texttt{new2(I1,N,A,M,K,Z) :- I1}=\texttt{I}+1,\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq 0,\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},\ \texttt{Z}>\texttt{M},\ \underline{\texttt{read((A,N),K,Z),\ read((A,N),I,M)}},\ \texttt{revreach((new1,I,N,A,Max))}.$$

A law from the Theory of Arrays (arrays are finite functions)

$$\texttt{read((A,N),K,Z), read((A,N),I,M)} \leftrightarrow (\texttt{K}=\texttt{I},\ \texttt{Z}=\texttt{M},\ \texttt{read((A,N),K,Z)}) \vee (\texttt{K}\neq\texttt{I},\ \texttt{read((A,N),K,Z), read((A,N),I,M)})$$

By Goal Replacement and splitting we get:

$$\texttt{new2(I1,N,A,M,K,Z) :- I1}=\texttt{I}+1,\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq 0,\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},\ \texttt{Z}>\texttt{M},\ \texttt{K}=\texttt{I},\ \texttt{Z}=\texttt{M},\ \texttt{read((A,N),K,Z)},\ \texttt{revreach((new1,I,N,A,Max))}.$$

$$\texttt{new2(I1,N,A,M,K,Z) :- I1}=\texttt{I}+1,\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq 0,\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},\ \texttt{Z}>\texttt{M},\ \texttt{K}\neq\texttt{I},\ \texttt{read((A,N),K,Z), read((A,N),I,M)},\ \texttt{revreach((new1,I,N,A,Max))}.$$

Goal Replacement

We iterate the transformation from RevSpInt.
After some unfoldings we get the following clause:

$$\texttt{new2(I1,N,A,M,K,Z) :- I1}=\texttt{I}+\texttt{1},\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq\texttt{0},\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},$$
$$\texttt{Z}>\texttt{M},\ \underline{\texttt{read((A,N),K,Z),\ read((A,N),I,M)}},$$
$$\texttt{revreach((new1,I,N,A,Max))}.$$

A law from the Theory of Arrays (arrays are finite functions)

$$\texttt{read((A,N),K,Z),\ read((A,N),I,M)}\ \leftrightarrow$$
$$(\texttt{K}=\texttt{I},\ \texttt{Z}=\texttt{M},\ \texttt{read((A,N),K,Z)})$$
$$\vee\ (\texttt{K}\neq\texttt{I},\ \texttt{read((A,N),K,Z)},\ \texttt{read((A,N),I,M)})$$

By Goal Replacement and splitting we get:

$$\texttt{new2(I1,N,A,M,K,Z) :- I1}=\texttt{I}+\texttt{1},\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq\texttt{0},\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},$$
$$\texttt{Z}>\texttt{M},\ \texttt{K}=\texttt{I},\ \texttt{Z}=\texttt{M},\ \texttt{read((A,N),K,Z)},\ \texttt{revreach((new1,I,N,A,Max))}.$$
$$\texttt{new2(I1,N,A,M,K,Z) :- I1}=\texttt{I}+\texttt{1},\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq\texttt{0},\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},$$
$$\texttt{Z}>\texttt{M},\ \texttt{K}\neq\texttt{I},\texttt{read((A,N),K,Z)},\texttt{read((A,N),I,M)},$$
$$\texttt{revreach((new1,I,N,A,Max))}.$$

Goal Replacement

We iterate the transformation from RevSpInt.
After some unfoldings we get the following clause:

$$\begin{aligned}\texttt{new2(I1,N,A,M,K,Z) :- } & \texttt{I1}=\texttt{I}+\texttt{1},\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq\texttt{0},\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},\\ & \texttt{Z}>\texttt{M},\ \underline{\texttt{read((A,N),K,Z)},\ \texttt{read((A,N),I,M)}},\\ & \texttt{revreach((new1,I,N,A,Max))}.\end{aligned}$$

A law from the Theory of Arrays (arrays are finite functions)

$$\begin{aligned}\texttt{read((A,N),K,Z)},\ \texttt{read((A,N),I,M)}\ \leftrightarrow\ & (\texttt{K}=\texttt{I},\ \texttt{Z}=\texttt{M},\ \texttt{read((A,N),K,Z)})\\ \vee\ & (\texttt{K}\neq\texttt{I},\ \texttt{read((A,N),K,Z)},\ \texttt{read((A,N),I,M)})\end{aligned}$$

By Goal Replacement and splitting we get:

$$\begin{aligned}&\texttt{new2(I1,N,A,M,K,Z) :- } \texttt{I1}=\texttt{I}+\texttt{1},\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq\texttt{0},\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},\\ &\quad \texttt{Z}>\texttt{M},\ \texttt{K}=\texttt{I},\ \texttt{Z}=\texttt{M},\ \texttt{read((A,N),K,Z)},\ \texttt{revreach((new1,I,N,A,Max))}.\\ &\texttt{new2(I1,N,A,M,K,Z) :- } \texttt{I1}=\texttt{I}+\texttt{1},\ \texttt{N}=\texttt{I1},\ \texttt{K}\geq\texttt{0},\ \texttt{K}<\texttt{I1},\ \texttt{M}>\texttt{Max},\\ &\qquad\qquad \texttt{Z}>\texttt{M},\ \texttt{K}\neq\texttt{I},\texttt{read((A,N),K,Z)},\texttt{read((A,N),I,M)},\\ &\qquad\qquad \texttt{revreach((new1,I,N,A,Max))}.\end{aligned}$$

Final Transformed Program

The output of the transformation strategy is the following program

```
incorrect :- I≥N, K≥0, K<N, Z>Max, new2(I,N,A,Max,K,Z).
new2(I1,N,A,Max,K,Z) :- I1=I+1, N=I1, K≥0, K<I, M>Max,
                        Z>M, read((A,N),I,M), new3(I,N,A,Max,K,Z).
new2(I1,N,A,M,K,Z) :- I1=I+1, N=I1, K≥0, K<I, M≤Max,
                      Z>Max, read((A,N),I,M), new3(I,N,A,Max,K,Z).
new3(I1,N,A,M,K,Z) :- I1=I+1, K≥0, K+1<I1, N≥I1, M>Max,
                        Z>M, read((A,N),I,M), new3(I,N,A,Max,K,Z).
new3(I1,N,A,Max,K,Z) :- I1=I+1, K≥0, K+1<I1, N≥I1, M≤Max,
                      Z>Max, read((A,N),I,M), new3(I,N,A,Max,K,Z).
```

which contains no constrained facts.

Thus, we have verified the property of interest.

The Greatest Common Divisor

We prove correctness wrt recursively defined properties

The GCD Program

```
x=m; y=n;
while(x != y) {
    if(x > y) x=x-y;
    else      y=y-x;
}
z=x;
```

Initial and error properties

$\varphi_{init}(m,n) \equiv m \geq 1 \wedge n \geq 1$

$\varphi_{error}(m,n,z) \equiv \exists d\ (gcd(m,n,d) \ \wedge\ d \neq z)$

CLP Encoding

```
phiInit(M,N) :- M≥1, N≥1.
phiError(M,N,Z) :- gcd(M,N,D), D≠Z.
gcd(X,Y,D) :- X>Y, X1=X−Y, gcd(X1,Y,D).
gcd(X,Y,D) :- X<Y, Y1=Y−X, gcd(X,Y1,D).
gcd(X,Y,D) :- X=Y, Y=D.
```

The Greatest Common Divisor

We prove correctness wrt recursively defined properties

The GCD Program

```
x=m; y=n;
while(x != y) {
    if(x > y) x=x-y;
    else      y=y-x;
}
z=x;
```

Initial and error properties

$\varphi_{init}(m,n) \equiv m \geq 1 \land n \geq 1$

$\varphi_{error}(m,n,z) \equiv \exists d\ (gcd(m,n,d)\ \land\ d \neq z)$

CLP Encoding

```
phiInit(M,N) :- M≥1, N≥1.
phiError(M,N,Z) :- gcd(M,N,D), D≠Z.
gcd(X,Y,D) :- X>Y, X1=X−Y, gcd(X1,Y,D).
gcd(X,Y,D) :- X<Y, Y1=Y−X, gcd(X,Y1,D).
gcd(X,Y,D) :- X=Y, Y=D.
```

Conclusions and Future Work

- Parametric verification framework (semantics and logic, constraint domain)
  - CLP as a metalanguage
  - agile way of synthesizing software verifiers (Rybalchenko)
- Semantics preserving transformation
  - iteration, incremental verification
  - use Horn clauses for passing information between verifiers (McMillan)
- Future work
  - automation of generalization
  - termination of goal replacement
  - more experiments, more theories (lists, heaps, ...)

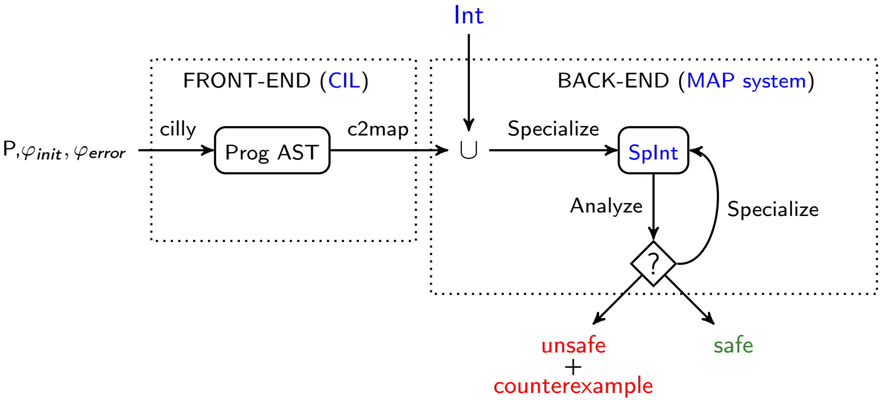
Conclusions and Future Work

- Parametric verification framework (semantics and logic, constraint domain)
    - CLP as a metalanguage
    - agile way of synthesizing software verifiers (Rybalchenko)
- Semantics preserving transformation
    - iteration, incremental verification
    - use Horn clauses for passing information between verifiers (McMillan)
- Future work
    - automation of generalization
    - termination of goal replacement
    - more experiments, more theories (lists, heaps, ...)

Software Model Checker Architecture

Fully automatic Software Model Checker for proving safety of C programs.

- CIL (C Intermediate Language)
  http://kerneis.github.com/cil/
- MAP Transformation System
  http://map.uniroma2.it/mapweb



Verification results using MAP, ARMC, HSF(C) and TRACER.

	MAP	ARMC	HSF(C)	TRACER	
				SPost	*WPre*
correct answ.	185	138	160	91	103
safe problems	154	112	138	74	85
unsafe problems	31	26	22	17	18
incorrect answ	0	9	4	13	14
missed bugs	0	1	1	0	0
false alarms	0	8	3	13	14
errors	0	18	0	20	22
timeout	31	51	52	92	77
score	339 (0)	210 (-40)	278 (-20)	113 (-52)	132 (-56)
tot time	10717.34	15788.21	15770.33	27757.46	23259.19
avg time	57.93	114.41	98.56	305.03	225.82

Time is in seconds. The time limit is five minutes.