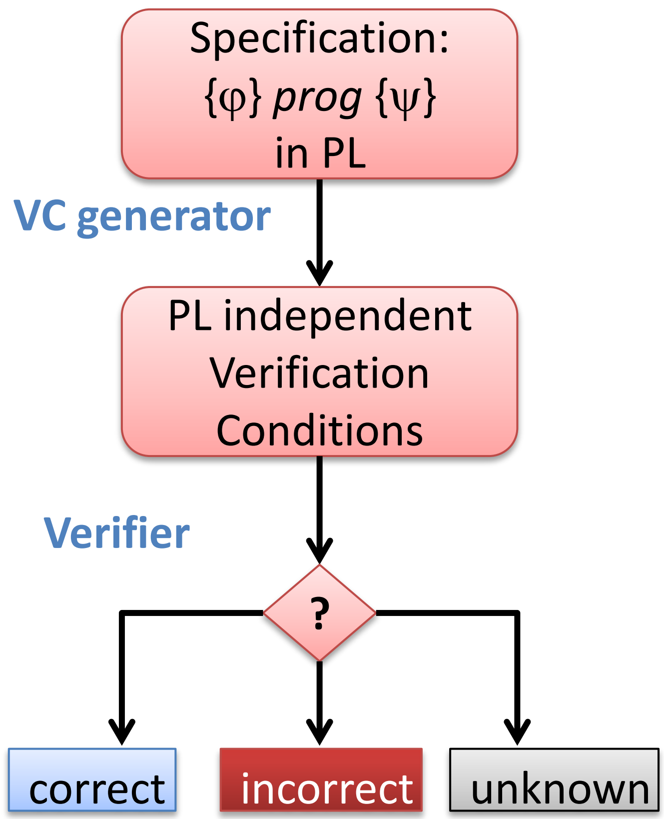
Verification of Imperative Programs through Transformation of Constraint Logic Programs

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

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

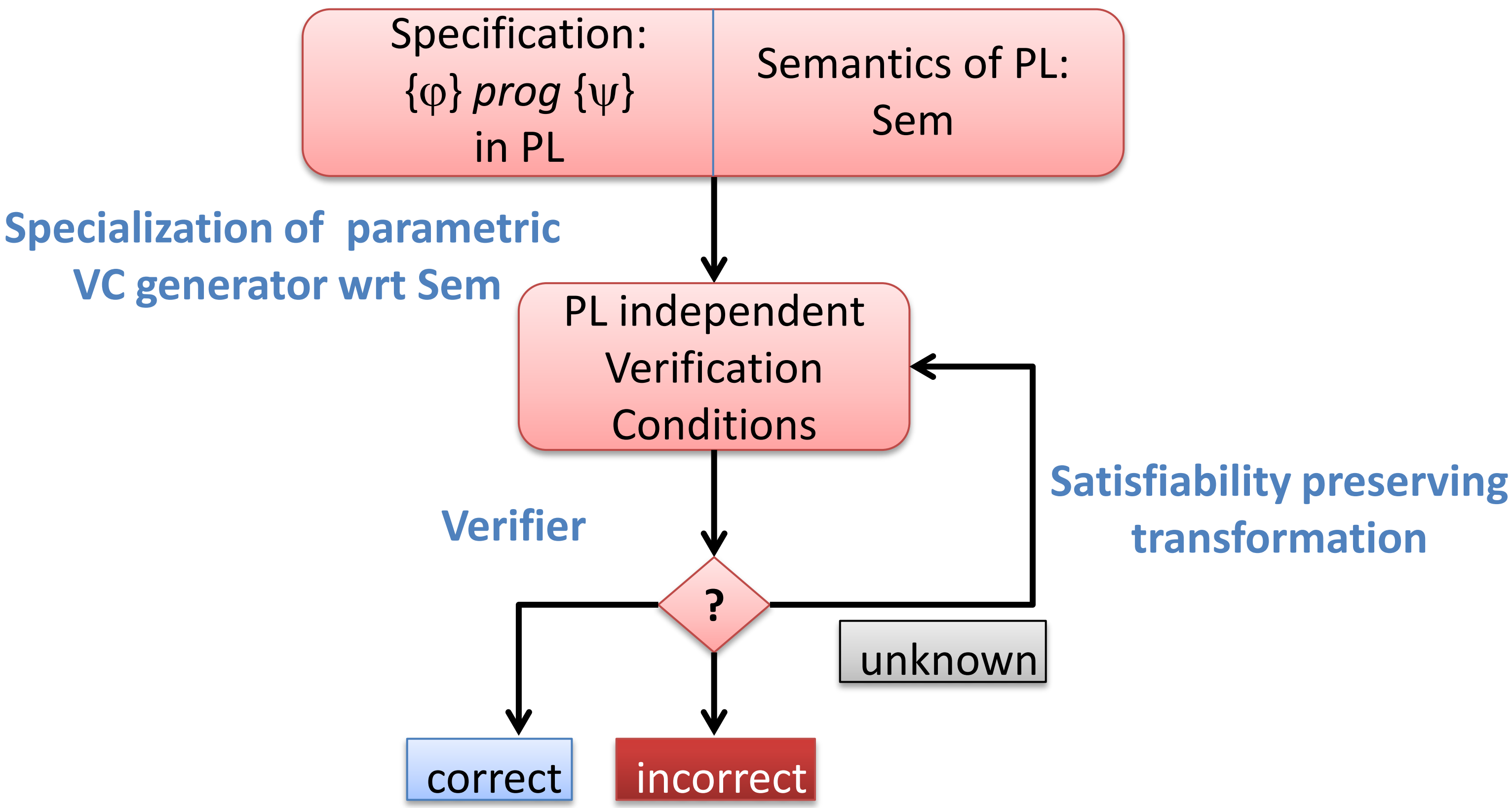
Milano, 25 Settembre 2014

Program Verification



- The VC generator has to be rewritten when PL is changed
- The work done is lost if the answer is “unknown”

Transformation-based Verification



- The VC generator is parametric wrt semantics of PL
- The work done is kept even if the answer is "unknown"

Parametric verification via CLP transformations

- Constraint Logic Programming (CLP) as a *metalanguage* for representing the semantics of PL
- The verification method based on CLP transformation is *parametric* with respect to:
  - programming language and its operational semantics
  - properties and proof rules
  - theory of data structures.
- The input and the output of transformation are semantically equivalent CLP programs. This allows:
  - *composition* of verification tasks
  - *iteration* for refining verification
  - *interoperation* with other verifiers that use CLP (Horn clause) format.

Outline of the Talk

- CLP representation of
  - the imperative program
  - the semantics of the imperative language (*interpreter*)
  - the property to be verified
- Verification method based on CLP program transformation
  - *Transformation rules and strategies* that preserve the semantics of CLP
  - *VC generation* by specialization of the interpreter
  - *VC transformation* by propagation of the property to be verified
- Improving precision via *iterated* VC transformation
- Experimental evaluation: The VeriMAP system

CLP as a metalanguage for imperative programs

CLP with integer constraints

- A CLP *clause* is an implication $\mathtt{c} \wedge \mathtt{G} \rightarrow \mathtt{H}$, written as:

  ```
  H :- c, G.
  ```

  where H is an atom, c is a constraint, and G is a conjunction of atoms

- A *constraint* is a conjunction of linear equalities/inequalities over integers ($p_1 = p_2$, $p_1 \geq p_2$, $p_1 > p_2$)

- A CLP *program* is a set of CLP clauses

- Semantics: *least model* of the program with the fixed interpretation of constraints.

Imperative Programs over Integers

- We consider an imperative language with integer variables, assignment, if-else, while-loop, and goto.
- Program *increase*:

```
while(x < n){
  x=x+1;
  y=x+y;
}
```

- Partial Correctness Specification

$\{x = 0 \wedge y = 0\}$ *increase* $\{x \leq y\}$

Encoding of an Imperative Program into CLP

A program is represented as a set of atoms at(*label*, *command*).

Program *increase*:

ℓ_0 : `while(x < n){`
ℓ_1 : `x=x+1;`
ℓ_2 : `y=x+y;`
ℓ_3 : `}`

CLP encoding of *increase*:

$\mathtt{at}(\ell_0, \mathtt{ite}(\mathtt{less}(\mathtt{int}(\mathtt{x}), \mathtt{int}(\mathtt{n})), \ell_1, \ell_h)).$
$\mathtt{at}(\ell_1, \mathtt{asgn}(\mathtt{int}(\mathtt{x}), \mathtt{plus}(\mathtt{int}(\mathtt{x}), \mathtt{int}(1)))).$
$\mathtt{at}(\ell_2, \mathtt{asgn}(\mathtt{int}(\mathtt{y}), \mathtt{plus}(\mathtt{int}(\mathtt{x}), \mathtt{int}(\mathtt{y})))).$
$\mathtt{at}(\ell_3, \mathtt{goto}(\ell_0)).$
$\mathtt{at}(\ell_h, \mathtt{halt}).$

CLP encoding of the operational semantics (1)

A *transition semantics* is defined by:

- a set of *configurations*, i.e., a CLP term: $\texttt{cf}(\texttt{C},\texttt{S})$

  where:

  - C is a labeled *command*
  - S is a *store*,
    i.e., a list of [variable identifier, value] pairs:

    $$[[\texttt{int}(\texttt{x}),2],\ [\texttt{int}(\texttt{y}),3]]$$

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

CLP encoding of the operational semantics (2)

`L: Id=Expr`	`tr( cf(cmd(L,asgn(Id,Expr)),S), cf(cmd(L1,C1),S1)) :-` `aeval(Expr,S,V),` *evaluate expression* `update(Id,V,S,S1),` *update store* `nextlabel(L,L1),` *next label* `at(L1,C1).` *next command*

`L: if (Expr) {`
`goto L1:`
`} else`
`goto L2`
`}`
	`tr( cf(cmd(L,ite(Expr,L1,L2)),S), cf(C,S)) :-`
	`beval(Expr,S),` *expression is true*
	`at(L1,C).` *next command*
	`tr( cf(cmd(L,ite(Expr,L1,L2)),S), cf(C,S)) :-`
	`beval(not(Expr),S),` *expression is false*
	`at(L2,C).` *next command*

`L: goto L1`	`tr( cf(cmd(L,goto(L1)),S), cf(C,S)) :- at(L1,C).` `at(L1,C).` *next command*

CLP encoding of (in)correctness

Given the specification $\{\varphi_{init}\}$ *prog* $\{\psi\}$ define $\varphi_{error} \equiv \neg\psi$

Definition (Program Incorrectness)

A program *prog* is *incorrect* w.r.t. φ_{init} and φ_{error} if from an initial configuration satisfying φ_{init} it is possible to reach a final configuration satisfying φ_{error}.
Otherwise, program *prog* is *correct*.

Definition (CLP encoding of incorrectness: The interpreter *Int*)

```
incorrect :- initConf(X), reach(X).
reach(X) :- tr(X,Y), reach(Y).          | reachability
reach(X) :- errorConf(X).               |
```

`initConf(X)` $\equiv$ `X` is a configuration satisfying φ_{init}

`errorConf(X)` $\equiv$ `X` is a configuration satisfying φ_{error}

Theorem (Correctness of Encoding)

prog is correct iff `incorrect` $\notin M(Int)$ *(the least model of Int)*

Running Example: *increase* (Cont'd)

Partial Correctness Specification

$\{x = 0 \land y = 0\}$ $\quad\varphi_{init}$

increase

$\{x \leq y\}$ $\quad\psi$

$\{x > y\}$ $\quad\varphi_{error} \equiv \neg\psi$

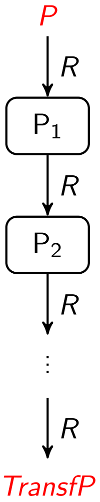
Initial and Error Configurations

```
initConf(cf(cmd(0,ite(...)), [[int(x),X],[int(y),Y],[int(n),N]]))
         :- X=0, Y=0.        φinit
errorConf(cf(cmd(h,halt), [[int(x),X],[int(y),Y],[int(n),N]]))
         :- X>Y.             φerror
```

Transformation of CLP

Unfold/Fold Program Transformation

[Burstall-Darlington 77,Tamaki-Sato 84, Etalle-Gabbrielli 96]



- transformation *rules*:
  $R \in \{$ Definition,
  Unfolding,
  Folding,
  Clause Removal$\}$

- the transformation rules *preserve the least model*:

$$\boxed{\texttt{incorrect} \in M(P) \quad \textit{iff} \quad \texttt{incorrect} \in M(\mathit{TransfP})}$$

- the rules must be guided by a *strategy*.

Rules for Transforming CLP Programs

R1. Definition. Introducing a new predicate

`newp(X) :- c, A`

Rules for Transforming CLP Programs

R1. Definition. Introducing a new predicate

`newp(X) :- c, A`

R2. Unfolding. A symbolic evaluation step (resolution)

given $\quad$ $\mathtt{H\ {:}\text{-}\ c,\ \underline{A},\ G}$

$\quad\quad\quad$ $\mathtt{\underline{A}\ {:}\text{-}\ d_1, G_1},\ \ \ldots,\ \ \mathtt{\underline{A}\ {:}\text{-}\ d_m, G_m}$

derive $\quad$ $\mathtt{H\ {:}\text{-}\ c, d_1, G_1, G},\ \ \ldots,\ \ \mathtt{H\ {:}\text{-}\ c, d_m, G_m, G}$

Rules for Transforming CLP Programs

R1. Definition. Introducing a new predicate

$\mathtt{newp(X)\ :\!-\ c, A}$

R2. Unfolding. A symbolic evaluation step (resolution)

given $\mathtt{H\ :\!-\ c,\ \underline{A},\ G}$

$\mathtt{\underline{A}\ :\!-\ d_1, G_1,\ \ \ldots,\ \ \underline{A}\ :\!-\ d_m, G_m}$

derive $\mathtt{H\ :\!-\ c, d_1, G_1, G,\ \ \ldots,\ \ H\ :\!-\ c, d_m, G_m, G}$

R3. Folding. Matching the body of a predicate definition

given $\mathtt{H\ :\!-\ d, \underline{A}, G}$

$\mathtt{newp(X)\ :\!-\ c, \underline{A}}$ and $\mathtt{d} \rightarrow \mathtt{c}$

derive $\mathtt{H\ :\!-\ d, newp(X), G}$

Rules for Transforming CLP Programs

R1. Definition. Introducing a new predicate

$$\texttt{newp(X) :- c, A}$$

R2. Unfolding. A symbolic evaluation step (resolution)

given $\quad \texttt{H :- c, }\underline{\texttt{A}}\texttt{, G}$

$\qquad\quad \underline{\texttt{A}}\texttt{ :- d}_1\texttt{, G}_1\texttt{, ..., }\underline{\texttt{A}}\texttt{ :- d}_m\texttt{, G}_m$

derive $\quad \texttt{H :- c, d}_1\texttt{, G}_1\texttt{, G, ..., H :- c, d}_m\texttt{, G}_m\texttt{, G}$

R3. Folding. Matching the body of a predicate definition

given $\quad \texttt{H :- d, }\underline{\texttt{A}}\texttt{, G}$

$\qquad\quad \texttt{newp(X) :- c, }\underline{\texttt{A}}$ $\quad$ and $\quad \texttt{d} \rightarrow \texttt{c}$

derive $\quad \texttt{H :- d, newp(X), G}$

R4. Clause Removal. Removal of clauses with unsatisfiable constraint or subsumed by others

An Example of Transformation

Initial program P:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

An Example of Transformation

Initial program P:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

definition:

```
newp(N,X) :- X>0, reach(N,X).
```

An Example of Transformation

Initial program P:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

`definition:`

```
newp(N,X) :- X>0, reach(N,X).
```

`folding:  % X=1 -> X>0`

```
incorrect :- X=1, newp(N,X).
```

An Example of Transformation

Initial program P:

```
   incorrect :- X=1, reach(N,X).
   reach(N,X) :- X<N, X'=X+1, reach(N,X').
   reach(N,X) :- X<0, X>=N.
```

```
definition:
   newp(N,X) :- X>0, reach(N,X).
folding:  % X=1 -> X>0
   incorrect :- X=1, newp(N,X).
unfolding:
   newp(N,X) :- X>0, reach(N,X).
```

An Example of Transformation

Initial program P:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

```
definition:
  newp(N,X) :- X>0, reach(N,X).
folding:  % X=1 -> X>0
  incorrect :- X=1, newp(N,X).
unfolding:
  newp(N,X) :- X>0, X<N, X'=X+1, reach(N,X').
```

An Example of Transformation

Initial program P:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

definition:

```
newp(N,X) :- X>0, reach(N,X).
```

folding: % X=1 -> X>0

```
incorrect :- X=1, newp(N,X).
```

unfolding:

```
newp(N,X) :- X>0, X<N, X'=X+1, reach(N,X').
newp(N,X) :- X>0, reach(N,X).
```

An Example of Transformation

Initial program P:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

```
definition:
   newp(N,X) :- X>0, reach(N,X).
folding:  % X=1 -> X>0
   incorrect :- X=1, newp(N,X).
unfolding:
   newp(N,X) :- X>0, X<N, X'=X+1, reach(N,X').
   newp(N,X) :- X>0, reach(N,X).
```

An Example of Transformation

Initial program P:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

`definition:`

```
newp(N,X) :- X>0, reach(N,X).
```

`folding:  % X=1 -> X>0`

```
incorrect :- X=1, newp(N,X).
```

`unfolding:`

```
newp(N,X) :- X>0, X<N, X'=X+1, reach(N,X').
newp(N,X) :- X>0, X<0, X>=N.
```

An Example of Transformation

Initial program *P*:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

definition:

```
newp(N,X) :- X>0, reach(N,X).
```

folding: % X=1 -> X>0

```
incorrect :- X=1, newp(N,X).
```

unfolding:

```
newp(N,X) :- X>0, X<N, X'=X+1, reach(N,X').
```

~~newp(N,X) :- X>0, X<0, X>=N.~~ clause removal

An Example of Transformation

Initial program P:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

definition:

```
newp(N,X) :- X>0, reach(N,X).
```

folding: % X=1 -> X>0

```
incorrect :- X=1, newp(N,X).
```

unfolding:

```
newp(N,X) :- X>0, X<N, X'=X+1, reach(N,X').
```

~~newp(N,X) :- X>0, X<0, X>=N.~~ clause removal

folding:

```
newp(N,X) :- X>0, X<N, X'=X+1, newp(N,X').
```

An Example of Transformation

Initial program *P*:

```
incorrect :- X=1, reach(N,X).
reach(N,X) :- X<N, X'=X+1, reach(N,X').
reach(N,X) :- X<0, X>=N.
```

`definition:`

```
newp(N,X) :- X>0, reach(N,X).
```

`folding:  % X=1 -> X>0`

```
incorrect :- X=1, newp(N,X).
```

`unfolding:`

```
newp(N,X) :- X>0, X<N, X'=X+1, reach(N,X').
```

~~`newp(N,X) :- X>0, X<0, X>=N.`~~ `clause removal`

`folding:`

```
newp(N,X) :- X>0, X<N, X'=X+1, newp(N,X').
```

Transformed program *TransfP*:

```
incorrect :- X=1, newp(N,X).
newp(N,X) :- X<N, X'=X+1, X>0, newp(N,X').
```

% No facts. $M(P) = M(\mathit{TransfP}) = \emptyset$. Thus, $\mathit{incorrect} \notin M(P)$.

The Transformation Strategy

Transform(P)

TransfP $= \emptyset$;
Defs $= \{$`incorrect :- initConf(X), reach(X)`$\}$;
while $\exists cl \in$ Defs **do**
    Cls $=$ Unfold(cl);
    Cls $=$ RemoveClauses(Cls);
    Defs $=$ (Defs $-$ $\{cl\}$) $\cup$ Define(Cls);
    TransfP $=$ TransfP $\cup$ Fold(Cls, Defs);
od

Theorem (Termination and Correctness of the Transformation Strategy)

- *For a suitable* Define, *Transform*(P) *terminates for all* P;
- `incorrect` $\in M(P)$ *iff* `incorrect` $\in M(TransfP)$

Generalization Strategies

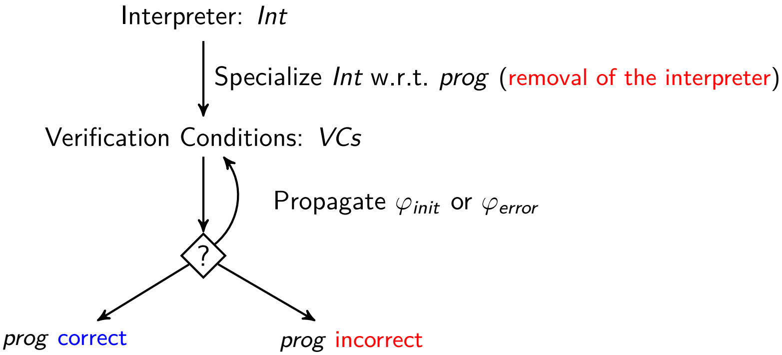
- The most critical transformation step in the Transform strategy is the *introduction of new predicate definitions* (Define) to be used for folding.
- Given `p(X) :- c(X,Y), q(Y).`

  Introduce `newp(Y) :- d(Y), q(Y).`

  where $\mathtt{c}(\mathtt{X},\mathtt{Y}) \to \mathtt{d}(\mathtt{Y})$ (`d(Y)` is a *generalization* of `c(X,Y)`)

  and *fold*: `p(X) :- c(X,Y), newp(Y).`

The Transformation-based Verification Method



- *prog* correct if no constrained facts appear in the VCs.
- *prog* incorrect if the fact `incorrect.` appears in the VCs.

Generation of Verification Conditions

Generating Verification Conditions via Specialization

Removal of the interpreter: An application of *Transform* that *specializes Int* w.r.t. *prog* by unfolding all occurrences of:

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

New predicate definitions correspond to a subset of the *program points*:

```
new1(X,Y,N) :- reach(cf(cmd(0,ite(...)),
               [[int(x),X],[int(y),Y],[int(n),N]])).
```

VC: The Specialized Interpreter for *increase* (Verification Conditions)

```
incorrect :- X=0, Y=0, new1(X,Y,N).
new1(X,Y,N) :- X<N, X1=X+1, Y1=X1+Y, new1(X1,Y1,N).
new1(X,Y,N) :- X≥N, X>Y.
```

The fact `incorrect.` is not in VC: we cannot infer that *increase* is incorrect.
A constrained fact is in VC: we cannot infer that *increase* is correct.

Propagation of Initial and Error Conditions

Propagation of φ_{init}

Propagation of the initial configuration φ_{init}: An application of *Transform* where new predicate definitions are introduced by *widening* [Cousot-Cousot 77].

Propagation of φ_{init}

Propagation of the initial configuration φ_{init}: An application of *Transform* where new predicate definitions are introduced by *widening* [Cousot-Cousot 77].

Definition 1.

```
incorrect :- X=0, Y=0, new1(X,Y,N).
```

Propagation of φ_{init}

Propagation of the initial configuration φ_{init}: An application of *Transform* where new predicate definitions are introduced by *widening* [Cousot-Cousot 77].

Definition 1.

```
incorrect :- X=0, Y=0, new1(X,Y,N).
```

Definition 2.

```
new2(X,Y,N) :- X=1, Y=1, N>0, new1(X,Y,N).
```

Propagation of φ_{init}

Propagation of the initial configuration φ_{init}: An application of *Transform* where new predicate definitions are introduced by *widening* [Cousot-Cousot 77].

Definition 1.

```
incorrect :- X=0, Y=0, new1(X,Y,N).
```

Definition 2.

```
new2(X,Y,N) :- X=1, Y=1, N>0, new1(X,Y,N).
```

Candidate new definition:

```
new3(Xr,Yr,Nr) :- Xr=1, Yr=1, X=2, Y=3, N>1, new1(X,Y,N).
```

Potential nontermination of $\widehat{T}$ransform is detected!

Propagation of φ_{init}

Propagation of the initial configuration φ_{init}: An application of *Transform* where new predicate definitions are introduced by *widening* [Cousot-Cousot 77].

Definition 1.
`incorrect :- X=0, Y=0, new1(X,Y,N).`

Definition 2.
`new2(X,Y,N) :- X=1, Y=1, N>0, new1(X,Y,N).`

Candidate new definition:
`new3(Xr,Yr,Nr) :- Xr=1, Yr=1, X=2, Y=3, N>1, new1(X,Y,N).`

Potential nontermination of $\widehat{\text{T}}$ransform is detected!

Definition 3. % Generalization based on widening
`new3(X,Y,N) :- X≥1, Y≥1, N>0, new1(X,Y,N).`

More complex generalization strategies based on combinations of widening and *convex hull* [Cousot-Halbwacs 78]

. . . Propagation of φ_{init}

VC$_1$: Transformed Verification Conditions for *increase*

. . . propagating the constraint $X = 0,\ Y = 0$.

```
incorrect :- N>0, X1=1, Y1=1, new2(X1,Y1,N).
new2(X,Y,N) :- X=1, Y=1, N>1, X1=2, Y1=3, new3(X1,Y1,N).
new3(X,Y,N) :- X1>=1, Y1>=X1, X<N, X1=X+1, Y1=X1+Y, new3(X1,Y1,N).
new3(X,Y,N) :- Y>=1, N>0, X>=N, X>Y.
```

The fact `incorrect.` is not in VC$_1$: we cannot infer that *increase* is incorrect.

A constrained fact is in VC$_1$: we cannot infer that *increase* is correct.

Program Reversal

P:

```
incorrect :- a(X), p(X).
p(X) :- c(X,Y), p(Y).
p(X) :- b(X).
```

$\downarrow$ Reversal

RevP:

```
incorrect :- b(X), p(X).
p(Y) :- c(X,Y), p(X).
p(X) :- a(X).
```

$\texttt{incorrect} \in M(\mathrm{P})$ iff $\texttt{incorrect} \in M(\mathrm{RevP})$

Propagation of φ_{error}

VC_1: Transformed Verification Conditions for *increase*

$\texttt{incorrect} \;\texttt{:-}\; \mathtt{N}>0,\ \mathtt{X1}=1,\ \mathtt{Y1}=1,\ \mathtt{new2}(\mathtt{X1},\mathtt{Y1},\mathtt{N}).$

$\underline{\mathtt{new2}(\mathtt{X},\mathtt{Y},\mathtt{N})}\texttt{:-}\ \mathtt{X}=1,\ \mathtt{Y}=1,\ \mathtt{N}>1,\ \mathtt{X1}=2,\ \mathtt{Y1}=3,\underline{\mathtt{new3}(\mathtt{X1},\mathtt{Y1},\mathtt{N})}.$

$\underline{\mathtt{new3}(\mathtt{X},\mathtt{Y},\mathtt{N})}\texttt{:-}\ \mathtt{X1}\geq 1,\ \mathtt{Y1}\geq \mathtt{X1},\ \mathtt{X}<\mathtt{N},\mathtt{X1}=\mathtt{X}+1,\ \mathtt{Y1}=\mathtt{X1}+\mathtt{Y},\underline{\mathtt{new3}(\mathtt{X1},\mathtt{Y1},\mathtt{N})}.$

$\mathtt{new3}(\mathtt{X},\mathtt{Y},\mathtt{N})\ \texttt{:-}\ \mathtt{Y}\geq 1,\ \mathtt{N}>0,\ \mathtt{X}\geq \mathtt{N},\ \mathtt{X}>\mathtt{Y}.$

Propagation of φ_{error}

VC_1: Transformed Verification Conditions for *increase*

$\mathtt{incorrect} \text{ :- } \mathtt{N}>0,\ \mathtt{X1}=1,\ \mathtt{Y1}=1,\ \mathtt{new2}(\mathtt{X1},\mathtt{Y1},\mathtt{N}).$
$\underline{\mathtt{new2}(\mathtt{X},\mathtt{Y},\mathtt{N})} \text{ :- } \mathtt{X}=1,\ \mathtt{Y}=1,\ \mathtt{N}>1,\ \mathtt{X1}=2,\ \mathtt{Y1}=3,\ \underline{\mathtt{new3}(\mathtt{X1},\mathtt{Y1},\mathtt{N})}.$
$\underline{\mathtt{new3}(\mathtt{X},\mathtt{Y},\mathtt{N})} \text{ :- } \mathtt{X1}\geq 1,\ \mathtt{Y1}\geq \mathtt{X1},\ \mathtt{X}<\mathtt{N},\ \mathtt{X1}=\mathtt{X}+1,\ \mathtt{Y1}=\mathtt{X1}+\mathtt{Y},\ \underline{\mathtt{new3}(\mathtt{X1},\mathtt{Y1},\mathtt{N})}.$
$\mathtt{new3}(\mathtt{X},\mathtt{Y},\mathtt{N}) \text{ :- } \mathtt{Y}\geq 1,\ \mathtt{N}>0,\ \mathtt{X}\geq \mathtt{N},\ \mathtt{X}>\mathtt{Y}.$

VC_2: Reversed VC_1

$\mathtt{incorrect} \text{ :- } \mathtt{Y}\geq 1,\ \mathtt{N}>0,\ \mathtt{X}\geq \mathtt{N},\ \mathtt{X}>\mathtt{Y},\ \mathtt{new3}(\mathtt{X},\mathtt{Y},\mathtt{N}).$
$\underline{\mathtt{new3}(\mathtt{X1},\mathtt{Y1},\mathtt{N})} \text{ :- } \mathtt{X1}\geq 1,\ \mathtt{Y1}\geq \mathtt{X1},\ \mathtt{X}<\mathtt{N},\ \mathtt{X1}=\mathtt{X}+1,\ \mathtt{Y1}=\mathtt{X1}+\mathtt{Y},\ \underline{\mathtt{new3}(\mathtt{X},\mathtt{Y},\mathtt{N})}.$
$\underline{\mathtt{new3}(\mathtt{X1},\mathtt{Y1},\mathtt{N})} \text{ :- } \mathtt{X}=1,\ \mathtt{Y}=1,\ \mathtt{N}>1,\ \mathtt{X1}=2,\ \mathtt{Y1}=3,\ \underline{\mathtt{new2}(\mathtt{X},\mathtt{Y},\mathtt{N})}.$
$\mathtt{new2}(\mathtt{X1},\mathtt{Y1},\mathtt{N}) \text{ :- } \mathtt{N}>0,\ \mathtt{X1}=1,\ \mathtt{Y1}=1.$

Propagation of φ_{error}

VC_1: Transformed Verification Conditions for *increase*

$\mathtt{incorrect} \text{ :- } N > 0,\ X1 = 1,\ Y1 = 1,\ \mathtt{new2}(X1, Y1, N).$
$\mathtt{new2}(X, Y, N) \text{ :- } X = 1,\ Y = 1,\ N > 1,\ X1 = 2,\ Y1 = 3, \mathtt{new3}(X1, Y1, N).$
$\mathtt{new3}(X, Y, N) \text{ :- } X1 \geq 1,\ Y1 \geq X1,\ X < N, X1 = X + 1,\ Y1 = X1 + Y, \mathtt{new3}(X1, Y1, N).$
$\mathtt{new3}(X, Y, N) \text{ :- } Y \geq 1,\ N > 0,\ X \geq N,\ X > Y.$

VC_2: Reversed VC_1

$\mathtt{incorrect} \text{ :- } Y \geq 1,\ N > 0,\ X \geq N,\ X > Y, \mathtt{new3}(X, Y, N).$
$\mathtt{new3}(X1, Y1, N) \text{ :- } X1 \geq 1,\ Y1 \geq X1,\ X < N, X1 = X + 1,\ Y1 = X1 + Y,\ \mathtt{new3}(X, Y, N).$
$\mathtt{new3}(X1, Y1, N) \text{ :- } X = 1,\ Y = 1,\ N > 1,\ X1 = 2,\ Y1 = 3,\ \mathtt{new2}(X, Y, N).$
$\mathtt{new2}(X1, Y1, N) \text{ :- } N > 0,\ X1 = 1,\ Y1 = 1.$

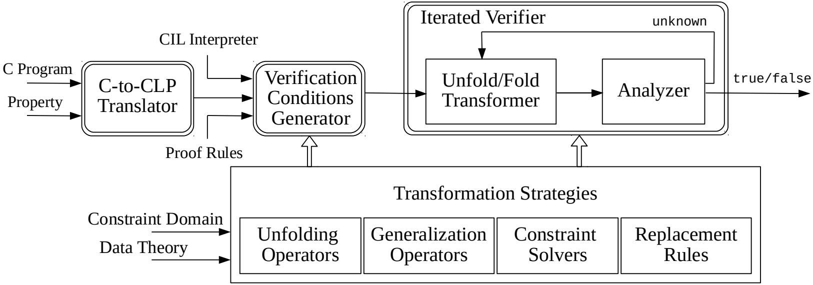
VC_3: Transformed VC_2

by propagating the constraint $Y \geq 1,\ N > 0,\ X \geq N,\ X > Y$.
$\mathtt{incorrect} \text{ :- } Y \geq 1,\ N > 0,\ X \geq N,\ X > Y, \mathtt{new4}(X, Y, N).$

No constrained facts: *increase* is correct.

Experimental Results

The VeriMAP tool `http://map.uniroma2.it/VeriMAP`
[DFPP PEPM 2013, VMCAI 2014, TACAS 2014]



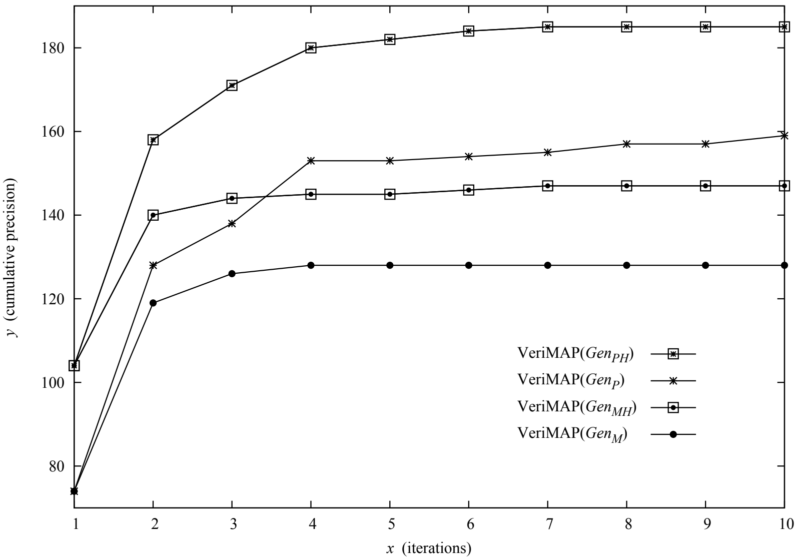
Experimental Evaluation

216 examples taken from: DAGGER, TRACER, InvGen, and TACAS 2013 Software Verification Competition.

		VeriMAP	ARMC	HSF(C)	TRACER
1	*correct answers*	185	138	160	103
2	safe problems	154	112	138	85
3	unsafe problems	31	26	22	18
4	*incorrect answers*	0	9	4	14
5	false alarms	0	8	3	14
6	missed bugs	0	1	1	0
7	*errors*	0	18	0	22
8	*timed-out problems*	31	51	52	77
9	*total score*	339 (0)	210 (-40)	278 (-20)	132 (-56)
10	*total time*	10717.34	15788.21	15770.33	23259.19
11	*average time*	57.93	114.41	98.56	225.82

- ARMC [Podelski, Rybalchenko PADL 2007]
- HSF(C) [Grebenshchikov et al. TACAS 2012]
- TRACER [Jaffar, Murali, Navas, Santosa CAV 2012]

Improving Precision by Iteration



Future Work

Exploit parametricity of the transformation-based approach and extend to:

- Recursive functions
- More data structure theories (lists, heaps, etc.)
- Other programming languages and properties

Thanks for your attention!

Try the VeriMAP tool `http://map.uniroma2.it/VeriMAP`