VeriMAP

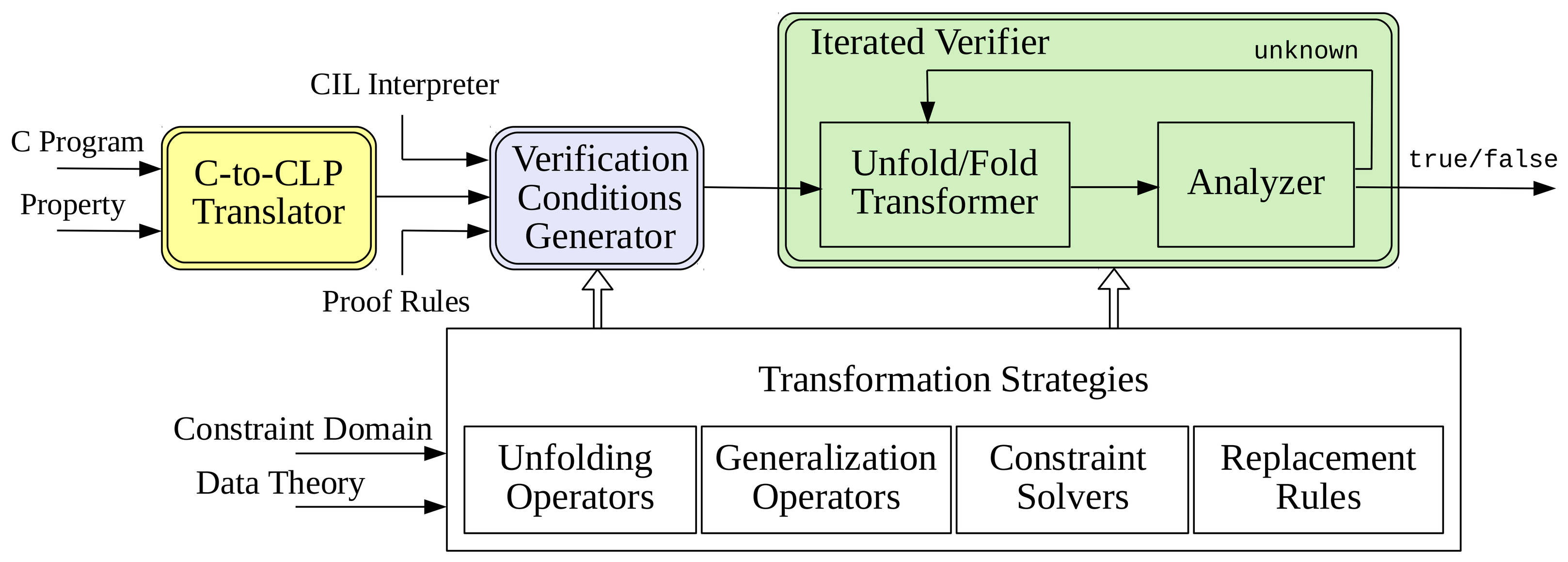
A Tool for Verifying Programs through Transformations

Emanuele De Angelis, Fabio Fioravanti, Alberto Pettorossi, and Maurizio Proietti
University of Chieti-Pescara, University of Rome 'Tor Vergata', and IASI-CNR of Rome - Italy

What is VeriMAP?

- a tool for the verification of **safety properties** of C programs
- based on **transformation of Constraint Logic Programs** (Horn clauses)
- uses CLP as a **metalanguage** for representing:
  - the operational semantics of the C language
  - the C program
  - the safety property to be verified
- **satisfiability preserving** CLP transformations
  - for generating Verification Conditions
  - for checking their satisfiability

The VeriMAP architecture



Verification of Safety Properties

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

```
int x, y, n;
while(x<n) {
    x=x+1;
    y=y+2;
}
```

Initial and error properties

$\varphi_{init}(x,y,n) \equiv x=0 \wedge y=0 \wedge n \geq 0$

$\varphi_{error}(x,y,n) \equiv y > 2x$

A program is incorrect w.r.t. φ_{init} and φ_{error} iff from an initial configuration satisfying φ_{init} it is possible to reach a final configuration satisfying φ_{error}.

C-to-CLP translator

- First `while`'s and `for`'s are translated into equivalent commands that use `if-else`'s and `goto`'s.
- Then, for each program command, *C-to-CLP* generates a CLP fact of the form `at(L,C)`, where `C` and `L` represent the command and its label.

```
1. ℓ0 : if (x<n) goto ℓ1;
        else goto ℓh;
2. ℓ1 : x=x+1;
3. ℓ2 : y=y+2;
4. ℓ3 : goto ℓ0;
5. ℓh : halt;
```

```
1. at(l0,ite(less(x,n),l1,lh)).
2. at(l1,asgn(x,expr(plus(x,1)),l2)).
3. at(l2,asgn(y,expr(plus(y,2)),l3)).
4. at(l3,goto(l0)).
5. at(lh,halt).
```

- Also facts for the initial and error properties are generated:

```
phiInit(cf(...,[(x,X),(y,Y),(n,N)])) :- X=0, Y=0, N>=0.
phiError(cf(...,[(x,X),(y,Y),(n,N)])) :- Y>2*X.
```

Encoding imperative programs using CLP - The Interpreter *Int*

```
incorrect :- initial(Cf), phiInit(Cf), reach(Cf).
reach(Cf) :- tr(Cf,Cf1), reach(Cf1).
reach(Cf) :- final(Cf), phiError(Cf).
```

+ clauses for `tr` (the operational semantics of the programming language)

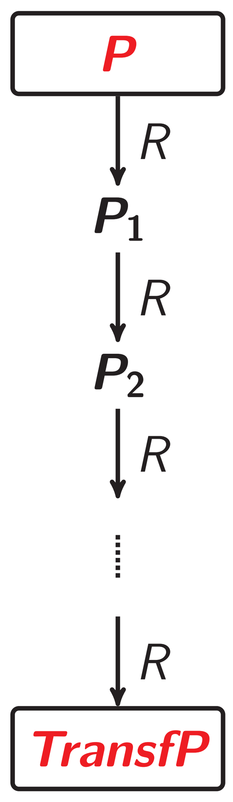
```
L: Id=Expr       tr(cf(cmd(L,asgn(Id,Expr)),E),cf(cmd(L1,C),E1)):-
                     aeval(Expr,E,V),       evaluate expression
                     update(Id,V,E,E1),     update environment
                     nextlabel(L,L1),       next label
                     at(L1,C).              next command

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

+ clauses for `at` (the encoding of the commands of the program)
+ clauses for `phiInit` and `phiError` (initial and error properties)

Correctness of the CLP encoding

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

Rule-based program transformation



- transformation **rules** *R*:
  - Definition: introduce a new predicate (e.g., a loop invariant)
  - Unfold: symbolic evaluation step (resolution)
  - Fold: matching a predicate definition (e.g., a loop invariant)
  - Clause Removal: remove unsat or subsumed clauses
  - Constraint Replacement: replace constraints using a theory
- the transformation rules **preserve** the least model semantics:

  `incorrect` $\in M(P)$ ***iff*** `incorrect` $\in M(TransfP)$
- rules are guided by a **strategy**

  (Unfold ; Constr. Repl. ; Clause Removal ; Definition ; Fold) *
- generalization operators ensure the termination of the strategy

Verification Condition Generator

Verification Conditions are generated by specializing the interpreter *Int* w.r.t. *prog*
all references to `tr` (operational semantics) and `at` (encoding of *prog*) are removed

```
incorrect :- X=0, Y=0, N>=0, new1(X,Y,N).
new1(X,Y,N) :- X<N, X1=X+1, Y1=Y+2, new1(X1,Y1,N).
new1(X,Y,N) :- X≥N, Y>2*X.
```

Unfold/Fold transformer

- The initial property is propagated by Unfold/Fold Transformation

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

Analyzer

We use a lightweight analyzer. Precision is achieved by iteration.

- If there is no constrained fact then the program is correct.
- If there is the fact incorrect then the program is incorrect.

Iterated Verification

We have propagated the constraints occurring in the initial property.
If the Analyzer returns `unknown`, we **reverse** the transition relation and **iterate** the transformation process, thus propagating the error property.

```
incorrect  :- Y>2*X,N>=0,X>=N,new4(X,Y,N).
new4(X1,Y1,N)  :- X1=X+1,Y1=Y+2,N=X+1,X>=0,Y>2*X,new5(X,Y,N).
new5(X1,Y1,N)  :- X1=X+1,Y1=Y+2,X>=0,N>=X+1,Y>2*X,new5(X,Y,N).
```

- No constrained fact is present: the program is correct.

Array programs

We apply **rewrite rules** for Constraint Replacement based on the Theory of Arrays.

- (Array Congruence) $\mathtt{I}=\mathtt{J}$, `read(A,I,U)`, `read(A,J,V)` $\rightarrow$ $\mathtt{U}=\mathtt{V}$
- (Read-over-Write1) $\mathtt{I}=\mathtt{J}$, `write(A,I,U,B)`, `read(B,J,V)` $\rightarrow$ $\mathtt{U}=\mathtt{V}$
- (Read-over-Write2) $\mathtt{I}\neq\mathtt{J}$, `write(A,I,U,B)`, `read(B,J,V)` $\rightarrow$ `read(A,J,V)`

Experimental Evaluation - Integer Programs

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

		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 time*	10717.34	15788.21	15770.33	23259.19
10	*average time*	57.93	114.41	98.56	225.82

URL: `http://map.uniroma2.it/VeriMAP/`