Branching Preserving Specialization for Software Model Checking

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

LOPSTR 2012
Leuven, Belgium, 20 September 2012

Summary

- Software model checking...
  - Imperative programs
  - Safety Checking (reachability)
- ... by Constraint Logic Program (CLP) Specialization
  - Transformation rules and automatic strategies
  - Generalization (termination of the specialization)
  - Branching preserving generalization
- Experimental results

Software Model Checking by CLP Specialization

Prog written in a language $\mathcal{L}$ and φ written in a logic $\mathcal{F}$

Phase 1: CLP encoding

$$\begin{array}{lcll} Prog & \longrightarrow & prog & \\ \mathcal{L} & \longrightarrow & L & \text{(interpreter)} \\ \varphi & \longrightarrow & prop & \\ \mathcal{F} & \longrightarrow & F & \text{(interpreter)} \end{array}$$

$$\boxed{Prog \models \varphi \ \text{ iff } \ prop \in M(L \cup F)}$$

Phase 2: *Spec* - Specialization of $(L \cup F)$ wrt $(prog, prop) \longrightarrow P_s$

Phase 3: *BUEval* - Bottom_Up computation of $M(P_s)$

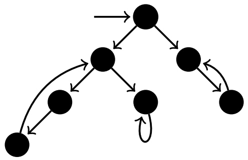
$$\boxed{prop \in M(L \cup F) \ \text{ iff } \ prop \in M(P_s).}$$

CLP encoding of imperative programs

After Phase 1 we get a CLP program encoding a transition system:

- a set of Configuration $\mathtt{cf(P, S)}$
  - Program P
  - State S a list of terms of the form $\mathtt{loc(id, val)}$
- Transition Relation $\mathtt{tr(\ cf(P, S),\ cf(P', S')\ )}$

  Operational Semantics of the language L



Example

Imperative Program:

```
int main()  {
  int x;
  int n;

  assume(x>0);

  while (x<n) {
    x = x + 1;
  }
  if (x<0)
   goto ERROR;
}
```

CLP encoding:

```
cf(

  comp(

    while(lt(var(x),var(n)),
     asgn(var(x),plus(var(x),int(1)))
    ),
    ite(lt(var(x),int(0)),
      error,
      skip)
    ),

    [loc(x,X),loc(n,N)] %state
 )
```

Assignment

```
...
ID = Exp;
...
```

```
tr( cf(asgn(var(ID),Exp),S), cf(skip,S1) ) :-
    aeval(Exp,S,Val),
    update(var(ID),Val,S,S1).
```

If-then-else

```
    ...
if (Cond) {
  Cmd1
} else {
  Cmd2
}
    ...
```

```
tr( cf(ite(Cond,Cmd1,_),S), cf(Cmd1,S) ) :-
    beval(Cond,S).

tr( cf(ite(Cond,_,Cmd2),S), cf(Cmd2,S) ) :-
    beval(not(Cond),S).
```

While

```
  ...
while (Cond) {
  Cmd1
}
  ...
```

```
tr( cf(while(Cond,Cmd1),S),
   cf(ite(Cond,comp(Cmd1,while(Cond,Cmd1)),skip),S) ) :-
        beval(Cond,S).
```

Composition of commands

```
...
Cmd1;
Cmd2
...
```

```
tr( cf(comp(Cmd1,Cmd2),S), cf(Cmd2,S1) ) :-
     tr( cf(Cmd1,S), cf(skip,S1) ).

tr( cf(comp(Cmd1,Cmd2),S), cf(comp(Cmd1',Cmd2),S1) ) :-
     tr( cf(Cmd1,S), cf(Cmd1',S1) ).
```

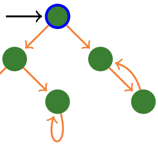
Checking safety of Imperative programs

$\mathcal{F} \longrightarrow F =$

```
ureach(X) :- unsafe(X).
ureach(X) :- tr(X,X'), ureach(X').
unsafe :- initial(X), ureach(X).
unsafe(cf(error,S)).
initial(cf(Prog,S)) :- init_constraint(S)
```

$\varphi \longrightarrow prop =$

```
safe :- not unsafe.
```



Rules for Specializing CLP Programs

R1 Atomic Definition $newp(X_1, \ldots, X_n) \leftarrow c \wedge A$

R2 Unfolding $p(X_1, \ldots, X_n) \leftarrow c \wedge q(X_1, \ldots, X_n)$ w.r.t.

$q(X_1, \ldots, X_n) \leftarrow d \wedge A$

yields

$p(X_1, \ldots, X_n) \leftarrow c \wedge d \wedge A$

R3 Atomic Folding $p(X_1, \ldots, X_n) \leftarrow c \wedge A$ w.r.t. A by using

$q(X_1, \ldots, X_n) \leftarrow d \wedge A$

yields

$p(X_1, \ldots, X_n) \leftarrow c \wedge q(X_1, \ldots, X_n)$ if $c \rightarrow d$

R4 Clause Removal

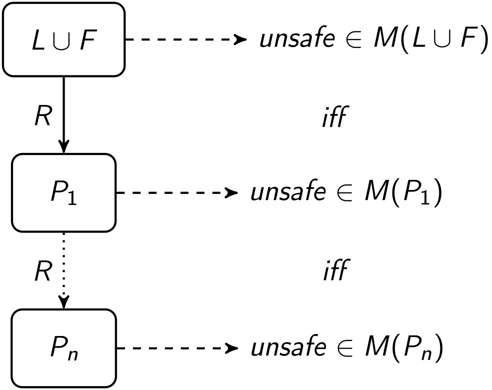
R4.1 ~~$p(X_1, \ldots, X_n) \leftarrow c \wedge q(X_1, \ldots, X_n)$~~ if c is unsatisfiable

R4.2 ~~$p(X_1, \ldots, X_n) \leftarrow c \wedge q(X_1, \ldots, X_n)$~~

$p(X_1, \ldots, X_n) \leftarrow d$ if $c \rightarrow d$ (subsumption)

Rule-based CLP Program Specialization

$$Prog \models safe \ \text{ iff } \ unsafe \notin M(L \cup F) \ \text{ iff } \ unsafe \notin M(P_i).$$



$R \in$ {Atomic Definition, Unfolding, Atomic Folding, Clause Removal}

Specialization strategy

Specialize($L \cup F$,`safe`) {

$P_s = \emptyset$;

$Def = \{$ `unsafe :- initial(X), ureach(X).` $\}$;

while ($\exists q \in Def$) **do**

Unf = Clause Removal(Unfold(q));

$Def = (Def - \{q\}) \cup$ *Generalize&Define*(Unf);

$P_s = P_s \cup$ *Fold*(Unf, Def)

od

}

Example

```
int main()  {
  int x;
  int n;

  assume(x>0);

  while (x<n) {
    x = x + 1;
  }
  if (x<0)
   goto ERROR;
}
```

```
initial( cf(

  comp(

   while(lt(var(x),var(n)),
    asgn(var(x),plus(var(x),int(1)))
   ),
   ite(lt(var(x),int(0)),
     error,
     skip)

   ), [loc(n,N),loc(x,X)] )) :- X>0.
```

Specialize($L \cup F$, *safe*) = {

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

}

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X) :- X>0, while(N,X).
2.  fold:
unsafe :- X>0, new1(N,X).
```

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X) :- X>0, while(N,X).
2.  fold:
unsafe :- X>0, new1(N,X).
3.  unfold:
```

Initial program:

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

Specialization strategy:

```
1. define:
new1(N,X) :- X>0, while(N,X).
2. fold:
unsafe :- X>0, new1(N,X).
3. unfold:
new1(N,X) :- X>0, while(N,X).
```

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X) :- X>0, while(N,X).
2.  fold:
unsafe :- X>0, new1(N,X).
3.  unfold:
new1(N,X) :- X>0, while(N,X).
```

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X) :- X>0, while(N,X).
2.  fold:
unsafe :- X>0, new1(N,X).
3.  unfold:
new1(N,X) :- X>0, X<N, X'=X+1, while(N,X').
```

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X) :- X>0, while(N,X).
2.  fold:
unsafe :- X>0, new1(N,X).
3.  unfold:
new1(N,X) :- X>0, X<N, X'=X+1, while(N,X').
new1(N,X) :- X>0, while(N,X).
```

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X) :- X>0, while(N,X).
2.  fold:
unsafe :- X>0, new1(N,X).
3.  unfold:
new1(N,X) :- X>0, X<N, X'=X+1, while(N,X').
new1(N,X) :- X>0, while(N,X).
```

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X) :- X>0, while(N,X).
2.  fold:
unsafe :- X>0, new1(N,X).
3.  unfold:
new1(N,X) :- X>0, X<N, X'=X+1, while(N,X').
new1(N,X) :- X>0, X<0, X>=N.
```

Initial program:

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

Specialization strategy:

```
1. define:
new1(N,X) :- X>0, while(N,X).
2. fold:
unsafe :- X>0, new1(N,X).
3. unfold:
new1(N,X) :- X>0, X<N, X'=X+1, while(N,X').
```

~~new1(N,X) :- X>0, X<0, X>=N.~~

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X) :- X>0, while(N,X).
2.  fold:
unsafe :- X>0, new1(N,X).
3.  unfold:
new1(N,X) :- X>0, X<N, X'=X+1, while(N,X').
```
~~new1(N,X) :- X>0, X<0, X>=N.~~
```
4.  fold:
new1(N,X) :- X>0, X<N, X'=X+1, new1(N,X').
```

Initial program:

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

Specialization strategy:

```
1. define:
new1(N,X) :- X>0, while(N,X).
2. fold:
unsafe :- X>0, new1(N,X).
3. unfold:
new1(N,X) :- X>0, X<N, X'=X+1, while(N,X').
```
~~new1(N,X) :- X>0, X<0, X>=N.~~

```
4. fold:
new1(N,X) :- X>0, X<N, X'=X+1, new1(N,X').
```

Specialized program:

```
unsafe :- X>0, new1(N,X).
new1(N,X) :- X<N, X'=X+1, X>0, new1(N,X').
```

% No facts. Prog is <u>safe!</u>

```
Specialize(L ∪ F, safe) {
  Ps = ∅;
  Def = {unsafe :- initial(X), ureach(X).};
  while ( ∃q ∈ Def ) do
      Unf = Clause Removal( Unfold( q ) );
      Def = (Def − {q}) ∪ Generalize&Define( Unf );
      Ps = Ps ∪ Fold(Unf, Def)
  od
}
```

Generalize&Define$(\cdot)$ may introduce infinitely many new definitions and leads to non termination of Specialize.

Generalizations in *Generalize&Define*$(\cdot)$ ensure termination...

$\gamma: \; H \leftarrow c \wedge A$

$\delta: \; H \leftarrow g \wedge A$

δ is a generalization of γ iff $c \sqsubseteq g$ iff $\mathcal{R} \models \forall X \; (c(X) \rightarrow g(X))$.

... but may prevent the proof of the property.

Generalization

```
int main()  {
  int x=0;  int y=0;
  int n;

  while (x<n) {
    x = x + 1;
    y = y + 1;
  }
  if (y>x)
   goto ERROR;

  return 0;
}
```

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

Generalization

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2.  fold:
unsafe :- X=0, Y=0, new1(N,X).
```

Generalization

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2.  fold:
unsafe :- X=0, Y=0, new1(N,X).
3.  unfold:
```

Generalization

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2.  fold:
unsafe :- X=0, Y=0, new1(N,X).
3.  unfold:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
```

Generalization

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2.  fold:
unsafe :- X=0, Y=0, new1(N,X).
3.  unfold:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
```

Generalization

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2.  fold:
unsafe :- X=0, Y=0, new1(N,X).
3.  unfold:
new1(N,X,Y) :- X=0, Y=0, X<N, X'=X+1,Y'=Y+1,
   while(N,X',Y').
```

Generalization

Initial program:

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

Specialization strategy:

```
1. define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2. fold:
unsafe :- X=0, Y=0, new1(N,X).
3. unfold:
new1(N,X,Y) :- X=0, Y=0, X<N, X'=X+1, Y'=Y+1,
   while(N,X',Y').
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
```

Generalization

Initial program:

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

Specialization strategy:

```
1. define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2. fold:
unsafe :- X=0, Y=0, new1(N,X).
3. unfold:
new1(N,X,Y) :- X=0, Y=0, X<N, X'=X+1, Y'=Y+1,
   while(N,X',Y').
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
```

Generalization

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2.  fold:
unsafe :- X=0, Y=0, new1(N,X).
3.  unfold:
new1(N,X,Y) :- X=0, Y=0, X<N, X'=X+1, Y'=Y+1,
   while(N,X',Y').
new1(N,X,Y) :- X=0, Y=0, X>Y, X>=N.
```

Generalization

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2.  fold:
unsafe :- X=0, Y=0, new1(N,X).
3.  unfold:
new1(N,X,Y) :- X=0, Y=0, X<N, X'=X+1, Y'=Y+1,
   while(N,X',Y').
```

~~new1(N,X,Y) :- X=0, Y=0, X>Y, X>=N.~~

Generalization

Initial program:

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

Specialization strategy:

```
1.  define:
new1(N,X,Y) :- X=0, Y=0, while(N,X,Y).
2.  fold:
unsafe :- X=0, Y=0, new1(N,X).
3.  unfold:
new1(N,X,Y) :- X=0, Y=0, X<N, X'=X+1, Y'=Y+1,
   while(N,X',Y').
```

we cannot fold

```
  new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

by using

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

Generalization

we need to introduce a new definition...

we may introduce

```
new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

Generalization

we need to introduce a new definition...

we may introduce

```
new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

```
4.  generalize & define:
  new2(N,X,Y) :- X>=0, Y>=0, while(N,X,Y).
```

Generalization

we need to introduce a new definition...

we may introduce

```
new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

```
4.  generalize & define:
  new2(N,X,Y) :- X>=0, Y>=0, while(N,X,Y).
5.  fold:
  new1(N,X,Y) :- X<N, X=1, Y=1, new2(N,X,Y).
```

Generalization

we need to introduce a new definition...

we may introduce

```
  new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

```
4.  generalize & define:
  new2(N,X,Y) :- X>=0, Y>=0, while(N,X,Y).
5.  fold:
  new1(N,X,Y) :- X<N, X=1, Y=1, new2(N,X,Y).
6.  unfold:
  new2(N,X,Y) :- X>=0, Y>=0, X<N, X'=1+X, Y'=1+Y,
    while(N,X',Y').
  new2(N,X,Y) :- X>=0, Y>=0, Y>X, X>=N.
```

Generalization

we need to introduce a new definition...

we may introduce

```
  new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

```
4.  generalize & define:
  new2(N,X,Y) :- X>=0, Y>=0, while(N,X,Y).
5.  fold:
  new1(N,X,Y) :- X<N, X=1, Y=1, new2(N,X,Y).
6.  unfold:
  new2(N,X,Y) :- X>=0, Y>=0, X<N, X'=1+X, Y'=1+Y,
    while(N,X',Y').
  new2(N,X,Y) :- X>=0, Y>=0, Y>X, X>=N.
```

Generalization

we need to introduce a new definition...

we may introduce

```
  new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

```
4.  generalize & define:
  new2(N,X,Y) :- X>=0, Y>=0, while(N,X,Y).
5.  fold:
  new1(N,X,Y) :- X<N, X=1, Y=1, new2(N,X,Y).
6.  unfold:
  new2(N,X,Y) :- X>=0, Y>=0, X<N, X'=1+X, Y'=1+Y,
    while(N,X',Y').
  new2(N,X,Y) :- X>=0, Y>=0, Y>X, X>=N.
```

Specialized program:

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

Generalization

Specialized program:

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

We have a constrained fact.

The Bottom Up computation of $M(P_s)$ does not terminate.

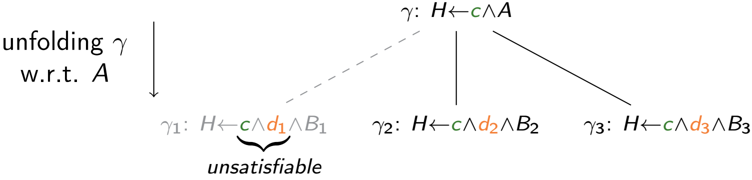
Thus, we are not able to prove, or disprove, the safety of the given imperative program!

Branching preserving generalization

$A \leftarrow d_1 \wedge B_1$

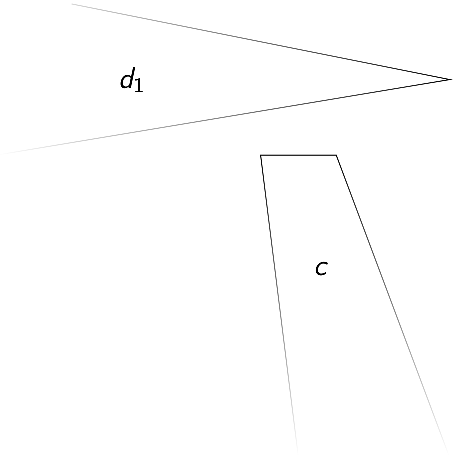
$A \leftarrow d_2 \wedge B_2$

$A \leftarrow d_3 \wedge B_3$



$$c \sqsubseteq \neg d_1$$

Branching preserving generalization



$\gamma : \; H \leftarrow c \wedge A$

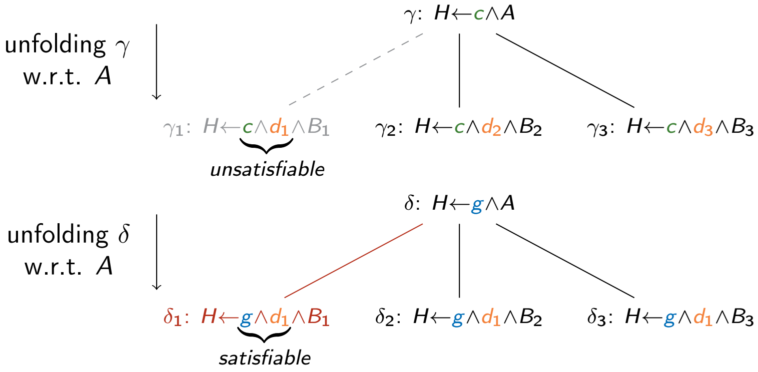
$c \wedge d_1$ is unsatisfiable

Branching preserving generalization

$A \leftarrow d_1 \wedge B_1$

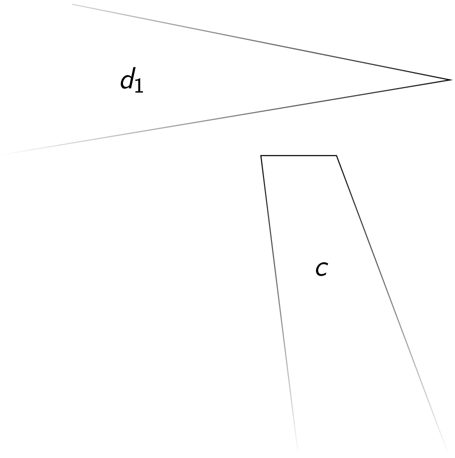
$A \leftarrow d_2 \wedge B_2$

$A \leftarrow d_3 \wedge B_3$



$$\begin{array}{ccc} g & & \\ \sqcup\!\mid & & \\ c & \sqsubseteq & \neg\, d_1 \end{array}$$

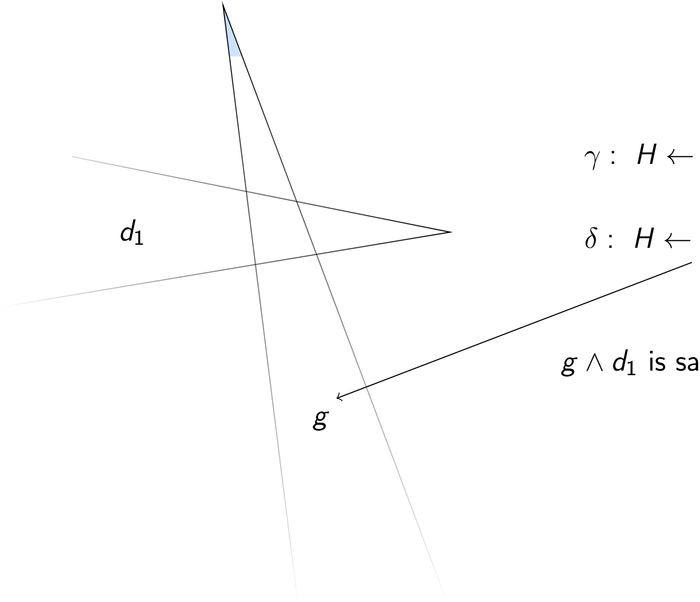
Branching preserving generalization



$\gamma : \; H \leftarrow c \wedge A$

$\delta : \; H \leftarrow g \wedge A$

Branching preserving generalization



$\gamma: \; H \leftarrow c \wedge A$

$\delta: \; H \leftarrow g \wedge A$

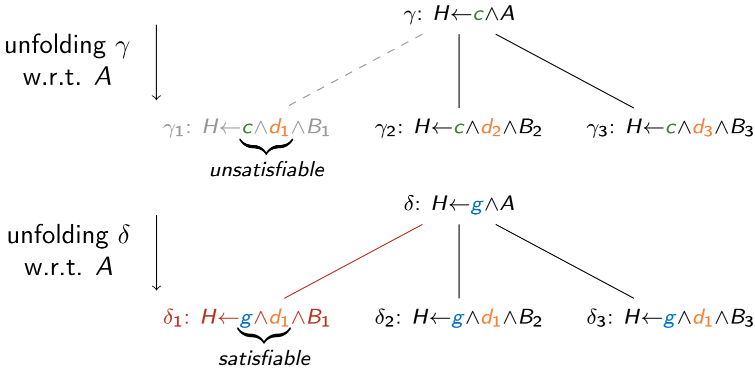
$g \wedge d_1$ is satisfiable

Branching preserving generalization

$A \leftarrow d_1 \wedge B_1$

$A \leftarrow d_2 \wedge B_2$

$A \leftarrow d_3 \wedge B_3$



$$c \sqsubseteq \neg d_1$$

Branching preserving generalization

$A \leftarrow d_1 \wedge B_1$

$A \leftarrow d_2 \wedge B_2$

$A \leftarrow d_3 \wedge B_3$

unfolding γ w.r.t. A

γ: $H \leftarrow c \wedge A$

γ_1: $H \leftarrow \underbrace{c \wedge d_1}_{\textit{unsatisfiable}} \wedge B_1$ γ_2: $H \leftarrow c \wedge d_2 \wedge B_2$ γ_3: $H \leftarrow c \wedge d_3 \wedge B_3$

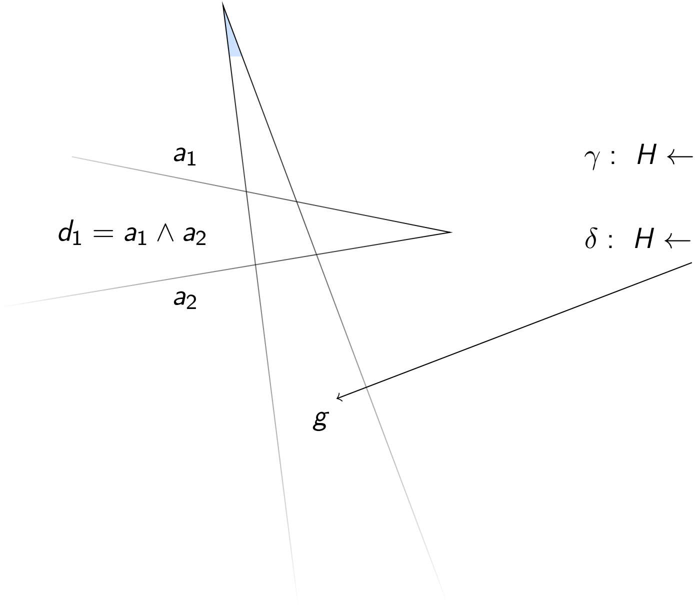
unfolding δ w.r.t. A

δ: $H \leftarrow g_{bp} \wedge A$

δ_1: $H \leftarrow \underbrace{g_{bp} \wedge d_1}_{\textit{unsatisfiable}} \wedge B_1$ δ_2: $H \leftarrow g_{bp} \wedge d_1 \wedge B_2$ δ_3: $H \leftarrow g_{bp} \wedge d_1 \wedge B_3$

Find a g_{bp} such that $c \sqsubseteq g_{bp} \sqsubseteq \neg d_1$

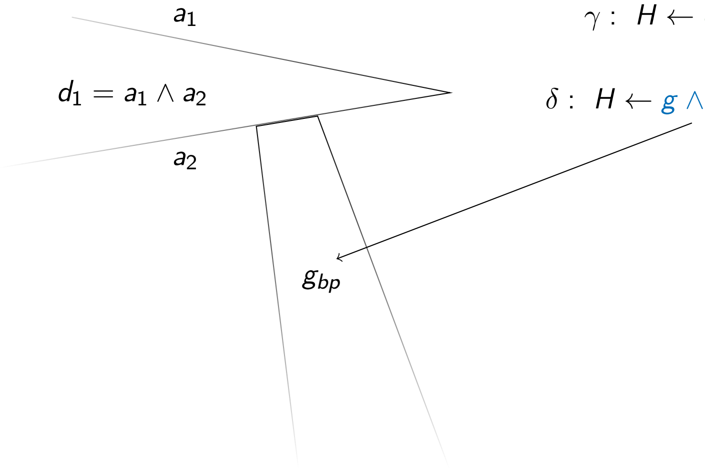
Branching preserving generalization



$\gamma : \; H \leftarrow c \wedge A$

$\delta : \; H \leftarrow g \wedge A$

Branching preserving generalization



$\gamma: \; H \leftarrow c \wedge A$

$\delta: \; H \leftarrow g \wedge \neg a_2 \wedge A$

Generalization

we need to introduce a new definition...

we may introduce

```
  new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

```
4.  generalize & define:
  new2(N,X,Y) :- X>=0, Y>=0, while(N,X,Y).
5.  fold:
  new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
6.  unfold:
  new2(N,X,Y) :- X>=0, Y>=0, X<N, X'=1+X, Y'=1+Y,
    while(N,X',Y').
  new2(N,X,Y) :- X>=0, Y>=0, Y>X, X>=N.
```

Specialized program:

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

Generalization with Branching Preserving

we need to introduce a new definition...

we may introduce

```
  new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

```
4.  generalize & define:
  new2(N,X,Y) :- X>=0, Y>=0, X>=Y, while(N,X,Y).
5.  fold:
  new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
6.  unfold:
  new2(N,X,Y) :- X>=0, Y>=0, X<N, X'=1+X, Y'=1+Y,
     while(N,X',Y').
  new2(N,X,Y) :- X>=0, Y>=0, Y>X, X>=N.
```

Specialized program:

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

Generalization

we need to introduce a new definition...

we may introduce

```
new1(N,X,Y) :- X<N, X=1, Y=1, while(N,X,Y).
```

or we may introduce a generalization

```
4.  generalize & define:
  new2(N,X,Y) :- X>=0, Y>=0, X>=Y, while(N,X,Y).
5.  fold:
  new1(N,X,Y) :- X<N, X=1, Y=1, new2(N,X,Y).
6.  unfold:
  new2(N,X,Y) :- X>=0, Y>=0, X<N, X'=1+X, Y'=1+Y,
    while(N,X',Y').
```

~~new2(N,X,Y) :- X>=0, Y>=0, Y>X, X>=N, X>=Y.~~

Specialized program:

```
unsafe :- X=0, Y=0, new1(N,X,Y).
new1(N,X,Y) :- X=0, Y=0, X'=1, Y'=1, X<N, new2(N,X',Y').
```

~~new2(N,X,Y) :- X>=0, Y>=0, Y>X, X>=N, X>=Y.~~

```
new2(N,X,Y) :- Y>=1, X>=1, X'=X+1, Y'=Y+1, X<N, new2(N,X',Y').
```

Generalization with Branching Preserving

Specialized program:

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

No facts.

Prog is safe!

Preliminary results

Program	MAP				ARMC	HSF(C)	TRACER	
	W	W_{bp}	$CHWM$	$CHWM_{bp}$			$SPost$	$WPre$
$ex1$	1.08	1.09	1.14	1.25	0.18	0.21	∞	1.29
$f1a$	∞	∞	0.35	0.36	∞	0.20	$\perp$	1.30
$f2$	∞	∞	0.75	0.88	∞	0.19	∞	1.32
$interp$	0.29	0.29	0.32	0.44	0.13	0.18	∞	1.22
$re1$	∞	0.33	0.33	0.33	∞	0.19	∞	∞
$selectSort$	4.34	4.70	4.59	5.57	0.48	0.25	∞	∞
$singleLoop$	∞	∞	∞	0.26	∞	∞	$\perp$	1.28
$substring$	88.20	171.20	5.21	5.92	931.02	1.08	187.91	184.09
$tracerP$	0.11	0.12	0.11	0.12	∞	∞	1.15	1.28

Table: Time (in seconds) taken for performing model checking.
'∞' means 'no answer within 20 minutes', and
'$\perp$' means 'termination with error'.

Conclusions

Program specialization is a framework for performing an Agile, Iterative and Evolutionary development of verification techniques and tools:

- soundness of abstraction
- parametricity w.r.t. languages and logics
- compositionality of program transformations
- modularity separation of language features and verification techniques

From LOPSTR submission up to now
we have extended our approach to deal with C programs.

Control Flow Analysis of C programs using Integer (int, short, unsigned long, ...),

e.g. o.s. device drivers

Current work:

extending F to deal with different properties (e.g. liveness),
extending L to deal with pointers.