Software Model Checking by Program Specialization

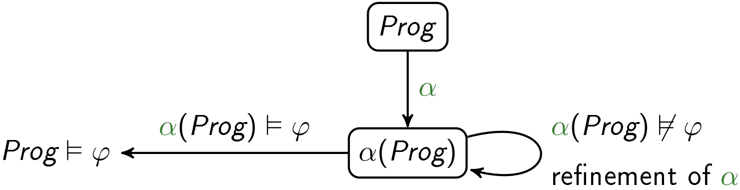
Emanuele De Angelis[1,2]
deangelis@sci.unich.it
www.sci.unich.it/∼deangelis

[1]University of Chieti-Pescara 'G. d'Annunzio'

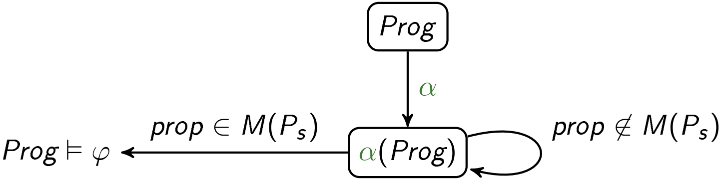
[2]CNR-IASI, Rome

ICLP-DC, Budapest, Hungary
4th September 2012

Software Model Checking



Software Model Checking by CLP Program Specialization



Prog written in $\mathcal{L}$ and φ specified in $\mathcal{M}$

Phase 1: Encode as a CLP program

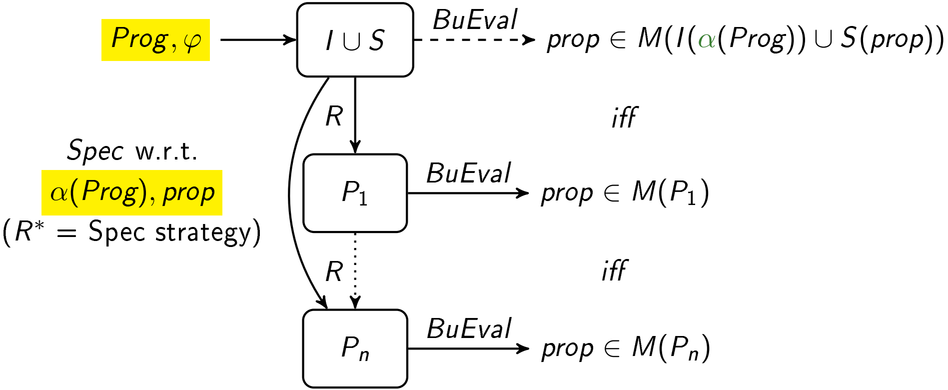
$$\begin{array}{lcl} Prog & \longrightarrow & \alpha(Prog) \\ \mathcal{L} & \longrightarrow & I\text{, interpreter for } \mathcal{L} \\ \varphi & \longrightarrow & prop \\ \mathcal{M} & \longrightarrow & S\text{, interpreter for } \mathcal{M} \end{array}$$

Phase 2: *Spec* - Specialize I and S w.r.t. $\alpha(Prog)$ and $prop \longrightarrow P_s$

Phase 3: *BuEval* - Bottom up Evaluation of $M(P_s)$

$$Prog \models \varphi \text{ iff } prop \in M(I(\alpha(Prog)) \cup S(prop)) \text{ iff } prop \in M(P_s).$$

Rule-based CLP Program Specialization



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

R1 - Unfolding

$$p(X_1, \ldots, X_n) \leftarrow c \wedge q(X_1, \ldots, X_n) \quad \text{w.r.t.} \quad q(X_1, \ldots, X_n) \leftarrow d \wedge A$$

gives

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

$P_i = \{$

$q(X) \leftarrow Y = X + 1 \wedge r(Y)$

$q(X) \leftarrow s(X)$

$p(X) \leftarrow q(X)$

$\}$

$$\xrightarrow[p \ w.r.t. \ q]{R1}$$

$P_{i+1} = \{$

$q(X) \leftarrow Y = X + 1 \wedge r(Y)$

$q(X) \leftarrow s(X)$

$p(X) \leftarrow Y = X + 1 \wedge r(Y)$

$p(X) \leftarrow s(X)$

$\}$

R2 - Folding

$$p(X_1, \ldots, X_n) \leftarrow c \wedge A \quad \text{w.r.t. } A \text{ by using} \quad q(X_1, \ldots, X_n) \leftarrow d \wedge A$$

gives

$$p(X_1, \ldots, X_n) \leftarrow c \wedge q(X_1, \ldots, X_n) \qquad \text{if} \quad c \Rightarrow d$$

$P_i = \{$
$q(Y) \leftarrow Y >= 0 \wedge r(Y)$
$p(X) \leftarrow Y = X + 1 \wedge Y = 0 \wedge r(Y)$
$\}$

$\xrightarrow{R2}$ *p w.r.t. r,s by using q*

$P_{i+1} = \{$
$q(X) \leftarrow Y = X + 1 \wedge r(Y)$
$p(X) \leftarrow Y = X + 1 \wedge Y = 0 \wedge q(X)$
$\}$

R3 - Clause removal

R3.1 $\sout{p(X_1,\ldots,X_n) \leftarrow c \wedge q(X_1,\ldots,X_n)}$ if c is unsatisfiable

R3.2 $\sout{p(X_1,\ldots,X_n) \leftarrow c \wedge q(X_1,\ldots,X_n)}$,
$p(X_1,\ldots,X_n) \leftarrow d$ if $c \rightarrow d$ (subsumption)

$P_i = \{$

$q(X){\leftarrow}Y{=}X{+}1, Y{<}X \wedge r(Y)$

$p(X){\leftarrow}X{>}0 \wedge r(X)$

$p(X){\leftarrow}r(X)$

$\}$

$\xrightarrow{R3}$

$P_{i+1} = \{$

$\sout{q(X){\leftarrow}Y{=}X{+}1, Y{<}X \wedge r(Y)}$

$\sout{p(X){\leftarrow}X{>}0 \wedge r(X)}$

$p(X){\leftarrow}r(X)$

$\}$

R4 - Definition introduction

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

$$P_i = \{$$
$$q(X) \leftarrow Y = X + 1 \wedge r(Y)$$
$$q(X) \leftarrow s(X)$$
$$p(X) \leftarrow q(X)$$
$$\}$$

$$\xrightarrow{R4}$$

$$P_{i+1} = \{$$
$$q(X) \leftarrow Y = X + 1 \wedge r(Y)$$
$$q(X) \leftarrow s(X)$$
$$p(X) \leftarrow Y = X + 1 \wedge r(Y)$$
$$newp(X) \leftarrow p(X) \wedge r(X)$$
$$\}$$

Specialization strategy

Spec(P,c) **{**

$P_s = \emptyset$;

$Def = \{c\}$;

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

Unf = Clause Removal(Unfold(q));

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

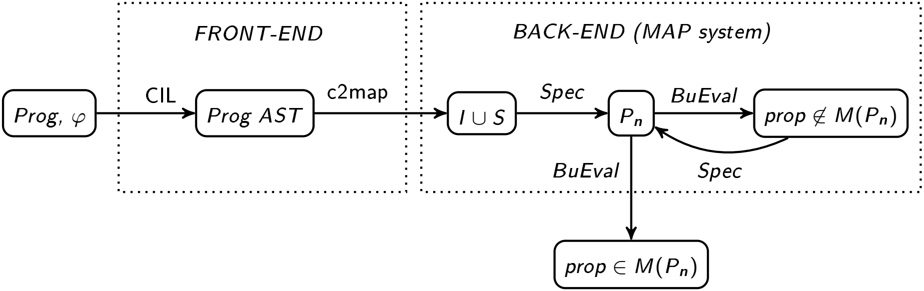
$P_s = P_s \cup$ Fold(Unf, Def)

done

}

$prop \in M(P)$ *iff* $prop \in M(P_s)$

Software Model Checker Architecture - C programs



CIL front-end:
http://http://kerneis.github.com/cil/
by Necula et al.

MAP system:
http://www.iasi.cnr.it/∼proietti/system.html
by the MAP group

Safety checking of C programs

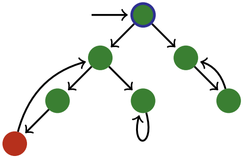
Phase 1: $\mathcal{M} \longrightarrow S, \varphi \longrightarrow prop$

$\mathcal{M} \longrightarrow S =$

```
ureach(X) :- unsafe(X).
ureach(X) :- t(X,X'), ureach(X').
unsafe :- initial(X), ureach(X).
unsafe(cf(error,E)).
initial(cf(call(main,[],id(undef),halt),E)).
```

$\varphi \longrightarrow prop =$

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



Safety checking of C programs

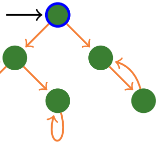
Phase 1: $\mathcal{M} \longrightarrow S, \varphi \longrightarrow prop$

$\mathcal{M} \longrightarrow S =$

```
ureach(X) :- unsafe(X).
ureach(X) :- t(X,X'), ureach(X').
unsafe :- initial(X), ureach(X).
unsafe(cf(error,E)).
initial(cf(call(main,[],id(undef),halt),E)).
```

$\varphi \longrightarrow prop =$

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



Safety checking of C programs

Phase 1: $\mathcal{L} \longrightarrow I$, $Prog \longrightarrow \alpha(Prog)$

$\mathcal{L} \longrightarrow I =$

```
t(cf(asgn(ID,E,L),S),cf(C,S1)) :-
 aeval(E,S,V), update(ID,V,S,S1), cmd(L,C). %ID=E
t(cf(ite(E,L1,_),S),cf(C,S)) :-
 beval(E,S), cmd(L,C).               % if( E ) { L1 }
t(cf(ite(E,_,L2),S),cf(C,S)) :-
 beval(not(E),S), cmd(L2,C).           % else{ L2 }
t(cf(goto(L),S),cf(C,S)) :- cmd(L,C).    % goto(L)
t(cf(call(F,ArgL,OID,Ret),S),cf(goto(Ep),S1)) :-
  prologue(F,ArgL,S,OID,Ret,Ep,S1).
t(cf(ret(E),S),cf(C,S1)) :-
  epilogue(E,S,S1,Ret), cmd(Ret,C).
```

Safety checking of C programs

Phase 2: *Spec* - Specialize $P_0 = I \cup S \cup \alpha Prog$ w.r.t. `initial`

$$\boxed{\text{Spec}(P_0,\texttt{initial}) = P_n}$$

Phase 3: *BuEval* - Bottom up Evaluation of $M(P_n)$

$$\boxed{Prog \text{ is } \mathit{safe} \text{ iff } \mathit{unsafe} \notin M(P_0) \text{ iff } \mathit{unsafe} \notin M(P_n).}$$

Example

Phase 1: from C to CIL

```
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;
}
```

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

  while (1) {
   while_continue:   ;
   if (x<n)  {  }
   else { goto while_break; }
    x = x + 1;
    y = y + 1;
  }
  while_break:   ;

  if (y>x)
   goto ERROR;
  return (0);
}
```

Example

Phase 1: from CIL to CLP

```
int main()
{
  x=0;                 cmd(ℓ0,asgn(id(x),aexp(const(0)),ℓ1)).
  y=0;                 cmd(ℓ1,asgn(id(y),aexp(const(0)),ℓ2)).
  while (1) {
   if (x<n)  { }       cmd(ℓ2,ite(bexp(lt(aexp(id(x)),aexp(id(n)))),ℓ3,ℓ5)).
   else { goto while_break; }
    x = x + 1;         cmd(ℓ3,asgn(id(x),aexp(plus(aexp(id(x)),aexp(const(1)))),ℓ4)).
    y = y + 1;         cmd(ℓ4,asgn(id(y),aexp(plus(aexp(id(y)),aexp(const(1)))),ℓ2)).
   }
  }
  if (y>x)             cmd(ℓ5,ite(bexp(gt(aexp(id(y)),aexp(id(x)))),ℓ6,ℓ7)).
      goto ERROR;      cmd(ℓ6,error).

  return 0;            cmd(ℓ7,ret(aexp(const(0)))).
}
```

Example

Phase 2

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

  while (1) {
   while_continue:   ;
   if (x<n)  {  }  else { goto while_break; }
    x = x + 1; y = y + 1;
  }
  while_break:   ;

  if (y>x)  goto ERROR;
  return (0);
}
```

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

Example

Phase 3

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

↓ BuEval

```
{
 new1(X,Y,N) :- Y>X, X>=N.
 new1(X,Y,N) :- Y>X, N=X+1. %X+1=<N,X'=X+1,Y'=Y+1,X'>=N,Y'>
 new1(X,Y,N) :- Y>X, N=X+2.
 new1(X,Y,N) :- Y>X, N=X+3.
 ....
}
```

BuEval diverges.

Unable to prove *Prog* safe.

Example

Phase 2

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

↓ Spec

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

No facts
BuEval terminates
Prog is safe!

Preliminary results

Simple IMPerative language $SIMP \subset C$

Programs	ARMC	TRACER	MAP
f1a	∞	$\perp$	0.08
f2	∞	$\perp$	7.58
Substring	719.39	180.09	10.20
prog_dagger	∞	$\perp$	5.37
seesaw	3.41	$\perp$	0.03
tracer_prog_d	∞	0.01	0.03
interpolants_needed	0.13	$\perp$	0.06
widen_needed	∞	$\perp$	0.07

Real world C programs (e.g. Device drivers)