Verification of Programs by Transformation of Constraint Logic Programs

Emanuele De Angelis (University “d’Annunzio”, Pescara, Italy),
Fabio Fioravanti (University “d’Annunzio”, Pescara, Italy),
Alberto Pettorossi (University “Tor Vergata”, Rome, Italy),
Maurizio Proietti (IASI-CNR, Rome, Italy),

Dave Schmidt Celebration Symposium, Manhattan, KS, USA
September, 19-20, 2013

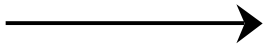
A bit of history on program transformation

- Burstall-Darlington (1977) transformation of functional programs
- Tamaki-Sato (1984) transformation of logic programs
- Etalle-Gabbrielli (1996) transformation of constraint logic programs

Transformation: from correct specifications to efficient implementations

1. 'improving'

fib(0) = a
fib(1) = b
fib(n+2) = fib(n+1) + fib(n)



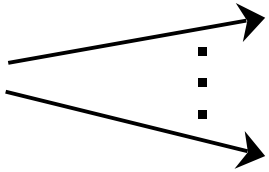
fib(n) = a where <a,b>=r(n)

r(0) = <a,b>
r(2n) = $<u^2+2uv, u^2+v^2>$
r(2n+1) = $<(u+v)^2+u^2, u^2+2uv>$
where <u,v>=r(n)

2. 'proving' (this talk)

3. 'synthesizing'

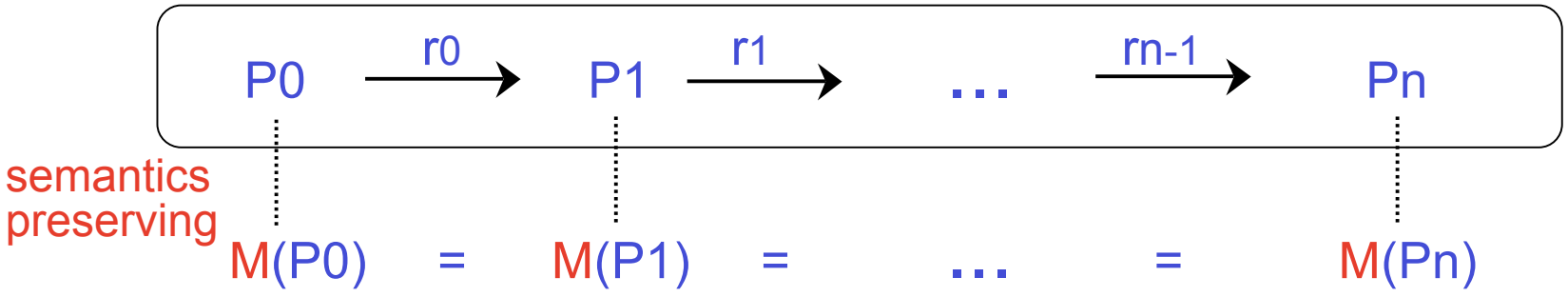
$\forall i,j\ 0 \le i < j \le n \rightarrow A[i] \le A[j]$



merge-sort

quick-sort

Program Transformation



- $P_i \xrightarrow{r} P_{i+1}$ rule $r \in R$
- various semantics M
- need for strategies for rule application

- various settings: functional, logic, and constraint logic programs
- a theoretical limitation: equivalence of programs is undecidable

Program Verification

Parametric framework:

1. program P in a language $\mathcal{L}$
2. property φ in a logic $\mathcal{F}$

The metalanguage is Constraint Logic Programming (CLP).

(a) a1. C-like commands (sequential)
a2. Incorrectness triples

(b) b1. Transition Systems (parallel)
b2. Computational Tree Logic (CTL)

(c) c1. Milner's Calculus of Communicating Systems
c2. Bisimulations

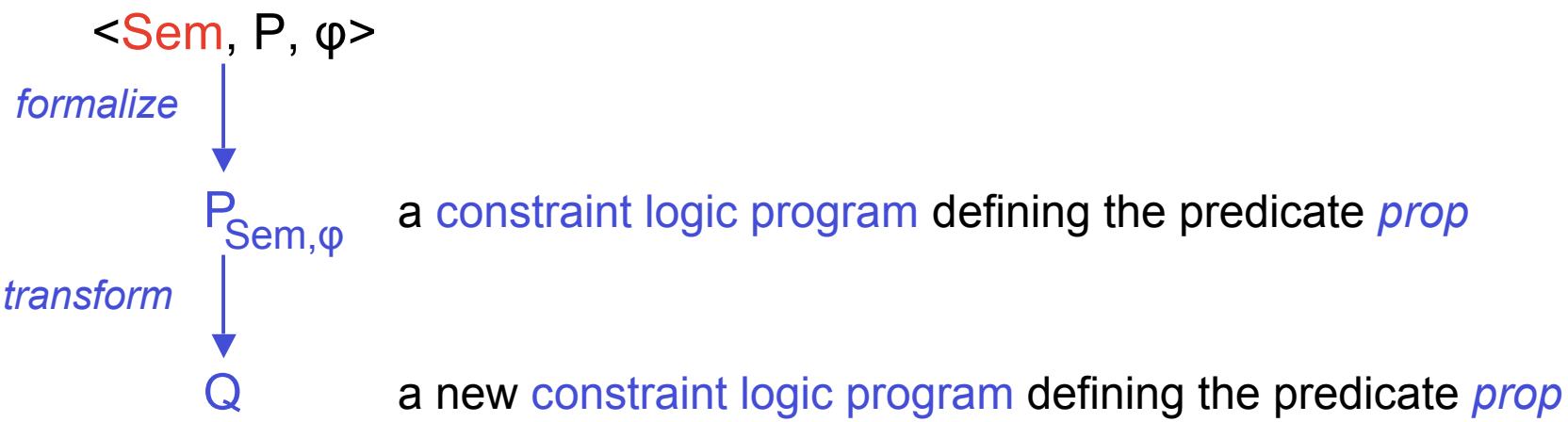
(d) d1. Constraint Logic Programs
d2. First-order predicate calculus properties

Program Verification by Program Transformation

Given a program P and a property φ,

in order to establish: Sem(P) $\models$ φ

we do:



such that Sem(P) $\models$ φ iff M(Q) $\models$ *prop*

(a1) C-like commands and their semantics in CLP

A configuration: $\langle\!\langle$ command c, environment δ $\rangle\!\rangle$ $\qquad \delta : Var \to \mathbb{Z}$

1. $\langle\!\langle \ell : x = e,\ \delta \rangle\!\rangle \Longrightarrow \langle\!\langle at(nextlab(\ell)),\ update(\delta, x, [\![e]\!]\delta) \rangle\!\rangle$
2. $\langle\!\langle \ell : goto\ \ell',\ \delta \rangle\!\rangle \Longrightarrow \langle\!\langle at(\ell'),\ \delta \rangle\!\rangle$
3. $\langle\!\langle \ell : \texttt{if (}e\texttt{)}\ \ell_1\ \texttt{else}\ \ell_2,\ \delta \rangle\!\rangle \Longrightarrow \langle\!\langle at(\ell_1),\ \delta \rangle\!\rangle \qquad$ if $[\![e]\!]\delta \neq 0$
4. $\langle\!\langle \ell : \texttt{if (}e\texttt{)}\ \ell_1\ \texttt{else}\ \ell_2,\ \delta \rangle\!\rangle \Longrightarrow \langle\!\langle at(\ell_2),\ \delta \rangle\!\rangle \qquad$ if $[\![e]\!]\delta = 0$

$\langle\!\langle \_, \_ \rangle\!\rangle$ formalized as `cf(_,_)`

```
1. tr(cf(cmd(L,asgn(int(X),E)),D), cf(cmd(L1,C),D1)) :-
          nextlab(L,L1), at(L1,C), eval(E,D,V), update(D,int(X),V,D1).
2. tr(cf(cmd(L,goto(L1)),D), cf(cmd(L1,C),D)) :- at(L1,C).
3. tr(cf(cmd(L,ite(E,L1,L2)),D), cf(cmd(L1,C),D)) :- V ≠ 0, eval(E,D,V), at(L1,C).
4. tr(cf(cmd(L,ite(E,L1,L2)),D), cf(cmd(L2,C),D)) :- V = 0, eval(E,D,V), at(L2,C).
```

(a2) Incorrectness Triples in CLP

An incorrectness triple: $\boxed{\{\{\varphi_{init}\}\}\ c\ \{\{\varphi_{error}\}\}}$

c $\quad \ell_0 : c_0,\ \ldots,\ \ell_h : \texttt{halt}$ acting on variables $z_1, \ldots, z_n$

$\varphi_{init}(z_1, \ldots, z_r)$ initial configuration

$\varphi_{error}(z_1, \ldots, z_r)$ error configuration

- $\{\{\varphi_{init}\}\}\ c\ \{\{\varphi_{error}\}\}$ iff $\exists \delta_0, \delta_h.\ \ \varphi_{init}(\delta_0(z_1), \ldots, \delta_0(z_r)) \ \wedge$
  $\langle\langle \ell_0 : c_0,\ \delta_0 \rangle\rangle \Longrightarrow^* \langle\langle \ell_h : \texttt{halt},\ \delta_h \rangle\rangle \ \wedge$
  $\varphi_{error}(\delta_h(z_1), \ldots, \delta_h(z_r))$

$\{\{x > 0\}\}\ \ell_0$: `if` $(x < n)\ \ell_1$ `else` ℓ_h; $\quad \ell_1$: $x = x+1$; $\quad \ell_2$: `goto` ℓ_0; $\quad \ell_h$: `halt` $\quad \{\{x < 0\}\}$

```
0. at(0,ite(less(int(x),int(n)),1,h)).
1. at(1,asgn(int(x),plus(int(x),int(1)))).
2. at(2,goto(0)).
h. at(h,halt).
```

(a2) Correctness of the Formalization in CLP

$\Longrightarrow^*$ formalized using `reach`

```
incorrect :- initConf(X), reach(X).
reach(X) :- tr(X,X1), reach(X1).
reach(X) :- errorConf(X).

initConf(cf(cmd(0,C), [[int(x),X],[int(n),N]])) :- at(0,C), phiInit(X).
errorConf(cf(cmd(h,C), [[int(x),X],[int(n),N]])) :- at(h,C), phiError(X).

phiInit(X)  :- X>0.
phiError(X) :- X<0.
```

Theorem. (Correctness of the CLP Formalization)
Let `Tr ∪ At ∪ Reach` be the CLP clauses formalizing $\{\{\varphi_{init}\}\}\ c\ \{\{\varphi_{error}\}\}$.
Then, $\{\{\varphi_{init}\}\}\ c\ \{\{\varphi_{error}\}\}$ does not hold (that is, c is correct)
iff `incorrect` $\notin \mathbf{M}$(`Tr ∪ At ∪ Reach`).

The Transformation Method

Step 1. Removal of the interpreter `tr`
Step 2. Propagation of the constraints of φ_{init} and φ_{error}

Transformation Strategy.

```
TransP = ∅;
Defs = {incorrect :- initConf(X), reach(X)};
while there exists a clause c ∈ Defs do
    Unf = RemoveClause(Unfold(c));
    Defs = (Defs – {c}) ∪ Generalize(Defs);
    TransP = TransP ∪ Fold(Unf,Def)
    od
```

Transformation Rules

r1. Definition Introduction: $\mathtt{newp}(X_1,\ldots,X_n) \leftarrow c \wedge A$

---

r2. Unfolding: $p(X_1,\ldots,X_n) \leftarrow c \wedge \underline{q(X_1,\ldots,X_n)}$

$\underline{q(X_1,\ldots,X_n)} \leftarrow \underline{d_1 \wedge A_1}, \ \ldots, \ \underline{q(X_1,\ldots,X_n)} \leftarrow \underline{d_m \wedge A_m}$

yields

$p(X_1,\ldots,X_n) \leftarrow c \wedge d_1 \wedge A_1, \ \ldots, \ p(X_1,\ldots,X_n) \leftarrow c \wedge d_m \wedge A_m$

---

r3. Folding: $p(X_1,\ldots,X_n) \leftarrow c \wedge \underline{A}$

$\underline{q(X_1,\ldots,X_n)} \leftarrow d \wedge \underline{A}$ and $c \rightarrow d$

yields

$p(X_1,\ldots,X_n) \leftarrow c \wedge \underline{q(X_1,\ldots,X_n)}$

---

r4. Clause Removal:

if c is unsatisfiable or $\big(p(X_1,\ldots,X_n) \leftarrow d$ and $c \rightarrow d\big)$,

then remove $p(X_1,\ldots,X_n) \leftarrow c \wedge q(X_1,\ldots,X_n)$

Verification (1.1)

$\{\{x>0\}\}$ ℓ_0: if $(x<n)$ ℓ_1 else ℓ_h; ℓ_1: $x=x+1$; ℓ_2: goto ℓ_0; ℓ_h: halt $\{\{x<0\}\}$

Initial program: Tr ∪ At ∪ Reach

Step 1. Removal of the interpreter tr.

Program Transformation:

- unfold:

```
incorrect:- X>0, reach(cf(cmd(0,ite(less(int(x),int(n)),1,h)),
                          [[int(x),X],[int(n),N]])).
```

- define:

```
new0(N,X) :- reach(cf(cmd(0,ite(less(int(x),int(n)),1,h)),
                      [[int(x),X],[int(n),N]])).
```

- fold:

```
incorrect:- X>0, new0(N,X).
```

- unfold:

```
new0(N,X) :- X<N, X'=X+1, reach(cf(cmd(0,ite(less(int(x),int(n)),1,h)),
                                   [[int(x),X'],[int(n),N]])).
new0(N,X) :- X<0, X>=N.
```

- fold:

```
new0(N,X) :- X<N, X'=X+1, new0(N,X').
new0(N,X) :- X<0, X>=N.
```

Derived program after removal of the interpreter tr:

```
incorrect:- X>0, new0(N,X).
new0(N,X) :- X<N, X'=X+1, new0(N,X').
new0(N,X) :- X<0, X>=N.
```

Verification (1.2)

Step 2. Propagation of the constraints in φ_{init}.

Program after removal of the interpreter `tr`:

```
incorrect:- X>0, new0(N,X).
new0(N,X) :- X<N, X'=X+1, new0(N,X').
new0(N,X) :- X<0, X>=N.
```

Program Transformation:

- define:
  ```
  new1(N,X) :- X>0, new0(N,X).
  ```
- fold:
  ```
  incorrect:- X>0, new1(N,X).
  ```
- unfold:
  ```
  new1(N,X) :- X>0, X<N, X'=X+1, new0(N,X').
  ~~new1(N,X) :- X>0, X<0, X>=N.~~
  ```
- fold:
  ```
  new1(N,X) :- X>0, X<N, X'=X+1, new1(N,X').
  ```

Derived program P:

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

(•) No constrained facts for `new1`.

`incorrect` $\notin M(P)$.

The C-like program is correct.

Verification (2.1)

$\{\{x=0, y=0\}\}\ \ell_0$: while $(x<n)$ $\{x=x+1;\ y=y+1\};\ \ell_h$: halt $\{\{y>x\}\}$

Program after removal of the interpreter tr:

```
incorrect:- X=0, Y=0, new0(N,X,Y).
new0(N,X,Y) :- X<N, X'=X+1, Y'=Y+1, new0(N,X',Y').
new0(N,X,Y) :- Y>X, X>=N.
```

Program Transformation:

- define:

  ```
  new1(N,X,Y) :- X=0, Y=0, new0(N,X,Y).
  ```
  (0)

- fold:

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

- unfold:

  ```
  new1(N,X,Y) :- X=0, Y=0, X<N, X'=X+1, Y'=Y+1, new0(N,X',Y').
  ```
  (1)

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

  We cannot fold.

- generalize:

  ```
  new2(N,X',Y') :- X'>=0, Y'>=0, new0(N,X',Y').
  ```
  (>=0)

Verification (2.2)

- generalize:

```
new2(N,X',Y') :- X'>=0, Y'>=0, new0(N,X',Y').                    (>=0)
```

- fold:

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

- unfold:

```
new2(N,X,Y) :- X>=0, Y>=0, X<N, X'=X+1, Y'=Y+1, new0(N,X',Y').
new2(N,X,Y) :- X>=0, Y>=0, Y>X, X>=N.
```

- fold:

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

Derived program P:

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

(•) There is a constrained fact for new2.

Unable to prove or disprove correctness of the C-like program.

Need for smart generalizations.

Verification (2.3)

- generalize 'by branching preserving':

  `new2(N,X',Y') :- X'>=0, Y'>=0, X'>=Y', new0(N,X',Y').` (bp)

- fold:

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

- unfold:

  `new2(N,X,Y) :- X>=0, Y>=0, X<N, X>=Y, X'=X+1, Y'=Y+1, new0(N,X',Y').`

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

- fold:

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

New derived program P_{bp} :

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

(•) No a constrained facts for new2.

incorrect $\notin$ M(P_{bp}).

The C-like program is correct.

Need for Good Generalizations

Generalizations should guarantees that a finite number of new definitions are introduced.

- Widening.
- Convex-Hull & Widening.

⋮

Suitable Generalizations (1)

Generalization guarantees that a finite number of new definitions are introduced.

- Widening.

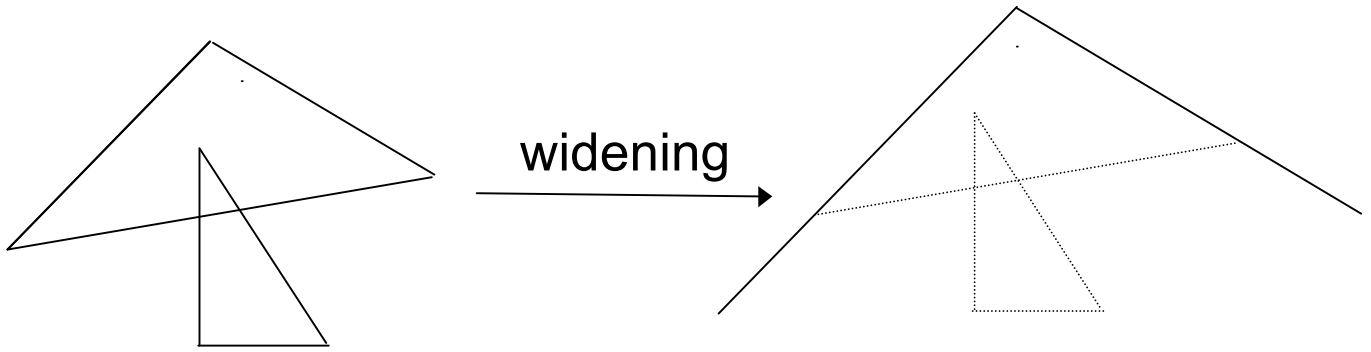
  new1(...) ← X=0 ∧ Y=0 ∧ B(X,Y)

  ⋮

  new2(...) ← X=1 ∧ Y=1 ∧ B(X,Y)

  by widening we get:

  new3(...) ← X>=0 ∧ Y>=0 ∧ B(X,Y)



Suitable Generalizations (2)

- Convex-Hull & Widening.

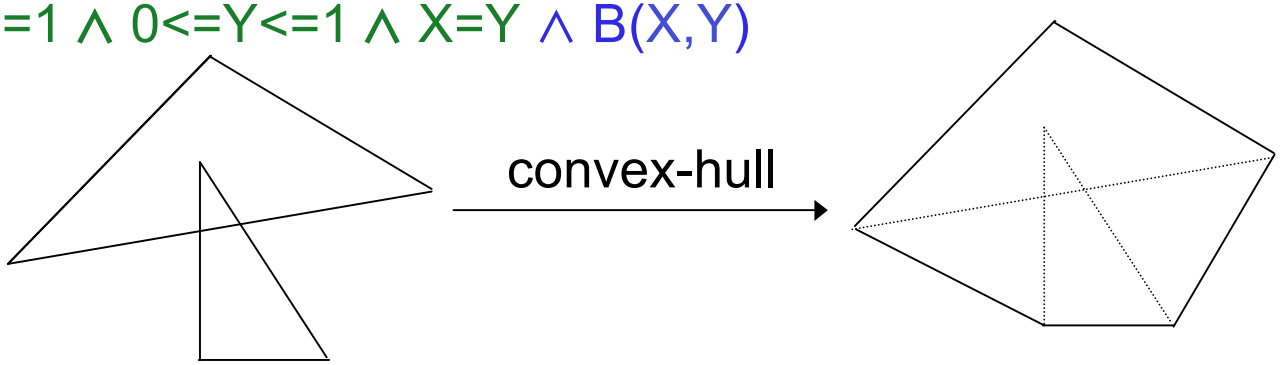
new1(...) ← X=0 ∧ Y=0 ∧ B(X,Y)

⋮

new2(...) ← X=1 ∧ Y=1 ∧ B(X,Y)

by convex-hull we get:

new3(...) ← 0<=X<=1 ∧ 0<=Y<=1 ∧ X=Y ∧ B(X,Y)



⋮

new4(...) ← 1<=X<=2 ∧ 1<=Y<=2 ∧ X=Y ∧ B(X,Y)

then, by widening we get:

new5(...) ← 0<=X ∧ 0<=Y ∧ X=Y ∧ B(X,Y)

Proofs of Array Programs (1)

$\{\{i \geq 0 \wedge n = dim(a) \wedge n \geq 1\}\}$

ℓ_0: $(x=1)$; ℓ_1: `while` $(x<n)$ { $a[i]=a[i-1]+1$; $i=i+1$; } ℓ_h: `halt`

$\{\{\exists j\ (0 \leq j \wedge j<n \wedge j+1<n \wedge a[j]>a[j+1])\}\}$

An execution: $[4,1,3,7,2] \Longrightarrow^* [4,5,6,7,8]$

$$\delta : (Var \to \mathbb{Z}) \cup \underline{(AVar \to (\mathbb{N} \to_{fin} \mathbb{Z}))}$$

5. $\langle\!\langle \ell : a[ie]=e,\ \delta \rangle\!\rangle \Longrightarrow \langle\!\langle at(nextlab(\ell)),\ update(\delta, a, write(\delta(a), [\![ie]\!]\delta, [\![e]\!]\delta)) \rangle\!\rangle$

```
5. tr(cf(cmd(L,asgn(arrayelem(A,IE),E)),D), cf(cmd(L1,C),D1)) :-
     nextlab(L,L1), at(L1,C),
     eval(IE,D,I), eval(E,D,V), lookup(D,array(A),FA), write(FA,I,V,FA1),
     update(D,array(A),FA1,D1).
```

`read(A,I,U)` iff `A[I]` is `U`

`write(A,I,U,B)` iff by assigning to `A[I]` the value `U` we get the array `B`.

Proofs of Array Programs (2)

THEORY OF ARRAYS:

A1. $I=J,\ read(A,I,U),\ read(A,J,V) \rightarrow U=V$

A2. $I=J,\ write(A,I,U,B),\ read(B,J,V) \rightarrow U=V$

A3. $I\neq J,\ write(A,I,U,B),\ read(B,J,V) \rightarrow read(A,J,V)$

EXTRA TRANSFORMATION RULES:

RR1. If $c \rightarrow I=J$ then
replace: $read(A,I,U),\ read(A,J,V)$ by: $read(A,I,U),\ U=V$

RR2. If $c \equiv (read(A,I,U),\ read(A,J,V),\ d)$ and $d \not\rightarrow I\neq J$ and $d \rightarrow U\neq V$ then
add: $I\neq J$

RW1. If $c \rightarrow I=J$ then
replace: $write(A,I,U,B),\ read(B,J,V)$
by: $write(A,I,U,B),\ U=V$

RW2. If $c \not\rightarrow I=J$ and $c \not\rightarrow I\neq J$ then
replace: H :- $c,\ write(A,I,U,B),\ read(B,J,V),\ G$ (*splitting*)
by: H :- $c,\ I=J,\ U=V,\ write(A,I,U,B),\ G$ and
H :- $c,\ I\neq J,\ write(A,I,U,B),\ read(A,J,V),\ G$

RW3. If $c \rightarrow I\neq J$ then
replace: $write(A,I,U,B),\ read(B,J,V)$
by: $write(A,I,U,B),\ read(A,J,V)$

Proofs of Recursively Defined Properties

$\{\{m \geq 1 \wedge n \geq 1\}\}$

ℓ_0: $x = m$;

ℓ_1: $y = n$;

ℓ_2: while $(x \neq y)$ { if $(x > y)$ $x = x - y$; else $y = y - x$; }

ℓ_3: $z = x$;

ℓ_h: halt

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

```
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.
```

Experimental Evaluation

	MAP	ARMC	HSF	TRACER	
				SPost	*WPre*
right answers	185	138	160	91	103
correct problems	154	112	138	74	85
incorrect problems	31	26	22	17	18
wrong answers	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
total time	10717	15788	15770	27757	23259
average time	58	114	99	305	226

Times are in seconds. Timeout = 300 seconds.

Benchmark of 216 programs (DAGGER, TRACER, InvGen, TACAS SVCOMP13).

(b1-b2) Transition Systems

A reactive system as a *Kripke structure*: $\langle S, I, R, L \rangle$

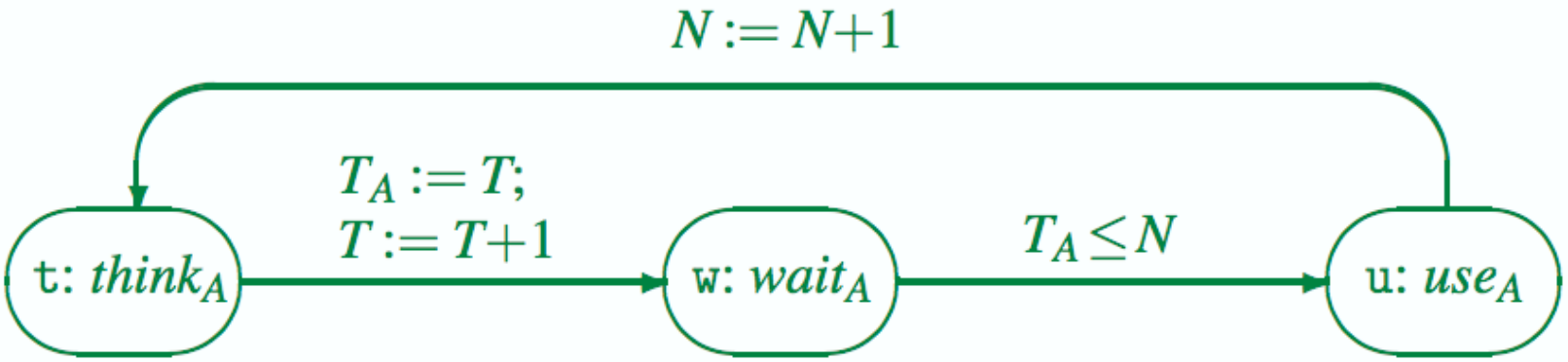
S: the (possibly infinite) set of *states*

I: the set of the *initial states*

R: a total binary *transition relation* denoted $\longrightarrow$

L: a *labeling function*: $S \rightarrow 2^{Elem}$

A *computation path*: an *infinite sequence* of states $s_0\ s_1 \ldots$ such that $\forall i \geq 0,\ s_i \longrightarrow s_{i+1}$.



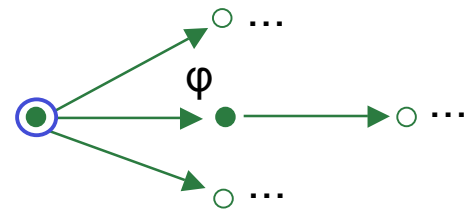
The Ticket Protocol: transitions for process A.

Computational Tree Logic (CTL)

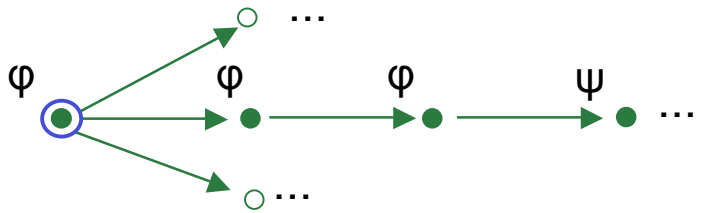
$\varphi ::= e \mid \mathrm{not}(\varphi) \mid \mathrm{and}(\varphi_1, \varphi_2) \mid \mathrm{ex}(\varphi) \mid \mathrm{eu}(\varphi_1, \varphi_2) \mid \mathrm{af}(\varphi)$

where $e \in \mathit{Elem}$, a set of elementary properties.

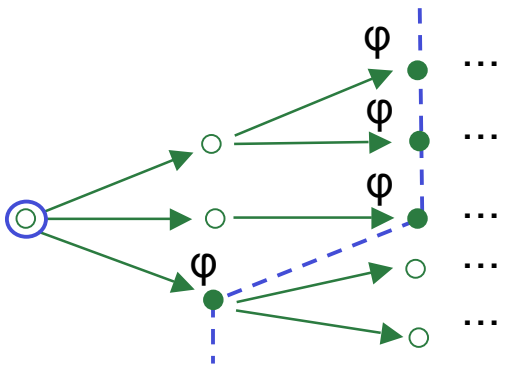
ex(φ) :



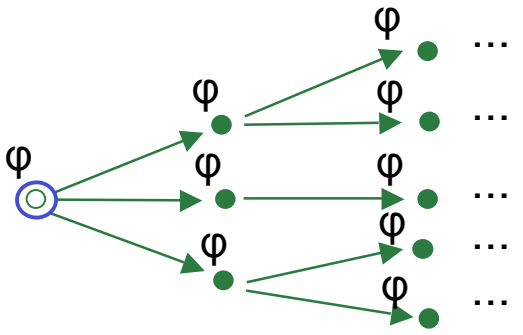
eu(φ,ψ) :



af(φ) :

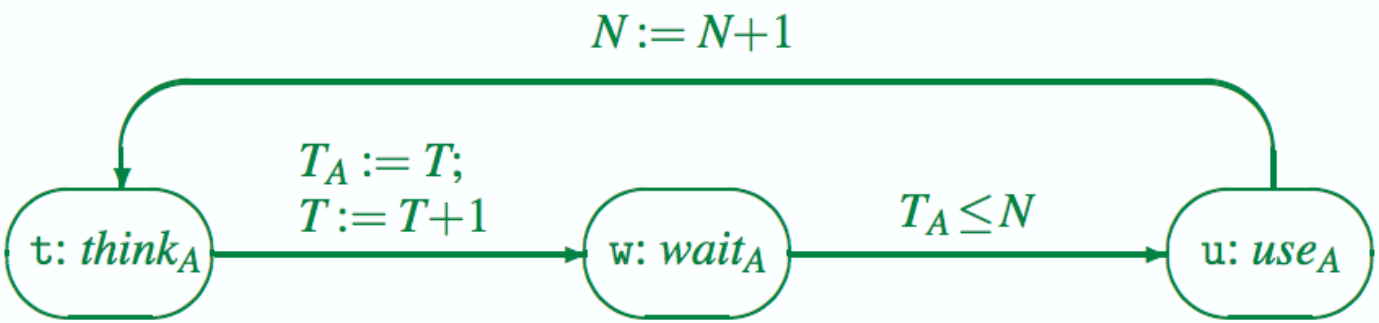


ag(φ) :



Starvation Freedom for process A: $\mathrm{ag}(\mathit{wait}_A \rightarrow \mathrm{af}(\mathit{use}_A))$

Ticket Protocol in CLP



The Ticket Protocol: transitions for process A.

- *state* of process A: $\langle \mathtt{s}_A, T_A \rangle$
  - $\mathtt{s}_A \in \{\mathtt{t}, \mathtt{w}, \mathtt{u}\}$ (that is, $think_A$, $wait_A$, and use_A)
  - T_A : the ticket of process A
- *state* of process B: $\langle \mathtt{s}_B, T_B \rangle$
- *global state*: $\langle \mathtt{s}_A, T_A, \mathtt{s}_B, T_B, T, N \rangle$
  - T: the next ticket
  - N: if $T_A \leq N$, then process A may access the shared resource.

1. $initial(\langle \mathtt{t}, T_A, \mathtt{t}, T_B, T, N \rangle) \leftarrow T = N \wedge T \geq 0$
2. $\mathtt{tr}(\langle \mathtt{t}, T_A, \mathtt{s}_B, T_B, T, N \rangle, \langle \mathtt{w}, T, \mathtt{s}_B, T_B, T1, N \rangle) \leftarrow T1 = T+1$
3. $\mathtt{tr}(\langle \mathtt{w}, T_A, \mathtt{s}_B, T_B, T, N \rangle, \langle \mathtt{u}, T_A, \mathtt{s}_B, T_B, T, N \rangle) \leftarrow T_A \leq N$
4. $\mathtt{tr}(\langle \mathtt{u}, T_A, \mathtt{s}_B, T_B, T, N \rangle, \langle \mathtt{t}, T_A, \mathtt{s}_B, T_B, T, N1 \rangle) \leftarrow N1 = N+1$

and analogous clauses for process B.

Computational Tree Logic in CLP

5. $sat(X,F) \leftarrow elem(X,F)$
6. $sat(X,\mathsf{and}(F_1,F_2)) \leftarrow sat(X,F_1) \wedge sat(X,F_2)$
7. $sat(X,\mathsf{not}(F)) \leftarrow \neg sat(X,F)$
8. $sat(X,\mathsf{ef}(F)) \leftarrow sat(X,F)$
9. $sat(X,\mathsf{ef}(F)) \leftarrow \mathtt{tr}(X,Y) \wedge sat(Y,\mathsf{ef}(F))$
10. $sat(X,\mathsf{af}(F)) \leftarrow sat(X,F)$
11. $sat(X,\mathsf{af}(F)) \leftarrow \mathtt{trs}(X,Ys) \wedge sat\_all(Ys,\mathsf{af}(F))$
12. $sat\_all([],F) \leftarrow$
13. $sat\_all([X|Xs],F) \leftarrow sat(X,F) \wedge sat\_all(Xs,F)$

$elem(X,F)$ iff F is an elementary property holding at state X
$\mathtt{trs}(X,Ys)$ iff Ys is a list of *all* the successor states of X

14. $\mathtt{trs}(\langle \mathtt{t},T_A,\mathtt{t},T_B,T,N\rangle,[\langle \mathtt{w},T,\mathtt{t},T_B,T1,N\rangle,\langle \mathtt{t},T_A,\mathtt{w},T,T1,N\rangle]) \leftarrow T1=T+1$
15. $\mathtt{trs}(\langle \mathtt{w},T_A,\mathtt{t},T_B,T,N\rangle,[\langle \mathtt{u},T_A,\mathtt{t},T_B,T,N\rangle,\langle \mathtt{w},T_A,\mathtt{w},T,T1,N\rangle]) \leftarrow T_A \leq N \wedge T1=T+1$
16. $\mathtt{trs}(\langle \mathtt{w},T_A,\mathtt{t},T_B,T,N\rangle,[\langle \mathtt{w},T_A,\mathtt{w},T,T1,N\rangle]) \leftarrow T_A > N \wedge T1=T+1$
17. $\mathtt{trs}(\langle \mathtt{u},T_A,\mathtt{t},T_B,T,N\rangle,[\langle \mathtt{t},T_A,\mathtt{t},T_B,T,N1\rangle,\langle \mathtt{u},T_A,\mathtt{w},T,T1,N\rangle]) \leftarrow T1=T+1 \wedge N1=N+1$

Starvation Freedom for Ticket Protocol

$prop \equiv_{def} \forall X(initial(X) \rightarrow sat(X, F))$ for some formula represented by the ground term F.

Starvation Freedom for process A:
$F \equiv \mathsf{ag}(wait_A \rightarrow \mathsf{af}(use_A))$, that is, $not(\mathsf{ef}(and(wait_A, not(\mathsf{af}(use_A)))))$.

By Lloyd-Topor transformation:

18. $prop \leftarrow \neg negprop$
19. $negprop \leftarrow initial(X) \wedge sat(X, \mathsf{ef}(and(wait_A, not(\mathsf{af}(use_A)))))$

By program transformation we get:
$prop \leftarrow$

- The Ticket Protocol enjoys Starvation Freedom.

Experimental Evaluation (2)

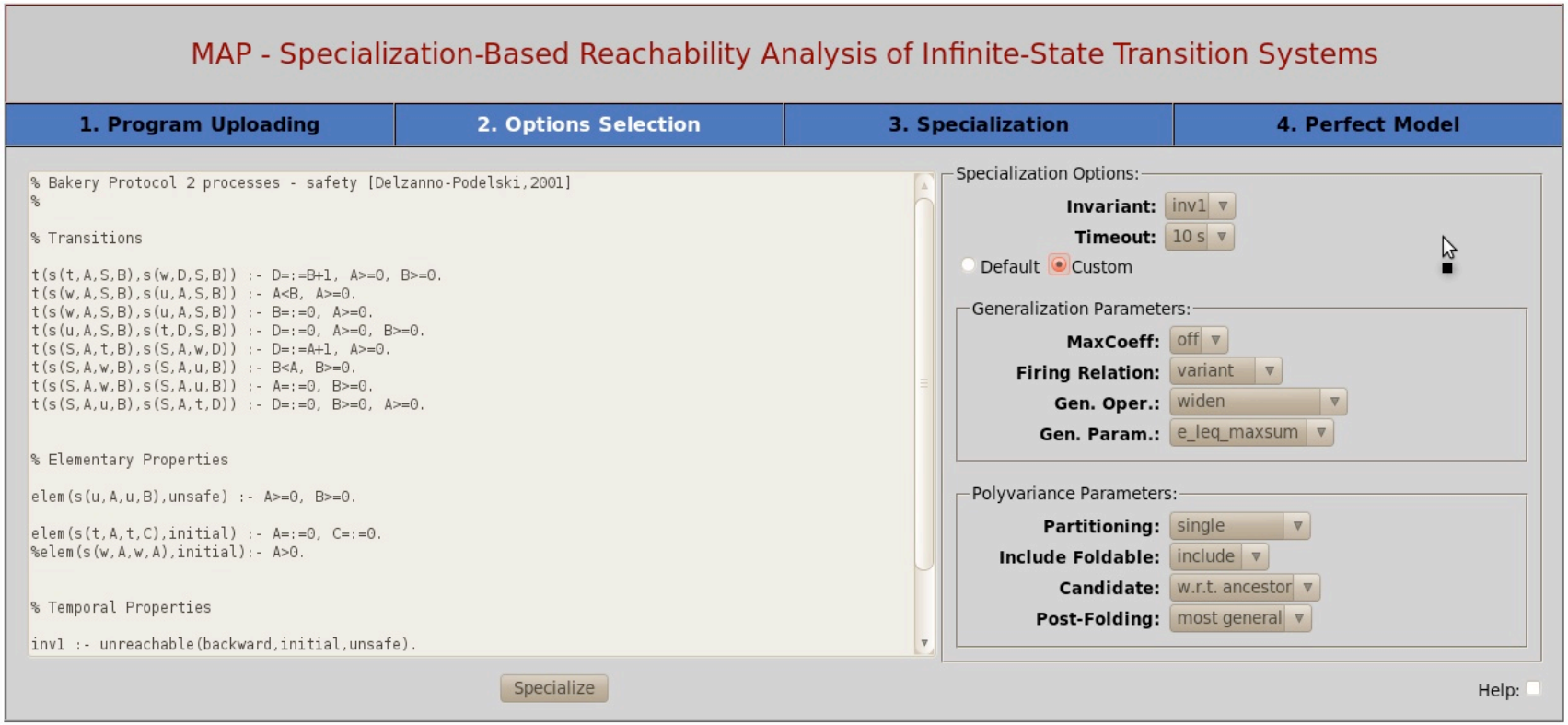
PROGRAM	MAP		ALV				HyTech	
	WM	*WS*	*default*	*A*	*F*	*L*	*Fw*	*Bw*
Bakery (safety)	30	20	20	30	90	30	∞	20
Bakery (liveness)	90	60	30	30	90	30	×	×
Peterson N	190	220	71690	⊥	∞	∞	70	∞
Ticket (safety)	20	20	∞	80	30	∞	∞	∞
Ticket (liveness)	80	110	∞	230	40	∞	×	×
Berkeley RISC	30	30	10	⊥	20	60	∞	20
DEC Firefly	20	20	10	⊥	20	80	∞	20
IEEE Futurebus+	110	2460	320	⊥	∞	670	∞	380
Illinois University	10	30	10	⊥	∞	140	∞	20
MESI	30	30	10	⊥	20	60	∞	20
Synapse N+1	20	20	10	⊥	10	30	∞	10
Xerox PARC Dragon	30	30	20	⊥	40	340	∞	20
Barber	1160	1220	340	⊥	90	360	∞	90
Bounded Buffer	3580	3580	10	10	∞	20	∞	10
Unbounded Buffer	3810	3810	10	10	40	40	∞	20
Consprodjava	25300	∞	∞	∞	∞	∞	∞	∞
CSM	6410	6540	79490	⊥	∞	∞	∞	∞
Consistency	70	60	∞	⊥	∞	∞	∞	2030
Insertion Sort	100	90	40	60	∞	70	∞	10
Selection Sort	∞	180	∞	390	∞	∞	∞	∞
Office Light Control	50	50	20	20	30	20	∞	∞
Reset Petri Net	20	20	∞	⊥	∞	10	∞	10
Kanban	8130	8000	∞	∞	∞	∞	700	∞
Train	30900	57260	42240	⊥	∞	30	∞	∞
no. of verified properties	23	23	17	9	12	16	2	14

Comparison of the MAP, ALV, and HyTech. Times in milliseconds.
'⊥' means 'Unable to verify'. '∞' means 'No answer' within 100 seconds. '×' means 'test not performed'.

MAP

MAP: a tool for program transformation and program specialization.

http://map.uniroma2.it/mapweb



References

E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

P. Cousot and N. Halbwachs. *Automatic discovery of linear restraints among variables of a program.* In: POPL'78, 84–96, 1978.

F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. *Generalization Strategies for the Verification of Infinite State Systems.* Theory and Practice of Logic Programming. Vol. 13, Special Issue 02, 175–199, 2013.

M. Leuschel, T. Massart. *Infinite State Model Checking by Abstract Interpretations and Program Specialization.* In: LOPSTR'99, LNCS 1817, 63–82. Springer, 2000.

ALV: T. Yavuz-Kahveci and T. Bultan. *Action Language Verifier: An Infinite-State Model Checker for Reactive Software Specifications.* Formal Methods in System Design, 35(3):325–367, 2009.

J. C. Peralta, J. P. Gallagher, and H. Saglam. Analysis of Imperative Programs through Analysis of Constraint Logic Programs. Proc. SAS'98, LNCS 1503, 246–261. Springer, 1998.

D. A. Schmidt. *Data flow analysis is model checking of abstract interpretations.* In: POPL'98, 38–48. ACM Press, 1998.

Thanks

to: Anindya, Kyung-Goo, Olivier, and John for their invitation to take part in this celebration.

Ami and John for their help in the travel arrangements.

Dave for his friendship and his beautiful example of dedication to research and teaching. *Ad maiora!*

(c1-c2) Calculus for Communicating Systems (1)

– The infinite set A of *names*. $\forall \ell \in A$ there exists a *co-name*, denoted by $\overline{\ell} \in \overline{A}$.
– The set Act of *actions*, which is $A \cup \overline{A} \cup \{\tau\}$, where τ is a distinguished element.
– The set Id of *identifiers* which are introduced by *definitions* of the form: $P =_{def} p$.

– The set $\mathscr{P}$ of *processes*:

$$p ::= 0 \mid \alpha.p \mid p_1 + p_2 \mid p_1 | p_2 \mid p\backslash L \mid P$$

The operational semantics as a binary relation $\xrightarrow{\alpha} \subseteq \mathscr{P} \times \mathscr{P}$.