Verifying relational program properties by transforming constrained Horn clauses

E. De Angelis[1,3]

joint work with:

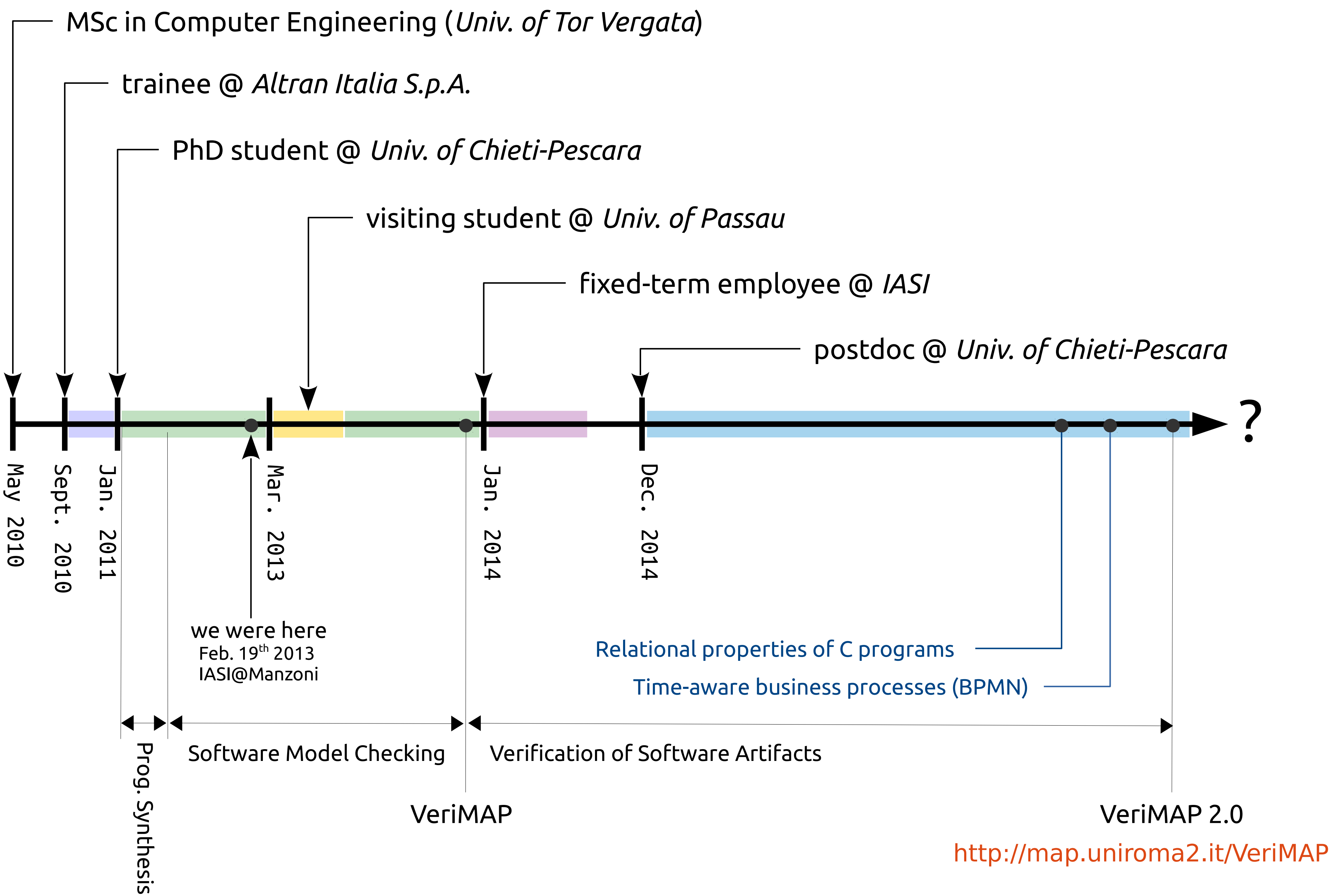
F. Fioravanti[1,3], A. Pettorossi[2,3], and M. Proietti[3]

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

[2] University of Rome `Tor Vergata'

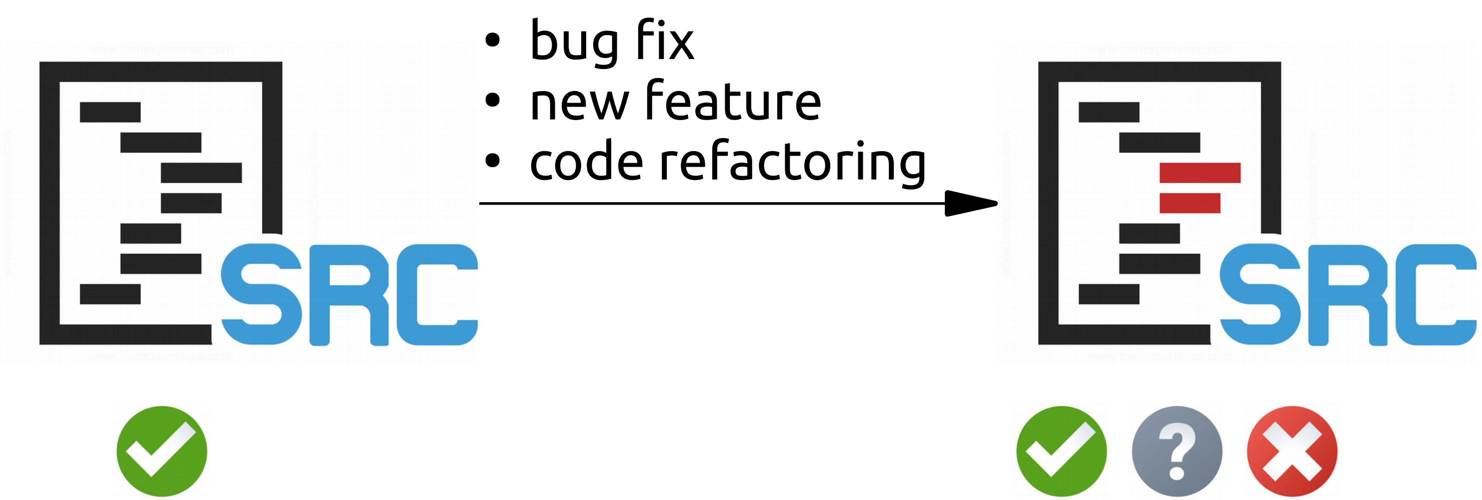
[3] CNR - Istituto di Analisi dei Sistemi ed Informatica

Biosketch



MSc in Computer Engineering (*Univ. of Tor Vergata*)
trainee @ *Altran Italia S.p.A.*
PhD student @ *Univ. of Chieti-Pescara*
visiting student @ *Univ. of Passau*
fixed-term employee @ *IASI*
postdoc @ *Univ. of Chieti-Pescara*
?
May 2010
Sept. 2010
Jan. 2011
Mar. 2013
Jan. 2014
Dec. 2014
we were here
Feb. 19th 2013
IASI@Manzoni
Relational properties of C programs
Time-aware business processes (BPMN)
Prog. Synthesis
Software Model Checking
Verification of Software Artifacts
VeriMAP
VeriMAP 2.0
http://map.uniroma2.it/VeriMAP

Relational verification



Proving **relations** between fragments of program versions may be easier than proving the correctness of the final version from scratch.

Example

P_1:
```
void sum_upto() {
   z1=f(x1);
}
int f(int n1){
   int r1;
   if (n1 <= 0) {
      r1 = 0;
   } else {
      r1 = f(n1 - 1) + n1;
   }
   return r1;
}
```
non-tail recursive

P_2:
```
void prod() {
      z2 = g(x2,y2);
   }
   int g(int n2, int m2){
      int r2;
      r2=0;
      while (n2 > 0) {
         r2 += m2;
         n2--;
      }
      return r2;
   }
```
iterative

global variables of P_1: $\{\mathtt{x1}, \mathtt{z1}\}$

global variables of P_2 : $\{\mathtt{x2}, \mathtt{y2}, \mathtt{z2}\}$

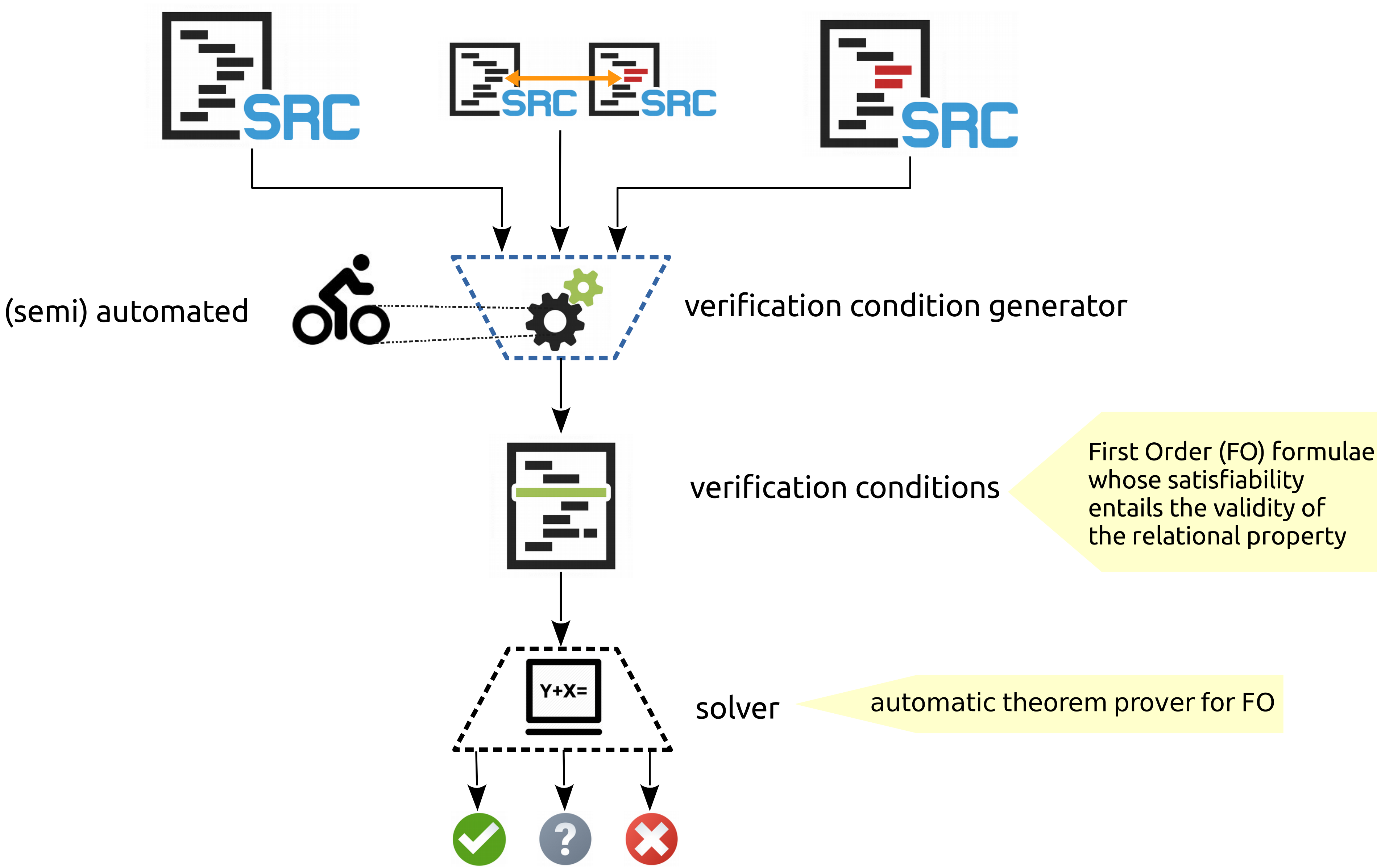
$$z1 = \sum_{i=0}^{x1} i$$

$$z2 = x2 \times y2$$

$$Leq\text{:}\quad \{\mathtt{x1} = \mathtt{x2},\ \mathtt{x2} \leq \mathtt{y2}\}\ \mathtt{sum\_upto} \sim \mathtt{prod}\ \{\mathtt{z1} \leq \mathtt{z2}\}$$

Verification methods

State-of-the-art



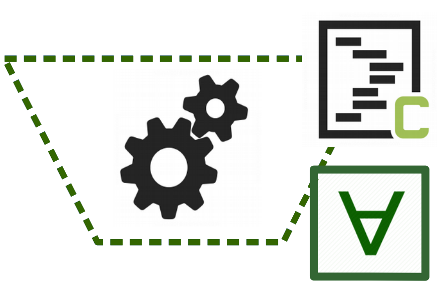
Weakeness

verification condition generator

specific for

programming language

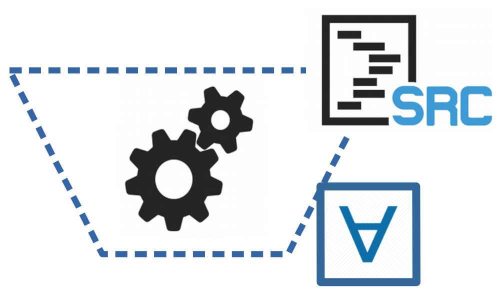
properties



...

Our goal & contribution

Achieve a higher level of **parametricity** with respect to



programming language &

properties

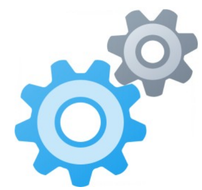
Verification method based on
Transformation of Constrained Horn Clauses (CHCs)

- **CHCs** as a metalanguage for representing  and  as a set of implications of the form $A_0 \leftarrow c, A_1, \ldots, A_n$
- **Transformations** of CHCs to compose clauses representing the programs and the relational property
- **Transformations** increase the effectiveness of solvers

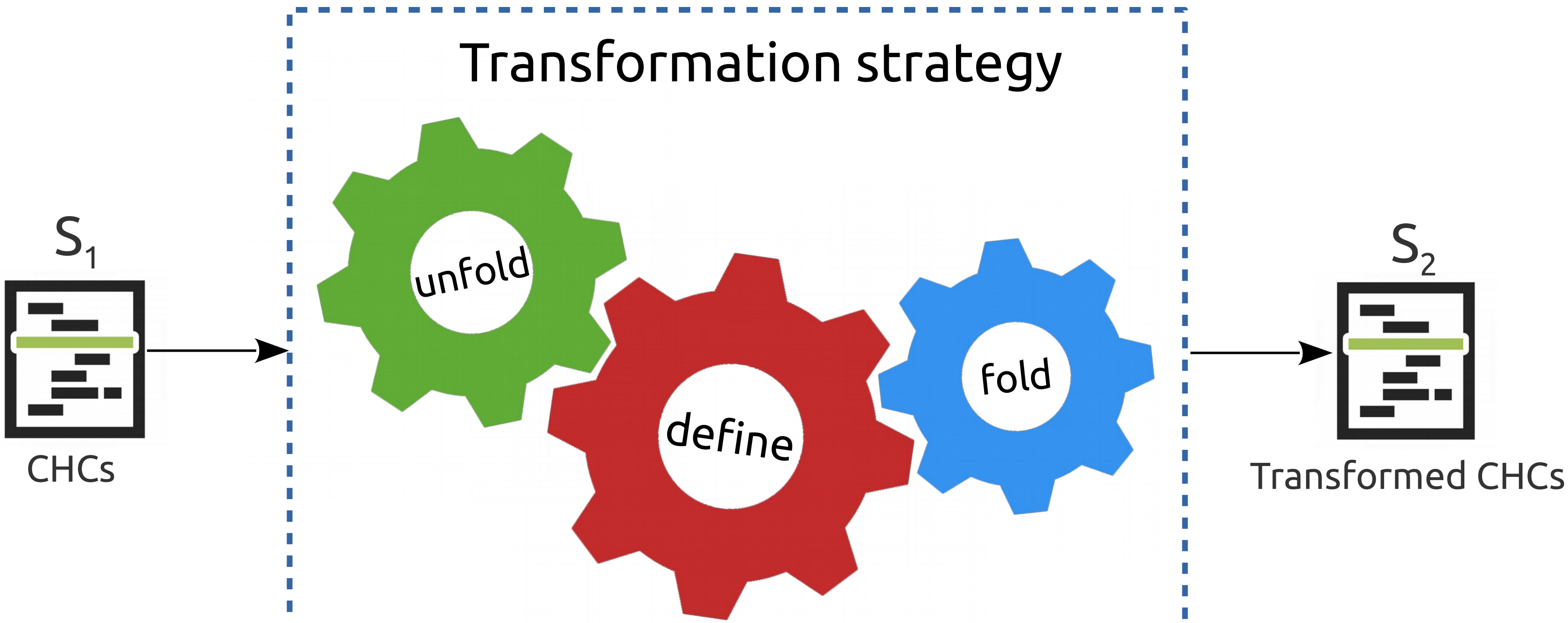
Transformation of CHCs

A technique that
- manipulates clauses
- preserves their satisfiability

Transformation strategies:

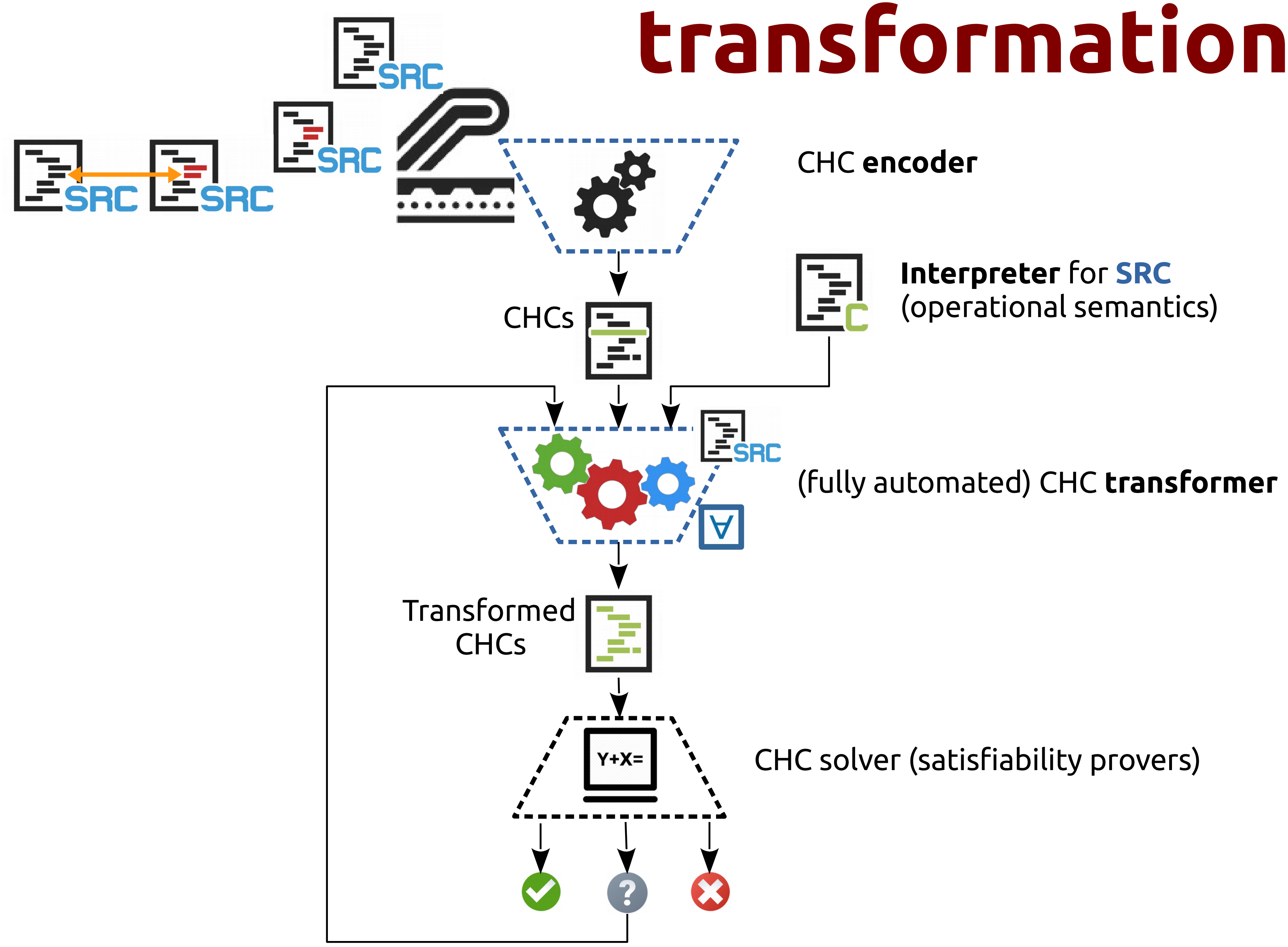


1. CHC specialization
2. Predicate pairing



S_1 is satisfiable *iff* S_2 is satisfiable

Relational verification by CHC transformation



Specifying relational properties using CHCs

The relational property $\{\varphi\}\ P_1 \sim P_2\ \{\psi\}$ is translated into the clause

$$post(X', Y') \leftarrow pre(X, Y), p1(X, X'), p2(Y, Y')$$

pre-relation	φ	$pre(X, Y)$
input/output relation	P_1	$p1(X, X')$
input/output relation	P_2	$p2(Y, Y')$
post-relation	ψ	$post(X', Y')$

Relational property: $\{\mathtt{x1}=\mathtt{x2},\ \mathtt{x2}\leq\mathtt{y2}\}$ $\mathtt{sum\_upto} \sim \mathtt{prod}$ $\{\mathtt{z1}\leq\mathtt{z2}\}$

CHC translation: $Z1' \leq Z2' \leftarrow X1 = X2, X2 \leq Y2, su(X1, Z1'), p(X2, Y2, Z2')$

check the **validity** of a relational property **reduces to** check the **satisfiability** of CHCs

Interpreter (a glimpse)

Operational semantics of the programming language

$$prog(X, X') \leftarrow initConf(C, X), reach(C, C'), finalConf(C', X')$$

input/output relation

initial C and final C' configurations

$$cf(cmd(Label, Command), Environment)$$

$$reach(C, C)$$

$$reach(C, C2) \leftarrow tr(C, C1), reach(C1, C2)$$

`x=e;`

$$tr(cf(cmd(L, asgn(X, expr(E))), Env), cf(cmd(L1, C), Env1)) \leftarrow$$
$$eval(E, Env, V), update(Env, X, V, Env1), nextlab(L, L1), at(L1, C)$$

Interpreters & CHC specialization

Take advantage of static information, that is,

- actual programs
- relational property

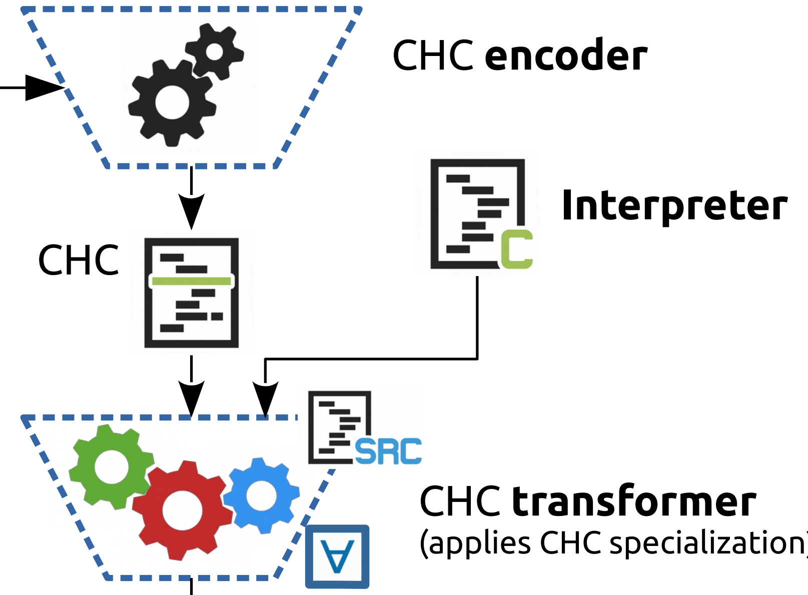
to customize the interpreter

By specializing the interpreter w.r.t. the static input, we get CHCs with no references to

- **reach**
- **tr**
- complex terms representing **configurations**

CHC specialization

P_1:
```
void sum_upto() {
  z1=f(x1);
}
int f(int n1){
  int r1;
  if (n1 <= 0) {
    r1 = 0;
  } else {
    r1 = f(n1 - 1) + n1;
  }
  return r1;
}
```



$$su(X1, Z1') \leftarrow f(X1, Z, X1, R, N1, Z1')$$
$$f(X, Z, N, R, N, 0) \leftarrow N \leq 0$$
$$f(X, Z, N, R, N, Z1) \leftarrow N \geq 1, N1 = N - 1, Z1 = R2 + N, f(X, Z, N1, R1, N2, R2)$$

Satisfiability of CHCs

$\{\varphi\}\ P_1 \sim P_2\ \{\psi\}$

$Z1' \leq Z2' \leftarrow X1 = X2, X2 \leq Y2, su(X1, Z1'), p(X2, Y2, Z2')$

P_1:

$su(X1, Z1') \leftarrow f(X1, Z, X1, R, N1, Z1')$

$f(X, Z, N, R, N, 0) \leftarrow N \leq 0$

$f(X, Z, N, R, N, Z1) \leftarrow N \geq 1, N1 = N - 1, Z1 = R2 + N, f(X, Z, N1, R1, N2, R2)$

P_2:

$p(X2, Y2, Z2') \leftarrow g(X2, Y2, Z, X2, Y2, 0, N, P, Z2')$

$g(X, Y, Z, N, P, R, N, P, R) \leftarrow N \leq 0$

$g(X, Y, Z, N, P, R, N2, P2, R2) \leftarrow N \geq 1, N1 = N - 1, R1 = P + R,$
$\qquad g(X, Y, Z, N1, P, R1, N2, P2, R2)$

- state-of-the-art solvers for CHCs with Linear Integer Arithmetic (LIA) are unable to prove their satisfiability
  - problem: LIA solvers should discover quadratic relations

$$Z1' = X1 \times (X1 - 1)/2 \qquad Z2' = X2 \times Y2$$

- reasoning on su and p separately is unhelpful

Predicate pairing

- **Solution 1**: use a solver for non-linear integer arithmetic
  drawback: satisfiability of constraints is **undecidable**
  (decide satisfiability of Diophantine equations)

- **Solution 2**: **predicate pairing** transformation
  - composes the predicates f and g into a new predicate fg equivalent to their **conjunction**
  - objective: discover linear relations among variables occurring in f and g may help solvers in proving the satisfiability of CHCs

Satisfiability of CHCs

Transformed CHCs

$$Z1' \leq Z2' \leftarrow X1 \leq Y2,\ fg(X1, Z, X1, R, N1, Z1', Y2, Z, 0, N2, P2, Z2')$$
$$fg(X, Z, N, R, N, 0, Y2, V, Z2, N, P2, Z2) \leftarrow N \leq 0$$
$$fg(X, Z, N, R, N, Z1, Y2, V, W, N2, P2, Z2) \leftarrow$$
$$N \geq 1,\ N1 = N-1,\ Z1 = R2+N,\ M = Y2+W,$$
$$fg(X, Z, N1, R1, S, R2, Y2, V, M, N2, P2, Z2)$$

Predicate Pairing makes it possible to infer linear relations among variables in the conjunction fg of predicates f and g

$$fg(X1, Z, N, R, N1, Z1', Y2, V, W, N2, P2, Z2') \leftarrow f(X1, Z, N, R, N1, Z1'), g(X1, Y2, V, N, Y2, W, N2, P2, Z2')$$

The conjunction fg enforces the linear constraint

$$(X1 > Y2) \vee (Z1' \leq Z2')$$

Hence the satisfiability of the first clause

Verification Problems

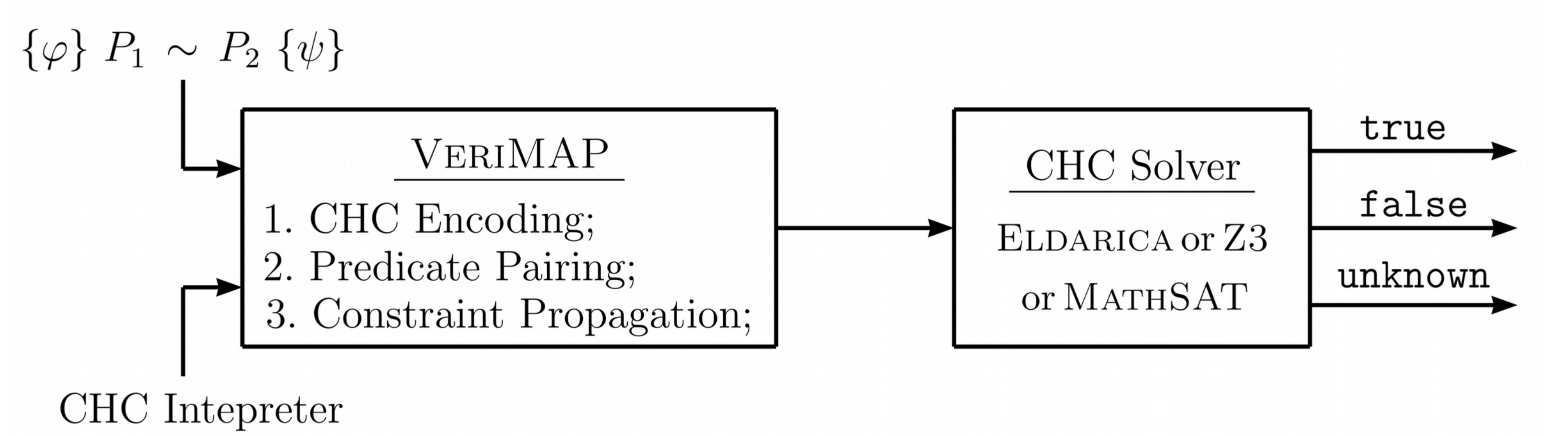
Types of Verified Properties and Programs

- **ITERATIVE**: equivalence of two iterative programs P1, P2

$$X'=Y' \leftarrow p1(X,X'),\ p2(Y,Y'),\ X=Y$$

- **RECURSIVE**: equivalence of two recursive programs
- **ITERATIVE-RECURSIVE**: equivalence of an iterative and a recursive program
- **ARRAYS**: equivalence of two programs on arrays
- **LEQ**: $X' \leqslant Y' \leftarrow p(X,X'),\ \ p(Y,Y'),\ X=Y$
- **MONOTONICITY**: $X' \leqslant Y' \leftarrow p(X,X'),\ \ p(Y,Y'),\ X \leqslant Y$
- **INJECTIVITY**: $X=Y \leftarrow p(X,X'),\ \ p(Y,Y'),\ X'=Y'$
- **FUNCTIONALITY**: $X'=Y' \leftarrow p(X,f(X),X'),\ p(Y,f(Y),Y'),\ X=Y$

Implementation & Experimental Evaluation



Properties	*n*	*Enc+Eld*	*Enc+Z3*	*PP+Eld*	*PP+MS*	*PP+Z3*	*CP+Eld*	*CP+MS*	*CP+Z3*
ITERATIVE	21	7	6	13	19	6	17	20	21
RECURSIVE	18	7	8	7	11	6	14	11	13
ITERATIVE-RECURSIVE	4	0	0	0	3	0	4	4	4
ARRAYS	5	0	1	1	1	4	2	2	5
LEQ	6	1	1	1	6	1	3	6	4
MONOTONICITY	18	4	4	11	16	8	11	17	14
INJECTIVITY	11	0	0	0	11	4	7	11	10
FUNCTIONALITY	7	5	5	6	7	5	6	7	7
Total	90	24	25	39	74	34	64	78	78

Conclusions

A method for **proving relational properties**

- **Independent of the programming language**
  - The only language specific element is the **interpreter**
  - Can be applied to prove relations between programs written in different programming languages
- **Improves effectiveness** of state-of-the art **CHC solvers**