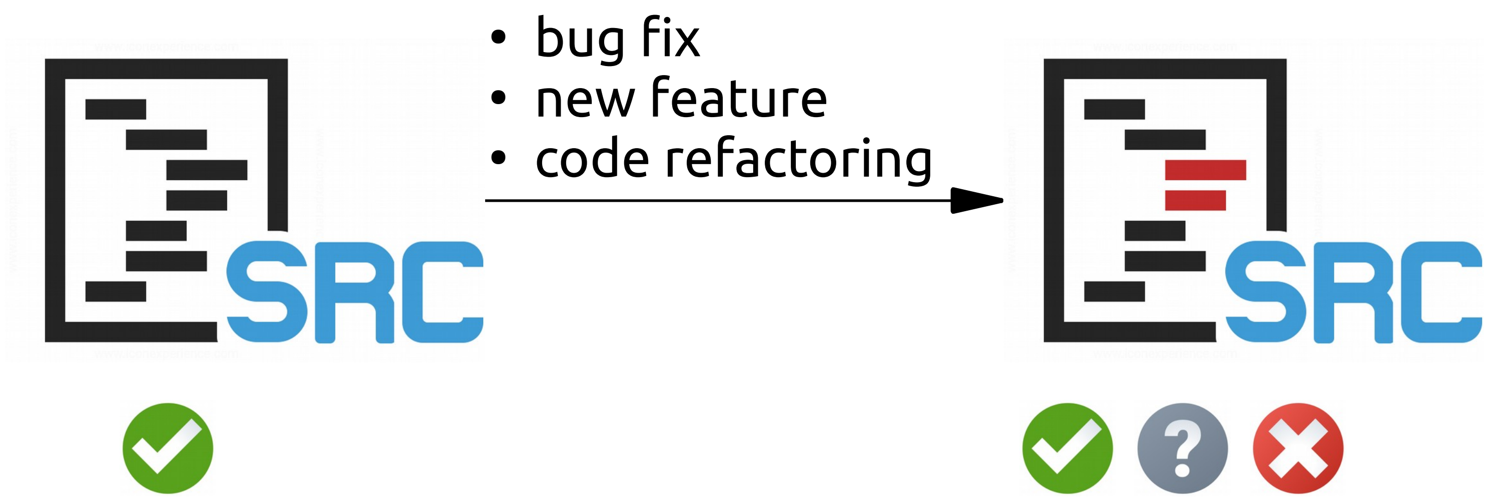
Verifying relational program properties by transforming constrained Horn clauses

E. De Angelis[1], F. Fioravanti[1],
A. Pettorossi[2], 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, Rome

Relational verification



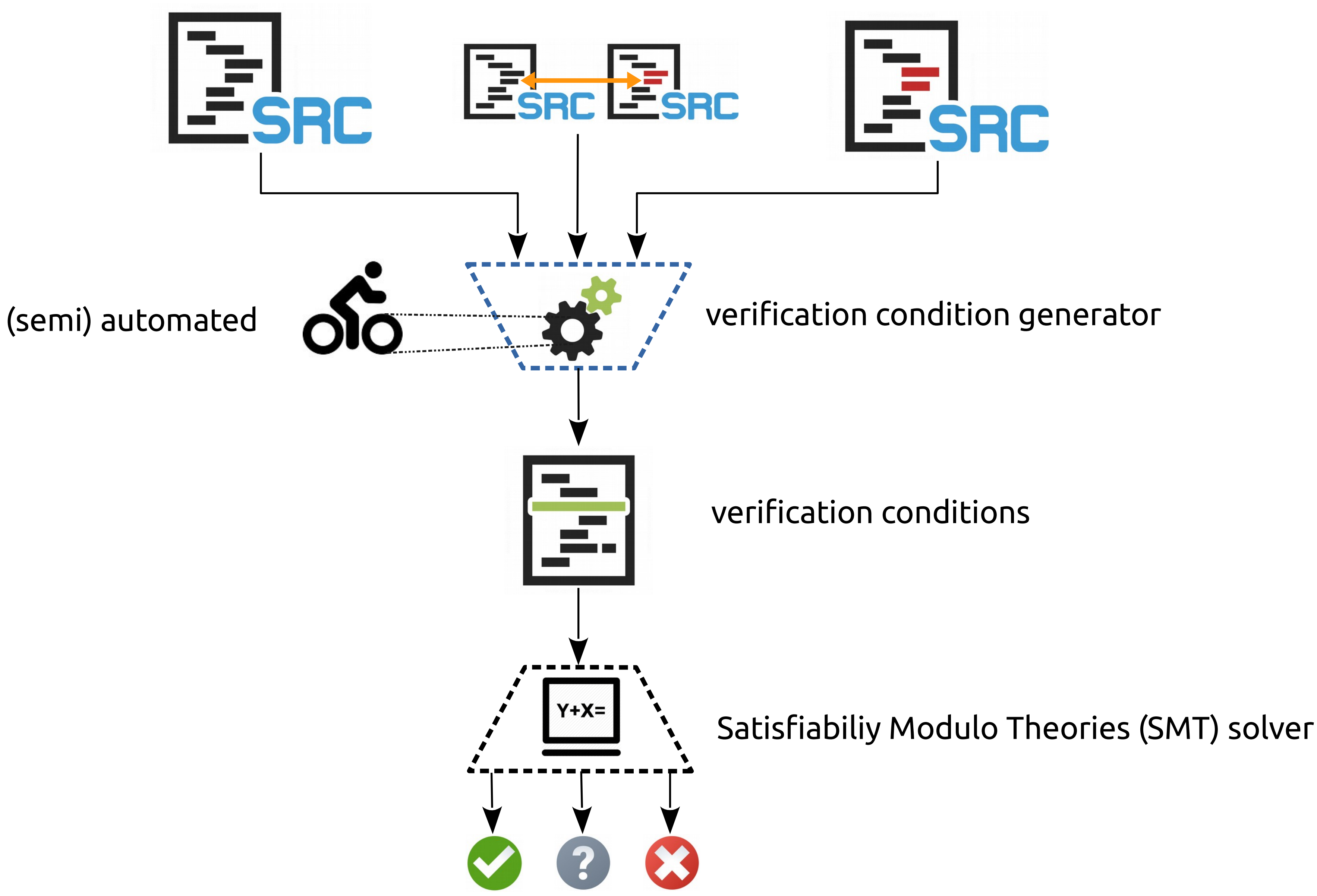
Relational properties

1. modified code still complies with its specification
2. unmodified code has not been affected by the changeset

Example: *program equivalence*

Verification methods

State-of-the-art



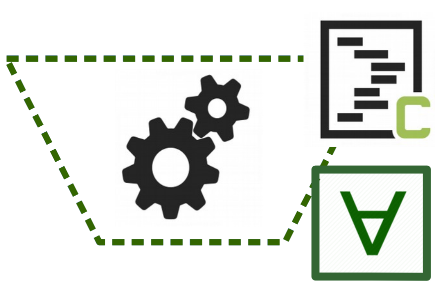
Weakeness

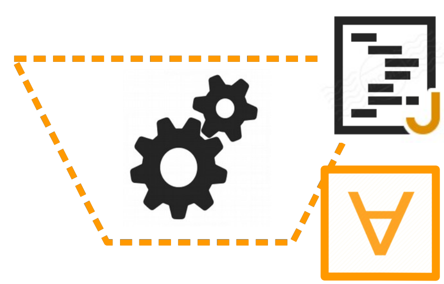
specific for

programming language

properties

verification condition generator

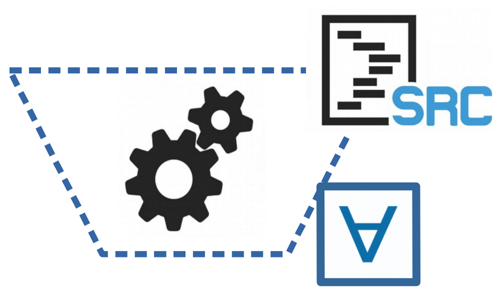




...

Our goal & contribution

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



programming language &

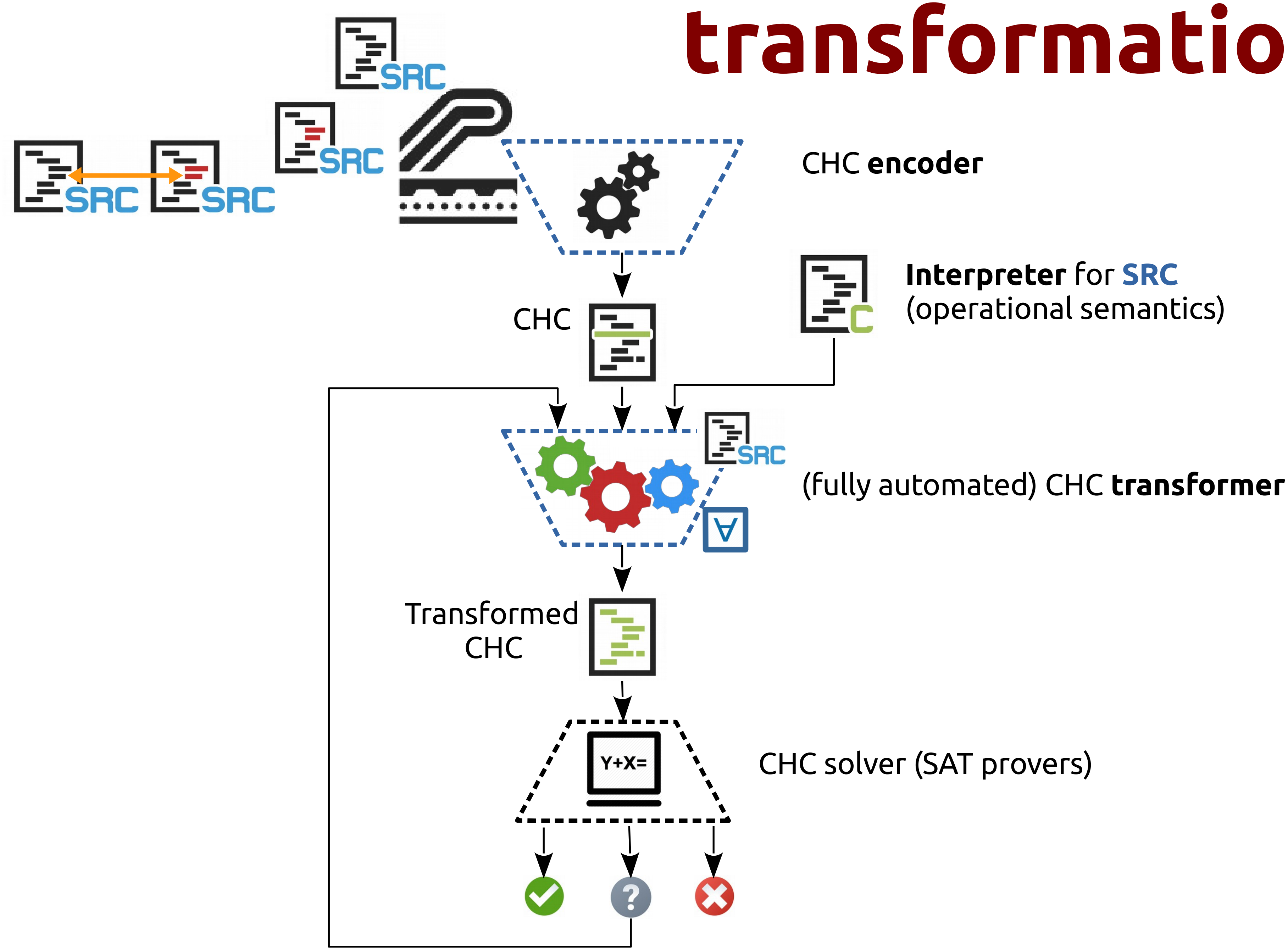
properties

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 satisfiability provers

Relational Verification by CHCs transformation



Specifying Relational Properties using CHCs

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

$$false \leftarrow pre(X,Y), p1(X,X'), p2(Y,Y'), neg\_post(X',Y')$$

pre-relation	φ	$pre(X,Y)$
input/output relation	P_1	$p1(X,X')$
input/output relation	P_2	$p2(Y,Y')$
negation of post-relation	$\neg\psi$	$neg\_post(X',Y')$

check the **validity** of a relational property

reduces to

check the **satisfiability** of its CHCs representation

Interpreter (a glimpse)

Operational semantics of the programming language

$$\underline{prog(X, X')} \leftarrow \underline{initConf(C, X)}, \boxed{reach(C, C')}, \underline{finalConf(C', X')}$$

input/output relation

initial C and final C' configurations

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

$$\boxed{\begin{array}{l} reach(C, C) \\ reach(C, C2) \leftarrow \boxed{tr(C, C1)}, reach(C1, C2) \end{array}}$$

`x=e;`

$$\boxed{\begin{array}{l} tr(cf(cmd(L, asgn(X, expr(E))), Env), cf(cmd(L1, C), Env1)) \leftarrow \\ \quad eval(E, Env, V), update(Env, X, V, Env1), nextlab(L, L1), at(L1, C) \end{array}}$$

Transformation of CHCs

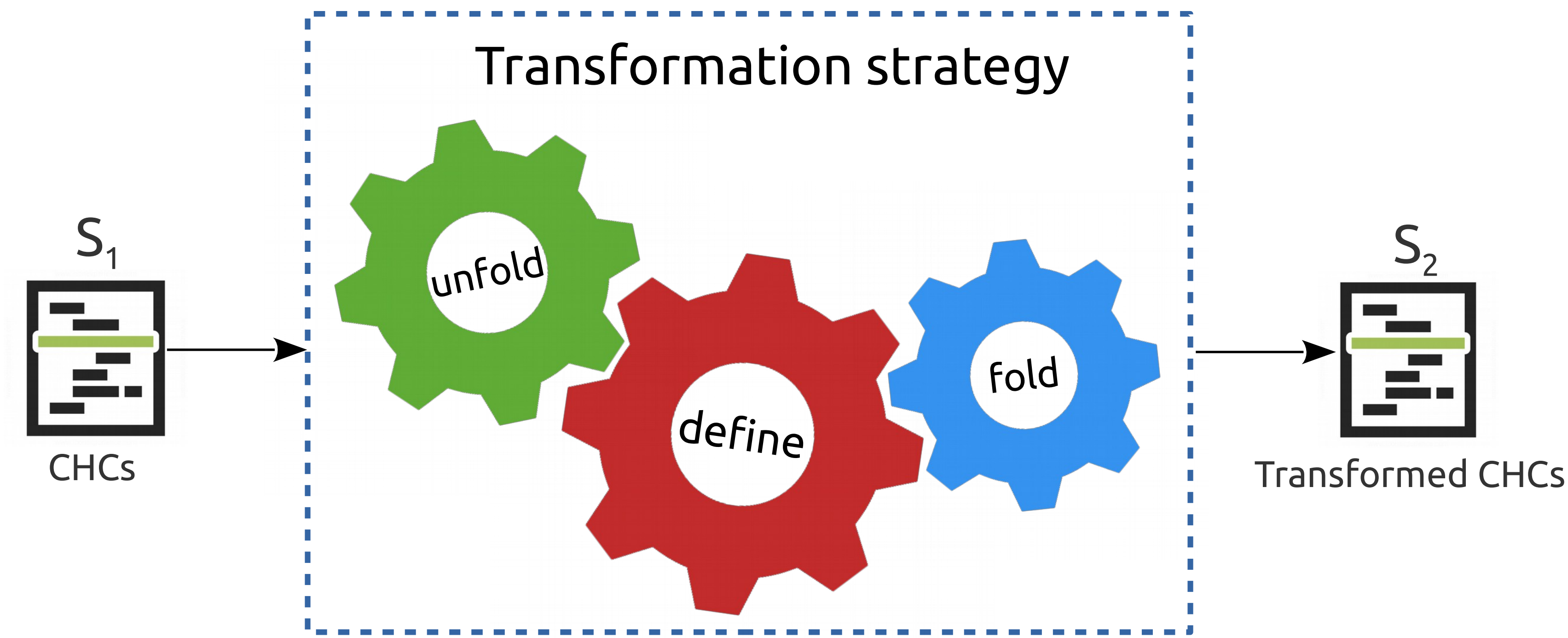
CHS solvers are often unable to prove satisfiability

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

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**

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 $\mathcal{V}_1$={ x1,z1} of P_1

global variables $\mathcal{V}_2$={ x2,y2,z2}of P_2

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

$$\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$$

$$z2=x2\times y2$$

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

CHCs specialization (1)

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;
}
```

CHCs

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

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

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

CHCs specialization (2)

P_2:
```
void prod() {
   z2 = g(x2,y2);
}
int g(int n2, int m2){
   int r2;
   r2=0;
   while (n2>= 1) {
     r2 += m2;
     n2--;
   }
   return r2;
}
```

CHCs

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

$$\begin{cases} g(X, Y, Z, N, P, R, N, P, R) \leftarrow N \leq 0 \\ g(X, Y, Z, N, P, R, N2, P2, R2) \leftarrow N \geq 1, N1 = N - 1, R1 = P + R, \\ \qquad g(X, Y, Z, N1, P, R1, N2, P2, R2) \end{cases}$$

Satisfiability of CHCs

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

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

$$P_1 \left\{ \begin{array}{l} su(X1, Z1') \leftarrow f(X1, Z, X1, R, N1, Z1') \\ f(X, Z, N, R, N, 0) \leftarrow N \leq 0 \\ f(X, Z, N, R, N, Z1) \leftarrow N \geq 1, N1 = N-1, Z1 = R2+N, f(X, Z, N1, R1, N2, R2) \end{array} \right.$$

$$P_2 \left\{ \begin{array}{l} p(X2, Y2, Z2') \leftarrow g(X2, Y2, Z, X2, Y2, 0, N, P, Z2') \\ g(X, Y, Z, N, P, R, N, P, R) \leftarrow N \leq 0 \\ g(X, Y, Z, N, P, R, N2, P2, R2) \leftarrow N \geq 1, N1 = N-1, R1 = P+R, \\ \qquad\qquad g(X, Y, Z, N1, P, R1, N2, P2, R2) \end{array} \right.$$

- state-of-the-art solvers for CHCs with Linear Integer Arithmetic (LIA) are unable to prove their satisfiability

  to prove their satisfiability, that is, the premise of clause (*) unsatisfiable, LIA solvers should discover quadratic relations

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

- reasoning on su and p separately is unhelpful

Predicate Pairing

- **Solution 1**: use a solver for non-linear integer arithmetic

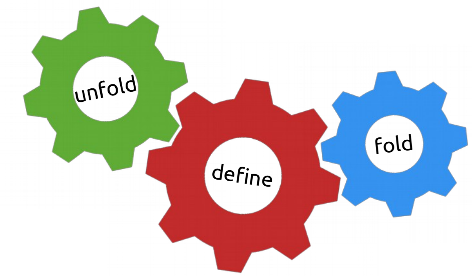
  drawback: satisfiability of constraints is undecidable

- **Solution 2**: use the transformation, again!

  Predicate Pairing transformation strategy

  - composes the predicates f and g into a new predicate fg equivalent to their conjunction
  - Objective: to discover relations among variables occurring in f and g to help the solvers in proving the satisfiability of the CHCs

Predicate Paring in action



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

1. Unfold

$$\begin{aligned} false \leftarrow\ & X1 \leq Y2, Z1' > Z2', \\ & f(X1, Z, X1, R, N1, Z1'), g(X1, Y2, Z, X1, Y2, 0, N2, P2, Z2') \end{aligned}$$

2. Define

$$\boxed{\begin{aligned} & fg(X1, Z, N, R, N1, Z1', Y2, V, W, N2, P2, Z2') \leftarrow \\ & \qquad f(X1, Z, N, R, N1, Z1'), g(X1, Y2, V, N, Y2, W, N2, P2, Z2') \end{aligned}}$$

3. Fold

$$false \leftarrow X1 \leq Y2, Z1' > Z2', fg(X1, Z, X1, R, N1, Z1', Y2, Z, 0, N2, P2, Z2')$$

Satisfiability of CHCs

Transformed CHCs

$$
\begin{aligned}
&\mathit{false} \leftarrow X1 \le Y2,\ \ Z1' > Z2',\ \ \mathit{fg}(X1, Z, X1, R, N1, Z1', Y2, Z, 0, N2, P2, Z2') \\
&\mathit{fg}(X, Z, N, R, N, 0, Y2, V, Z2, N, P2, Z2) \leftarrow N \le 0 \\
&\mathit{fg}(X, Z, N, R, N, Z1, Y2, V, W, N2, P2, Z2) \leftarrow \\
&\qquad\qquad N \ge 1,\ \ N1 = N-1,\ \ Z1 = R2+N,\ \ M = Y2+W, \\
&\qquad\qquad \mathit{fg}(X, Z, N1, R1, S, R2, Y2, V, M, N2, P2, Z2)
\end{aligned}
$$

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

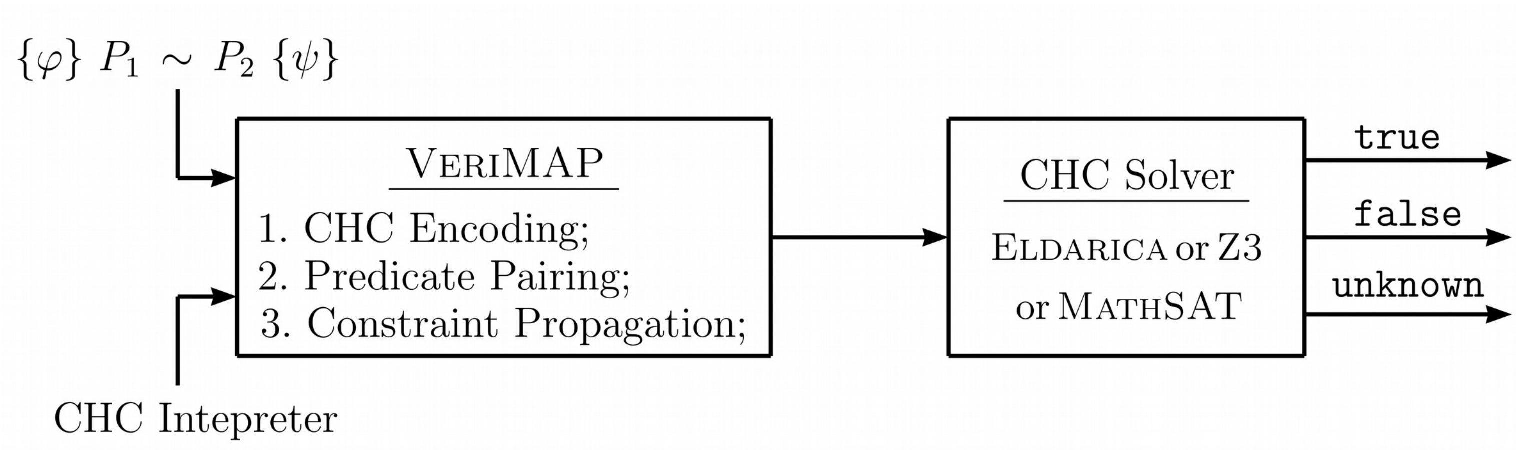
$$
\boxed{
\begin{aligned}
&\mathit{fg}(X1, Z, N, R, N1, Z1', Y2, V, W, N2, P2, Z2') \leftarrow \\
&\qquad f(X1, Z, N, R, N1, Z1'), g(X1, Y2, V, N, Y2, W, N2, P2, Z2')
\end{aligned}
}
$$

The conjunction fg enforces the linear constraint

$$\boxed{(X1 > Y2) \vee (Z1' \le Z2')}$$

Hence the satisfiability of the first clause

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**
  - translating the property in CHCs
  - transforming the CHCs to better exploit the interaction between predicates
- **Independent of the programming language**
  - The only language specific element is the **interpreter**
  - Can be applied to prove relations between programs that may be written in different programming languages
- **Improves effectiveness** of state-of-the art **CHC solvers**