Constrained Horn Clauses in Verification: 10 Years later

Philipp Rümmer
Uppsala University

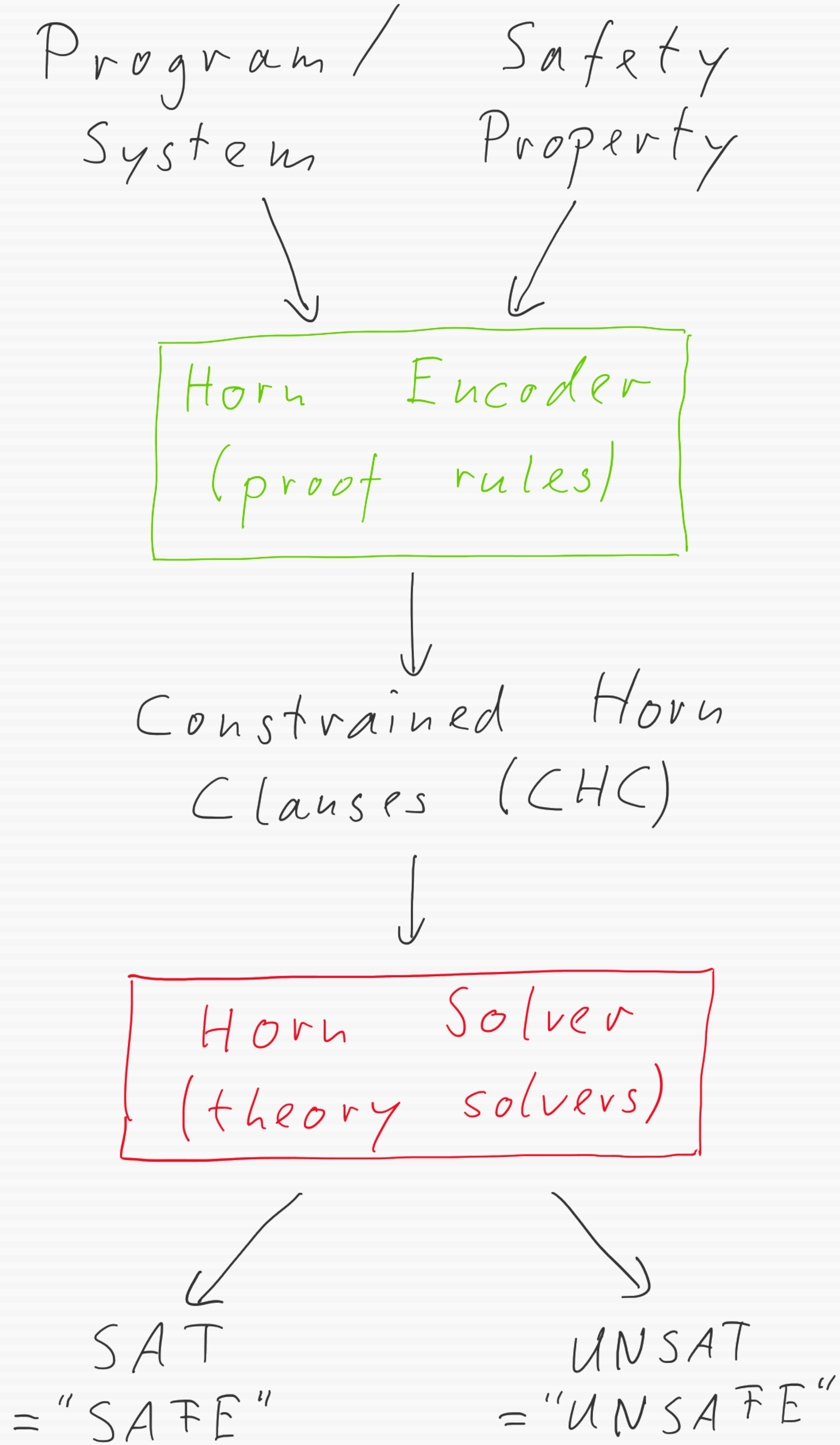
7th September 2020
LOPSTR, AoE

Joint work with ...

- Anoud Alshnakat
- Peter Backeman
- Marc Brockschmidt
- Zafer Esen
- Florent Garnier
- Dilian Gurov
- Hossein Hojjat
- Radu Iosif
- Temesghen Kahsai
- Rody Kersten
- Filip Konecny
- Viktor Kuncak
- Jerome Leroux
- Chencheng Liang
- Christian Lidström
- Huascar Sanchez
- Martin Schäf
- Ali Shamakhi
- Pavle Subotic
- Wang Yi
- Aleksandar Zeljic

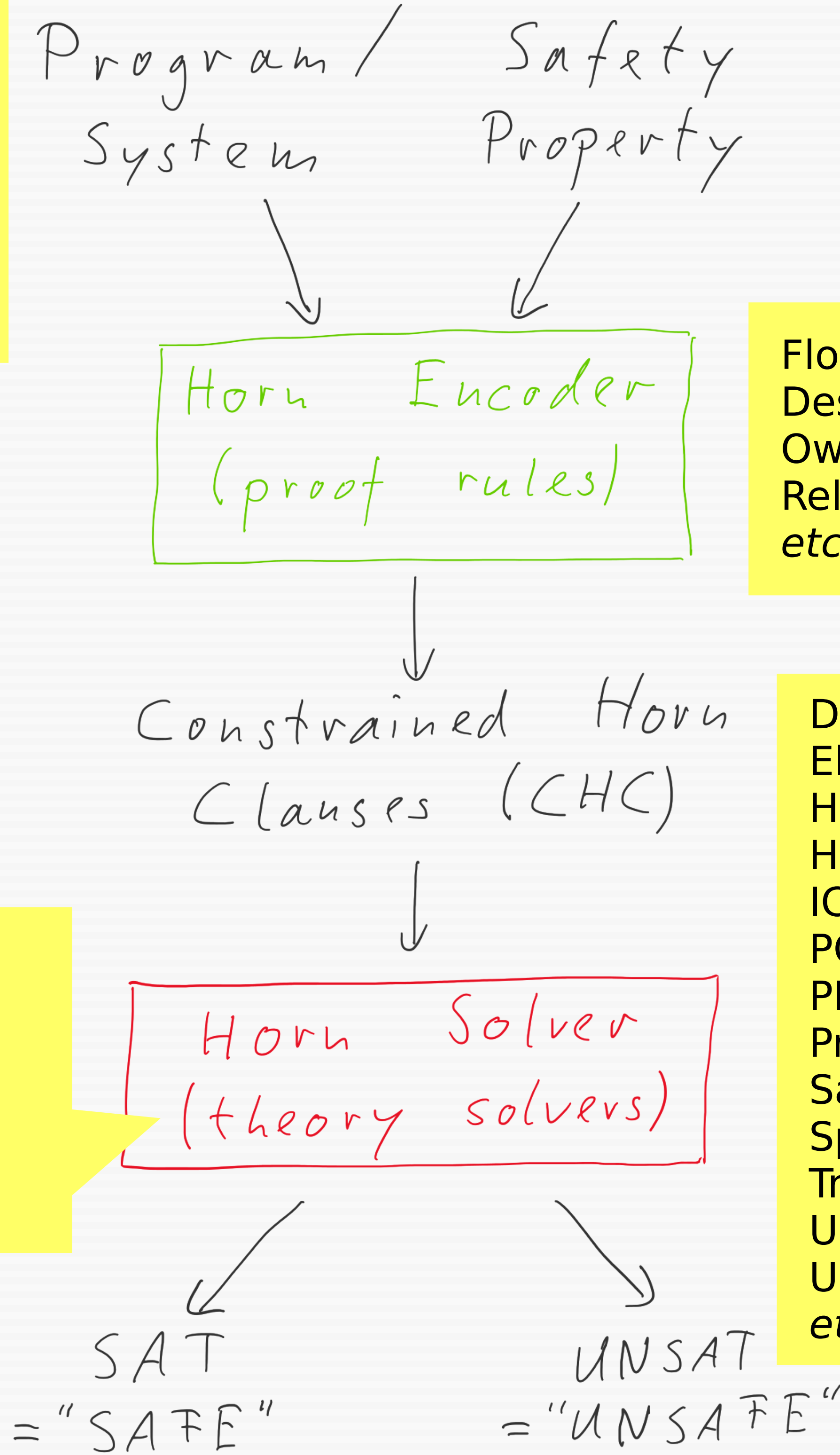
+ based on the work of many other people!

- Anoud Alshnakat
- Peter Backeman
- Marc Brockschmidt
- Zafer Esen
- Florent Garnier
- Dilian Gurov
- Hossein Hojjat
- Radu Iosif
- Temesghen Kahsai
- Rody Kersten
- Filip Konecny
- Viktor Kuncak
- Jerome Leroux
- Chencheng Liang
- Christian Lidström
- Huascar Sanchez
- Martin Schäf
- Ali Shamakhi
- Pavle Subotic
- Wang Yi
- Aleksandar Zeljic



Program / System
Safety Property
Horn Encoder
(proof rules)
Constrained Horn Clauses (CHC)
Horn Solver
(theory solvers)
SAT
= "SAFE"
UNSAT
= "UNSAFE"

C
Java
Ada
Rust
Networks of TA
BIP models
etc.

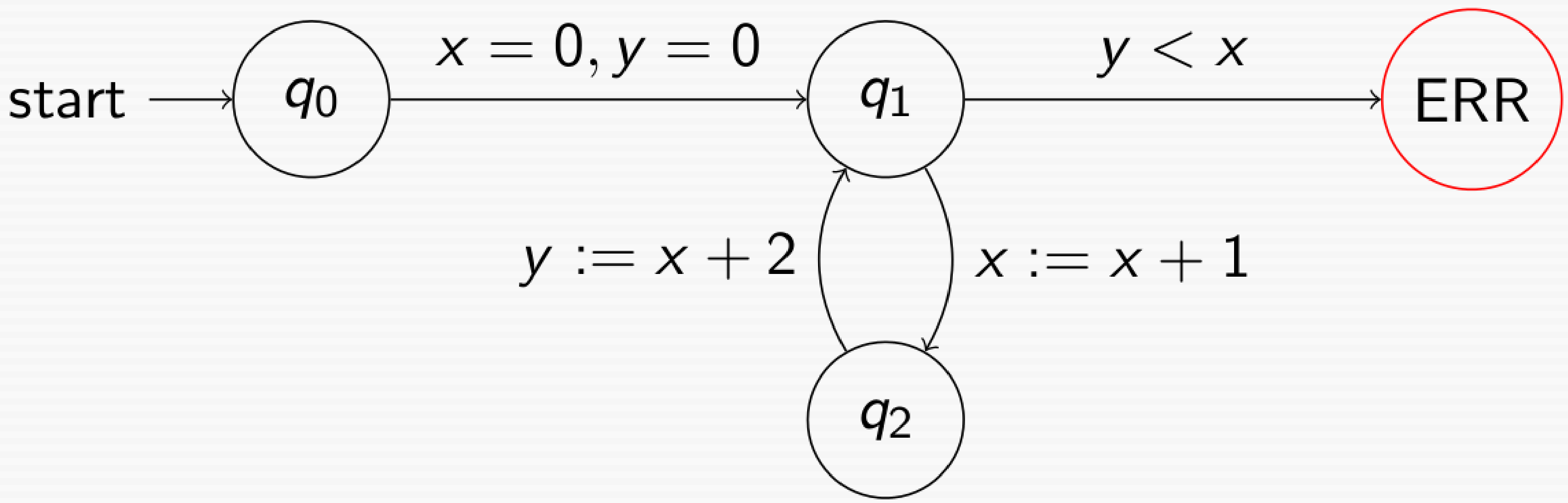


Floyd-Hoare
Design by contract
Owicki-Gries
Rely Guarantee
etc.

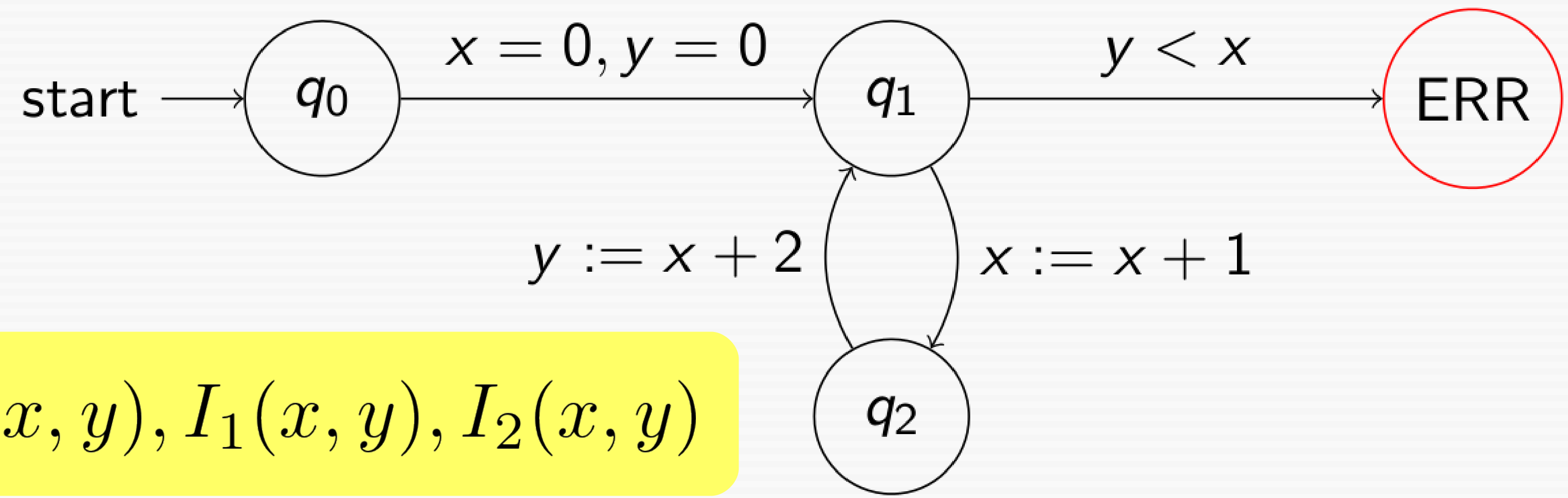
Duality
Eldarica(-abs)
Hoice
HSF
IC3IA
PCSat
PECOS
ProphIC3
Sally
Spacer
TransfHORNer
Ultimate TreeAutomizer
Ultimate Unihorn
etc.

Linear Integers
Linear Rationals
Bit-vectors
Algebraic data-types
Arrays
etc.

Ex 1: Floyd-style invariants

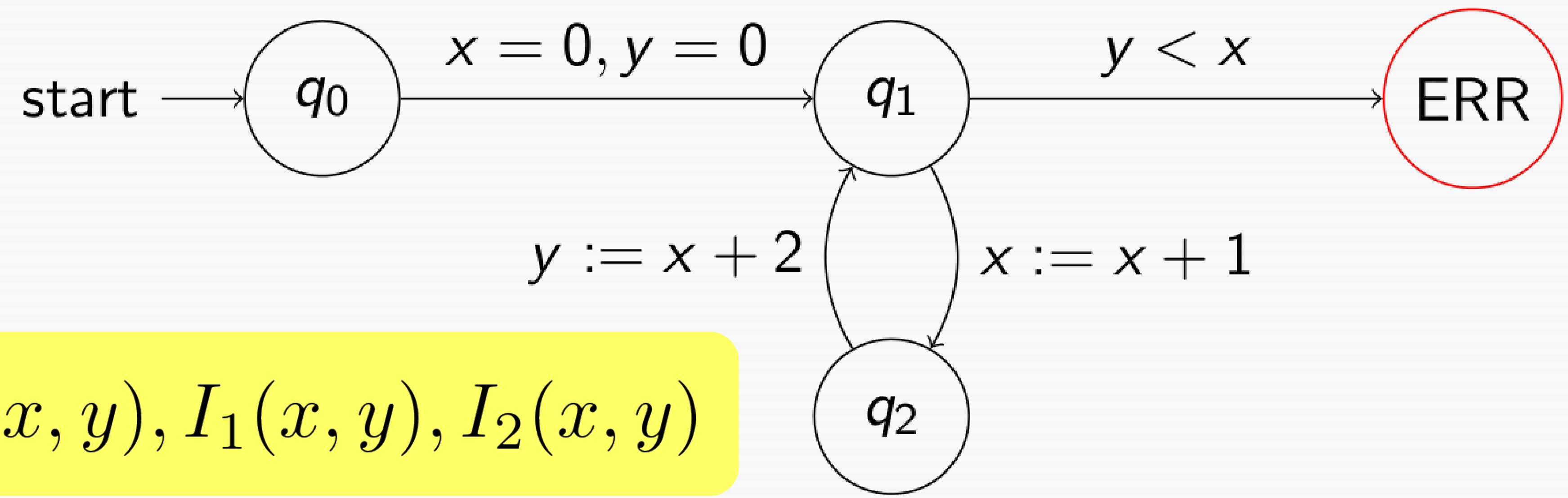


Ex 1: Floyd-style invariants



$$I_0(x, y), I_1(x, y), I_2(x, y)$$

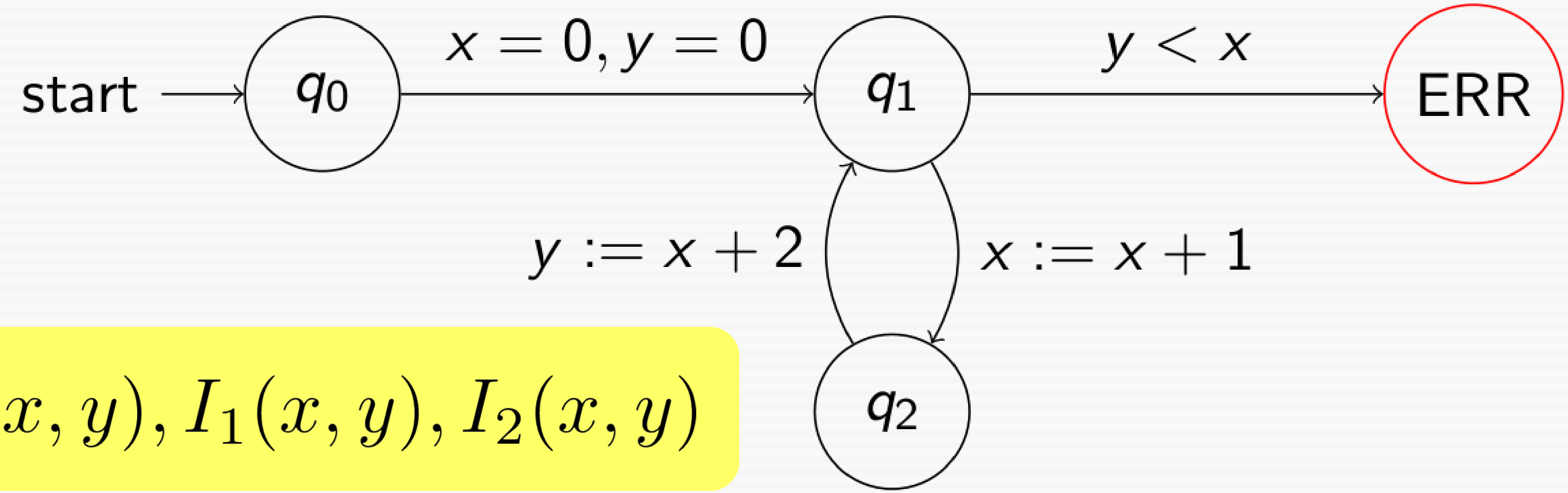
Ex 1: Floyd-style invariants



$$I_0(x, y), I_1(x, y), I_2(x, y)$$

- When the program is in q_0, $I_0(x, y)$ holds

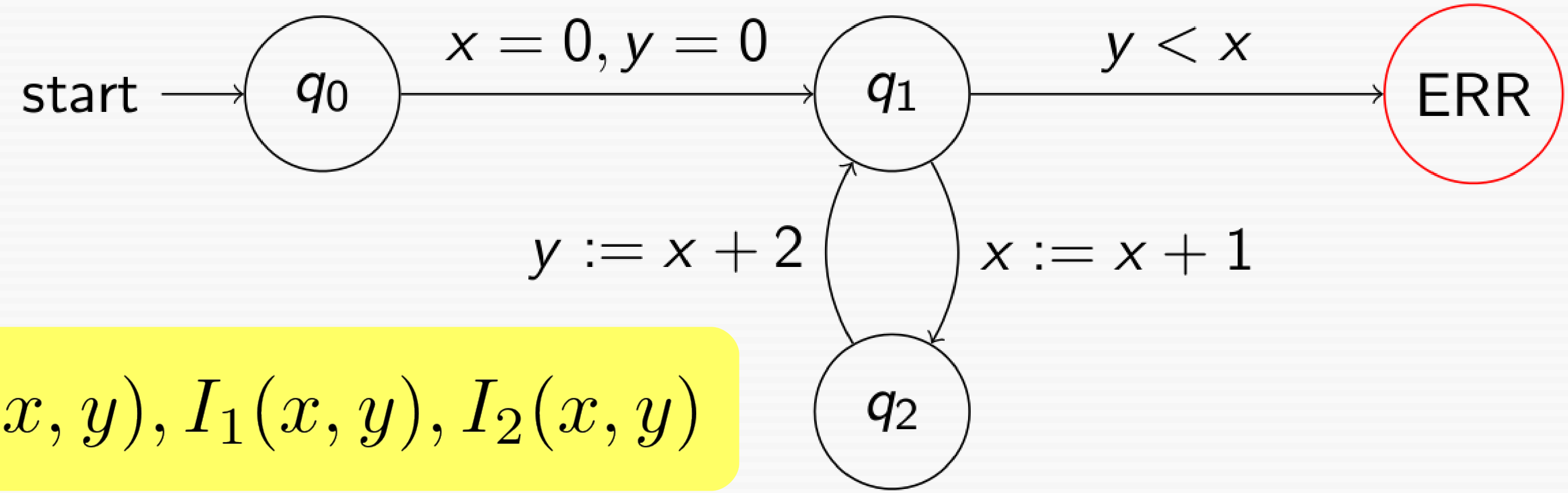
Ex 1: Floyd-style invariants



$$I_0(x, y), I_1(x, y), I_2(x, y)$$

- When the program is in q_0, $I_0(x, y)$ holds
- When the program is in q_0 and $I_0(x, y)$ holds, then after transition to q_1 the formula $I_1(x, y)$ holds

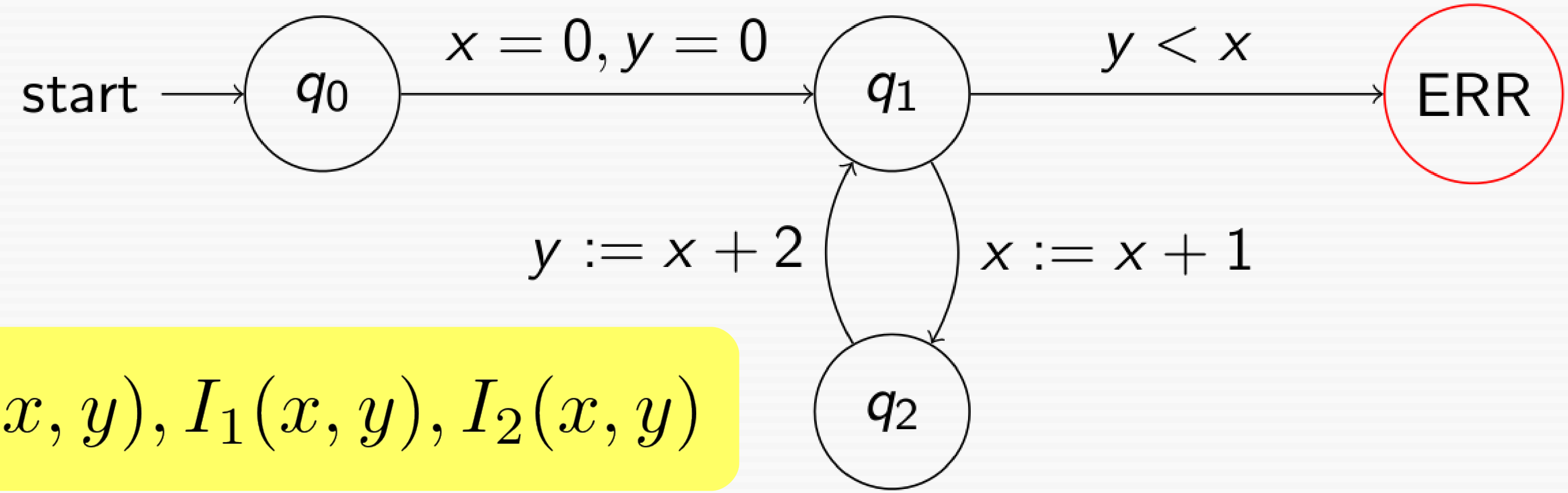
Ex 1: Floyd-style invariants



$I_0(x, y), I_1(x, y), I_2(x, y)$

- When the program is in q_0, $I_0(x, y)$ holds
- When the program is in q_0 and $I_0(x, y)$ holds, then after transition to q_1 the formula $I_1(x, y)$ holds
- etc.

Ex 1: Floyd-style invariants



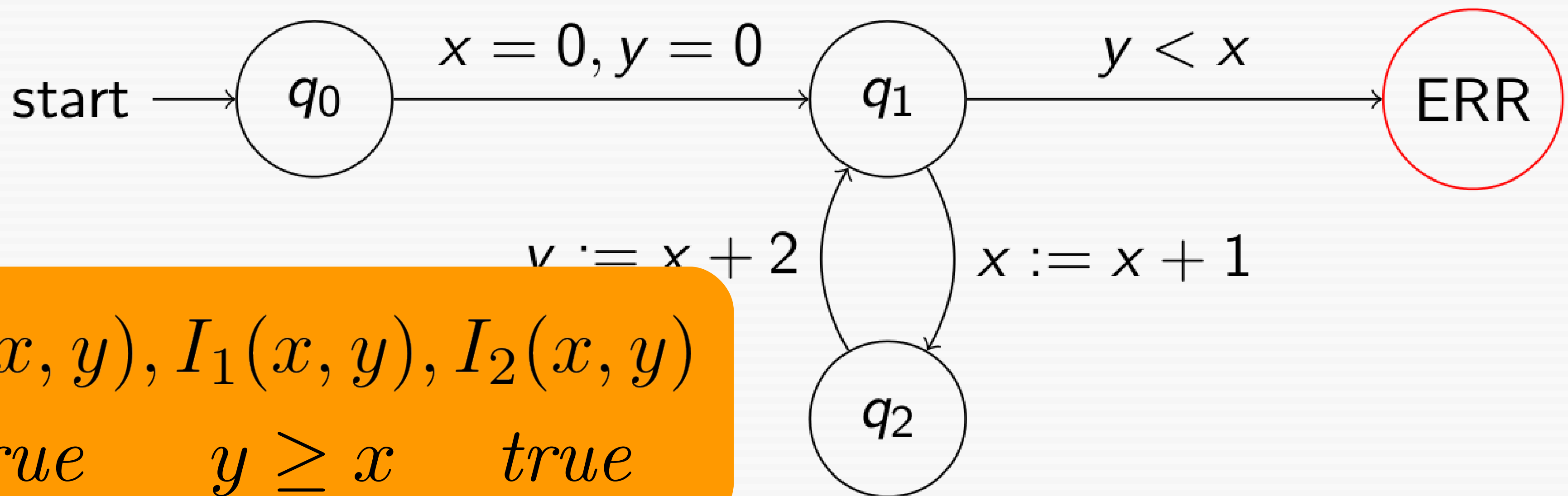
$$I_0(x, y), I_1(x, y), I_2(x, y)$$

- When th … $y)$ holds
- When th … $I_0(x, y)$ holds, th … $_1$ the formula
- etc.

Constraints:

$$\forall x, y.\ true \rightarrow I_0(x, y)$$
$$\forall x, y.\ I_0(x, y) \rightarrow I_1(0, 0)$$
$$\forall x, y.\ I_1(x, y) \rightarrow I_2(x + 1, y)$$
$$\forall x, y.\ I_2(x, y) \rightarrow I_1(x, x + 2)$$
$$\forall x, y.\ I_1(x, y) \wedge y < x \rightarrow false$$

Ex 1: Floyd-style invariants



$$I_0(x, y), I_1(x, y), I_2(x, y)$$

true $\quad y \geq x \quad$ true

- When th … $y)$ holds
- When th … $I_0(x, y)$ holds, th … $_1$ the formula …
- etc.

Constraints:

$$\forall x, y.\ \mathit{true} \rightarrow I_0(x, y)$$
$$\forall x, y.\ I_0(x, y) \rightarrow I_1(0, 0)$$
$$\forall x, y.\ I_1(x, y) \rightarrow I_2(x + 1, y)$$
$$\forall x, y.\ I_2(x, y) \rightarrow I_1(x, x + 2)$$
$$\forall x, y.\ I_1(x, y) \wedge y < x \rightarrow \mathit{false}$$

In Machine-Readable Format

SMT-LIB

```
(set-logic HORN)

(declare-fun I0 (Int Int) Bool)
(declare-fun I1 (Int Int) Bool)
(declare-fun I2 (Int Int) Bool)

(assert (forall ((x Int) (y Int)) (I0 x y)))
(assert (forall ((x Int) (y Int)) (=> (I0 x y) (I1 0 0))))
(assert (forall ((x Int) (y Int)) (=> (I1 x y) (I2 (+ x 1) y))))
(assert (forall ((x Int) (y Int)) (=> (I2 x y) (I1 x (+ x 2)))))
(assert (forall ((x Int) (y Int)) (=> (and (I1 x y) (< y x)) false)))

(check-sat)
(get-model)
```

In Machine-Readable Format

SMT-LIB

```
(set-logic HORN)

(declare-fun I0 (Int Int) Bool)
(declare-fun I1 (Int Int) Bool)
(declare-fun I2 (Int Int) Bool)

(assert (forall ((x Int) (y Int)) (I0 x y)))
(assert (forall ((x Int) (y Int)) (=> (I0 x y) (I1 0 0))))
(assert (forall ((x Int) (y Int)) (=> (I1 x y) (I2 (+ x 1) y))))
(assert (forall ((x Int) (y Int)) (=> (I2 x y) (I1 x (+ x 2)))))
(assert (forall ((x Int) (y Int)) (=> (and (I1 x y) (< y x)) false)))

(check-sat)
(get-model)
```

Prolog

```
i0(X, Y)   :- 1=1.
i1(X', Y') :- i0(X, Y), X'=0, Y'=0.
i2(X', Y)  :- i1(X, Y), X'=X+1.
i1(X, Y')  :- i2(X, Y), Y'=X+2.
false      :- i1(X, Y), Y < X.
```

eldarica

Are the following Horn clauses satisfiable? / Is the program safe?

```
(set-logic HORN)

(declare-fun I0 (Int Int) Bool)
(declare-fun I1 (Int Int) Bool)
(declare-fun I2 (Int Int) Bool)

(assert (forall ((x Int) (y Int)) (I0 x y)))
(assert (forall ((x Int) (y Int)) (=> (I0 x y) (I1 0 0))))
(assert (forall ((x Int) (y Int)) (=> (I1 x y) (I2 (+ x 1) y))))
(assert (forall ((x Int) (y Int)) (=> (I2 x y) (I1 x (+ x 2)))))
(assert (forall ((x Int) (y Int)) (=> (and (I1 x y) (< y x)) false)))

(check-sat)
(get-model)
```


▶ tutorial home permalink

'▶' shortcut: Alt+B

```
sat
(define-fun I0 ((A Int) (B Int)) Bool true)
(define-fun I1 ((A Int) (B Int)) Bool (and (>= B A) (>= A 0)))
(define-fun I2 ((A Int) (B Int)) Bool (and (>= (- B A) (- 1)) (>= A 1)))
```

More formally ...

Definition
Suppose
- $\mathcal{L}$ is some constraint language;
- $\mathcal{R}$ is a set of relation symbols;

Then a *Constrained Horn Clause (CHC)* is a formula

$$\forall \bar{x}.\, C \wedge B_1, \ldots, B_n \rightarrow H$$

in which
- C is a constraint in $\mathcal{L}$ (no symbols from $\mathcal{R}$);
- each B_i is a literal of the form $r(t_1, \ldots, t_m)$;
- H is either $false$, or of the same form as the B_i.

More formally ...

Definition

Suppose

- $\mathcal{L}$ is some constraint language;
- $\mathcal{R}$ is a set of relation symbols;

Then a *Constrained Horn Clause (CHC)* is a formula

Constraint

$$\forall \bar{x}.\ \overbrace{C}^{\text{Constraint}} \wedge \underbrace{B_1, \ldots, B_n}_{\text{Body}} \rightarrow \underbrace{H}_{\text{Head}}$$

Body **Head**

in which

- C is a constraint in $\mathcal{L}$ (no symbols from $\mathcal{R}$);
- each B_i is a literal of the form $r(t_1, \ldots, t_m)$;
- H is either false, or of the same form as the B_i.

More formally

Combination of theories; e.g., integers, rationals, arrays, etc.

Definition

Suppose

- $\mathcal{L}$ is some constraint language;
- $\mathcal{R}$ is a set of relation symbols;

Constraint

Then a *Constrained Horn Clause (CHC)* is a formula

$$\forall \bar{x}.\ C \wedge B_1, \ldots, B_n \rightarrow H$$

Body

Head

in which

- C is a constraint in $\mathcal{L}$ (no symbols from $\mathcal{R}$);
- each B_i is a literal of the form $r(t_1, \ldots, t_m)$;
- H is either false, or of the same form as the B_i.

More formally

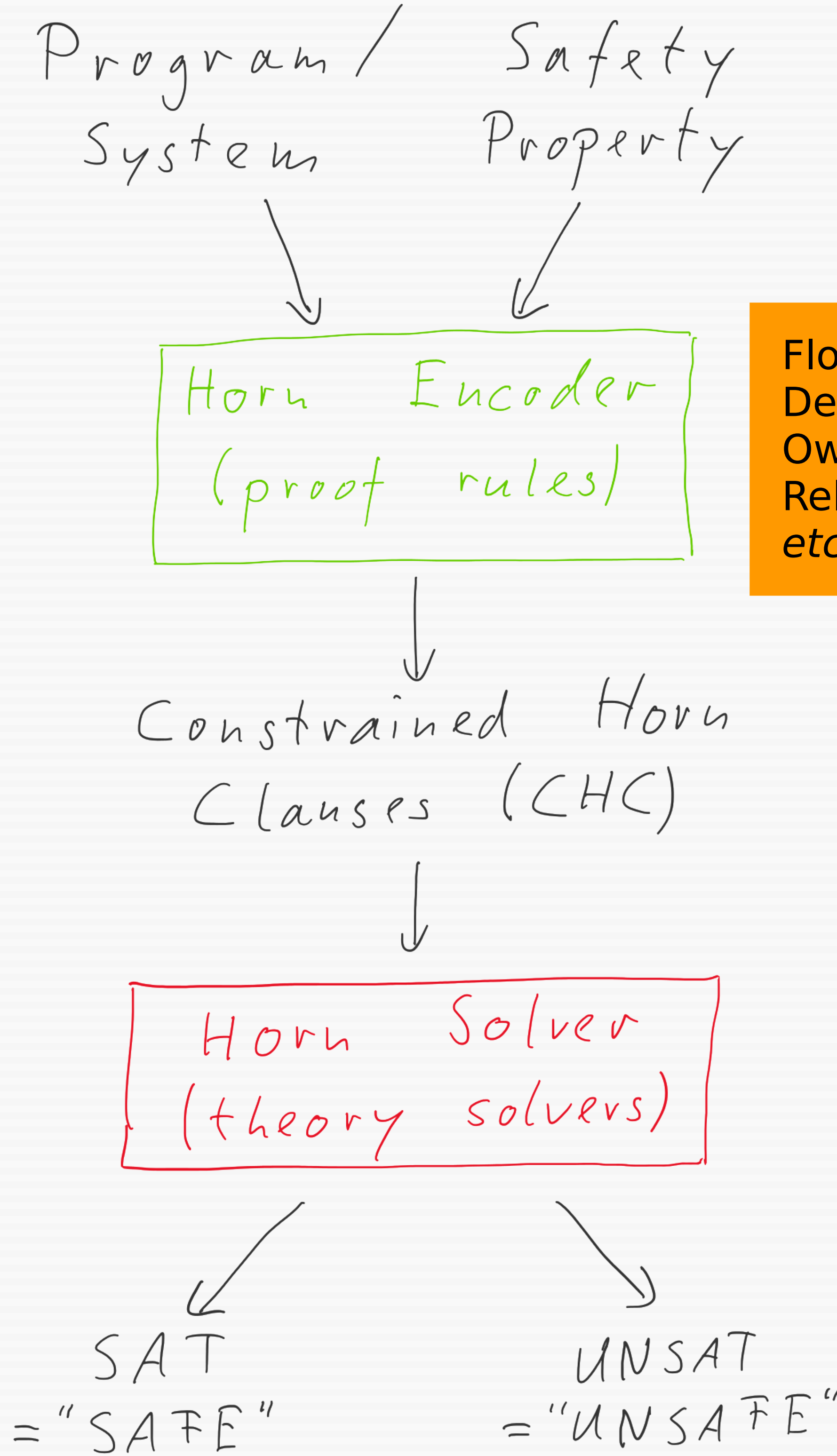
Combination of theories; e.g., integers, rationals, arrays, etc.

Definition
Suppose
- $\mathcal{L}$ is some constraint language;
- $\mathcal{R}$ is a set of relation symbols;

Then a *Constrained Horn Clause (CHC)* is a formula

Definition
A set $\mathcal{C}$ of Horn clauses is *satisfiable* if it is satisfiable in the first-order/model-theoretic sense.

[This means: for some interpretation of $\mathcal{R}$ symbols all clauses become valid.]



Floyd-Hoare
Design by contract
Owicki-Gries
Rely Guarantee
etc.

From Proof Rules to CHC

$$\frac{P \Rightarrow R[x/t]}{\{P\}\ x = t\ \{R\}}\ \textsc{Assign}'$$

$$\frac{\{P\}\ S\ \{Q\} \qquad \{Q\}\ T\ \{R\}}{\{P\}\ S; T\ \{R\}}\ \textsc{Comp}$$

$$\frac{\{P \wedge B\}\ S\ \{R\} \qquad \{P \wedge \neg B\}\ T\ \{R\}}{\{P\}\ \mathit{if}\ B\ \mathit{then}\ S\ \mathit{else}\ T\ \{R\}}\ \textsc{Cond}$$

$$\frac{P \Rightarrow I \qquad \{I \wedge B\}\ S\ \{I\} \qquad I \wedge \neg B \Rightarrow R}{\{P\}\ \mathit{while}\ B\ \mathit{do}\ S\ \{R\}}\ \textsc{Loop}'$$

From Proof Rules to CHC

$$\frac{P \Rightarrow R[x/t]}{\{P\}\ x = t\ \{R\}}\ \textsc{Assign}'$$

$$\frac{\{P\}\ S\ \{Q\} \qquad \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}\ \textsc{Comp}$$

$$\frac{\{P \wedge B\}\ S\ \{R\} \qquad \{P \wedge \neg B\}\ T\ \{R\}}{\{P\}\ \mathit{if}\ \ B\ \ \mathit{then}\ \ S\ \ \mathit{else}\ \ T\ \{R\}}\ \textsc{Cond}$$

$$\frac{P \Rightarrow I \qquad \{I \wedge B\}\ S\ \{I\} \qquad I \wedge \neg B \Rightarrow R}{\{P\}\ \mathit{while}\ \ B\ \ \mathit{do}\ \ S\ \{R\}}\ \textsc{Loop}'$$

From Proof Rules to CHC

$$\frac{P \Rightarrow R[x/t]}{\{P\}\ x = t\ \{R\}}\ \textsc{Assign}'$$

$$\frac{\{P\}\ S\ \{Q\} \qquad \{Q\}\ T\ \{R\}}{\{P\}\ S; T\ \{R\}}\ \textsc{Comp}$$

$$\frac{\{P \wedge B\}\ S\ \{R\} \qquad \{P \wedge \neg B\}\ T\ \{R\}}{\{P\}\ \mathit{if}\ B\ \mathit{then}\ S\ \mathit{else}\ T\ \{R\}}\ \textsc{Cond}$$

$$\frac{P \Rightarrow I \qquad \{I \wedge B\}\ S\ \{I\} \qquad I \wedge \neg B \Rightarrow R}{\{P\}\ \mathit{while}\ B\ \mathit{do}\ S\ \{R\}}\ \textsc{Loop}'$$

Example 2

$$\frac{I(n,x) \wedge x < n \Rightarrow I(n, x+1)}{\{I(n,x) \wedge x < n\}\ x = x + 1\ \{I(n,x)\}}$$

$$\frac{\dfrac{n \geq 0 \Rightarrow P(n,0)}{\{n \geq 0\}\ x = 0\ \{P(n,x)\}} \qquad \dfrac{P(n,x) \Rightarrow I(n,x) \qquad \{I(n,x) \wedge x < n\}\ x = x+1\ \{I(n,x)\} \qquad I(n,x) \wedge x \not< n \Rightarrow x = n}{\{P(n,x)\}\ \mathit{while}\ x < n\ \mathit{do}\ x = x + 1\ \{x = n\}}}{\{n \geq 0\}\ x = 0; \mathit{while}\ x < n\ \mathit{do}\ x = x + 1\ \{x = n\}}$$

From Proof Rules to CHC

Constraints/CHCs

$$n \geq 0 \Rightarrow P(n, 0)$$

$$P(n, x) \Rightarrow I(n, x)$$

$$I(n, x) \wedge x < n \Rightarrow I(n, x + 1)$$

$$I(n, x) \wedge x \not< n \Rightarrow x = n$$

$$\frac{\{P\}\ S\ \{Q\} \qquad \{Q\}\ T\ \{R\}}{\{P\}\ S; T\ \{R\}}\ \textsc{Comp}$$

$$\frac{P \Rightarrow I \qquad \{I \wedge B\}\ S\ \{I\} \qquad I \wedge \neg B \Rightarrow R}{\{P\}\ \mathit{while}\ B\ \mathit{do}\ S\ \{R\}}\ \textsc{Loop}'$$

Example 2

$$\frac{I(n, x) \wedge x < n \Rightarrow I(n, x + 1)}{\{I(n, x) \wedge x < n\}\ x = x + 1\ \{I(n, x)\}}$$

$$\frac{\dfrac{n \geq 0 \Rightarrow P(n, 0)}{\{n \geq 0\}\ x = 0\ \{P(n, x)\}} \qquad \dfrac{P(n, x) \Rightarrow I(n, x) \qquad \vdots \qquad I(n, x) \wedge x \not< n \Rightarrow x = n}{\{P(n, x)\}\ \mathit{while}\ x < n\ \mathit{do}\ x = x + 1\ \{x = n\}}}{\{n \geq 0\}\ x = 0; \mathit{while}\ x < n\ \mathit{do}\ x = x + 1\ \{x = n\}}$$

From Proof Rules to CHC

Constraints/CHCs

$$n \geq 0 \Rightarrow P(n, 0)$$
$$P(n, x) \Rightarrow I(n, x)$$
$$I(n, x) \wedge x < n \Rightarrow I(n, x + 1)$$
$$I(n, x) \wedge x \not< n \Rightarrow x = n$$

Solution/Model

$$P(n, x) \;\equiv\; n \geq 0 \wedge x = 0$$
$$I(n, x) \;\equiv\; n \geq x \wedge x \geq 0$$

Example 2

$$\dfrac{\dfrac{n \geq 0 \Rightarrow P(n,0)}{\{n \geq 0\}\ x = 0\ \{P(n,x)\}} \qquad \dfrac{P(n,x) \Rightarrow I(n,x) \quad \dfrac{I(n,x) \wedge x < n \Rightarrow I(n,x+1)}{\{I(n,x) \wedge x < n\}\ x = x + 1\ \{I(n,x)\}} \quad I(n,x) \wedge x \not< n \Rightarrow x = n}{\{P(n,x)\}\ \mathit{while}\ x < n\ \mathit{do}\ x = x + 1\ \{x = n\}}}{\{n \geq 0\}\ x = 0;\ \mathit{while}\ x < n\ \mathit{do}\ x = x + 1\ \{x = n\}}$$

From Proof Rules to CHC

Constraints/CHCs

$$n \geq 0 \Rightarrow P(n,0)$$
$$P(n,x) \Rightarrow I(n,x)$$
$$I(n,x) \land x < n \Rightarrow I(n,x+1)$$
$$I(n,x) \land x \not< n \Rightarrow x = n$$

Solution/Model

$$P(n,x) \equiv n \geq 0 \land x = 0$$
$$I(n,x) \equiv n \geq x \land x \geq 0$$

Substitute to obtain a closed proof ...

Example 2

$$\dfrac{\dfrac{n \geq 0 \Rightarrow P(n,0)}{\{n \geq 0\}\ x = 0\ \{P(n,x)\}} \quad \dfrac{P(n,x) \Rightarrow I(n,x) \quad \dfrac{I(n,x) \land x < n \Rightarrow I(n,x+1)}{\{I(n,x) \land x < n\}\ x = x+1\ \{I(n,x)\}} \quad I(n,x) \land x \not< n \Rightarrow x = n}{\{P(n,x)\}\ while\ x < n\ do\ x = x+1\ \{x = n\}}}{\{n \geq 0\}\ x = 0;\ while\ x < n\ do\ x = x+1\ \{x = n\}}$$

Function calls

$$\frac{P \Rightarrow R[x/t]}{\{P\}\ x = t\ \{R\}}\ \text{Assign}'$$

$$\frac{\{P\}\ S\ \{Q\} \qquad \{Q\}\ T\ \{R\}}{\{P\}\ S; T\ \{R\}}\ \text{Comp}$$

$$\frac{\{P \wedge B\}\ S\ \{R\} \qquad \{P \wedge \neg B\}\ T\ \{R\}}{\{P\}\ \mathit{if}\ B\ \mathit{then}\ S\ \mathit{else}\ T\ \{R\}}\ \text{Cond}$$

$$\frac{P \Rightarrow I \qquad \{I \wedge B\}\ S\ \{I\} \qquad I \wedge \neg B \Rightarrow R}{\{P\}\ \mathit{while}\ B\ \mathit{do}\ S\ \{R\}}\ \text{Loop}'$$

$$\frac{P \Rightarrow Pre_f[\bar{a}_f/\bar{t}] \qquad P \wedge Post_f[\bar{a}_f/\bar{t}] \Rightarrow R[x/r_f]}{\{P\}\ x = f(\bar{t})\ \{R\}}\ \text{Call}$$

Function calls

$$\frac{P \Rightarrow R[x/t]}{\{P\}\ x = t\ \{R\}}\ \textsc{Assign}'$$

$$\frac{\{P\}\ S\ \{Q\} \qquad \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}\ \textsc{Comp}$$

$$\frac{\{P \wedge B\}\ S\ \{R\} \qquad \{P \wedge \neg B\}\ T\ \{R\}}{\{P\}\ \mathit{if}\ B\ \mathit{then}\ S\ \mathit{else}\ T\ \{R\}}\ \textsc{Cond}$$

$$\frac{P \Rightarrow I \qquad \{I \wedge B\}\ S\ \{I\} \qquad I \wedge \neg B \Rightarrow R}{\{P\}\ \mathit{while}\ B\ \mathit{do}\ S\ \{R\}}\ \textsc{Loop}'$$

$$\frac{P \Rightarrow \mathit{Pre}_f[\bar{a}_f/\bar{t}] \qquad P \wedge \mathit{Post}_f[\bar{a}_f/\bar{t}] \Rightarrow R[x/r_f]}{\{P\}\ x = f(\bar{t})\ \{R\}}\ \textsc{Call}$$

Function calls

$$\frac{P \Rightarrow R[x/t]}{\{P\}\ x = t\ \{R\}}\ \textsc{Assign}'$$

$$\frac{\{P\}\ S\ \{Q\} \qquad \{Q\}\ T\ \{R\}}{\{P\}\ S; T\ \{R\}}\ \textsc{Comp}$$

$$\frac{\{P \land B\}\ S\ \{R\} \qquad \{P \land \neg B\}\ T\ \{R\}}{\{P\}\ \mathit{if}\ \ B\ \ \mathit{then}\ \ S\ \ \mathit{else}\ \ T\ \{R\}}\ \textsc{Cond}$$

$$\frac{P \Rightarrow I \qquad \{I \land B\}\ S\ \{I\} \qquad I \land \neg B \Rightarrow R}{\{P\}\ \mathit{while}\ \ B\ \ \mathit{do}\ \ S\ \{R\}}\ \textsc{Loop}'$$

$$\frac{P \Rightarrow \mathit{Pre}_f[\bar{a}_f/\bar{t}] \qquad P \land \mathit{Post}_f[\bar{a}_f/\bar{t}] \Rightarrow R[x/r_f]}{\{P\}\ x = f(\bar{t})\ \{R\}}\ \textsc{Call}$$

+ proof obligations
ensuring correctness
of contract

Example 3: Functions

$$f(x) = \begin{cases} x - 10, & \text{if } x > 100 \\ f(f(x+11)), & \text{if } x \leq 100 \end{cases}$$

Verify $x \leq 100 \rightarrow f(x) = 91$

Example 3: Functions

$$f(x) = \begin{cases} x - 10, & \text{if } x > 100 \\ f(f(x+11)), & \text{if } x \leq 100 \end{cases}$$

Verify $x \leq 100 \rightarrow f(x) = 91$

```
int f(int x) {
  if (x > 100) {
    int t0 = x - 10;
    return t0;
  } else {
    int t0 = x + 11;
    int t1 = f(t0);
    int t2 = f(t1);
    return t2;
  }
}
```

Example 3: Functions

$$f(x) = \begin{cases} x - 10, & \text{if } x > 100 \\ f(f(x+11)), & \text{if } x \leq 100 \end{cases}$$

Verify $x \leq 100 \rightarrow f(x) = 91$

```
int f(int x) {
  if (x > 100) {
    int t0 = x - 10;
    return
  } else {
    int t0 = x
    int t1 = f(t
    int t2 = f(t1
    return t
  }
}
```

Assume that f has:

Pre-condition `true`

Post-condition `post_f(x, result)`

Encoding as CHC

```
i0(X0, X)      :- X0=X.                              % int f(int x) {
i1(X0, X)      :- i0(X0, X), X > 100.                %   if (x > 100) {
i2(X0, T0)     :- i1(X0, X), T0=X-10.                %     int t0 = x - 10;
post_f(X0, T0) :- i2(X0, T0).                        %     return t0;
i3(X0, X)      :- i0(X0, X), X =< 100.               %   } else {
i4(X0, T0)     :- i3(X0, X), T0=X+11.                %     int t0 = x + 11;
i5(X0, T1)     :- i4(X0, T0), post_f(T0, T1).        %     int t1 = f(t0);
i6(X0, T2)     :- i5(X0, T1), post_f(T1, T2).        %     int t2 = f(t1);
post_f(X0, T2) :- i6(X0, T2).                        %     return t2;
                                                     %   }
                                                     % }

false :- post_f(X, R), X =< 100, \+(R = 91).         % Assertion
```

State invariants record function input and current value of variables

ncoding as CHC

```
i0(X0, X)       :- X0=X.                            % int f(int x) {
i1(X0, X)       :- i0(X0, X), X > 100.              %   if (x > 100) {
i2(X0, T0)      :- i1(X0, X), T0=X-10.              %     int t0 = x - 10;
post_f(X0, T0) :- i2(X0, T0).                       %     return t0;
i3(X0, X)       :- i0(X0, X), X =< 100.             %   } else {
i4(X0, T0)      :- i3(X0, X), T0=X+11.              %     int t0 = x + 11;
i5(X0, T1)      :- i4(X0, T0), post_f(T0, T1).      %     int t1 = f(t0);
i6(X0, T2)      :- i5(X0, T1), post_f(T1, T2).      %     int t2 = f(t1);
post_f(X0, T2) :- i6(X0, T2).                       %     return t2;
                                                    %   }
                                                    % }

false :- post_f(X, R), X =< 100, \+(R = 91).        % Assertion
```

s CHC

State invariants record function input and current value of variables

Upon return, assert that post-condition holds

```
i0(X0, X)      :- X0=X.                            % int f(int x) {
i1(X0, X)      :- i0(X0, X), X > 100.              %   if (x > 100) {
i2(X0, T0)     :- i1(X0, X), T0=X-10.              %     int t0 = x - 10;
post_f(X0, T0) :- i2(X0, T0).                      %     return t0;
i3(X0, X)      :- i0(X0, X), X =< 100.             %   } else {
i4(X0, T0)     :- i3(X0, X), T0=X+11.              %     int t0 = x + 11;
i5(X0, T1)     :- i4(X0, T0), post_f(T0, T1).      %     int t1 = f(t0);
i6(X0, T2)     :- i5(X0, T1), post_f(T1, T2).      %     int t2 = f(t1);
post_f(X0, T2) :- i6(X0, T2).                      %     return t2;
                                                   %   }
                                                   % }

false :- post_f(X, R), X =< 100, \+(R = 91).       % Assertion
```

State invariants record function input and current value of variables

Upon return, assert that post-condition holds

Function calls can assume the post-condition → non-linear clauses

```
i0(X0, X)      :- X0=X.                               % int f(int x) {
i1(X0, X)      :- i0(X0, X), X > 100.                 %   if (x > 100) {
i2(X0, T0)     :- i1(X0, X), T0=X-10.                 %     int t0 = x - 10;
post_f(X0, T0) :- i2(X0, T0).                         %     return t0;
i3(X0, X)      :- i0(X0, X), X =< 100.                %   } else {
i4(X0, T0)     :- i3(X0, X), T0=X+11.                 %     int t0 = x + 11;
i5(X0, T1)     :- i4(X0, T0), post_f(T0, T1).         %     int t1 = f(t0);
i6(X0, T2)     :- i5(X0, T1), post_f(T1, T2).         %     int t2 = f(t1);
post_f(X0, T2) :- i6(X0, T2).                         %     return t2;
                                                      %   }
                                                      % }

false :- post_f(X, R), X =< 100, \+(R = 91).          % Assertion
```

State invariants record function input and current value of variables

Upon return, assert that post-condition holds

Function calls can assume the post-condition → non-linear clauses

```
i0(X0, X)      :- X0=X.                               % int f(int x) {
i1(X0, X)      :- i0(X0, X), X > 100.                 %   if (x > 100) {
i2(X0, T0)     :- i1(X0, X), T0=X-10.                 %     int t0 = x - 10;
post_f(X0, T0) :- i2(X0, T0).                         %     return t0;
i3(X0, X)      :- i0(X0, X), X =< 100.                %   } else {
i4(X0, T0)     :- i3(X0, X), T0=X+11.                 %     int t0 = x + 11;
i5(X0, T1)     :- i4(X0, T0), post_f(T0, T1).         %     int t1 = f(t0);
i6(X0, T2)     :- i5(X0, T1), post_f(T1, T2).         %     int t2 = f(t1);
post_f(X0, T2) :- i6(X0, T2).                         %     return t2;
                                                      %   }
                                                      % }

false :- post_f(X, R), X =< 100, \+(R = 91).          % Assertion
```

Property expressed in terms of post-condition

State inv
record func
and curren
varia
s

i0(X0, X)
i1(X0, X)
i2(X0, T0
post_f(X0
i3(X0, X)
i4(X0, T0
i5(X0, T1
i6(X0, T2
post_f(X0

false :- p

) {
00) {
= x - 10;
t0;

= x + 11;
= f(t0);
= f(t1);
t2;

expressed
ms of
ndition

eldarica

Are the following Horn clauses satisfiable? / Is the program safe?

```
i0(X0, X)       :- X0=X.
i1(X0, X)       :- i0(X0, X), X > 100.
i2(X0, T0)      :- i1(X0, X), T0=X-10.
post_f(X0, T0) :- i2(X0, T0).
i3(X0, X)       :- i0(X0, X), X =< 100.
i4(X0, T0)      :- i3(X0, X), T0=X+11.
i5(X0, T1)      :- i4(X0, T0), post_f(T0, T1).
i6(X0, T2)      :- i5(X0, T1), post_f(T1, T2).
post_f(X0, T2) :- i6(X0, T2).

false :- post_f(X, R), X =< 100, \+(R = 91).
```

DISCLAIMER: Eldarica is a 3rd party tool offered by Uppsala University. By clicking '►', you
Eldarica to be analyzed. Please refer to the terms of use and privacy policy of Eldarica.

► | tutorial | home | permalink

'►' shortcut: Alt+B

```
SOLVABLE
i0(A,B) :- (B = A).
i1(A,B) :- ((B = A), (A >= 101)).
i2(A,B) :- (((B - A) = -10), (A >= 101)).
i3(A,B) :- ((B = A), (A =< 100)).
i4(A,B) :- (((B - A) = 11), (A =< 100)).
i5(A,B) :- ((A =< 100), ((B = 91); (((A - B) >= -1), (B >= 92)))).
i6(A,B) :- ((B = 91), (A =< 100)).
post_f(A,B) :- ((B = 91); (((A - B) >= 10), (B >= 92))).
```

Fragments of CHC

Fragments of CHC

- **Linear**:
  $\leq$1 literals per clause body
- **Non-linear/general:**
  some clause with $\geq$1 body literals
  → function calls, concurrency, etc.

Fragments of CHC

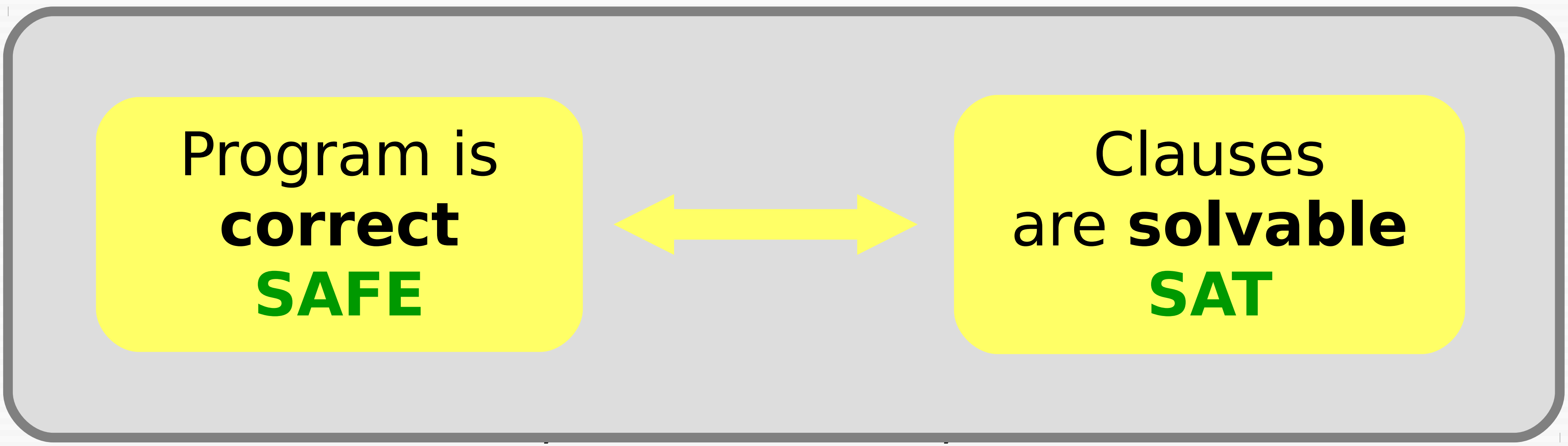
- **Linear**:
  ≤1 literals per clause body
- **Non-linear/general:**
  some clause with ≥1 body literals
  → function calls, concurrency, etc.
- **“Transition systems” (TS):**
  exactly three clauses (init, trans, err)
  (linear clauses can be reduced to this)

Summary so far

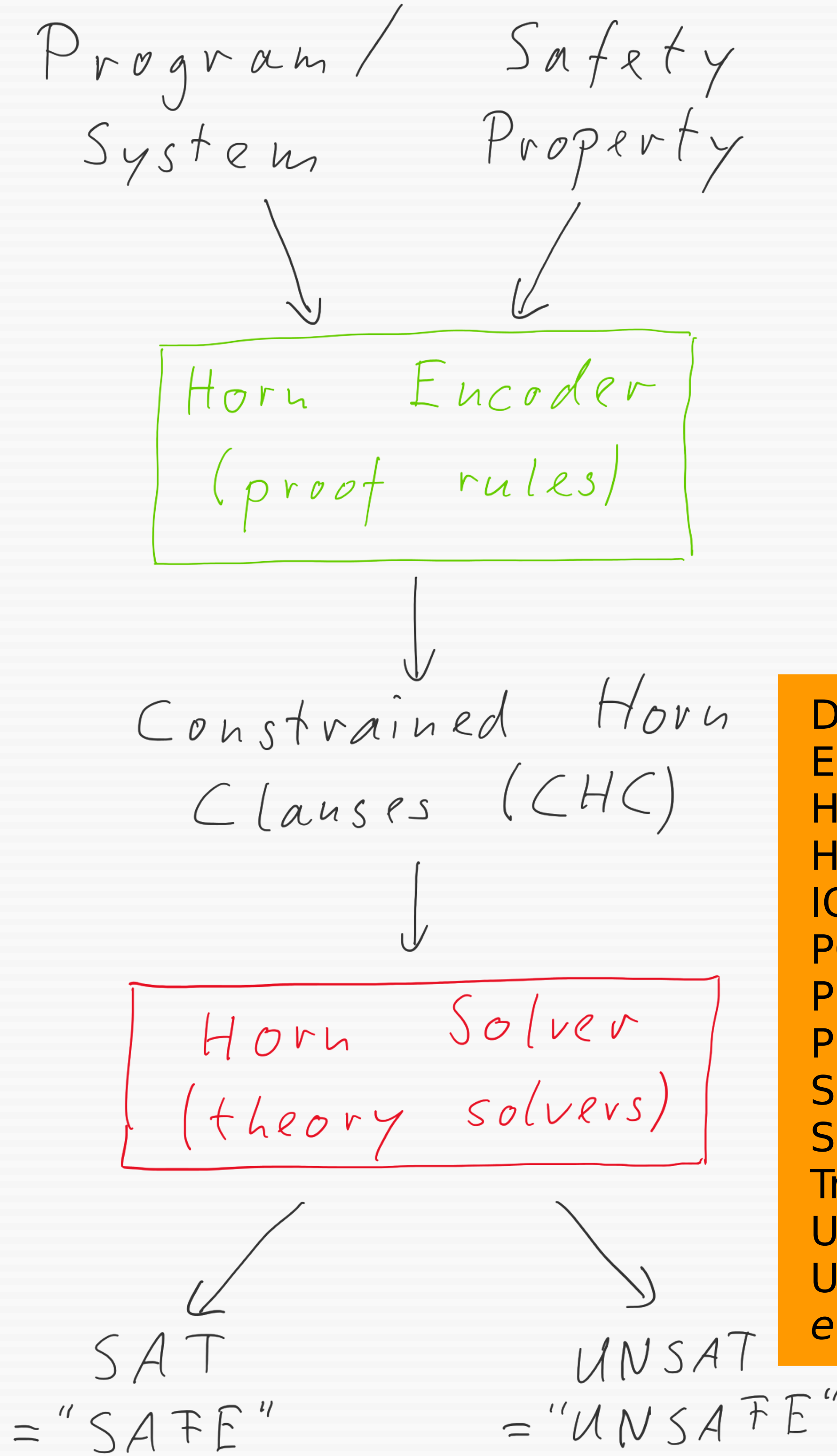
- Relation symbols in CHCs represent
  Program annotations
- for instance
  state **invariants**
  pre-/post-conditions
  class/process invariants
- CHCs encode **preservation**:
  initiation, consecution, etc.
- CHCs also encode **safety properties**:
  invariants exclude **error states**

Summary so far

- Relation symbols in CHCs represent
  Program annotations



- CHCs also encode **safety properties**:
  invariants exclude **error states**



Duality
Eldarica(-abs)
Hoice
HSF
IC3IA
PCSat
PECOS
ProphIC3
Sally
Spacer
TransfHORNer
Ultimate TreeAutomizer
Ultimate Unihorn
etc.

Algorithms in CHC

- CEGAR, predicate abstraction
- IC3, Spacer
- Syntax-guided synthesis (SyGuS)
- Decision trees, data-driven methods
- Transformation, unfold/fold, etc.
- Abstract interpretation

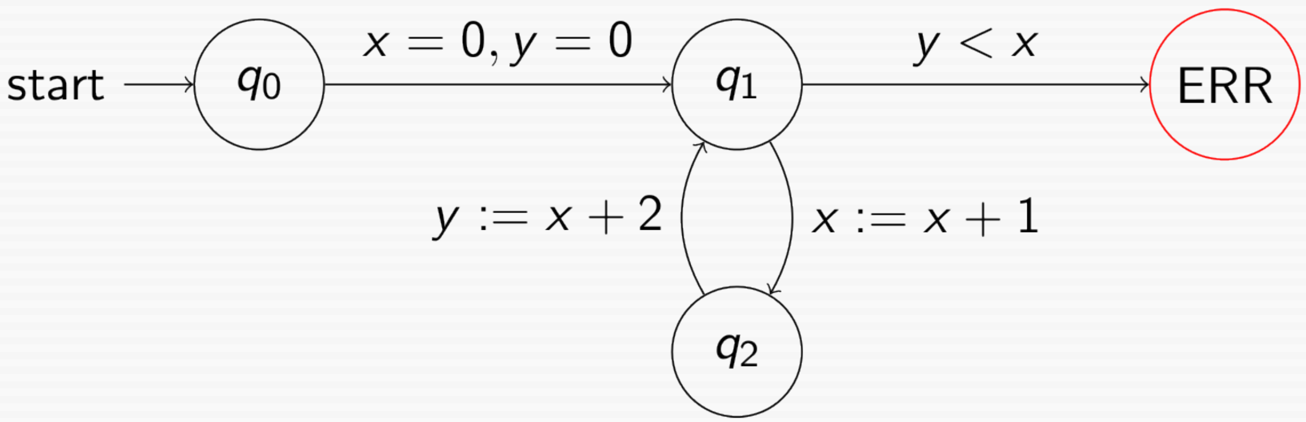
- *to be continued ...*

Algorithms in CHC

- CEGAR, predicate abstraction
- IC3, Spacer
- Syntax-guided synthesis (SyGuS)
- Decision trees, data-driven methods
- Transformation, unfold/fold, etc.
- Abstract interpretation

- *to be continued ...*

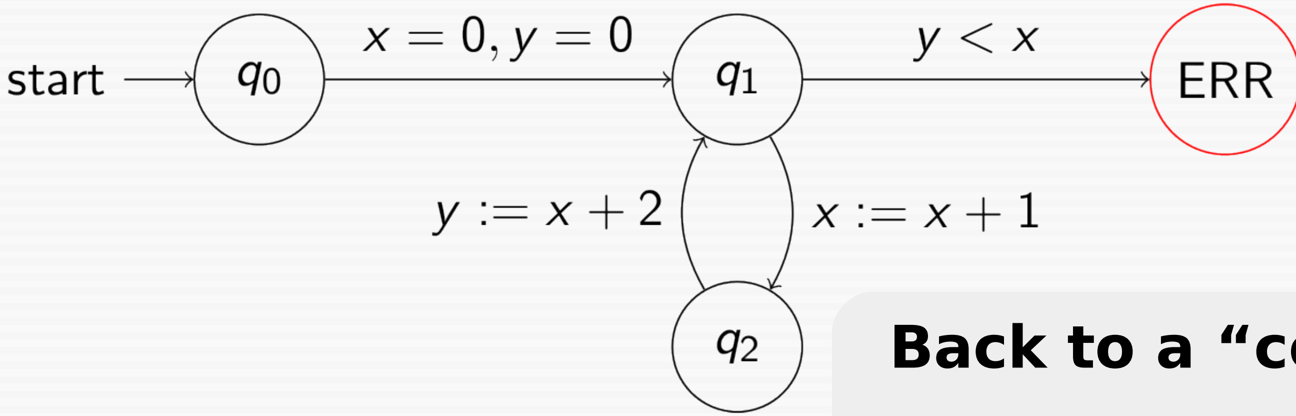
Linear CHC



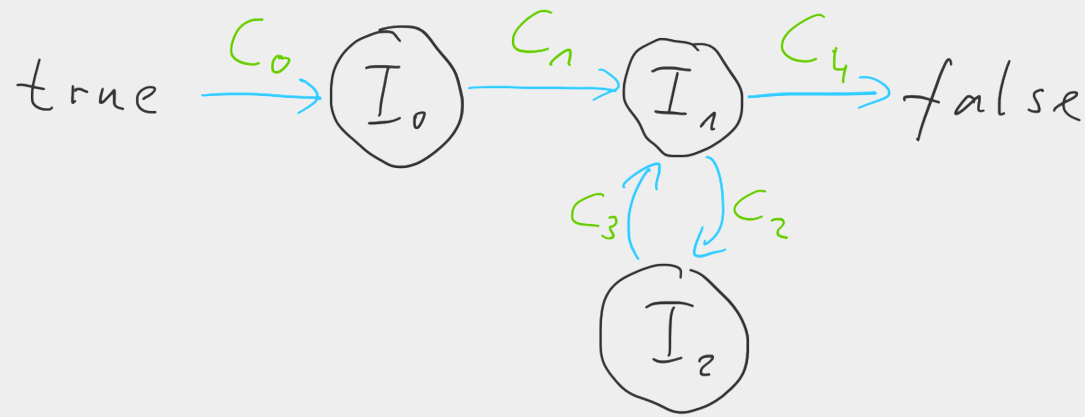
Constraints

$$
\begin{aligned}
C_0 &: \quad \forall x, y.\ true \rightarrow I_0(x, y) \\
C_1 &: \quad \forall x, y.\ I_0(x, y) \rightarrow I_1(0, 0) \\
C_2 &: \quad \forall x, y.\ I_1(x, y) \rightarrow I_2(x + 1, y) \\
C_3 &: \quad \forall x, y.\ I_2(x, y) \rightarrow I_1(x, x + 2) \\
C_4 &: \quad \forall x, y.\ I_1(x, y) \wedge y < x \rightarrow false
\end{aligned}
$$

Linear CHC



Back to a “control-flow graph”:



Constraint

$C_0 : \quad \forall x, y.\ \mathit{true} \rightarrow I_0(x$

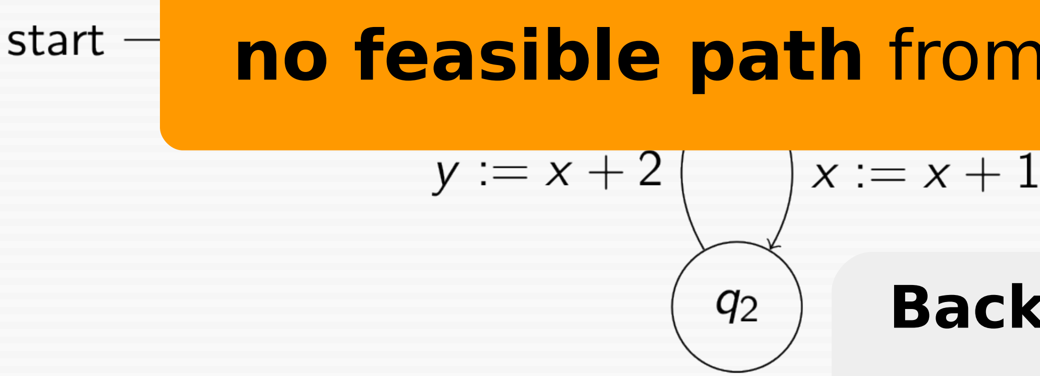
$C_1 : \quad \forall x, y.\ I_0(x, y) \rightarrow I$

$C_2 : \quad \forall x, y.\ I_1(x, y) \rightarrow I_2(x + 1, y)$

$C_3 : \quad \forall x, y.\ I_2(x, y) \rightarrow I_1(x, x + 2)$

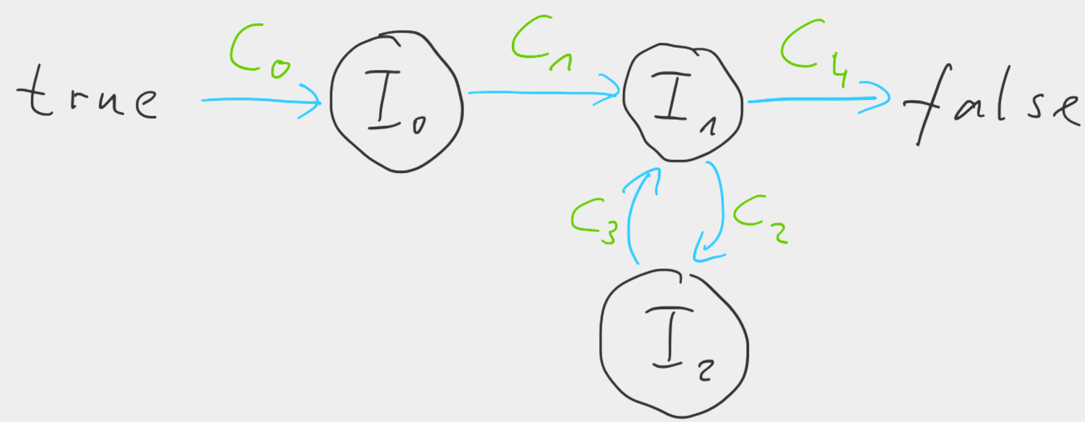
$C_4 : \quad \forall x, y.\ I_1(x, y) \wedge y < x \rightarrow \mathit{false}$

Linear CHC



Horn clauses are sat
iff
no feasible path from `true` to `false` exists

Back to a "control-flow graph":



Constraint

$C_0 : \quad \forall x, y.\ \mathit{true} \rightarrow I_0($

$C_1 : \quad \forall x, y.\ I_0(x, y) \rightarrow$

$C_2 : \quad \forall x, y.\ I_1(x, y) \rightarrow I_2(x + 1, y)$

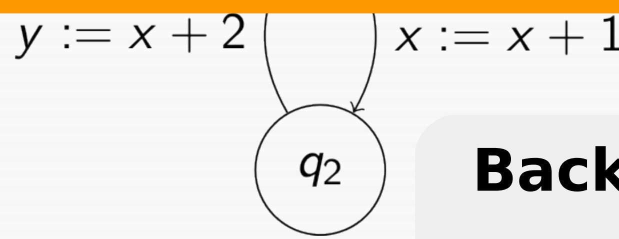
$C_3 : \quad \forall x, y.\ I_2(x, y) \rightarrow I_1(x, x + 2)$

$C_4 : \quad \forall x, y.\ I_1(x, y) \wedge y < x \rightarrow \mathit{false}$

Linear CHC

"Duality"

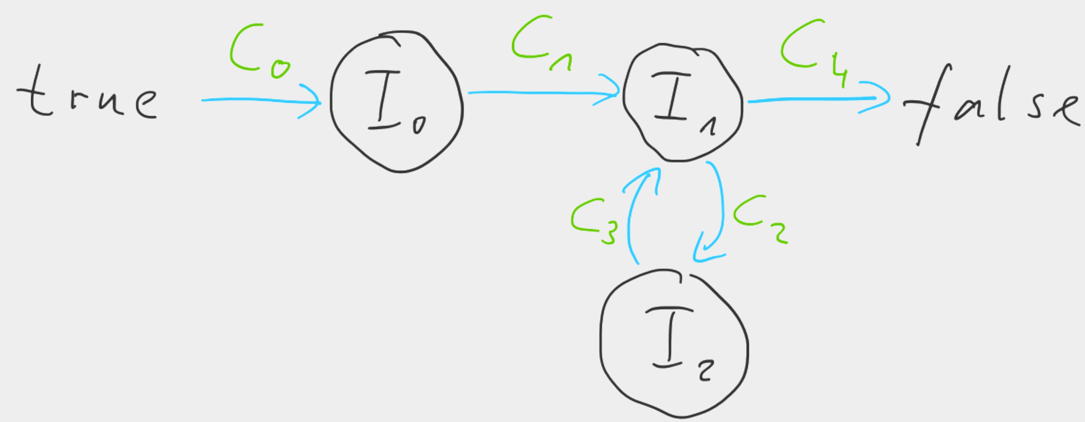
Horn clauses are **sat**
iff
no feasible path from `true` to `false` exists



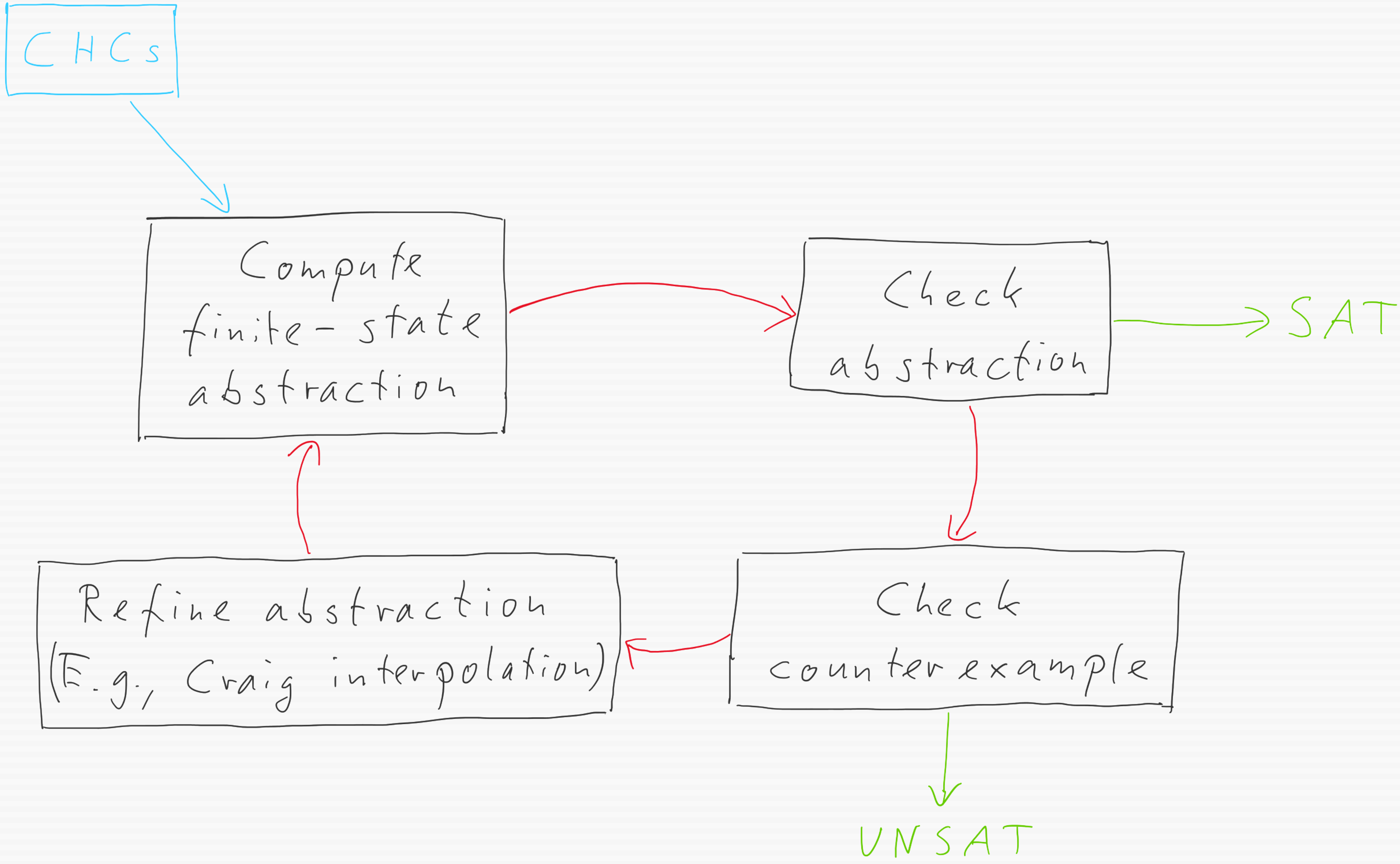
Constraints

$$C_0: \quad \forall x, y.\ \mathit{true} \rightarrow I_0(x \ldots$$
$$C_1: \quad \forall x, y.\ I_0(x, y) \rightarrow I \ldots$$
$$C_2: \quad \forall x, y.\ I_1(x, y) \rightarrow I_2(x + 1, y)$$
$$C_3: \quad \forall x, y.\ I_2(x, y) \rightarrow I_1(x, x + 2)$$
$$C_4: \quad \forall x, y.\ I_1(x, y) \land y < x \rightarrow \mathit{false}$$

Back to a "control-flow graph":



CEGAR

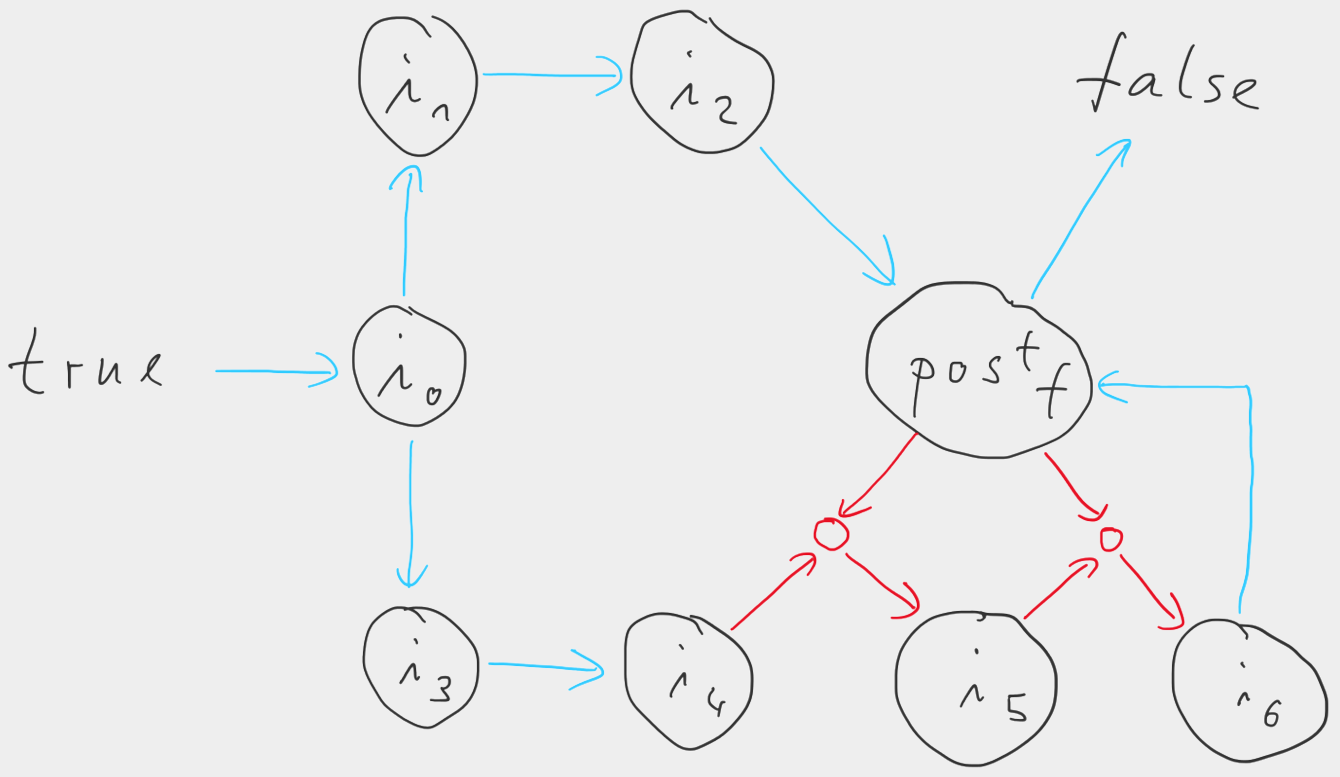


CHCs
Compute finite-state abstraction
Check abstraction
SAT
Check counterexample
UNSAT
Refine abstraction (E.g., Craig interpolation)

Non-linear CHC

```
i0(X0, X)      :- X0=X.                               % int f(int x) {
i1(X0, X)      :- i0(X0, X), X > 100.                 %   if (x > 100) {
i2(X0, T0)     :- i1(X0, X), T0=X-10.                 %     int t0 = x - 10;
post_f(X0, T0) :- i2(X0, T0).                         %     return t0;
i3(X0, X)      :- i0(X0, X), X =< 100.                %   } else {
i4(X0, T0)     :- i3(X0, X), T0=X+11.                 %     int t0 = x + 11;
i5(X0, T1)     :- i4(X0, T0), post_f(T0, T1).         %     int t1 = f(t0);
i6(X0, T2)     :- i5(X0, T1), post_f(T1, T2).         %     int t2 = f(t1);
post_f(X0, T2) :- i6(X0, T2).                         %     return t2;
                                                      %   }
                                                      % }
false :- post_f(X, R), X =< 100, \+(R = 91).          % Assertion
```

“Control-flow hyper-graph”




```
i0(X0, X)                                                  x) {
i1(X0, X)                                                  100) {
i2(X0, T0                                                  0 = x - 10;
post_f(X0                                                  rn t0;
i3(X0, X)      :- i0(X0, X), X =< 100.         %   } else {
i4(X0, T0)     :- i3(X0, X), T0=X+11.          %     int t0 = x + 11;
i5(X0, T1)     :- i4(X0, T0), post_f(T0, T1).  %     int t1 = f(t0);
i6(X0, T2)     :- i5(X0, T1), post_f(T1, T2).  %     int t2 = f(t1);
post_f(X0, T2) :- i6(X0, T2).                  %     return t2;
                                               %   }
                                               % }
false :- post_f(X, R), X =< 100, \+(R = 91).   % Assertion
```

Linear → Non-Linear CHC

	Linear CHC	Non-linear CHC
Abstract reachability:	graph	hyper-graph
Counterexample:	path	dag/tree
Craig Interpolant:	sequence	tree

CHC-COMP 2018

Arie Gurfinkel

Philipp Ruemmer, Grigory Fedyukovich, Adrien Champion

1st Competition on Solving Constrained Horn Clauses

UNIVERSITY OF WATERLOO

Competition affiliated with Workshop on Horn Clauses for Verification and Synthesis (HCVS)

CHC-COMP 2018

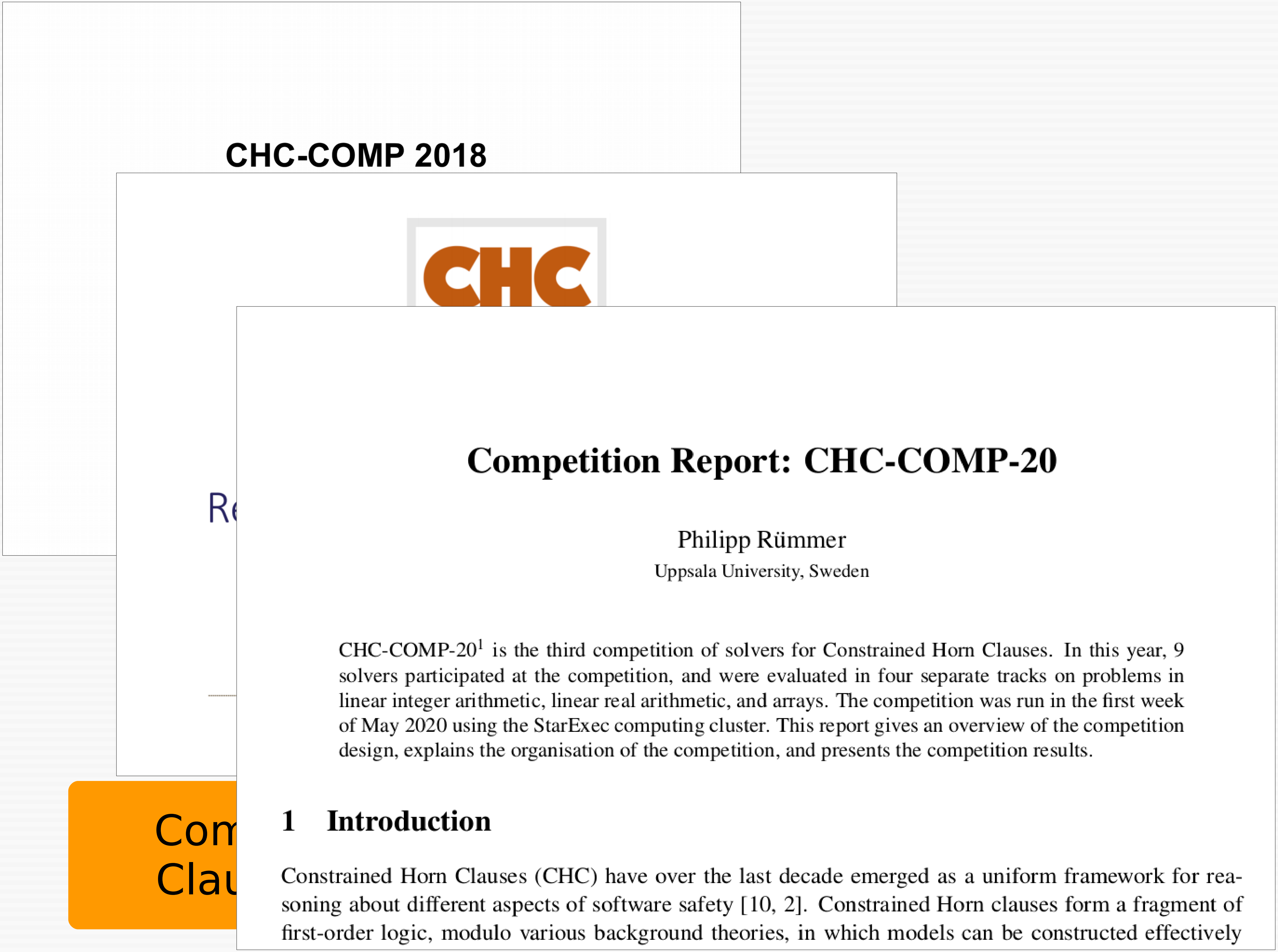
CHC COMP

Report on the Second Edition of the CHC Competition

Grigory Fedyukovich

April 7, Prague

Competition affiliated with Workshop on Horn Clauses for Verification and Synthesis (HCVS)



Competition Report: CHC-COMP-20

Philipp Rümmer
Uppsala University, Sweden

CHC-COMP-20[1] is the third competition of solvers for Constrained Horn Clauses. In this year, 9 solvers participated at the competition, and were evaluated in four separate tracks on problems in linear integer arithmetic, linear real arithmetic, and arrays. The competition was run in the first week of May 2020 using the StarExec computing cluster. This report gives an overview of the competition design, explains the organisation of the competition, and presents the competition results.

1 Introduction

Constrained Horn Clauses (CHC) have over the last decade emerged as a uniform framework for reasoning about different aspects of software safety [10, 2]. Constrained Horn clauses form a fragment of first-order logic, modulo various background theories, in which models can be constructed effectively

Competition Design in 2020

- 4 tracks:
  - LIA-nonlin
  - LIA-lin
  - LIA-lin-arrays
  - LRA-TS

- 8 solvers competing, 1 hors concours
- StarExec; 1800s timeout; 64GB memory
- https://chc-comp.github.io/

Competition Design in 2020

Linear arithmetic constraints

Non-linear clauses

- 4 tracks:
  - LIA-nonlin
  - LIA-lin
  - LIA-lin-arrays
  - LRA-TS
- 8 solvers competing, 1 hors concours
- StarExec; 1800s timeout; 64GB memory
- https://chc-comp.github.io/

Solver	Score	#sat	#unsat	CPU time (s)	Wall-clock (s)	Speedup	SotAC
Spacer	554	292	262	6.03	6.11	0.99	0.28
Eldarica (HC)	513	265	248	43.58	19.10	2.28	0.23
Eldarica-abs	513	266	247	52.07	35.96	1.45	0.23
U. Unihorn	420	212	208	75.73	49.11	1.54	0.21
PCSat	331	156	175	92.10	29.54	3.12	0.20
U. TreeAutomizer	118	34	84	41.17	30.00	1.37	0.17
Any solver	560	298	262				

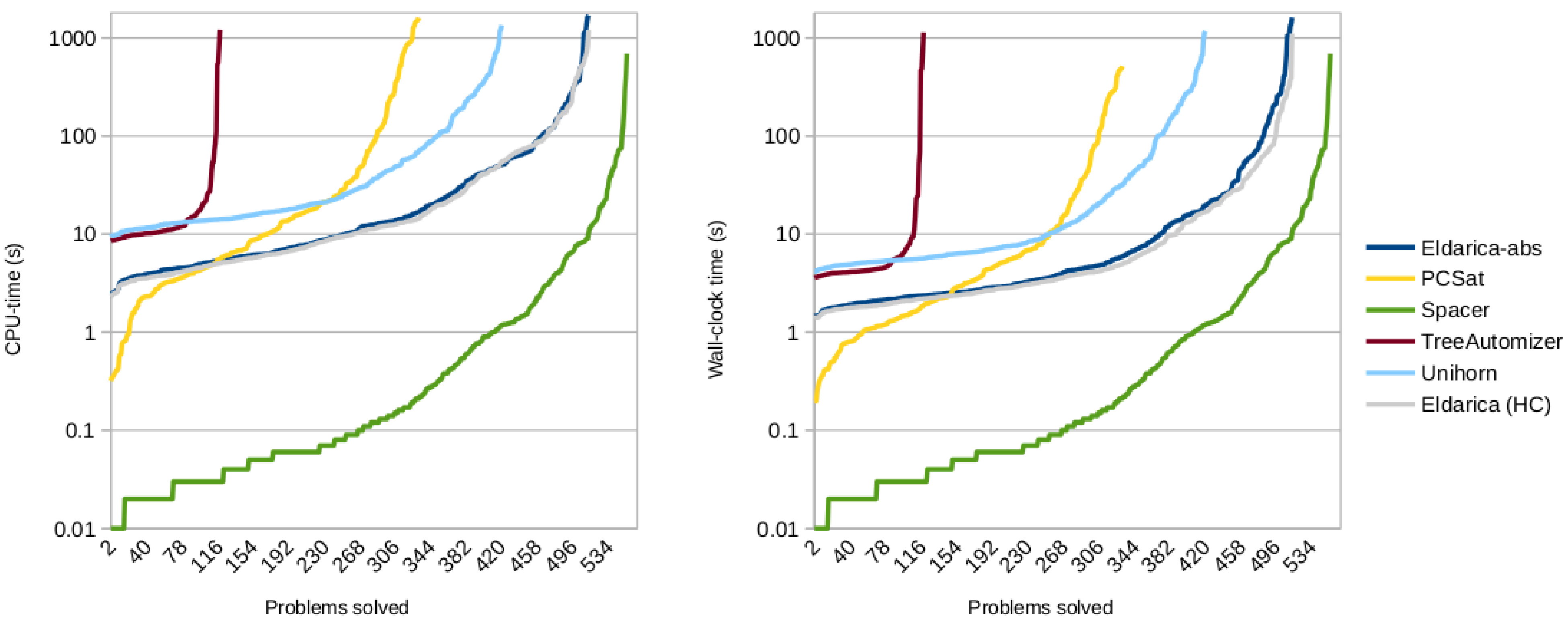


Figure 1: Solver performance on the 565 benchmarks of the LIA-nonlin track

Solver	Score	#sat	#unsat	CPU time (s)	Wall-clock (s)	Speedup	SotAC
Spacer	518	330	188	11.94	12.03	0.99	0.22
Eldarica-abs	477	300	177	57.26	39.59	1.45	0.20
Eldarica (HC)	476	300	176	48.58	20.00	2.43	0.20
U. Unihorn	407	240	167	43.57	26.21	1.66	0.17
IC3IA	400	260	140	46.09	46.23	1.00	0.20
PCSat	329	191	138	37.91	12.23	3.10	0.17
U. TreeAutomizer	307	166	141	50.30	37.43	1.34	0.17
Any solver	558	356	202				

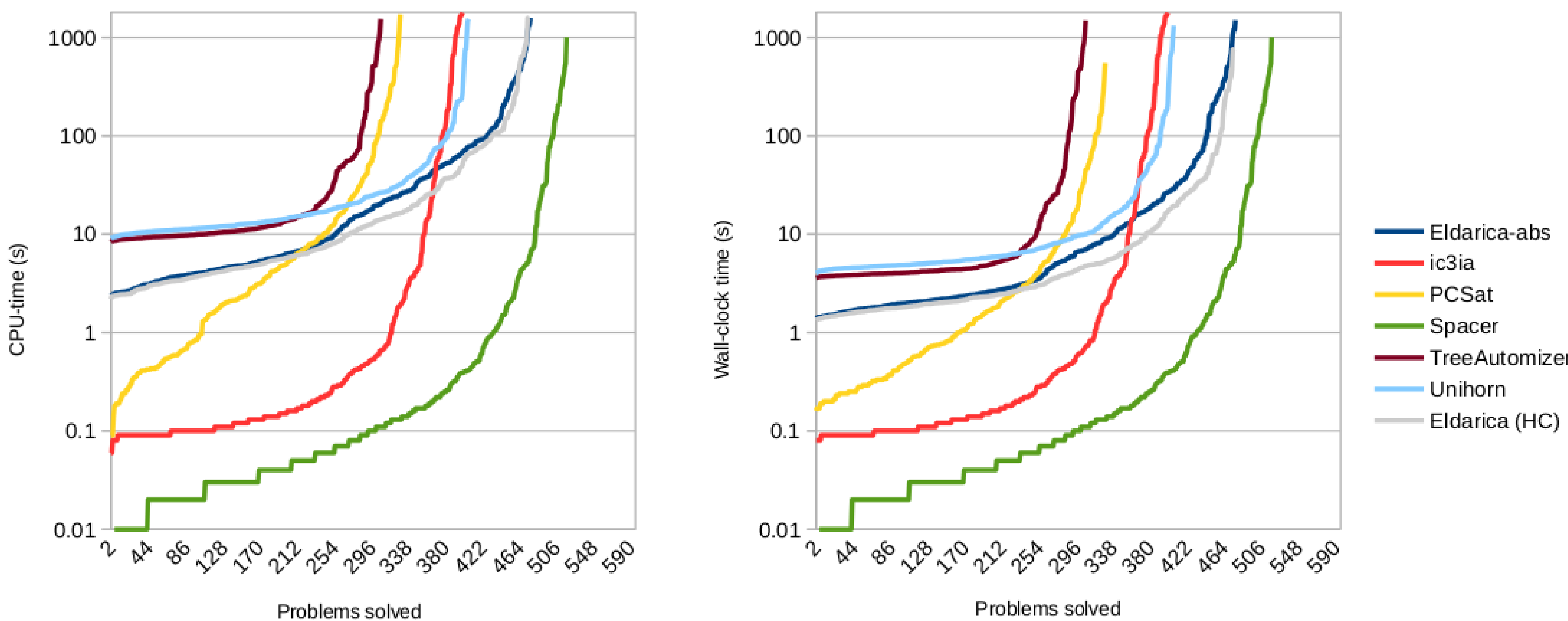
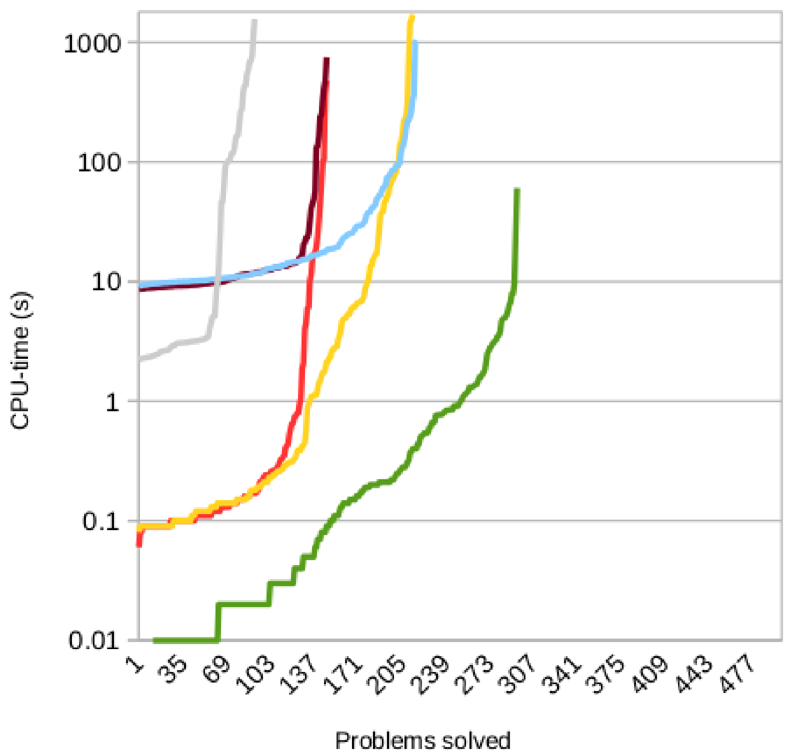


Figure 2: Solver performance on the 596 benchmarks of the LIA-lin track

Solver	Score	#sat	#unsat	CPU time (s)	Wall-clock (s)	Speedup	SotAC
Spacer	295	203	92	0.81	0.89	0.91	0.37
U. Unihorn	217	144	73	39.73	24.12	1.65	0.26
ProphIC3	214	140	74	38.24	19.17	1.99	0.34
IC3IA	147	92	55	9.17	9.30	0.99	0.24
U. TreeAutomizer	147	100	47	31.49	21.46	1.47	0.22
Eldarica (HC)	91	91	0	106.80	68.05	1.57	0.24
Any solver	350	250	100				



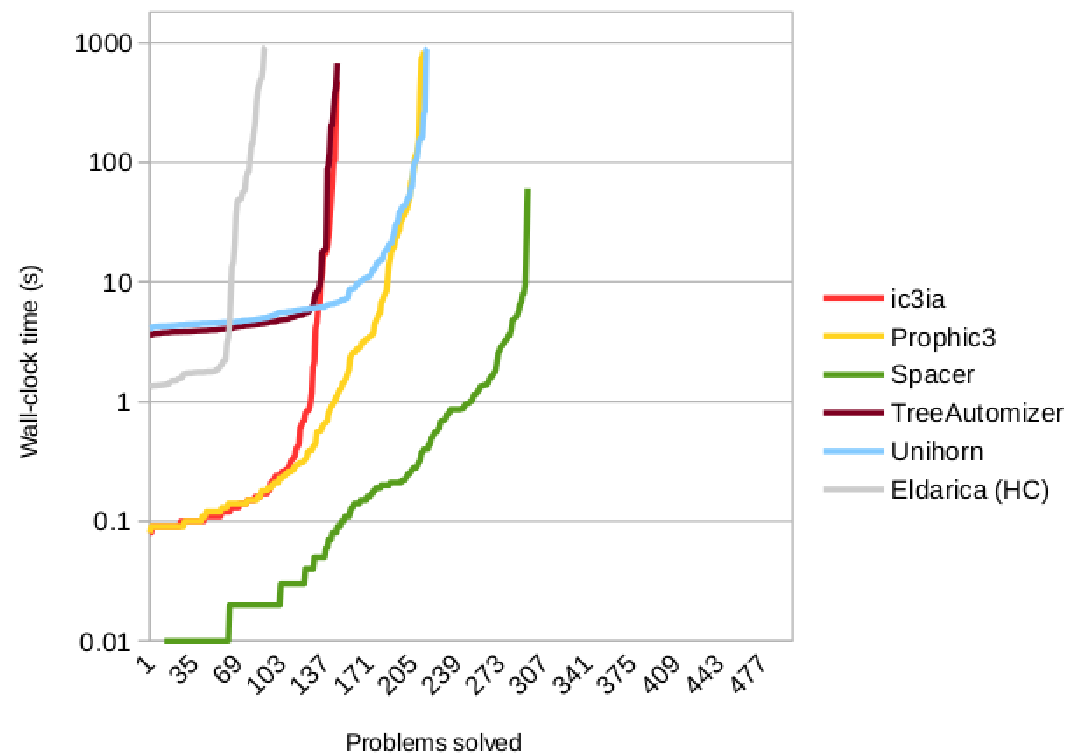


Figure 3: Solver performance on 500 benchmarks of the LIA-lin-arrays track (one benchmark on which Spacer and Ultimate Unihorn give conflicting answers is not counted)

Solver	Score	#sat	#unsat	CPU time (s)	Wall-clock (s)	Speedup	SotAC
IC3IA	468	378	90	136.94	137.05	1.00	0.29
Sally-parallel	439	360	79	138.81	47.37	2.93	0.24
Sally-decomposing-itp	438	357	81	107.61	107.68	1.00	0.24
Spacer	346	270	76	176.75	176.86	1.00	0.22
U. TreeAutomizer	168	131	37	239.75	202.11	1.19	0.19
U. Unihorn	160	103	57	213.33	158.57	1.35	0.18
Any solver	481	388	93				

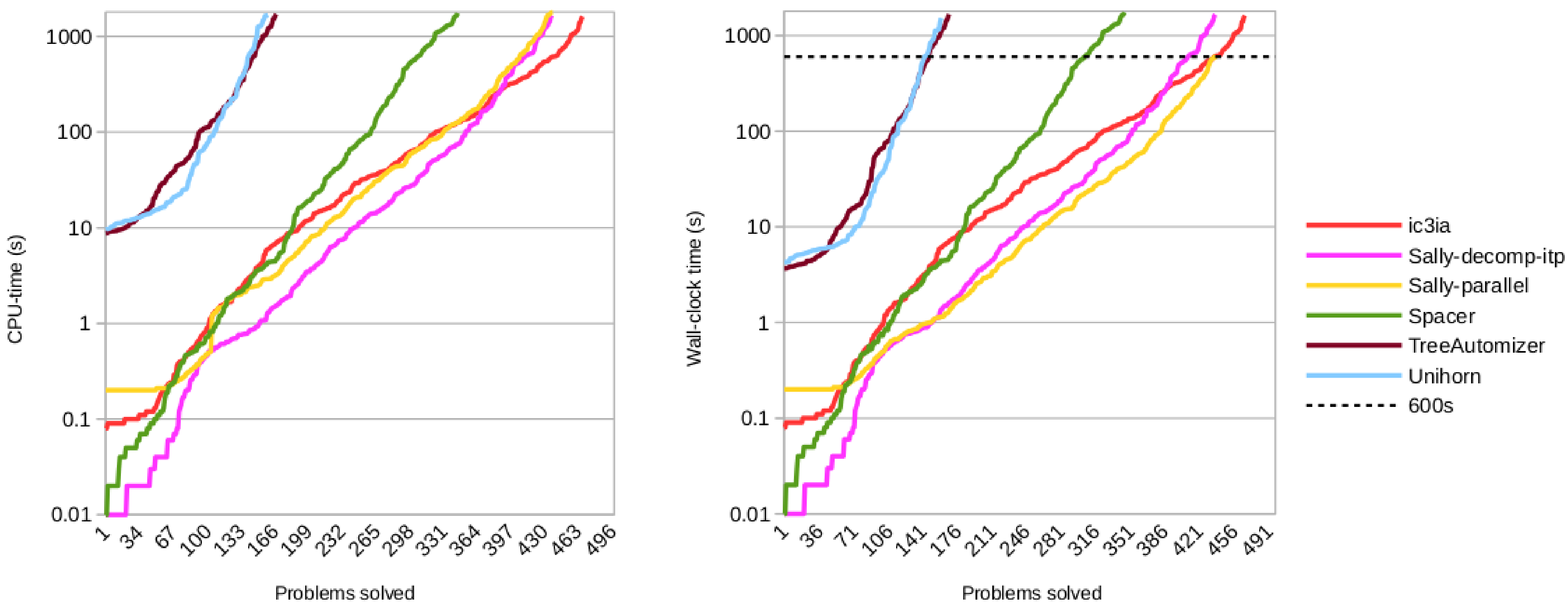
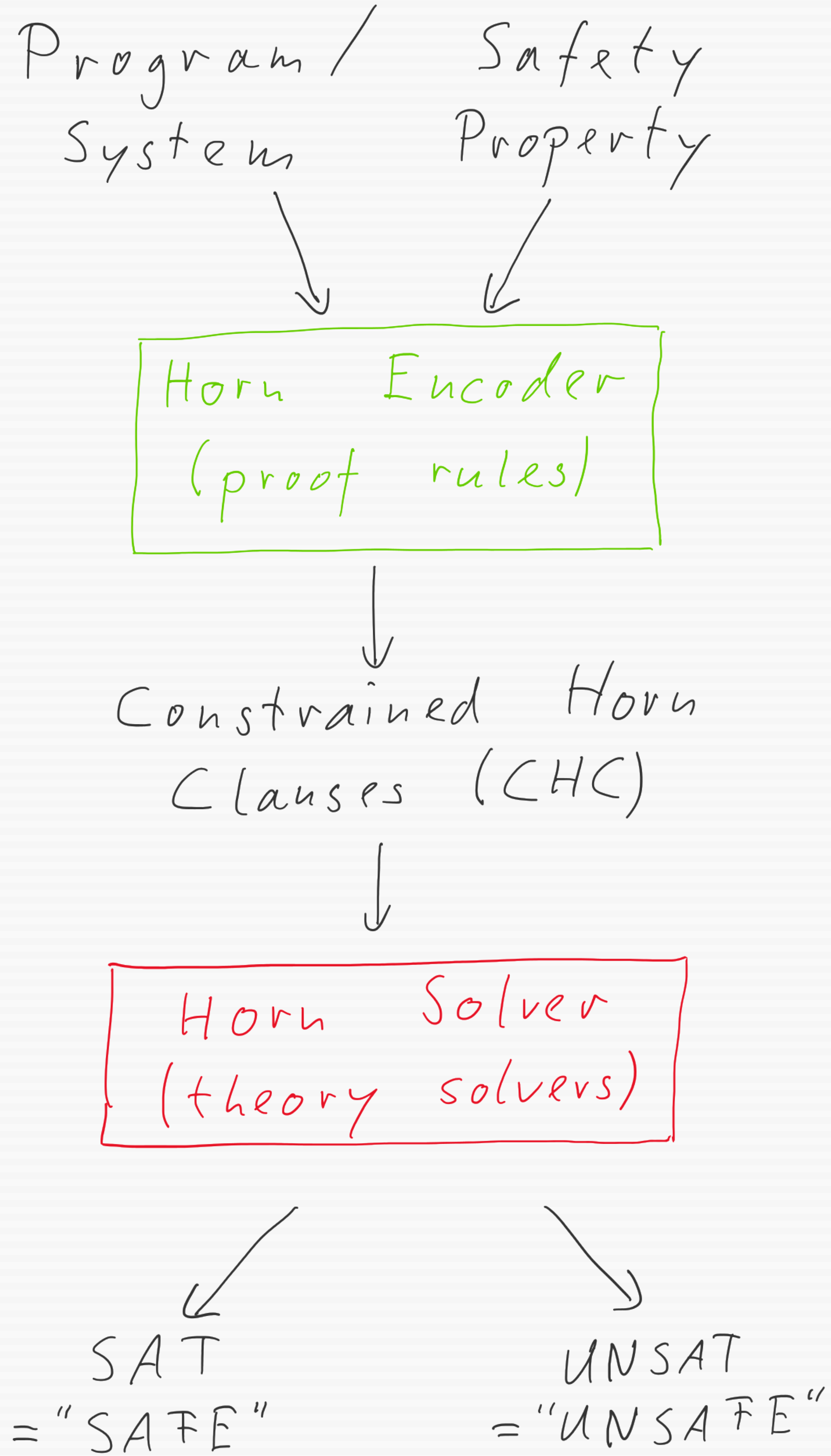


Figure 4: Solver performance on the 499 benchmarks of the LRA-TS track

What next?

- More tracks?
- More benchmarks?
- More solvers?

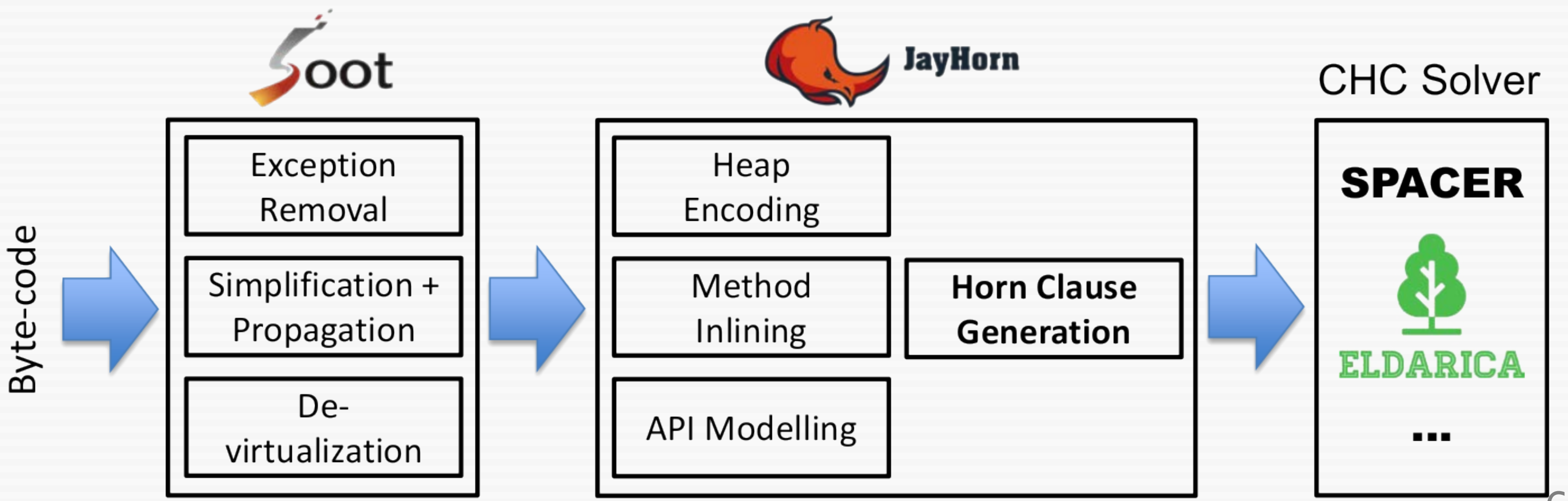
- C
- Java
- Ada
- Rust
- Networks of TA
- BIP models
- *etc.*



Verifying Java Programs

[1] Temesghen Kahsai, PR, Huascar Sanchez, Martin Schäf
JayHorn: A Framework for Verifying Java programs. CAV 2016

- Horn-based verification tool for Java, written in Java
- Open source, MIT licence



McCarthy 91 Example

```
import org.sosy_lab.sv_benchmarks.Verifier;

public class McCarthy91 {
    private static int f(int n) {
        if (n > 100)
            return n - 10;
        else
            return f(f(n + 11));
    }

    public static void main(String[] args) {
        int x = Verifier.nondetInt();
        int y = f(x);
        assert(x > 101 || y == 91);
    }
}
```

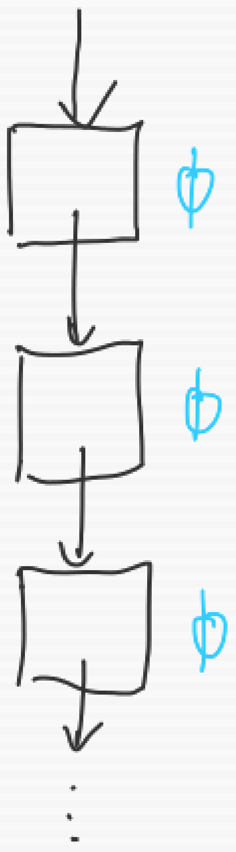
Representation of Heap

Representation of Heap

- Encoding using McCarthy Arrays
  - Precise, relatively complete
  - Hard to infer invariants automatically
- Refinement types, etc.
  - Incomplete
  - Easier to automate
- (Separation logic, ownership systems, dynamic frames, etc.)

Representation of Heap

- Encoding using McCarthy Arrays
    - Precise, relatively complete
    - Hard to infer invariants automatically
- Refinement types, etc.
    - Incomplete
    - Easier to automate
- (Separation logic, ownership systems, dynamic frames, etc.)

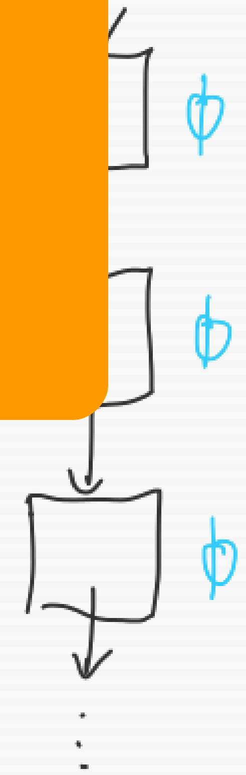


Representation of Heap

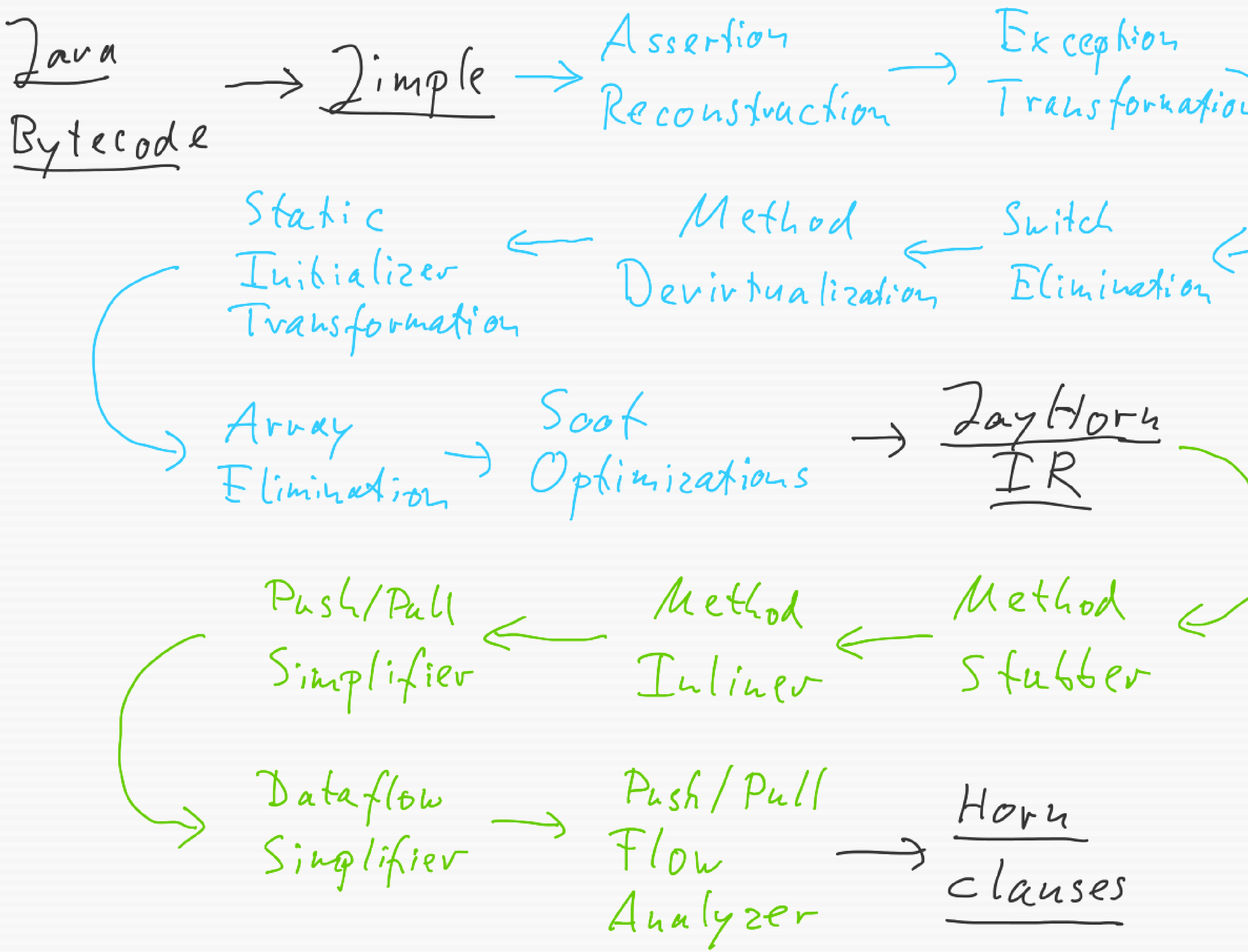
One of our current projects:
A theory of heap to abstract
from those different encodings

Zafer Esen, PR. Towards an SMT-LIB Theory of Heap. HCVS 2020

- Refinement types, etc.
    - Incomplete
    - Easier to automate
- (Separation logic, ownership systems, dynamic frames, etc.)



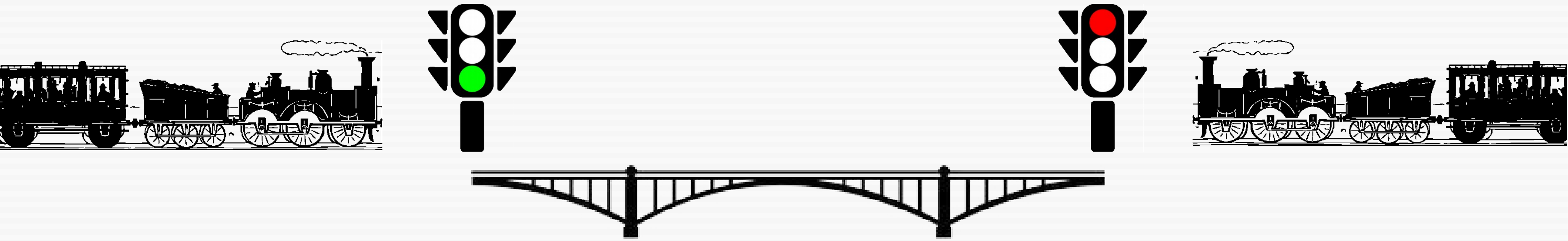
Data-Flow


Java Bytecode
Jimple
Assertion Reconstruction
Exception Transformation
Switch Elimination
Method Devirtualization
Static Initializer Transformation
Array Elimination
Soot Optimizations
JayHorn IR
Method Stubber
Method Inliner
Push/Pull Simplifier
Dataflow Simplifier
Push/Pull Flow Analyzer
Horn clauses

Verifying Networks of Timed Automata

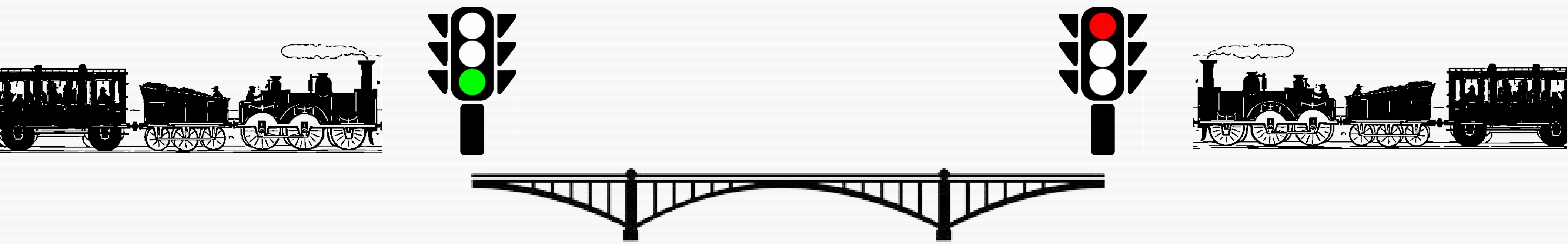
[2] Hossein Hojjat, PR, Pavle Subotic, Wang Yi. Horn Clauses for Communicating Timed Systems. HCVS 2014

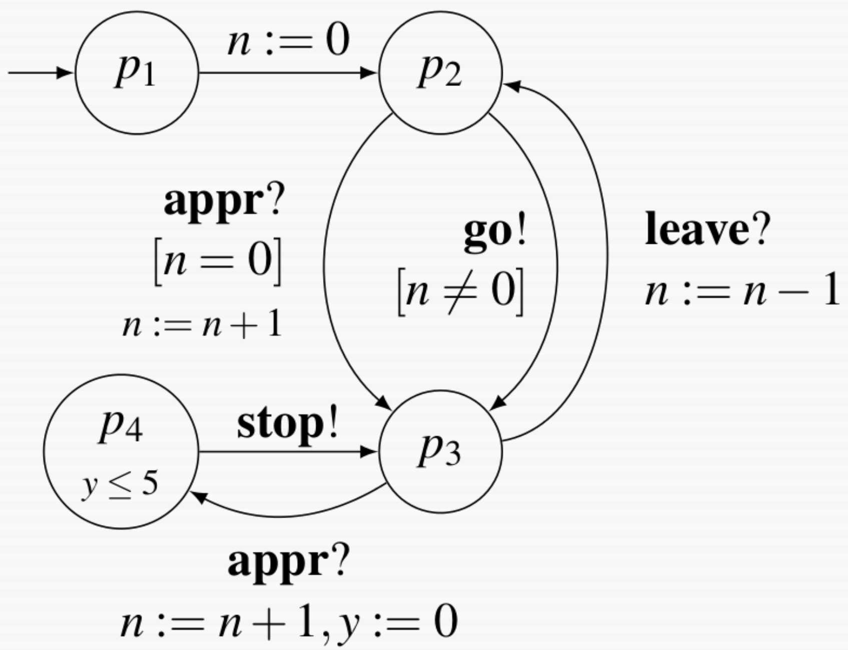
Train Crossing Model

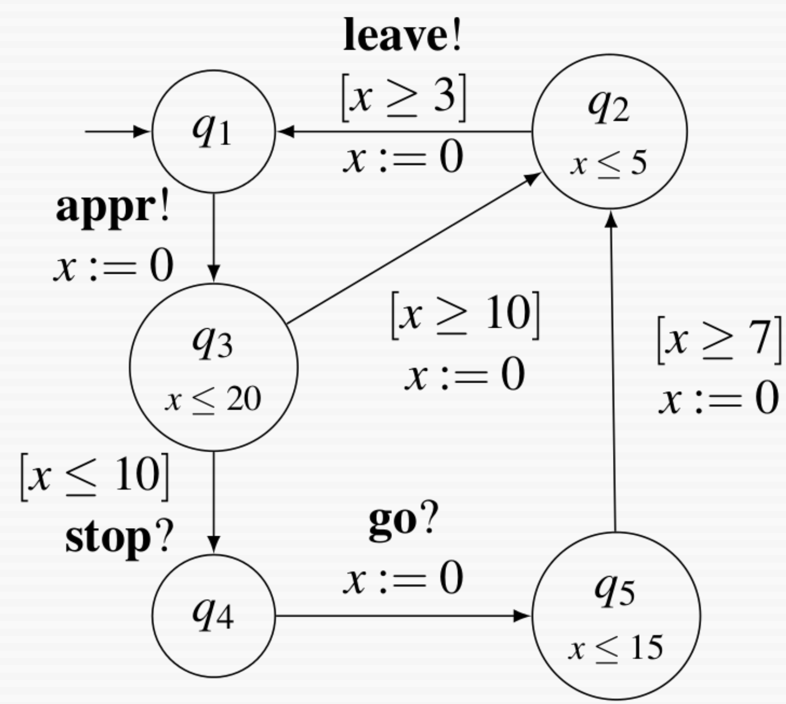


[FORTE'94]

Train Crossing Model

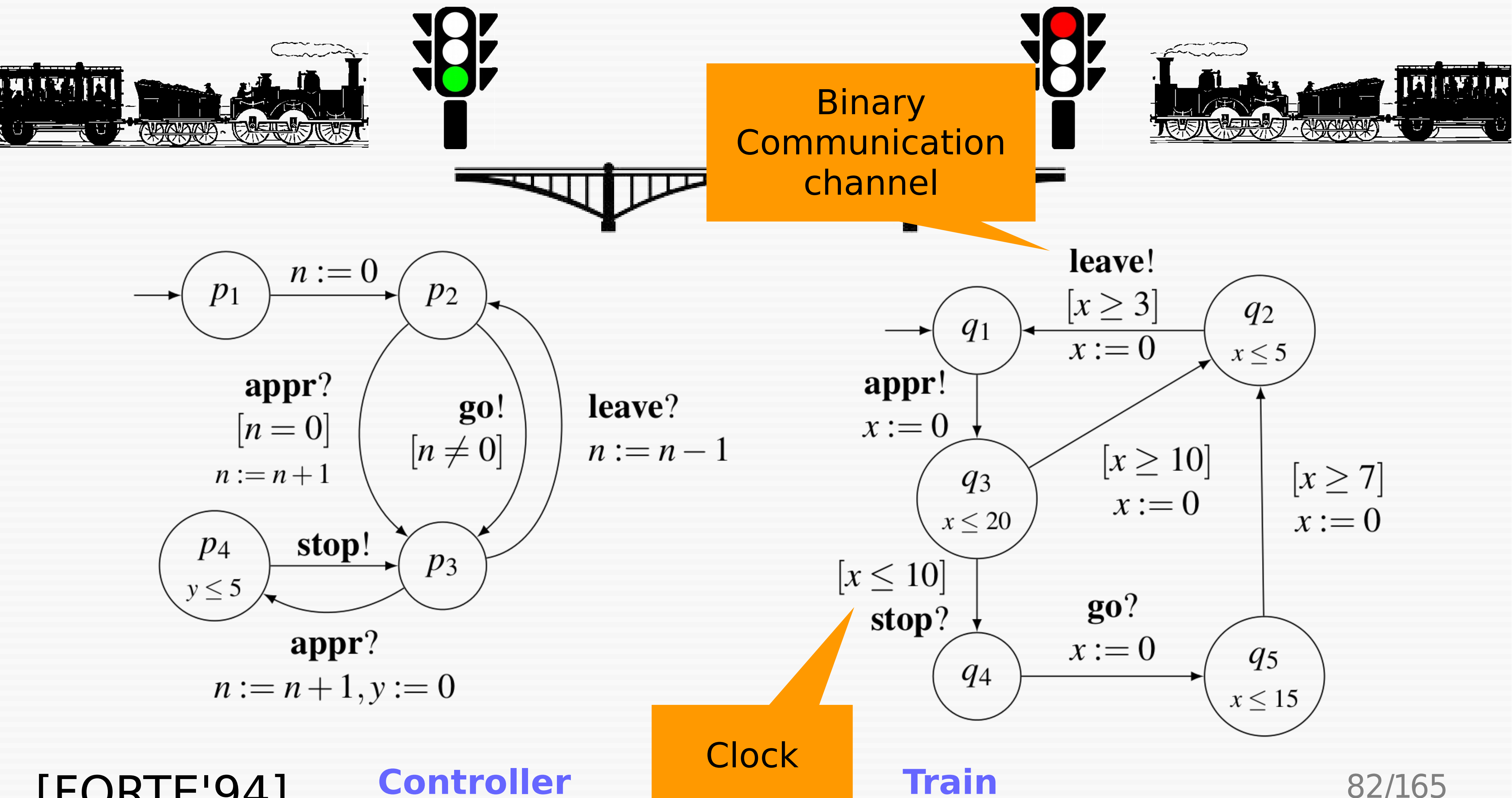






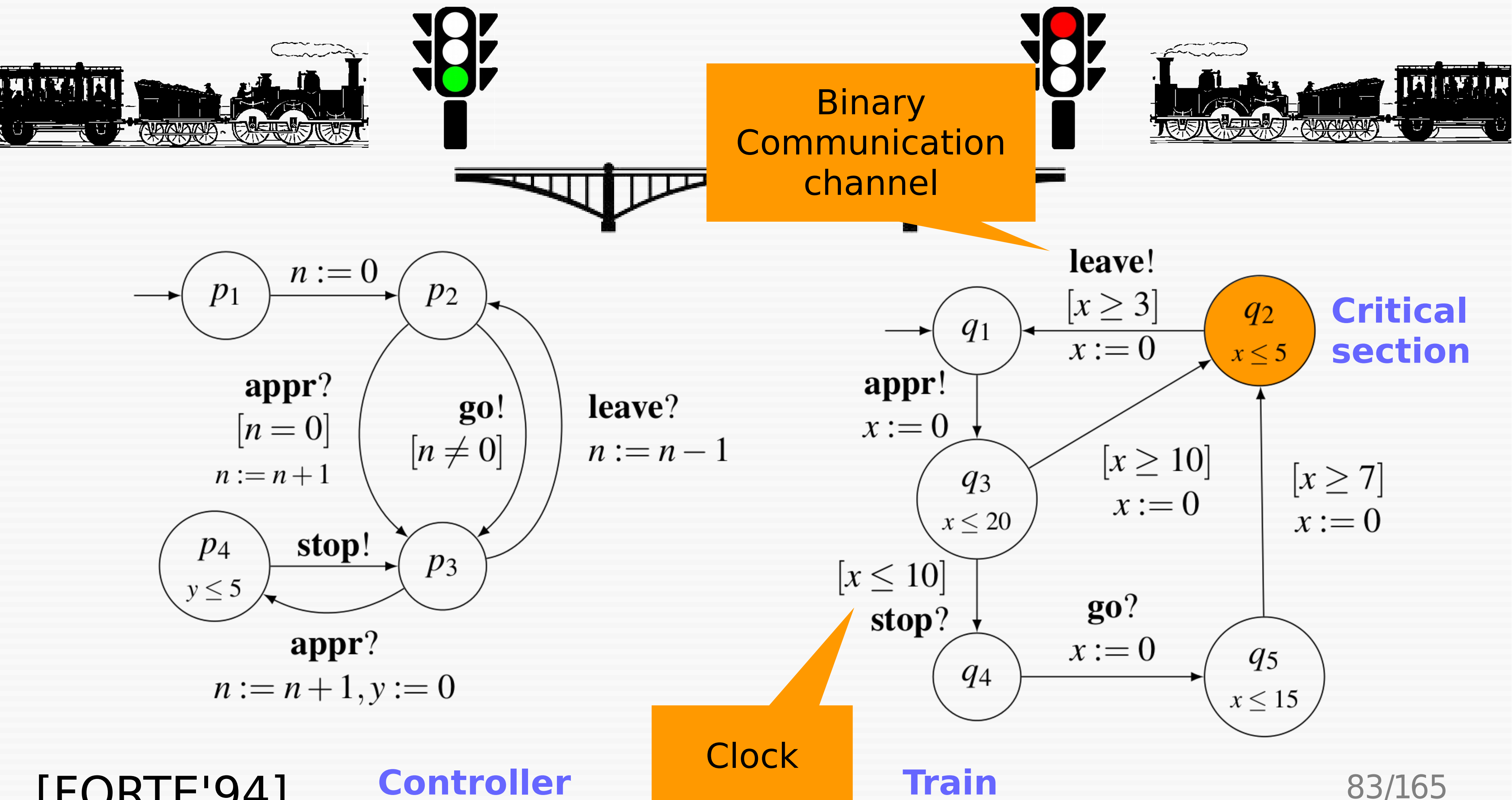
[FORTE'94] **Controller** **Train**

Train Crossing Model



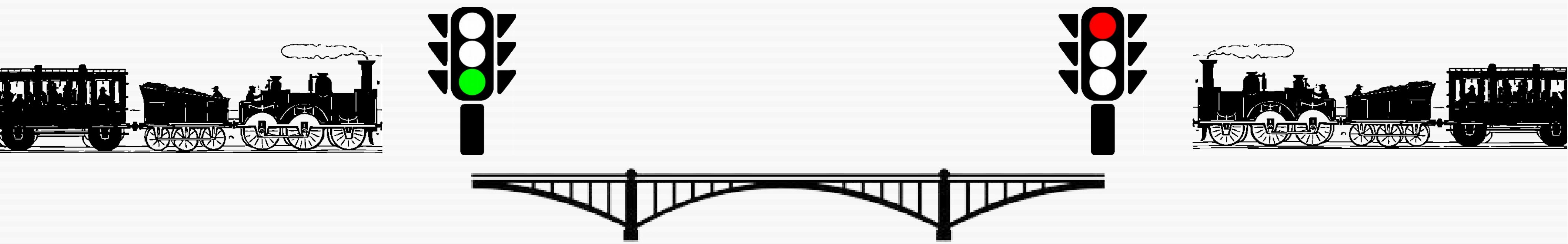
[FORTE'94]

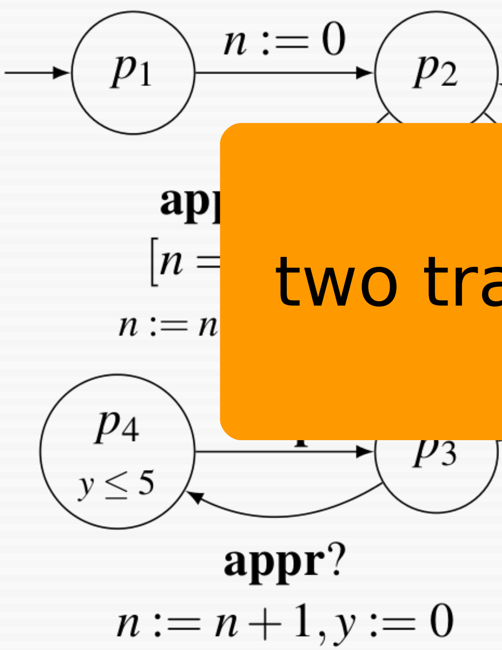
Train Crossing Model

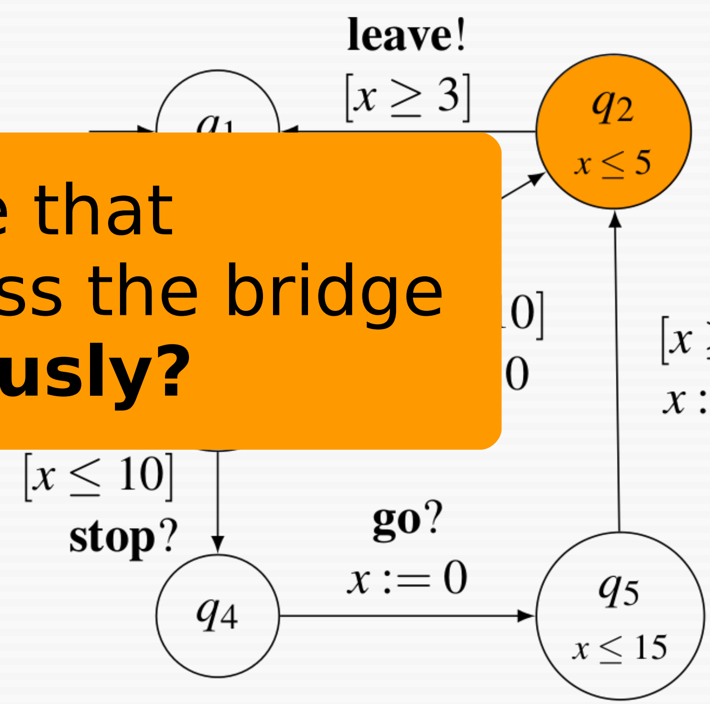


[FORTE'94] **Controller** **Train**

Train Crossing Model







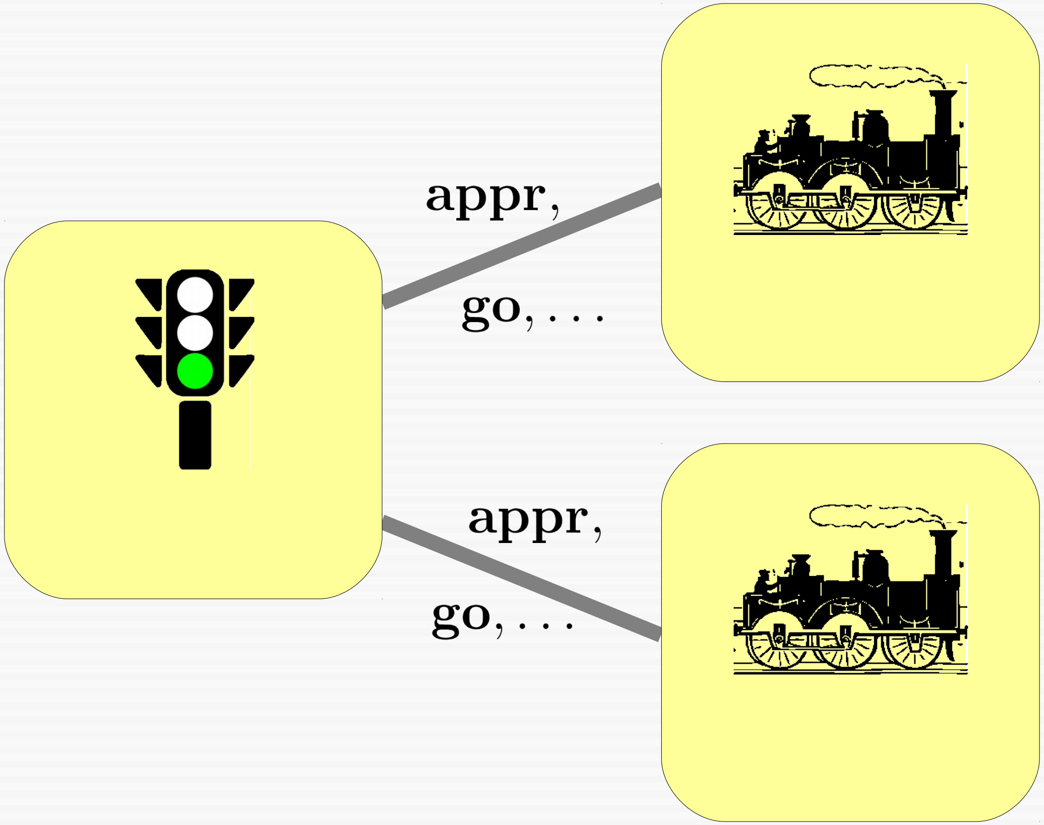
Is it possible that
two trains try to cross the bridge
simultaneously?

[FORTE'94]

Controller

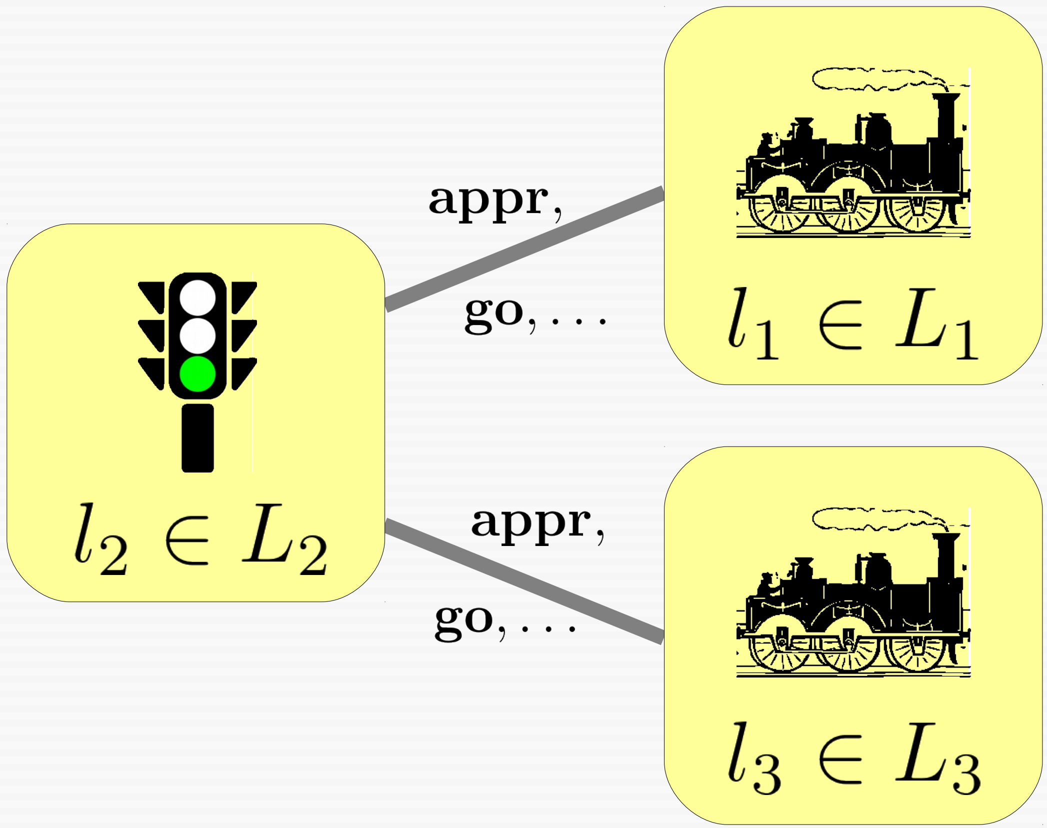
Train

With Two Trains

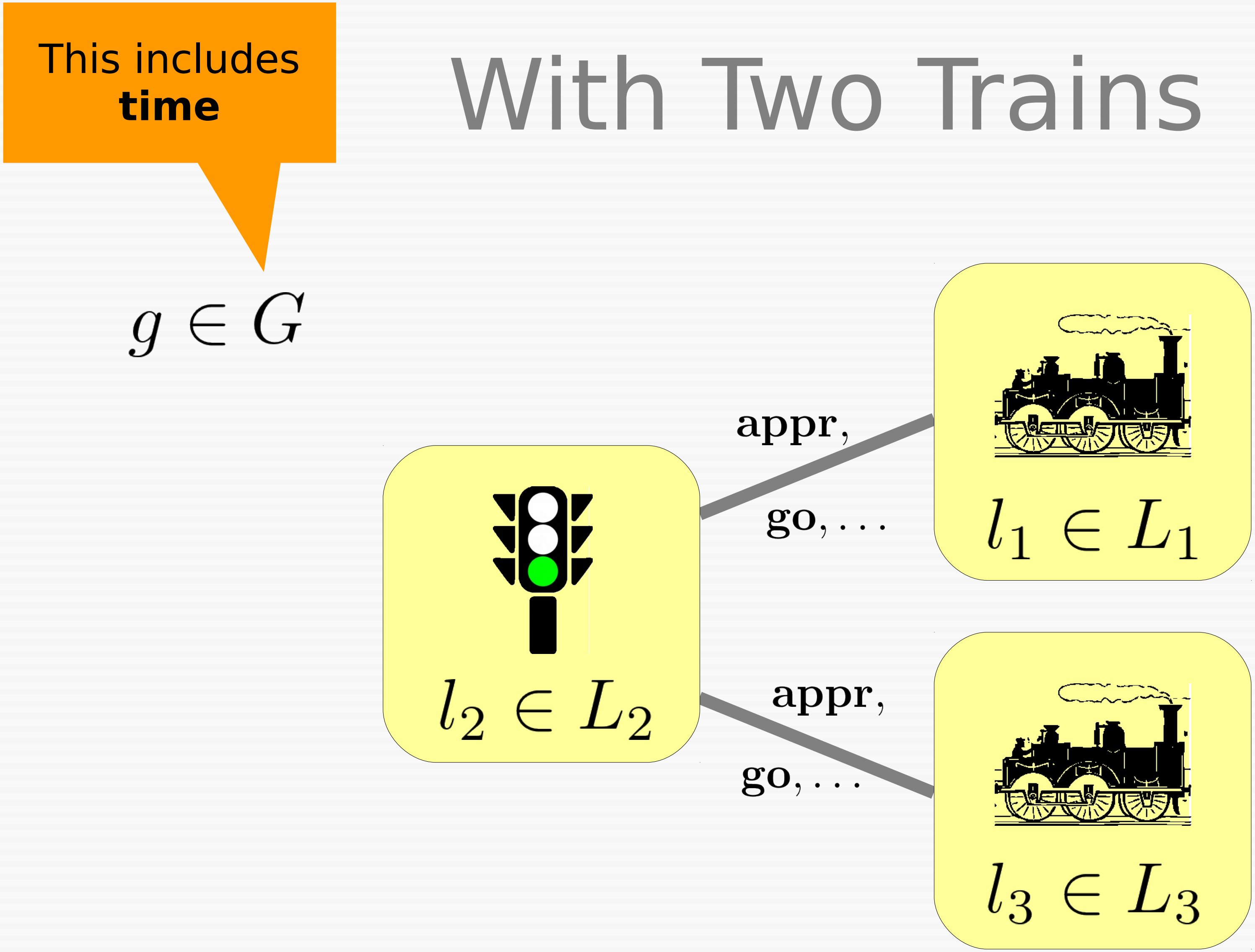


With Two Trains

$g \in G$

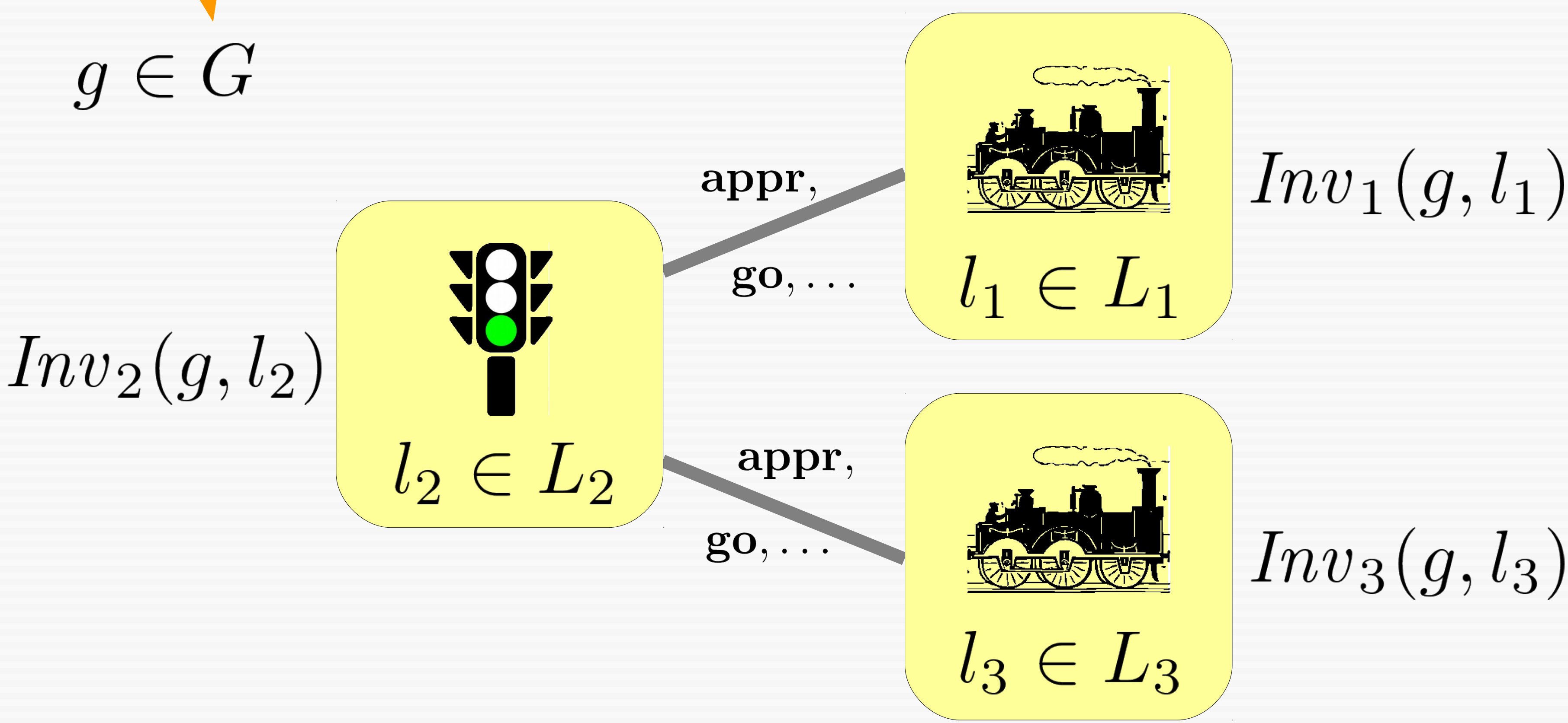


With Two Trains



This includes **time**

With Two Trains

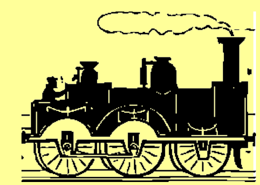


System invariant: $Inv_1(g, l_1) \wedge Inv_2(g, l_2) \wedge Inv_3(g, l_3)$

With Two Trains

$g \in G$

appr, **go**, ...



$l_1 \in L_1$

$Inv_1(g, l_1)$

Horn Constraints

Local transitions:

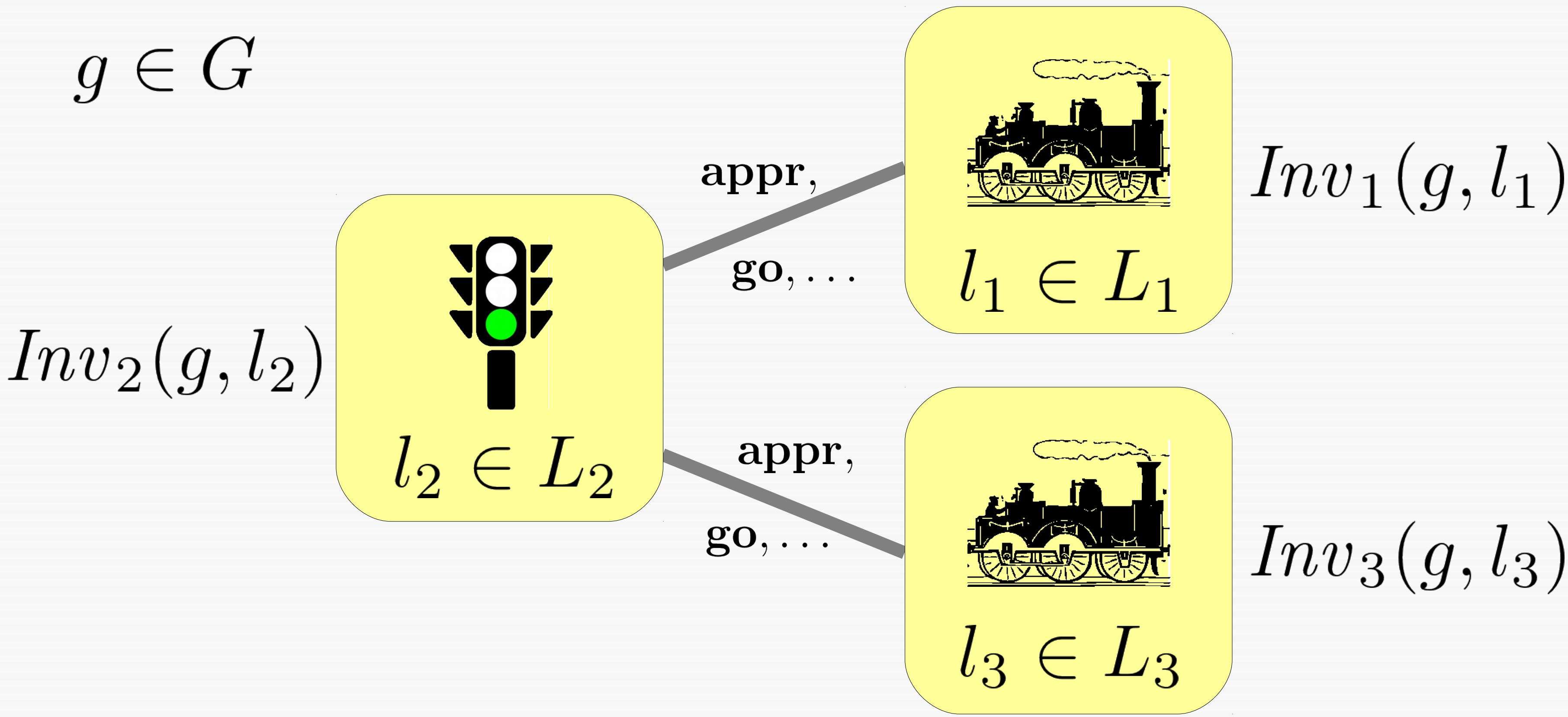
$$(\langle g, l_1 \rangle \rightsquigarrow \langle g', l_1' \rangle) \wedge Inv_1(g, l_1) \rightarrow Inv_1(g', l_1')$$

Owicki-Gries-style non-interference:

$$(\langle g, l_1 \rangle \rightsquigarrow \langle g', l_1' \rangle) \wedge Inv_1(g, l_1) \wedge Inv_2(g, l_2) \rightarrow Inv_2(g', l_2)$$

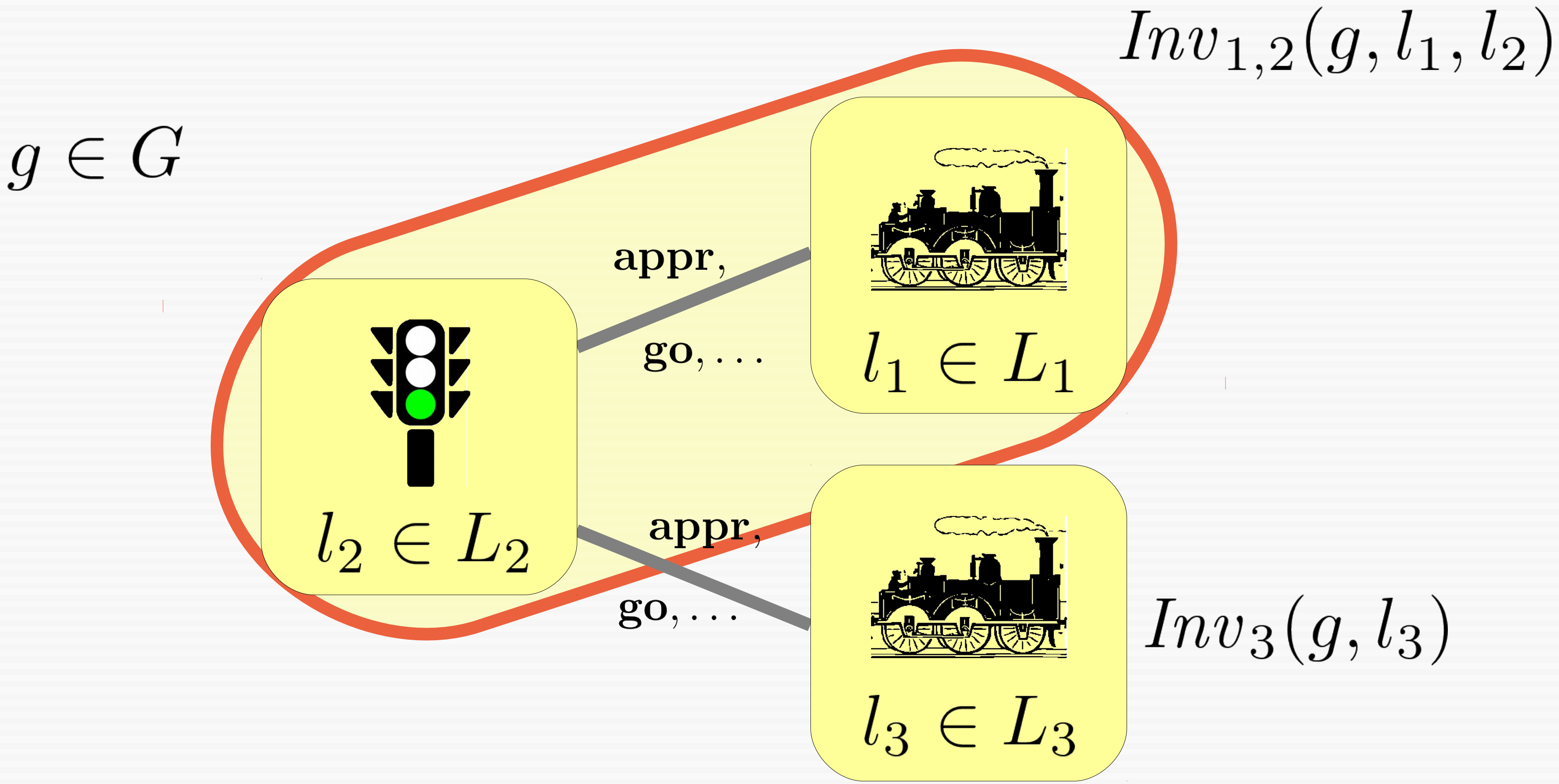
\+ time elapse, synch., initiation, assertions

With Two Trains

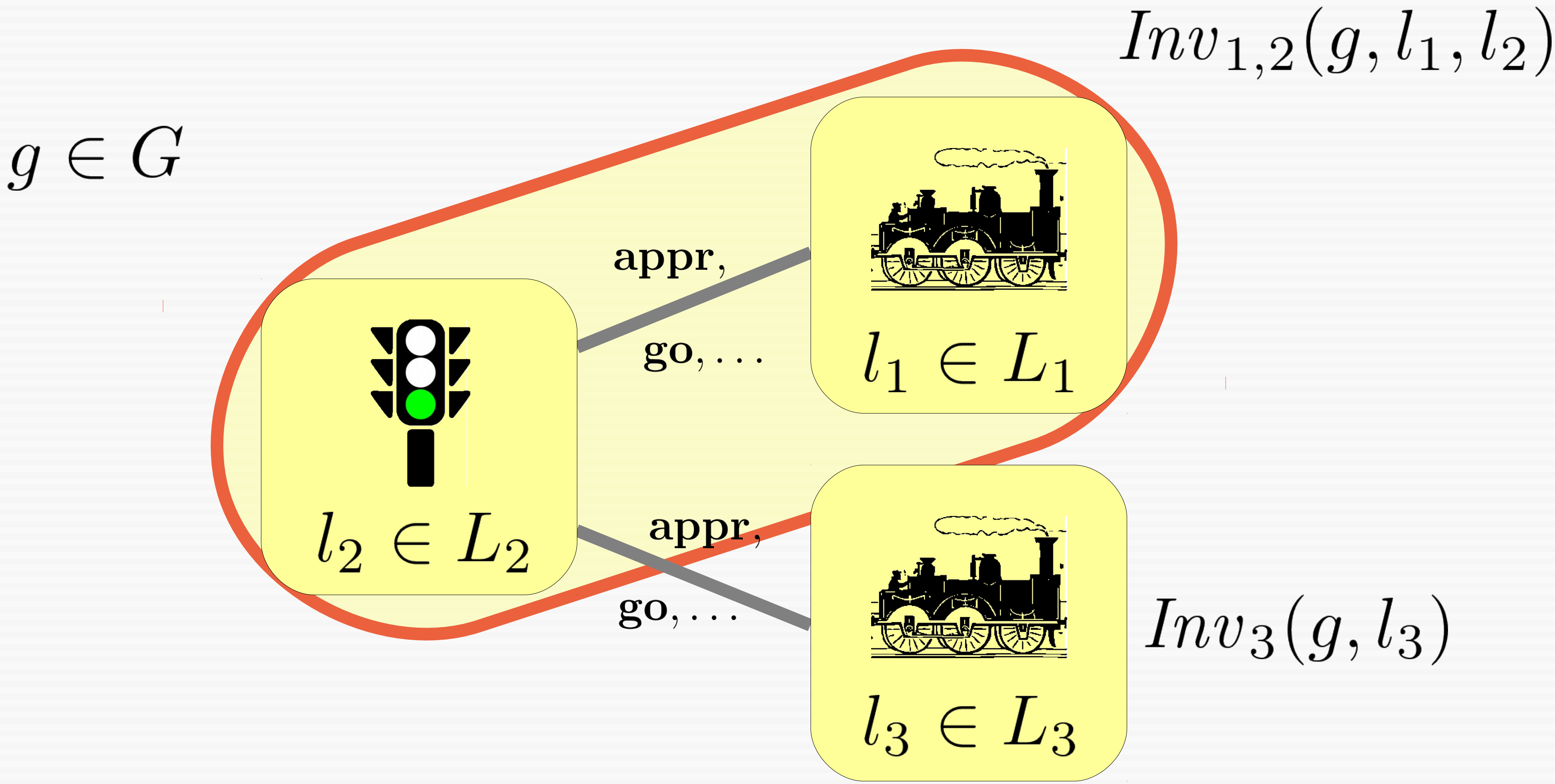


System invariant: $Inv_1(g, l_1) \wedge Inv_2(g, l_2) \wedge Inv_3(g, l_3)$

With Two Trains

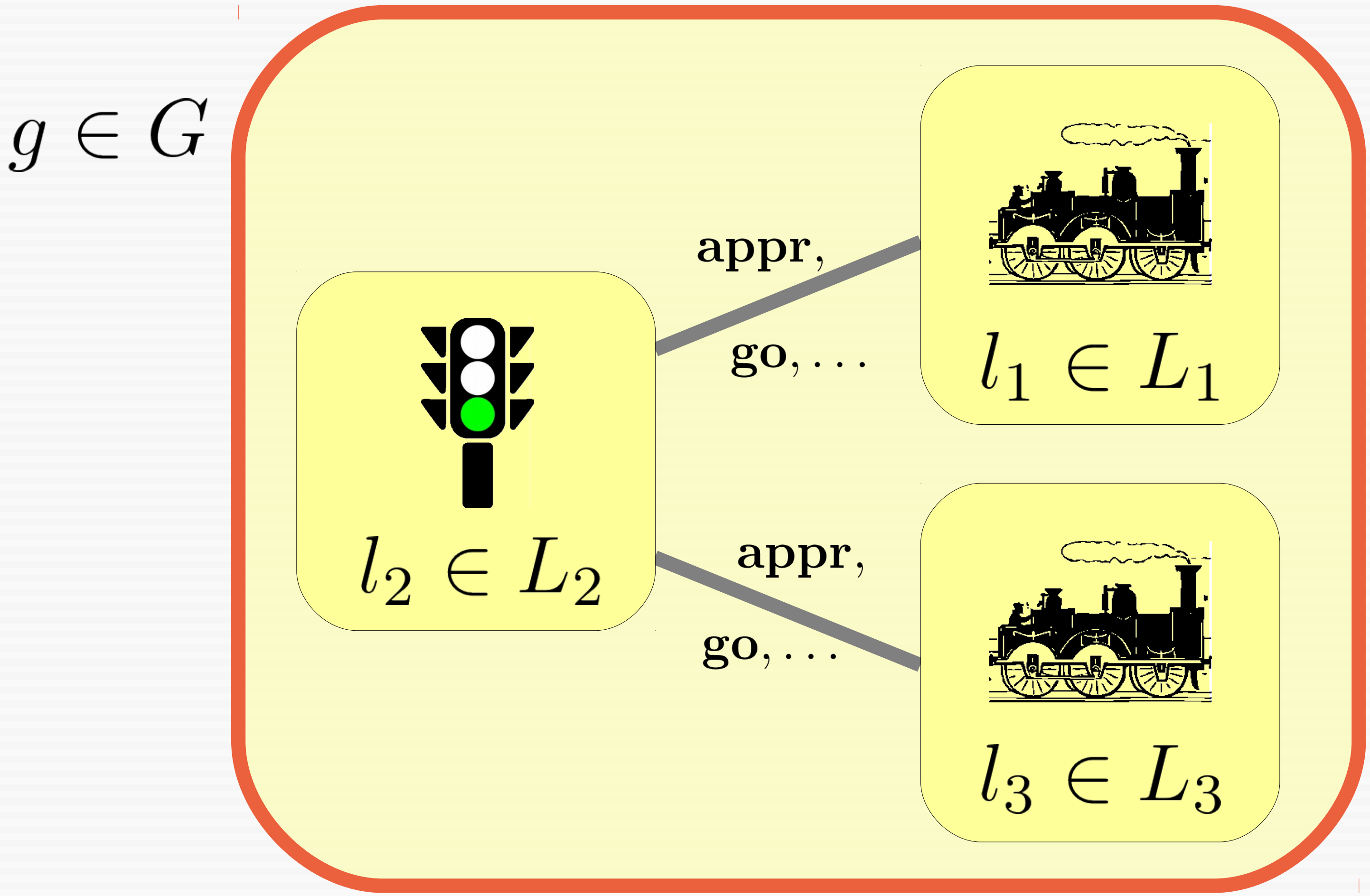


With Two Trains



System invariant: $Inv_{1,2}(g, l_1, l_2) \wedge Inv_3(g, l_3)$

With Two Trains



System invariant: $Inv_{1,2,3}(g, l_1, l_2, l_3)$

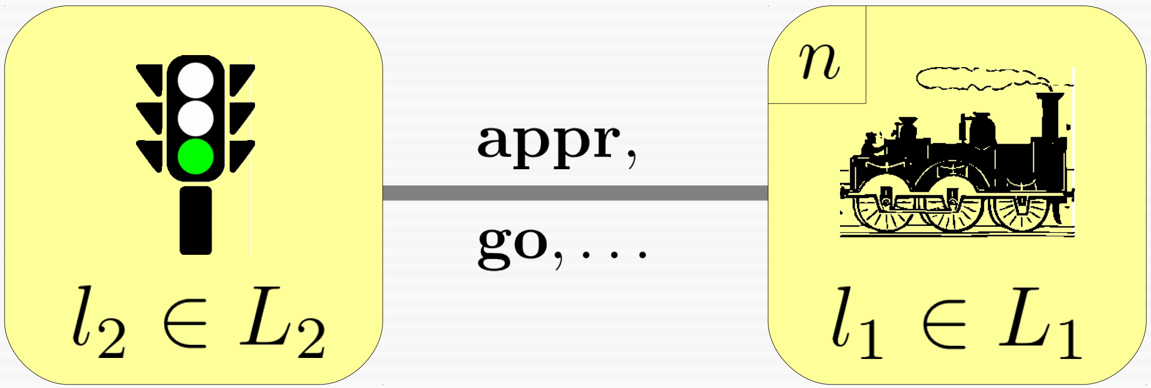
Spectrum of Possible Invariant Schemata

Modular
Separate invariant
for each process

↔

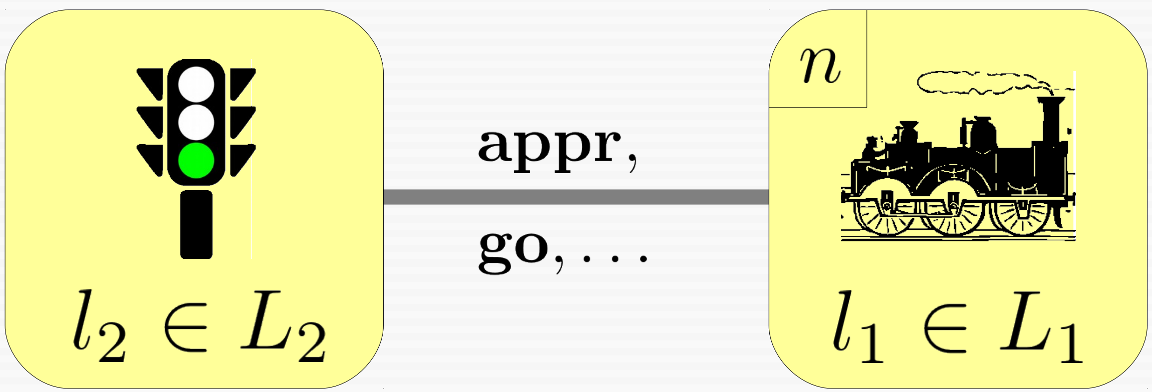
Monolithic
Single invariant
for whole system

Parameterised systems



Parameterised systems

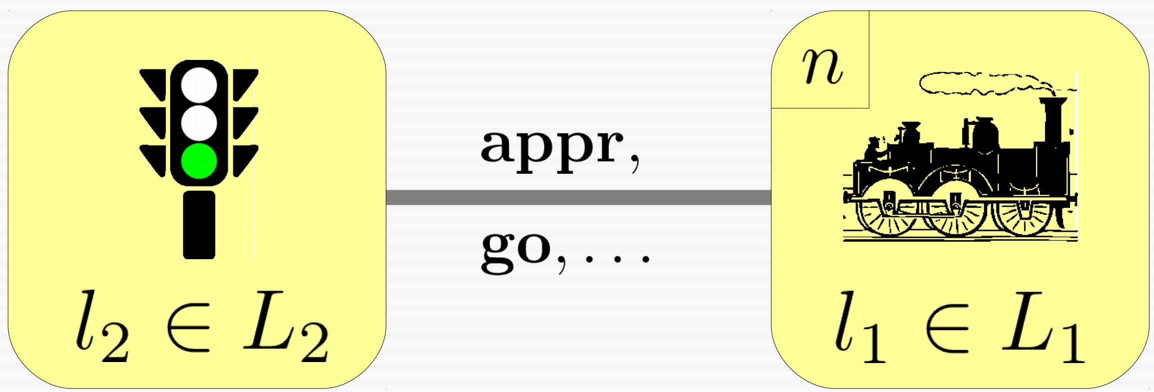
Can we verify mutual exclusion for **any number** of trains?



Parameterised systems

$g \in G$

$Inv_2(g, l_2)$



$Inv_1(g, n, l_1)$

$$\forall n.\ Inv_1(g, n, \bar{l}_1[n]) \wedge Inv_2(g, l_2)$$

Parameterised systems

Horn Constraints

Local transitions:

$$\big(\langle g, l_1\rangle \overset{n}{\leadsto} \langle g', l_1'\rangle\big) \wedge Inv_1(g, n, l_1) \rightarrow Inv_1(g', n, l_1')$$

Owicki-Gries-style non-interference:

$$\left(\begin{array}{l}\big(\langle g, l_1\rangle \overset{n_1}{\leadsto} \langle g', l_1'\rangle\big) \wedge n_1 \neq n_2 \wedge \\ Inv_1(g, n_1, l_1) \wedge Inv_1(g, n_2, l_1')\end{array}\right) \rightarrow Inv_1(g', n_2, l_1')$$

+ time elapse, synch., initiation, assertions

$$\forall n.\ Inv_1(g, n, \bar{l}_1[n]) \wedge Inv_2(g, l_2)$$

Parameterised systems

Horn Constraints

Local transitions:

$$\big(\langle g, l_1\rangle \overset{n}{\rightsquigarrow} \langle g', l_1'\rangle\big) \wedge Inv_1(g, n, l_1) \rightarrow Inv_1(g', n, l_1')$$

Owicki-Gries-style non-interference:

"Self-reflection"

$$\begin{pmatrix} \big(\langle g, l_1\rangle \overset{n_1}{\rightsquigarrow} \langle g', l_1'\rangle\big) \wedge n_1 \neq n_2 \wedge \\ Inv_1(g, n_1, l_1) \wedge Inv_1(g, n_2, l_1') \end{pmatrix} \rightarrow Inv_1(g', n_2, l_1')$$

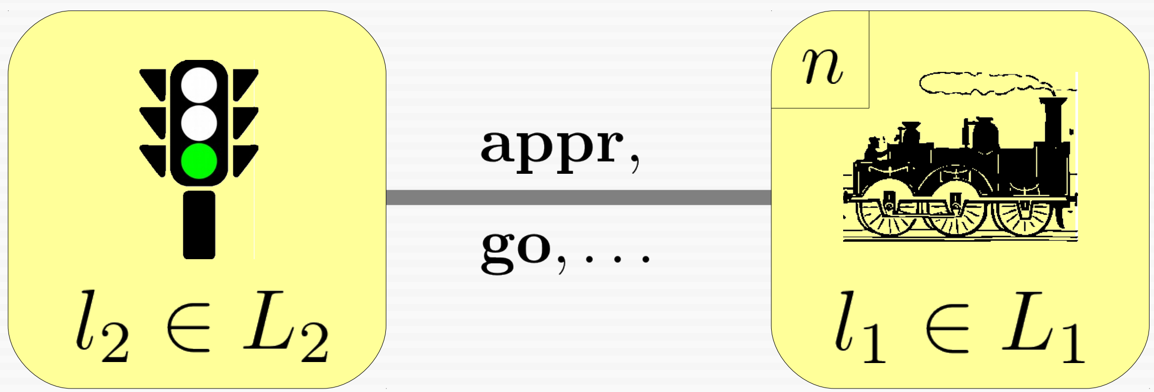
+ time elapse, synch., initiation, assertions

$$\forall n.\ Inv_1(g, n, \bar{l}_1[n]) \wedge Inv_2(g, l_2)$$

Parameterised systems

$g \in G$

$Inv_2(g, l_2)$

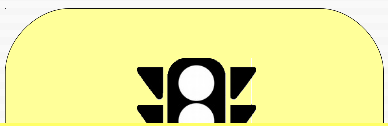


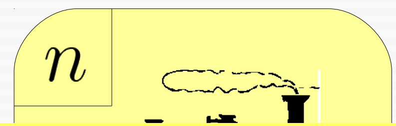
$Inv_1(g, n, l_1)$

$$\forall n.\ Inv_1(g, n, \bar{l}_1[n]) \wedge Inv_2(g, l_2)$$

Parameterised systems

$g \in G$





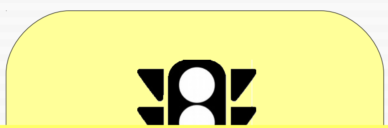
Final invariant schema:

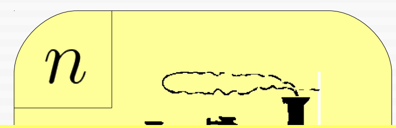
Need to analyse behaviour of ≥3 trains in combination to prove mutual exclusion:

$$\forall n_1, n_2, n_3.\ distinct(n_1, n_2, n_3) \rightarrow$$
$$Inv(g, l_2, n_1, \bar{l}_1[n_1], n_2, \bar{l}_1[n_2], n_3, \bar{l}_1[n_3])$$

Parameterised systems

$g \in G$





Final invariant schema:

Need to analyse behaviour of ≥3 trains in combination to prove mutual exclusion:

$$\forall n_1, n_2, n_3.\ \mathit{distinct}(n_1, n_2, n_3) \rightarrow \\ \mathit{Inv}(g, l_2, n_1, \bar{l}_1[n_1], n_2, \bar{l}_1[n_2], n_3, \bar{l}_1[n_3])$$

k-indexed invariant/ Ashcroft invariant

$g \in G$

I $)$

Nee
com

d invariant/
t invariant

Eldarica Web Interface

Eldarica is a model checker developed and maintained by EPFL and Uppsala University.

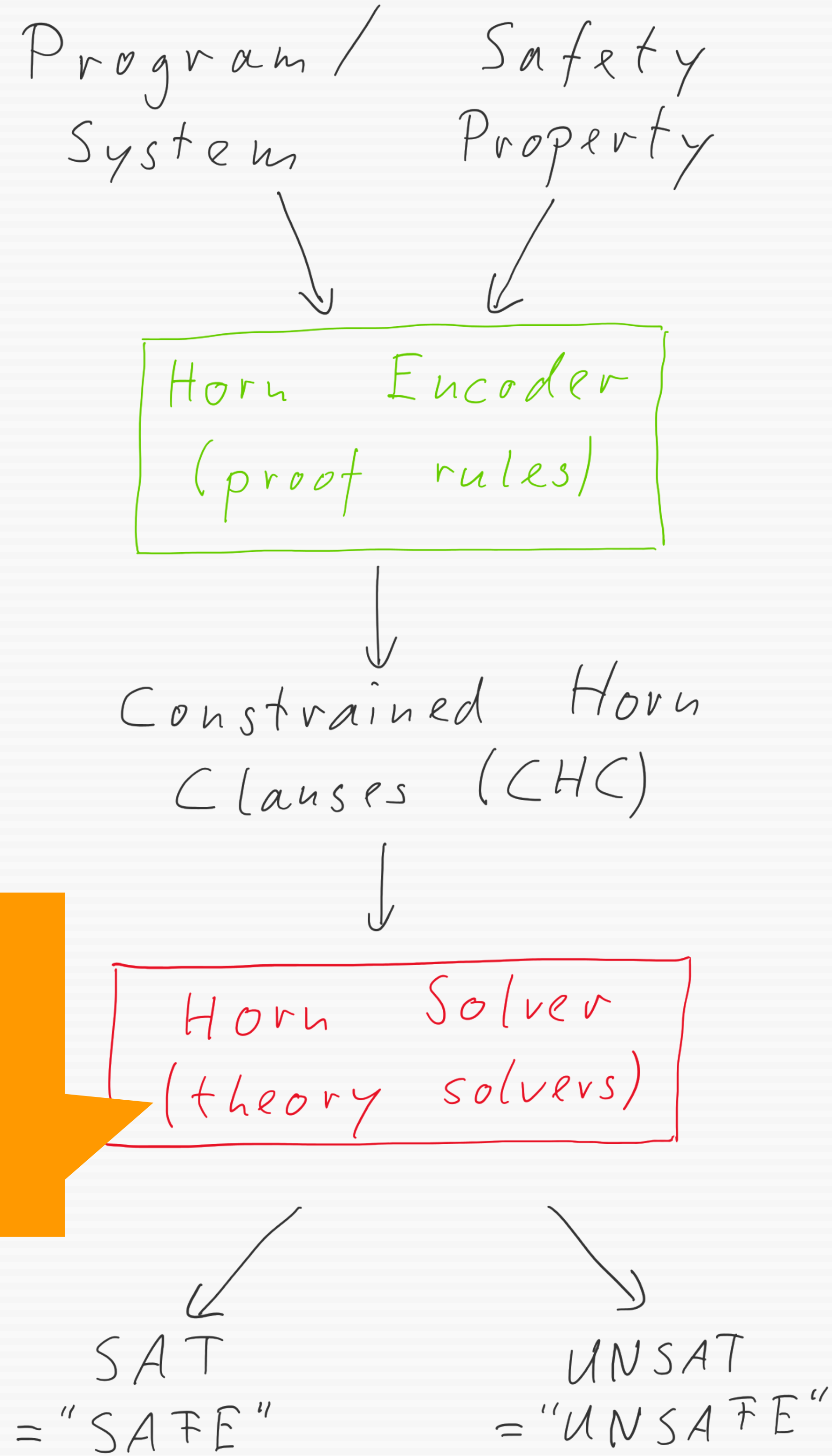
Load a predefined example: trains ... or enter a program:

```
/**
 * A simplified, but parameterised version of the train crossing
 * model from the FORTE'94 paper
 */

chan appr,     // signal approaching train
     stop,     // tell train to stop
     go,       // tell train to cross
     leave;    // signal leaving train

thread Controller {
  int n = 0;  // number of approaching and waiting trains
  clock y;

  while (1) {
    // bridge is free; let a waiting or approaching train pass

    if (n > 0) {
      chan_send(go);
    } else {
      chan_receive(appr); n++;
    }

    // a train is currently passing, stop other arriving trains

    while (1) {
      if (nondet()) {
        atomic { chan_receive(appr); n++; y = 0; }
        within(y <= 5) chan_send(stop);
      } else {
        chan_receive(leave); n--; break;
      }
    }
  }
}
```

Check | Permalink | C integer semantics: Mathematical (unbounded)

Verdict: *PROGRAM IS SAFE*

SAFE

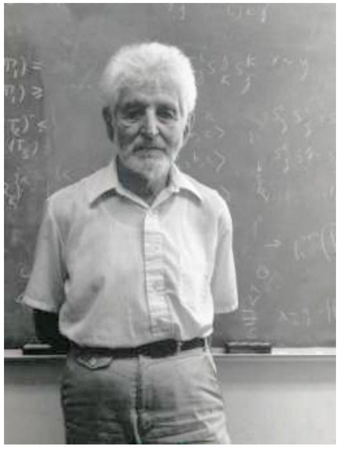


Linear Integers
Linear Rationals
Bit-vectors
Algebraic data-types
Arrays
etc.

Bitvector Interpolation

[3] Peter Backeman, PR, Aleksandar Zeljic. Bit-Vector Interpolation and Quantifier Elimination by Lazy Reduction. FMCAD 2018

Recap: Craig Interpolation

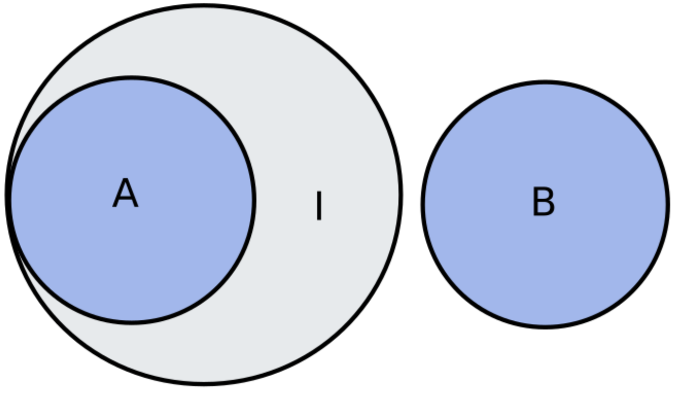


- Given an unsatisfiable formula $A \wedge B$

  a (reverse) *Interpolant* is a formula I s.t.:

  (a) $A \Rightarrow I$ and $B \Rightarrow \neg I$

  (b) I contains only non-logical symbols occuring both in A and B.



- Interpolant sequences/trees can be reduced to this

Fixed-Length Bit-Vectors

- Formalisation of machine arithmetic
- Domains: $x \in \mathbb{B}^n$
- Operators:
  - Arithmetic: bvadd, bvmul, . . .
  - Bit-Vector: concat, extract, shift, . . .
  - Bit-wise: bvand, bvor, . . .
- Efficient solvers (you-know-which)
  - But usually **no interpolation**

Interpolants for Bit-Vector Formulas?

- Solution 1: Bit-Blasting
  - Encode into propositional logic

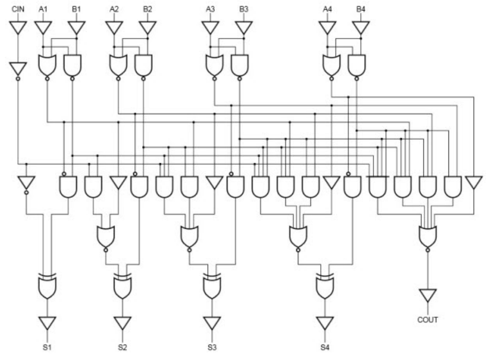
- Solution 2: Integer Encoding
  - Encode into integer arithmetic

Bit-Blasting

- Blast every bit-vector to bits:

  If $x \in \mathbb{B}^8$ then $x \leadsto x_0, x_1, \ldots, x_7$

- Model operations exactly



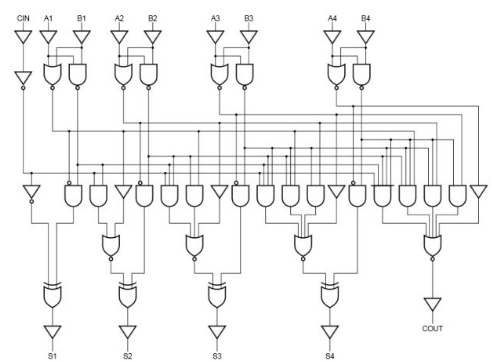
- Interpolation in SAT is well understood

Bit-Blasting

- Blast every bit-vector to bits:

  If $x \in \mathbb{B}^8$ then $x \rightsquigarrow x_0, x_1, \ldots, x_7$

- Model operations exactly



- Interpolation in SAT is well understood
- **But: this gives bit-level interpolants**

Integer Encoding

- If $x \in \mathbb{B}^8$ then $x \rightsquigarrow x'$ with $0 \leq x' < 2^8$
- Model overflows arithmetically, e.g.:

$$x = \text{bvadd}_8(y, 1) \rightsquigarrow$$
$$x' = y' + 1 - 2^8\sigma_1 \wedge 0 \leq x' < 2^8 \wedge 0 \leq \sigma_1 \leq 1$$

[4] A. Griggio, "Effective word-level interpolation for software verification," FMCAD 2011

Integer Encoding

- If $x \in \mathbb{B}^8$ then $x \rightsquigarrow x'$ with $0 \leq x' < 2^8$
- Model overflows arithmetically, e.g.:

$$x = \text{bvadd}_8(y, 1) \quad \rightsquigarrow$$
$$x' = y' + 1 - 2^8\sigma_1 \wedge 0 \leq x' < 2^8 \wedge 0 \leq \sigma_1 \leq 1$$

- **Hard LIA form., complicated interpolants**

[4] A. Griggio, "Effective word-level interpolation for software verification," FMCAD 2011

Integer Encoding

- If $x \in \mathbb{B}^8$ then $x \rightsquigarrow x'$ with $0 \leq x' < 2^8$
- Model overflows arithmetically, e.g.:

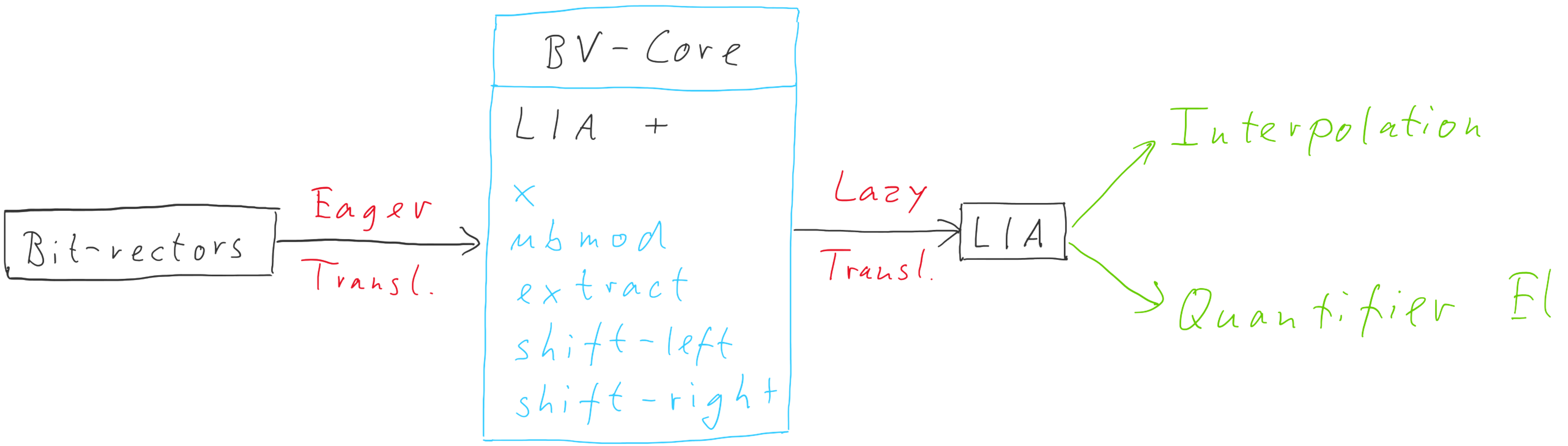
$$x = \text{bvadd}_8(y, 1) \quad \rightsquigarrow$$
$$x' = y' + 1 - 2^8\sigma_1 \wedge 0 \leq x' < 2^8 \wedge 0 \leq \sigma_1 \leq 1$$

- **Hard LIA form., complicated interpolants**
- **Many operations are difficult to encode**

[4] A. Griggio, "Effective word-level interpolation for software verification," FMCAD 2011

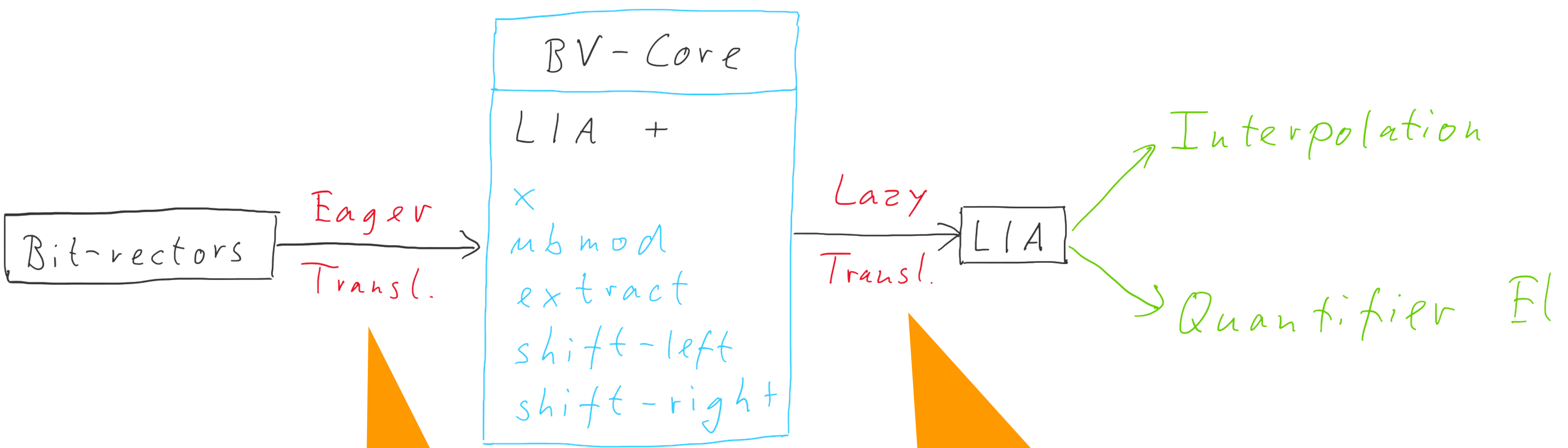
Lazy Reduction

- **Lazily** convert from a core language to integer arithmetic:



Lazy Reduction

- **Lazily** convert from a core language to integer arithmetic:



Pre-processing

During proof construction; can dynamically choose between multiple possible encodings

BV-Core Language

- LIA, extended with further predicates:

BV-Core Language

- LIA, extended with further predicates:

$$\text{ubmod}_w(s, r) \quad \Leftrightarrow \quad 0 \leq r < 2^w \wedge \left(r \equiv s \mod w\right)$$

BV-Core Language

- LIA, extended with further predicates:

$$\text{ubmod}_w(s, r) \quad \Leftrightarrow \quad 0 \leq r < 2^w \wedge \left(r \equiv s \mod w\right)$$

$\text{ubmod}_8(17, 17), \text{ubmod}_8(256, 0), \text{ubmod}_8(4039, 214)$

BV-Core Language

- LIA, extended with further predicates:

$$\begin{aligned} \mathsf{ubmod}_w(s,r) &\Leftrightarrow 0 \le r < 2^w \wedge \left(r \equiv s \mod w\right) \\ \times(s,t,r) &\Leftrightarrow s \cdot t = r \end{aligned}$$

BV-Core Language

- LIA, extended with further predicates:

$$\begin{aligned} \text{ubmod}_w(s, r) \quad &\Leftrightarrow \quad 0 \leq r < 2^w \wedge \left(r \equiv s \mod w\right) \\ \times(s, t, r) \quad &\Leftrightarrow \quad s \cdot t = r \end{aligned}$$

- Eager translation rules (applied on flat NNF):

BV-Core Language

- LIA, extended with further predicates:

$$\text{ubmod}_w(s, r) \quad \Leftrightarrow \quad 0 \leq r < 2^w \wedge \left(r \equiv s \mod w\right)$$
$$\times(s, t, r) \quad \Leftrightarrow \quad s \cdot t = r$$

- Eager translation rules (applied on flat NNF):

$$\text{bvule}_w(s, t) \;\rightsquigarrow\; s \leq t$$

BV-Core Language

- LIA, extended with further predicates:

$$\begin{aligned} \mathsf{ubmod}_w(s, r) &\Leftrightarrow 0 \le r < 2^w \wedge \big(r \equiv s \mod w\big) \\ \times(s, t, r) &\Leftrightarrow s \cdot t = r \end{aligned}$$

- Eager translation rules (applied on flat NNF):

$$\begin{aligned} \mathsf{bvule}_w(s, t) &\rightsquigarrow s \le t \\ \mathsf{bvadd}_w(s, t) = r &\rightsquigarrow \mathsf{ubmod}_w(s + t, r) \end{aligned}$$

BV-Core Language

- LIA, extended with further predicates:

$$\begin{aligned} \mathsf{ubmod}_w(s,r) &\Leftrightarrow 0 \le r < 2^w \wedge \big(r \equiv s \mod w\big) \\ \times(s,t,r) &\Leftrightarrow s \cdot t = r \end{aligned}$$

- Eager translation rules (applied on flat NNF):

$$\begin{aligned} \mathsf{bvule}_w(s,t) &\rightsquigarrow s \le t \\ \mathsf{bvadd}_w(s,t) = r &\rightsquigarrow \mathsf{ubmod}_w(s+t,r) \\ \mathsf{bvmul}_w(s,t) = r &\rightsquigarrow \exists x.(\times(s,t,x) \wedge \mathsf{ubmod}_w(x,r)) \end{aligned}$$

BV-Core Language

- LIA, extended with further predicates:

$$\text{ubmod}_w(s, r) \Leftrightarrow 0 \leq r < 2^w \wedge \left(r \equiv s \mod w\right)$$
$$\times(s, t, r) \Leftrightarrow s \cdot t = r$$

- Eager translation rules (applied on flat NNF):

$$\begin{array}{rcl}
\mathsf{bvadd}_w(s,t) = r & \rightarrow & ubmod_w(s+t, r) \\
\mathsf{bvmul}_w(s,t) = r & \rightarrow & \exists x.\ \left(\times(s,t,x) \wedge ubmod_w(x,r)\right) \\
\mathsf{bvsle}_w(s,t) & \rightarrow & \exists x, y.\ (sbmod_w(s,x) \wedge sbmod_w(t,y) \wedge x \leq y) \\
\neg\mathsf{bvsle}_w(s,t) & \rightarrow & \exists x, y.\ (sbmod_w(s,x) \wedge sbmod_w(t,y) \wedge x > y)
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{bvneg}_w(s) = r & \rightarrow & ubmod_w(-s, r) \\
\mathsf{ze}_{w+w'}(s) = r & \rightarrow & s = r \\
\mathsf{bvule}_w(s,t) & \rightarrow & s \leq t \\
\neg\mathsf{bvule}_w(s,t) & \rightarrow & s > t
\end{array}$$

$$\mathsf{bvudiv}_w(s,t) = r \rightarrow \left(t = 0 \wedge r = 2^w - 1\right) \vee \left(t \geq 1 \wedge \exists x. (\times(t,r,x) \wedge s - t < x \leq s)\right)$$

Example (taken from [4])

- BV-Formula:

$$A = \neg \text{bvule}_8(\text{bvadd}_8(y_4, 1), y_3) \wedge y_2 = \text{bvadd}_8(y_4, 1)$$
$$B = \text{bvule}_8(\text{bvadd}_8(y_2, 1), y_3) \wedge y_7 = 3 \wedge y_7 = \text{bvadd}_8(y_2, 1)$$

- Infix notation:

$$A = y_4 + 1 > y_3 \wedge y_2 = y4 + 1$$
$$B = y_2 + 1 \leq y_3 \wedge y_7 = 3 \wedge y_7 = y2 + 1$$

Example (taken from [4])

- BV-Formula:

$$A = \neg \mathrm{bvule}_8(\mathrm{bvadd}_8(y_4, 1), y_3) \wedge y_2 = \mathrm{bvadd}_8(y_4, 1)$$

$$B = \mathrm{bvule}_8(\mathrm{bvadd}_8(y_2, 1), y_3) \wedge y_7 = 3 \wedge y_7 = \mathrm{bvadd}_8(y_2, 1)$$

- BV-Core representation:

$$A_{\mathrm{core}} = \mathrm{ubmod}_8(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\mathrm{core}} = \mathrm{ubmod}_8(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

Example (taken from [4])

- BV-Formula:

$$A = \neg \mathsf{bvule}_8(\mathsf{bvadd}_8(y_4, 1), y_3) \wedge y_2 = \mathsf{bvadd}_8(y_4, 1)$$

$$B = \mathsf{bvule}_8(\mathsf{bvadd}_8(y_2, 1), y_3) \wedge y_7 = 3 \wedge y_7 = \mathsf{bvadd}_8(y_2, 1)$$

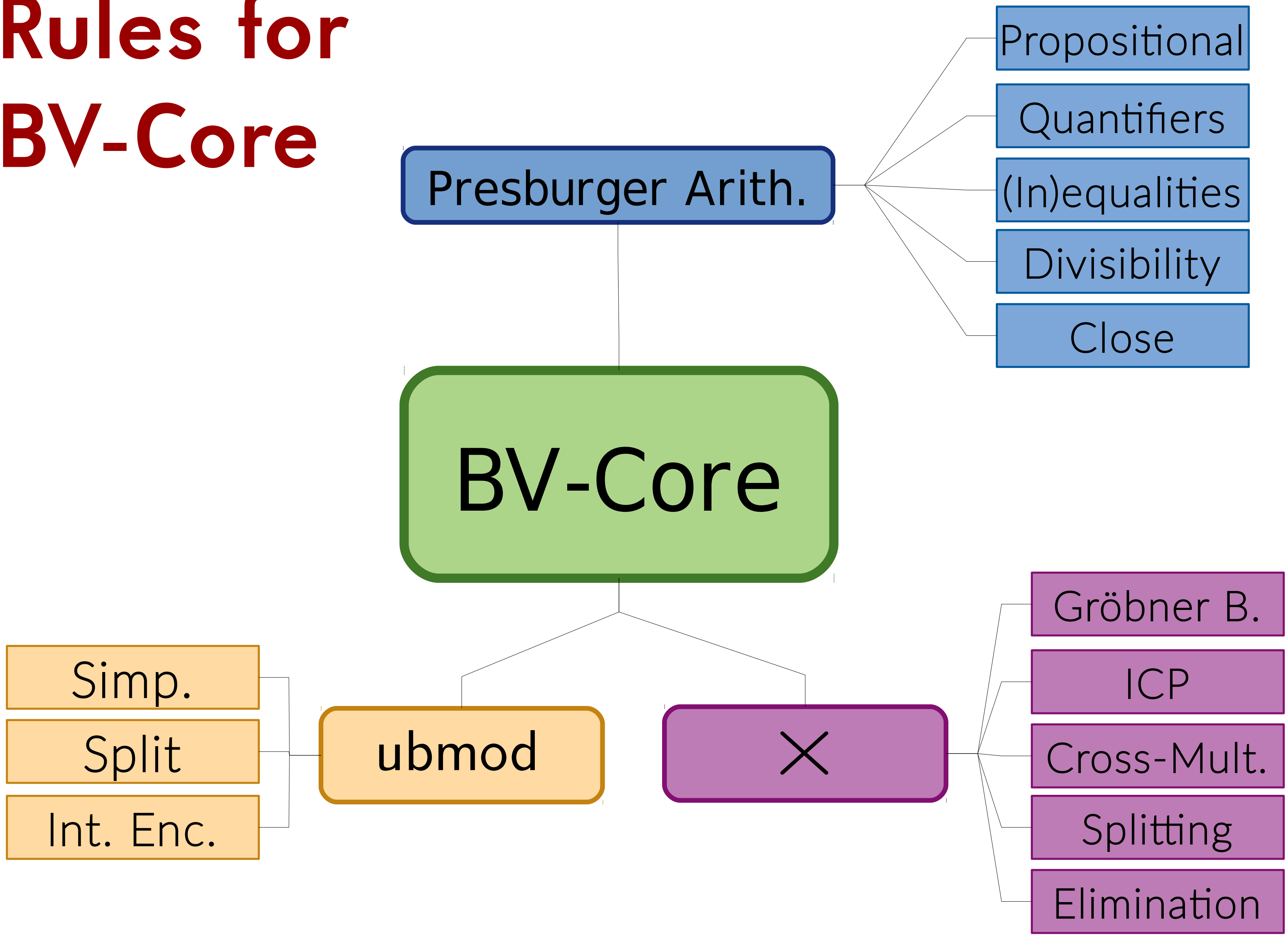
- BV-Core representation:

$$A_{\text{core}} = \mathsf{ubmod}_8(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

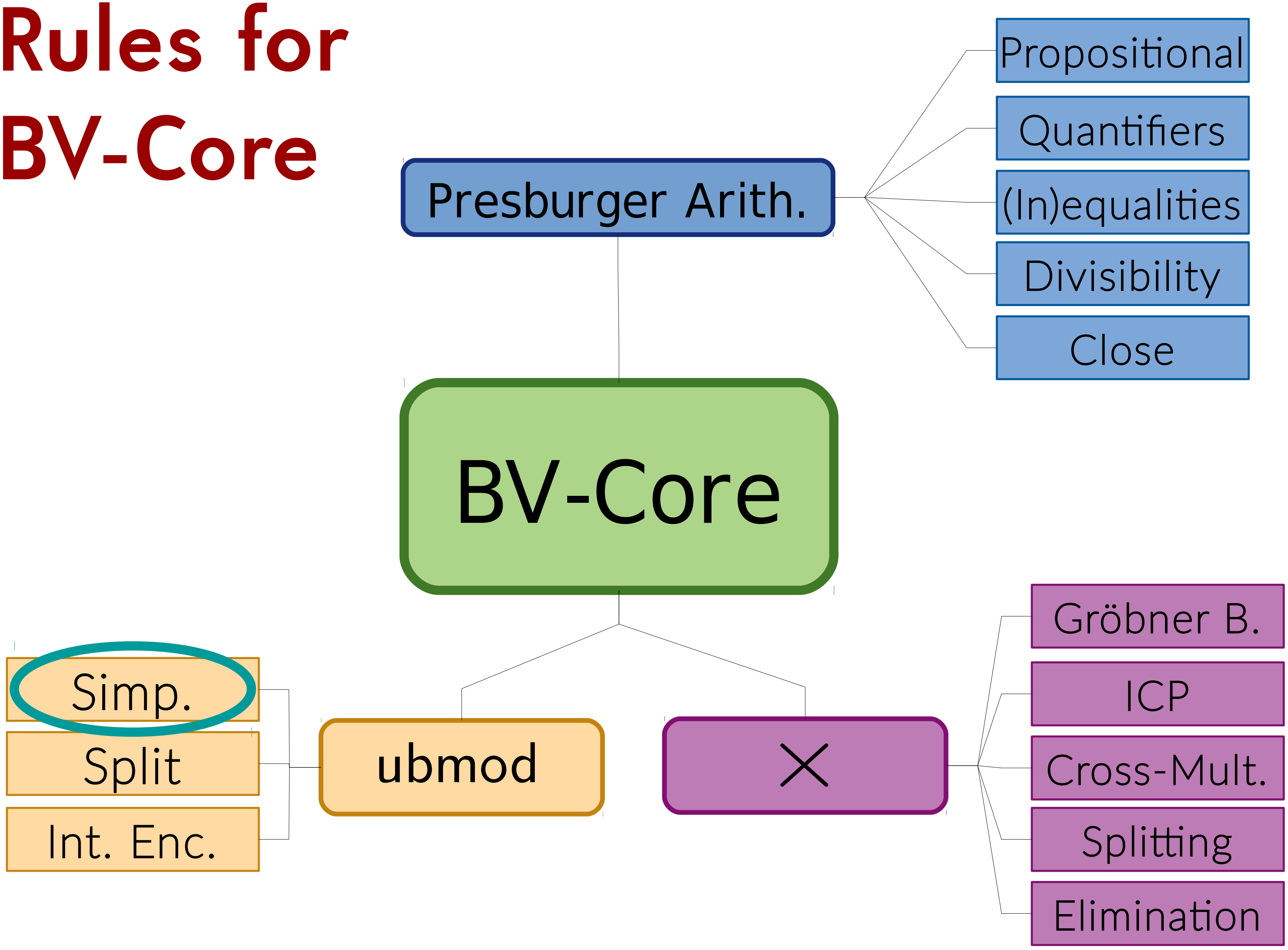
$$B_{\text{core}} = \mathsf{ubmod}_8(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

+ domain constraints (omitted)

Rules for BV-Core



Rules for BV-Core



BV-Core Simplification Rules

- Eliminate ubmod if only one branch possible:

$$\text{ubmod}_w(s, r) \quad \rightsquigarrow \quad s = r + 2^w k$$

for $\lfloor \frac{lbound(s)}{2^w} \rfloor = k = \lfloor \frac{ubound(s)}{2^w} \rfloor$

- Eliminate nested ubmod

BV-Core Simplification Rules

- Eliminate ubmod if only one branch possible:

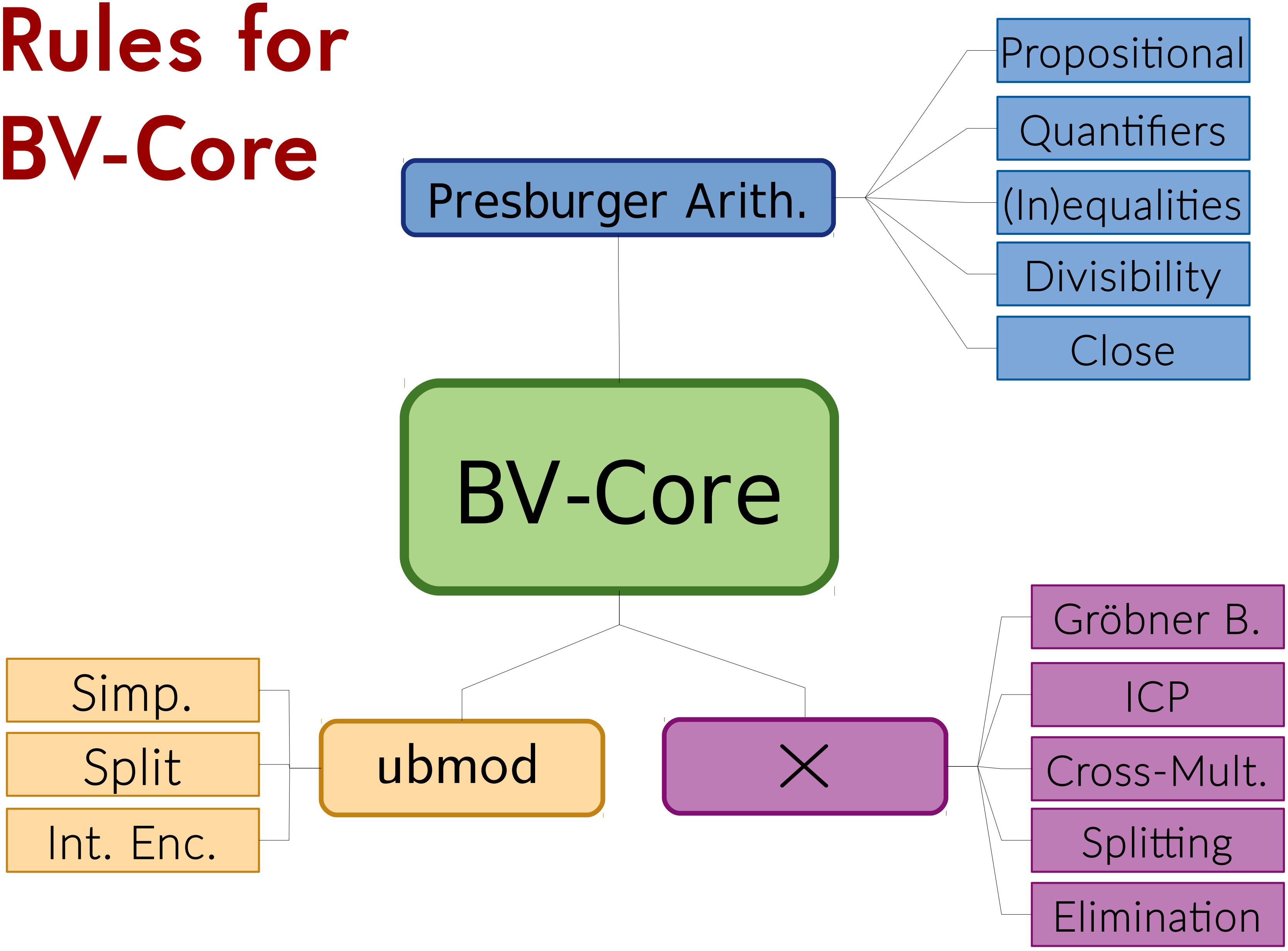
$$\text{ubmod}_w(s, r) \quad \rightsquigarrow \quad s = r + 2^w k$$

for $\lfloor \frac{lbound(s)}{2^w} \rfloor = k = \lfloor \frac{ubound(s)}{2^w} \rfloor$

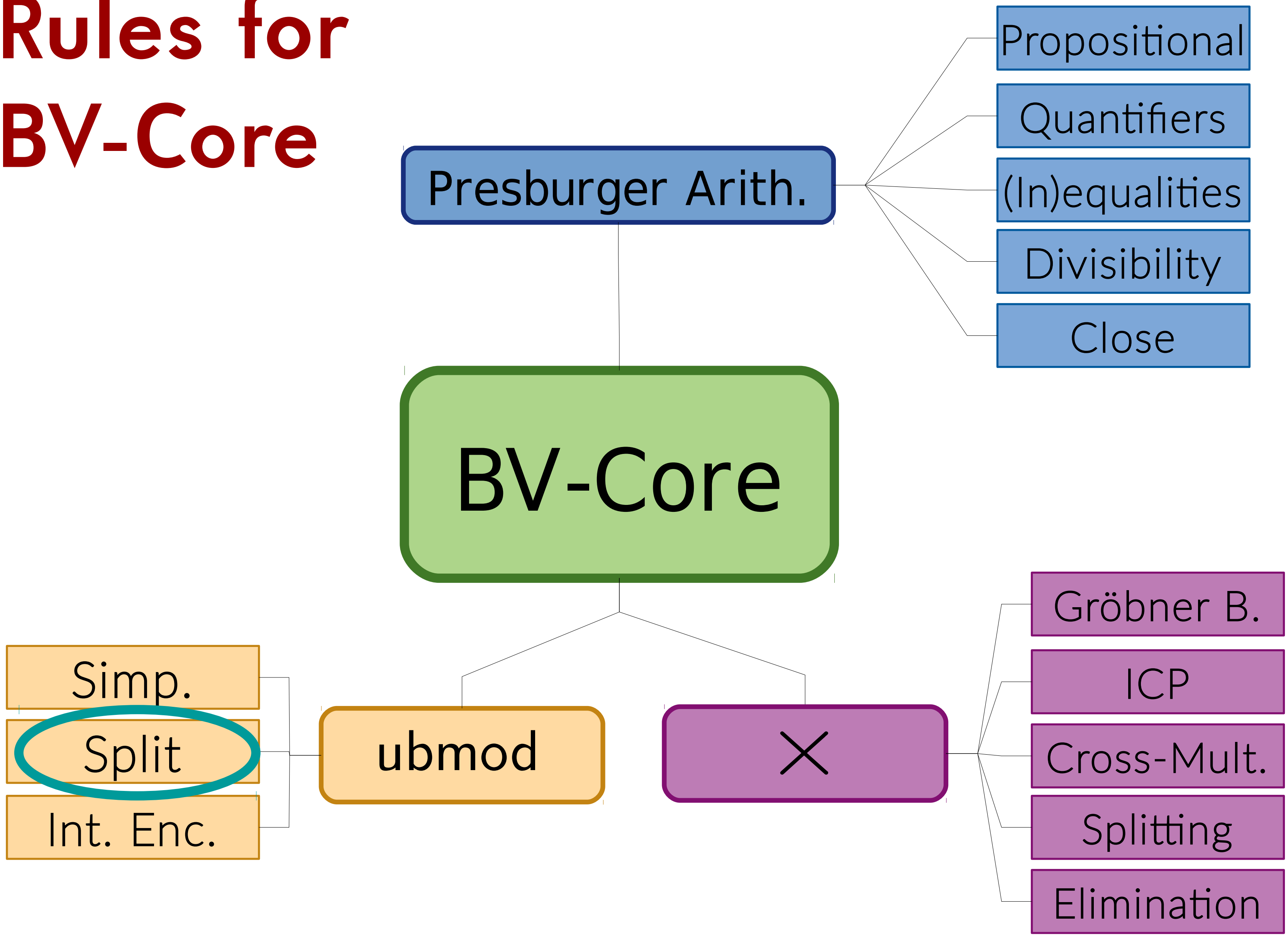
- Eliminate nested ubmod

- **Applied aggressively during proving**

Rules for BV-Core



Rules for BV-Core



Rule: BMOD-SPLIT

- Given tight bounds in ubmod_w consider cases explicitly:

$$\text{ubmod}_w(s, r) \quad \rightsquigarrow$$

$$0 \leq r < 2^w \wedge \bigvee_{i=l}^{u} s = r + 2^w i$$

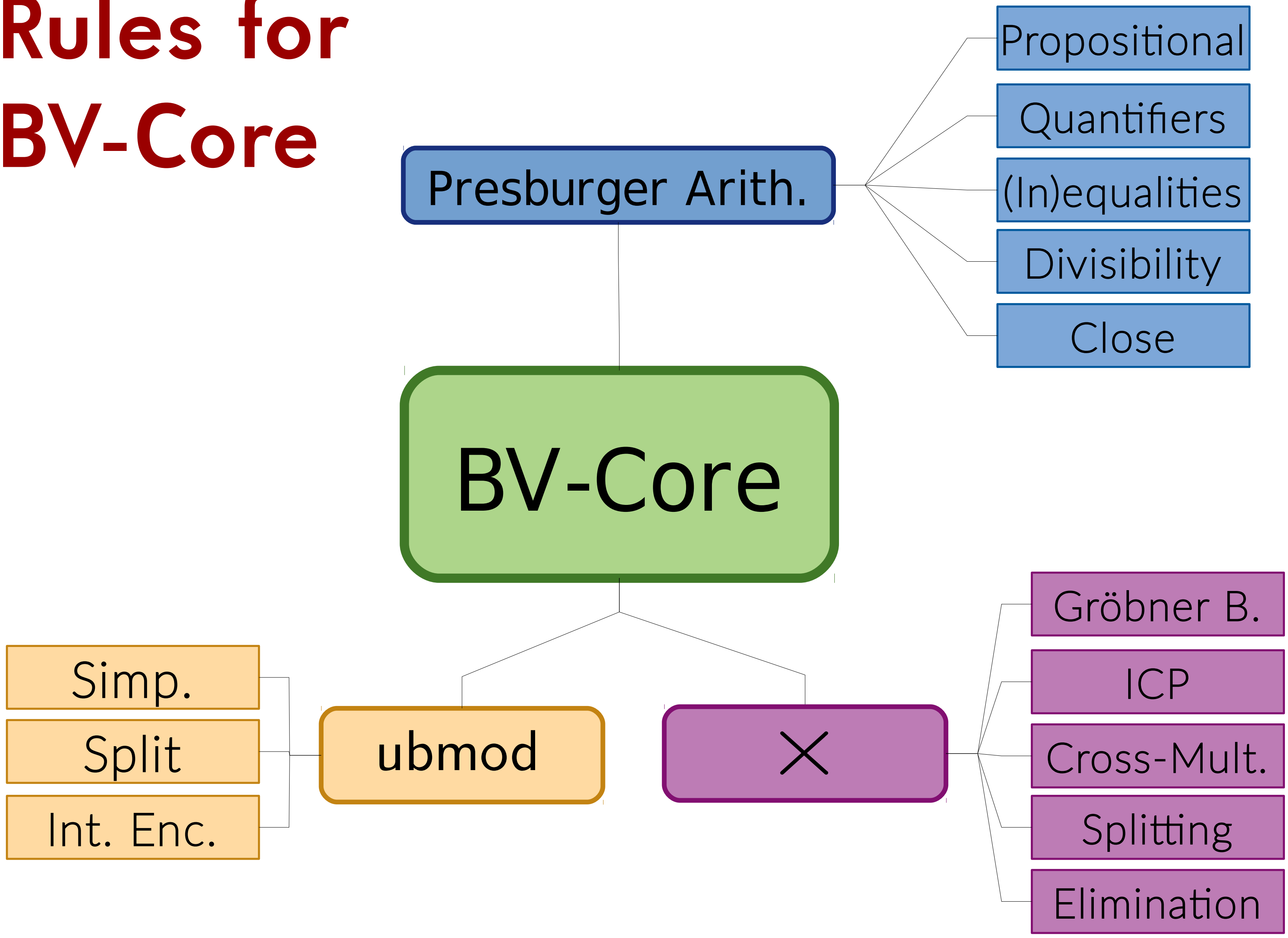
Rule: BMOD-SPLIT

- Given tight bounds in ubmod_w consider cases explicitly:

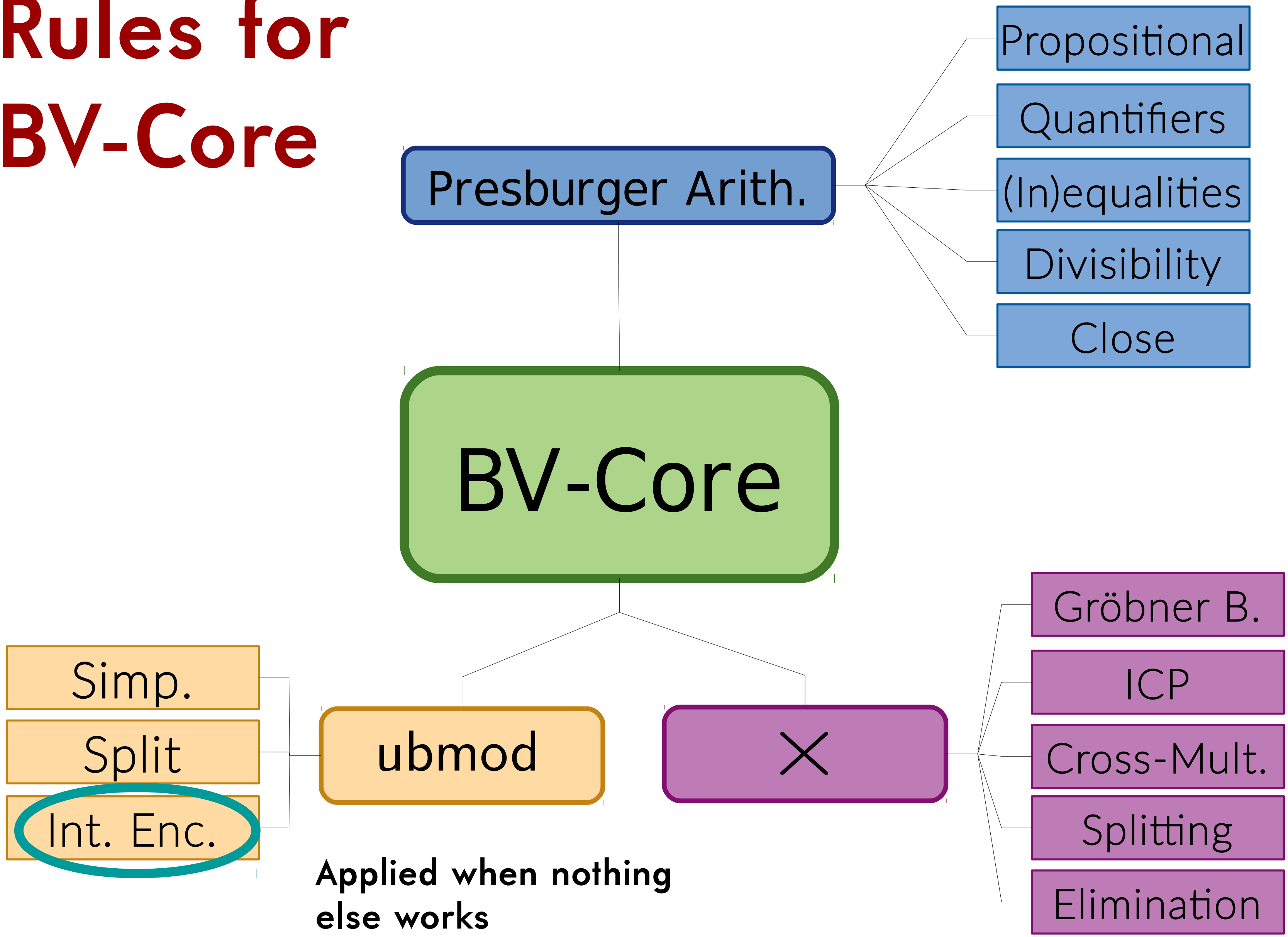
$$\text{ubmod}_w(s, r) \quad \rightsquigarrow$$
$$0 \leq r < 2^w \wedge \bigvee_{i=l}^{u} s = r + 2^w i$$

- **Applied when literal with small number of cases is found**

Rules for BV-Core



Rules for BV-Core



Example – Proof

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

Example – Proof

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

Example – Proof

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

$$A_{\text{core}}, B_{\text{core}} \vdash$$

Example – Proof

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash \qquad A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$$

$$A_{\text{core}}, B_{\text{core}} \vdash$$

Example – Proof

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

$$\ldots, y_2 + 1 = c_2, y_2 = 2 \vdash$$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash \qquad A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$$

$$A_{\text{core}}, B_{\text{core}} \vdash$$

Example – Proof

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

$$*$$

$$\ldots, y_2 + 1 = c_2, y_2 = 2 \vdash$$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash \qquad A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$$

$$A_{\text{core}}, B_{\text{core}} \vdash$$

Example – Proof

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

$$*$$

$$\ldots, y_2 + 1 = c_2, y_2 = 2 \vdash \qquad \ldots, y_2 + 1 = c_2, y_2 = 258 \vdash$$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash \qquad A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$$

$$A_{\text{core}}, B_{\text{core}} \vdash$$

Example – Interpolation

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

$$*$$

$$\ldots, y_2 + 1 = c_2, y_2 = 2 \vdash \qquad \ldots, y_2 + 1 = c_2, y_2 = 258 \vdash$$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash \qquad A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$$

$$A_{\text{core}}, B_{\text{core}} \vdash$$

Example – Interpolation

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

$$*$$

$$\ldots, y_2 + 1 = c_2, y_2 = 2 \vdash$$

$$\blacktriangleright\ y_2 < y_3$$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash$$

$$*$$

$$\ldots, y_2 + 1 = c_2, y_2 = 258 \vdash$$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$$

$$A_{\text{core}}, B_{\text{core}} \vdash$$

Example – Interpolation

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

$$*$$

$$\ldots, y_2 + 1 = c_2, y_2 = 2 \vdash$$

▶ $y_2 < y_3$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash$$

$$*$$

$$\ldots, y_2 + 1 = c_2, y_2 = 258 \vdash$$

▶ $y_2 < y_3$

$$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$$

$$A_{\text{core}}, B_{\text{core}} \vdash$$

Example – Interpolation

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$

$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$4 \leq y_2 + 1 \leq 256$

$*$

$\ldots, y_2 + 1 = c_2, y_2 = 2 \vdash$ $\blacktriangleright\ y_2 < y_3$

$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash$ $\blacktriangleright\ y_2 < y_3$

$*$

$\ldots, y_2 + 1 = c_2, y_2 = 258 \vdash$ $\blacktriangleright\ y_2 < y_3$

$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$ $\blacktriangleright\ y_2 < y_3$

$A_{\text{core}}, B_{\text{core}} \vdash$ $\blacktriangleright\ y_2 < y_3$

Example – Interpolation

$$A_{\text{core}} = ubmod_w(y_4 + 1, c_1) \wedge c_1 > y_3 \wedge y_2 = c_1$$
$$B_{\text{core}} = ubmod_w(y_2 + 1, c_2) \wedge c_2 \leq y_3 \wedge y_7 = 3 \wedge y_7 = c_2$$

$$4 \leq y_2 + 1 \leq 256$$

Final Interpolant: $y_2 < y_3$

Compare with [4]: $-255 \leq y_2 - y_3 + 256\lfloor -1 \frac{y_2}{256} \rfloor$

$\cdots$

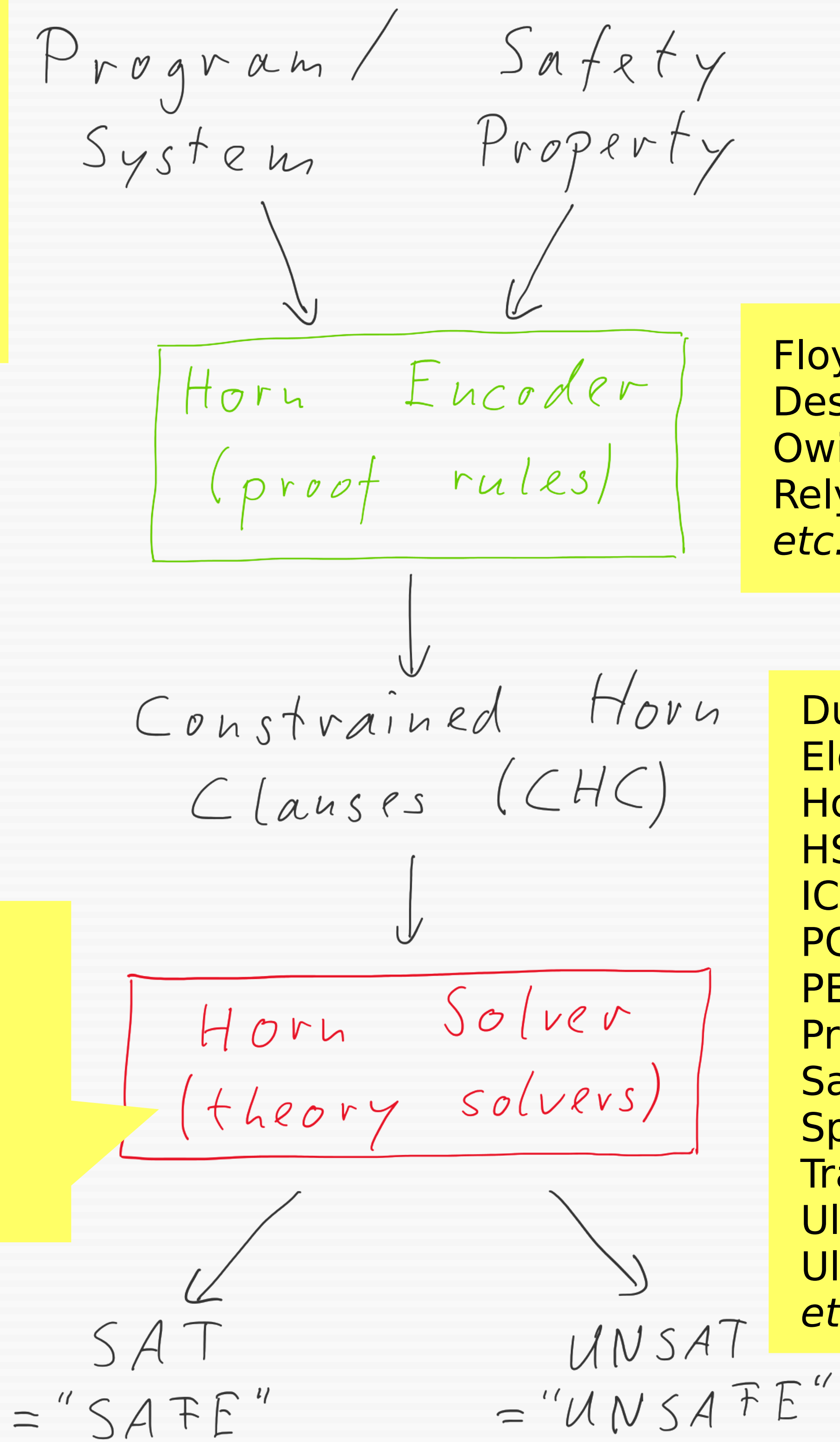
$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 \vdash$ ▶ $y_2 < y_3$

$A_{\text{core}}, B_{\text{core}}, y_2 + 1 = c_2 + 256 \vdash$ ▶ $y_2 < y_3$

$A_{\text{core}}, B_{\text{core}} \vdash$ ▶ $y_2 < y_3$

What is left?

- C
- Java
- Ada
- Rust
- Networks of TA
- BIP models
- *etc.*



- Floyd-Hoare
- Design by contract
- Owicki-Gries
- Rely Guarantee
- *etc.*

- Duality
- Eldarica(-abs)
- Hoice
- HSF
- IC3IA
- PCSat
- PECOS
- ProphIC3
- Sally
- Spacer
- TransfHORNer
- Ultimate TreeAutomizer
- Ultimate Unihorn
- *etc.*

- Linear Integers
- Linear Rationals
- Bit-vectors
- Algebraic data-types
- Arrays
- *etc.*

Proposed Extensions of CHCs

- Well-foundedness predicates
- Existential quantifiers in clause heads
- Universal quantifiers in clause bodies
- General fixed-point operators
- Optimisation with Horn clauses
- Non-Horn constraints
- *etc.*

Other Horn Encodings

- Owicki-Gries
- Rely-guarantee
- Various forms of thread communication
- Parameterised systems
- Timed systems
- Synchronous programs
- Equivalence/Regression verification
- Games
- Networks, SDN
- *etc.*

Convergence Heuristics

Local reasoning ⟷ **Global reasoning**

Convergence Heuristics

Local reasoning ⟷ **Global reasoning**

IC3: one counterexample at a time

CEGAR: one path at a time

Syntax-guided synthesis: all constraints at once

Convergence Heuristics

Local reasoning ⟷ **Global reasoning**

IC3: one counterexample at a time

CEGAR: one path at a time

Syntax-guided synthesis: all constraints at once

← Fast, might diverge

→ Guarantees convergence, less scalable

Convergence Heuristics

Local reasoning ⟷ **Global reasoning**

IC3: one counterexample at a time

CEGAR: one path at a time

Syntax-guided synthesis: all constraints at once

← Fast, might diverge

→ Guarantees convergence, less scalable

[5] Jérôme Leroux, PR, Pavle Subotic. Guiding Craig Interpolation with Domain-specific Abstractions. Acta Informatica 2016
[6] Hari Govind V.K., YuTing Chen, Sharon Shoham, Arie Gurfinkel: Global Guidance for Local Generalization in Model Checking. CAV 2020

Conclusions

Horn solvers and CHC ...

- provide highly optimised model checking engines
- enable experimentation with program logics and proof rules
- simplify implementation of verifiers

Questions?