Programming with Boolean Satisfaction

Michael Codish

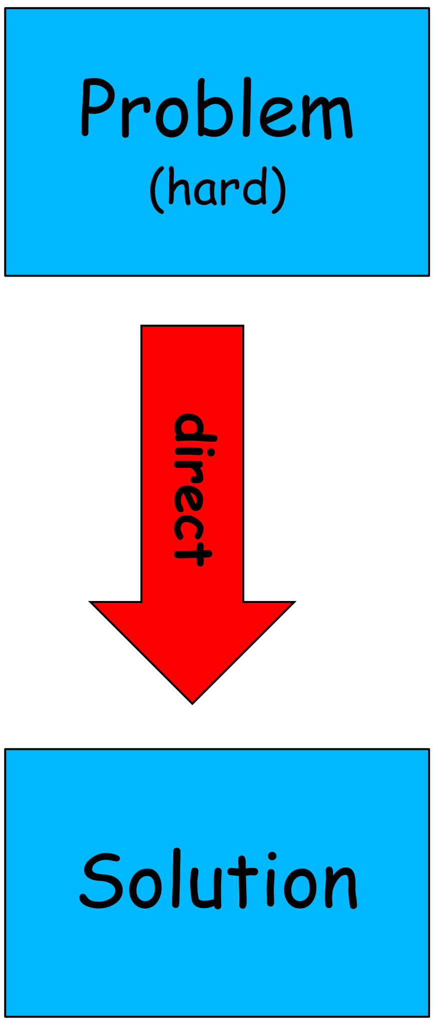
Department of Computer Science
Ben Gurion University
Beer-Sheva , Israel

Joint work with: Vitaly Lagoon,
Amit Metodi & Peter Stuckey

CP meets CAV - 2012

I. Its all about solving hard problems

Solving hard problems
(Programming)



Theory tells us

- Look for approximations
- Look for easier sub-classes

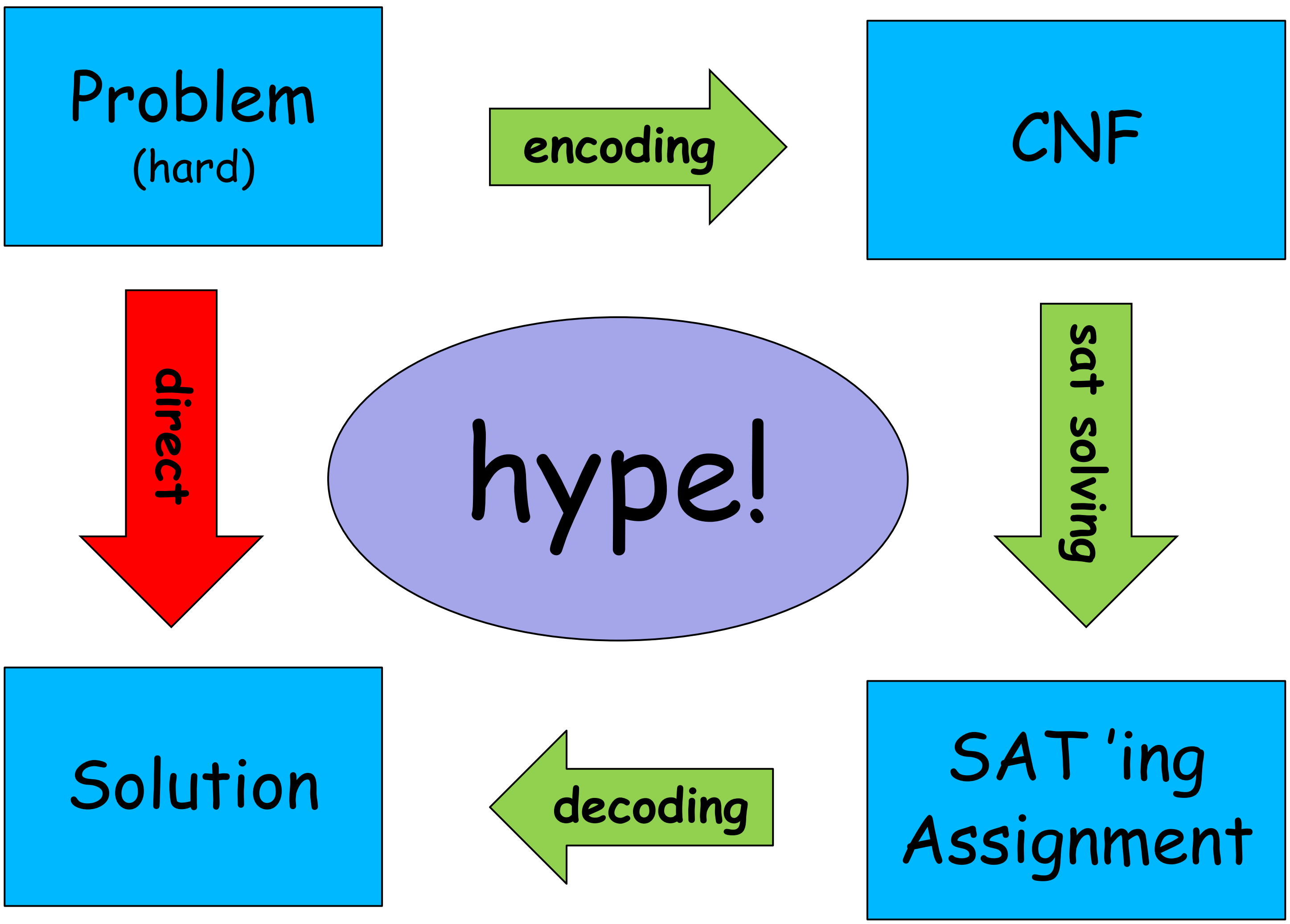
Practice tells us

- Apply heuristics
- Try to be clever

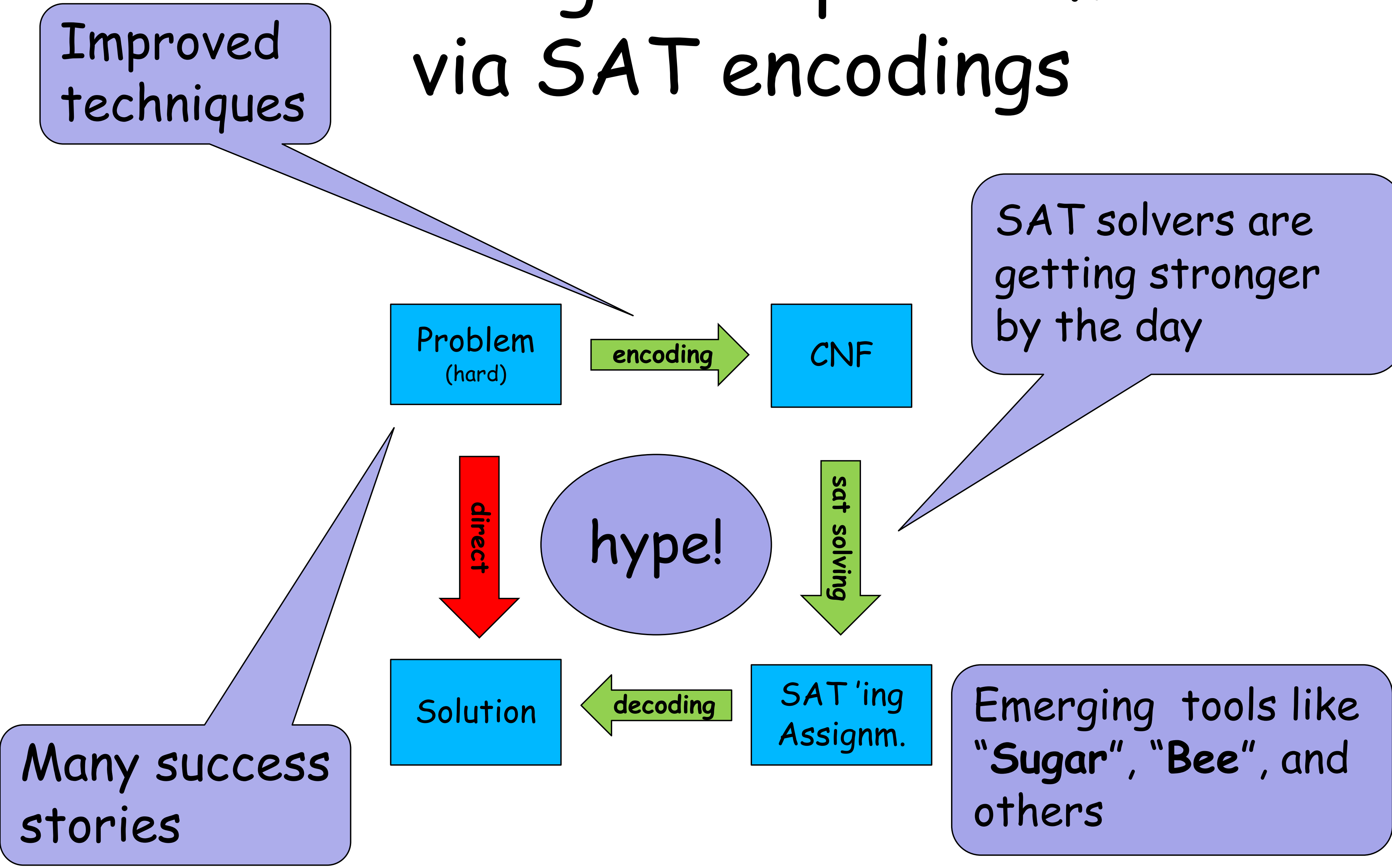
Theory also tells us

- It is all equivalent to SAT

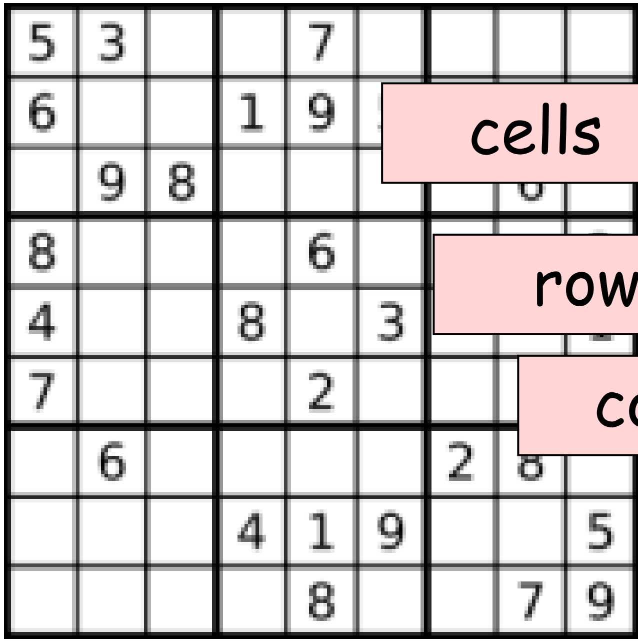
Solving hard problems via SAT encodings



Solving hard problems via SAT encodings



Example: encoding Sudoku



cells

rows

columns

boxes

"unit claus

$\bigwedge_{ij} one($ … $\wedge$

$\bigwedge_{ik}$ … $X_{i9k}) \wedge$

… $, X_{9jk}) \wedge$

… $\ldots)$

… uts

φ

At least

$$one(b_1, \ldots, b_n) \quad = \quad (b_1 \vee \cdots \vee b_n) \wedge$$

At most

$$\bigwedge_{i<j} (\bar{b}_i \vee \bar{b}_j)$$

X_{ijk} = cell (i,j) contains value k

Example: solving Sudoku

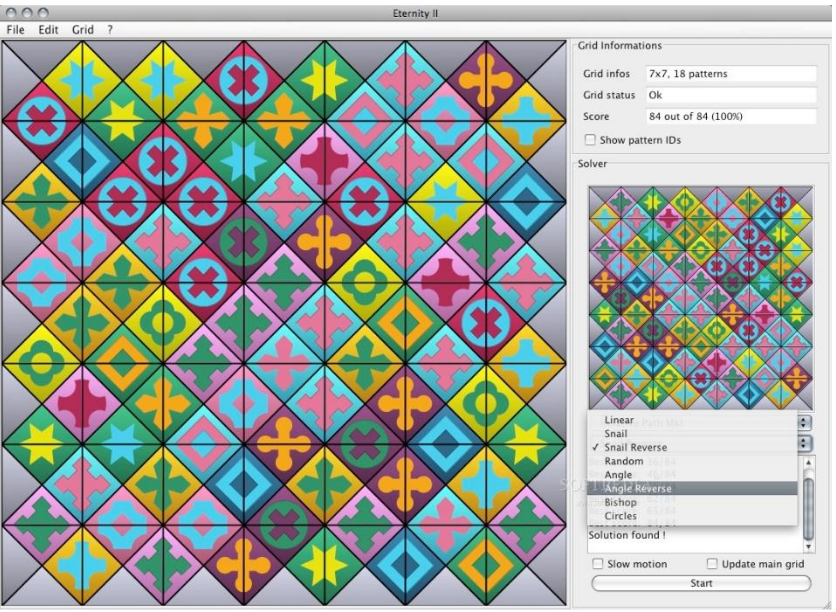
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

φ

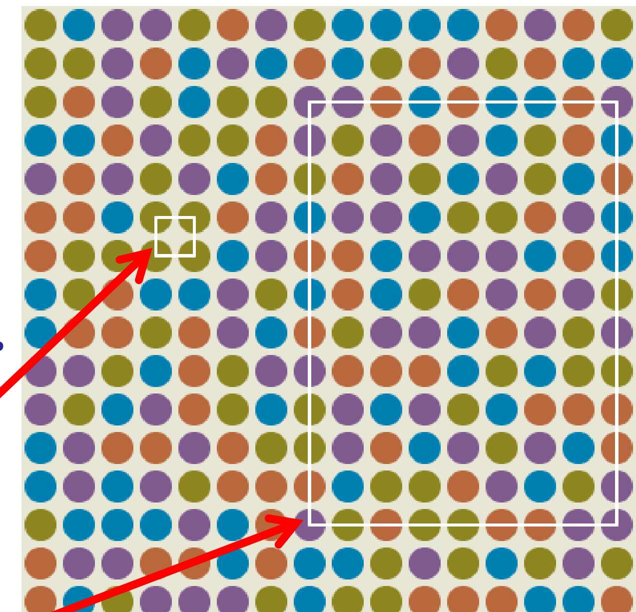
SAT Solver

solution

We Can Solve Also More Interesting Problems



But, there are also less interesting problems that we cannot solve

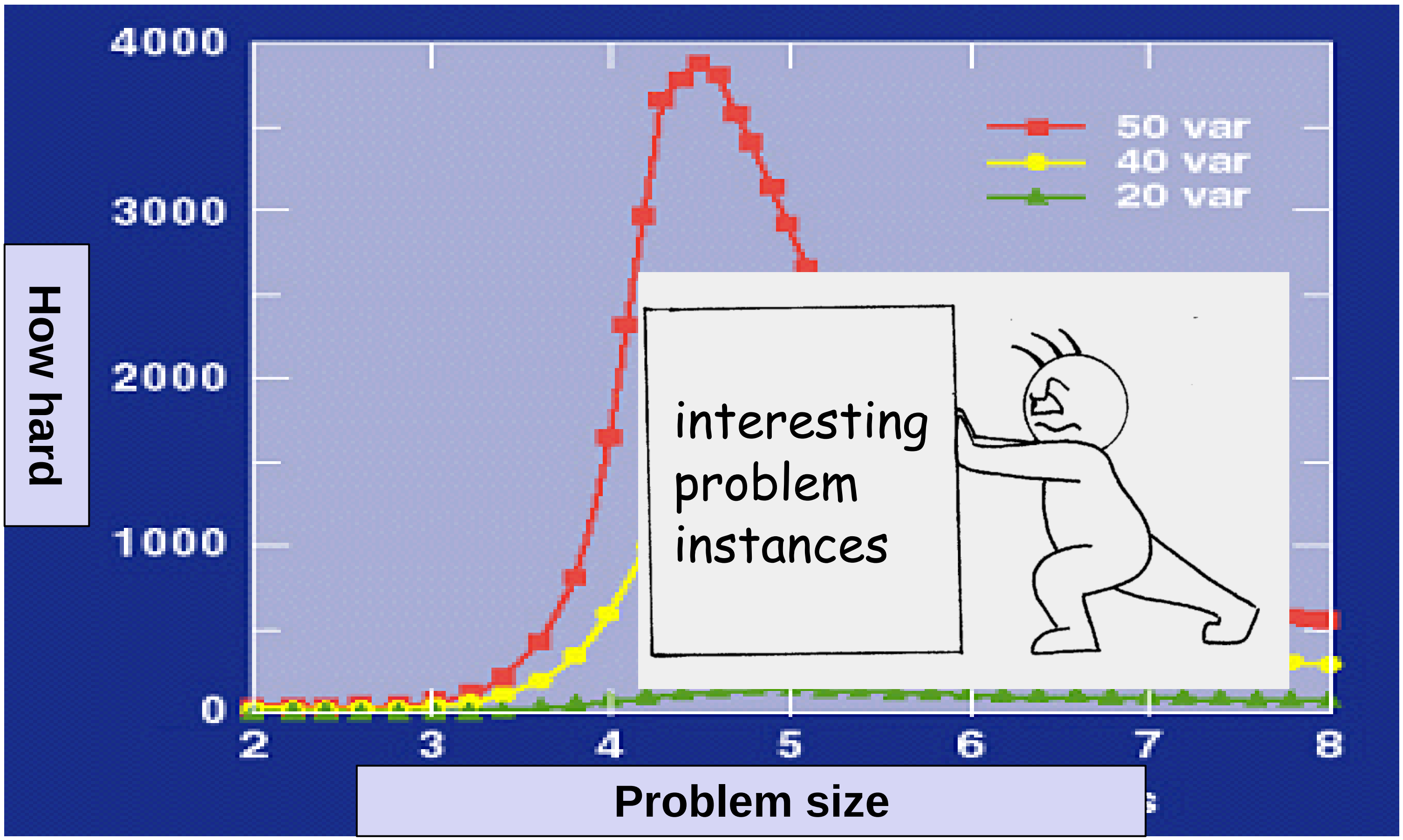


Eternity II: 2 million $ prize unclaimed

17 challenge: 17^2 $ prize unclaimed

3 months ago !
(Steinbach & Posthoff)

We will always have the phase transition



We seek better encodings so that our preferred problem instances will be solvable

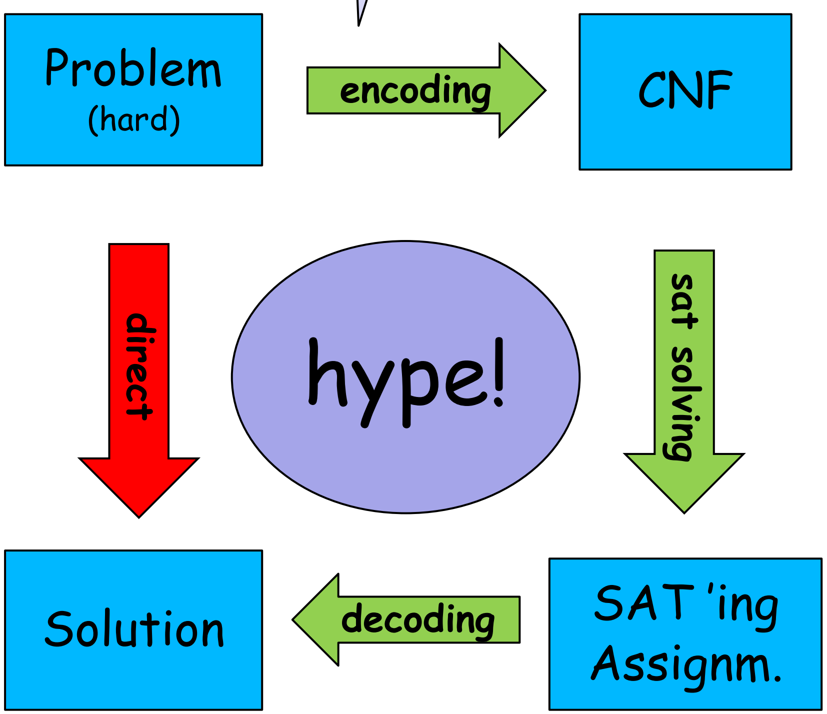
Solving hard problems
ia SAT encodings

tedious; often repetitive; generating millions of clauses

This was a great talk....

(a few years ago)

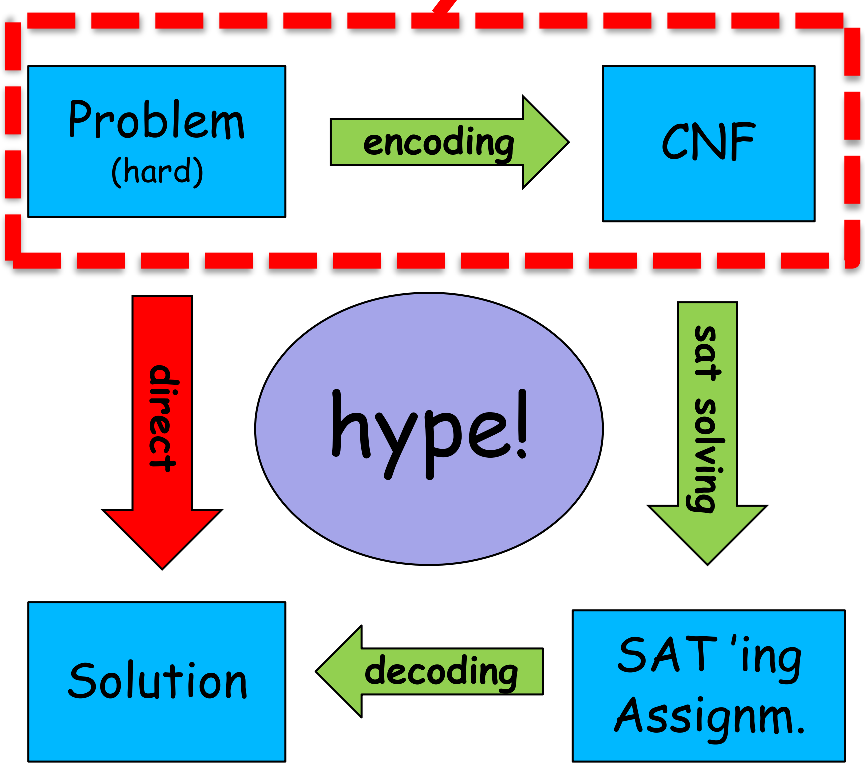
"and i have some conclusions"



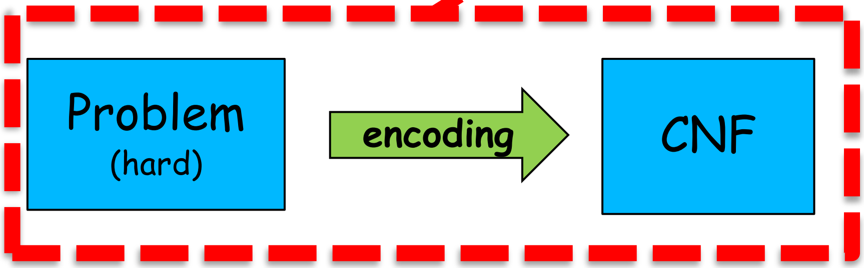
I have been doing this for the past few years

Me: primarily for applications of termination analysis

II. Programming with Boolean Satisfaction (CP meets CAV 2012)

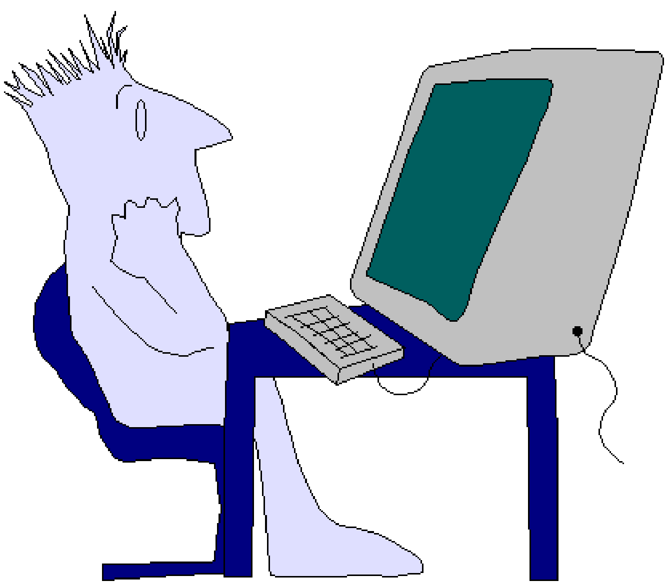


II. Programming with Boolean Satisfaction



Q. What makes a program(mer) work better?

- higher-level languages
- compilers & tools (p.e.) (optimizing)
- Data Structures / algorithms (what costs)
- hardware (understanding it)



Q. What makes a progr rk better?

SAT encoding

unit propagation /arc consistency

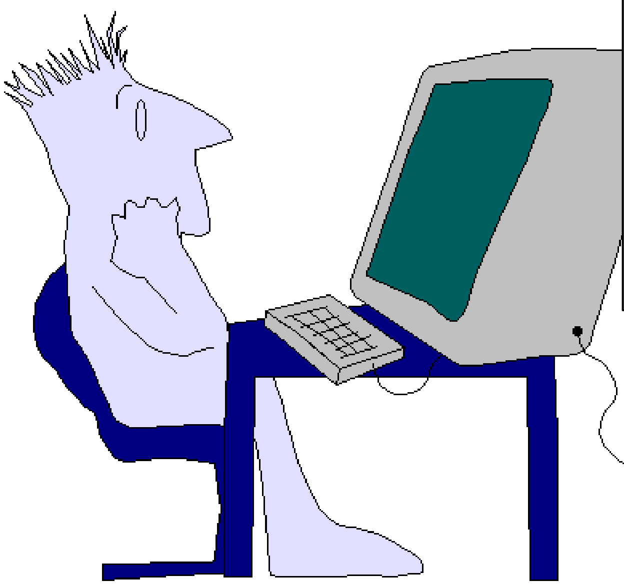
default value (1 or 0)

clause / variable ordering

Choice of SAT solver

hardware

SAT solver



Q. What makes a

progr rk better?

SAT encoding

higher-level languages

SAT encoding

compilers & tools (p.e.)

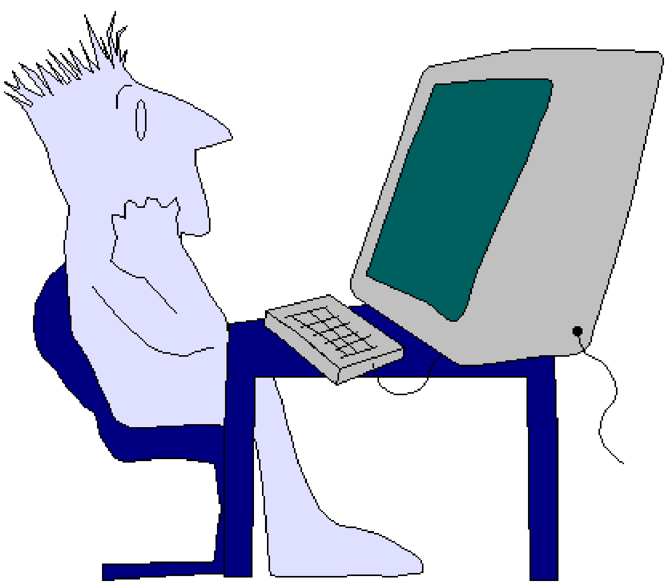
representa

& modeling

Data S lgorithms

hardware

SAT solver

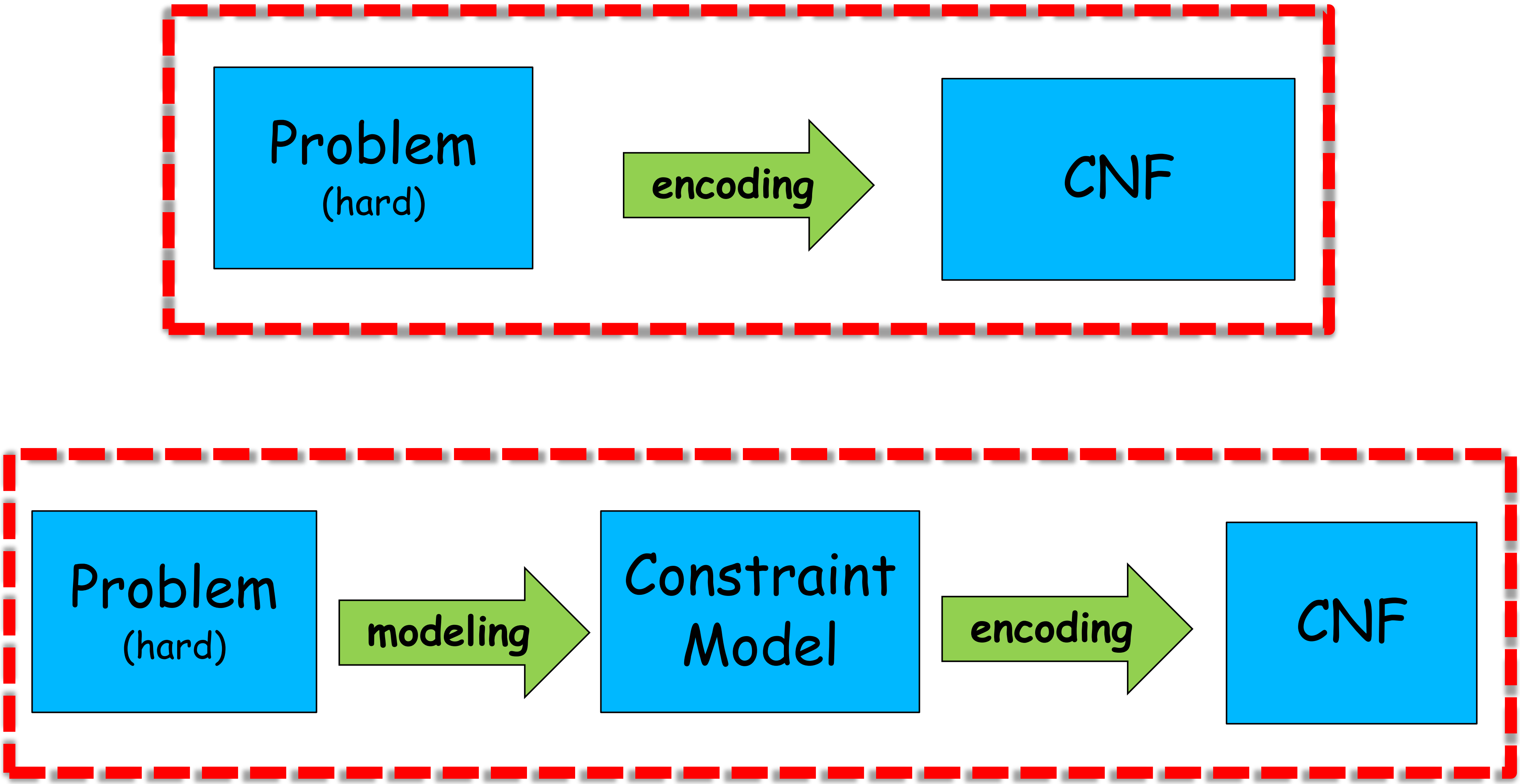


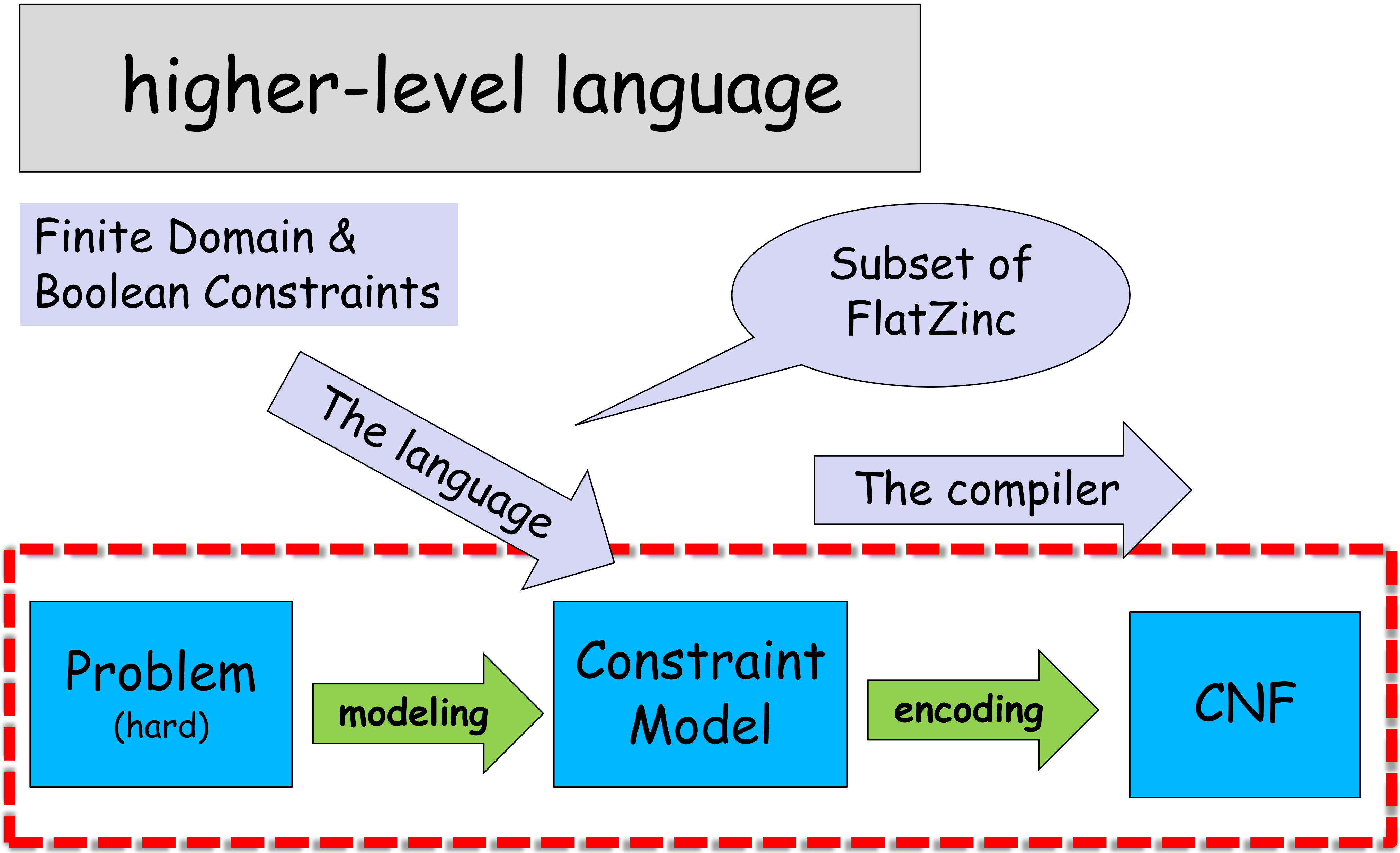
Outline

- Introduction:
  - Solving hard problems via SAT ✓
  - Focus on **programming** with SAT ✓
  - The need for higher-level languages ✓

- Higher low-level Language
- (the basics for) A Compiler to CNF
- Example: Model Based Diagnosis
- Representing Finite Domain Integers
- Example: Magic Labels
- Conclusion

higher-level language ?




higher-level language
Finite Domain & Boolean Constraints
Subset of FlatZinc
The language
The compiler
Problem
(hard)
modeling
Constraint Model
encoding
CNF

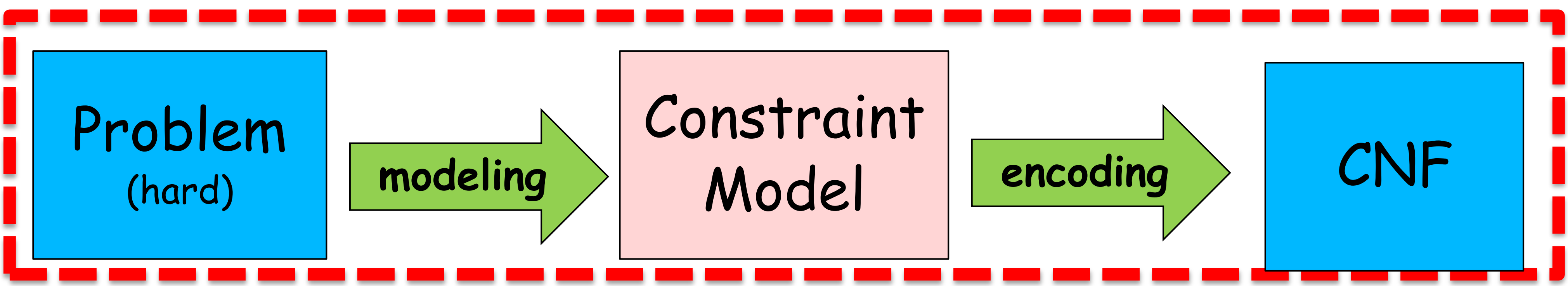
Example: encoding Sudoku

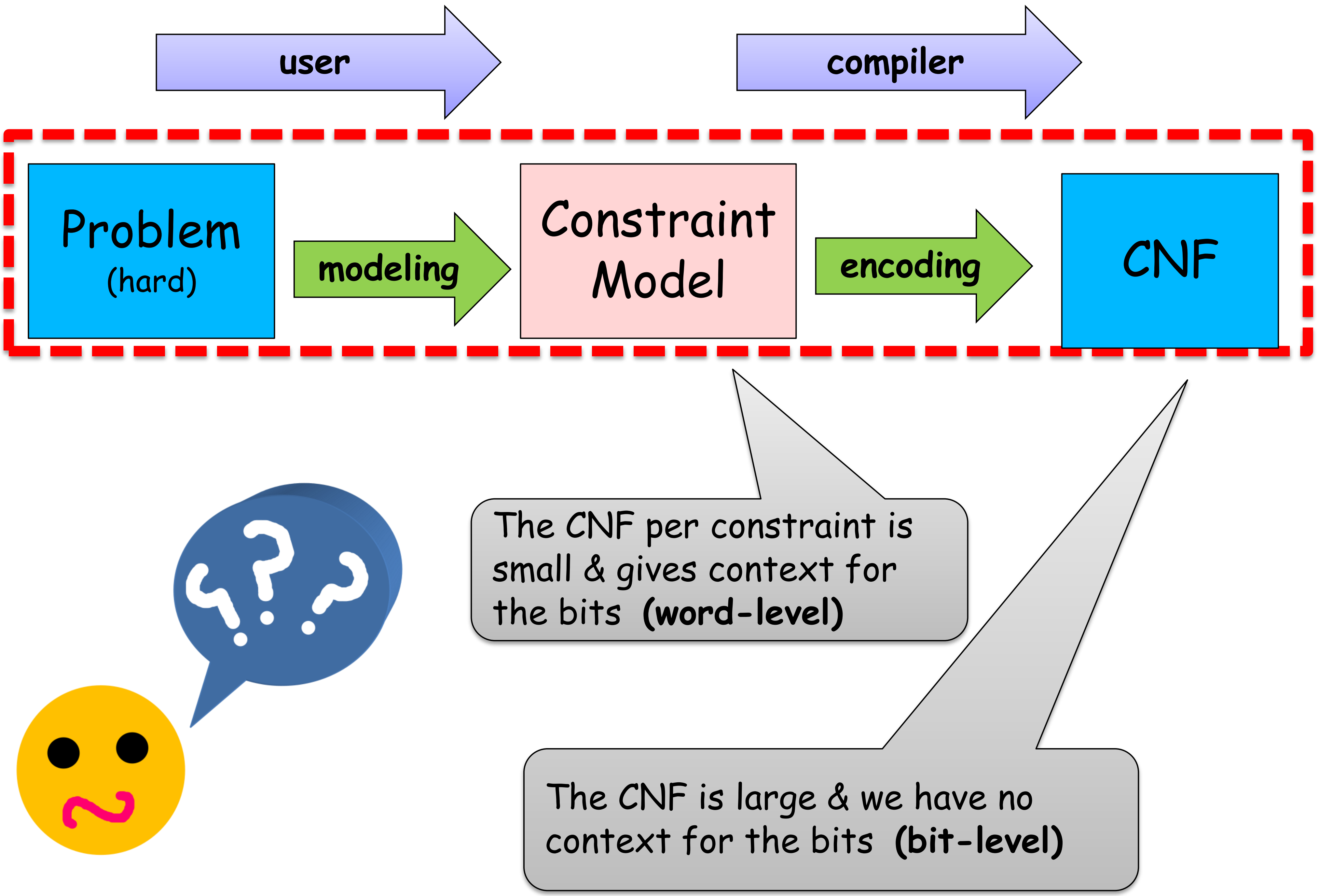
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

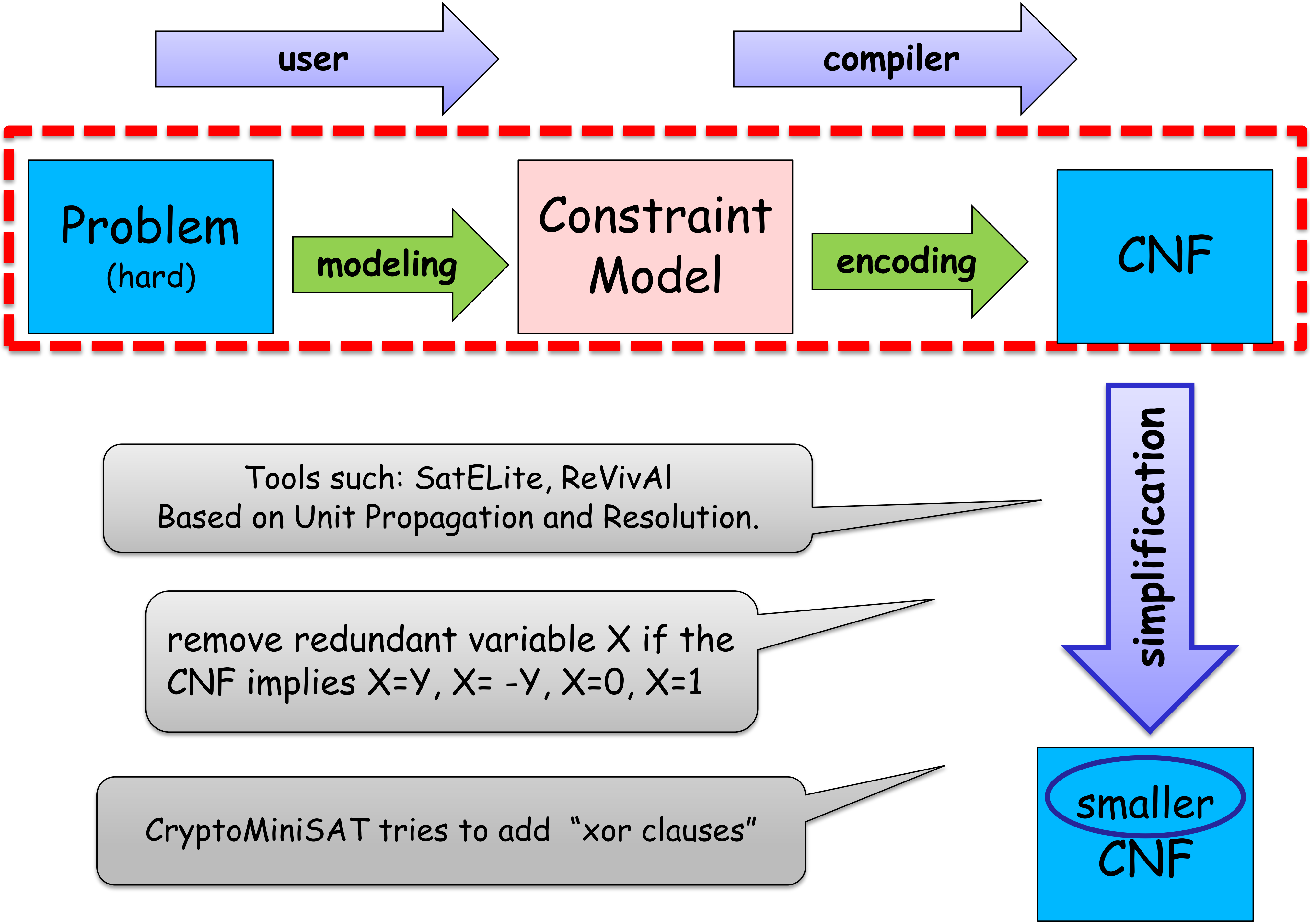
$$\mathtt{new\_int}(X_{1,1}, 1, 9)$$
$$\vdots$$
$$\mathtt{new\_int}(X_{9,9}, 1, 9)$$

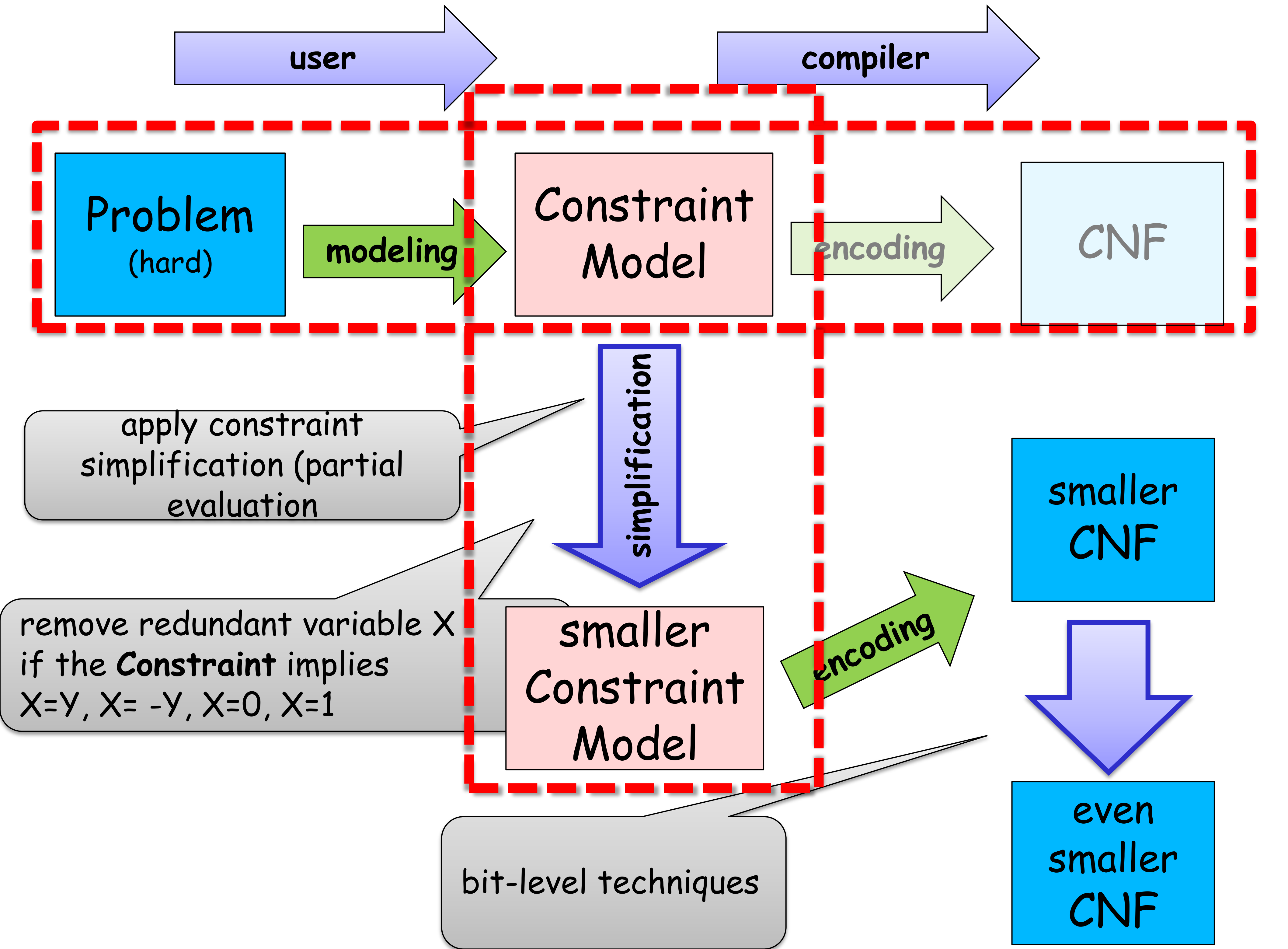
$$\mathtt{allDiff}([X_{1,1}, \ldots, X_{1,9}])$$
$$\vdots$$

$$\mathtt{int\_eq}(X_{1,1}, 5)$$
$$\mathtt{int\_eq}(X_{1,2}, 3)$$
$$\vdots$$




user
compiler
Problem
(hard)
modeling
Constraint Model
encoding
CNF
The CNF per constraint is small & gives context for the bits **(word-level)**
The CNF is large & we have no context for the bits **(bit-level)**

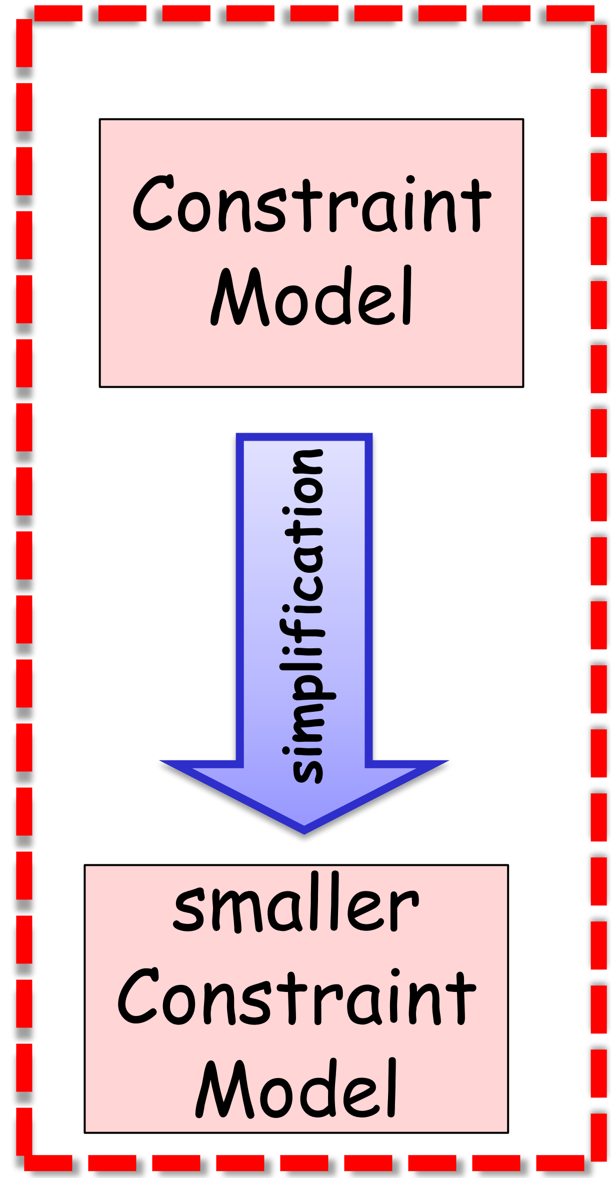

user
compiler
Problem
(hard)
modeling
Constraint Model
encoding
CNF
simplification
Tools such: SatELite, ReVivAl
Based on Unit Propagation and Resolution.
remove redundant variable X if the CNF implies X=Y, X= -Y, X=0, X=1
CryptoMiniSAT tries to add "xor clauses"
smaller
CNF


user
compiler
Problem
(hard)
modeling
Constraint Model
encoding
CNF
simplification
apply constraint simplification (partial evaluation
remove redundant variable X if the **Constraint** implies X=Y, X= -Y, X=0, X=1
smaller Constraint Model
encoding
smaller CNF
even smaller CNF
bit-level techniques

Equi-propagation is the process of inferring equations implied by a "few" constraints.

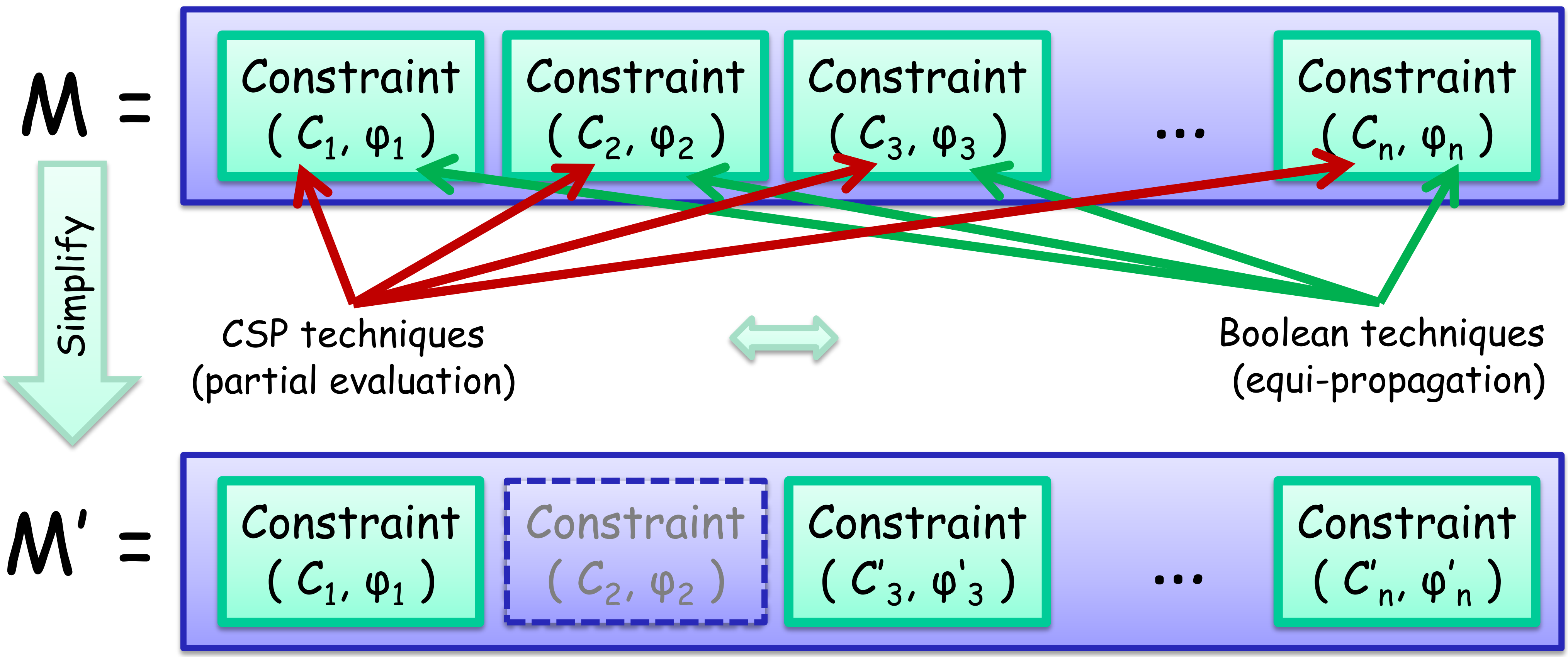
of the form X=L where L is a constant or a literal: X=Y, X= -Y, X=0, X=1

such x can be removed from **all** constraints.



Implemented: complete / adhoc equi-propagation

constraint simplification is word-level (looking at the bits)



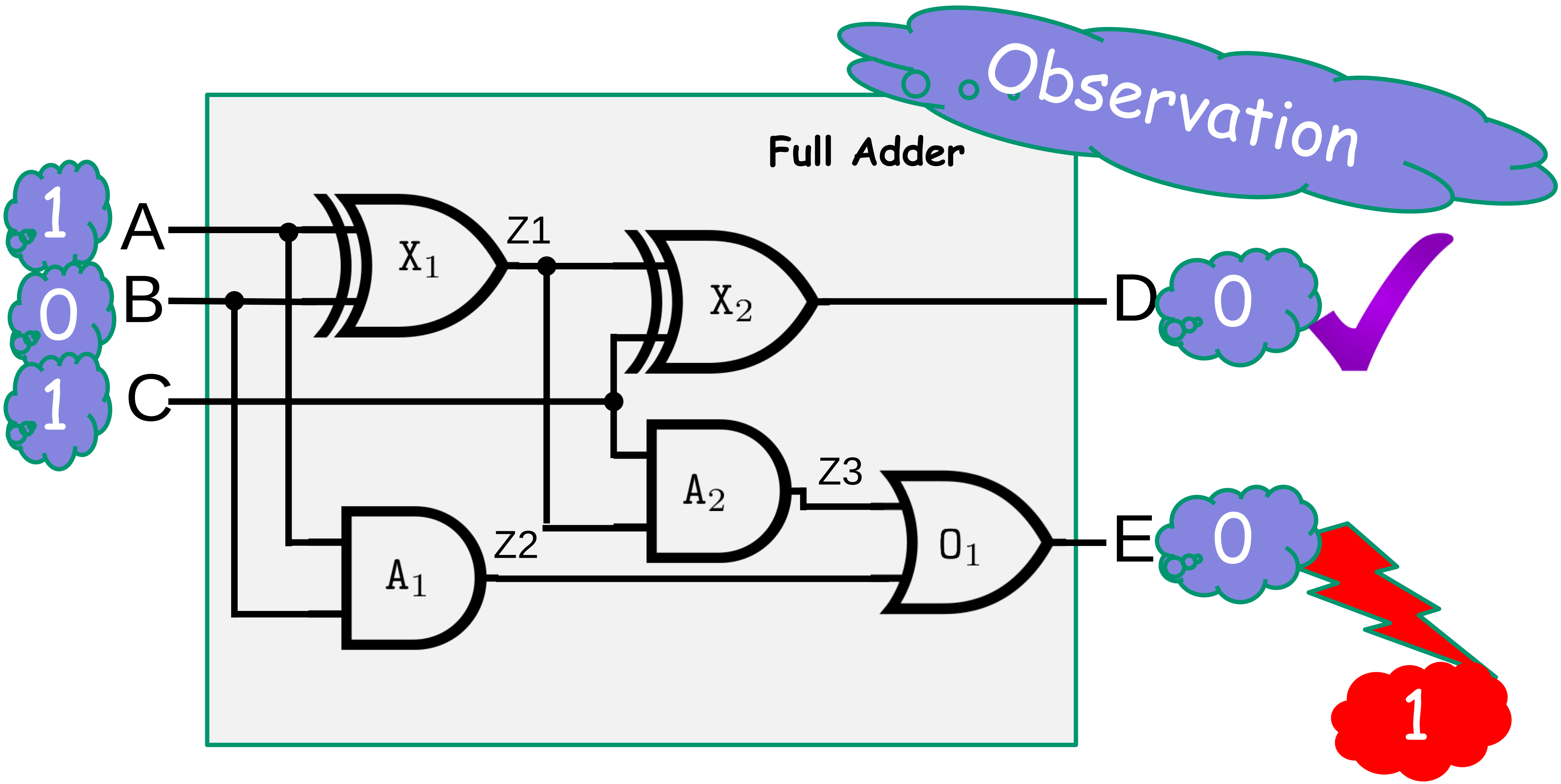
Equi-propagation for Optimized SAT Encoding;

Amit Metodi, Michael Codish, Vitaly Lagoon and Peter Stuckey; CP 2011

Outline

- Introduction:
  - Solving hard problems via SAT ✓
  - Focus on **programming** with SAT ✓
  - The need for higher-level languages ✓
- Higher low-level Language ✓
- (the basics for) A Compiler to CNF ✓
- Example: Model Based Diagnosis
- Representing Finite Domain Integers
- Example: Magic Labels
- Conclusion

Example: Model Based Diagnosis

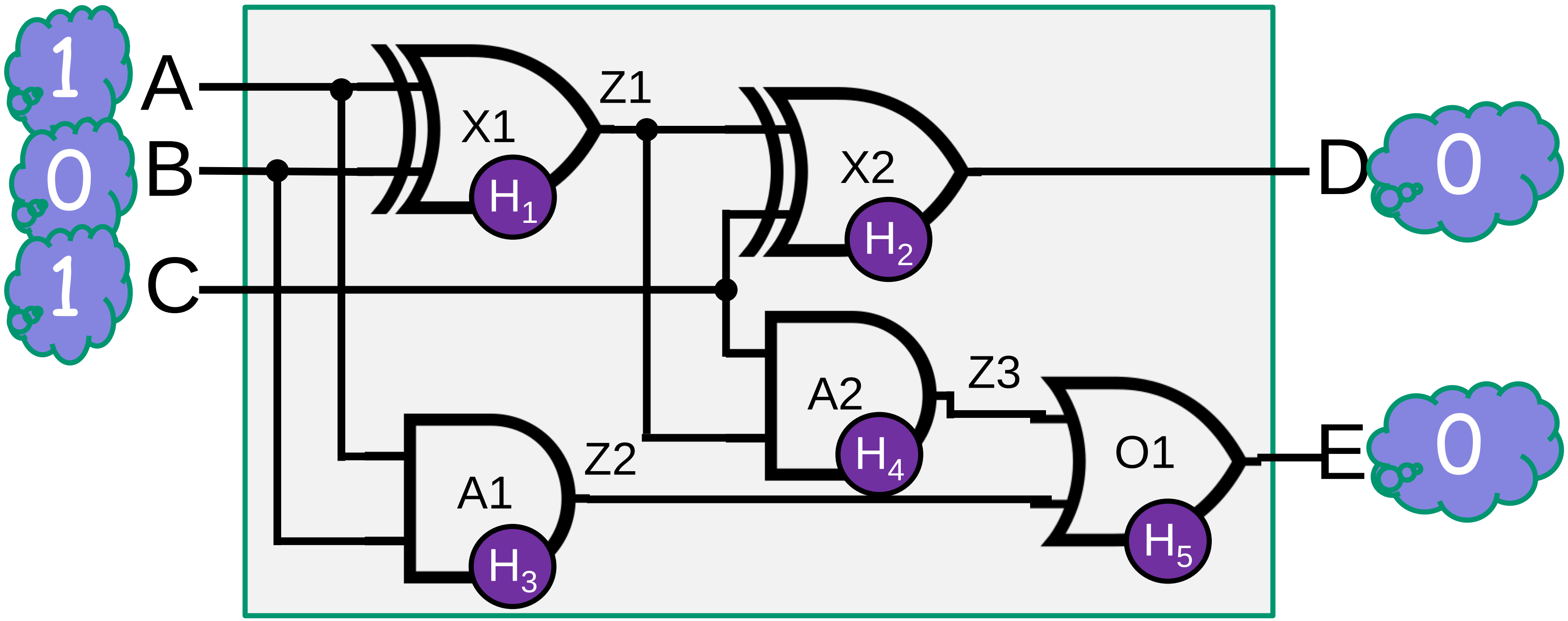


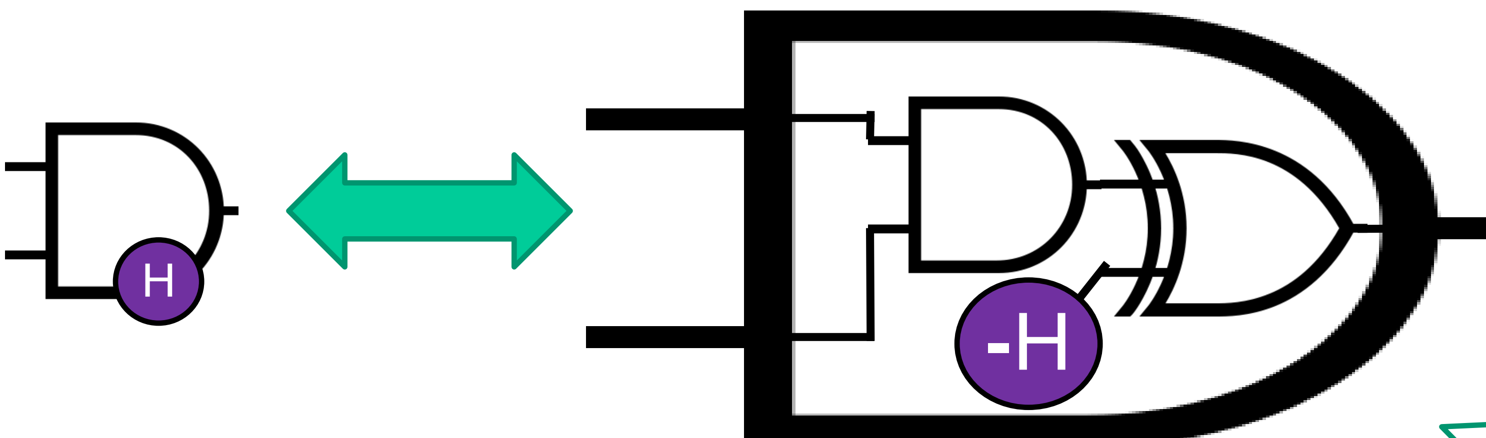
Diagnoses:

$$\{X_1, X_2\},\ \{O_1\},\ \{A_2\},\ \{A_1, O_1\}, \ldots$$

min-cardinality

Modeling MBD: introduce health variables

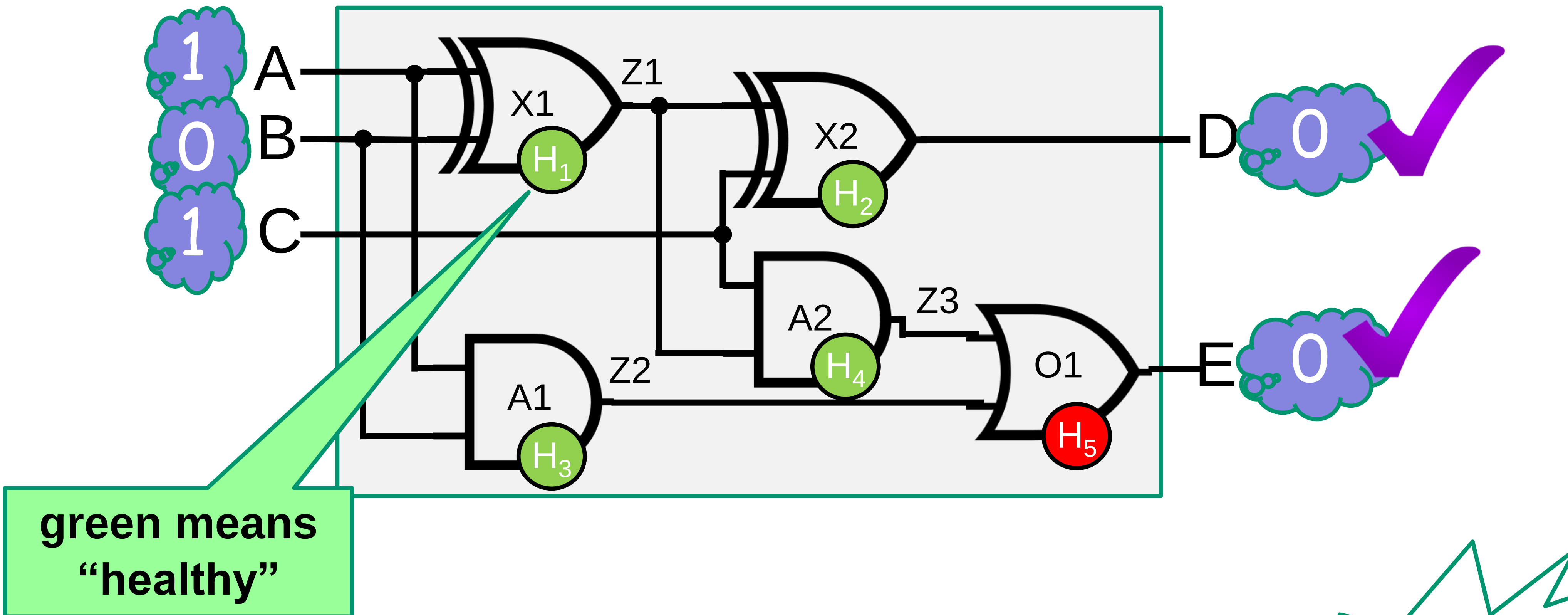




minimize

$\text{sum}([-H_1, -H_2, -H_3, -H_4, -H_5]) \leq K$

Modeling MBD: introduce health variables



$$\text{sum}(\ [\ -H_1,\ -H_2,\ -H_3,\ -H_4,\ -H_5\ ]\ ) \leq 1$$

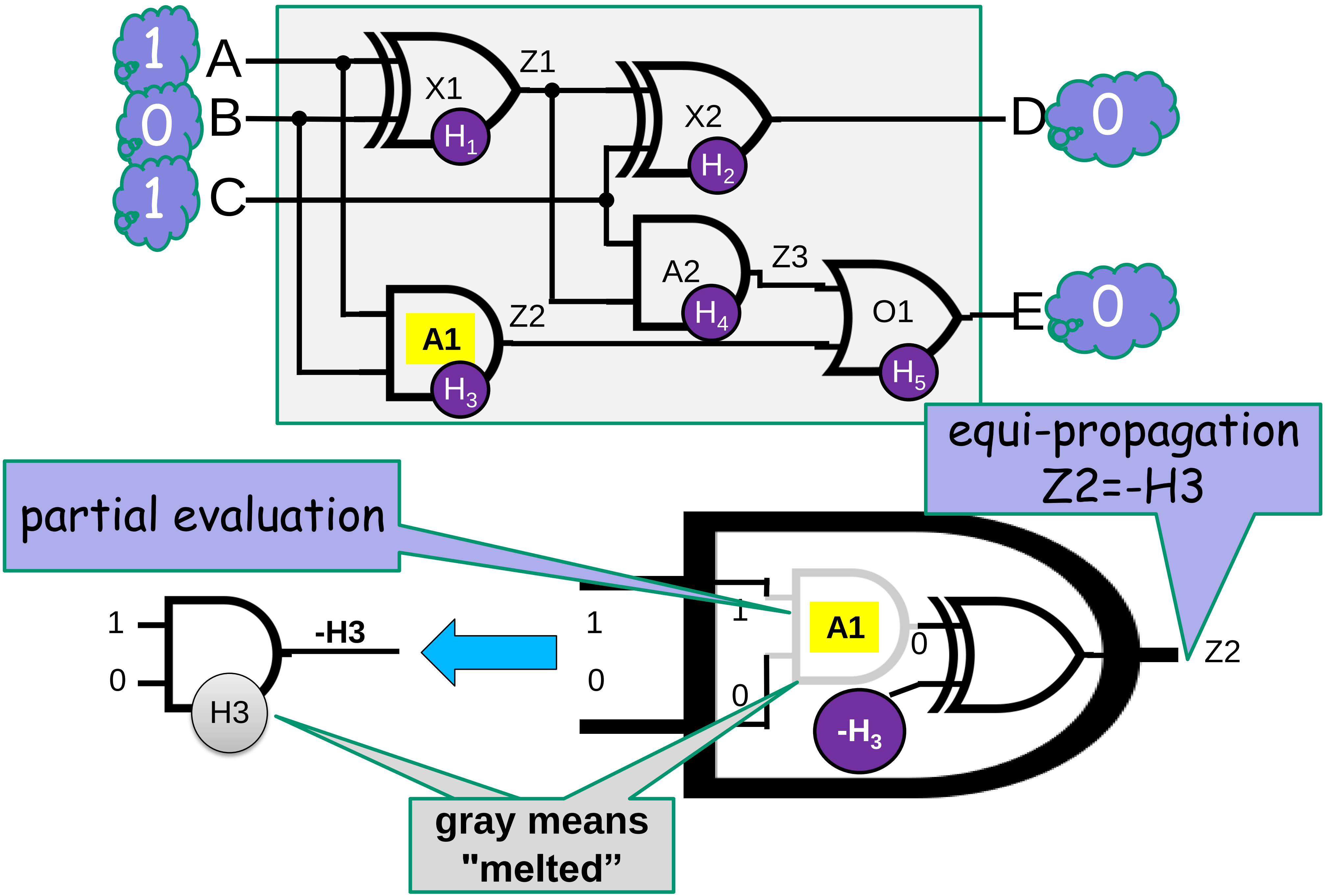
minimize

encoding to SAT is straighforward

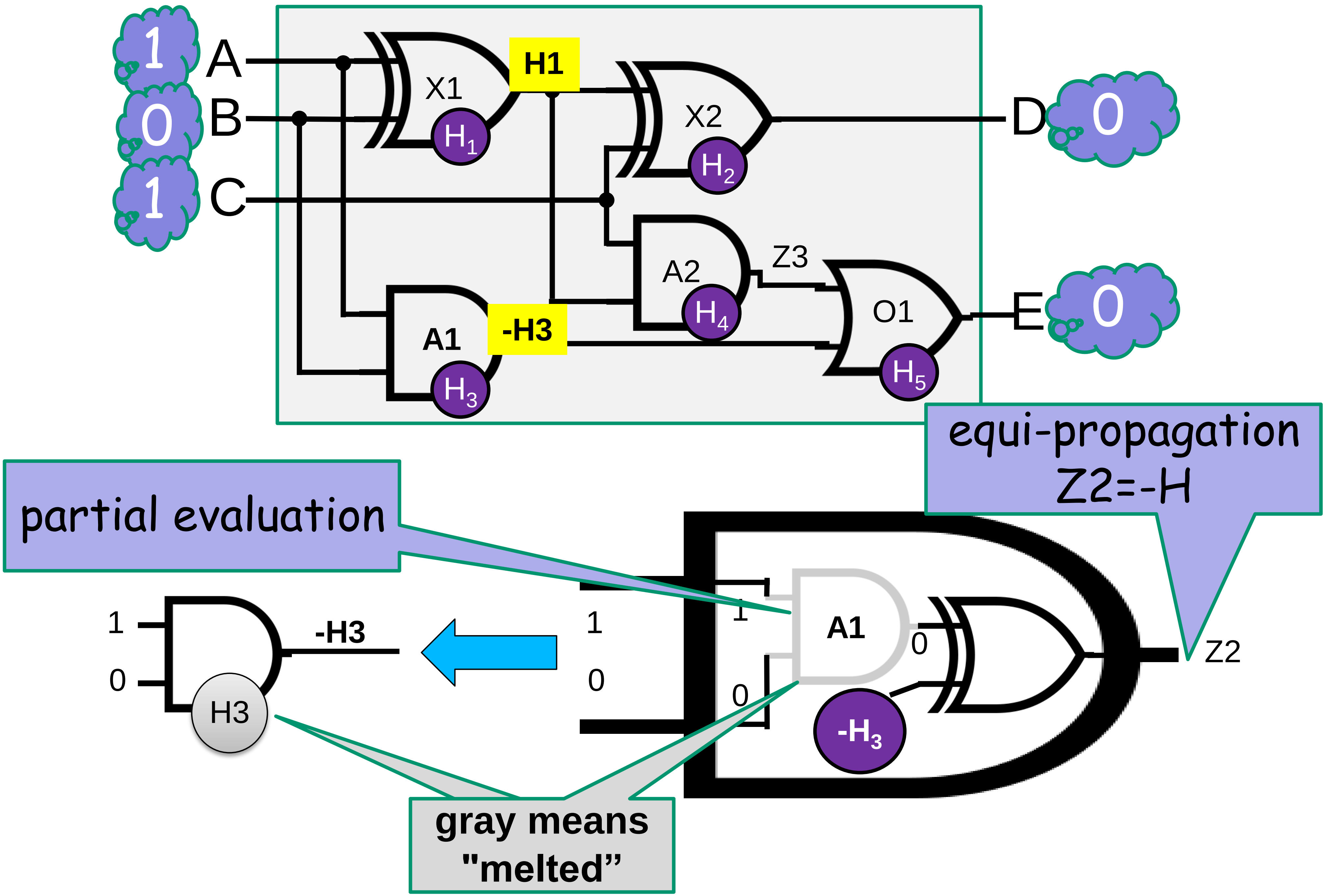
standard:
Smith 2005

Not competitive with other MBD tools

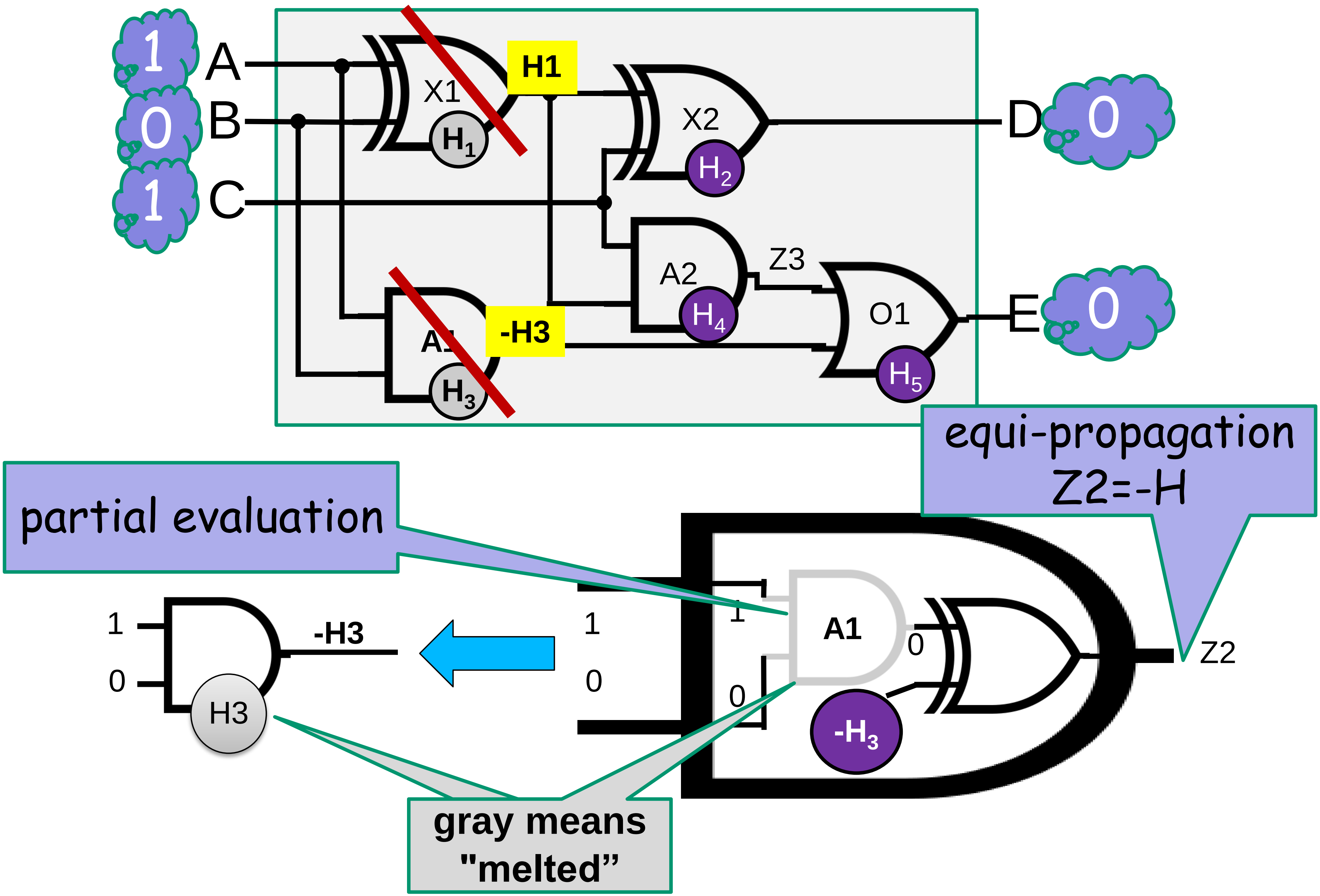
Simplify the encoding



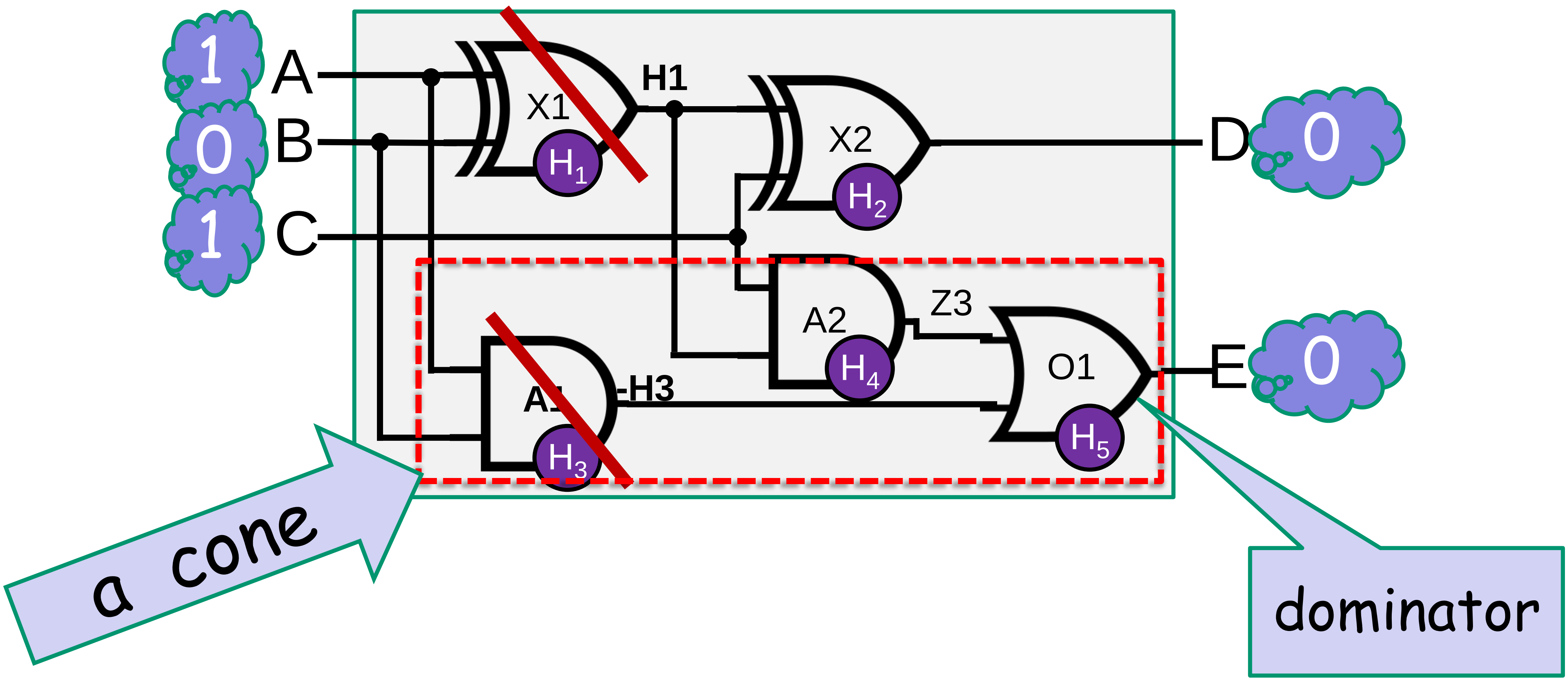
Simplify the encoding - I



Simplify the encoding - I

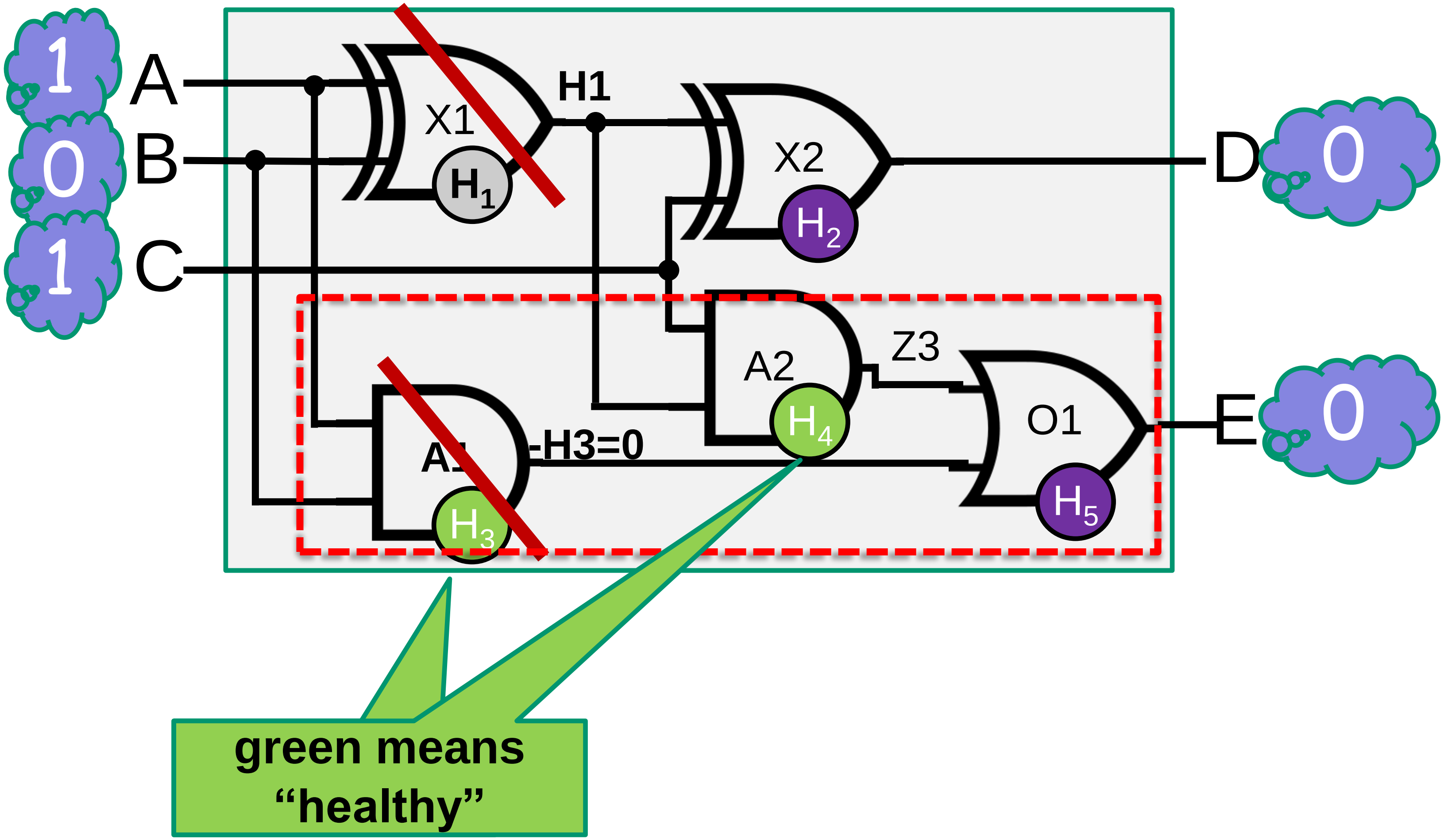


Simplify the encoding - II



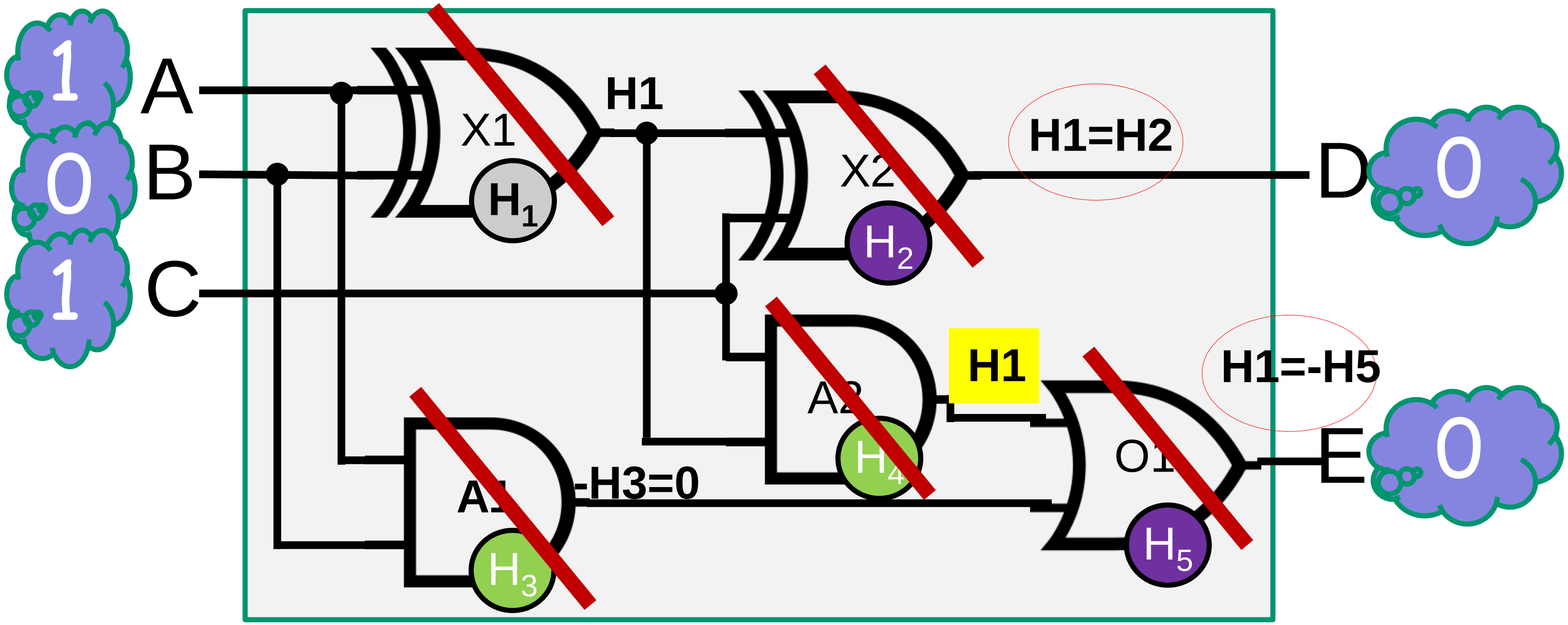
claim: A minimal cardinality diagnosis will always indicate at most one unhealthy gate per "cone". And wlog it is the "dominator"

Simplify the encoding - II



$$\text{sum}([-H_1, -H_2, 0, 0, -H_5]) \le K$$

Simplify the encoding - II



No SAT solving;
Diagnostics (min-cardinality) found by:

- preprocessing(cones)
- partial evaluation
- equi-propagation

$-H_5$]) ≤ K

, H_1]) ≤ K

H_1]) ≤ K

minimize K → H_1 =1

Compiling Model-Based Diagnosis to Boolean Satisfaction;

Amit Metodi, Roni Stern, Meir Kalech, Michael Codish; AAAI 2012 (to appear)

very good experimental results.

overtakes all current MBD systems

finds (for the first time) minimal cardinality diagnosis for the entire standard benchmarks

Outline

- Introduction:
  - Solving hard problems via SAT ✓
  - Focus on **programming** with SAT ✓
  - The need for higher-level languages ✓
- Higher low-level Language ✓
- (the basics for) A Compiler to CNF ✓
- Example: Model Based Diagnosis ✓
- Representing Finite Domain Integers
- Example: Magic Labels
- Conclusion

Modeling SMALL Finite Domain CSP

representing numbers (integers)

Binary

integer variable X
with domain {0,…,d}
is represented in

$$b = O(\log(d))$$

bits

Unary

integer variable X
with domain {0,…,d}
is represented in

$$b = O(d)$$

bits

Direct encod

$x_i \leftrightarrow (X = i)$

(X = 3) = [0,0,0,1,0,0]

encode "exactly-one"

encode "ordered"

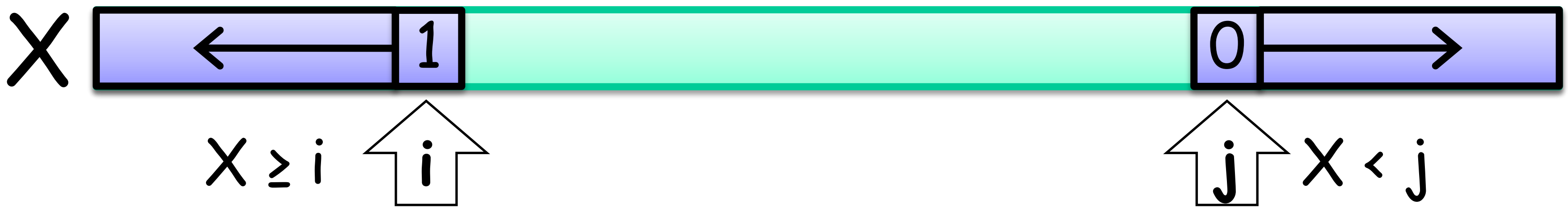
Order encoding

$x_i \leftrightarrow (X \geq i)$

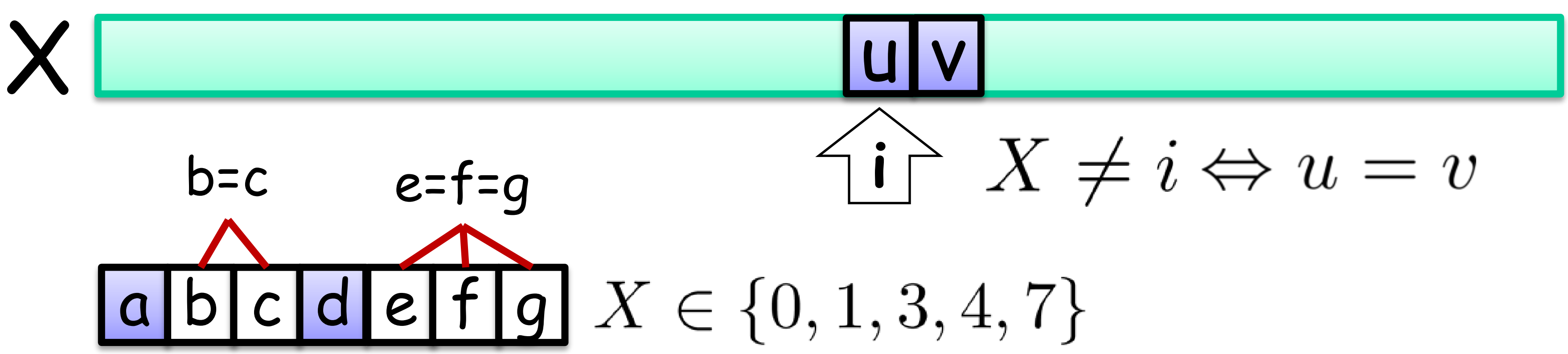
(X = 3) = [1,1,1,0,0]

Why Order Encoding ?

✓ **good for representing ranges (Sugar)**



✓ **good for arbitrary sets (Bee)**



Why Order Encoding ?

✓ **Lots of equi-propagation**

$$c = \left(\underbrace{[x_1, x_2, x_3]}_{X} + \underbrace{[y_1, y_2, y_3]}_{Y} = 3 \right)$$

$$\left(\begin{array}{r} c \wedge oe(X) \\ oe(Y) \end{array} \right) \models \left\{ \begin{array}{l} x_1 = -y_3 \wedge x_2 = -y_2 \wedge \\ x_3 = -y_1 \end{array} \right\}$$

order encoding

The Encoding to SAT needs NO Clauses. It is obtained by unification

$$\begin{array}{lcl} X & = & [-y_3, -y_2, -y_1] \\ Y & = & [y_1, y_2, y_3] \end{array}$$

Why Order Encoding ?

✓ **Lots of equi-propagation**

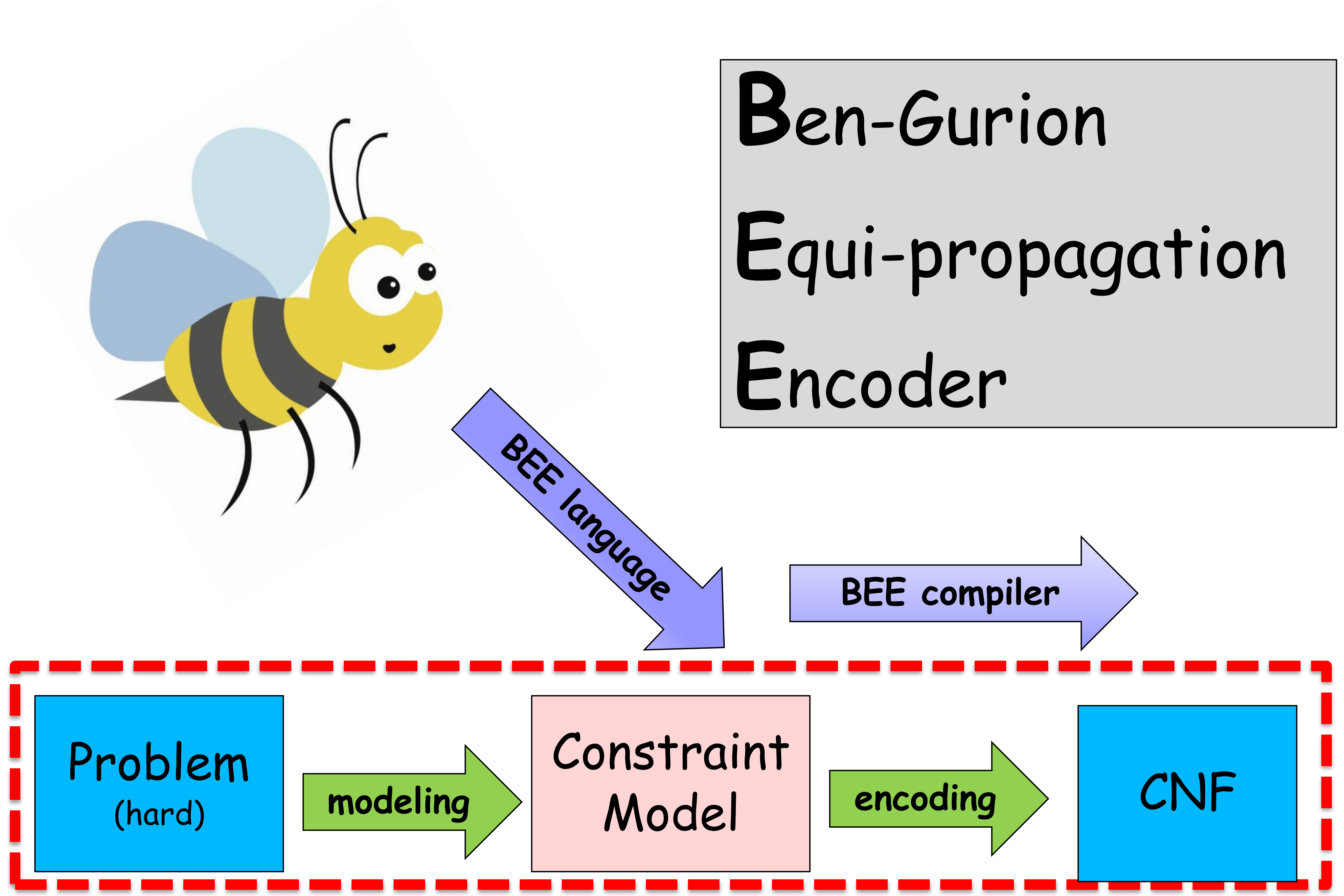
$$c = \left(\underbrace{[x_1, x_2, x_3]}_{X} + \underbrace{[y_1, y_2, y_3]}_{Y} = 3 \right)$$

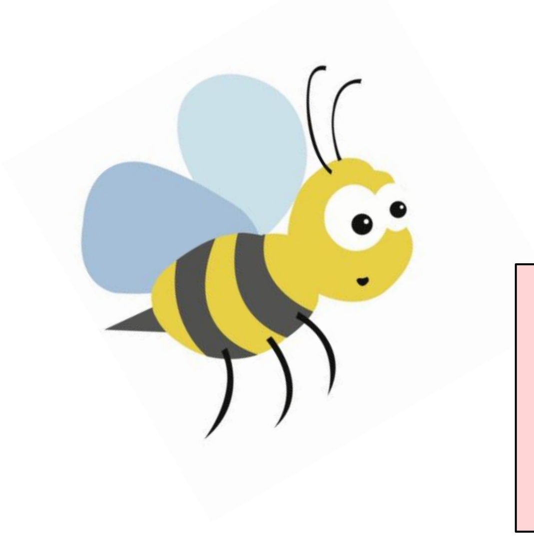
$$\begin{array}{lcl} X & = & [-y_3, -y_2, -y_1] \\ Y & = & [y_1, y_2, y_3] \end{array}$$

$$\begin{array}{l} [0,0,0] + [1,1,1] = [1,1,1] \\ [1,0,0] + [1,1,0] = [1,1,1] \\ [1,1,0] + [1,0,0] = [1,1,1] \\ [1,1,1] + [0,0,0] = [1,1,1] \end{array}$$

Implementing Equi-propagation

1. Using BDD's.
   - Can be prohibitive for global constraints.
   - Complete
2. Ad-Hoc rules (per constraint type)
   - Fast, precise in practice
   - Incomplete
3. Using SAT (on small groups of constraints)
   - Not too slow
   - Complete


Ben-Gurion
Equi-propagation
Encoder
BEE language
BEE compiler
Problem
(hard)
modeling
Constraint
Model
encoding
CNF



Constraint Model

bit-blasting

constraint simplification

encoding

CNF

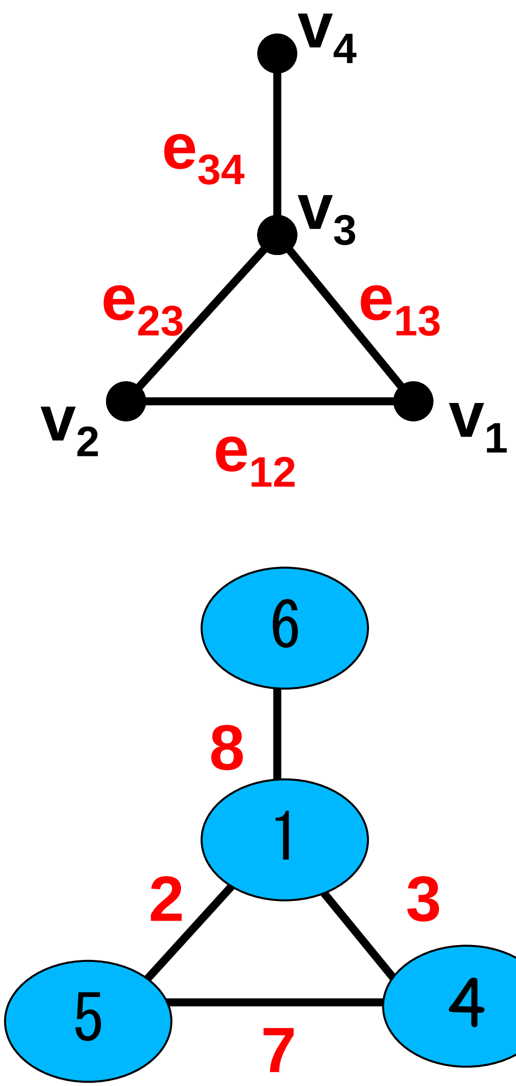
choice of representation
(default is order encoding)

Constraint
(C1, φ_1)

partial evaluation
equi-propagation
decomposition

standard techniques
(but encoding technique may differ after simplification)

Example: Magic Labels (VMTL)



$\texttt{new\_int}(v_1, 1, 8) \ldots \texttt{new\_int}(v_4, 1, 8)$

$\texttt{new\_int}(e_{12}, 1, 8) \ldots \texttt{new\_int}(e_{34}, 1, 8)$

$\texttt{new\_int}(k, 14, 14)$

$\texttt{allDiff}(v_1, v_2, v_3, v_4, e_{12}, e_{13}, e_{23}, e_{34})$

$$
\begin{aligned}
v_1 + e_{12} + e_{13} &= k \\
v_2 + e_{12} + e_{23} &= k \\
v_3 + e_{13} + e_{23} + e_{34} &= k \\
v_4 + e_{34} &= k
\end{aligned}
$$

simplifying sum constraints

$$\texttt{int\_plus}(\ [1, A_2, A_3, A_4, A_5, A_6, A_7, A_8],\ [1, B_2, B_3, B_4, B_5, B_6, B_7, B_8],\ [\underbrace{1, \ \ldots\ldots \ , 1}_{\texttt{14 times}}, 0, 0]\ )$$

$$\texttt{int\_plus}(\ [1, 1, 1, 1, 1, 1, A_7, A_8],\ [1, 1, 1, 1, 1, 1, B_7, B_8],\ [\underbrace{1, \ \ldots\ldots \ , 1}_{\texttt{14 times}}, 0, 0]\ )$$

bound propagation ?

A & B take values {6,7,8}

is it a CSP thing?

no. it is **equi-propagation**

simplifying sum constraints

```
int_plus(
  [1, A2, A3, A4, A5, A6, A7, A8],
  [1, B2, B3, B4, B5, B6, B7, B8],
  [1,  ......  , 1, 0, 0]
   \___ 14 times ___/
)
```

e.p. →

```
int_plus(
  [1, 1, 1, 1, 1, 1, A7, A8],
  [1, 1, 1, 1, 1, 1, B7, B8],
  [1,  ......  , 1, 0, 0]
   \___ 14 times ___/
)
```

p.e. ↓

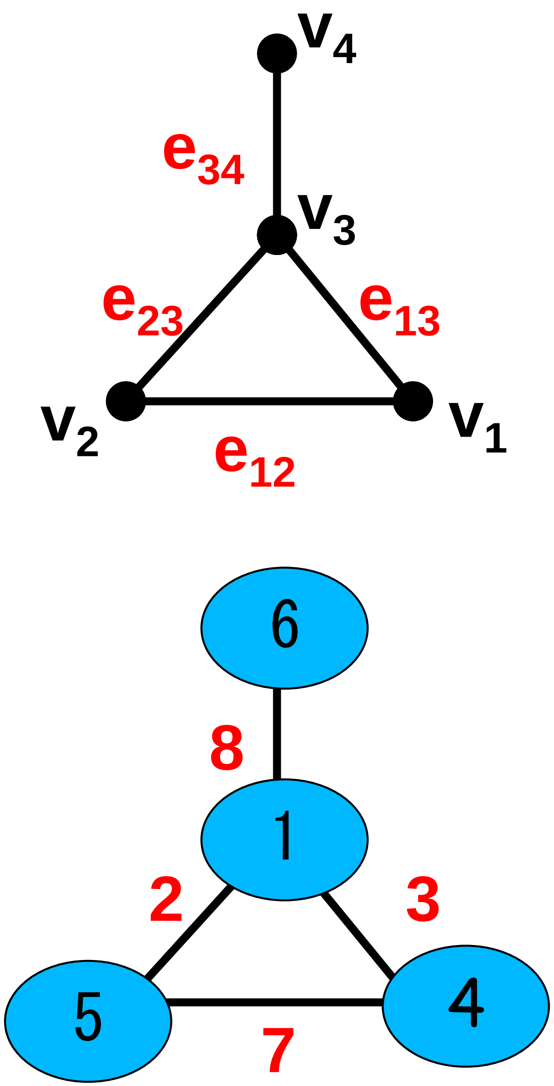
int_plus([A7, A8], [B7, B8], [1, 1, 0, 0])

← **e.p.**

binding : $B_7 = -A_8,\ B_8 = -A_7$)

~~int_plus([A7, A8], [−A8, −A7], [1, 1, 0, 0])~~

back to the VMTL example



$new\_int(v_1, 1, 8) \ldots new\_int(v_4, 1, 8)$

$new\_int(e_{12}, 1, 8) \ldots new\_int(e_{34}, 1, 8)$

$new\_int(k, 14, 14)$

$\texttt{allDiff}(v_1, v_2, v_3, v_4, e_{12}, e_{13}, e_{23}, e_{34})$

$$
\begin{aligned}
v_1 + e_{12} + e_{13} &= k \\
v_2 + e_{12} + e_{23} &= k \\
v_3 + e_{13} + e_{23} + e_{34} &= k \\
v_4 + e_{34} &= k
\end{aligned}
$$

VMTL: Simplifying Constraints

$$(1) \quad \texttt{int\_array\_plus}([V_4, E_4], 14)$$
$$(2) \quad \texttt{allDiff}([V_1, V_2, V_3, V_4, E_1, E_2, E_3, E_4]),$$

could take values **6,7,8**

$$V_4 = [1, 1, 1, 1, 1, 1, V_{4,7}, V_{4,8}]$$
$$E_4 = [1, 1, 1, 1, 1, 1, \text{-}V_{4,8}, \text{-}V_{4,7}]$$

$$(2) \quad \texttt{allDiff}([V_1, V_2, V_3, V_4, E_1, E_2, E_3, E_4]),$$

$V_4 \neq E_4$

$$V_4 = [1, 1, 1, 1, 1, 1, V_{4,7}, V_{4,7}]$$
$$E_4 = [1, 1, 1, 1, 1, 1, \text{-}V_{4,7}, \text{-}V_{4,7}]$$

could take values **6,8**

Example: Magic Labels (VMTL)

$$new\_int(v_1, 1, 8) \ldots new\_int(v_4, 1, 8)$$
$$new\_int(e_{12}, 1, 8) \ldots new\_int(e_{34}, 1, 8)$$

$$new\_int(k, 14, 14)$$

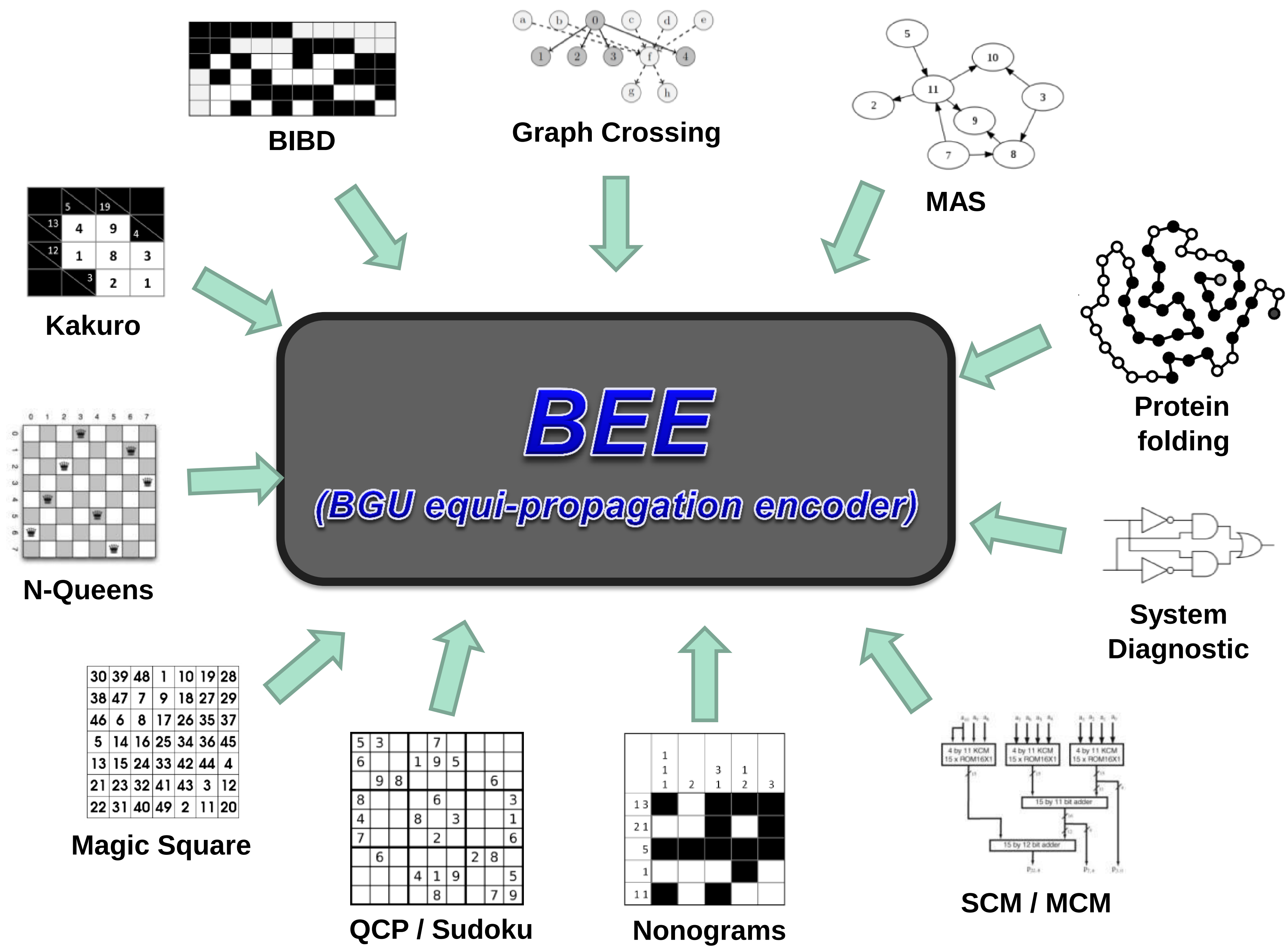
$$\texttt{allDiff}(v_1, v_2, v_3, v_4, e_{12}, e_{13}, e_{23}, e_{34})$$

$$\begin{aligned} v_1 + e_{12} + e_{13} &= k \\ v_2 + e_{12} + e_{23} &= k \\ v_3 + e_{13} + e_{23} + e_{34} &= k \\ v_4 + e_{34} &= k \end{aligned}$$

909 clauses
136 Bits

↓

298 clauses
49 Bits


BEE
(BGU equi-propagation encoder)
BIBD
Graph Crossing
MAS
Kakuro
Protein folding
N-Queens
System Diagnostic
Magic Square
QCP / Sudoku
Nonograms
SCM / MCM

Compiling Finite Domain Constraints to SAT with Bee (tool paper & release);

Amit Metodi and Michael Codish; ICLP 2012

Implementation

Currently we use CryptoMiniSAT (or MiniSAT)

The compiler is written in Prolog (SWI)
(equating Boolean variables using unification)

SAT, BDD and Adhoc rules to implement E.P.

Where now?

- applications
- implementation
- complete equi-propagation (on chunks)
  - how to implement it
  - how to decide where to apply it

Conclusions

New Emerging Paradigm where we program
with SAT (or SMT) solvers;

High"er-level (constraint based) language to aid in the encoding lets us focus on the modeling

The notion of E.P. captures many standard CSP techniques and more.

Making the CNF smaller is not the real goal;
It is more about restricting the search space by identifying equalities that must hold