Compiling Finite Domain Constraints to SAT with BEE

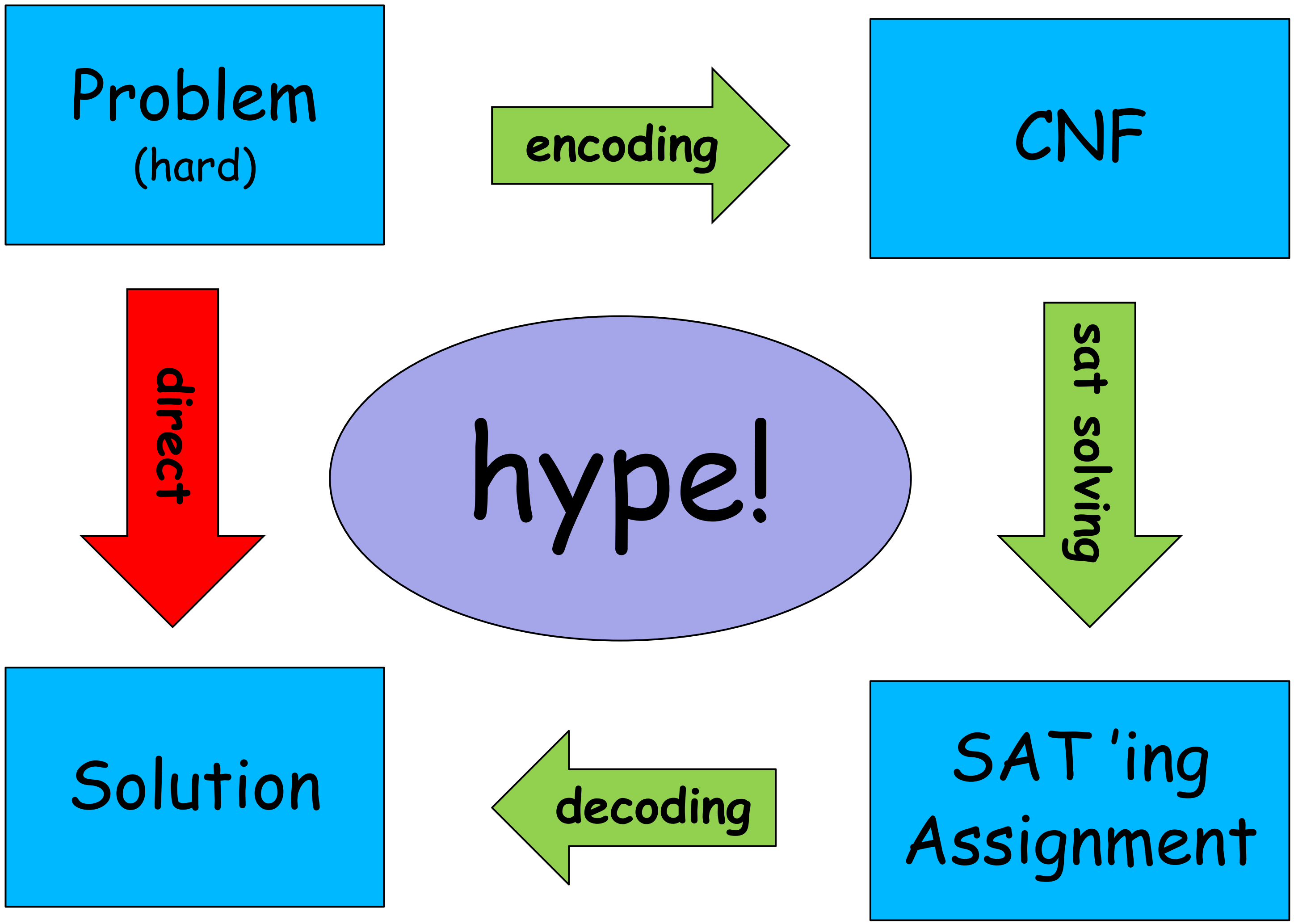
Michael Codish

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

Joint work: Yoav Fekete & Amit Metodi
cadence

In Collaboration with: Vitaly Lagoon & Peter Stuckey
MathWorks

It is all about: Solving hard problems via SAT encodings



Ben-Gurion

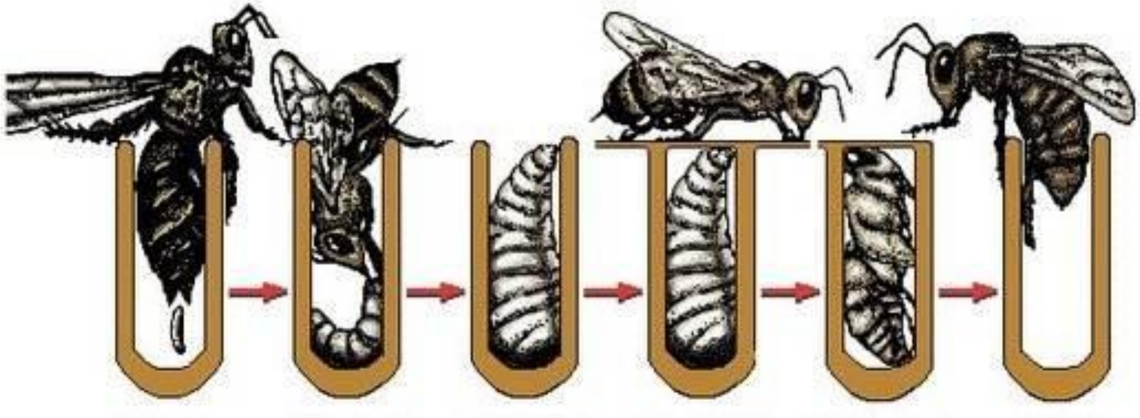
Equi-propagation

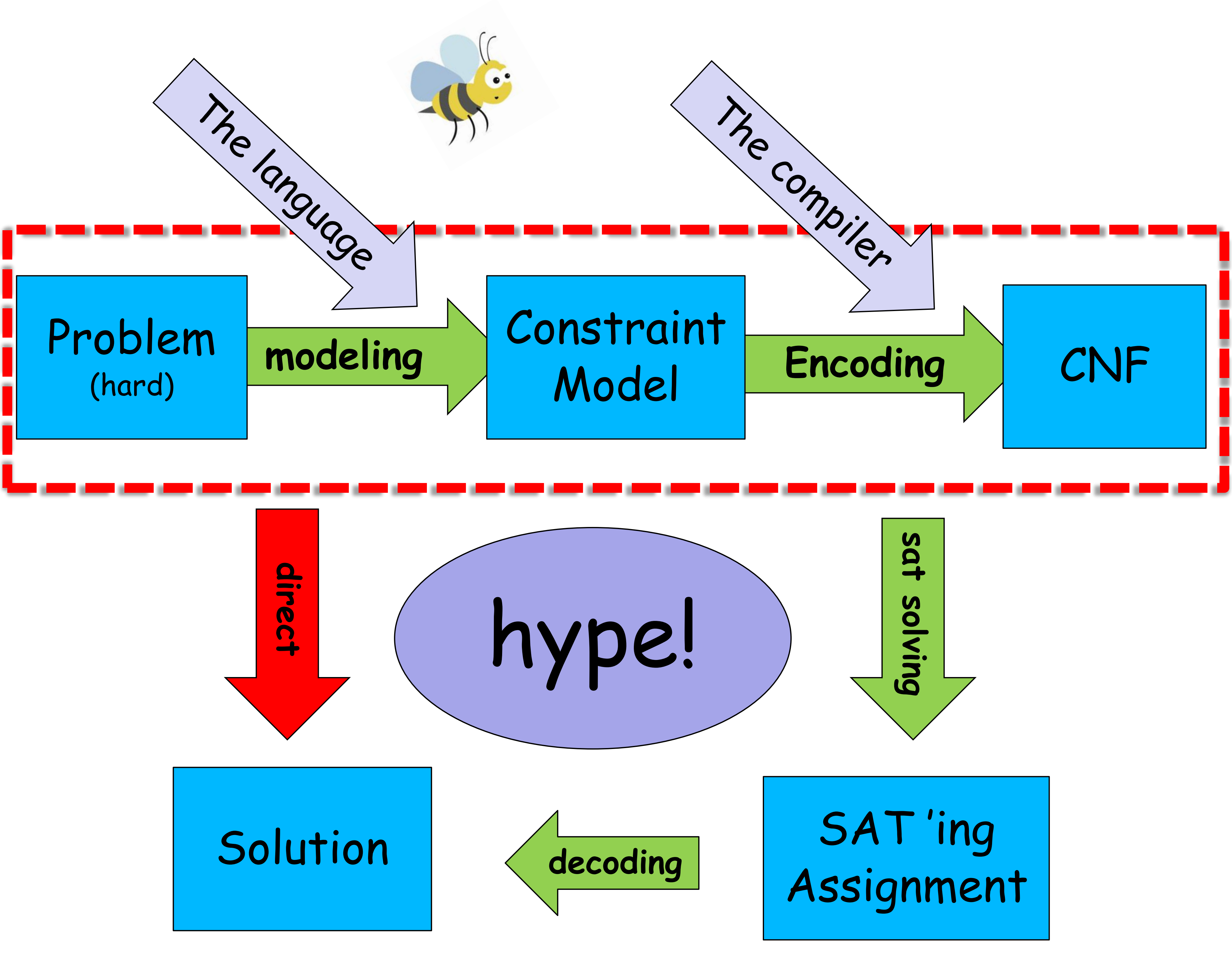
Encoder

was born (*) with two objectives:

- Facilitate the (user) process of **encoding** a (constraint) problem to CNF
- **Compile** constraint models to CNF while applying optimizations in order to generate (usually) smaller and better CNF formulas.

(*) Amit Metodi, Michael Codish, Vitaly Lagoon, Peter J. Stuckey: Boolean Equi-propagation for Optimized SAT Encoding. CP **2011**: 621-636

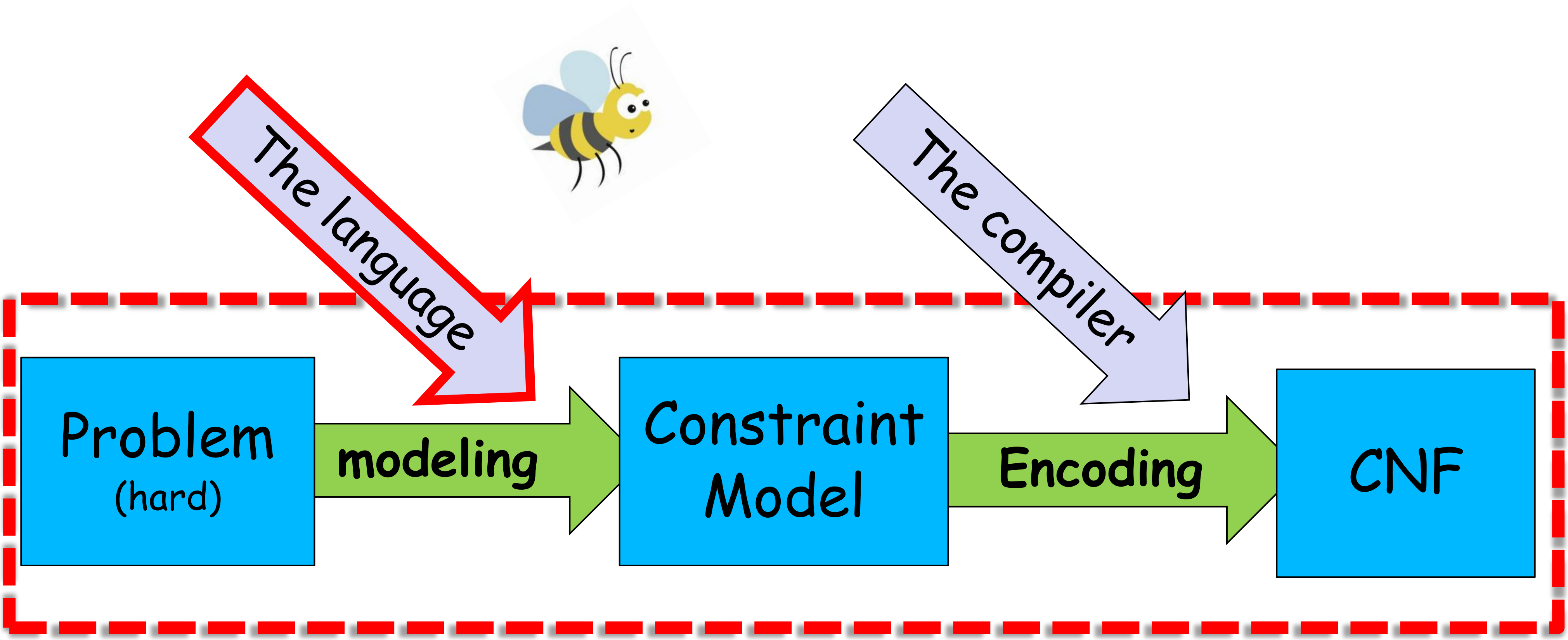



The language
The compiler
Problem
(hard)
modeling
Constraint
Model
Encoding
CNF
direct
hype!
sat solving
Solution
decoding
SAT'ing
Assignment

Outline

- Introduction
- BEE in a nutshell
  - Order encoding (representing integers)
  - Equi-propagation (ad-hoc)

- The "new" stuff
  - Complete Equi-Propagation
  - Cardinality Constraints in BEE
  - The binary extension of BEE


The language
The compiler
Problem
(hard)
modeling
Constraint Model
Encoding
CNF

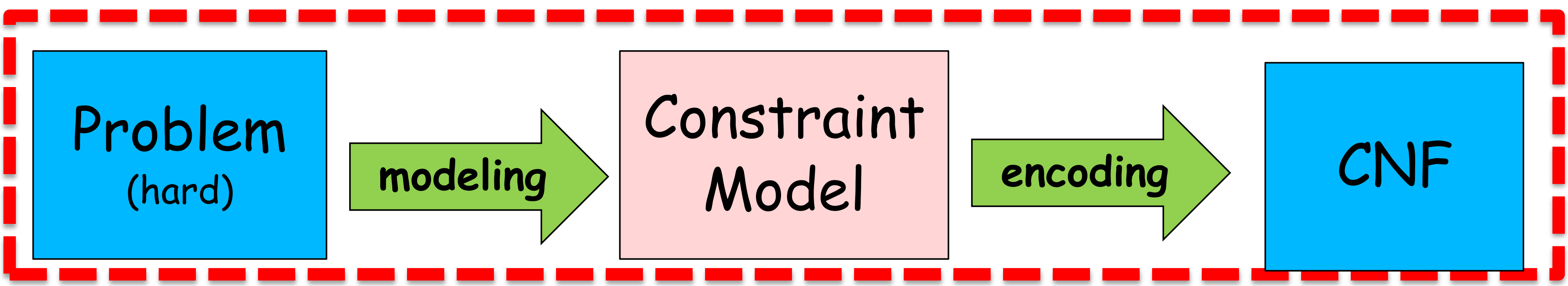
Example: encoding Sudoku

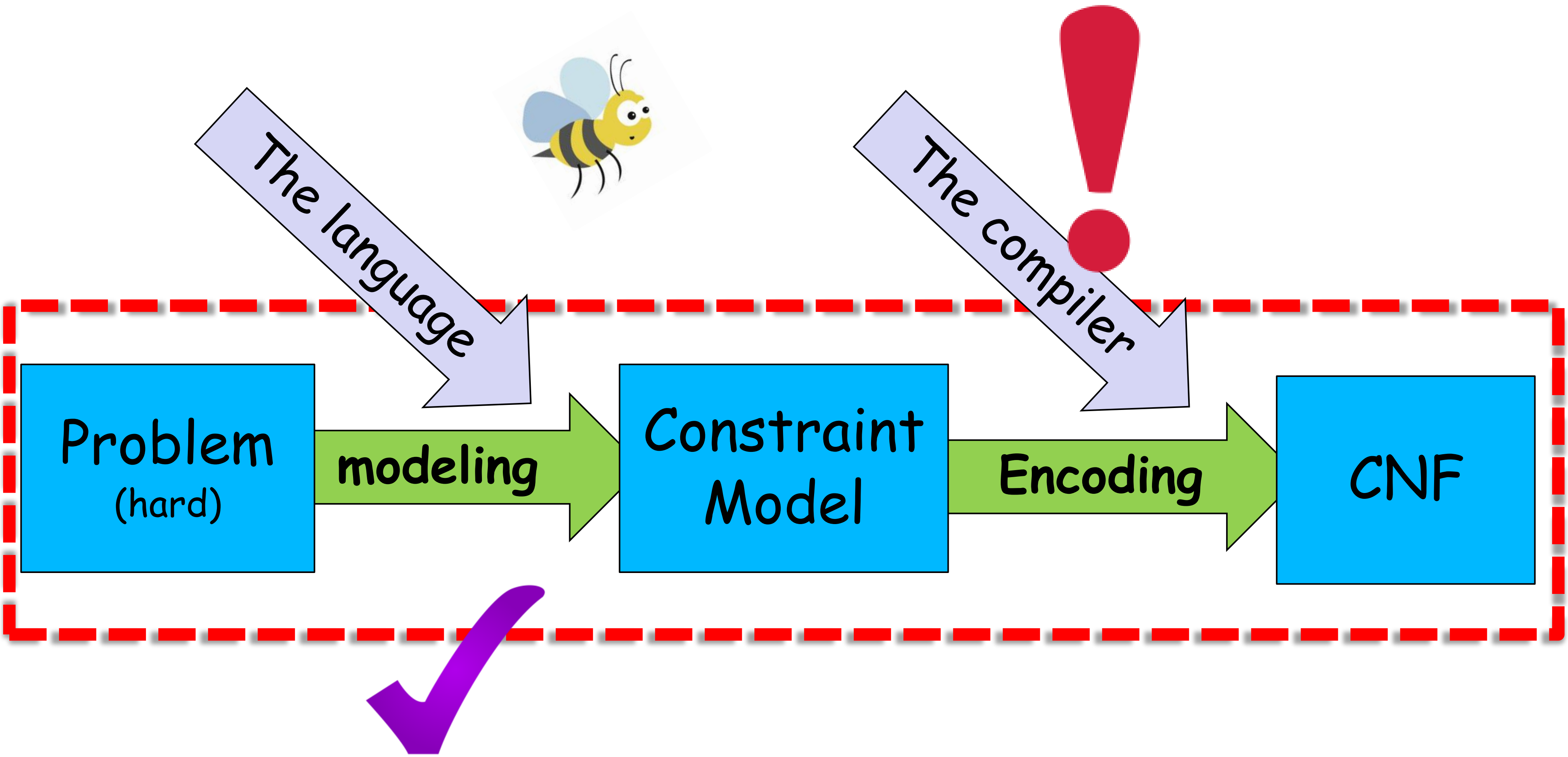
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

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

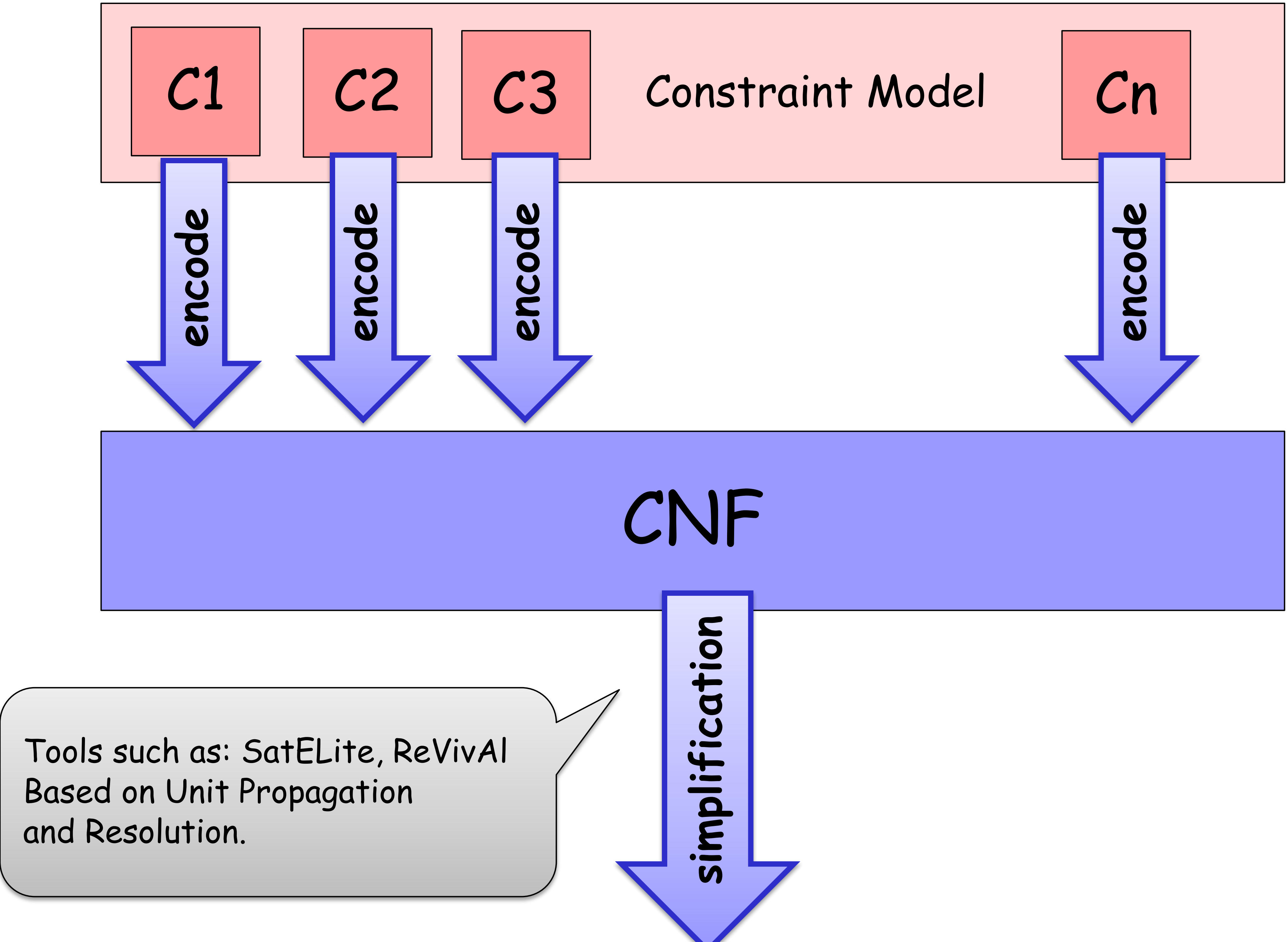
$$\mathtt{allDiff}([X_{1,1}, \ldots, X_{1,9}])$$
$$\vdots$$

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

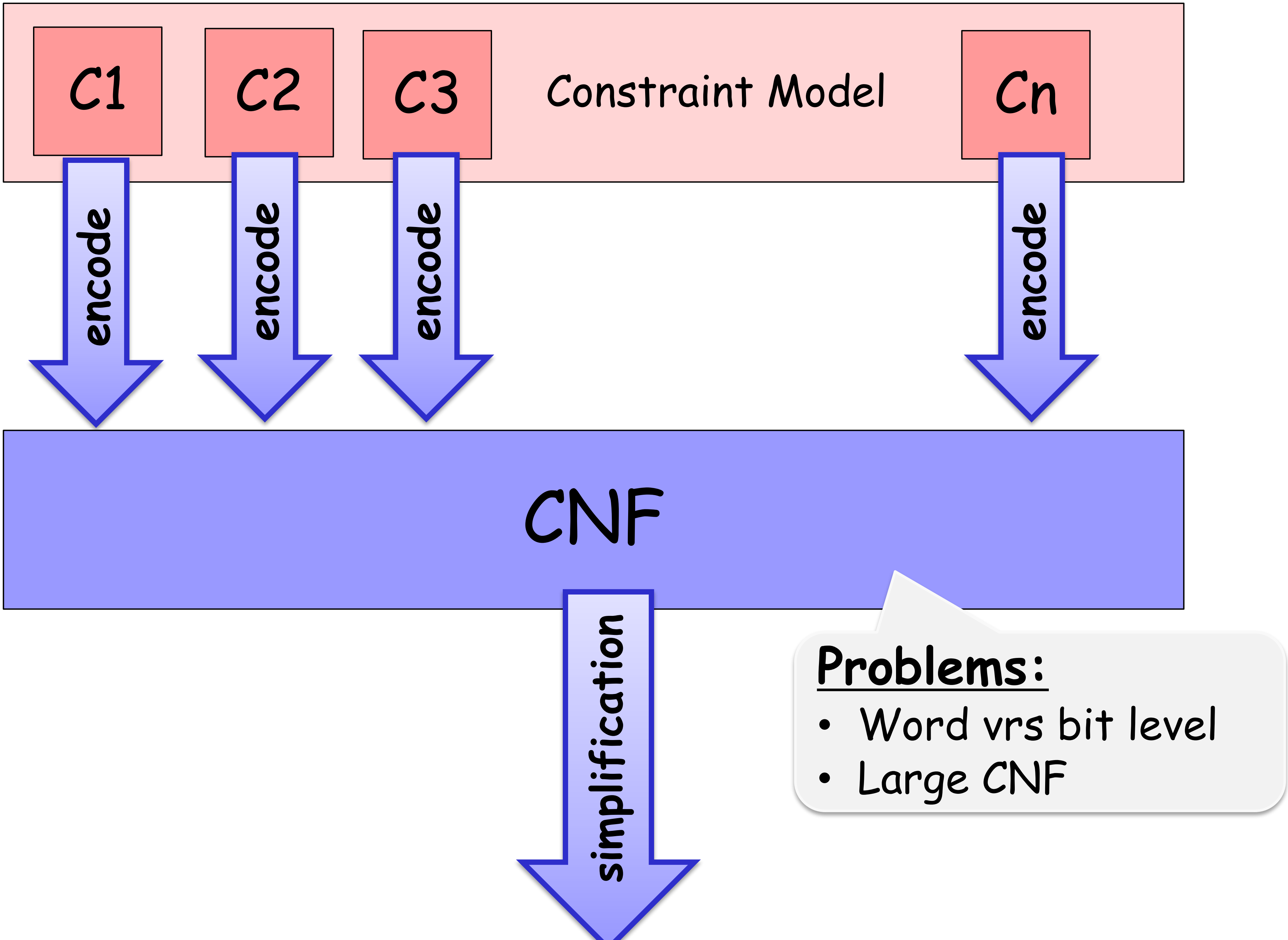


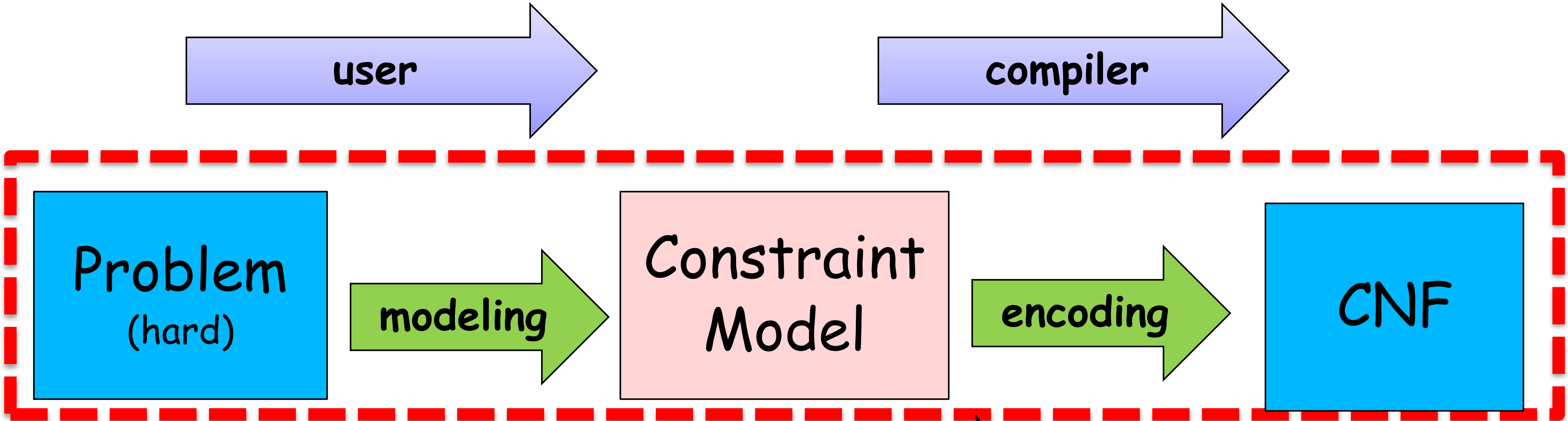

The language
The compiler
Problem
(hard)
modeling
Constraint Model
Encoding
CNF

The Usual Approach



The Usual Approach



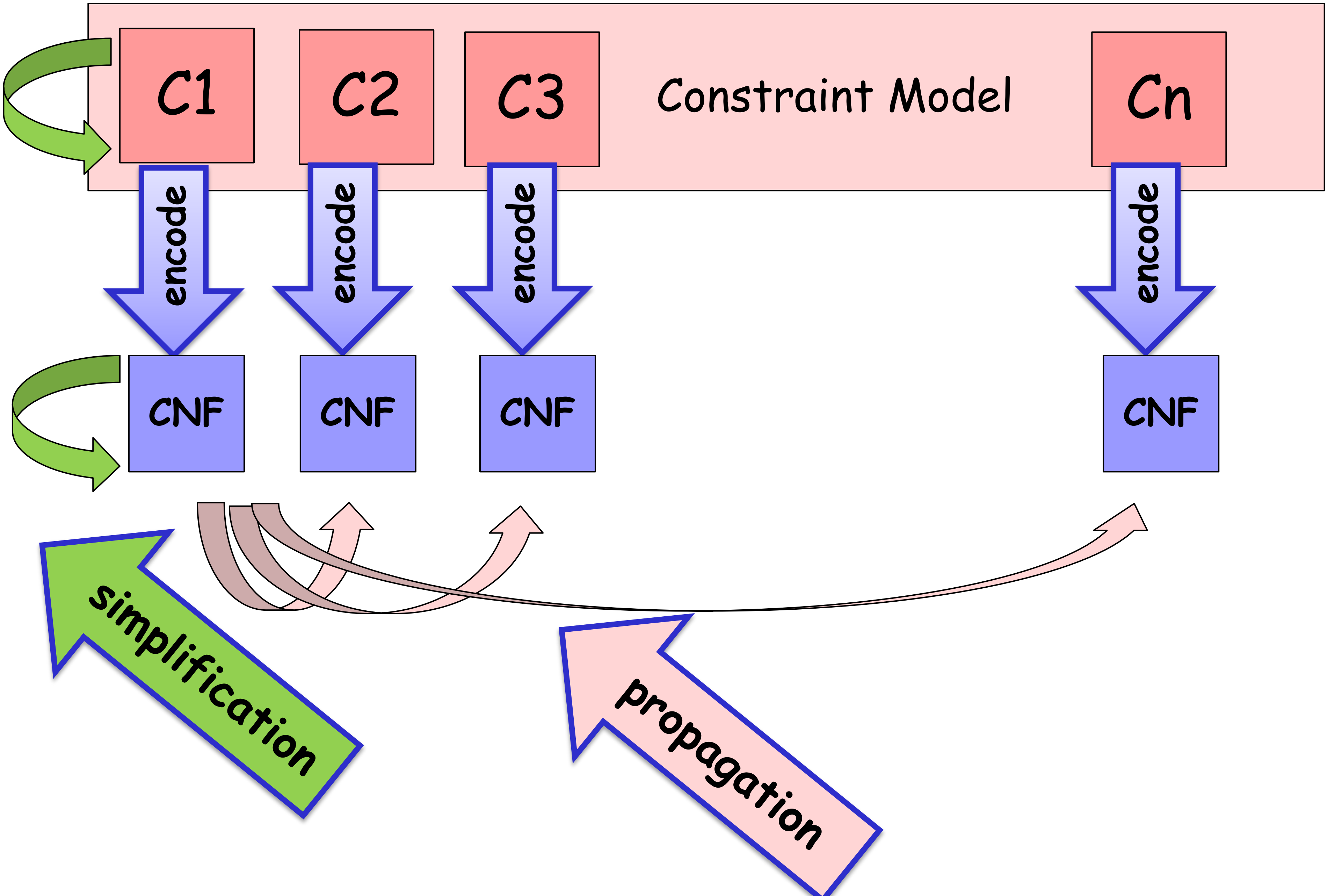


The CNF you want to optimize did not fall out of the sky

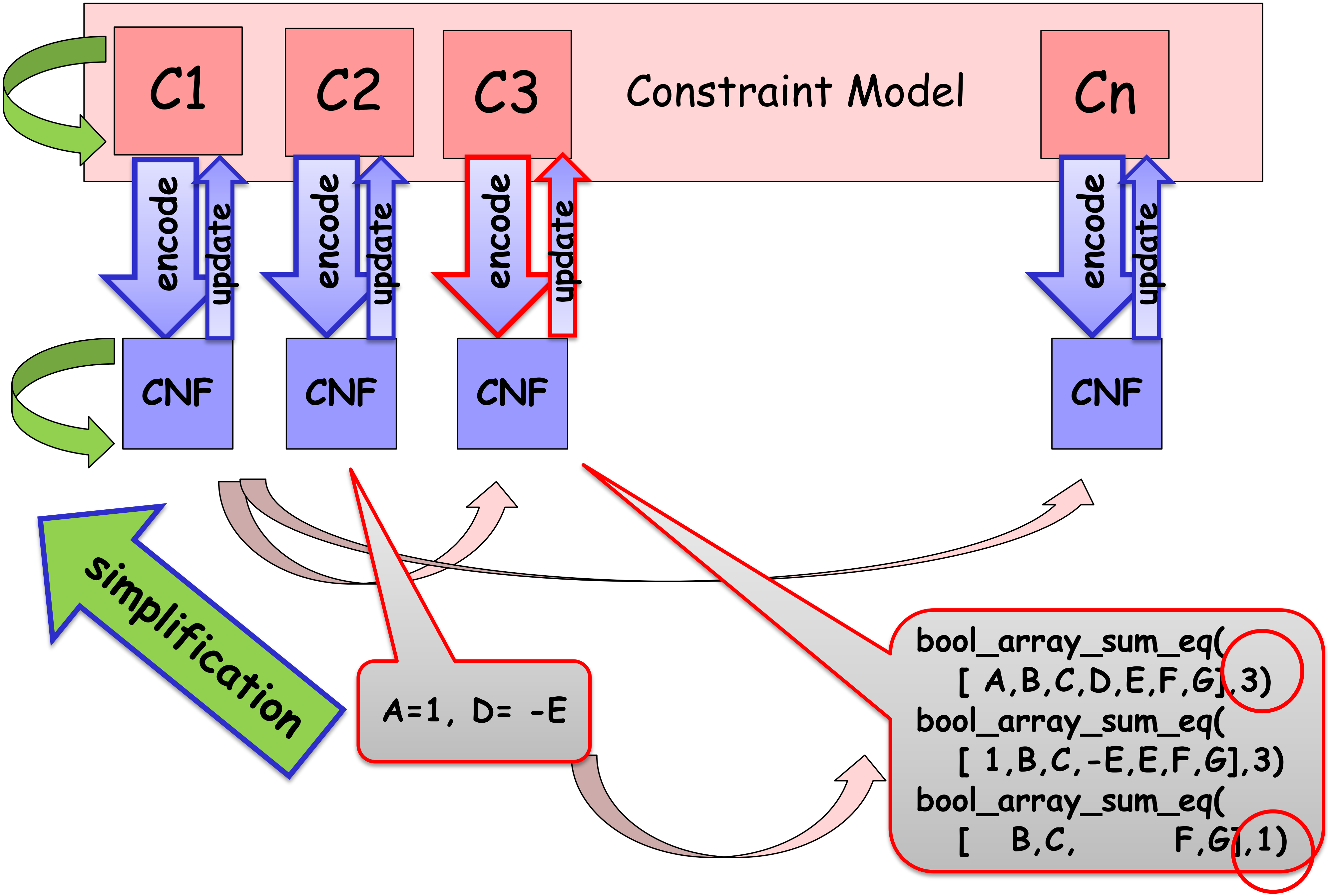
Let the constraint model drive the CNF optimization

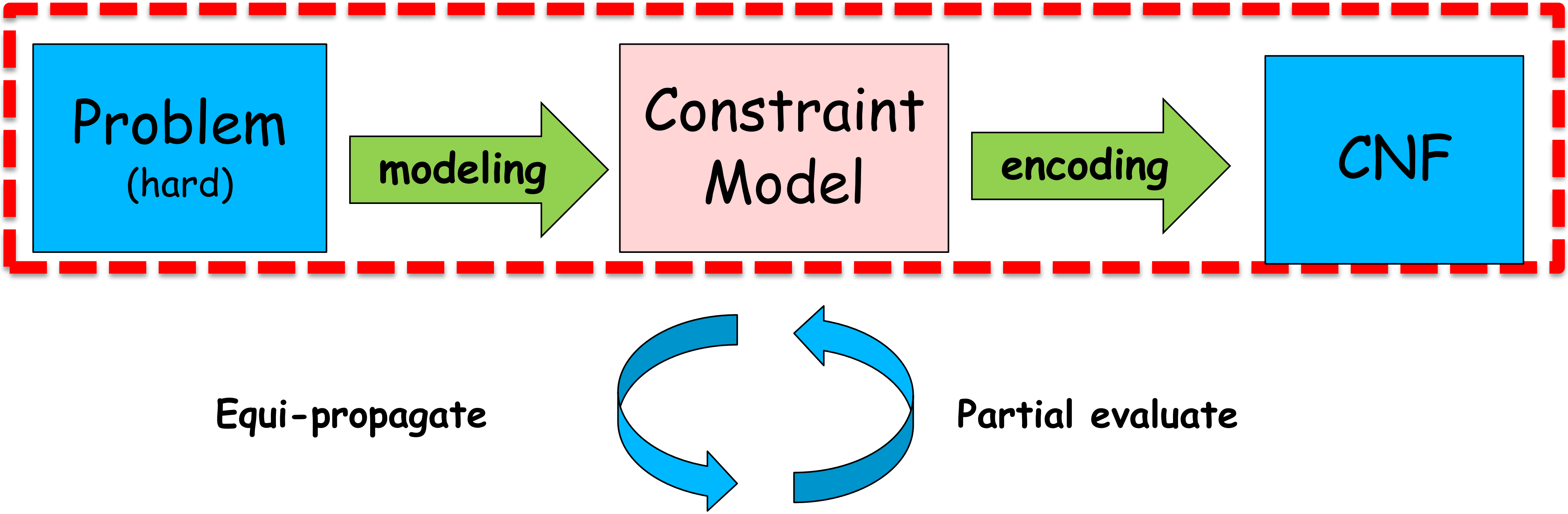
Optimize it while generating it

The BEE Approach



The BEE Approach





1. view each "single" constraint as a Boolean formula
2. derive ("all") implied equalities between literals and constants
3. apply them to simplify **all** constraints

more powerful reasoning but on smaller CNF portions

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

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

TWO DESIGN CHOICES

- Representing numbers

Order encoding (unary)

$X = [x_1, \ldots, x_i, \ldots, x_n]$

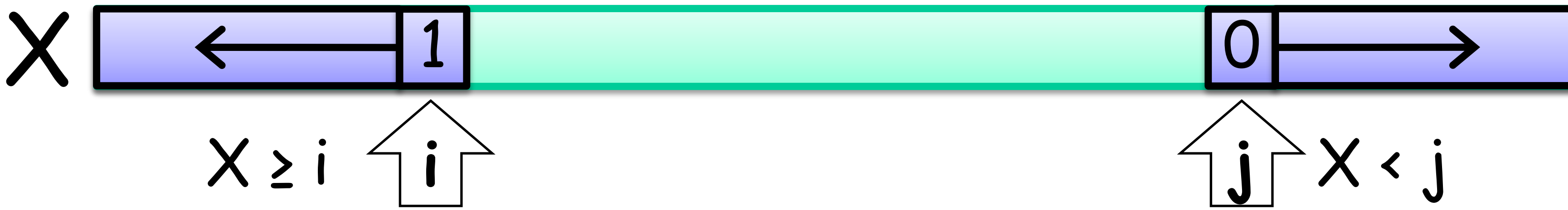
$x_i \leftrightarrow (X \geq i)$

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

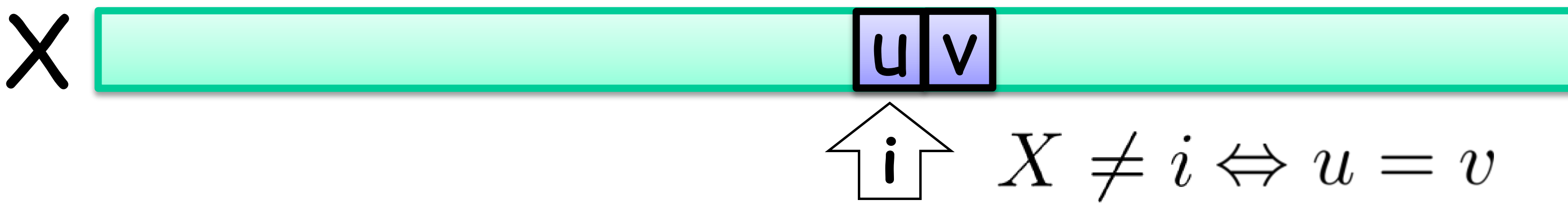
why?

Lots of equi-propagation

1



2



3

$[x_1, x_2, x_3] + [y_1, y_2, y_3] = 3$

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

$x_1 = -y_3$

$x_2 = -y_2$

$x_3 = -y_1$

TWO DESIGN CHOICES

➢ Implementing Equi-Propagation

1. Using BDD's.
   - Prohibitive for global constraints.
   - Complete
2. Using SAT (on small groups of constraints)
   - In practice, surprisingly, "not slow"
   - Complete
3. Ad-Hoc rules (per constraint type)
   - Fast, precise in practice
   - Incomplete

Ad-Hoc Rules: int_plus

- **Equi-Propagation**

$c = \texttt{int\_plus}(X, Y, Z)$ where $X = \langle x_1, \ldots, x_n \rangle$, $Y = \langle y_1, \ldots, y_m \rangle$, and $Z = \langle z_1, \ldots, z_{n+m} \rangle$	
if in E	then add in $\mu_c(\mathtt{E})$
$X \geq i,\ Y \geq j$	$Z \geq i + j$
$X < i,\ Y < j$	$Z < i + j - 1$
$Z \geq k,\ X < i$	$Y \geq k - i$
$Z < k,\ X \geq i$	$Y < k - i$
$X = i$	$z_{i+1} = y_1, \ldots, z_{i+m} = y_m$
$Z = k$	$x_1 = \neg y_k, \ldots, x_k = \neg y_1$

- **Partial Evaluation**

$c = \texttt{int\_plus}(X, Y, Z)$ where $X = \langle x_1, \ldots, x_n \rangle$, $Y = \langle y_1, \ldots, y_m \rangle$, and $Z = \langle z_1, \ldots, z_{n+m} \rangle$	
if	then replace with
$X = i$	$\texttt{true}$
$Z = k$	$\texttt{true}$
$X \geq i,\ Z \geq i$	$\texttt{int\_plus}([x_{i+1}, \ldots, x_n], Y, [z_{i+1}, \ldots, z_{n+m}])$
$X \leq i,\ Z \leq i + m$	$\texttt{int\_plus}([x_1, \ldots, x_i], Y, [z_1, \ldots, z_{i+m}])$

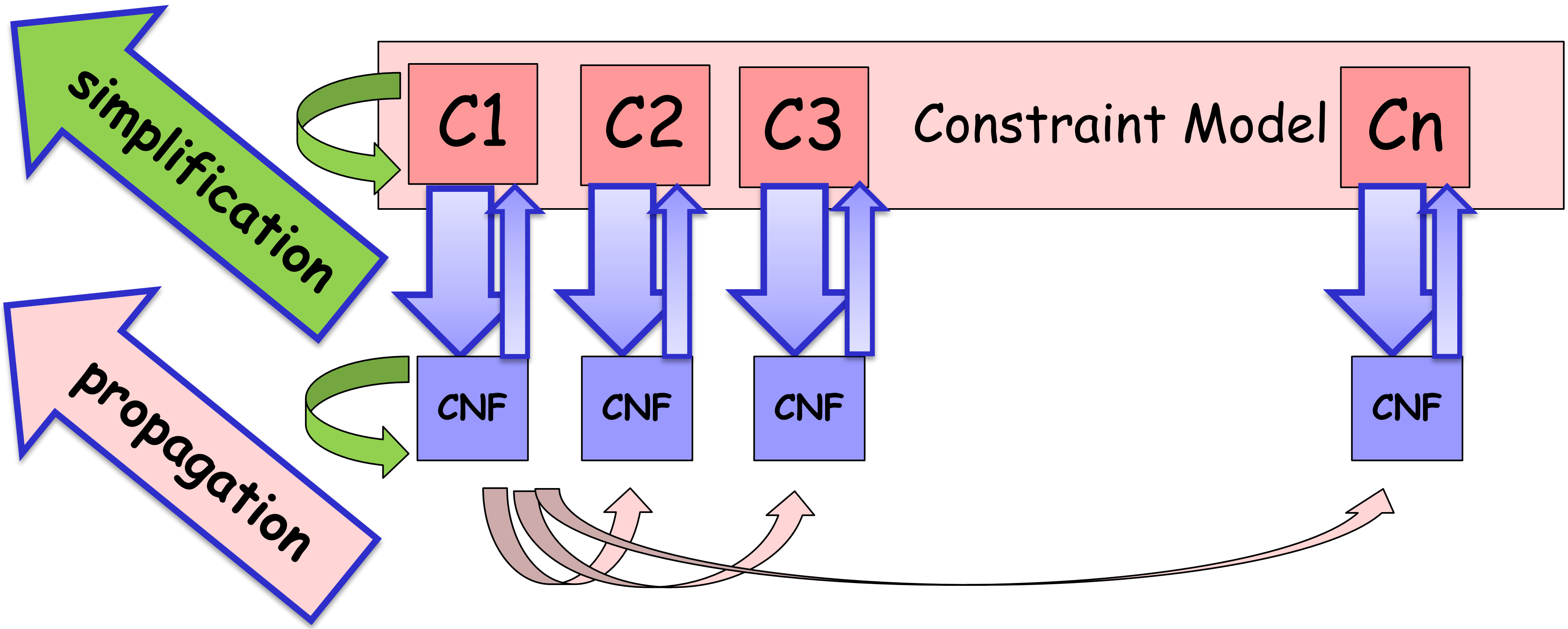
Outline

- Introduction
- BEE in a nutshell

http://amit.metodi.me/research/bee/

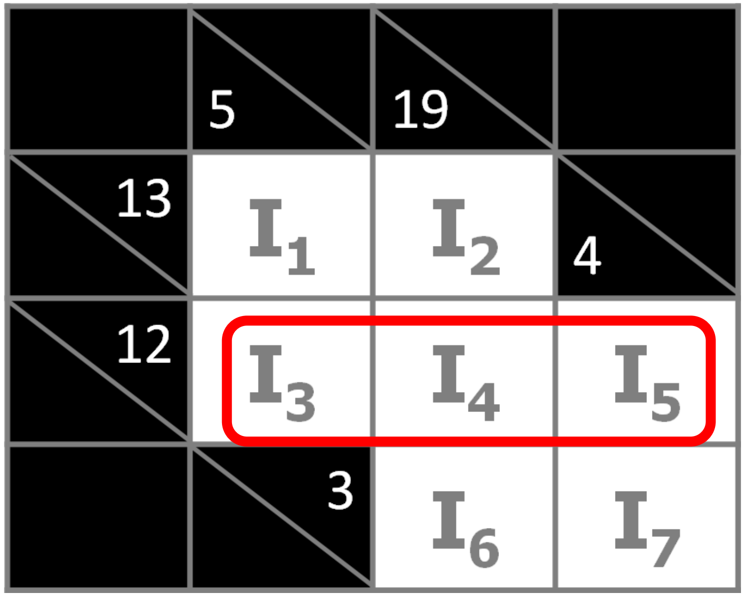
- The "new" stuff
  - Complete Equi-Propagation
  - Cardinality Constraints in BEE
  - The binary extension of BEE

Complete Equi-propagation



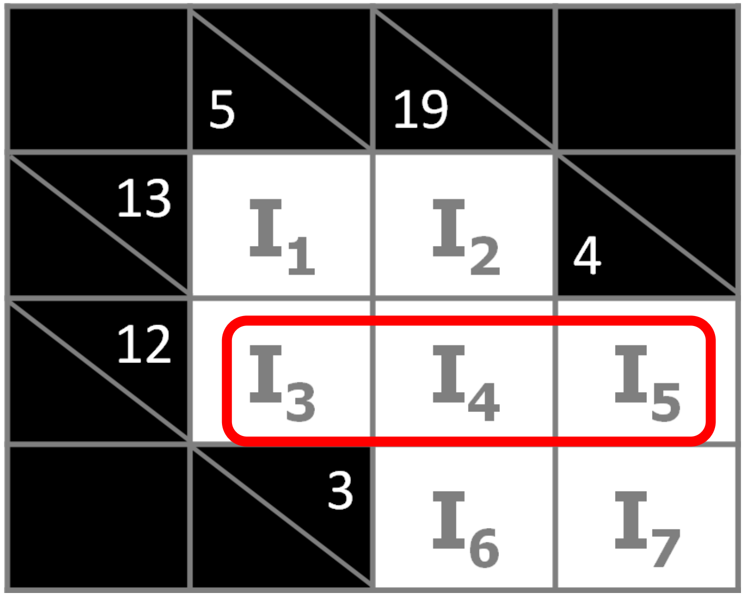
designate specific **sets** of constraints for **complete** equi-propagation (using a SAT solver)

Example: Kakuro



new_int($I_1, 1, 9$)	int_array_plus($[I_1, I_2], 13$)	allDiff($[I_1, I_2]$)
new_int($I_2, 1, 9$)	int_array_plus($[I_1, I_3], 5$)	allDiff($[I_1, I_3]$)
new_int($I_3, 1, 9$)	int_array_plus($\boxed{[I_3, I_4, I_5]}, 12$)	allDiff($\boxed{[I_3, I_4, I_5]}$)
new_int($I_4, 1, 9$)	int_array_plus($[I_2, I_4, I_6], 19$)	allDiff($[I_2, I_4, I_6]$)
new_int($I_5, 1, 9$)	int_array_plus($[I_6, I_7], 3$)	allDiff($[I_6, I_7]$)
new_int($I_6, 1, 9$)	int_array_plus($[I_5, I_7], 4$)	allDiff($[I_5, I_7]$)
new_int($I_7, 1, 9$)		

Example: Kakuro



new_int(I_1, 1, 9)	int_array_plus([I_1, I_2], 13)	allDiff([I_1, I_2])
new_int(I_2, 1, 9)	int_array_plus([I_1, I_3], 5)	allDiff([I_1, I_3])
new_int(I_3, 1, 9)	int_array_plus([I_3, I_4, I_5], 12)	allDiff([I_3, I_4, I_5])
new_int(I_4, 1, 9)	int_array_plus([I_2, I_4, I_6], 19)	allDiff([I_2, I_4, I_6])
new_int(I_5, 1, 9)	int_array_plus([I_6, I_7], 3)	allDiff([I_6, I_7])
new_int(I_6, 1, 9)	int_array_plus([I_5, I_7], 4)	allDiff([I_5, I_7])
new_int(I_7, 1, 9)		

CEP is similar to Backbones

Backbones are about detecting variables which take fixed values in all solutions

$$\varphi \models x = 1$$
$$\varphi \models x = 0$$

CEP is also about detecting equations between variables which take fixed values in all solutions

$$\varphi \models x = 1$$
$$\varphi \models x = 0$$
$$\varphi \models x = y$$
$$\varphi \models x = -y$$

Backbones using SAT

Assume φ with n=5 variables

	x_1	x_2	x_3	x_4	x_5	
θ_1	1	1	0	0	1	$\varphi_0 = \varphi$
θ_2	1	0	0	1	0	$\varphi_1 = \varphi_0 \wedge \neg\theta_1$
θ_3			*unsat*			$\varphi_2 = \varphi_1 \wedge (\neg x_1 \vee x_3)$

iteration #1: sat(φ)

iteration #2: sat(φ) (& diff)

iteration #3: sat(φ) (and flip at least one that didn't flip yet)

At most n sat calls;
Incremental;
Only the last call is unsat.

Backbones for Equality (CEP)

Essentially the same; Define

$$\varphi' = \varphi \wedge \{\ e_{ij} \leftrightarrow (x_i \leftrightarrow x_j) \mid 0 \leq i < j \leq n\ \}$$

and then apply a backbone algorithm

But, we have added $O(n^2)$ new variables (???)

Backbones for Equality (CEP)

	x_1	x_2	x_3	x_4	x_5	e_{12}	e_{13}	e_{14}	e_{15}	e_{23}	e_{24}	e_{25}	e_{34}	e_{35}	e_{45}	
θ_1	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	$\varphi_0 = \varphi$
θ_2	1	0	0	1	0	0	0	1	0	1	0	1	0	1	0	$\varphi_1 = \varphi_0 \wedge \neg\theta_1$
θ_3	1	0	0	0	1	0	0	0	1	1	1	0	1	0	0	φ_2
θ_4			*unsat*													φ_3

$$\varphi_2 = \varphi_1 \wedge \begin{pmatrix} \neg x_1 \vee x_3 \vee e_{13} \vee \\ \neg e_{24} \vee \neg e_{25} \vee e_{45} \end{pmatrix}$$

$$\varphi_3 = \varphi_2 \wedge (\neg x_1 \vee x_3 \vee e_{13} \vee e_{45})$$

iteration #1 and #2: sat(φ) (two different assignments

iteration #3: sat(φ) (and flip at least one that didn't flip yet)

iteration #4: sat(φ) (and flip at least one that didn't flip yet)

Backbones for Equality (CEP)

n^2

?

	x_1	x_2	x_3	x_4	x_5	e_{12}	e_{13}	e_{14}	e_{15}	e_{23}	e_{24}	e_{25}	e_{34}	e_{35}	$_{45}$	
θ_1	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	φ_0
θ_2	1	0	0	1	0	0	0	1	0	1	0	1	0	1	0	φ_1
θ_3	1	0	0	0	1	0	0	0	1	1	1	0	1	0	0	φ_2
θ_4			*unsat*													φ_3

Theorem

Let φ be a CNF, X a set of n variables, and $\Theta = \{\theta_1, \ldots, \theta_m\}$ the sequence of assignments encountered by the CEP algorithm for φ and X. Then, $m \leq n + 1$.

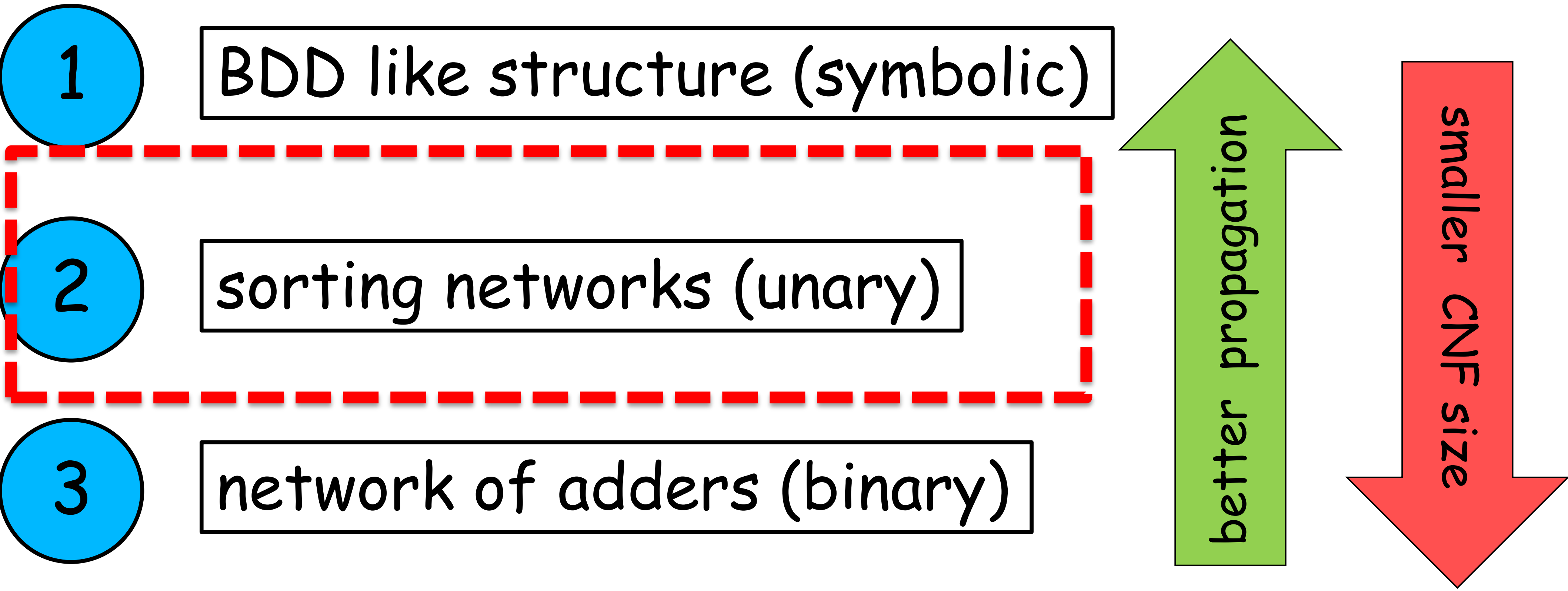
Outline

- Introduction
- BEE in a nutshell

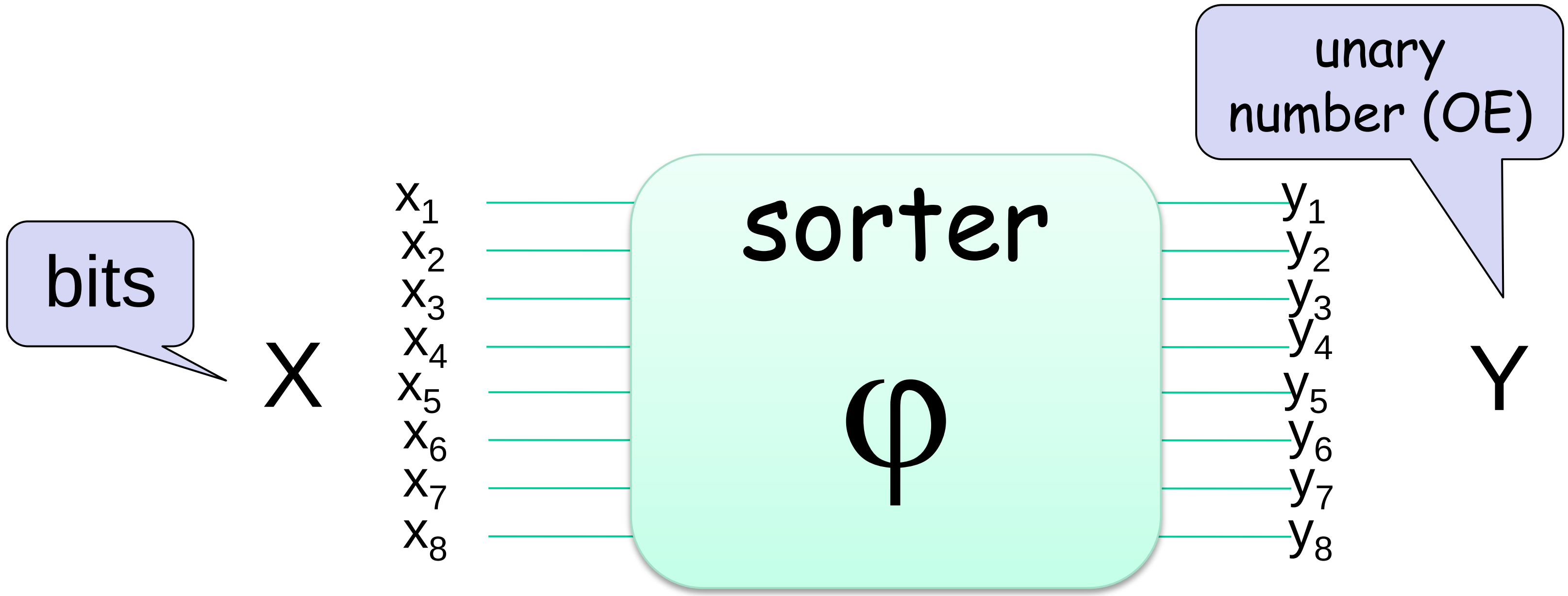
http://amit.metodi.me/research/bee/

- The "new" stuff
  - Complete Equi-Propagation
  - Cardinality Constraints in BEE
  - The binary extension of BEE

Cardinality Constraints



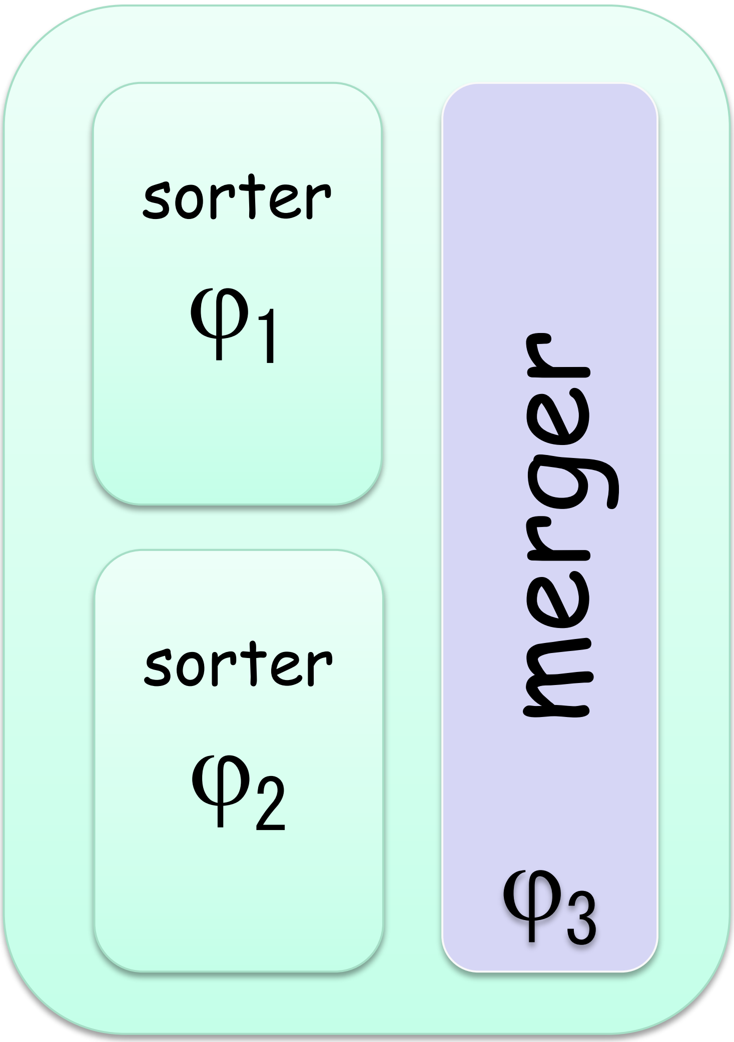
Sat encoding - cardinality constraints



$$\sum x_i \leq 4 \qquad \sum x_i \geq 4$$

$$\boxed{\varphi \wedge \neg y_5} \qquad \boxed{\varphi \wedge y_4}$$

sorting networks (defined recursively)



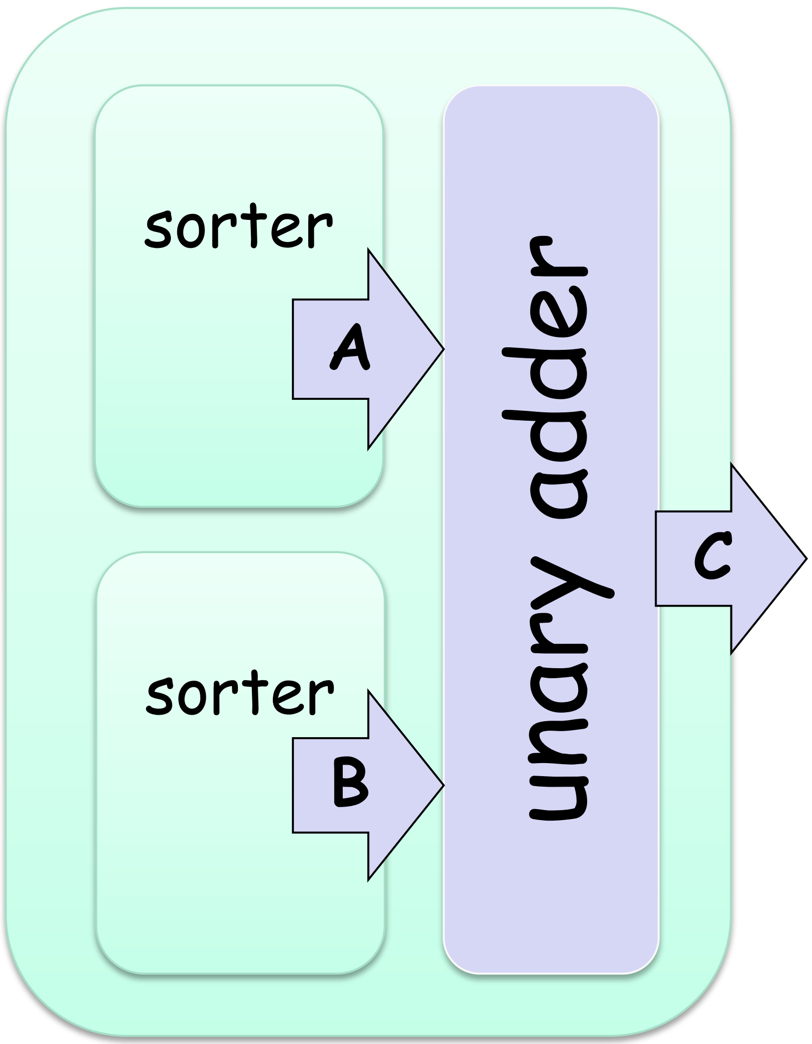
defined recursively; so it is all in the merger

Many adapt this approach applying Batcher's Odd Even Sorting Network

Another option is Parberry's "pairwise" sorting networks

The odd-even merger is basically a unary adder and consists of **O(n log n)** "comparators".

Totalizers (same but with different merger)

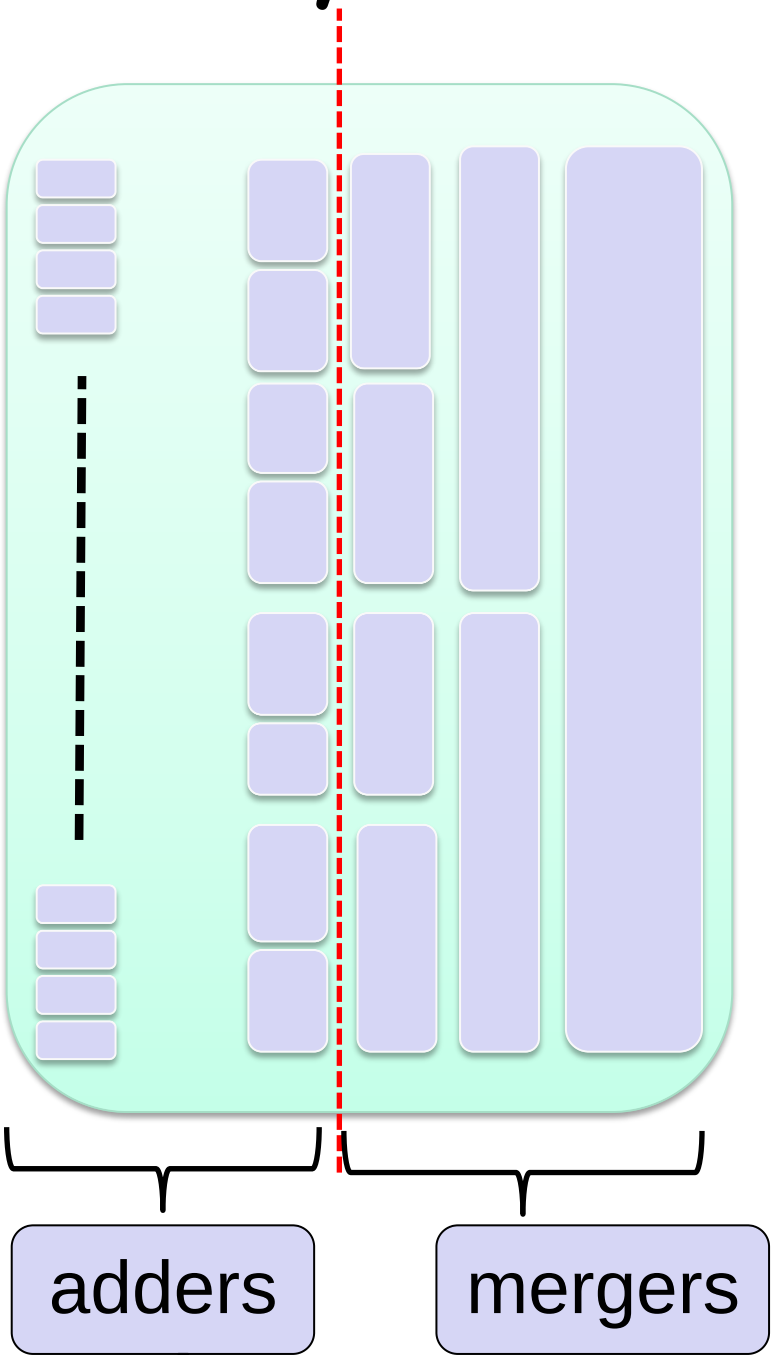


Totalizers: define the merger with a direct encoding **O(n²)** clauses

$$\bigwedge_{i,j} \quad A \geq i \ \& \ B \geq j \rightarrow C \geq i+j \qquad A \leq i \ \& \ B \leq j \rightarrow C \leq i+j$$

(direct) adders are larger than mergers but have better propagation properties

Hybrid



(direct) adders are larger than mergers but have better propagation properties

But, for small n, adders are actually smaller than mergers

Anyway, the size penalty can pay off (if under control)

While constructing, first use mergers. Then, as things get smaller, introduce adders

Experiments illustrating the advantage of the hybrid approach:

Ignasi Abio, Robert Nieuwenhuis, Albert Oliveras, Enric Rodriguez-Carbonell; A parametric approach for smaller and better encodings of cardinality constraints; CP 2013

bSettings.pl (for cardinality constraints)

```
/*
  Name: 'unaryAdderType'
  Constraint: 'int_plus'
  Possible values:
  'uadder' - (default) use O(N^2) encoding
  'merger' - decompose to comparators O(NlogN) encoding
  'hybrid' - hybrid approach:
            BEE will decide if to decompose like merger or
            encode like uadder - based on the generated CNF size.
*/
:- defineSetting(unaryAdderType,uadder).

/*
  Name: 'sumBitsDecompose'
  Constraint: 'bool_array_sum_op' / 'bool_array_pb_op'
  Possible values:
  'simple'   - (default) divide and conquer technique
  'buckets'  - split to buckets, sum each bucket
               and use linear constraints to sum buckets
  'pairwise' - pairwise sorting network
*/
:- defineSetting(sumBitsDecompose,simple).
```

Outline

- Introduction
- BEE in a nutshell

http://amit.metodi.me/research/bee/

- The “new” stuff
  - Complete Equi-Propagation
  - Cardinality Constraints in BEE
  - The binary extension of BEE

Binary Extension of BEE

Bit Blasting is obvious; But it is more about how the simplifications work

Where possible, blast into the unary core

Binary Multiplication

$$
\begin{array}{rcccccccccc}
 & & & & & x_4 & x_3 & x_2 & x_1 & x_0 \\
 & & & & \times & y_4 & y_3 & y_2 & y_1 & y_0 \\
\hline
 & & & & & z_{04} & z_{03} & z_{02} & z_{01} & z_{00} \\
 & & & & z_{14} & z_{13} & z_{12} & z_{11} & z_{10} & \\
 & & & z_{24} & z_{23} & z_{22} & z_{21} & z_{20} & & \\
 & & z_{34} & z_{33} & z_{32} & z_{31} & z_{30} & & & \\
+ & z_{44} & z_{43} & z_{42} & z_{41} & z_{40} & & & & \\
\hline
\end{array}
$$

$$z_{ij} \leftrightarrow x_i \wedge y_j$$

unary sums

Binary Multiplication (square)

$$
\begin{array}{ccccccccc}
 & & & & x_4 & x_3 & x_2 & x_1 & x_0 \\
 & & & \times & x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & z_{04} & z_{03} & z_{02} & z_{01} & z_{00} \\
 & & & z_{14} & z_{13} & z_{12} & z_{11} & \boxed{\mathbf{z_{01}}} & \\
 & & z_{24} & z_{23} & z_{22} & \mathbf{z_{12}} & \mathbf{z_{02}} & & \\
 & z_{34} & z_{33} & \mathbf{z_{23}} & \mathbf{z_{13}} & \mathbf{z_{03}} & & & \\
+ \; z_{44} & \mathbf{z_{34}} & \mathbf{z_{24}} & \mathbf{z_{14}} & \mathbf{z_{04}} & & & & \\
\hline
\end{array}
$$

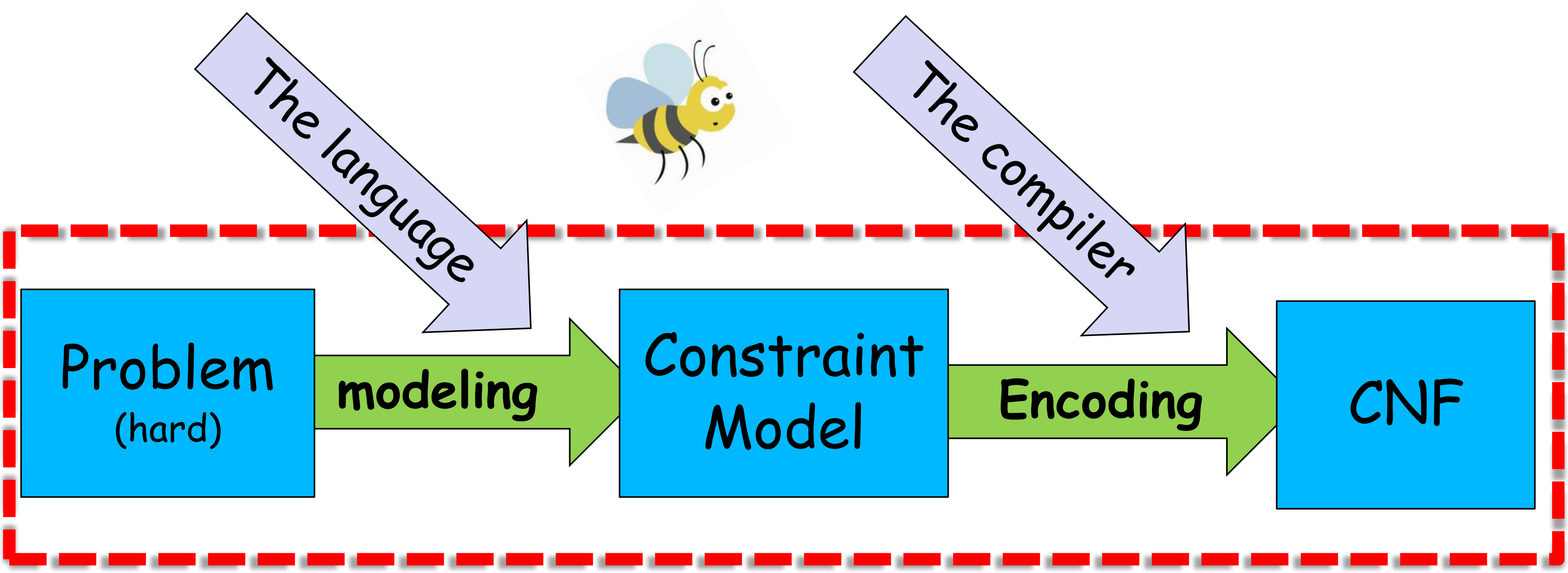
equi propagation: $z_{ij} = z_{ji}$

Binary Multiplication (square)

$$
\begin{array}{cccccccccc}
 & & & & & x_4 & x_3 & x_2 & x_1 & x_0 \\
 & & & & \times & x_4 & x_3 & x_2 & x_1 & x_0 \\
\hline
 & & & & & z_{04} & z_{03} & \boxed{z_{02}} & \boxed{z_{01}} & z_{00} \\
 & & & & z_{14} & z_{13} & z_{12} & z_{11} & \boxed{\mathbf{z_{01}}} & \\
 & & & z_{24} & z_{23} & z_{22} & \mathbf{z_{12}} & \boxed{\mathbf{z_{02}}} & & \\
 & & z_{34} & z_{33} & \mathbf{z_{23}} & \mathbf{z_{13}} & \mathbf{z_{03}} & & & \\
+ & z_{44} & \mathbf{z_{34}} & \mathbf{z_{24}} & \mathbf{z_{14}} & \mathbf{z_{04}} & & & & \\
\hline
\end{array}
$$

$$
\begin{array}{cccccccccc}
 & & & z_{23} & & z_{12} & & & & \\
 & z_{34} & & z_{14} & z_{13} & z_{03} & & \boxed{z_{01}} & & \\
+ & z_{44} & z_{24} & z_{33} & z_{04} & z_{22} & \boxed{z_{02}} & z_{11} & 0 & z_{00} \\
\hline
\end{array}
$$

Conclusion



- The "new" stuff
  - Complete Equi-Propagation
  - Cardinality Constraints in BEE
  - The binary extension of BEE