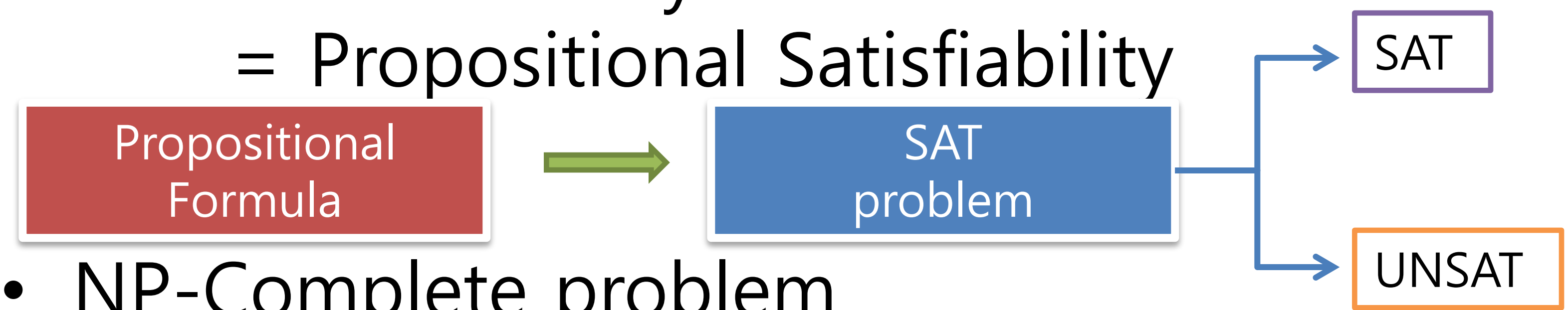
Application of Propositional Logic - How to Solve Sudoku?

Moonzoo Kim

SAT Basics (1/3)

- SAT = Satisfiability
  = Propositional Satisfiability



- NP-Complete problem
  - We can use SAT solver for many NP-complete problems
    - Hamiltonian path
    - 3 coloring problem
    - Traveling sales man's problem
- Recent interest as a verification engine

SAT Basics (2/3)

- A set of propositional variables and Conjunctive Normal Form (CNF) clauses involving variables
  - $(x_1 \vee x_{2'} \vee x_3) \wedge (x_2 \vee x_{1'} \vee x_4)$
  - x_1, x_2, x_3 and x_4 are variables (true or false)
- Literals: Variable and its negation
  - x_1 and x_1'
- A clause is satisfied if one of the literals is true
  - x_1=true satisfies clause 1
  - x_1=false satisfies clause 2
- Solution: An assignment/interpretation/model that satisfies all clauses

SAT Basics (3/3)

- DIMACS SAT Format
  - Ex. $(x_1 \vee x_2' \vee x_3)$

    $\wedge\ (x_2 \vee x_1' \vee x_4)$

```
p cnf 4 2
1 -2 3 0
2 -1 4 0
```

Model/
solution

x_1	x_2	x_3	x_4	Formula
T	T	T	T	T
T	T	T	F	T
T	T	F	T	T
T	T	F	F	T
T	F	T	T	T
T	F	T	F	F
T	F	F	T	T
T	F	F	F	F
F	T	T	T	T
F	T	T	F	T
F	T	F	T	F
F	T	F	F	F
F	F	T	T	T
F	F	T	F	T
F	F	F	T	T
F	F	F	F	T

DPLL (Davis-Putnam-Logemann-Lveland) Algorithm

```
/* The Quest for Efficient Boolean Satisfiability Solvers
 *  by L.Zhang and S.Malik, Computer Aided Verification 2002 */
DPLL(a formula φ, assignment) {
    necessary = deduction(φ, assignment);
    new_asgnment = union(necessary, assignment);
    if (is_satisfied(φ, new_asgnment))
        return SATISFIABLE;
    else if (is_conflicting(φ, new_asgnmnt))
        return UNSATISFIABLE;
    var = choose_free_variable(φ, new_asgnmnt);
    asgn1 = union(new_asgnmnt, assign(var, 1));
    if (DPLL(φ, asgn1) == SATISFIABLE)
        return SATISFIABLE;
    else {
        asgn2 = union (new_asgnmnt, assign(var,0));
        return DPLL (φ, asgn2);
    }
}
```

Example

$$\{p \lor r\} \land \{\neg p \lor \neg q \lor r\} \land \{p \lor \neg r\}$$

$p=T$

$p=F$

$$\{T \lor r\} \land \{\neg T \lor \neg q \lor r\} \land \{T \lor \neg r\}$$

$$\{F \lor r\} \land \{\neg F \lor \neg q \lor r\} \land \{F \lor \neg r\}$$

SIMPLIFY

SIMPLIFY

$$\{\neg q, r\}$$

$$\{r\} \land \{\neg r\}$$

SIMPLIFY

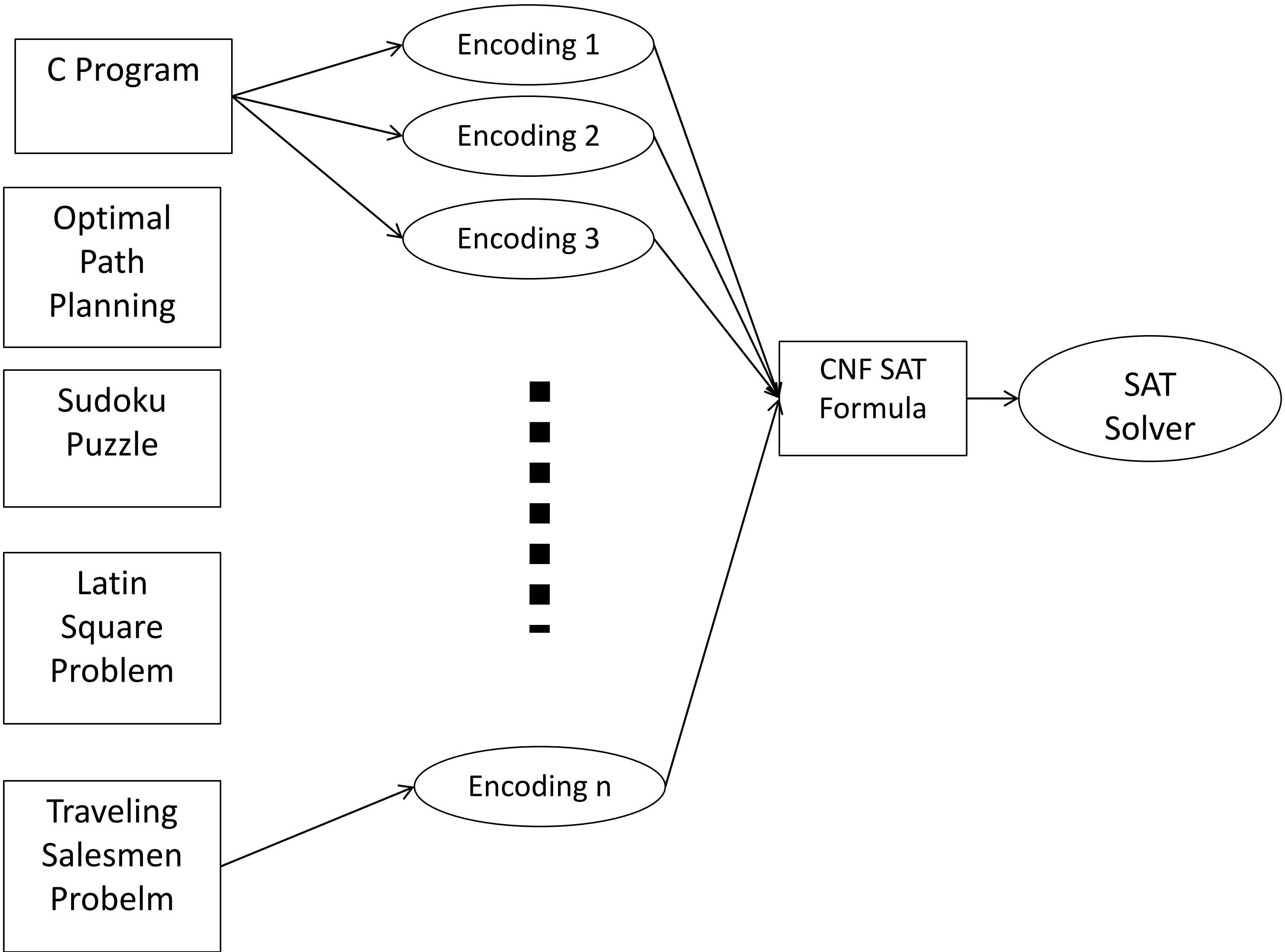
$$\{\}$$

SAT Solvers

- Most SAT solvers receives DIMACS CNF formulas
- Dozens of industry-strength SAT solvers available
  - MiniSAT
  - PicoSAT
  - SAT4J
  - borg-sat
  - clasp
  - sathys
  - tts
  - …

http://www.cril.univ-artois.fr/SAT11/

Solving Various Problems using SAT Solver

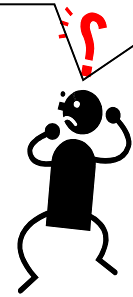


What is Sudoku ?

Problem

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

Given a problem, the objectvie is to find a satisfying assignment w.r.t. Sudoku rules.



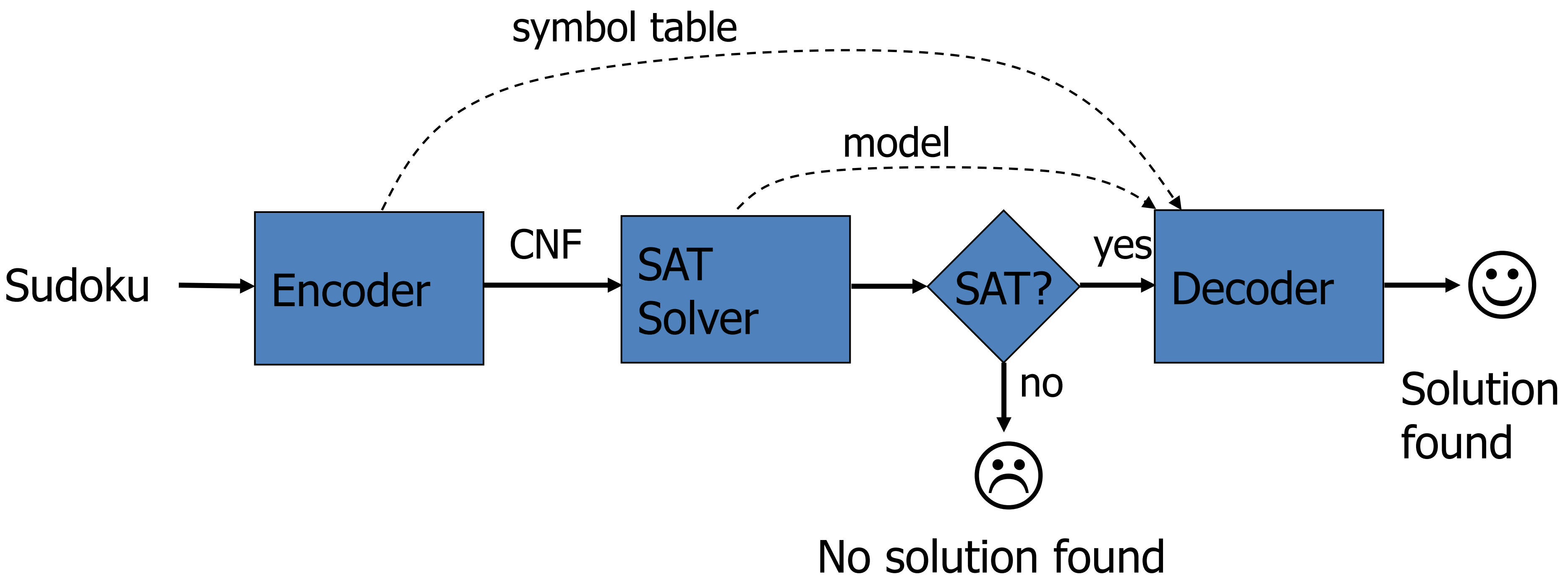
Solution

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

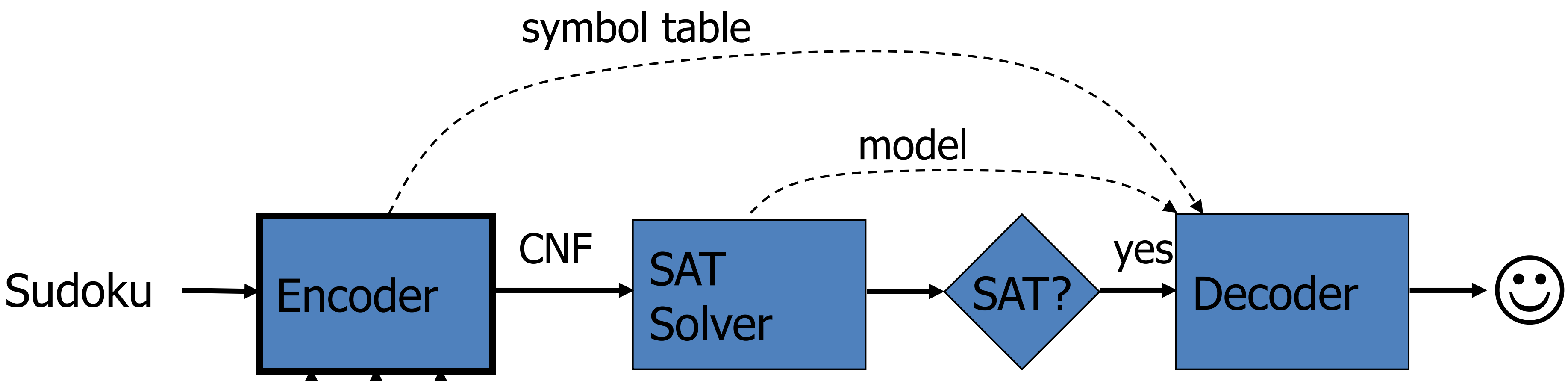
Sodoku rules

- ✓ There is a number in each **cell**.
- ✓ A number appears once in each **row**.
- ✓ A number appears once in each **column**.
- ✓ A number appears once in each **block**.

Sudoku as SAT Problem



Previous Encodings



Minimal encoding [Lynce & Ouaknine, 2006]

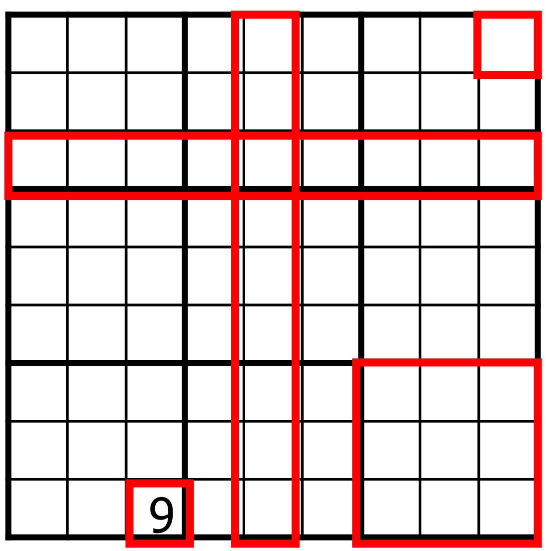
Extended encoding [Lynce & Ouaknine, 2006]

Efficient encoding [Weber, 2005]

Encoding

- Kowledge compilation into a **target language**
  problem knowlege ➡ CNF

- Knowlede about Sudoku



A number appears once in each **cell**

A number appears once in each **row**

A number appears once in each **col**

A number appears once in each **block**

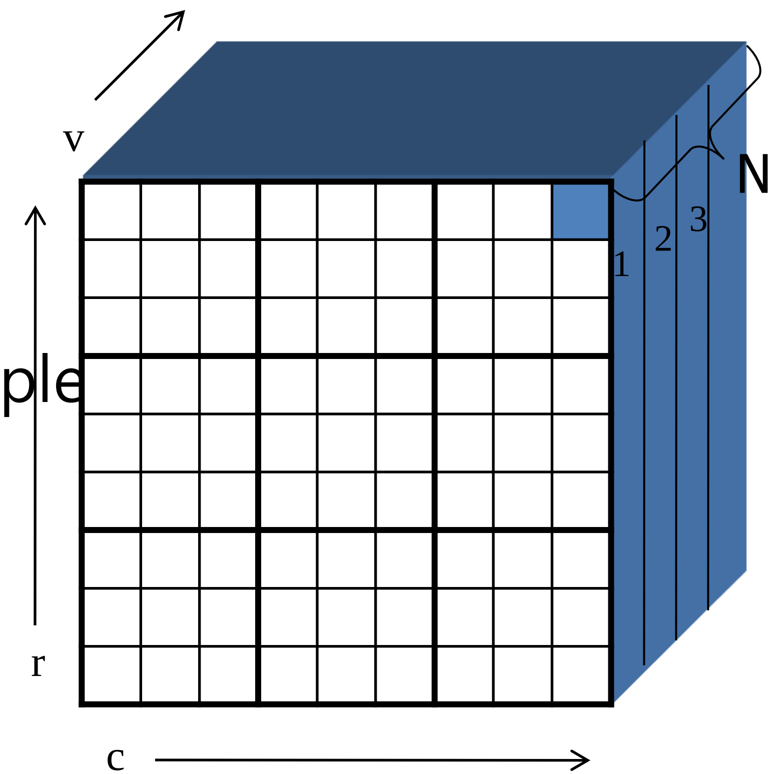
rules ➡ **CNF**

A pre-assigned number

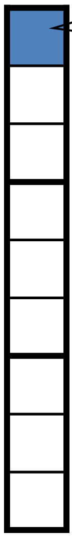
facts ➡ **CNF**

Variables

- Each cell has one number from 1..N
  - [1,1]=1 or [1,1]=2 or …… or [1,1]=N
  - Each cell needs N boolean variables to consider all cases
- Total number of variables
  - N^3
- Boolean variable name as a triple
  - (r,c,v) (i.e., x_{rcv}) iff [r,c] $=$ v
  - $\neg$(r,c,v) (i.e.,$\neg x_{rcv}$) iff [r,c] $\neq$ v



Cell Rule ➔ CNF



A number appears once in each **cell**

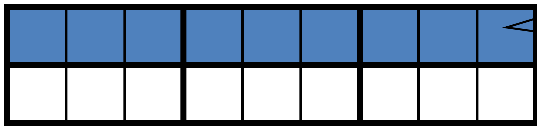
There is **at least** one number in each cell (definedness)

$$Cell_d = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \vee_{v=1}^{N} (r,c,v)$$

There is **at most** one number in each cell (uniqueness)

$$Cell_u = \wedge_{r=1}^{N} \wedge_{c=1}^{N} \wedge_{v_i=1}^{N-1} \wedge_{v_j=v_i+1}^{N} \neg((r,c,v_i) \wedge (r,c,v_j))$$

Row Rule → CNF



A number appears once in each **row**

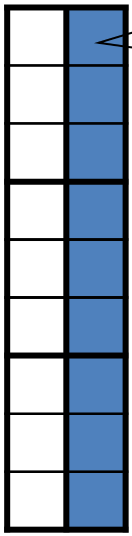
Each number appears **at least** once in each row (definedness)

$$Row_d = \wedge_{r=1}^{N} \wedge_{v=1}^{N} \vee_{c=1}^{N} (r,c,v)$$

Each number appears **at most** once in each row (uniqueness)

$$Row_u = \wedge_{r=1}^{N} \wedge_{v=1}^{N} \wedge_{c_i=1}^{N-1} \wedge_{c_j=c_i+1}^{N} \neg((r,c_i,v) \wedge (r,c_j,v))$$

Column Rule ➔ CNF



A number appears once in each **column**

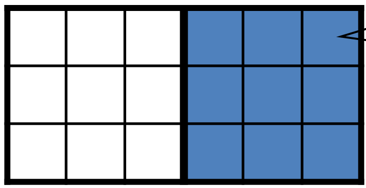
Each number appears **at least once** in each column (definedness)

$$Col_d = \wedge_{c=1}^{N} \wedge_{v=1}^{N} \vee_{r=1}^{N} (r,c,v)$$

Each number appears **at most** once in each column (uniqueness)

$$Col_u = \wedge_{c=1}^{N} \wedge_{v=1}^{N} \wedge_{r_i=1}^{N-1} \wedge_{r_j=r_i+1}^{N} \neg((r_i,c,v) \wedge (r_j,c,v))$$

Block Rule → CNF



A number appears once in each **block**

Each number appears **at least** once in each block (definedness)

$$Block_d = \wedge_{r_{offs}=1}^{subN} \wedge_{c_{offs}=1}^{subN} \wedge_{v=1}^{N} \vee_{r=1}^{subN} \vee_{c=1}^{subN} (r_{offs} * subN + r, c_{offs} * subN + c, v)$$

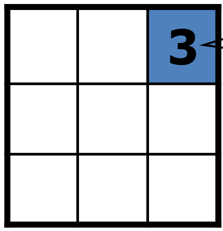
Each number appears **at most** once in each block (uniqueness)

$$Block_u = \wedge_{r_{offs}=1}^{subN} \wedge_{c_{offs}=1}^{subN} \wedge_{v=1}^{N} \wedge_{r=1}^{N} \wedge_{c=r+1}^{N}$$

$$\neg((r_{offs} * subN + (r \bmod subN), c_{offs} * subN + (r \bmod subN), v)$$

$$\wedge (r_{offs} * subN + (c \bmod subN), c_{offs} * subN + (c \bmod subN), v))$$

Pre-Assigned Fact ➔ CNF



A pre-assigned number

As a constant; the number is never changed

It can be represented as a **unit clause**

$$Assigned = \wedge_{i=1}^{k} \{(r, c, a) \mid \exists_{1 \le a \le N} \bullet [r, c] = a\}$$

where k is a number of pre-assigned numbers

Previous Encodings

Minimal encoding [Lynce & Ouaknine, 2006]

$$\phi = Cell_d \cup Row_u \cup Col_u \cup Block_u \cup Assigned$$

<u>sufficient to characterize the puzzle</u>

Extended encoding [Lynce & Ouaknine, 2006]

$$\phi = Cell_d \cup Cell_u \cup Row_d \cup Row_u \cup Col_d \cup Col_u \cup Block_d \cup Block_u \cup Assigned$$

<u>minimal encoding with redundant clauses</u>

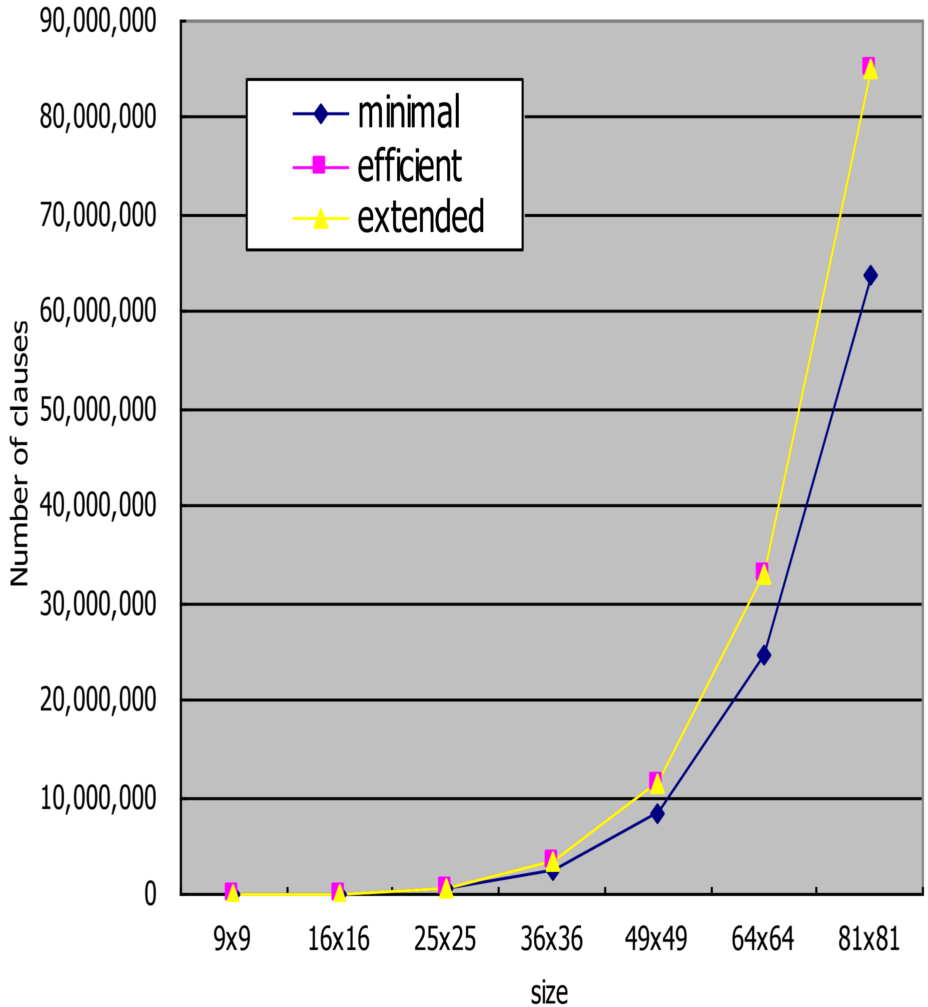
Efficient encoding [Weber, 2005]

$$\phi = Cell_d \cup Cell_u \cup Row_u \cup Col_u \cup Block_u \cup Assigned$$

<u>between minimal encoding and extended encoding</u>

Exponential Growth in Clauses

size	minimal	efficient	extended
9x9	8829	11745	11988
16x16	92416	123136	123904
25x25	563125	750625	752500
36x36	2450736	3267216	3271104
49x49	8473129	11296705	11303908
64x64	24776704	33034240	33046528
81x81	63779481	85037121	85056804



Experimental Results

		minimal encoding			efficient encoding			extended encoding		
size	level	vars	clauses	time	vars	clauses	time	vars	clauses	time
9x9	easy	729	8854	0.00	729	11770	0.00	729	12013	0.00
9x9	hard	729	8859	0.00	729	11775	0.00	729	12018	0.00
16x16	easy	4096	92520	0.10	4096	123240	0.09	4096	124008	0.01
16x16	hard	4096	92514	0.46	4096	123234	0.21	4096	124002	0.01
25x25	easy	15625	563417	9.07	15625	750917	17.48	15625	752792	0.07
25x25	hard	15625	563403	time	15625	750903	time	15625	752778	0.21
36x36	easy	46656	2451380	time	46656	3267860	time	46656	3271748	0.50
36x36	hard	46656	2451400	time	46656	3267880	time	46656	3271768	0.67
49x49	easy	117649	8474410	time	117649	11297986	time	117649	11305189	1.47
64x64	easy	262144	24779088	stack	262144	33036624	stack	262144	33048912	stack
81x81	easy	531441	63783464	stack	531441	85041104	stack	531441	85060787	stack

Experimental Results

		minimal encoding			efficient encoding			extended encoding		
size	level	vars	clauses	time	vars	clauses	time	vars	clauses	time
9x9	easy	729	8854	0.00	729	11770	[illegible]	29	12013	0.00
9x9	hard	729	8859	0.00	729	11775	[illegible]	29	12018	0.00
16x16	easy	4096	92520	0.10	4096	123240	[illegible]	96	124008	0.01
16x16	hard	4096	92514	0.46	4096	[illegible]	[illegible]	[illegible]	24002	0.01
25x25	easy	15625	563417	9.07	15625	750917	17.48	15625	752792	0.07
25x25	hard	15625	563403	time	15625	750903	time	15625	752778	0.21
36x36	easy	46656	2451380	[illegible]	5656	3267860	time	46656	3271748	0.50
36x36	hard	46656	2451400	[illegible]	5656	3267880	time	46656	3271768	0.67
49x49	easy	117649	8474410	[illegible]	7649	11297986	time	117649	11305189	1.47
64x64	easy	262144	[illegible]	[illegible]	[illegible]	6624	stack	262144	33048912	stack
81x81	easy	531441	[illegible]	stack	531441	[illegible]	stack	531441	85060787	stack

Solution found

No solution found