

Building a Fast CSP Solver Based on SAT

Neng-Fa Zhou

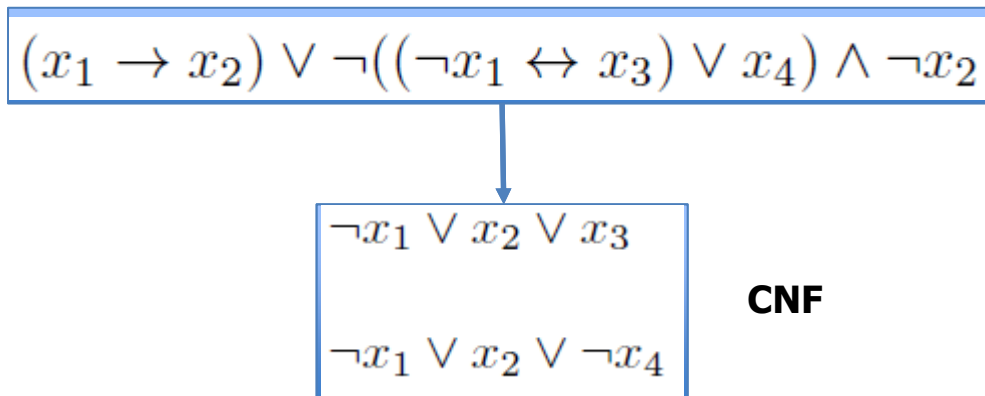
CUNY Brooklyn College & Graduate Center

Outline

- SAT, SAT solving, and SAT encodings
- Picat: a modeling language for SAT
- PicatSAT: Compiling CSP to SAT
 - Compiling arithmetic constraints
 - Decomposing global constraints
- Experimental results

The Satisfiability Problem (SAT)

Given a Boolean formula, the SAT problem is to determine if the formula is satisfiable. If yes, it finds an assignment for the variables that makes the formula satisfiable.



G.S. Tseytin: On the complexity of derivation in propositional calculus, 1969.

SAT Solving (CDCL)

1. *Choice*: Assign a value to a selected variable.
2. *Unit propagation*: Use this assignment to determine values for the other variables.
3. *Backjump*: If a conflict is found, add the negation of the conflict-causing clause as a new clause and backtrack to the choice that made the conflict occur.
4. Continue from step 1.

- Martin Davis, Hilary Putnam: A Computing Procedure for Quantification Theory, 1960.
- Martin Davis, George Logemann, Donald Loveland: A Machine Program for Theorem Proving, 1962.
- Joao Marques-Silva, Ines Lynce and Sharad Malik: Conflict-Driven Clause Learning SAT Solvers, 2009.

SAT Encodings

$$X :: \{a_1, a_2, \dots, a_n\}$$

- Direct encoding

$$\begin{aligned} B_i &\leftrightarrow X = a_i \\ B_1 \vee B_2 \vee \dots \vee B_n \\ \text{at_most_one}([B_1, B_2, \dots, B_n]) \end{aligned}$$

- Order encoding

$$B_i \leftrightarrow X \leq a_i$$

- Log encoding (sign-and-magnitude)

$$\begin{aligned} X.m &= \langle B_{k-1} \dots B_1 B_0 \rangle \\ X.s &= 0 \text{ or } 1 \end{aligned}$$

- Johan de Kleer: A Comparison of ATMS and CSP Techniques, 1989.
- James M. Crawford, Andrew B. Baker: Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems, 1994.
- Kazuo Iwama, Shuichi Miyazaki: SAT - Variable Complexity of Hard Combinatorial Problems, 1994.

CSP Solvers based on SAT

- BEE (order encoding)
- FznTini (log encoding)
- meSAT (order and direct encodings)
- PicatSAT (log and direct encodings)
- Savile Row (order and direct encodings)
- Sugar (order encoding) and its successors

- Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, Mutsunori Banbara: Compiling finite linear CSP into SAT, 2009.
- Jinbo Huang: Universal Booleanization of Constraint Models, 2008.
- Amit Metodi, Michael Codish: Compiling finite domain constraints to SAT with BEE, 2012.
- Mirko Stojadinovic, Filip Maric: meSAT - multiple encodings of CSP to SAT, 2014.
- Neng-Fa Zhou and Hakan Kjellerstrand: Optimizing SAT Encodings for Arithmetic Constraints, 2017.

Outline

- SAT, SAT solving, and SAT encodings
- Picat: a modeling language for SAT
- PicatSAT: Compiling CSP to SAT
 - Compiling arithmetic constraints
 - Decomposing global constraints
- Experimental results

Picat



– picat-lang.org

- Pattern-matching, Intuitive, Constraints, Actors, Tabling

– Core logic programming concepts

- Logic variables (arrays and maps are terms)
- Implicit pattern-matching and explicit unification
- Explicit non-determinism

– Language constructs for scripting and modeling

- Functions, loops, list comprehensions, and assignments

– Facilities for combinatorial search

- Tabling for dynamic programming and planning
- The `cp`, `sat`, `mip`, and `smt` modules for CSPs

- Neng-Fa Zhou, Hakan Kjellerstrand, Jonathan Fruhman: Constraint Solving and Planning with Picat, 2015.

Picat's Constraint Modules



```
import sat.  import cp.  import mip.  import smt.
```

- Constraints

- Domain

- $X :: \text{Domain}$
 - $X \text{ notin Domain}$

- Arithmetic

- $(X \# = Y), (X \# \neq Y), (X \# > Y), (X \# \geq Y), \dots$

- Boolean

- $(X \# /\ \ Y), (X \# \backslash / \ Y), (X \# \leq \Rightarrow Y), (X \# \Rightarrow Y), (X \# \wedge Y), (\# \sim X)$

- Table

- `table_in(VarTuple,Tuples), table_notin(VarTuple,Tuples)`

- Global

- `all_different(L), element(I,L,V), circuit(L), cumulative(...), ...`

- Solver invocation: `solve(Options,Vars)`

Modeling Sudoku in Picat

```
import sat.

sudoku(A) =>
    N = len(A),
    A :: 1..N,
    foreach(Row in 1..N)
        all_different(A[Row])
    end,
    foreach(Col in 1..N)
        all_different([A[Row,Col] : Row in 1..N])
    end,
    M = floor(sqrt(N)),
    foreach(Row in 1..M..(N-M+1), Col in 1..M..(N-M+1))
        Square = [A[Row+Dr,Col+Dc] : Dr in 0..M-1, Dc in 0..M-1],
        all_different(Square)
    end,
    solve(A).
```

A = { {_, 6, _, 1, _, 4, _, 5, _},
 {_, _, 8, 3, _, 5, 6, _, _},
 {2, _, _, _, _, _, _, _, 1},
 {8, _, _, 4, _, 7, _, _, 6},
 {_, _, 6, _, _, _, 3, _, _},
 {7, _, _, 9, _, _, _, _, 4},
 {5, _, _, _, _, _, _, _, 2},
 {_, _, 7, 2, _, 6, 9, _, _},
 {_, 4, _, 5, _, 8, _, 7, _}}.

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

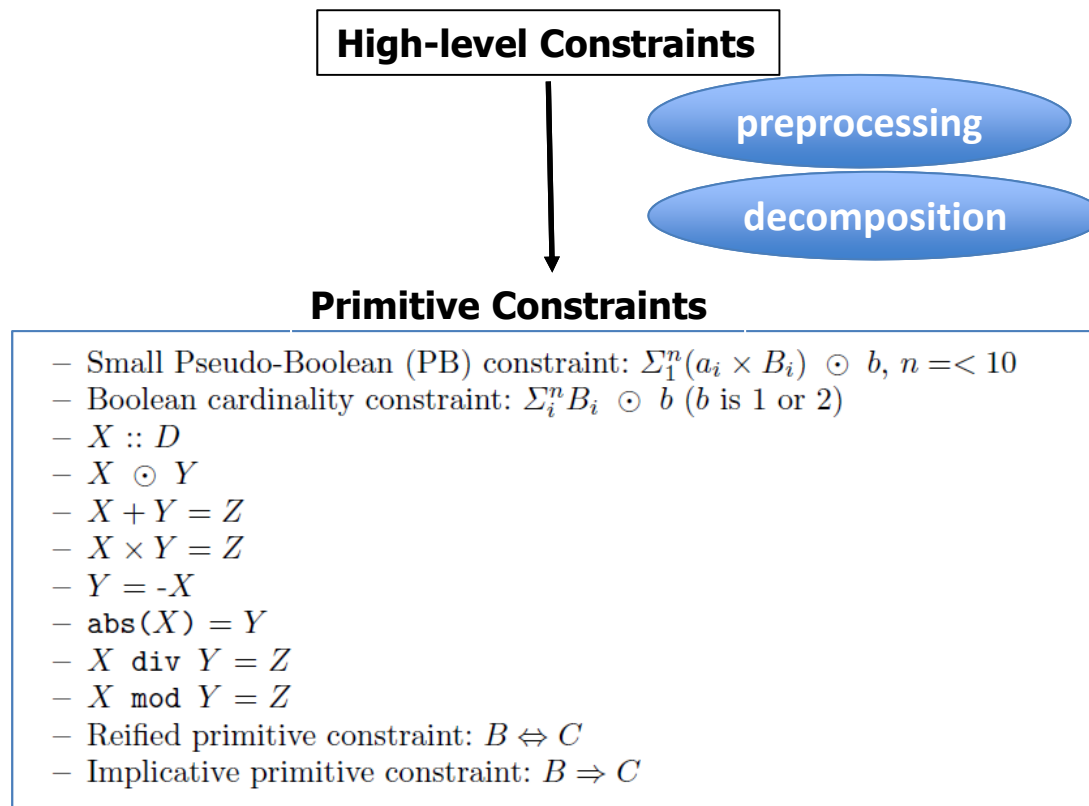
Outline

- SAT, SAT solving, and SAT encodings
- Picat: a modeling language for SAT
- **PicatSAT: Compiling CSP to SAT**
 - Compiling arithmetic constraints
 - Decomposing global constraints
- Experimental results



- A compiler that translates CSP into CNF
- Adopts log and direct encodings
- Performs numerous optimizations
 - Preprocessing constraints to exclude no-good values (CP)
 - Common subexpression elimination (language compilers)
 - Logic optimization (hardware design)
 - Compile-time equivalence reasoning
- Employs efficient decomposers for global constraints
- Offered in Picat as a constraint module, named sat
- Written in Picat with more than 10,000 LOC
- One of the top solvers in recent solver competitions

The PicatSAT Compiler



- Constraints are made to be arc-consistent or interval consistent
- No primitive constraints are duplicated
- Avoid creating large-domain variables
- Avoid creating domains with negative values

Encoding Small PB Constraints

- Espresso

% $X+Y = 2*U+V$, where $[X,Y,U,V] :: 0..1$

```
.i 4
.o 1
0000 0
0001 1
0010 1
0011 -
0100 1
0101 0
0110 1
0111 -
1000 1
1001 0
1010 1
1011 -
1100 1
1101 1
1110 0
1111 -
```



```
.i 4
.o 1
.p 5
1-00 1
-01- 1
01-0 1
110- 1
00-1 1
.e
```

```
 $\neg X \vee U \vee V$   
 $Y \vee \neg U$   
 $X \vee \neg Y \vee V$   
 $\neg X \vee \neg Y \vee U$   
 $X \vee Y \vee \neg V$ 
```

Encoding the at-most-one Constraint

- Chen's two-product encoding

	V_1	V_2	...	V_j	...	V_q
U_1	X_{11}	X_{12}	...			X_{1q}
U_2	X_{21}	X_{22}	...			X_{2q}
...						
U_i	...			X_{ij}	...	
...						
U_p	X_{p1}	X_{p2}	...			X_{pq}

$\text{amo}([X_{11}, X_{12}, \dots, X_{pq}])$

$\text{amo}([U_1, U_2, \dots, U_p])$

$\text{amo}([V_1, V_2, \dots, V_q])$

$X_{ij} \rightarrow U_i \wedge V_j$

for i in $1..p$ and j in $1..q$

Jingchao Chen: A New SAT Encoding of the At-Most-One Constraint, 2010.

Sign-and-Magnitude Log Encoding

$X :: D$

- Each domain variable is encoded as a vector of Boolean variables
 - $X.m = \langle B_{n-1}, \dots, B_1, B_0 \rangle$
 - $X.s$ is the sign bit
- No negative zero is allowed
 - $X.m = \langle 0, \dots, 0, 0 \rangle \Rightarrow X.s = 0$

Sign-and-Magnitude Log Encoding (Example)

$x :: [-2, -1, 1, 2]$

$x.m = \langle x_1, x_0 \rangle$

$x.s = s$

- Naïve Encoding

$\neg s \vee \neg x_1 \vee \neg x_0$	$(x \neq -3)$
$\neg s \vee x_1 \vee x_0$	$(x \neq -0)$
$s \vee x_1 \vee x_0$	$(x \neq 0)$
$s \vee \neg x_1 \vee \neg x_0$	$(x \neq 3)$

- Optimized Encoding
(Using Espresso)

$x_0 \vee x_1$
 $\neg x_0 \vee \neg x_1$

Breaking Arithmetic Constraints

- Combine power-of-2 terms

$$2^{k-1} \times X_{k-1} + \dots + 2^0 \times X_0 \longrightarrow \underline{X} \quad :: \quad 0..2^k - 1$$

- Move out non-linear terms
- Factor out terms with common coefficients
- Decompose the constraint (Huffman coding)

```
decompose( $a_1 \times X_1 + a_2 \times X_2 + \dots + a_n \times X_n \odot b$ ):  
  add all the terms  $a_i \times X_i$  into a priority queue  $Q$   
  while the constraint is not primitive:  
    remove two terms  $a_i \times X_i$  and  $a_j \times X_j$  from  $Q$   
    where  $a_i = a_j$ , and  
     $X_i$  and  $X_j$  have the smallest domains  
    post  $T = X_i + X_j$   
    add the term  $a_i \times T$  into  $Q$   
  post the primitive constraint
```

The Comparison Constraint: $X \geq Y$

- Signed comparison

$$\begin{aligned} &X.s = 0 \wedge Y.s = 1 \vee \\ &X.s = 1 \wedge Y.s = 1 \Rightarrow X.m \leq Y.m \vee \\ &X.s = 0 \wedge Y.s = 0 \Rightarrow X.m \geq Y.m \end{aligned}$$

- Unsigned comparison

$$X.m = \langle X_{n-1}X_{n-2} \dots X_1X_0 \rangle, Y.m = \langle Y_{n-1}Y_{n-2} \dots Y_1Y_0 \rangle$$

$$T_0 \Leftrightarrow (X_0 \geq Y_0)$$

$$T_1 \Leftrightarrow (X_1 > Y_1) \vee (X_1 = Y_1 \wedge T_0)$$

$$\vdots$$

$$T_{n-1} \Leftrightarrow (X_{n-1} > Y_{n-1}) \vee (X_{n-1} = Y_{n-1} \wedge T_{n-2})$$

The Addition Constraint: $X+Y = Z$

- Unsigned addition (ripple-carry adders)

$$\begin{array}{r} X_{n-1} \dots X_1 X_0 \\ + Y_{n-1} \dots Y_1 Y_0 \\ \hline Z_n Z_{n-1} \dots Z_1 Z_0 \end{array} \quad \text{Carriers are used}$$

- Signed addition

$$\begin{aligned} X.s = 0 \wedge Y.s = 0 &\Rightarrow Z.s = 0 \wedge X.m + Y.m = Z.m \\ X.s = 1 \wedge Y.s = 1 &\Rightarrow Z.s = 1 \wedge X.m + Y.m = Z.m \\ X.s = 0 \wedge Y.s = 1 \wedge Z.s = 1 &\Rightarrow X.m + Z.m = Y.m \\ X.s = 0 \wedge Y.s = 1 \wedge Z.s = 0 &\Rightarrow Y.m + Z.m = X.m \\ X.s = 1 \wedge Y.s = 0 \wedge Z.s = 0 &\Rightarrow X.m + Z.m = Y.m \\ X.s = 1 \wedge Y.s = 0 \wedge Z.s = 1 &\Rightarrow Y.m + Z.m = X.m \end{aligned}$$

The Full Adder

$$X_i + Y_i + C_{in} = C_{out}Z_i$$

$$\begin{aligned} &X_i \vee \neg Y_i \vee C_{in} \vee Z_i \\ &X_i \vee Y_i \vee \neg C_{in} \vee Z_i \\ &\neg X_i \vee \neg Y_i \vee C_{in} \vee \neg Z_i \\ &\neg X_i \vee Y_i \vee \neg C_{in} \vee \neg Z_i \\ &\neg X_i \vee C_{out} \vee Z_i \\ &X_i \vee \neg C_{out} \vee \neg Z_i \\ &\neg Y_i \vee \neg C_{in} \vee C_{out} \\ &Y_i \vee C_{in} \vee \neg C_{out} \\ &\neg X_i \vee \neg Y_i \vee \neg C_{in} \vee Z_i \\ &X_i \vee Y_i \vee C_{in} \vee \neg Z_i \end{aligned}$$

A Specialized Adder for $X+1 = Y$

$$\begin{array}{rcccccc} & X_{i+4} & X_{i+3} & X_{i+2} & X_{i+1} & X_i & \\ + & & & & & C_{in} & \\ \hline C_{out} & Y_{i+4} & Y_{i+3} & Y_{i+2} & Y_{i+1} & Y_i & \end{array}$$

- Use full adders
 - Use one carry variable for each bit position
 - Need 50 clauses
- Use specialized adders
 - Use one carry variable for every 5 bit positions
 - Need 25 clauses

The Multiplication Constraint: $X * Y = Z$

- The Shift-and-Add Algorithm

$$X.m * Y.m = Z.m \quad X.m = \langle X_{n-1}, \dots, X_1, X_0 \rangle$$

$$X_0 = 0 \Rightarrow S_0 = 0$$

$$X_0 = 1 \Rightarrow S_0 = Y$$

$$X_1 = 0 \Rightarrow S_1 = S_0$$

$$X_1 = 1 \Rightarrow S_1 = (Y \ll 1) + S_0$$

$\vdots$

$$X_i = 0 \Rightarrow S_i = S_{i-1}$$

$$X_i = 1 \Rightarrow S_i = (Y \ll i) + S_{i-1}$$

$\vdots$

$$X_{n-1} = 0 \Rightarrow S_{n-1} = S_{n-2}$$

$$X_{n-1} = 1 \Rightarrow S_{n-1} = (Y \ll (n-1)) + S_{n-2}$$

$$Z = S_{n-1}$$

Equivalence Reasoning

- Equivalence reasoning is an optimization that reasons about a possible value for a Boolean variable or the relationship between two Boolean variables.

$$X = \text{abs}(Y) \quad \Rightarrow \quad X.m = Y.m, X.s = 0$$

$$X = -Y \quad \Rightarrow \quad X.m = Y.m, X.s = Y.s = 0 \rightarrow X.m = 0$$

$$X = Y \bmod 2^K \Rightarrow X_0 = Y_0, X_1 = Y_1, \dots, X_{k-1} = Y_{k-1}$$

$$X = Y \text{ div } 2^K \Rightarrow X_0 = Y_K, X_1 = Y_{K+1}, \dots$$

- No clauses are needed to encode $X = \text{abs}(Y)$.

Constant Propagation on $X+Y = Z$

$$\boxed{X_i + Y_i = C_{out}Z_i}$$

Rule-1 : $X_i = 0 \Rightarrow C_{out} = 0 \wedge Z_i = Y_i$.

Rule-2 : $X_i = 1 \Rightarrow C_{out} = Y_i \wedge Z_i = \neg Y_i$.

Rule-3 : $Z_i = 0 \Rightarrow C_{out} = X_i \wedge X_i = Y_i$

Rule-4 : $Z_i = 1 \Rightarrow C_{out} = 0 \wedge X_i = \neg Y_i$.

- Example-1

$$\begin{array}{rcccc}
 & X_2 & X_1 & X_0 & \\
 + & 1 & 0 & 0 & \\
 \hline
 Z_3 & Z_2 & Z_1 & Z_0 &
 \end{array}
 \quad \longrightarrow \quad
 \begin{array}{l}
 X_0 = Z_0 \\
 X_1 = Z_1 \\
 \neg X_2 = Z_2 \\
 X_2 = Z_3
 \end{array}$$

- Example-2

$$\begin{array}{rcccc}
 & X_2 & X_1 & X_0 & \\
 + & Y_2 & Y_1 & Y_0 & \\
 \hline
 1 & 0 & 1 & 1 &
 \end{array}
 \quad \longrightarrow \quad
 \begin{array}{l}
 \neg X_0 = Y_0 \\
 \neg X_1 = Y_1 \\
 X_2 = Y_2 \\
 X_2 = 1 \\
 Y_2 = 1
 \end{array}$$

Constant Propagation on $X * Y = Z$

$$X_0 = 0 \Rightarrow S_0 = 0$$

$$X_0 = 1 \Rightarrow S_0 = Y$$

$$X_1 = 0 \Rightarrow S_1 = S_0$$

$$X_1 = 1 \Rightarrow S_1 = (Y \ll 1) + S_0$$

$\vdots$

$$X_i = 0 \Rightarrow S_i = S_{i-1}$$

$$X_i = 1 \Rightarrow S_i = (Y \ll i) + S_{i-1}$$

$\vdots$

$$X_{n-1} = 0 \Rightarrow S_{n-1} = S_{n-2}$$

$$X_{n-1} = 1 \Rightarrow S_{n-1} = (Y \ll (n-1)) + S_{n-2}$$

$$Z = S_{n-1}$$

Rule 5 : $X_i = 0 \Rightarrow$ copy all of the bits of S_{i-1} into S_i .

Rule 6 : $X_i = 1 \Rightarrow$ copy the lowest i bits of S_{i-1} into S_i .

Rule 7 : $X.m = \langle X_{n-1} \dots X_i 0 \dots 0 \rangle \wedge X_i = 1 \Rightarrow$
 $Z_i = Y_0 \wedge Z_k = 0 \text{ for } k \in 0..(i-1).$

Outline

- SAT, SAT solving, and SAT encodings
- Picat: a modeling language for SAT
- PicatSAT: Compiling CSP to SAT
 - Compiling arithmetic constraints
 - Decomposing global constraints
- Experimental results

all_different(L)

`all_different([V1, V2, ..., Vn])`

- Standard

$V_i \neq V_j$ for $i, j = 1, \dots, n, i < j$.

- Use `at_most_one`

- Let $D = D_1 \cup D_2 \cup \dots \cup D_n$

- If $|D| > n$: $\forall_a \in D$: `at_most_one`($[V_1 = a, V_2 = a, \dots, V_n = a]$)

- If $|D| = n$: $\forall_a \in D$: `exactly_one`($[V_1 = a, V_2 = a, \dots, V_n = a]$)

- A hybrid of log and direct encodings

Table Constraints

X	Y
1	1
1	2
2	1
2	2
3	1

X ₁	X ₀	Y ₁	Y ₀
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1

- Truth table encoding

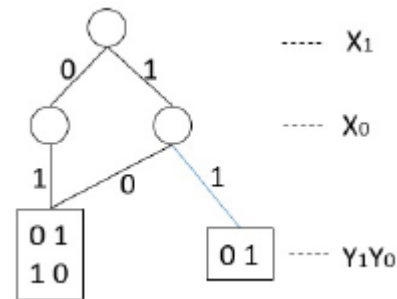
$$\neg X_1 \vee \neg X_0 \vee Y_0$$

$$X_1 \vee X_0$$

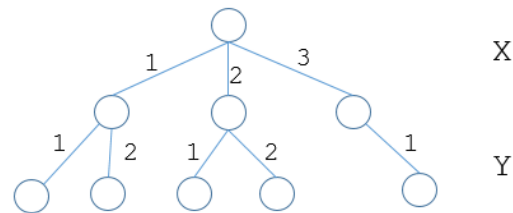
$$Y_1 \vee Y_0$$

$$\neg Y_1 \vee \neg Y_0$$

- BDD and truth table



- Trie encoding



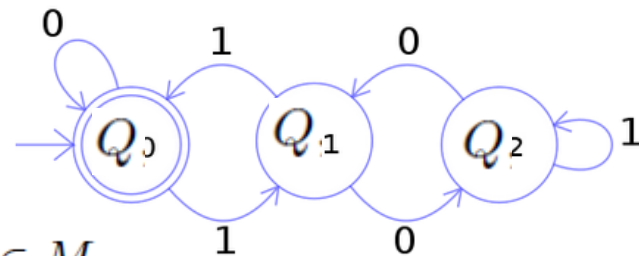
The element and regular Constraints

- $\text{element}(I, [E_1, E_2, \dots, E_n], V)$
 - If E_i 's are ground

$$(I, V) \in [(1, E_1), (2, E_2), \dots, (n, E_n)]$$
 - If at least one E_i is variable

$$\text{for each } i \in 1..n: I = i \Rightarrow V = E_i$$

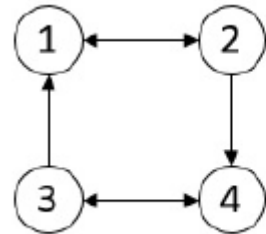
- $\text{regular}(L, Q, S, M, Q_0, F)$
 - Introduce state variables $Q_1 \dots Q_n$
 - Transition constraints: $(Q_i, L_i, Q_{i+1}) \in M$
 - Preprocessing



Gilles Pesant: A Regular Language Membership Constraint for Finite Sequences of Variables, 2004.
 Claude-Guy Quimper, Toby Walsh: Global Grammar Constraints, 2006.

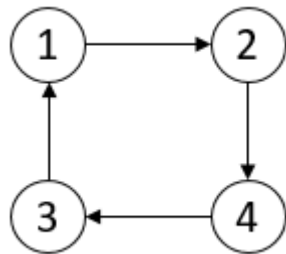
The Global Constraint circuit(G)

circuit(G)
 $G = [V_1, V_2, \dots, V_n]$

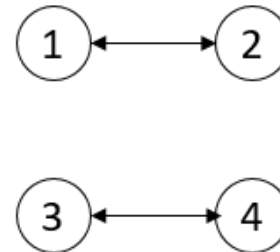


$V_1 :: [2]$
 $V_2 :: [1,4]$
 $V_3 :: [1,4]$
 $V_4 :: [3]$

$G = [2,4,1,3]$



$G = [2,1,4,3]$



- Alexander Hertel, Philipp Hertel, and Alasdair Urquhart: Formalizing dangerous SAT encodings, 2007.
- Andrew Johnson: Stedman and Erin Triples encoded as a SAT Problem, 2018.
- Fangzhen Lin and Jicheng Zhao: On tight logic programs and yet another translation from normal logic programs to propositional logic, 2003.
- Neng-Fa Zhou: In Pursuit of an Efficient SAT Encoding for the Hamiltonian Cycle Problem, 2019.

Distance Encoding for circuit

`circuit`($[V_1, V_2, \dots, V_n]$)

- Variables

- $H_{ij} = 1$ if the arc (i, j) is in the cycle.
- Use a domain variable P_i for vertex i 's position.

- Constraints

- Channeling constraints

$$\text{For each } i \in 1..n, j \in 1..n: H_{ij} \Leftrightarrow V_i = j \quad (1)$$

- Degree constraints

$$\text{For each } j \in 1..n: \sum_{i=1}^n H_{ij} = 1 \quad (2)$$

- Visit vertex 1 first: $P_i = 1$

For each $i \in 2..n$:

$$H_{1i} \Rightarrow P_i = 2 \quad (3'')$$

$$H_{i1} \Rightarrow P_i = n \quad (4'')$$

- Transition constraints

For each $i \in 2..n, j \in 2..n, i \neq j$:

$$H_{ij} \Rightarrow P_j = P_i + 1 \quad (5'')$$

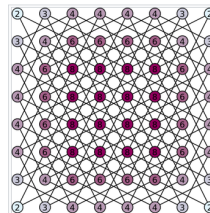
Preprocessing circuit

- HCP can be treated as a single-agent path finding problem.

– Path heuristics:



- The agent cannot reach a vertex at time t if there are no paths of length $t-1$ from vertex 1 to the vertex.
- The agent cannot occupy a vertex at time t if there are no paths of length $n-t+1$ from the vertex to vertex 1.



– The Hall's theorem

- If the start vertex 1 has exactly two neighbors, then the agent must visit one of the neighbors at time 2, and visit the other neighbor at time n .

The cumulative Constraint

$\text{cumulative}([S_1, S_2, \dots, S_n], [D_1, D_2, \dots, D_n], [R_1, R_2, \dots, R_n], \text{Limit})$

- Occupation constraints

for each time t_i and each task j :

$$O_{ij} \leftrightarrow S_j \leq t_i < S_j + D_j$$

- Resource constraints

for each time t_i :

$$\sum_{j=1}^n O_{ij} * R_j \leq \text{Limit}$$

- Time points

- Time decomposition: all the time points in the make span
- Task decomposition: only the start or end points

- Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Explaining the cumulative propagator, 2011.

Outline

- SAT, SAT solving, and SAT encodings
- Picat: a modeling language for SAT
- PicatSAT: Compiling CSP to SAT
 - Compiling arithmetic constraints
 - Decomposing global constraints
- Experimental results

Experimental Results (MiniZinc Challenge, Free Search)

Benchmark	Picat19	Picat18	or-tools	HaifCSP	Choco	Chuffed
cargo (5)	3	3	3	3	0	3
concert-hall-cap(5)	4	3	4	0	4	4
elitserien (5)	2	0	5	5	4	5
gfd-schedule (5)	5	4	5	4	0	4
largescheduling(5)	0	0	0	0	0	0
mapping (5)	3	3	4	5	2	3
neighbours (5)	4	4	2	0	1	2
on-call-rostering(5)	3	3	4	0	0	0
oocsp_racks (5)	5	5	4	5	1	5
opt-cryptanalysis(5)	5	5	0	0	0	0
proteindesign12(5)	3	0	1	1	1	3
racp (5)	2	1	3	1	1	3
rotating-workforce(5)	5	5	4	1	3	5
seat-moving (5)	5	4	3	3	2	4
soccer-computational(5)	5	5	4	5	3	5
steiner-tree (5)	3	3	4	4	2	4
team-assignment(5)	3	3	3	3	1	3
test-scheduling(5)	2	3	3	0	3	2
train (5)	2	2	2	2	2	2
vrplc (5)	3	2	4	1	0	2
<i>Total (100)</i>	67	58	62	43	30	59
Benchmark	Picat19	Picat18	or-tools	HaifCSP	Choco	Chuffed

Experimental Results (XCSP Competition, COP)

Benchmark	Picat19	Picat18	Choco	Concrete	Scop-both	Scop-order
Auction(16)	2	2	2	2	n/a	n/a
Bacp(24)	8	8	4	4	n/a	n/a
CrosswordDesign(13)	1	2	3	3	n/a	n/a
Fapp(18)	4	7	6	6	n/a	n/a
GolombRuler(13)	6	6	6	7	n/a	n/a
GraphColoring(11)	7	6	5	7	n/a	n/a
Knapsack(14)	10	6	6	6	n/a	n/a
LowAutocorrelation(14)	10	10	3	3	n/a	n/a
Mario(10)	7	7	10	10	n/a	n/a
NurseRostering(21)	2	2	2	2	n/a	n/a
PeacableArmies(14)	6	4	4	4	n/a	n/a
PizzaVoucher(10)	8	7	3	4	n/a	n/a
PseudoBoolean-opt(13)	5	5	3	3	n/a	n/a
QuadraticAssignment(19)	3	2	2	2	n/a	n/a
Rcsp(16)	16	14	12	12	n/a	n/a
Rlfap(25)	22	5	5	5	n/a	n/a
SteelMillSlab(17)	4	2	2	2	n/a	n/a
StillLife(13)	13	13	3	3	n/a	n/a
SumColoring(14)	5	5	3	3	n/a	n/a
Tal(10)	7	3	4	4	n/a	n/a
TemplateDesign(15)	12	11	11	10	n/a	n/a
TravelingTournament(14)	2	2	2	2	n/a	n/a
TravellingSalesman(12)	3	3	1	2	n/a	n/a
<i>COP Total (346)</i>	163	132	102	106	n/a	n/a
Benchmark	Picat19	Picat18	Choco	Concrete	Scop-both	Scop-order

Experimental Results (XCSP Competition, CSP)

Benchmark	Picat19	Picat18	Choco	Concrete	Scop-both	Scop-order
Bibd(12)	8	8	4	4	8	8
CarSequencing(17)	17	17	11	2	13	17
ColouredQueens(12)	4	4	3	3	3	3
Crossword(13)	4	2	5	3	3	3
Dubois(12)	12	12	7	4	12	12
Eternity(15)	7	6	7	6	6	6
Frb(16)	6	3	3	4	5	5
GracefulGraph(11)	6	6	6	6	6	6
Haystacks(10)	8	10	4	3	10	10
Langford(11)	9	9	7	3	9	9
MagicHexagon(11)	4	3	4	5	3	3
MysteryShopper(10)	10	10	10	11	10	10
PseudoBoolean-dec(13)	7	4	4	4	7	8
Quasigroups(16)	16	7	6	7	8	8
Rlfap(12)	12	12	11	8	12	12
SocialGolfers(12)	8	8	6	6	8	8
SportsScheduling(10)	5	4	3	4	4	4
StripPacking(12)	12	11	7	6	11	11
Subisomorphism(11)	7	2	7	4	3	3
<i>CSP Total (236)</i>	162	138	115	93	141	146
<i>Total (583)</i>	325	270	217	199	n/a	n/a
Benchmark	Picat19	Picat18	Choco	Concrete	Scop-both	Scop-order

Summary of PicatSAT

- Adopts log and direct encodings
- Performs numerous optimizations
 - Preprocessing constraints to exclude no-good values (CP)
 - Common subexpression elimination (language compilers)
 - Logic optimization (hardware design)
 - Compile-time equivalence reasoning
- Employs efficient decomposers for global constraints
- Available on picat-lang.org

Thanks

- **Other contributors**

- Roman Barták
- Agostino Dovier
- Jonathan Fruhman
- Sanders Hernandez
- Håkan Kjellerstrand
- Jie Mei
- Yi-Dong Shen
- Taisuke Sato

- **Funding**

- NSF CCF1018006 and CCF1618046
- PSC CUNY
- NYC Media Lab