Building a Fast CSP Solver Based on SAT

Neng-Fa Zhou

CUNY Brooklyn College & Graduate Center

N.F. Zhou, PicatSAT
• 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.

\[(x_1 \rightarrow x_2) \lor \neg((\neg x_1 \leftrightarrow x_3) \lor x_4) \land \neg x_2\]

\[\neg x_1 \lor x_2 \lor x_3\]

\[\neg x_1 \lor x_2 \lor \neg x_4\]

CNF

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.

SAT Encodings

\[ X \equiv \{a_1, a_2, \ldots, a_n\} \]

- **Direct encoding**
  
  \[ B_i \leftrightarrow X = a_i \]
  \[ B_1 \lor B_2 \lor \ldots \lor B_n \]
  \[ \text{at most one}(\{B_1, B_2, \ldots, B_n\}) \]

- **Order encoding**

  \[ B_i \leftrightarrow X \leq a_i \]

- **Log encoding (sign-and-magnitude)**

  \[ X.m = <B_{k-1} \ldots B_1 B_0> \]
  \[ X.s = 0 \text{ or } 1 \]

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

- 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.
• 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

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

- **Constraints**
  - **Domain**
    - $X ::$ Domain
    - $X$ notin Domain
  - **Arithmetic**
    - $(X \neq Y)$, $(X \neq! Y)$, $(X \neq> Y)$, $(X \neq\geq Y)$, ...
  - **Boolean**
    - $(X \neq/ Y)$, $(X \neq/ Y)$, $(X \neq<=> Y)$, $(X \neq=> Y)$, $(X \neq^ Y)$, $(#~ X)$
  - **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).
• 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
PicatSAT

• 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

High-level Constraints

• 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

Primitive Constraints

- Small Pseudo-Boolean (PB) constraint: $\Sigma_{i=1}^{n} (a_i \times B_i) \odot b$, $n \leq 10$
- Boolean cardinality constraint: $\Sigma_{i=1}^{n} B_i \odot b$ (b is 1 or 2)
- $X :: D$
- $X \odot Y$
- $X + Y = Z$
- $X \times Y = Z$
- $Y = -X$
- $\text{abs}(X) = Y$
- $X \text{ div } Y = Z$
- $X \text{ mod } Y = Z$
- Reified primitive constraint: $B \leftrightarrow C$
- Implicative primitive constraint: $B \Rightarrow C$

preprocessing
decomposition
• Espresso

\% X+Y = 2*U+V,
\% where [X,Y,U,V] :: 0..1, UV \neq 11

.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 -

\neg X \lor U \lor V
Y \lor \neg U
\neg X \lor \neg Y \lor V
\neg X \lor \neg Y \lor U
X \lor Y \lor \neg V

N.F. Zhou, PicatSAT
Encoding the at-most-one Constraint

• Chen’s two-product encoding

\[
\begin{align*}
\text{amo}([X_{11}, X_{12}, \ldots, X_{pq}]) \\
\text{amo}([U_1, U_2, \ldots, U_p]) \\
\text{amo}([V_1, V_2, \ldots, V_q]) \\
X_{ij} \rightarrow U_i \land V_j \quad \text{for } i \text{ in } 1..p \text{ and } j \text{ in } 1..q
\end{align*}
\]

• Each domain variable is encoded as a vector of Boolean variables
  – $X.m = <B_{n-1},...,B_1,B_0>$
  – $X.s$ is the sign bit

• No negative zero is allowed
  – $X.m = <0,...,0,0> \Rightarrow X.s = 0$
Sign-and-Magnitude Log Encoding (Example)

\[ X :: [-2,-1,1,2] \]
\[ X.m = <X_1,X_0> \]
\[ X.s = S \]

• Naïve Encoding

\[ \neg S \lor \neg X_1 \lor \neg X_0 \quad (X \neq -3) \]
\[ \neg S \lor X_1 \lor X_0 \quad (X \neq -0) \]
\[ S \lor X_1 \lor X_0 \quad (X \neq 0) \]
\[ S \lor \neg X_1 \lor \neg X_0 \quad (X \neq 3) \]

• Optimized Encoding (Using Espresso)

\[ X_0 \lor X_1 \]
\[ \neg X_0 \lor \neg X_1 \]
Breaking Arithmetic Constraints

• Combine power-of-2 terms

\[ 2^{k-1} \times X_{k-1} + \ldots + 2^0 \times X_0 \rightarrow X :: 0..2^k - 1 \]

• Move out non-linear terms

• Factor out terms with common coefficients

• Decompose the constraint (Huffman coding)

\[
\text{decompose}(a_1 \times X_1 + a_2 \times X_2 + \ldots + a_n \times X_n \odot b):
\]
\[
\text{add all the terms } a_i \times X_i \text{ into a priority queue } Q
\]
\[
\text{while the constraint is not primitive:}
\]
\[
\text{remove two terms } a_i \times X_i \text{ and } a_j \times X_j \text{ from } Q
\]
\[
\text{where } a_i = a_j, \text{ and}
\]
\[
X_i \text{ and } X_j \text{ have the smallest domains}
\]
\[
\text{post } T = X_i + X_j
\]
\[
\text{add the term } a_i \times T \text{ into } Q
\]
\[
\text{post the primitive constraint}
\]
The Comparison Constraint: \( X \geq Y \)

- **Signed comparison**

\[
X.s = 0 \land Y.s = 1 \lor \\
X.s = 1 \land Y.s = 1 \Rightarrow X.m \leq Y.m \lor \\
X.s = 0 \land Y.s = 0 \Rightarrow X.m \geq Y.m
\]

- **Unsigned comparison**

\[
X.m = \langle X_{n-1}X_{n-2} \ldots X_1X_0 \rangle, \ Y.m = \langle Y_{n-1}Y_{n-2} \ldots Y_1Y_0 \rangle
\]

\[
T_0 \iff (X_0 \geq Y_0) \\
T_1 \iff (X_1 > Y_1) \lor (X_1 = Y_1 \land T_0) \\
\vdots \\
T_{n-1} \iff (X_{n-1} > Y_{n-1}) \lor (X_{n-1} = Y_{n-1} \land T_{n-2})
\]
The Addition Constraint: $X+Y = Z$

- **Unsigned addition (ripple-carry adders)**

  \[
  \begin{array}{c}
  X_{n-1} \ldots X_1 X_0 \\
  + Y_{n-1} \ldots Y_1 Y_0 \\
  \hline
  Z_n Z_{n-1} \ldots Z_1 Z_0
  \end{array}
  \]

  Carriers are used

- **Signed addition**

  \[
  \begin{align*}
  X.s = 0 \land Y.s = 0 & \Rightarrow Z.s = 0 \land X.m+Y.m = Z.m \\
  X.s = 1 \land Y.s = 1 & \Rightarrow Z.s = 1 \land X.m+Y.m = Z.m \\
  X.s = 0 \land Y.s = 1 \land Z.s = 1 & \Rightarrow X.m+Z.m = Y.m \\
  X.s = 0 \land Y.s = 1 \land Z.s = 0 & \Rightarrow Y.m+Z.m = X.m \\
  X.s = 1 \land Y.s = 0 \land Z.s = 0 & \Rightarrow X.m+Z.m = Y.m \\
  X.s = 1 \land Y.s = 0 \land Z.s = 1 & \Rightarrow Y.m+Z.m = X.m \\
  \end{align*}
  \]
The Full Adder

\[ X_i + Y_i + C_{in} = C_{out} Z_i \]

\[
\begin{align*}
X_i \lor \neg Y_i \lor C_{in} \lor Z_i \\
X_i \lor Y_i \lor \neg C_{in} \lor Z_i \\
\neg X_i \lor \neg Y_i \lor C_{in} \lor \neg Z_i \\
\neg X_i \lor Y_i \lor \neg C_{in} \lor \neg Z_i \\
\neg X_i \lor C_{out} \lor Z_i \\
\neg X_i \lor \neg C_{out} \lor \neg Z_i \\
\neg Y_i \lor \neg C_{in} \lor C_{out} \\
Y_i \lor C_{in} \lor \neg C_{out} \\
\neg X_i \lor \neg Y_i \lor \neg C_{in} \lor Z_i \\
X_i \lor Y_i \lor C_{in} \lor \neg Z_i 
\end{align*}
\]
A Specialized Adder for $X+1 = Y$

\[
\begin{array}{cccccccc}
X_{i+4} & X_{i+3} & X_{i+2} & X_{i+1} & X_i \\
+ & & & & C_{in} \\
C_{out} & Y_{i+4} & Y_{i+3} & Y_{i+2} & Y_{i+1} & Y_i
\end{array}
\]

• 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: $X \cdot Y = Z$

- **The Shift-and-Add Algorithm**

  \[ X.m \cdot Y.m = Z.m \quad X.m = <X_{n-1},...,X_1,X_0> \]

  \[ X_0 = 0 \Rightarrow S_0 = 0 \]
  \[ X_0 = 1 \Rightarrow S_0 = Y \]
  \[ X_1 = 0 \Rightarrow S_1 = S_0 \]
  \[ X_1 = 1 \Rightarrow S_1 = (Y \ll 1) + S_0 \]
  \[ \vdots \]
  \[ X_i = 0 \Rightarrow S_i = S_{i-1} \]
  \[ X_i = 1 \Rightarrow S_i = (Y \ll i) + S_{i-1} \]
  \[ \vdots \]
  \[ X_{n-1} = 0 \Rightarrow S_{n-1} = S_{n-2} \]
  \[ X_{n-1} = 1 \Rightarrow S_{n-1} = (Y \ll (n-1)) + S_{n-2} \]
  \[ Z = S_{n-1} \]
• Equivalence reasoning is an optimization that reasons about a possible value for a Boolean variable or the relationship between two Boolean variables.

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

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

\[
X = Y \mod 2^K \quad \Rightarrow X_0 = Y_0, \ X_1 = Y_1, \ldots, \ X_{k-1} = Y_{k-1}
\]

\[
X = Y \div 2^K \quad \Rightarrow X_0 = Y_K, \ X_1 = Y_{K+1}, \ldots
\]

• No clauses are needed to encode \( X = \text{abs}(Y) \).
Constant Propagation on \( X + Y = Z \)

\[
X_i + Y_i = C_{out}Z_i
\]

**Rule-1** : \( X_i = 0 \Rightarrow C_{out} = 0 \land Z_i = Y_i \).

**Rule-2** : \( X_i = 1 \Rightarrow C_{out} = Y_i \land Z_i = \neg Y_i \).

**Rule-3** : \( Z_i = 0 \Rightarrow C_{out} = X_i \land X_i = Y_i \).

**Rule-4** : \( Z_i = 1 \Rightarrow C_{out} = 0 \land X_i = \neg Y_i \).

- **Example-1**

\[
\begin{array}{cccc}
X_2 & X_1 & X_0 \\
+ & 1 & 0 & 0 \\
\hline
Z_3 & Z_2 & Z_1 & Z_0
\end{array}
\]

\[
X_0 = Z_0 \\
X_1 = Z_1 \\
\neg X_2 = Z_2 \\
X_2 = Z_3
\]

- **Example-2**

\[
\begin{array}{cccc}
X_2 & X_1 & X_0 \\
+ & Y_2 & Y_1 & Y_0 \\
\hline
1 & 0 & 1 & 1
\end{array}
\]

\[
\neg X_0 = Y_0 \\
\neg X_1 = Y_1 \\
X_2 = Y_2 \\
X_2 = 1 \\
Y_2 = 1
\]
Constant Propagation on $X \times Y = Z$

\[X_0 = 0 \Rightarrow S_0 = 0\]
\[X_0 = 1 \Rightarrow S_0 = Y\]
\[X_1 = 0 \Rightarrow S_1 = S_0\]
\[X_1 = 1 \Rightarrow S_1 = (Y << 1) + S_0\]
\[\vdots\]
\[X_i = 0 \Rightarrow S_i = S_{i-1}\]
\[X_i = 1 \Rightarrow S_i = (Y << i) + S_{i-1}\]
\[\vdots\]
\[X_{n-1} = 0 \Rightarrow S_{n-1} = S_{n-2}\]
\[X_{n-1} = 1 \Rightarrow S_{n-1} = (Y << (n-1)) + S_{n-2}\]
\[Z = S_{n-1}\]

**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** : $X.m = <X_{n-1} \ldots X_i 0 \ldots 0>$ $\land$ $X_i = 1$ \Rightarrow $Z_i = Y_0 \land Z_k = 0$ for $k \in 0..(i - 1)$.
• 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([V_1, V_2, \ldots, V_n])

- **Standard**
  \[ V_i \neq V_j \text{ for } i, j = 1, \ldots, n, i < j. \]

- **Use \texttt{at\_most\_one}**
  - Let \( D = D_1 \cup D_2 \cup \ldots \cup D_n \)
  - If \(|D| > n\): \( \forall a \in D: \texttt{at\_most\_one}([V_1 = a, V_2 = a, \ldots, V_n = a]) \)
  - If \(|D| = n\): \( \forall a \in D: \texttt{exactly\_one}([V_1 = a, V_2 = a, \ldots, V_n = a]) \)

- **A hybrid of log and direct encodings**
Table Constraints

- Truth table encoding

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X₀</th>
<th>X₁</th>
<th>Y₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \neg X₁ \lor \neg X₀ \lor Y₀ \]
\[ X₁ \lor X₀ \]
\[ Y₁ \lor Y₀ \]
\[ \neg Y₁ \lor \neg Y₀ \]

- BDD and truth table

- Trie encoding
The element and regular Constraints

- element\((I, [E_1, E_2, \ldots, E_n], V)\)
  - If \(E_i\)'s are ground
    \((I, V) \in [(1, E_1), (2, E_2), \ldots, (n, E_n)]\)
  - If at least one \(E_i\) is variable
    for each \(i \in 1..n: I = i \Rightarrow V = E_i\)

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

The Global Constraint circuit($G$)

$$\text{circuit}(G)$$

$$G = [V_1, V_2, \ldots, V_n]$$

- 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.
Distance Encoding for circuit

\[ \text{circuit}([V_1, V_2, \ldots, V_n]) \]

- **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
    \[
    \text{For each } i \in 1..n, \ j \in 1..n: \ H_{ij} \Leftrightarrow V_i = j
    \]
    \[ (1) \]
  - Degree constraints
    \[
    \text{For each } j \in 1..n: \sum_{i=1}^{n} H_{ij} = 1
    \]
    \[ (2) \]
  - Visit vertex 1 first: \( P_1 = 1 \)
    \[
    \text{For each } i \in 2..n: \]
    \[
    H_{1i} \Rightarrow P_i = 2 \\
    H_{i1} \Rightarrow P_i = n
    \]
    \[ (3") \]
  - Transition constraints
    \[
    \text{For each } i \in 2..n, \ j \in 2..n, \ i \neq j:\]
    \[
    H_{ij} \Rightarrow P_j = P_i + 1
    \]
    \[ (5") \]
• 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 he 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

cumulative([S_1,S_2,...,S_n],[D_1,D_2,...,D_n],[R_1,R_2,...,R_n],Limit)

- **Occupation constraints**
  
  for each time \( t_i \) and each task \( j \):
  \[
  O_{ij} \iff S_j \leq t_i < S_j + D_j
  \]

- **Resource constraints**
  
  for each time \( t_i \):
  \[
  \sum_{j=1}^{n} O_{ij} * R_j \leq Limit
  \]

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

• 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)

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

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

N.F. Zhou, PicatSAT
## Experimental Results (XCSP Competition, CSP)

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

N.F. Zhou, PicatSAT
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
• **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**