# The Picat-SAT Compiler

**Neng-Fa Zhou**

(CUNY Brooklyn College & GC)

Joint work with ***Håkan*** **Kjellerstrand**

# Outline

- Why Picat-SAT?
- Picat's constraint modules
- Booleanization of constraints
  - Sign-and-magnitude log encoding of domain variables
  - Primitive, reified, and implicative constraints
  - The addition constraint X+Y = Z
  - The multiplication constraint X*Y = Z
  - Table constraints
  - Global constraints
- Experimental results
- Conclusion

# Why Picat-SAT?

- SAT solvers' performance has drastically improved during the past 20 years
- SAT has become the backbone of many software systems
- It is viable to build SAT-based CSP solvers
- There are so many papers but so few compilers
  - Sugar, FznTini, BEE, and meSAT
- Build a SAT compiler that is solid, efficient, and practically usable

# Picat's Constraint Modules

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

- Constraints
  - Domain
    - X `::` Domain, X notin Domain
  - Arithmetic
    - (X #= Y), (X #!= Y), (X #> Y), (X #>= Y), …
  - Boolean
    - (X #/\ Y), (X #\/ Y), (X #<=> Y), (X #=> Y), (X #^ 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)

# Picat's Constraint Modules

## SEND + MORE = MONEY

```
import sat.

go =>
      Vars=[S,E,N,D,M,O,R,Y], % generate variables
      Vars :: 0..9,           % define the domains
      all_different(Vars),    % generate constraints
      S #!= 0,
      M #!= 0,
      1000*S+100*E+10*N+D+1000*M+100*O+10*R+E
              #= 10000*M+1000*O+100*N+10*E+Y,
      solve(Vars),            % search
      writeln(Vars).
```

# Picat's Constraint Modules

## Sudoku

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

```
import sat.

sudoku(Board) =>
   N = Board.len,
   Vars = Board.vars(),
   Vars :: 1..N,
   foreach (Row in Board) all_different(Row) end,
   foreach (J in 1..N)
      all_different([Board[I,J] : I in 1..N])
   end,
   M = round(sqrt(N)),
   foreach (I in 1..M..N-M, J in 1..M..N-M)
      all_different([Board[I+K,J+L] : K in 0..M-1, L in 0..M-1])
   end,
   solve(Vars).
```

# Booleanization of Constraints

- Domain variables
- Primitive, reified, and implicative constraints
- The addition constraint X+Y = Z
- The multiplication constraint X*Y = Z
- Table constraints
- Global constraints

# Encoding Domain Variables

- Sparse encoding
  - B ⇔ X = a
    One Boolean variable is used for each value
- Order encoding [Sugar, BEE, meSAT]
  - B ⇔ X ≤ a
- Log encoding [FznTini]
  - V: $[a_1,\ldots,a_n] \Leftrightarrow \langle B_m,\ldots,B_1,B_0\rangle$
    Each domain variable is encoded as a binary string
    $m = \log_2(\max(|a_1|,|a_n|))$

# Log-encoding

- Pros and Cons
  - Pros: compact, and good for compiling arithmetic constraints
  - Cons: less propagation power
- Encoding domains that have negative values
  - 2's Complement Representation [FznTini]
    - Only one representation for 0
  - Sign-and-Magnitude Representation [PicatSAT]
    - Efficient codes for Y = abs(X) and Y = -X
    - One extra clause for banning negative zero

# Encoding Domain Variables

$X :: [-2,-1,1,2] \Leftrightarrow \langle S, X_1, X_0 \rangle$

- A Naïve Code

$$S \vee X_1 \vee X_0$$
$$\neg S \vee X_1 \vee X_0$$
$$S \vee \neg X_1 \vee \neg X_0$$
$$\neg S \vee \neg X_1 \vee \neg X_0$$

- An Optimized Code (Using Espresso)

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

# Primitive, Reified, and Implicative Constraints

- Primitive constraints
  - $\sum B_i \geq c$ (at-least), $\sum B_i \leq c$ (at-most)
  - $X = Y$, $X \geq Y$, $X > Y$, $X \neq Y$
  - $-X = Y$, $abs(X) = Y$
  - $X+Y = Z$, $X*Y = Z$
- Reified and implicative constraints
  - $B \Leftrightarrow C$ ($B \Rightarrow C$ and $not(C) \Rightarrow not(B)$)
  - $B \Rightarrow C$

# The Addition Constraint X+Y = Z

- Unsigned addition

$$\begin{array}{rc} & X_{n-1} \ldots X_1 \; X_0 \\ + & Y_{n-1} \ldots Y_1 \; Y_0 \\ \hline Z_n & Z_{n-1} \ldots Z_1 \; Z_0 \end{array}$$

Boolean variables are used for carries

- Signed addition

$$\begin{array}{l} X.\mathtt{s} = 0 \wedge Y.\mathtt{s} = 0 \Rightarrow Z.\mathtt{s} = 0 \wedge X.\mathtt{m}{+}Y.\mathtt{m} = Z.\mathtt{m} \\ X.\mathtt{s} = 1 \wedge Y.\mathtt{s} = 1 \Rightarrow Z.\mathtt{s} = 1 \wedge X.\mathtt{m}{+}Y.\mathtt{m} = Z.\mathtt{m} \\ X.\mathtt{s} = 0 \wedge Y.\mathtt{s} = 1 \wedge Z.\mathtt{s} = 1 \Rightarrow X.\mathtt{m}{+}Z.\mathtt{m} = Y.\mathtt{m} \\ X.\mathtt{s} = 0 \wedge Y.\mathtt{s} = 1 \wedge Z.\mathtt{s} = 0 \Rightarrow Y.\mathtt{m}{+}Z.\mathtt{m} = X.\mathtt{m} \\ X.\mathtt{s} = 1 \wedge Y.\mathtt{s} = 0 \wedge Z.\mathtt{s} = 0 \Rightarrow X.\mathtt{m}{+}Z.\mathtt{m} = Y.\mathtt{m} \\ X.\mathtt{s} = 1 \wedge Y.\mathtt{s} = 0 \wedge Z.\mathtt{s} = 1 \Rightarrow Y.\mathtt{m}{+}Z.\mathtt{m} = X.\mathtt{m} \end{array}$$

# The Multiplication Constraint X*Y = Z

- Shift-and-Add Algorithm (Pen and Pencil)

$Y \Leftrightarrow \langle Y_{n-1}, \ldots, Y_1, Y_0 \rangle$

$$
\begin{aligned}
&Y_0 = 0 \Rightarrow S_0 = 0 \\
&Y_0 = 1 \Rightarrow S_0 = X \\
&Y_1 = 0 \Rightarrow S_1 = S_0 \\
&Y_1 = 1 \Rightarrow S_1 = (X << 1) + S_0 \\
&\vdots \\
&Y_{n-1} = 0 \Rightarrow S_{n-1} = S_{n-2} \\
&Y_{n-1} = 1 \Rightarrow S_{n-1} = (X << (n-1)) + S_{n-2} \\
&Z = S_{n-1}
\end{aligned}
$$

# Table Constraints

```
table_in({X,Y},[{1,2},{1,3},{2,1},{2,2},{3,1}])
```

$X \Leftrightarrow \langle X_1, X_0 \rangle \qquad Y \Leftrightarrow \langle Y_1, Y_0 \rangle$

| $X_1$ | $X_0$ | $Y_1$ | $Y_0$ |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Espresso →

$\neg X_0 \wedge Y_1 \wedge Y_0$
$X_1 \wedge X_0 \wedge Y_1$
$\neg X_1 \wedge \neg X_0$
$\neg X_1 \wedge \neg Y_1$
$\neg Y_1 \wedge \neg Y_0$

DNF →

$X_0 \vee \neg Y_1 \vee \neg Y_0$
$\neg X_1 \vee \neg X_0 \vee \neg Y_1$
$X_1 \vee X_0$
$X_1 \vee Y_1$
$Y_1 \vee Y_0$

CNF

# Global Constraints

- `all_different([V1,V2,…,Vn])`

  `Vi ≠ Vj for i,j = 1,…,n, i < j`

  If it is an assignment-type constraint, then
  Let `a1,a2,…,an` be the values.
  $\sum$ `(Vi = aj) = 1` for each `aj`.

- `circuit([V1,V2,…,Vn])`

  `all_different([V1,V2,…,Vn])`

  and constraints for preventing sub-cycles

# Global Constraints

- `cumulative(Ss,Ds,Rs,Limit)`
  - Time decomposition
  - Task decomposition
- `element(I,L,V)`
  - Decomposed to a table constraint if L is ground
  - Decomposed to Boolean constraints, otherwise

# Implementation

- Specialization techniques
  - X+c = Y, a*X = Y,…
- Common sub-expression elimination
  - X+Y+Z1 < b1, X+Y-Z2 > b2
- The default SAT solver: lingeling 587f
- 5,000 lines of codes in Picat

# Experimental Results (Picat-SAT vs. Sugar)

| Benchmark | Picat | | | Sugar | | | Azucar | | |
|---|---|---|---|---|---|---|---|---|---|
| | #vars | #cls | time | #vars | #cls | time | #vars | #cls | time |
| golombRuler-10-55 | 6438 | 27194 | **36.50** | 4584 | 190594 | 92.53 | 5635 | 44937 | 60.32 |
| golombRuler-8 | 3024 | 12698 | **0.22** | 5111 | 422046 | 3.04 | 2988 | 35847 | 0.44 |
| jss-ft10 | 13613 | 68880 | 5.90 | 76617 | 703313 | 6.24 | 8227 | 119126 | **1.50** |
| knightTour-5 | 3051 | 21401 | 8.19 | 1296 | 20510 | **0.83** | 1667 | 9123 | 2.36 |
| knightTour-7 | 11970 | 114818 | 374.36 | 4944 | 141818 | 99.68 | 5919 | 44059 | **85.53** |
| magicSquare-5x5 | 2742 | 20302 | 1.68 | 3504 | 73717 | 0.72 | 2052 | 16239 | **0.29** |
| magicSquare-9x9 | 33606 | 178328 | **48.58** | 60960 | 5463881 | 167.24 | 19405 | 317128 | 57.36 |
| nqueens-200 | 1003112 | 3997099 | 1925.52 | 159196 | 18646098 | **131.96** | 284400 | 4004662 | 397.82 |
| oss-gp03-01 | 785 | 3535 | **0.19** | 11126 | 46075 | 0.69 | 786 | 6231 | 2.13 |
| oss-gp10-01 | 25540 | 139168 | **2.90** | 343189 | 6514888 | 40.40 | 17216 | 544880 | 5.79 |
| socialGolfer-3-2-5 | 315 | 3360 | 0.12 | 810 | 2355 | 0.31 | 1680 | 5250 | **0.10** |
| socialGolfer-6-3-7 | 9702 | 163863 | **13.19** | 22545 | 79236 | 13.78 | 46248 | 184980 | 14.04 |
| tdsp-C1-1 | 6361 | 25187 | 0.67 | 1226 | 17631 | **0.26** | 4586 | 36997 | 0.64 |
| tdsp-C2-1 | 16079 | 67032 | 5.09 | 2720 | 61572 | **0.72** | 11746 | 112436 | 7.12 |

Picat-SAT is competitive with Sugar on the Sugar benchmark suite overall, but is slower than Sugar on knightTour and nqueens because of the poor encoding of all_different .

# Experimental Results (Picat-SAT vs. Sugar)

| Benchmark | Picat | | | Sugar | | | Azucar | | |
|---|---|---|---|---|---|---|---|---|---|
| | #vars | #cls | time | #vars | #cls | time | #vars | #cls | time |
| gcj_small | 215 | 4746 | 0.275 | n/a | n/a | n/a | n/a | n/a | n/a |
| gcj_large | 980 | 6721 | 1.985 | n/a | n/a | n/a | n/a | n/a | n/a |
| sendmory | 370 | 2095 | 0.089 | 3775 | 88809 | 0.590 | 2501 | 85902 | 0.661 |
| sendmory_neg | 403 | 4082 | 0.144 | 3775 | 88809 | 0.608 | 4294 | 209632 | 1.730 |
| crypta | 3575 | 24452 | 1.353 | n/a | n/a | n/a | 443530 | 721220562 | n/a |
| crypta_neg | 3793 | 34676 | 3.054 | n/a | n/a | n/a | 430578 | 667692122 | n/a |

Picat-SAT significantly outperforms Sugar on arithmetic-intensive benchmarks.

# Experimental Results (Picat-SAT on Minizinc'15)

without global constraints

| Benchmark | OR-Tools | Opturion CPX | iZplus | Picat-SAT |
|---|---|---|---|---|
| freepizza(MIN) | 4.03 | 10.96 | **12.00** | 3.01 |
| grid-colouring(MIN) | 1.91 | 11.27 | 5.28 | **11.54** |
| nmseq(SAT) | **13.46** | 10.75 | 3.19 | 0.60 |
| project-planning(MIN) | 5.00 | **14.67** | 10.33 | 0.00 |
| radiation(MIN) | 0.00 | **11.04** | 8.56 | 10.40 |
| triangular(MAX) | 7.75 | 1.00 | 9.00 | **12.25** |
| zephyrus(MIN) | **9.00** | 7.50 | 7.50 | 6.00 |
| (TOTAL) | 41.16 | 67.19 | 55.86 | 43.79 |

with global constraints

| Benchmark | OR-Tools | Opturion CPX | iZplus | Picat-SAT |
|---|---|---|---|---|
| costas-array(SAT) | **7.35** | 4.86 | 5.06 | 0.73 |
| cvrp(MIN) | 7.07 | 8.59 | **12.30** | 2.04 |
| gfd-schedule(MIN) | 2.48 | 10.53 | **11.99** | 5.00 |
| is(MIN) TOTAL | 7.56 | 7.88 | **12.56** | 0.00 |
| largescheduling(MIN) | **13.00** | 10.00 | 0.00 | 0.00 |
| mapping(MIN) | 2.00 | **14.91** | 7.00 | 5.09 |
| multi-knapsack(MAX) | 4.18 | **10.75** | 4.26 | 9.80 |
| opd(MIN) | 8.07 | 2.50 | 7.00 | **12.43** |
| open_stacks(MIN) | 10.10 | 5.19 | 2.00 | **12.71** |
| p1f(MIN) | **14.53** | 3.95 | 2.01 | 3.51 |
| roster(MIN) | **11.72** | 4.44 | 11.40 | 2.44 |
| spot5(MAX) | 8.00 | 6.68 | 2.34 | **12.98** |
| tdtsp(MIN) | 8.24 | **11.26** | 10.50 | 0.00 |
| (TOTAL) | 104.31 | 101.54 | 88.43 | 66.72 |

# Picat-CP vs. Picat-SAT on Minizinc'13

| Benchmark | CP | SAT |
|---|---|---|
| black-hole | 4 | 5 |
| cargo | 0 | 2 |
| celar | 5 | 4 |
| filters | 3 | 5 |
| fjsp | 3 | 5 |
| ghoulomb | 1 | 2 |
| javarouting | 3 | 5 |
| l2p | 1 | 5 |
| league | 2 | 5 |
| mario | 4 | 5 |
| nmseq | 0 | 2 |
| nonogram | 0 | 5 |
| on-call-rostering | 5 | 5 |
| pattern-set-mining | 5 | 1 |
| pentominoes-int | 4 | 0 |
| proteindesign12 | 0 | 0 |
| radiation | 5 | 5 |
| rcpsp | 5 | 5 |
| rubik | 2 | 5 |
| vrp | 5 | 5 |
| *Total* | 62 | 76 |

- The settings and the result
  - An optimization instance is considered *solved* if at least one solution was found.
  - For CP, the labeling option *ffc* was used.
  - Total number instances: 100.
  - Time limit:  900s per instance.
  - SAT outperforms CP.

# Summary and Future Work

- Picat-SAT
  - Sign-and-magnitude log encoding
  - Uses some of the encoding algorithms and tools from hardware design (addition and multiplication, Espresso)
  - Solid, efficient, and practically usable
- Future Work
  - More specialization and optimization techniques
  - New encoding algorithms for PB and table constraints
  - Efficient decomposition algorithms for global constraints