

REAL WORLD MODEL CHECKING USING SAT SOLVERS

Adrien Bustany

WHAT IS MODEL CHECKING ANYWAY?

*Get an exhaustive proof of a given set of
properties on a model*

YEAH, LIKE WHAT?

YEAH, LIKE WHAT?

INTERLOCKING SYSTEMS

Check that the signals around a junction on a train track
keep it safe

YEAH, LIKE WHAT?

ELECTRONIC CIRCUITS

Check that a circuit behaves according to its specification

YEAH, LIKE WHAT?

ABSENCE OF UNDEFINED BEHAVIOUR IN CODE

Integer overflow, null pointer dereference...

YEAH, LIKE WHAT?

AUTOMATIC TEST CASE GENERATION

Find the right set of inputs to trigger a given situation

YEAH, LIKE WHAT?

PROBLEM OPTIMIZATION

To some extent, iteratively proving that a better solution
exists

HOW DO YOU DESCRIBE A MODEL?

No standard format, usually using a declarative language that declares inputs, outputs and the logical relations between them

HOW DO YOU CHECK A MODEL?

Many approaches:

- BDD (binary decision diagrams)
- Abstract interpretation
- SMT
- SAT

Explore the set of all possible states to prove that no properties are violated.

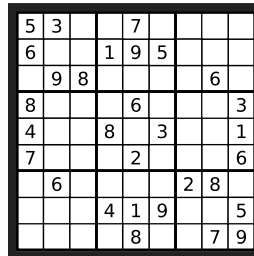
HOW DO YOU CHECK A MODEL?

In other words:

1. Be smart (teach "semantics" to the prover): saves on computation, but increases complexity and is potentially less generic
2. Be brutal: explore all states ← What we do 😊

LET'S MODEL A SUDOKU

Rules of Sudoku:



A 9x9 Sudoku grid with some numbers filled in. The grid is divided into 3x3 groups. The numbers are as follows:

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	

1. A grid is 9x9 cells, 3x3 groups
2. All cells must be filled with a value
3. The value of each cell is in $[1, 9]$
4. A value can appear at most once in a row, column and group

LET'S MODEL A SUDOKU

A bit more formal

```
int [1,9] grid [9,9];

∀ value ∈ [1, 9] (
  ∀ line ∈ [0, 8] (
    ∃ column ∈ [0, 8] (grid[line, column] = value)
  ) ∧
  ∀ column ∈ [0, 8] (
    ∃ line ∈ [0, 8] (grid[line, column] = value)
  ) ∧
  ∀ gline ∈ [0, 2], gcolumn ∈ [0, 2] (
    ∃ line ∈ [0, 2], column ∈ [0, 2] (
      grid[gline * 3 + line, gcolumn * 3 + column] = value
    )
  )
)
```

FORMAL PROBLEM SOLVING USING SAT

So we've formalized our problem using simple propositional logic... How do we solve it?

Let's *lower* it to simple forms!

EXPANSION RULES

The expansion rules describe the process of transforming a set of statements in the "high level" logic to a lower level one.

EXPANSION RULES

COMPOSITES DATA STRUCTURES

```
int [1,9] array [3];
```

gets expanded to

```
int [1,9] array_0;  
int [1,9] array_1;  
int [1,9] array_2;
```

EXPANSION RULES

QUANTIFIERS

$\forall \text{ value } \in [1, 3] P(x)$

gets expanded to

$P(1) \wedge P(2) \wedge P(3)$

and

$\exists \text{ value } \in [1, 3] P(x)$

gets expanded to

$P(1) \vee P(2) \vee P(3)$

EXPANSION RULES

INTEGERS

```
int [0,8] value;
```

gets expanded to

```
bool value_0; // bit 0  
bool value_1; // bit 1  
bool value_2; // bit 2  
bool value_3; // bit 3
```

...and integer operations to bitwise operations

EXPANSION RULES

Rinse and repeat until you reach a form that your SAT solver understands!

NOT EVERYTHING IN LIFE IS COMBINATIONAL

A Sudoku is fun, but it's pretty static.

What if we have inputs changing over time?

NOT EVERYTHING IN LIFE IS COMBINATIONAL

Let's add one additional dimension: time

```
// x is a stream of value over time
// one could see it as x0, x1, ..., xN
int [0, 5] x;

// y has value FALSE at t = 0, and then flips at each cycle
y := FALSE, ~y;

// z contains the value of y at the previous cycle
z := pre(y);
```

Welcome to the world of sequential logic!

Subject for a future talk?