Constraint Satisfaction for First-Order Logic

William McCune

Computer Science Department
University of New Mexico

http://www.cs.unm.edu/~mccune/

Constraint Satisfaction

A Constraint Satisfaction Problem (CSP):

- A set of variables: $X_1, X_2, \cdots, X_n$.
- A corresponding set of domains: $D_1, D_2, \cdots, D_n$.
  - Each domain is a set of possible values for that variable.
- A set of constraints: $C_1, C_2, \cdots, C_m$.
  - Each constraint refers to a subset of the variables and specifies the acceptable combinations of values for those variables.

Solution: assignment of a value to each variable so that all constraints are satisfied.

We'll be talking about *finite-domain* constraint satisfaction.

First-Order Language (with Equality)

- Well-formed terms
  - variables: $x, y, z, u, \cdots$
  - function symbols, including constants
- Well-formed formulas
  - well-formed terms
  - predicate (relation) symbols, including equality
  - connectives: $\&, |, \neg, \rightarrow, \leftrightarrow, \forall, \exists$.

What makes the language only *first-order*? The inability use predicates or functions as variables. For example, an induction axiom:

$$\forall P(P(0) \;\&\; \forall x(P(x) \rightarrow P(x+1)) \rightarrow \forall n P(n))$$

Interpretation of a First-Order Language

- One domain D (may be infinite).
- Each constant in the language is assigned a member of D.
- Each function symbol is assigned a function from $D \times \cdots \times D$ to D.
- Each predicate (relation) symbol is assigned a relation on $D \times \cdots \times D$.
- Logic connectives behave as expected.
- Quantifiers range over the domain.

Given a *closed* (no free variables) formula F and an interpretation I,

$$evaluate(F, I) \in \{True, False\}.$$

A *model* of a formula is an interpretation in which the formula is $True$.

Finite First-Order Satisfiability as a Constraint-Satisfaction Problem

Consider a first-order theory of sets involving

predicates: $\in, \subseteq$; functions: $\cup, \sim$; constants: $\emptyset, U$;

Let the theory be specified by the following formulas:

$$
\begin{aligned}
&x \in U \\
&x \notin \emptyset \\
&x \subseteq y \leftrightarrow \forall z(z \in x \rightarrow z \in y) \\
&x \in y \cup z \leftrightarrow x \in y \ \mid \ x \in z \\
&x \in \sim y \leftrightarrow x \in U \ \ \& \ \ x \notin y
\end{aligned}
$$

[*some other formulas involving these functions and predicates*]

Say we want to find a finite model of the theory.

Finite First-Order Satisfiability as a CSP (cont'd)

Relations, domain is {True, False}.

$\in$:

	0	1	2	3
0				
1				
2				
3				

$\subseteq$:

	0	1	2	3
0				
1				
2				
3				

Functions, domain is {0,1,2,3}.

$\cup$:

	0	1	2	3
0				
1				
2				
3				

$\sim$:

0	1	2	3

$\emptyset$: $\square$ $\quad U$: $\square$

Each cell is a CSP variable;

Corresponding domains are either {True,False} or {0,1,2,3};

Constraints are the formulas on the preceding page.

Finite First-Order Satisfiability as a CSP (cont'd)

Another example: find a non-commutative group of order 6.

Constraints:

$$E * x = x. \quad x * E = x.$$
$$x' * x = E. \quad x * x' = E.$$
$$(x * y) * z = x * (y * z).$$
$$A * B \neq B * A.$$

Domains: $\{$0,1,2,3,4,5$\}$.

Variables: $*$:

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

$'$:

0	1	2	3	4	5

E: $\boxed{}$ A: $\boxed{}$ B: $\boxed{}$

Comparison and Terminology

Constraint Satisfaction	*First-Order Satisfiability*
Variables	Cells in interpretation
Domains (arbitrary, multiple)	$\{0, 1, \cdots, n-1\}$ and $\{True, False\}$
Constraints (various languages)	First-order logic formulas
(arithmetic built in)	(arithmetic added on)
Solution	Model of formulas

Mace4: A Program for Finite First-Order Satisfiability

- "Models And CounterExamples"
- It takes a set of first-order formulas
  - and looks for finite models;
  - if the input is a conjecture, the models are counterexamples;
  - it iterates through domain sizes;
  - the search is complete (not local search)
- Developed independently from finite-domain constraint-satisfaction systems, but it has much in common with them.
  - backtracking search
  - methods for selecting cells (variables) to instantiate
  - constraint propagation

Mace4 Example: Non-Commutative Group

Input:

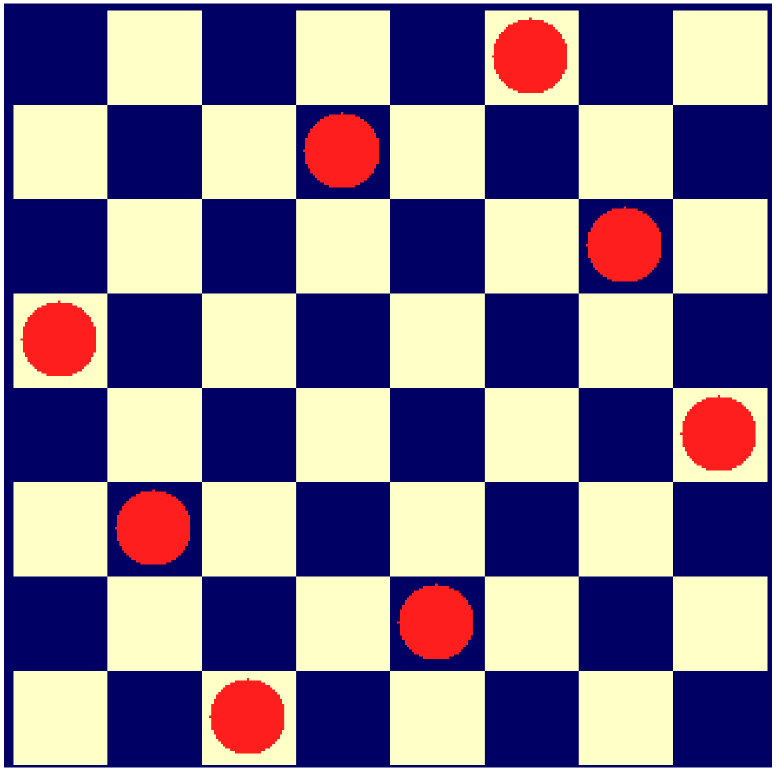
```
formulas(assumptions).
  E * x = x.    x * E = x.
  x' * x = E.  x * x' = E.
  (x * y) * z = x * (y * z).
end_of_list.

formulas(goals).
  x * y = y * x.
end_of_list.
```

Output contains:

```
interpretation( 6, [number=1, seconds=0], [
        function(E, [ 0 ]),
        function(c1, [ 1 ]),
        function(c2, [ 2 ]),
        function('(_), [ 0, 1, 2, 4, 3, 5 ]),
        function(*(_,_), [
                           0, 1, 2, 3, 4, 5,
                           1, 0, 3, 2, 5, 4,
                           2, 4, 0, 5, 1, 3,
                           3, 5, 1, 4, 0, 2,
                           4, 2, 5, 0, 3, 1,
                           5, 3, 4, 1, 2, 0 ])]).
```

The n-Queens Puzzle

- Place n queens on an $n \times n$ chessboard so that none threatens any other.
- A solution for $n = 8$.



The n-Queens Puzzle, Standard CSP Representation

- Variables: $X_1, X_2, \cdots, X_n$.
- All Domains are: $\{0, 1, \cdots, n-1\}$.
  $X_i = m$ means there is a queen at row i, column m.
- Constraints:

  $i \neq j \Rightarrow X_i \neq X_j$ (no 2 queens in the same column).

  $i \neq j \Rightarrow X_i - X_j \neq i - j$ (no 2 queens in the same $\backslash$ diagonal).

  $i \neq j \Rightarrow X_i - X_j \neq j - i$ (no 2 queens in the same $/$ diagonal).

  (The constraint that 2 queens cannot be in the same row is always satisfied in this representation.)

  Note that we're using arithmetic.

Arithmetic for Mace4

- Arithmetic is not part of first-order logic.
- Mace4 works on *finite* structures.
- Solution: for domain size n, use ring of integers mod n, and order the domain in the natural way: $0 < 1 < \cdots < n - 1$.
- Use the Mace4 commands

  ```
  set(integer_ring).
  set(order_domain).
  ```

  Then we can use the operations $+$, $-$ (unary), $--$ (binary), $*$ and the relations $<$, $\leq$.
- Keep in mind that we are using *modular* arithmetic.

The n-Queens Puzzle, in Mace4, version 1

```
set(integer_ring).  % <+,-,*> is ring of integers mod n (-- is binary minus)
set(order_domain).  % < and <= order the domain

formulas(assumptions).

% n-Queens Puzzle
%
% In this representation, Q(i)=n means that Row i Column n has a queen.
% The constraint that no queens can be in the same row is always satisfied
% in this representation, because Q is a function; that is,
% Q(x) != Q(z) -> x != z  is always satisfied.

x != z -> Q(x) != Q(z).   % No 2 queens in the same column.

% We have to be careful that diagonals do not wrap around, because
% modular arithmetic wraps around.  Thus, the < conditions.

x < z & Q(x) < Q(z) -> z -- x != Q(z) -- Q(x).  % No 2 queens in \ diagonal.

x < z & Q(z) < Q(x) -> z -- x != Q(x) -- Q(z).  % No 2 queens in / diagonal.

end_of_list.
```

The n-Queens Puzzle, in Mace4, version 1 Solutions

```
function(Q(_), [0,4,7,5,2,6,1,3] )
function(Q(_), [0,5,7,2,6,3,1,4] )
function(Q(_), [0,6,3,5,7,1,4,2] )
function(Q(_), [0,6,4,7,1,3,5,2] )
function(Q(_), [1,3,5,7,2,0,6,4] )
function(Q(_), [1,4,6,0,2,7,5,3] )
function(Q(_), [1,4,6,3,0,7,5,2] )
function(Q(_), [1,5,0,6,3,7,2,4] )
function(Q(_), [1,5,7,2,0,3,6,4] )
function(Q(_), [1,6,2,5,7,4,0,3] )
function(Q(_), [1,6,4,7,0,3,5,2] )
function(Q(_), [1,7,5,0,2,4,6,3] )
function(Q(_), [2,0,6,4,7,1,3,5] )
function(Q(_), [2,4,1,7,0,6,3,5] )
function(Q(_), [2,4,1,7,5,3,6,0] )
function(Q(_), [2,4,6,0,3,1,7,5] )
function(Q(_), [2,4,7,3,0,6,1,5] )
function(Q(_), [2,5,1,4,7,0,6,3] )
function(Q(_), [2,5,1,6,0,3,7,4] )
function(Q(_), [2,5,1,6,4,0,7,3] )
function(Q(_), [2,5,3,0,7,4,6,1] )
function(Q(_), [2,5,3,1,7,4,6,0] )
...
```

(92 solutions found).

The n-Queens Puzzle, in Mace4, version 2

```
set(integer_ring).  % <+,-,*> is ring of integers mod n (-- is binary minus)
set(order_domain).  % relations < and <= order the domain

formulas(assumptions).

% Relation Q(x,y) means there is a queen at row x, column y.

all x exists y Q(x,y).           % Each row has at *least* one queen.

Q(x,y1) & Q(x,y2) -> y1 = y2.    % Each row has at most one queen.

Q(x1,y) & Q(x2,y) -> x1 = x2.    % Each column has at most one queen.

% Since we're using mod arithmetic, we have to be careful that
% diagonals don't wrap around.  Thus the <= conditions.

Q(x1,y1) & Q(x2,y2) & (x1 <= x2 & y1 <= y2 & x2 -- x1 = y2 -- y1) ->
           x1 = x2 & y1 = y2.     % Each \ diagonal has at most one queen.

Q(x1,y1) & Q(x2,y2) & (x2 <= x1 & y1 <= y2 & x1 -- x2 = y2 -- y1) ->
           x1 = x2 & y1 = y2.     % Each / diagonal has at most one queen.

end_of_list.
```

The n-Queens Puzzle, in Mace4, version 2 Solution

```
interpretation( 8, [number = 1,seconds = 0], [
    function(f1(_), [0,4,7,5,2,6,1,3]),
    relation(Q(_,_), [
        1,0,0,0,0,0,0,0,
        0,0,0,0,1,0,0,0,
        0,0,0,0,0,0,0,1,
        0,0,0,0,0,1,0,0,
        0,0,1,0,0,0,0,0,
        0,0,0,0,0,0,1,0,
        0,1,0,0,0,0,0,0,
        0,0,0,1,0,0,0,0])]).
```

(92 solutions found).

Sudoku

Initial State

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

Solution

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

Sudoku in Mace4, Part 1

```
formulas(assumptions).

S(x, y1) = S(x, y2) -> y1 = y2.  % At most one of each in each row.

S(x1, y) = S(x2, y) -> x1 = x2.  % At most one of each in each column.

% "At least" rules.  These are not necessary, but they reduce the search.

all x all z exists y S(x,y) = z.  % At least one of each in each row.

all y all z exists x S(x,y) = z.  % At least one of each in each column.

% For 9x9 puzzles, the intervals are {0,1,2}, {3,4,5}, {6,7,8};
% same_interval(x,y) is an equivalence relation.

same_interval(x,x).
same_interval(x,y) -> same_interval(y,x).
same_interval(x,y) & same_interval(y,z) -> same_interval(x,z).

same_interval(0,1).  same_interval(1,2).
same_interval(3,4).  same_interval(4,5).
same_interval(6,7).  same_interval(7,8).

-same_interval(0,3).  -same_interval(3,6).  -same_interval(0,6).

% The preceding formulas completely specify the same_interval relation.
```

Sudoku in Mace4, Part 2

```
% Rule 3a: At most one of each in each region.

  (
     S(x1, y1) = S(x2, y2) &
     same_interval(x1,x2) &
     same_interval(y1,y2)
  ->
     x1 = x2  &  y1 = y2
  ).

% The initial state of the puzzle.

S(0,0) = 5.   S(0,1) = 3.   S(0,4) = 7.
S(1,0) = 6.   S(1,3) = 1.   S(1,4) = 0.   S(1,5) = 5.
S(2,1) = 0.   S(2,2) = 8.   S(2,7) = 6.
S(3,0) = 8.   S(3,4) = 6.   S(3,8) = 3.
S(4,0) = 4.   S(4,3) = 8.   S(4,5) = 3.   S(4,8) = 1.
S(5,0) = 7.   S(5,4) = 2.   S(5,8) = 6.
S(6,1) = 6.   S(6,6) = 2.   S(6,7) = 8.
S(7,3) = 4.   S(7,4) = 1.   S(7,5) = 0.   S(7,8) = 5.
S(8,4) = 8.   S(8,7) = 7.   S(8,8) = 0.

end_of_list.
```

Sudoku in Mace4, Solution

```
S :
       | 0 1 2 3 4 5 6 7 8
    ---+------------------
     0 | 5 3 4 6 7 8 0 1 2
     1 | 6 7 2 1 0 5 3 4 8
     2 | 1 0 8 3 4 2 5 6 7
     3 | 8 5 0 7 6 1 4 2 3
     4 | 4 2 6 8 5 3 7 0 1
     5 | 7 1 3 0 2 4 8 5 6
     6 | 0 6 1 5 3 7 2 8 4
     7 | 2 8 7 4 1 0 6 3 5
     8 | 3 4 5 2 8 6 1 7 0
```

```
same_interval :
       | 0 1 2 3 4 5 6 7 8
    ---+------------------
     0 | 1 1 1 0 0 0 0 0 0
     1 | 1 1 1 0 0 0 0 0 0
     2 | 1 1 1 0 0 0 0 0 0
     3 | 0 0 0 1 1 1 0 0 0
     4 | 0 0 0 1 1 1 0 0 0
     5 | 0 0 0 1 1 1 0 0 0
     6 | 0 0 0 0 0 0 1 1 1
     7 | 0 0 0 0 0 0 1 1 1
     8 | 0 0 0 0 0 0 1 1 1
```

```
f1 :
       | 0 1 2 3 4 5 6 7 8
    ---+------------------
     0 | 6 7 8 1 2 0 3 4 5
     1 | 4 3 2 6 7 5 0 1 8
     2 | 1 0 5 3 4 6 7 8 2
     3 | 2 5 7 8 6 1 4 3 0
     4 | 7 8 1 5 0 4 2 6 3
     5 | 3 1 4 2 5 7 8 0 6
     6 | 0 2 6 4 8 3 1 5 7
     7 | 5 4 0 7 3 8 6 2 1
     8 | 8 6 3 0 1 2 5 7 4
```

```
f2 :
       | 0 1 2 3 4 5 6 7 8
    ---+------------------
     0 | 6 2 7 8 4 0 1 5 3
     1 | 2 5 4 0 8 3 6 1 7
     2 | 3 6 1 5 0 8 4 7 2
     3 | 5 1 8 2 7 6 0 3 4
     4 | 1 7 5 6 2 4 3 0 8
     5 | 7 3 2 4 5 1 8 6 0
     6 | 0 8 6 1 3 2 7 4 5
     7 | 4 0 3 7 1 5 2 8 6
     8 | 8 4 0 3 6 7 5 2 1
```

Exactly one solution for this puzzle.

Zebra Puzzle

There are five houses in a row. Each has a different nationality, color, drink, pet, and cigarette.

1. The Englishman lives in the red house.
2. The Spaniard owns the dog.
3. The Norwegian lives in the first house on the left.
4. Kools are smoked in the yellow house.
5. The man who smokes Chesterfields lives next to the man with the fox.
6. The Norwegian lives next to the blue house.
7. The Winston smoker owns snails.
8. The Lucky Strike smoker drinks orange juice.
9. The Ukrainian drinks tea.
10. The Japanese smokes Parliaments.
11. Kools are smoked in the house next to the house where the horse is kept.
12. Coffee is drunk in the green house.
13. The Green house is immediately to the right of the ivory house.
14. Milk is drunk in the middle house.

Where does the zebra live, and in which house do they drink water?

Zebra Puzzle, Version 1 for Mace4

```
set(integer_ring).  % <+,-,*> is the ring of integers mod n
set(order_domain).  % "<" is the less-than relation on the domain

formulas(assumptions).  % Assume a domain of size 5: {0,1,2,3,4}  (houses)

% The clues.

    England(x) <-> Red(x).
    Spain(x) <-> Dog(x).
    Norway(0).
    Kool(x) <-> Yellow(x).
    Chesterfield(x) & Fox(y) -> neighbors(x,y).
    Norway(x) & Blue(y) -> neighbors(x,y).
    Winston(x) <-> Snail(x).
    Lucky(x) <-> Juice(x).
    Ukraine(x) <-> Tea(x).
    Japan(x) <-> Parlaiment(x).
    Kool(x) & Horse(y) -> neighbors(x,y).
    Coffee(x) <-> Green(x).
    Green(x) & Ivory(y) -> successor(y,x).
    Milk(2).

% Definitions of successor and neighbor.

    successor(x,y) <-> x+1 = y & x < y.
    neighbors(x,y) <-> successor(x,y) | successor(y,x).
```

```
% Each house has at least one nationality, pet, drink, color, smoke.

  England(x) | Spain(x) | Ukraine(x) | Japan(x) | Norway(x).
  Dog(x) | Snail(x) | Horse(x) | Zebra(x) | Fox(x).
  Water(x) | Milk(x) | Juice(x) | Tea(x) | Coffee(x).
  Red(x) | Blue(x) | Yellow(x) | Ivory(x) | Green(x).
  Lucky(x) | Winston(x) | Kool(x) | Chesterfield(x) | Parlaiment(x).

% Each property applies to at most one house.

  England(x) & England(y) -> x = y.
  Spain(x) & Spain(y) -> x = y.
  Ukraine(x) & Ukraine(y) -> x = y.
  Japan(x) & Japan(y) -> x = y.
  Norway(x) & Norway(y) -> x = y.
  Dog(x) & Dog(y) -> x = y.
  Snail(x) & Snail(y) -> x = y.
  Horse(x) & Horse(y) -> x = y.
  Zebra(x) & Zebra(y) -> x = y.
  Fox(x) & Fox(y) -> x = y.
  Water(x) & Water(y) -> x = y.
  Milk(x) & Milk(y) -> x = y.
  Juice(x) & Juice(y) -> x = y.
  Tea(x) & Tea(y) -> x = y.
  Coffee(x) & Coffee(y) -> x = y.
  Red(x) & Red(y) -> x = y.
  Blue(x) & Blue(y) -> x = y.
  Yellow(x) & Yellow(y) -> x = y.
  Ivory(x) & Ivory(y) -> x = y.
  Green(x) & Green(y) -> x = y.
  Lucky(x) & Lucky(y) -> x = y.
  Winston(x) & Winston(y) -> x = y.
  Kool(x) & Kool(y) -> x = y.
  Chesterfield(x) & Chesterfield(y) -> x =
  Parlaiment(x) & Parlaiment(y) -> x = y.

end_of_list.
```

Zebra Puzzle, Version 2 for Mace4

```
set(integer_ring).   % <+,-,*> is the ring of integers mod n.
set(order_domain).   % "<" is the less-than relation on the domain.

formulas(assumptions).   % Assume a domain of size 5: {0,1,2,3,4}

    England = Red.                  Lucky = Juice.
    Spain = Dog.                    Ukraine = Tea.
    Norway = 0.                     Japan = Parlaiment.
    Kool = Yellow.                  neighbors(Kool,Horse).
    neighbors(Chesterfield,Fox).    Coffee = Green.
    neighbors(Norway,Blue).         successor(Green,Ivory).
    Winston = Snail.                Milk = 2.

% Definitions of successor and neighbors.

    successor(x,y) <-> x+1 = y & x < y.
    neighbors(x,y) <-> successor(x,y) | successor(y,x).

end_of_list.

list(distinct).      % Objects in each list are distinct.
    [England,Spain,Ukraine,Japan,Norway].
    [Dog,Snail,Horse,Zebra,Fox].
    [Water,Milk,Juice,Tea,Coffee].
    [Red,Blue,Yellow,Ivory,Green].
    [Lucky,Winston,Kool,Chesterfield,Parlaiment].
end_of_list.
```

Questions and Problems

- Problem Representation and Structure
  - What can FOL (Mace4) learn from CSP? And vice versa?
  - Mace4 can extended for multisorted FOL (multiple domains)
  - Ordinary arithmetic for Mace4
  - Are there applications for which FOL is better?
  - Find more applications
- Search Methods
  - What can FOL (Mace4) learn from CSP? And vice versa?
  - Local search for Mace4
  - How well does Mace4 scale up?
  - Multiple solutions and isomorphism