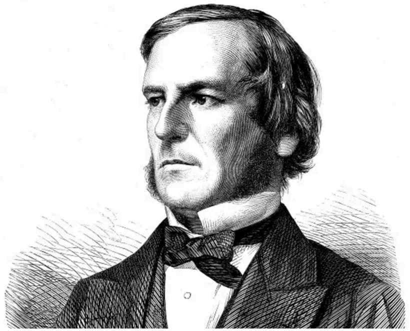
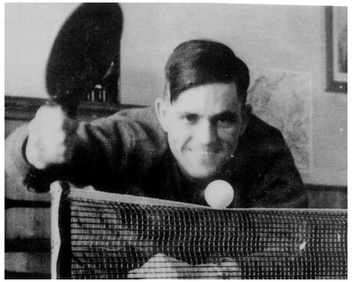
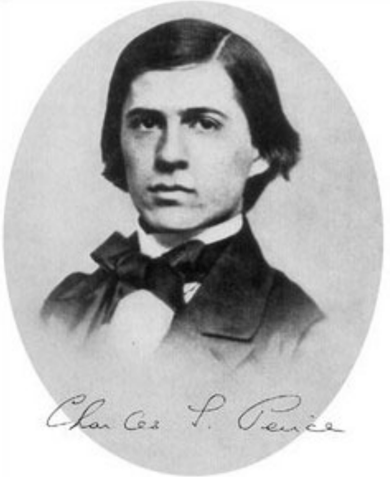
Language



George Boole 1815—1864



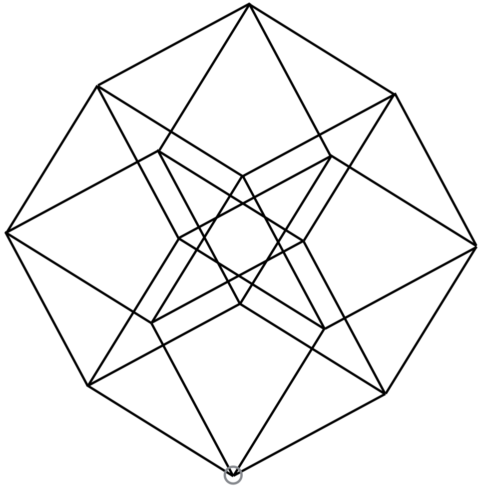
Gerhard Gentzen 1909—1945

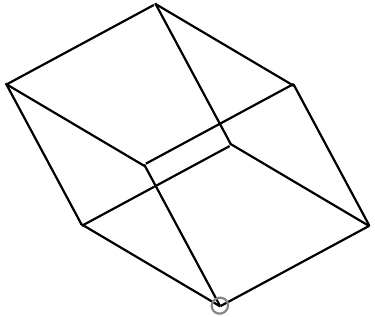


Charles Peirce 1839—1914

https://en.wikipedia.org/wiki/Predicate_(mathematical_logic)

inf1a-cl
Michael Fourman

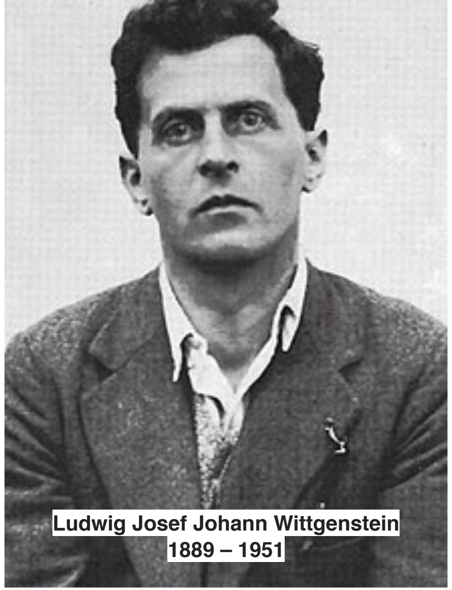




We have been studying
truth and logic

What is language?

syntax + semantics
grammar + meaning

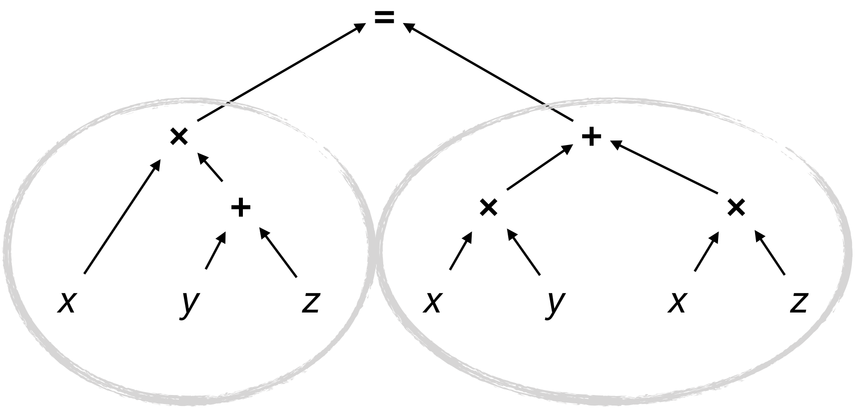


Ludwig Josef Johann Wittgenstein
1889 – 1951

In algebra we make statements about numbers.

$$x (y + z) = x y + x z$$

is a statement

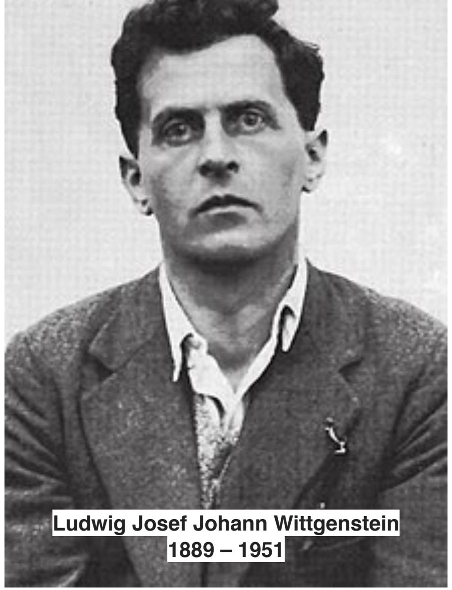


<exp> ::= <var>
| <const>
| <exp> + <exp>
| <exp> × <exp>
| …

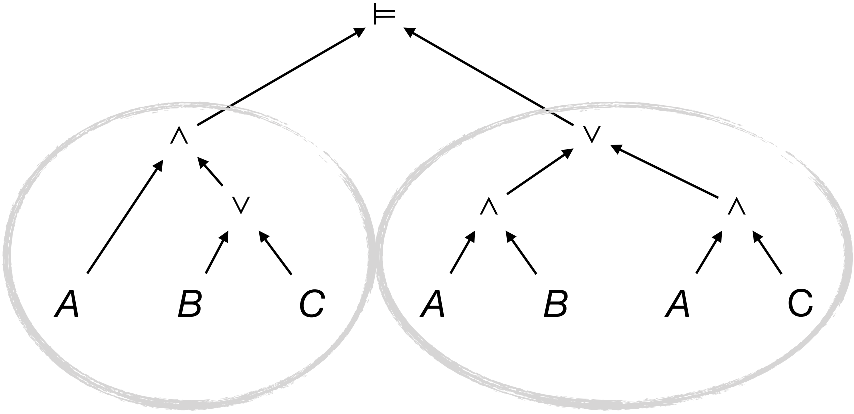
We have been studying
truth and logic

What is language?

syntax + semantics
grammar + meaning



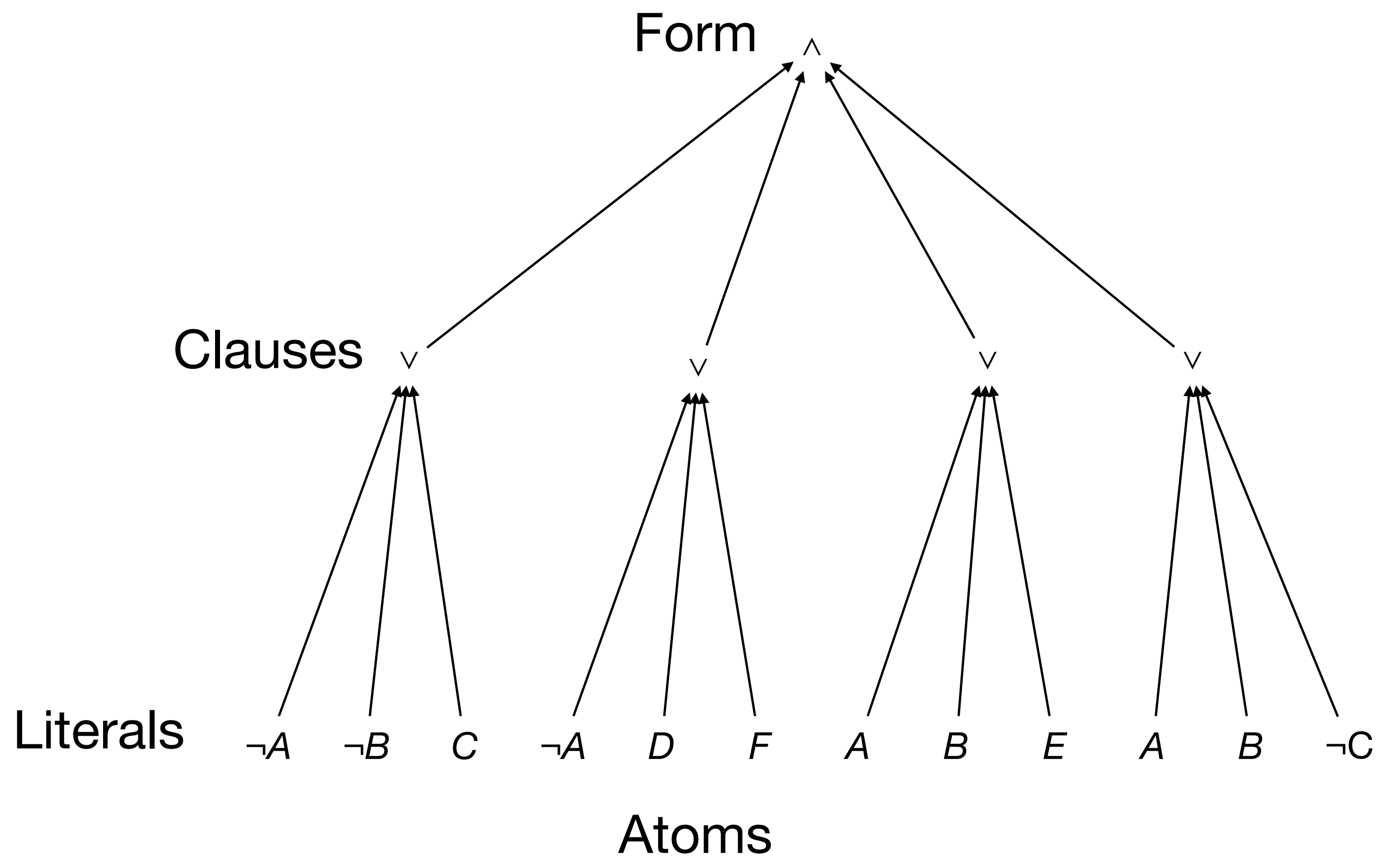
In Logic we make statements about predicates.
$A \wedge (B \vee C) \vDash (A \wedge B) \vee (A \wedge C)$
is a statement

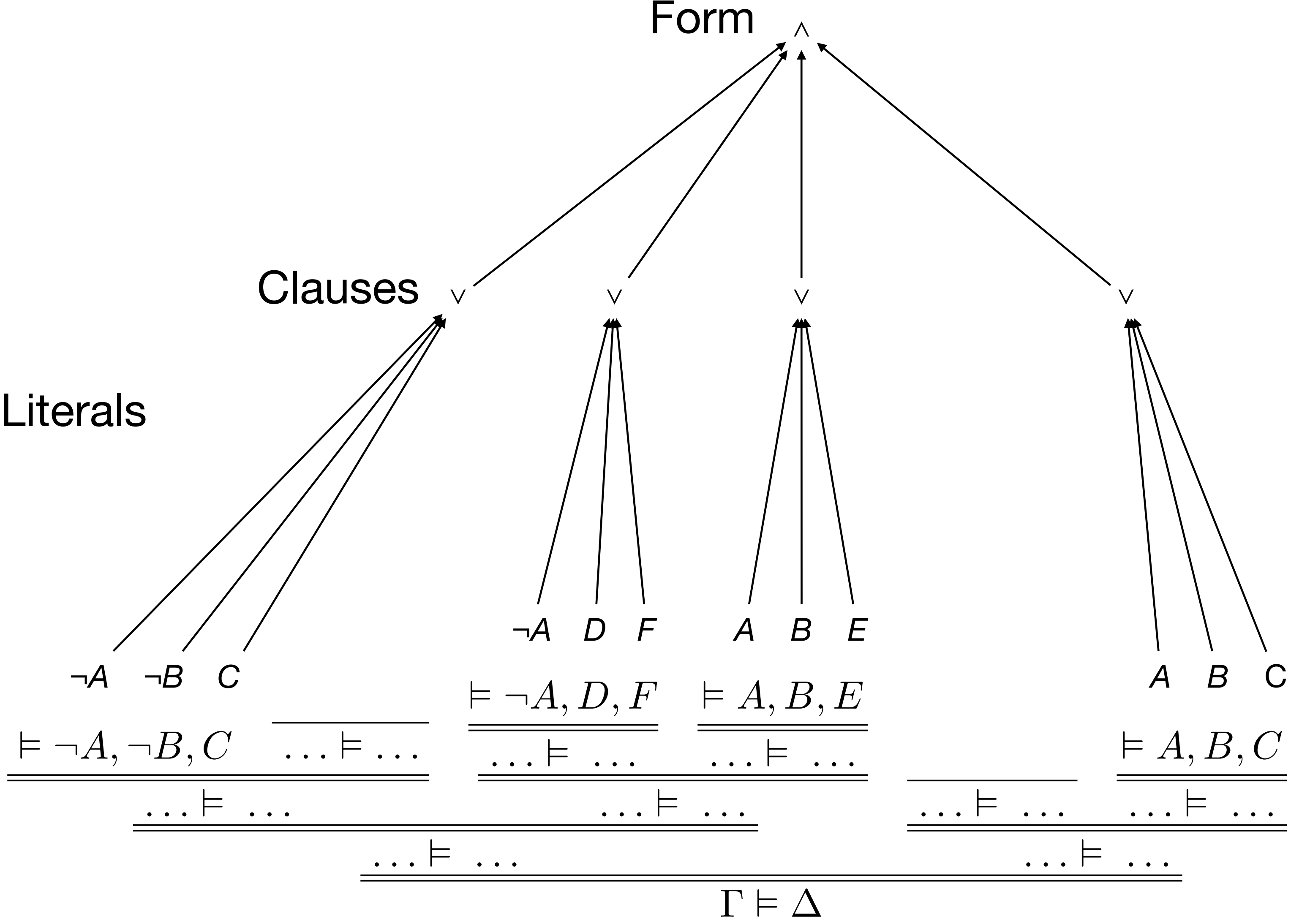


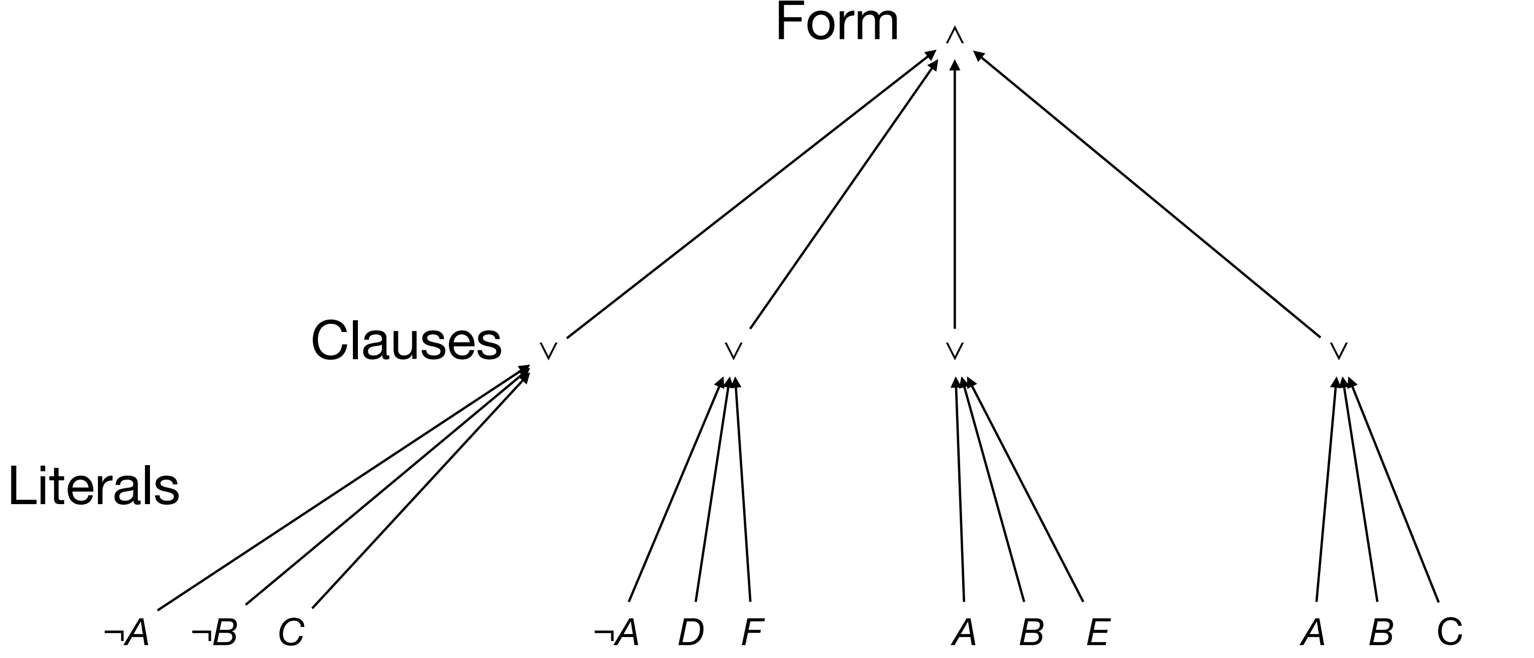
We will formalise this part
of the language
where we use a, b, c
as variables that range
over predicates,
A, B, C, are formal
variables

<exp> ::= <var>
| <exp> ∨ <exp>
| <exp> ∧ <exp>
| …

But we won't yet do that propositional language in Haskell instead, we will do something simpler.




Form
Clauses
Literals
¬A ¬B C
¬A D F
A B E
A B C
⊨ ¬A, ¬B, C
... ⊨ ...
⊨ ¬A, D, F
⊨ A, B, E
⊨ A, B, C
Γ ⊨ Δ



$$\models \neg A, \neg B, C \quad \models \neg A, D, F \quad \models A, B, E \quad \models A, B, \neg C$$

$$\models \neg A, D, F \qquad \models A, B, E$$

$$\models \neg A, \neg B, C \qquad \ldots \models \ldots \qquad \ldots \models \ \ldots \qquad \ldots \models \ \ldots \qquad \models A, B, C$$

$$\ldots \models \ \ldots \qquad \ldots \models \ \ldots \qquad \ldots \models \ \ldots \qquad \ldots \models \ \ldots$$

$$\ldots \models \ \ldots \qquad \ldots \models \ \ldots$$

$$\Gamma \models \Delta$$

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

Idea: to check a sudoku solution

represent the puzzle in logic
s i j k is true iff the entry in square i j is k
We can describe the initial puzzle
s 1 2 7, s 1 6 6, s 2 1 9, s 2 8 4, s 2 9 1
s 3 3 8, s 3 6 9, s 3 8 5, ….
we will check the initial entries are all true

Then check some rules:

```
-- every square is filled
and [ or [ s i j k | k <- [1..9] ]
    | i <- [1..9], j <- [1..9] ]
-- no square is filled twice
and [ or [ not (S i j k), not (s i j k') ]
    | i <- [1..9], j <- [1..9], k <- [1..9],
      k' <- [1..9], k' < k ]
-- and more conditions ...
```

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

Idea:
to solve a sudoku puzzle

represent the puzzle in logic
S i j k is true iff the entry in square i j is k
We can describe the initial puzzle
s 1 2 , s 1 6 6, s 2 1 9, s 2 8 4, s 2 9 1
s 3 3 8, s 3 6 9, s 3 8 5, ….

Write the rules, as constraints, require the initial entries are all true, and solve (find a state that in
entries, and satisfies the constraints)

```
-- every square is filled
And [ Or [ P (S i j k) | k <- [1..9] ]
    | i <- [1..9], j <- [1..9] ]
-- no square is filled twice
And [ Or [ N (S i j k), N (S i j k') ]
    | i <- [1..9], j <- [1..9], k <- [1..9],
      k' <- [1..9], k' < k ]
-- and more conditions ...
```

```
-- every square is filled
and [ or [ s i j k | k <- [1..9] ]
    | i <- [1..9], j <- [1..9] ]
-- no square is filled twice
and [ or [ not (S i j k), not (s i j k') ]
    | i <- [1..9], j <- [1..9], k <- [1..9],
      k' <- [1..9], k' < k ]
-- and more conditions ...
```

translating a checker into a logical specification

```
-- every square is filled
And [ Or [ P (S i j k) | k <- [1..9] ]
    | i <- [1..9], j <- [1..9] ]
-- no square is filled twice
And [ Or [ N (S i j k), N (S i j k') ]
    | i <- [1..9], j <- [1..9], k <- [1..9],
      k' <- [1..9], k' < k ]
-- and more conditions ...
```

We want to find an inhabited model in which all of the following are valid

$$\vDash \neg A, \neg B, C \quad \vDash \neg A, D, F \quad \vDash A, B, E \quad \vDash A, B, \neg C$$

We want to find an inhabited model in which all of the following are valid

$$\models \neg A, \neg B, C \quad \models \neg A, D, F \quad \models A, B, E \quad \models A, B, \neg C$$

We need to find a state Δ such that:

$$\Delta \models \neg A, \neg B, C \quad \Delta \models \neg A, D, F \quad \Delta \models A, B, E \quad \Delta \models A, B, \neg C$$

We start by adding one literal at a time:

We want to find an inhabited model in which all of the following are valid

$$\vDash \neg A, \neg B, C \quad \vDash \neg A, D, F \quad \vDash A, B, E \quad \vDash A, B, \neg C$$

We need to find a state Δ such that:

$$\Delta \vDash \neg A, \neg B, C \quad \Delta \vDash \neg A, D, F \quad \Delta \vDash A, B, E \quad \Delta \vDash A, B, \neg C$$

We start by adding one literal at a time:

$$A \vDash \neg A, \neg B, C \quad A \vDash \neg A, D, F \quad A \vDash A, B, E \quad A \vDash A, B, \neg C$$

We want to find an inhabited model in which all of the following are valid

$$\models \neg A, \neg B, C \quad \models \neg A, D, F \quad \models A, B, E \quad \models A, B, \neg C$$

We need to find a state Δ such that:

$$\Delta \models \neg A, \neg B, C \quad \Delta \models \neg A, D, F \quad \Delta \models A, B, E \quad \Delta \models A, B, \neg C$$

We start by adding one literal at a time:

$$A \models \neg A, \neg B, C \quad A \models \neg A, D, F \quad A \models A, B, E \quad A \models A, B, \neg C$$

And simplify:

$$\frac{A \models \neg B, C}{A \models \neg A, \neg B, C} \quad \frac{A \models D, F}{A \models \neg A, D, F} \quad \frac{}{A \models A, B, E} \quad \frac{}{A \models A, B, \neg C}$$

```
data Literal a = P a | N a deriving Eq
```

The Literal type consists of atoms labelled as positive P or negative N
It's like having two copies of the type a of atoms
and labelling one copy with P and the other with N

We will build formulae with lots of different kinds of atom
the first atom type uses an enumerated type like those we've used before

```
data Atom = A|B|C|D|W|X|Y|Z deriving Eq
P A :: Literal Atom
N B :: Literal Atom
```

For Sudoku we will use symbols $S_{h,i,j,k,e}$ as atoms where the indices are numbers h, i, j, k $[1..3]$, and e $[1..9]$ indicating the entry of the digit e, in position j, k, of the 3×3 square indexed by h, i.

```
data Square = S Int Int Int Int Int
```

For the time being, we use `Atom` for our examples and move on to clauses and forms.

We could simply use a list of lists `[[Literal Atom]]` but we will use lists of Literals in various ways, sometimes as conjunctions and sometimes as disjunctions.

In order not to confuse ourselves, we label a list representing a clause with `Or` so we don't forget.

A Form is a conjunction of Clauses.

Finally, a valuation, Val, is a consistent list of literals.

```
data Atom = A|B|C|D|W|X|Y|Z deriving Eq
data Literal a = P a | N a deriving Eq
data Clause a  = Or [ Literal a ]
data Form a    = And [ Clause a ]
data Val a     = Val [ Literal a ]
```