$$\frac{}{\Gamma, a \vDash \Delta, a}\ (I)$$

$$\frac{\Gamma, a, b \vDash \Delta}{\Gamma, a \wedge b \vDash \Delta}\ (\wedge L) \qquad \frac{\Gamma \vDash a, b, \Delta}{\Gamma \vDash a \vee b, \Delta}\ (\vee R)$$

$$\frac{\Gamma, a \vDash \Delta \quad \Gamma, b \vDash \Delta}{\Gamma, a \vee b \vDash \Delta}\ (\vee L) \qquad \frac{\Gamma \vDash a, \Delta \quad \Gamma \vDash b, \Delta}{\Gamma \vDash a \wedge b, \Delta}\ (\wedge R)$$

$$\frac{\Gamma \vDash a, \Delta}{\Gamma, \neg a \vDash \Delta}\ (\neg L) \qquad \frac{\Gamma, a \vDash \Delta}{\Gamma \vDash \neg a, \Delta}\ (\neg R)$$

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\neg a, \neg c \vee b \vDash \neg a, c}\ I \qquad \cfrac{\cfrac{\cfrac{a, b \vDash c}{b, \vDash \neg a, c}\ \neg R}{b, \neg c \vDash \neg a, c}\ \neg L \qquad \cfrac{a, b \vDash c}{b, b \vDash \neg a, c}\ \neg R}{b, \neg c \vee b \vDash \neg a, c}\ \vee L}{\neg a \vee b, \neg c \vee b \vDash \neg a, c}\ \vee L}{(\neg a \vee b) \wedge (\neg c \vee b) \vDash \neg a \vee c}\ \wedge R}{\vDash \neg((\neg a \vee b) \wedge (\neg c \vee b)), (\neg a \vee c)}\ \neg R}{\vDash \neg((\neg a \vee b) \wedge (\neg c \vee b)) \vee (\neg a \vee c)}\ \vee R$$

$$\frac{\Gamma, a \vDash \Delta \quad \Gamma, b \vDash \Delta}{\Gamma, a \vee b \vDash \Delta}\ (\vee L)$$

$$\frac{\neg a, \neg c \vee b \vDash \neg a, c \quad b, \neg c \vee b \vDash \neg a, c}{\neg a \vee b, \neg c \vee b \vDash \neg a, c}\ \vee L$$

$$\Gamma, a \vee b \vDash \Delta$$

$$\frac{\Gamma, a \vDash \Delta \quad \Gamma, b \vDash \Delta}{\Gamma, a \vee b \vDash \Delta}\ (\vee L)$$

$$\frac{\neg a, \neg c \vee b \vDash \neg a, c \quad b, \neg c \vee b \vDash \neg a, c}{\neg a \vee b, \neg c \vee b \vDash \neg a, c}\ \vee L$$

$$\Gamma, a \vee b \vDash \Delta$$

$\Gamma :=$ $\qquad a :=$ $\qquad b :=$ $\qquad \Delta :=$

$$\frac{\Gamma, a \vDash \Delta \quad \Gamma, b \vDash \Delta}{\Gamma, a \vee b \vDash \Delta}\ (\vee L)$$

$$\frac{\neg a, \neg c \vee b \vDash \neg a, c \quad b, \neg c \vee b \vDash \neg a, c}{\neg a \vee b, \neg c \vee b \vDash \neg a, c}\ \vee L$$

$$\Gamma, a \vee b \vDash \Delta$$

$\Gamma := \{\neg c \vee b\}$ $\quad a := \neg a$ $\quad b := b$ $\quad \Delta := \{\neg a, c\}$

$$\frac{\Gamma, a \vDash \Delta \quad \Gamma, b \vDash \Delta}{\Gamma, a \vee b \vDash \Delta}\ (\vee L)$$

$\Gamma, a \vDash \Delta$ $\qquad$ 3. apply the assignments to the premises of the rule $\qquad$ $\Gamma, b \vDash \Delta$

$\neg c \vee b, \neg a \vDash \neg a, c$ $\qquad$ $\neg c \vee b, b \vDash \neg a, c$

order is irrelevant since Γ, Δ are sets

$$\frac{\neg a, \neg c \vee b \vDash \neg a, c \quad b, \neg c \vee b \vDash \neg a, c}{\neg a \vee b, \neg c \vee b \vDash \neg a, c}\ \vee L$$

1. match these

$$\Gamma, a \vee b \vDash \Delta$$

2. to give these assignments

$\Gamma := \{\neg c \vee b\}$ $\quad a := \neg a$ $\quad b := b$ $\quad \Delta := \{\neg a, c\}$

Evaluating lambda expressions

```
    (\x -> x > 0) 3
=
    3 > 0
=
    True

   (\x -> x * x) 3
=
   3 * 3
=
   9
```

Evaluating lambda expressions

```
    (\x -> x > 0) 3
=
    3 > 0
=
    True

   (\x -> x * x) 3
=
   3 * 3
=
   9
```

Evaluating lambda expressions

The general rule for evaluating lambda expressions is

$$\begin{aligned} & (\lambda x.\, N)\, V \\ = \quad & \\ & N[x := V] \end{aligned}$$

This is sometimes called the β rule (or beta rule).
Here N is an arbitrary expression, V is an arbitrary value, and $N[x := V]$ is N with each free occurrence of x replaced by V.

All you need to know for now is that the following lines have the same effect

```
f x = code
f = (\x -> code)
```

and, after either of those declarations, the following applications have the same results

```
f argument
(\x -> code) argument
```

Evaluating λ-expressions

```
every :: [t] -> (t -> Bool) -> Bool
some  :: [t] -> (t -> Bool) -> Bool
every xs p = and [ p e | e <- xs ]
some  xs p = or  [ p s | s <- xs ]

eg = every xs (\x -> some body (\y -> x `loves` y))

   = and [ (\x -> some body (\y -> x `loves` y)) e | e <- body ]

   = and [ some body (\y -> e `loves` y)) | e <- body ]

   = and [ or [ (\y -> e `loves` y)) s | s <- body] | e <- body ]

   = and [ or [ e `loves` s | s <- body] | e <- body ]
```

We have been studying
truth and logic

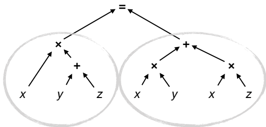
What is language?

syntax + semantics
grammar + meaning

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

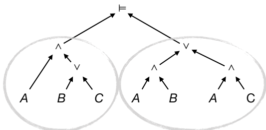
What is language?

syntax + semantics
grammar + meaning

In Logic we make statements about predicates.

$A \wedge (\,B \vee C\,) \vDash (A \wedge B) \vee (A \wedge C)$

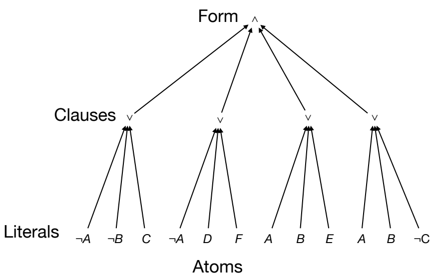
is a statement

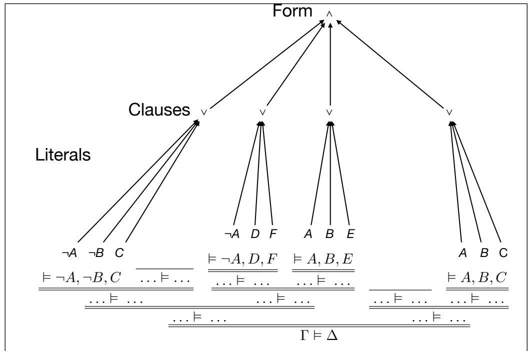


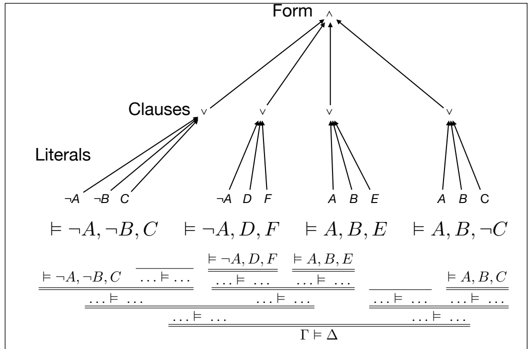
We will formalise this
part of the language
where we use a, b, c
as variables that range
over predicates,
A, B, C, are formal
variables

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

But we don't yet have the (Haskell) tools
instead, we will do
something simple.







	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 includes the initial 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

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

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:

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

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

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:

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

And simplify:

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

```
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
```

The Literal declaration

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 ]
```