First Order Logic

A logic is a formal language with strings over the alphabet Σ:

$$\Sigma = \{\wedge, \vee, \neg, (,), \forall, \exists, x, R_1, \ldots, R_k\}$$

A well-formed formula (WFF) is a string over this alphabet and can be defined inductively as follows. A string ϕ is a WFF if it

- is an atomic formula of the form $R_i(x_1, \ldots, x_n)$ with R_i called a relation symbol, or
- has one of the following forms: $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg\phi_1$, where ϕ_1 and ϕ_2 are smaller formulas, or
- has the form $\exists x_i[\phi_1]$ or $\forall x_i[\phi_1]$, where ϕ_1 is a smaller formula.

Notes: Quantifiers bind variables within their scope (square brackets). If a variable is not bound in a formula then we call it a ***free variable***. Formulas with no free variables are called ***sentences*** or ***statements***. Also it is not necessary to specify the implication operator as part of the alphabet because $(p \rightarrow q) \equiv (\neg p \vee q)$

First Order Logic

WFFs:

- $R_1(x_1) \wedge R_2(x_1, x_2, x_3)$
- $\forall x_1 [R_1(x_1) \wedge R(x_1, x_2, x_3)]$
- $\forall x_1 \exists x_2 \exists x_3 [R_1(x_1) \wedge R_2(x_1, x_2, x_3)]$

Observation: Only the last WFF above is a sentence.

We say, *for all* x_1 *there exist* x_2 *and* x_3 *such that* $R_1(x_1)$ *and* $R_2(x_1, x_2, x_3)$ *are true.*

First Order Logic

We set this system up so that we can reason about sentences such as,

1. $\forall a, b, c, n[(a, b, c > 0 \wedge n > 2) \rightarrow a^n + b^n \neq c^n]$. (Fermat's Last Theorem)
2. $\forall n \exists a, b, c, d[(a, b, c, d, n \geq 0) \rightarrow a^2 + b^2 + c^2 + d^2 = n]$. (Lagrange's Four-Square Theorem)

First order logic is a language rich enough to formalize mathematics, but as we will see, it has its limitations (c.f. *Principia Mathematica*, Whitehead and Russell, 1913).

And it is rich enough to serve as a Turing-complete programming language.

First Order Logic

In order for a logic to make sense we need to assign it meaning. We do this by assigning the syntax to specific mathematical constructs, called the ***model***.

A model is made up of a ***universe*** and a set of relations, one for each relation symbol in the logic.

Example: Let our Σ be

$$\Sigma = \{\wedge, \vee, \neg, (,), \forall, \exists, x, R_1(\cdot, \cdot)\},$$

then a model for this logic is $M_1 = (\mathbb{N}, \leq)$, with $x \mapsto \mathbb{N}$ and $R_1 \mapsto \leq$. Here, $\mathbb{N}$ is the universe and the relation $\leq \in \mathbb{N} \times \mathbb{N}$ is the ***interpretation*** of the binary relation symbol R_1. $\Box$

NOTE: It is not customary to build an explicit interpretation for the logical operators of the language, although we could, but this would just clutter our notation since those interpretation will not change. An explicit model of the logical operators is the universe of boolean values with the standard boolean relations/functions assigned to the boolean operators in the logic.

First Order Logic

Given a logic and let M be a model, we then write $Th(M)$ for the (usually infinite) collection of all sentences in the logic that are true in M. We call $Th(M)$ the *theory of* M.

If some sentence s is true in M then we say that M *satisfies* the sentence s and we write

$$M \models s$$

Observe that

$$M \models s \text{ iff } s \in Th(M)$$

Therefore we often write $Th(M) \models s$ instead of $M \models s$.

First Order Logic

Example: Given our logic $\Sigma = \{\wedge, \vee, \neg, (,), \forall, \exists, x, R_1(\cdot,\cdot)\}$ with the model $M_1 = (\mathbb{N}, \leq)$ with the obvious interpretation, it is easy to show that

$$(\mathbb{N}, \leq) \models \forall x \forall y [R_1(x,y) \vee R_1(y,x)].$$

It is clearly satisfied, since for any assignment $x \mapsto a$ and $y \mapsto b$ for $a, b \in \mathbb{N}$ we have $a \leq b$ or $b \leq a$. $\square$

Example: Given our logic $\Sigma = \{\wedge, \vee, \neg, (,), \forall, \exists, x, R_1(\cdot,\cdot)\}$ with the model $M_2 = (\mathbb{N}, <)$ with the obvious interpretation, we show that sentence

$$(\mathbb{N}, <) \not\models \forall x \forall y [R_1(x,y) \vee R_1(y,x)].$$

In this case the sentence is **not satisfied** because for the assignment $x \mapsto a$ and $y \mapsto a$ for $a \in \mathbb{N}$ we have $a < a$ or $a < a$. $\square$

First Order Logic

Example: Given a logic $\Sigma = \{\wedge, \vee, \neg, (,), \forall, \exists, x, R_1(\cdot, \cdot, \cdot)\}$ with the model $M_3 = (\mathbb{R}, +)$ with the obvious interpretation, we can show that sentence

$$(\mathbb{R}, +) \models \forall y \exists x [R_1(x, x, y)].$$

Here, the sentence is satisfied because for any assignments $x \mapsto a$ and $y \mapsto b$ for $a, b \in \mathbb{R}$ we have $+(a, a, b)$ or in standard mathematical notation $b = a + a$. Note, that if we had chosen a model $(\mathbb{N}, +)$ the sentence would be not be satisfied. $\square$

A Decidable Theory

$\boxed{\textbf{Theorem: } \text{The theory } Th(\mathbb{N},+) \text{ is decidable.}}$

Observations: What does it mean for a theory to be decidable? It means that we can decide whether a particular sentence belongs to the theory or not. That is, we can treat the theory $Th(\mathbb{N},+)$ as a language and we can construct a decider for this language. We can easily show that the following holds,

$$Th(\mathbb{N},+) \models \forall x \exists y[y = x + x].$$

However, switching the quantifiers gives us,

$$Th(\mathbb{N},+) \not\models \exists x \forall y[y = x + x].$$

Therefore this sentence is not a member of the theory. The book shows decidability by constructing an FA that decides the language.

An Undecidable Theory

The following theorem has deep philosophical consequences. It shows that mathematics cannot be mechanized. It also shows that certain problems in number theory are not algorithmic, which was a big surprise to mathematicians in the early 1900's. It was then believed that all problems in mathematics can be solved algorithmically, one just had to come up with a clever algorithm.

Theorem: The theory $Th(\mathbb{N}, +, \times)$ is undecidable.

Observations: This means, there is no algorithm that will halt on all sentences ϕ over the appropriate alphabet. What is perhaps most surprising is the simple structure of this undecidable logic. This means that there are fundamental algorithmic limitations in mathematics. The proof is by reduction of A_{TM} to $Th(\mathbb{N}, +, \times)$.

Proofs

Given a theory $Th(M)$, a proof π of a statement ϕ is a sequence of statements

$$S_1, S_2, \ldots, S_l$$

such that each $S_i \in Th(M)$ and $S_l = \phi$. Furthermore, each S_{i+1} must follow from S_i by an application of an inference rule[a] of that logic.

If ϕ is provable, i.e. it has a proof, then we write

$$Th(M) \vdash \phi$$

If the logic is *sound* then

$$Th(M) \vdash \phi \Rightarrow M \models \phi$$

If the logic is *complete* then

$$M \models \phi \Rightarrow Th(M) \vdash \phi$$

[a] For inference rules for first-order logic see http://en.wikipedia.org/wiki/List_of_rules_of_inference

Sound and Complete Logic

We call a logic sound and complete if for some model M we have

$$Th(M) \vdash \phi \Leftrightarrow M \models \phi$$

Incompleteness Theorem

For the following we assume that proof systems have the following properties,

1. The correctness of a proof of a statement can be checked by machine. Formally,

   $$\{\langle\phi,\pi\rangle | \pi \text{ is a proof of } \phi\}$$

   is decidable.

2. The system of proofs is *sound*.

Incompleteness Theorem

Theorem: The collection of provable statements in $Th(\mathbb{N}, +, \times)$ is Turing-recognizable.

Proof: The following algorithm P accepts its input ϕ if ϕ is provable. The algorithm P tries out every possible string as a candidate for a proof π of ϕ using the proof checker assumed in the previous slide. If it finds a string that is a proof, it accepts. $\square$

Incompleteness Theorem

Kurt Gödels famous incompleteness theorem:

Theorem: Some true statements in $Th(\mathbb{N}, +, \times)$ are not provable.

Proof: Proof by contradiction. We assume that all statements are provable. We show that with this assumption we can construct a machine D that can decide $Th(\mathbb{N}, +, \times)$; a contradiction to our earlier results. We conclude that not all statements are provable.

On input ϕ the algorithm D operates by running algorithm P in parallel on inputs ϕ and $\neg\phi$. One of these two statements is true and thus by our assumption is provable. Therefore P must halt on one of the two inputs. By our assumptions on proof systems, if ϕ is provable, then ϕ is true and if $\neg\phi$ is true then ϕ is false. So algorithm D can decide the truth or falsity of ϕ, that is, D is a decider for $Th(\mathbb{N}, +, \times)$. This is a contradiction, therefore our assumption must be false. $\square$