Second-Order Logic

Consider the first-order formula $\exists\, x\ (Px \to \forall\, x\ Px)$.

This formula is valid, meaning it is true in every model with every variable assignment.

In particular, it is true regardless of how we interpret P.

Because of this, we may be tempted to write $\forall\, P\ \exists\, x\ (Px \to \forall\, x\ Px)$.

Second-order logic allows us to give in to this temptation.

Second-Order Logic: Syntax

The logical symbols of second-order logic include all those used in first-order logic as well as the following:

- Predicate variables: For each positive integer n, we have the n-place predicate variables $X_1^n, X_2^n, \ldots$
- Function variables: For each positive integer n, we have the n-place function variables $F_1^n, F_2^n, \ldots$

The first-order variables $v_1, v_2, \ldots$ will be called *individual* variables to avoid confusion.

A signature Σ consists of constant, function, and predicate symbols exactly as in first-order logic.

Terms

The terms over a signature Σ are defined inductively as follows:

- $U =$ all expressions over the logical symbols and symbols in Σ.

- $B =$ individual variables and constants from Σ.
- $F =$ the set of term-building operations obtained from the function symbols in Σ and the logical function variables.

To be more precise about F, let f be a function symbol (either from Σ or a function variable F_i^n of arity n). The term-building operation $\mathcal{F}_f$ is defined as follows:

$$\mathcal{F}_f(\alpha_1, \ldots, \alpha_n) = f\alpha_1, \ldots, \alpha_n$$

The set of *terms* is the set of expressions generated from the constant symbols and individual variables by the $\mathcal{F}_f$ operations.

Atomic Formulas

Atomic formulas are, as before, a predicate symbol applied to 0 or more terms. The predicate symbol can be equality, a predicate symbol from Σ, or a predicate variable X_i^n.

Formulas

As in first-order logic, we have the following formula-building operations:

- $\mathcal{E}_\neg(\alpha) = (\neg\alpha)$
- $\mathcal{E}_\rightarrow(\alpha, \beta) = (\alpha \rightarrow \beta)$
- $\mathcal{Q}_i(\alpha) = \forall\, v_i\ \alpha$

We add the following additional formula-building operations:

- $\mathcal{Q}^P_{i,n}(\alpha) = \forall\, X_i^n\ \alpha$
- $\mathcal{Q}^F_{i,n}(\alpha) = \forall\, F_i^n\ \alpha$

The set of *well-formed formulas* (*wffs* or just *formulas*) is the set of expressions generated from the atomic formulas by the formula building operations. A free variable is defined as before, and a sentence is a formula with no free variables.

Second-Order Logic: Semantics

A model M serves the same function in second-order logic as it does in first-order logic: it provides a domain and an interpretation for each symbol in the signature Σ.

However, the role of the variable assignment function s must be extended.

Let V be the set of all variables: individual, predicate, or function.

We define a variable assignment s to be a function on V which assigns to each variable in V a suitable object:

- $s(v_i)$ is a member of $dom(M)$,
- $s(X_i^n)$ is an n-ary relation on $dom(M)$, and
- $s(F_i^n)$ is an n-ary function on $dom(M)$.

Given a wff ϕ, we say that M *satisfies* ϕ with s and write $\models_M \phi[s]$ if ϕ is true in the model M with variable assignment s.

We now define this formally.

Terms

We first define the extension $\overline{s}$, a function from the set of all terms into the domain of M:

1. For each variable x, $\overline{s}(x) = s(x)$.
2. For each constant symbol c, $\overline{s}(c) = c^M$.
3. If $t_1,\ldots,t_n$ are terms and f is an n-ary function symbol from Σ, then $\overline{s}(ft_1,\ldots,t_n) = f^M(\overline{s}(t_1),\ldots,\overline{s}(t_n))$.
4. If $t_1,\ldots,t_n$ are terms, then $\overline{s}(F_i^n t_1,\ldots,t_n) = s(F_i^n)(\overline{s}(t_1),\ldots,\overline{s}(t_n))$.

Atomic Formulas

1. $\models_M t_1 = t_2[s]$ iff $\overline{s}(t_1) = \overline{s}(t_2)$.
2. For an n-ary predicate symbol P from Σ,
   $\models_M Pt_1, \ldots, t_n[s]$ iff $\langle \overline{s}(t_1), \ldots, \overline{s}(t_n) \rangle \in P^M$.
3. $\models_M X_i^n t_1, \ldots, t_n[s]$ iff $\langle \overline{s}(t_1), \ldots, \overline{s}(t_n) \rangle \in s(X_i^n)$.

Formulas

1. $\models_M (\neg\phi)[s]$ iff $\not\models_M \phi[s]$.
2. $\models_M (\phi \to \psi)[s]$ iff $\not\models_M \phi[s]$ or $\models_M \psi[s]$.
3. $\models_M \forall x\, \phi[s]$ iff $\models_M \phi[s(x|d)]$ for every $d \in \mathit{dom}(M)$.
4. $\models_M \forall X_i^n\, \phi[s]$ iff $\models_M \phi[s(X_i^n|R)]$ for every n-ary relation R on $\mathit{dom}(M)$.
5. $\models_M \forall F_i^n\, \phi[s]$ iff $\models_M \phi[s(F_i^n|f)]$ for every n-place function f on $\mathit{dom}(M)$.

where, as before, $s(x|d)$ signifies the function which is the same as s everywhere except at x where its value is d.

For notational convenience, we will omit the subscripts and superscripts from the X and F variables when they are unnecessary.

Second-Order Logic

Recall that for each $n \geq 2$, we have a first-order sentence λ_n which translates, "there are at least n distinct objects":

$\lambda_2 = \exists x\, \exists y\, x \neq y,$
$\lambda_3 = \exists x\, \exists y\, \exists z\, (x \neq y \wedge x \neq z \wedge y \neq z),$
$\ldots$

Earlier, we showed that the class of all infinite models is axiomatized by $\lambda_2, \lambda_3, \ldots$, but is not axiomatized by any single first-order sentence.

However, there *is* a second-order sentence which axiomatizes the infinite models. A set is infinite iff there is an ordering on it having no last element:

$$\exists X\ [\forall u \forall v \forall w\,(Xuv \to Xvw \to Xuw) \wedge \forall u \neg Xuu \wedge \forall u \exists v\, Xuv].$$

Alternatively, a set is infinite if there is a one-to-one function that is not onto:

$$\exists F\ [\forall x \forall y\,(Fx = Fy \to x = y) \wedge \exists z \forall x\, Fx \neq z].$$

Second-Order Logic

The previous example shows that the compactness theorem fails for second-order logic.

Theorem

There is an unsatisfiable set of second-order sentences, every finite subset of which is satisfiable.

Proof

Let λ_∞ be a second-order sentence which axiomatizes the infinite models. Consider the set $\{\neg\lambda_\infty, \lambda_2, \lambda_3, \ldots\}$. Every finite subset of this set is satisfiable in some finite model. However, the entire set is not satisfiable in any finite model or infinite model.

The Löwenheim-Skolem theorem also fails for second-order logic. For example, there is a sentence in the second-order language of equality (empty signature) that is true in a model iff its cardinality is $2^{\aleph_0}$.

Prenex Normal Form

A *prenex* formula is one of the form $Q_1 x_1 \cdots Q_n x_n \alpha$, where each Q_i is a quantifier and α is quantifier-free.

Theorem

For any formula we can find a logically equivalent prenex formula.

Proof

We use the following quantifier identities which are easy to prove.

- $\neg \forall x\, \alpha \leftrightarrow \exists x\, \neg \alpha$
- $\neg \exists x\, \alpha \leftrightarrow \forall x\, \neg \alpha$
- $(\alpha \rightarrow \forall x\, \beta) \leftrightarrow \forall x\, (\alpha \rightarrow \beta)$, for x not free in α.
- $(\alpha \rightarrow \exists x\, \beta) \leftrightarrow \exists x\, (\alpha \rightarrow \beta)$, for x not free in α.
- $(\forall x\, \alpha \rightarrow \beta) \leftrightarrow \exists x\, (\alpha \rightarrow \beta)$, for x not free in β.
- $(\exists x\, \alpha \rightarrow \beta) \leftrightarrow \forall x\, (\alpha \rightarrow \beta)$, for x not free in β

By using these identities together with renaming of bound variables, it is easy to manipulate any formula to a logically equivalent prenex formula.

Skolemization

Skolem Normal Form Theorem

For any first-order formula, there is a logically equivalent second-order formula consisting of:

1. First a string (possibly empty) of existential individual and function quantifiers, followed by
2. A string (possibly empty) of universal individual quantifiers, followed by
3. A quantifier-free formula

Proof

First, given any first order formula, we can find an equivalent formula ϕ in prenex normal form. If ϕ is in the form described above, we are done.

Otherwise, ϕ must contain a sub-formula of the form $\forall v_1 \ldots \forall v_n \exists z \, \psi(z)$.

This formula is equivalent to $\exists F \forall v_1 \ldots \forall v_n \, \psi(F v_1 \ldots v_n)$.

By repeatedly applying this rule (from the outside inwards), every nested existential quantifier can be eliminated. The result is a formula of the form described above.

Recall that a universal formula is a first-order prenex formula all of whose quantifiers are universal. Similarly, an existential formula is a first-order prenex formula all of whose quantifiers are existential.

Corollary

For any first-order formula ϕ, there is a universal formula θ in an expanded language containing function symbols such that ϕ is satisfiable iff θ is satisfiable.

Corollary

For any first-order formula ϕ, there is an existential formula θ in an expanded language containing function symbols such that ϕ is valid iff θ is valid.

Example

Find an equisatisfiable universal formula:

$$\exists y_1 \forall x_1 \; [(\forall y_2 \exists x_2 \, x_1 + y_2 < x_2) \rightarrow (\forall x_3 \exists y_3 \, y_1 + x_3 > y_3)]$$

iff

$$\exists y_1 \forall x_1 \exists y_2 \forall x_2 \forall x_3 \exists y_3 \; [(x_1 + y_2 < x_2) \rightarrow (y_1 + x_3 > y_3)]$$

iff

$$\exists y_1 \exists F_1 \forall x_1 \forall x_2 \forall x_3 \exists y_3 \; [(x_1 + F_1 x_1 < x_2) \rightarrow (y_1 + x_3 > y_3)]$$

iff

$$\exists y_1 \exists F_1 \exists F_2 \forall x_1 \forall x_2 \forall x_3 \, [(x_1 + F_1 x_1 < x_2) \rightarrow (y_1 + x_3 > F_2 x_1 x_2 x_3)]$$

which is satisfiable iff

$$\forall x_1 \forall x_2 \forall x_3 \; [(x_1 + F_1 x_1 < x_2) \rightarrow (y_1 + x_3 > F_2 x_1 x_2 x_3)]$$

is satisfiable.

Skolemization

This also gives us undecidability results for universal and existential formulas.

Corollary

1. The set of Gödel numbers of satisfiable universal first-order sentences is not recursive.
2. The set of Gödel numbers of valid existential first-order sentences is not recursive.

Proof

(2) Given any sentence σ, we can effectively find an existential sentence that is valid iff σ is valid. Thus, if we could solve the validity problem for existential sentences, we could solve it for arbitrary sentences, contradicting Church's Theorem.

Resolution

Recall that a literal is an atomic formula or its negation, a clause is a disjunction of literals, and a formula in CNF consists of a conjunction of clauses.

Earlier, we discussed propositional resolution, in which two propositional clauses are resolved to form a new *resolvent* clause according to the following principle:

> For any two clauses C_1 and C_2, if there is a literal L_1 in C_1 that is complementary to a literal L_2 in C_2, then delete L_1 and L_2 from C_1 and C_2, respectively, and construct the disjunction of the remaining clauses.

The resolution principle can be used to form a decision procedure for satisfiability of a formula in propositional logic:

- Given a formula ϕ, convert it to an equisatisfiable CNF formula, and let Φ be the set of clauses.
- While two clauses in Φ can be resolved to obtain a new clause not in Φ, add the resolvent to Φ
- If the empty clause is in Φ, return *unsatisfiable*.
- Return *satisfiable*.

The method of skolemization leads naturally to a semi-decision procedure for satisfiability of arbitrary first-order logic formulas.

- Given a first-order formula ϕ, skolemize ϕ to obtain a universal formula which is satisfiable iff ϕ is satisfiable.
- Drop the universal quantifiers to obtain a quantifier-free formula ψ.
- Convert ψ to conjunctive normal form.
- Use first-order resolution to check the CNF formula.

First-order resolution differs from propositional resolution because it requires an additional *unification* step.

A *substitution* is a finite set of the form $\{t_1/v_1, \ldots, t_n/v_n\}$. If θ is a substitution and E is an expression, then $E\theta$ is the expression obtained by substituting each t_i for each v_i in E.

A substitution θ is called a *unifier* for E_1 and E_2 iff $E_1\theta = E_2\theta$.

E_1 and E_2 are said to be *unifiable* if they have a unifier.

It is not hard to define an algorithm which determines whether two expressions are unifiable and computes the unifier for them.

Suppose we have

$C = c_1 \vee \cdots \vee c_k \vee P(s_1, \ldots, s_m)$ and

$D = d_1 \vee \cdots \vee d_l \vee \neg P(t_1, \ldots, t_m)$

such that $\{(s_i, t_i) \mid i = 1 \ldots m\}$ is unifiable.

We can compute a unifier θ and then apply propositional resolution to the two clauses, resulting in the new clause

$[\theta](c_1 \vee \cdots \vee c_k \vee d_1 \vee \cdots \vee d_l)$

(where $[\theta](S)$ denotes applying the substitution θ to the expression S).

Most of the time, this naive resolution rule is sufficient.

Example with naive resolution

Consider the formula:

$(\neg P(x) \vee P(f(x))) \wedge (\neg P(f(f(y))) \vee Q(y))$.

One possible ground instance to which propositional resolution may be applied is:

$(\neg P(f(g(c))) \vee P(f(f(g(c))))) \wedge (\neg P(f(f(g(c)))) \vee Q(g(c)))$,

which resolves to

$(\neg P(f(g(c))) \vee Q(g(c)))$.

On the other hand, if we resolve using a *most general unifier* $(x \leftarrow f(y))$, we get the clause

$(\neg P(f(y)) \vee Q(y))$

of which the above resolvent is just an instance.

Unfortunately, there are some examples for which naive resolution is insufficient.

Consider the following formula:

$$(P(x,x) \vee P(b,x)) \wedge (\neg P(x,x) \vee \neg P(b,x)).$$

If we want to perform naive resolution, there are four possible pairs of complimentary literals. But the result for each pair is a trivial tautology:

$$\begin{aligned}
(P(x,x), \neg P(x,x)) &\rightarrow (P(b,x) \vee \neg P(b,x)) \\
(P(x,x), \neg P(b,x)) &\rightarrow (P(b,b) \vee \neg P(b,b)) \\
(P(b,x), \neg P(x,x)) &\rightarrow (P(b,b) \vee \neg P(b,b)) \\
(P(b,x), \neg P(b,x)) &\rightarrow (P(b,b) \vee \neg P(b,b))
\end{aligned}$$

What went wrong?

What went wrong? One ground instance of this formula is

$$(P(b,b) \vee P(b,b)) \wedge (\neg P(b,b) \vee \neg(P(b,b)).$$

Applying propositional resolution to this instance results in the empty clause.

The problem is that an instance may unify literals in the same clause as well as complimentary literals in different clauses, so the first order resolution rule must be modified to take this into account.

Resolution

If S is a set of literals, let S^- denote $\{-p \mid p \in S\}$ (recall that $-p$ is the negation of p where double negation is collapsed: $-\neg q \equiv q$).

Let A and B be two first order clauses (represented as sets of literals). Let A_0 and B_0 be copies of A and B respectively, in which the variables have been renamed so that A_0 and B_0 have no variables in common.

A *first order resolvent* of A and B is any clause $[\theta]((A_0 - A_1) \cup (B_0 - B_1))$ where A_1 and B_1 are nonempty subsets of A_0 and B_0 respectively such that θ is a most general unifier for *all* the literals in $A_1 \cup B_1^-$.

Naive resolution is just an instance of this rule in which A_1 and B_1 each contain only a single literal.

A *first order resolvent* of A and B is any clause $[\theta]((A_0 - A_1) \cup (B_0 - B_1))$ where A_0 and B_0 are renamed copies of A and B respectively and A_1 and B_1 are nonempty subsets of A_0 and B_0 respectively such that θ is a most general unifier for *all* the literals in $A_1 \cup B_1^-$.

Consider again our previous example for which naive resolution failed:

$$(P(x,x) \vee P(b,x)) \wedge (\neg P(x,x) \vee \neg P(b,x)).$$

We first rename to get:

$$(P(x,x) \vee P(b,x)) \wedge (\neg P(y,y) \vee \neg P(b,y)),$$

so that $A_0 = \{P(x,x), P(b,x)\}$ and $B_0 = \{\neg P(y,y), \neg P(b,y)\}$.

If we choose A_1 and B_1 to be single literals, we will fail as before.

But if we choose $A_1 = A_0$ and $B_1 = B_0$ and $\theta = (x \leftarrow b, y \leftarrow b)$, we can successfully produce the empty clause.

Theorem

Suppose A and B are first order clauses, A' and B' are instances and C' is a propositional resolvent of A' and B'. Then there exists a first order resolvent C of A and B such that C' is an instance of C.

Corollary

If a set S of first order clauses is inconsistent, then the empty clause is derivable using (only) first order resolution.

Many-Sorted Logic

In standard first-order logic, the domain of a model consists of a single set of objects.

However, often it is more intuitive to have several sets, one for each of several different kinds of objects.

Many-sorted logic allows us to do exactly this.

A *sort* is very much like a *type* in programming languages.

Many-Sorted Logic: Syntax

The syntax of many-sorted logic is identical to that of regular first-order logic.

However, additional information is given which can be used to determine whether a formula is *well-sorted*.

Let $S = \{s_1, \ldots, s_n\}$ be a set of *sorts*.

For each variable v_k, we associate a sort $sort(v_k) = s_i$.

Suppose Σ is a signature.

- Each constant $c \in \Sigma$ has an associated sort $sort(c) = s_i$.
- Each function symbol $f \in \Sigma$ of arity n has a sort which is an $n+1$-tuple $sort(f) = \langle s_{i_1}, \ldots, s_{i_n}, s_{i_{n+1}} \rangle$.
- Each predicate symbol $p \in \Sigma$ of arity n has a sort which is an n-tuple $sort(p) = \langle s_{i_1}, \ldots, s_{i_n} \rangle$.

We define $\overline{sort}$, a function from terms to sorts as follows:

- If t is a variable or constant term, then $\overline{sort}(t) = sort(t)$.
- If $t = ft_1 \ldots t_n$, then $\overline{sort}(t) = s_{i_{n+1}}$, where $sort(f) = \langle s_{i_1}, \ldots, s_{i_n}, s_{i_{n+1}} \rangle$.

Given a signature Σ, a set of sorts S, and a sort function $sort$, we can determine whether a Σ-formula is well-sorted as follows.

We define a function $well$ from expressions to $\{true, false\}$.

We first define $well$ for terms.

- For every variable or constant term t, $well(t) = true$.
- If $t = ft_1 \ldots t_n$ and $sort(f) = \langle s_{i_1}, \ldots, s_{i_n}, s_{i_{n+1}} \rangle$, then $well(t) = well(t_1) \wedge \cdots \wedge well(t_n) \wedge \overline{sort}(t_1) = s_{i_1} \wedge \cdots \wedge \overline{sort}(t_n) = s_{i_n}$.

For atomic formulas,

- $well(t_1 = t_2) = well(t_1) \wedge well(t_2) \wedge \overline{sort}(t_1) = \overline{sort}(t_2)$.
- If $sort(p) = \langle s_{i_1}, \ldots, s_{i_n} \rangle$, then $well(pt_1 \ldots t_n) = well(t_1) \wedge \cdots \wedge well(t_n) \wedge \overline{sort}(t_1) = s_{i_1} \wedge \cdots \wedge \overline{sort}(t_n) = s_{i_n}$.

A formula is well-sorted iff each of its atomic formulas is well-sorted.

Many-Sorted Logic

Example

Let $\Sigma = \{1, +, car, cdr\}$, $S = \{\mathcal{N}, \mathcal{L}\}$, and suppose we have variables v_i and l_i. Define $sort$ as follows:

- $sort(v_i) = \mathcal{N}$, $sort(l_i) = \mathcal{L}$
- $sort(1) = \mathcal{N}$
- $sort(+) = \langle \mathcal{N}, \mathcal{N}, \mathcal{N} \rangle$
- $sort(car) = \langle \mathcal{L}, \mathcal{N} \rangle$
- $sort(cdr) = \langle \mathcal{L}, \mathcal{L} \rangle$

Are the following well-sorted?

- $1 + v_5$ *yes*
- $car(1 + v_5)$ *no*
- $cdr(l_5) + 1$ *no*
- $car(l_5) + 1$ *yes*
- $car(l_3) = 1 + car(cdr(l_3))$ *yes*

Many-Sorted Logic: Semantics

Given a signature Σ, a set of sorts S, and a sorting function $sort$, a *many-sorted model* consists of the following:

1. For each $s_i \in S$, a nonempty set called the *domain* of s_i, written $dom(s_i)$.
2. A mapping from each constant c in Σ of sort $sort(c) = s_i$ to an element c^M of $dom(s_i)$.
3. A mapping from each n-ary function symbol f in Σ of sort $sort(f) = \langle s_{i_1}, \ldots, s_{i_n}, s_{i_{n+1}} \rangle$ to f^M, an n-ary function from $dom(s_{i_1}) \times \ldots \times dom(s_{i_n})$ to $dom(s_{i_{n+1}})$.
4. A mapping from each n-ary predicate symbol p in Σ of sort $sort(p) = \langle s_{i_1}, \ldots, s_{i_n} \rangle$ to an n-ary relation $p^M \subseteq dom(s_{i_1}) \times \ldots \times dom(s_{i_n})$.

The definitions of truth and satisfaction are analagous to the standard ones. The only one that is slightly different is that $\forall\, v_i\ \phi$ is interpreted as taking v_i to range over all the elements of $dom(sort(v_i))$.

Many-Sorted Logic

Though convenient, many-sorted logic does not give us any more power than regular first-order logic.

Let Σ be a signature, S a set of sorts, and $sort$ a sorting function.

Let Σ^* be a new signature which includes Σ as well as a new predicate Q_{s_i} for each $s_i \in S$.

Given a Σ-formula ϕ, define ϕ^* to be ϕ with every instance of $\forall\, v_i\ \psi$ replaced with $\forall\, v_i\ (Q_{sort(v_i)}(v_i) \rightarrow \psi)$.

Many-Sorted Logic

Given a many-sorted model M, define a standard model M^* as follows.

- $dom(M^*)$ is the union of $dom(s_i)$, for $s_i \in S$.
- For constant symbols c, $c^{M^*} = c^M$.
- For function symbols f, f^{M^*} is an arbitrary extension of f^M to a total function on $dom(M^*)$.
- For predicate symbols p, $p^{M^*} = p^M$.
- For the new predicate symbols Q_{s_i}, $Q_{s_i}^{M^*} = dom(s_i)$

Theorem

A many-sorted sentence σ is true in M iff σ^* is true in M^*.

Proof

Use induction to show that $\models_M \phi[s]$ iff $\models_{M^*} \phi^*[s]$.