Logic and Proof

First-order resolution

This note aims to give a detailed, step-by-step explanation of first-order resolution. While the resolution procedure itself is not complicated – and being proficient at negation, Skolemisation, resolution and unification is enough to handle most proofs thrown at you – understanding the motivation for introducing all these techniques and how they fit together will give you a deeper grasp of what is going on behind the scenes why first-order resolution works in the first place – insight that will be especially helpful when encountering trickier questions that involve factoring, variable renaming, etc.

Out of the two clausal proof techniques discussed in Section 6, only resolution can be adapted to first-order logic. In fact, we're not even really adapting the proof technique, but changing the problem itself to something that can be handled by propositional resolution. The two first-order concepts that need to be adapted or eliminated are *variables* and *quantifiers*. FOL formulas without either are just logically connected relation symbols applied to function symbols applied to constant symbols; they are essentially propositional formulas, where the propositional variables have the form $P(a)$, $Q(f(b), g(g(c)))$, etc. More formally, given a first-order language, we can generate the set of *ground atoms* (formulas without any connectives) from the Herbrand universe of the language, by applying every relation symbol to every ground term (in H) in all possible ways. The resulting set (called the *Herbrand base*) can be treated as the set of propositional variables for a propositional logic – that is, formulas now look like

$$((P(a) \to Q(f(b), g(g(c)))) \to P(a)) \to P(a)$$

(instead of just $((P \to Q) \to P) \to P$), but, due to the absence of variables or quantifiers, they are perfectly propositional. If we can transform first-order formulas into clauses containing such ground atoms only, we can do standard propositional resolution (in this case called *ground resolution*) to get refutation proofs.

This is all well and good, but it depends on us being able to translate arbitrary first-order formulas such as

$$\Big(\forall x.\, \exists y.\, \forall z.\, (P(\underline{x}) \land Q(\underline{z}, \underline{y})) \lor (\exists w.\, R(\underline{w}))\Big) \leftrightarrow \exists y.\, \forall v.\, S(\underline{y}, \underline{v})$$

into a set of clauses containing *only* ground atoms. This is certainly a nontrivial task, and we need several tools and some meta-theoretic results to be able to pull it off. The main issues are:

- Formulas can have quantifiers in arbitrary positions
- Formulas can have variables
- It's not clear what conjunctive normal form is in the presence of quantifiers
- Not every FOL formula is logically equivalent to one in CNF
- Where do we get a set of ground clauses from
- How does reasoning only about ground clauses justify making any conclusions about the original first-order formula

We will go through the full first-order resolution procedure in detail, highlighting how these issues are handled and why they are correct. As a running example, we will consider the following formula (based on Exercise 8.4):

$$\exists x.\ \forall y.\ (P(y) \to \exists x.\ Q(x)) \to (P(x) \to Q(x))$$

Negate and convert to NNF

As before, we have to start by negating the formula to get a refutation proof. This is usually followed by several applications of de Morgan's laws for connectives and quantifiers, as well as rewriting $\to$ and $\leftrightarrow$ to use $\wedge$ and $\vee$, until the formula resembles NNF with quantifiers (only $\wedge$ and $\vee$ connectives, only negated atoms, quantifiers in any position).

$$\begin{aligned}
&\neg\exists x.\ \forall y.\ (P(y) \to \exists x.\ Q(x)) \to (P(x) \to Q(x)) \\
\simeq\ &\forall x.\ \exists y.\ \neg(P(y) \to \exists x.\ Q(x)) \to (P(x) \to Q(x)) && \text{(quantifier de Morgan)} \\
\simeq\ &\forall x.\ \exists y.\ (P(y) \to \exists x.\ Q(x)) \wedge P(x) \wedge \neg Q(x) && \text{(negation of implication)} \\
\simeq\ &\forall x.\ \exists y.\ (\neg P(y) \vee \exists x.\ Q(x)) \wedge P(x) \wedge \neg Q(x) && \text{(implication as disjunction)}
\end{aligned}$$

Adjust quantifier scope

We have two choices on where to move the quantifiers in the resulting formula. They can be achieved by repeated applications of the standard "pulling quantifiers through connectives" laws, as long as we ensure that there is no inadvertent variable capture – we can avoid this by α-renaming bound variables to ensure distinct names. One choice is *prenex normal form*, where all quantifiers are collected on the outside: e.g. $\forall x.\ \exists y.\ \forall z.\ \varphi(x, y, z)$, where $\varphi(x, y, z)$ does not contain quantifiers. The advantages of prenex form is that the inner formula (sometimes called the matrix) is quantifier-free and can be converted to CNF as if it was propositional. However, it maximises the scope of every quantifier, leading to potential (and not uncommon) complications with Skolemisation.

$$\begin{aligned}
&\forall x.\ \exists y.\ (\neg P(y) \vee \exists x.\ Q(x)) \wedge P(x) \wedge \neg Q(x) \\
\simeq\ &\forall x.\ \exists y.\ (\neg P(y) \vee \exists z.\ Q(z)) \wedge P(x) \wedge \neg Q(x) && (\alpha\text{-renaming}) \\
\simeq\ &\forall x.\ \exists y.\ (\exists z.\ \neg P(y) \vee Q(z)) \wedge P(x) \wedge \neg Q(x) && (\text{pull } \exists \text{ through } \vee) \\
\simeq\ &\forall x.\ \exists y.\ \exists z.\ (\neg P(y) \vee Q(z)) \wedge P(x) \wedge \neg Q(x) && (\text{pull } \exists \text{ through } \wedge)
\end{aligned}$$

In practice, a better approach is to convert it into *miniscope form*, where the scope of every quantifier is as small as possible. The distributivity laws of $\forall$ over conjunction, and $\exists$ over disjunction are useful here.

$$\begin{aligned}
&\forall x.\ \exists y.\ (\neg P(y) \vee \exists x.\ Q(x)) \wedge P(x) \wedge \neg Q(x) \\
\simeq\ &\forall x.\ \exists y.\ (\neg P(y) \vee \exists z.\ Q(z)) \wedge P(x) \wedge \neg Q(x) && (\alpha\text{-renaming}) \\
\simeq\ &\forall x.\ (\exists y.\ \neg P(y) \vee \exists z.\ Q(z)) \wedge P(x) \wedge \neg Q(x) && (\text{push } \exists \text{ under } \wedge) \\
\simeq\ &\forall x.\ ((\exists y.\ \neg P(y)) \vee \exists z.\ Q(z)) \wedge P(x) \wedge \neg Q(x) && (\text{push } \exists \text{ under } \vee) \\
\simeq\ &((\exists y.\ \neg P(y)) \vee \exists z.\ Q(z)) \wedge (\forall x.\ P(x) \wedge \neg Q(x)) && (\text{push } \forall \text{ under } \wedge) \\
\simeq\ &((\exists y.\ \neg P(y)) \vee \exists z.\ Q(z)) \wedge (\forall x.\ P(x)) \wedge (\forall x.\ \neg Q(x)) && (\text{distribute } \forall \text{ over } \wedge)
\end{aligned}$$

Any FOL formula is logically equivalent to one in prenex or miniscope form. In prenex form conversion to CNF is straightforward – while this is not the case in miniscope form, the conversion can be delayed until after Skolemisation.

Eliminate existential quantifiers

While there is no way to convert a FOL formula to a logically equivalent one without any existential quantifiers, it *is* possible to do that while preserving consistency. This is precisely the role of Skolemisation: it changes the meaning of the formula, but doesn't alter whether it is satisfiable or not. As mentioned above, Skolemisation is easier if the formula is in miniscope form, as the number of arguments the Skolem functions must take will be minimised.

$$\begin{aligned} &((\exists y.\, \neg P(y)) \vee \exists z.\, Q(z)) \wedge (\forall x.\, P(x)) \wedge (\forall x.\, \neg Q(x)) \\ \simeq\ &(\neg P(a) \vee Q(b)) \wedge (\forall x.\, P(x)) \wedge (\forall x.\, \neg Q(x)) && \text{(Skolemise)} \end{aligned}$$

For prenex formulas there might be $\forall$s outside of $\exists$s. Remember that we cannot just swap quantifiers: $\forall x.\, \exists y.\, P(x, y)$ is not the same as $\exists y.\, \forall x.\, P(x, y)$.

$$\begin{aligned} &\forall x.\, \exists y.\, \exists z.\, (\neg P(y) \vee Q(z)) \wedge P(x) \wedge \neg Q(x) \\ \simeq\ &\forall x.\, (\neg P(f(x)) \vee Q(g(x))) \wedge P(x) \wedge \neg Q(x) && \text{(Skolemise)} \end{aligned}$$

Eliminate universal quantifiers

After Skolemisation, the formula will either be in prenex conjunctive normal form (all $\forall$s at the front, matrix in CNF), or miniscope form with only universal quantifiers. In the first case (again, theoretically simple but not practical) all we need to do is drop the quantifiers and convert the matrix into clausal form, with the clauses possibly containing free variables. If the Skolemised formula is in miniscope form, we can still drop all quantifiers, no matter how deeply nested they are in the formula. After this, the quantifier-free formula with variables can be easily converted to CNF by distributing disjunctions over conjunctions, and then rewritten in clausal form.

Since we're trying to be very precise about variable scope and renaming, this reckless treatment of universal quantifiers should seem suspicious. Recall that the free variables in a clause are implicitly universally quantified over, in each clause independently. If the formula is in prenex CN form, the matrix is a conjunction of clauses: $\forall x.\, \forall y.\, C_1 \wedge C_2 \wedge C_3$. But we know that universal quantification distributes over conjunction, so this is equivalent to $(\forall x, y.\, C_1) \wedge (\forall x, y.\, C_2) \wedge (\forall x, y.\, C_3)$ – that is, separating the clauses and arbitrarily renaming variables in each clause is all fine.

$$\begin{aligned} &\forall x.\, (\neg P(f(x)) \vee Q(g(x))) \wedge P(x) \wedge \neg Q(x) \\ \simeq\ &(\forall x.\, \neg P(f(x)) \vee Q(g(x))) \wedge (\forall x.\, P(x)) \wedge (\forall x.\, \neg Q(x)) && \text{(distribute)} \\ \simeq\ &(\forall x.\, \neg P(f(x)) \vee Q(g(x))) \wedge (\forall y.\, P(y)) \wedge (\forall z.\, \neg Q(z)) && \text{(rename (optional))} \\ \simeq\ &\{\neg P(f(x)), Q(g(x))\} \quad \{P(y)\} \quad \{\neg Q(z)\} && \text{(convert to clauses)} \end{aligned}$$

What if the Skolemised formula was in miniscope form? We can convert it to prenex CN form, and drop the quantifiers at the front, as before – or, we can take a shortcut by dropping quantifiers right away,

and converting the resulting NNF formula (with variables) into CNF. The justification for the shortcut is that (as long as all variable names are unique), converting to prenex form is always possible without changing the overall structure of the formula, so the manipulations performed on the quantifier-free NNF formula are the same as what we would be doing to the matrix after conversion to prenex form. Since after conversion to clausal form repeated variable names in different clauses usually don't cause confusion (we can rename them ad-hoc), we don't even need to worry about α-renaming the Skolemised formula – no variable capture can occur, since we drop all binding constructs at once.

$$\begin{aligned}
&(\neg P(a) \vee Q(b)) \wedge (\forall x.\, P(x)) \wedge (\forall x.\, \neg Q(x)) \\
\simeq\ &(\neg P(a) \vee Q(b)) \wedge P(x) \wedge \neg Q(x) && \text{(drop quantifiers (already in CNF))} \\
\simeq\ &\{\neg P(a), Q(b)\} \quad \{P(x)\} \quad \{\neg Q(x)\} && \text{(convert to clauses)}
\end{aligned}$$

Whichever approach we take, we should end up with a set of first-order clauses containing variables but no quantifiers. All variables are implicitly universally quantified per clause, so repeated variable names between clauses can always be distinguished via α-renaming: $\{P(x)\}\ \{\neg Q(x)\}$ is equivalent to $\{P(x)\}, \{\neg Q(y)\}$.

The Skolem–Gödel–Herbrand Theorem

After Skolemisation and converting to clauses, we may end up with a set of ground clauses immediately which can be refuted via ground resolution. In general, however, the clauses will contain variables which are implicitly bound by a universal quantifier over a single clause. How can we adapt ground resolution to this setting?

The key point to note is that every first-order clause stands for a possibly infinite number of ground clauses. For example, consider the clause $C = \{P(f(x), a), Q(g(y, b))\}$ – rewritten in FOL syntax, this is $\varphi = \forall xy.\, P(f(x), a) \vee Q(g(y, b))$. Since universal quantification generalises conjunction, we can extract any number of *instances* from this clause using the equivalence $\forall x.\, P(x) \simeq (\forall x.\, P(x)) \wedge P(a)$, such as:

$$\{P(f(a), a), Q(g(f(x), b))\} \qquad \{P(f(x), a), Q(g(g(a, y), b))\}$$
$$\{P(f(g(a, b)), a), Q(g(f(a), b))\}$$

The last one of these is special is that it is a *ground instance* containing no variables. We can generate an arbitrary number of ground instances by replacing variables with elements of the Herbrand universe (set of ground terms) of the clause. Thus, the task of resolving first-order clauses can be reduced to resolving ground instances of those clauses using ground resolution.

However, it's not obvious that getting a contradiction from ground instances of the first-order clauses will also imply a contradiction for the original universally quantified formula – after all, we would be making a conclusion about an infinite number of clauses using a finite process. Fortunately, this is exactly what the Skolem–Gödel–Herbrand theorem states: a set S of clauses is unsatisfiable iff there is a finite unsatisfiable set S' of ground instances of the clauses of S. For example, to show that $(\forall x.\, P(x)) \wedge \neg P(f(a))$ is unsatisfiable, it is enough to show that it's unsatisfiable with a particular

instantiation of x:

$$(\forall x.\, P(x)) \wedge \neg P(f(a)) \simeq (\forall x.\, P(x)) \wedge P(f(a)) \wedge \neg P(f(a)) \simeq (\forall x.\, P(x)) \wedge \bot \simeq \bot$$

This result should be intuitively obvious, but as with many of these fundamental results, the formal proof is surprisingly involved. In any case, it gives us a way to establish first-order refutation proofs in an efficient manner, as long as we choose the ground instances wisely: in refuting $\{P(x)\}$ and $\{\neg P(f(a))\}$ the only ground instance we care about is $\{P(f(a))\}$, not $P(f(f(a)))$ or $P(b)$, if we also have the constant b in our language.

Unification

The second to last piece of the puzzle is a methodical, algorithmic way of generating ground instances of a clause: unification. Given two first-order terms, unification determines if there is a common ground instance of the terms, and outputs the corresponding substitution that, when applied to the terms, makes them equal. The fairly straightforward algorithm is discussed in the notes. Unification is performed any time we are two clauses that do not have syntactically identical complementary literals, but there is a pair of first-order atoms which can be unified and resolved. In our running example, we ended with the clause set:

$$① \{\neg P(a), Q(b)\} \quad ② \{P(x)\} \quad ③ \{\neg Q(x)\}$$

Clearly ① and ② are not directly resolvable, since $P(a)$ and $P(x)$ are not the same formula. However, ② has two ground instances: $P(a)$ and $P(b)$, which are implicitly part of the clause set:

$$① \{\neg P(a), Q(b)\} \quad ② \{P(x)\} \quad ③ \{\neg Q(x)\} \quad ④ \{P(a)\} \quad ⑤ \{P(b)\}$$

We can now resolve ① and ④ to get ⑥ $\{Q(b)\}$. We can do the same thing with this clause and ③, or combine the two steps (expansion and choice of appropriate ground instance) via unification: $Q(x)$ and $Q(b)$ can be unified with $[b/x]$, so the clauses ③ and ⑥ can be resolved to give $\square$. "Behind the scenes", the unification mechanism finds the ground instance of ③ that would lead to a contradiction when resolved with another clause, and by the Skolem–Gödel–Herbrand theorem this ground contradiction propagates to contradict the entire first-order formula.

An important thing to remember is that unification affects both argument clauses: the MGU has to be applied to every member of the clause, as we are instantiating every bound occurrence of a universally quantified variable. For example, resolving $\{P(x), Q(a, f(x))\}$ and $\{\neg Q(y, f(b)), R(y)\}$ is possible using the MGU $[b/x, a/y]$: the common instances are $\{P(b), Q(a, f(b))\}$ and $\{\neg Q(a, f(b)), R(a)\}$, because the substitution is applied to *every* occurrence of x and y, not just in the arguments of Q. That is, the clause we get after the resolution is $\{P(a), R(b)\}$, *not* $\{P(x), R(y)\}$.

Factoring

One last concept that has to be discussed is *factoring*: is not always required, but may also save the day in a tricky proof. The problem is this: resolution is only "productive" in the presence of *unit clauses*, since resolving with a unit clause is the only way to make a clause shorter. If our clause set doesn't have unit clauses, we will never be able to reach the empty clause and finish the proof. In the propositional

setting we can give up right away, but with first-order clauses it may be possible to continue by factoring a clause containing unifiable literals. For example, if we have a clause $\{P(x, b), P(a, y)\}$, one of its ground instances must be $\{P(a, b), P(a, b)\}$ – but disjunction is idempotent, so this is equivalent to $\{P(a, b)\}$. That is, it may be possible to unify a clause with "itself" to get a shorter instance that allows us to continue the resolution proof.

Factoring is of course not always possible: if all literals in every clause are distinct, we won't be able to unify anything. However, factoring also results in logically weaker clauses, and factoring everything in sight may not be helpful either. The main difficulty of first-order resolution is that a proof may require several attempts until the right combination of resolution steps, unification and factoring is found; however, since first-order resolution is complete, valid formulas are guaranteed to have a resolution proof.