Some Useful Proofs

The slides Appendix1 (A1) contain various proofs about resolution. The theorems in A1b and A1c are important as they give the basis for the soundness of the resolution principle. The Skolemisation theorem on A1ci means that it is sound to consider the clausal form representation of a problem, rather than the general first order representation when using refutation as a proof technique to show (un)satisfiability. (This was called (***) on Optional Slide 4di.) The theorem on A1bi means that when proving theorems about resolution id is allowed to restrict them to Herbrand interpretations and models as opposed to arbitrary models and interpretations, which is usually much easier. (This was called Useful theorem (*) on Slide 4bii.) There is also a proof of the property Subfree, mentioned in Optional part of Slides 6.

Some of the information on unifiers should be familiar to you from Prolog. But notice that Prolog does not test for the occurs check condition, the check, for equation x\equiv y, that x is not in y. This is done for efficiency, but it can lead to unsoundness (of Prolog). The traditional counterexample to this unsoundness is succeeding to show that \(\forall x\exists y P(x,y) \equiv \exists y \forall x P(x,y)\) (which is incorrect). The (Skolemised) clausal form of the Data+negated conclusion (i.e. of \(\forall x\exists y P(x,y)\) and \(\forall y \exists x \neg P(x,y)\)) comprises the two clauses \(P(x,f(x))\) and \(\neg P(g(y),y)\). (Remember that each \(\exists\) quantifier must give rise to different Skolem functions.) These two literals do not unify as the occurs check fails. The unification algorithm first gives \(x=g(y)\) and \(f(x)=y\), and then \(x=g(g(y))\) and \(f(g(y))=y\), but the latter fails the occurs check. However, if you try the Prolog query \(P(g(y),y)\), with the data \(P(x,f(x))\) it succeeds. If you try to write the answer out - well, try it!

## Soundness of a single Resolution step

**A1bi**

Recall from Slides 4 that the Soundness proof of resolution requires only to consider Herbrand models and to show that clauses \(S = \mu R(C1,C2)\), where \(C1\) and \(C2\) are in \(S\) and \(R(C1,C2)\) is their resolvent. i.e. if \(M\) is an H-model of \(\Sigma\) then \(M\) is an H-model of \(S\cup R(C1,C2)\). (Note that \(R(C1,C2)\) does not introduce any terms already occurring in \(\Sigma\).

**Theorem (Single step Soundness):** Let \(C1 = \forall [GvH]\), \(C2 = \forall [\neg EvF]\), \(R = \forall [(HvF)\theta]\) and \(G\theta = E0\) and mgu\((G,E)\) = \(\theta\). (Here, \(G\) and \(E\) are atoms, \(F\) and \(H\) are clauses and the \(\forall\) indicates universal quantification over variables in the clause.) Then,

if \(M\) is a H-model of \(\forall [GvH]\) and \(\forall [\neg EvF]\), then \(M\) is a H-model of \(\forall [(HvF)\theta]\).

**Proof:**

- Variables in \(C1\) and \(C2\) can be renamed so that \(C1\) and \(C2\) are "standardised apart" (i.e. have no variables in common).
- The implicit universal quantifiers can be drawn out into a prefix to yield \(\forall [C1 \wedge C2] \equiv \forall [C1\theta \wedge C2\theta]\)

\[= \forall [(GvH) \theta \wedge (EvF)\theta] = \forall [(\neg H \rightarrow G) \theta \wedge (E \rightarrow F)\theta] = \forall [(\neg H \rightarrow G) \wedge (E \rightarrow F)\theta] = \forall [(HvF)\theta]\]

The step (***) is the crucial one. It says that if \(M\) is a H-model of \(\forall [C1 \wedge C2]\) then \(M\) is also a H-model of \(\forall [C1\theta \wedge C2\theta]\). This follows from the fact that if \(\theta\) is the mgu of the step then it only uses terms in \(\text{Sig}(C1,C2)\), though it may use variables too. (DIY! An outline proof is shown on A1bi.)

Note that the contrapositive of Single Step Soundness states that if \(\forall [C1\theta \wedge C2\theta]\) has no H-model then \(\forall [C1 \wedge C2]\) has no H-model.

---

**Note:** Some of the information on unifiers should be familiar to you from Prolog. But notice that Prolog does not test for the occurs check condition, the check, for equation \(x \equiv y\), that \(x\) is not in \(y\). This is done for efficiency, but it can lead to unsoundness (of Prolog). The traditional counterexample to this unsoundness is succeeding to show that \(\forall x \exists y P(x,y) \equiv \exists y \forall x P(x,y)\) (which is incorrect). The (Skolemised) clausal form of the Data+negated conclusion (i.e. of \(\forall x \exists y P(x,y)\) and \(\forall y \exists x \neg P(x,y)\)) comprises the two clauses \(P(x,f(x))\) and \(\neg P(g(y),y)\). (Remember that each \(\exists\) quantifier must give rise to different Skolem functions.) These two literals do not unify as the occurs check fails. The unification algorithm first gives \(x=g(y)\) and \(f(x)=y\), and then \(x=g(g(y))\) and \(f(g(y))=y\), but the latter fails the occurs check. However, if you try the Prolog query \(P(g(y),y)\), with the data \(P(x,f(x))\) it succeeds. If you try to write the answer out - well, try it!
Soundness of a single Resolution step (continued)

The proof idea of (*) is given next.

Let S be a set of clauses using signature Σ. Starting from a model I of S construct HM, a H-model of S as follows. Each atom in the HB is assigned a truth value in HM given by

\[ P(t_1, ..., t_n) = I(P)(I(t_1), ..., I(t_n)). \]

Let C be a clause in C and suppose C is assigned true by I. Then for each substitution of domain variables for values in C, some literal in C is assigned true by I. Without loss of generality, consider one substitution σ for C and suppose that after the substitution I makes literal L in C true. In case literal L is positive in C (and = P(g1, ..., gn)), then by the definition of HM above the assignment in HM makes true all atoms P(t1, ..., tn) such that for all i, I(ti) = gi. These atoms will be ground instances of L. (Similar considerations apply for negative L, or if C is false in I). C is true in HM, since the substitution \( \sigma \) was arbitrary.

Skolemisation Theorem

A Skolemisation part of conversion to clausal form can be implemented by the function Sk1 below. Then we can show (see also below) that if \( \forall V Sk1(E,V) \) has a model iff \( \forall V E \) has a model, for free variables V in E. (*)

\[ \text{Skolem}(A) = \text{Sk1}(A, \emptyset) \]

\[ \text{Sk1}(A,V) = A, \text{ if } A \text{ is a literal} \]

\[ \text{Sk1}(A \wedge B,V) = \text{Sk1}(A,V) \wedge \text{Sk1}(B,V), \text { where "op" is } \wedge / \vee \]

\[ \text{Sk1}(\forall x.A,V) = \forall x.\text{Sk1}(A,V \cup \{x\}) \]

\[ \text{Sk1}(\exists x.A,V) = \exists x.\text{Sk1}(A[x/f(V')],V), \]

where f is a unique function, V ⊇ V', V occur in A

No other cases are necessary as negations are adjacent to atoms.

Required to show: Skolem(E) has a model iff E has a model. Since E is a sentence it has no free variables and the property (*) will yield the result immediately. We prove the property (*) by induction on the structure of E. Assume as Induction Hypothesis that property (*) holds for immediate subterms of E. Next we show the property holds for E.

Case E is a literal:

M is a model of \( \forall V .\text{Sk1}(E,V) \) iff M is a model of \( \forall V .E \) (defn. of Sk1)

Case E is \( \forall x .A \):

M is a model of \( \forall V .\text{Sk1}(A,V) \) iff M is a model of \( \forall V .(\forall x .A) \) (Equiv.)

Case E is \( \exists x .A \):

M is a model of \( \forall V .\text{Sk1}(A,V) \) iff M is a model of \( \forall V .\text{Sk1}(A[x/f(V')],V) \) (defn. of Sk1)

iff M is a model of \( \forall V .A[x/f(V')] \) (Ind. Hyp.) iff M is a model of \( \forall V .A \) (below)

The very last step in the case for \( \exists x .A \) is the one that does the Skolemisation and it is proved next. The notation x/f(V') means x is replaced by f(V'):

Suppose M is a model of \( \forall V .\exists x .A \). To give a model for \( \forall V .A[x/f(V')] \), we need to extend M so it includes an interpretation for f.

For each vector D', of elements from the domain of M, \( \exists x .A[V'/D',x] \) is true (since \( \forall V .\exists x .A \), so interpret f by : f(D') = some x: A[V'/D', x/z] is true.

Then A[V'/D', x/f(D')] is true in M and M is a model of \( \forall V .A[x/f(V')] \)

Suppose now that M is a model of \( \forall V .A[x/f(V')] \).

Then for each vector D' of elements from the domain of M, A[V'/D', x/f(D')] is true. Hence \( \exists x .A[V'/D'] \) is true and so \( \forall V \exists x .A \) is true too.

The details of the other parts are easier and are left as an exercise.
About Subsumption:

Slides 6 discussed how using subsumed clauses leads to redundancy in a proof and informally introduced the Property Subfree:

**Property SubFree**: Let \( S \) be a set of unsatisfiable clauses. Then, there is a refutation \( \text{Ref} \) from \( S \) such that for each clause \( C_k \) at depth \( \geq 0 \) and used in \( \text{Ref} \) to derive a clause at depth \( >k \), \( C_k \) is not subsumed by any different clause derived at depth \( \leq k \).

In other words, no resolvent in the refutation \( R \) is subsumed by a clause in \( S \) or by a previously generated clause.

The proof of Property SubFree uses this fact (illustrated on slides 6biv/bv):

- if \( C \) subsumes \( D \) and a step in a refutation uses \( D \) (resolving with \( K \)) to derive \( R \), then either \( C \) subsumes \( R \), or resolving \( C \) and \( K \) leads to resolvent \( R' \) that subsumes \( R \).

The proof of this fact is not difficult and is left as an exercise.

Here we show that the Property SubFree holds for refutations formed using saturation search. The proof uses the notion of maximum depth of a refutation, which is the stage in the generation of resolvents in a refutation by saturation search at which the empty clause is formed. A resolvent \( R \) is derived in a refutation at depth \( k \) if \( k \) is the stage in the saturation search at which \( R \) is derived. A given clause is derived at depth \( 0 \). The inductive proof constructs a subsumption free refutation in stages from an arbitrary refutation.

---

Continued from A1dii

By the aforementioned fact, either \( C \) subsumes \( R_1 \) or \( C \) resolves with \( D_1 \) to form \( R_1' \) which subsumes \( R_1 \). A new refutation is constructed from \( \text{Ref} \) as follows.

Clauses in \( \text{Ref} \) at depth \( \leq n \) remain the same. Clause \( R_1 \) is replaced by \( C \) if \( C \) subsumes \( R_1 \), otherwise it is replaced by the resolvent \( R_1' \) of \( C \) with \( D_1 \). In both cases the replacement clause subsumes \( R_1 \). In steps at depths \( >n \), a similar replacement is made using the new subsuming clause \( C \), until either depth \( m \) is reached, the empty clause is formed at depth \( <m \), or the new resolvent is unchanged from the resolvent in \( \text{Ref} \). There are a finite number of such replacements as the maximum depth \( m \) cannot be increased by using subsumed clauses, it can only be decreased. In effect, these replacements allow for new subsumptions by \( R_1' \) to be propagated through the remainder of the refutation \( \text{Ref}' \).

After repeating such replacements as necessary for all clauses derived at depth \( n \), the resulting refutation \( \text{Ref}' \) will have maximal depth \( \leq m \) (in case [ ] was reached earlier than at depth \( m \), possibly with some duplicated clauses. Moreover, the last violation of Property SubFree, if any, is now at depth \( <n \), due to the minimality of \( C \) (you should try to show this). Hence by the induction hypothesis a refutation can be found from \( \text{Ref}' \) that does not violate Property SubFree. In applying the hypothesis, some clauses may be made redundant (if they are no longer used), and duplicated clauses are finally removed.

(Exercise: If interested in proofs, you might like to construct a (simple) example of a refutation that violates the Property and then to follow the construction to obtain a refutation that does satisfy it.)