AUTOMATED REASONING

SLIDES 2-6 Proofs and Things (Appendix 1)

Proof of Soundness of Resolution
Proof of Skolemisation Theorem
About Substitutions and Unifiers

KB - AR - 09

Some Useful Proofs A1ai

The slides Appendix1 (A1) contain various proofs about resolution and a little background information on unifiers. 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 4di.) The theorem on A1bi means that when proving theorems about resolution it is allowed to restrict them to Herbrand interpretations and models as opposed to arbitrary models and interpretations. This is usually much easier. (This was called Useful theorem (*) on 4bii.) There is also a proof of the property Subfree introduced in Slides 6.

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 xi=ti, that xi is not in ti. This is done for efficiency, but it can lead to unsoundness (of Prolog). The traditional counterexample to this unsoundness is succeeding to show that $\forall x \exists y P(x,y) \models \exists y \forall x P(x,y)$ (which is *incorrect*). The (Skolemised) clausal form of the Data+negated conclusion (i.e $\forall x \exists y P(x,y)$ and $\forall y \exists x \neg P(x,y)$) is 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(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 - well, try it!

Soundness of Resolution (a single step) A1bi

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

That this suffices to show Soundness relies on the following *Useful Theorem* (*) (If interested, you can find a proof in the Chapter 1 of notes on my website.)

Useful Theorem (*)
Corresponding to any model of S there is a Herbrand model of S.
or equivalently, If S has no Hmodels then S has no models.
So when showing S has no models, it is sufficient to show S has no H-models.

(Note also: If S has no models it clearly has no Hmodels, so with the above theorem we have the property that S has no models iff S has no Hmodels.)

To show that S=>*[] implies that S has no models (Soundness) uses induction on the length of the refutation of S.
Base Case: k=0. S must contain the empty clause and is clearly unsatisfiable.
Case k>0. Assume as (IH) that the property holds for refutations of length k-1.
Such a refutation has the form (for some C1 and C2 in S) S=>S+R(C1,C2)=>*[].
By (IH) S+R(C1,C2) has no models ==> S+R(C1,C2) has no H-models
==> S has no H-models (by A1bii) ==> S has no models (by (*)).

Next we show that the resolvent between two clauses is logicallly implied by those clauses.

Theorem: Let $C1 = \forall[G \vee H]$, $C2 = \forall[\neg E \vee F]$, $R = \forall[(H \vee F)\theta]$ and $G\theta = E\theta$ 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 $G \vee H$ and $\neg E \vee F$, then M is a H-model of $(H \vee F)\theta$ (universal quantification is assumed implicit).

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

$\forall[C1 \wedge C2] \models \forall[C1\theta \wedge C2\theta]$ (*) $\equiv \forall[(G \vee H)\theta \wedge (\neg E \vee F)\theta]$
$\equiv \forall[(\neg H \rightarrow G)\theta \wedge (E \rightarrow F)\theta] \equiv \forall[(\neg H\theta \rightarrow G\theta) \wedge (E\theta \rightarrow F\theta)]$
$\models \forall[(\neg H\theta \rightarrow F\theta)] \equiv \forall[(H \vee F)\theta]$

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 easily from the fact that if θ is the mgu of the step then it only uses terms from Sig(C1,C2). (DIY!).

It is not difficult to extend the proofs to include factoring.
ie $S \models_H F$, where C is in S and F is a factor of C, and
if S=>*[ ] by derivations using resolution and factoring then S has no models. A1bii

Skolemisation Theorem

A1ci

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

Skolem(A) = Sk1(A,∅)
Sk1(A,V)= A, if A is a literal
Sk1(A op B,V) = Sk1(A,V) op Sk1(B,V), where "op" is ∧ / ∨
Sk1(∀x.A, V) = ∀x.Sk1(A,V ∪{x})
Sk1(∃x.A,V) = ∃x.Sk1(A[x/f(V')],V),
where f is a unique function,V⊇V', V' occur in A
Other cases are unnecessary as negations are adjacent to atoms.

Want to show: Skolem(E) has a model iff E has a model.
Since E has no free variables, the property (*) will yield the result immediately.
We prove the property (*) by induction on the structure of E.

Case E is a literal:
M is a model of ∀V .Sk1(E,V) iff M is a model of ∀V.E (defn. of Sk1)

Case E is A op B:
M is a model of ∀V .Sk1(A op B,V)
iff M is a model of ∀V [Sk1(A,V) op Sk1(B,V)] (defn. of Sk1)
iff M is a model of ∀V [Sk1(A,V)] 'op' M is a model of ∀V [Sk1(B,V)
iff M is a model of ∀V A 'op' M is a model of ∀V B (Ind. Hyp.)
iff M is a model of ∀V [A op B]

Case E is ∀x .A:
M is a model of ∀V.Sk1(∀x.A,V) iff M is a model of ∀V,x.Sk1(A,V∪{x}) (defn. Sk1)
iff M is a model of ∀V,x.A (Ind. Hyp.) iff M is a model of ∀V.(∀x. A) (Equiv.)

Case E is ∃x. A:
M is a model of ∀V.Sk1(∃x.A,V) iff M is a model of ∀V.Sk1(A[x/(f(V')],V) (defn. Sk1)
iff M is a model of ∀V.A[x/f(V')] (Ind. Hyp.) iff M is a model of ∀V.∃x.A (below)

The very last step 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 ∀V.∃x .A. To give a model for ∀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, ∃x .A[V'/D',x] is true (since ∀V.∃x .A), so interpret f by : f(D') = some z: A[V'/D', x/z] is true.
Then A[V'/D', x/f[D')] is true in M and M is a model of ∀V. A[x/f(V')]

Suppose now that M is a model of ∀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 ∃x .A [V'/D'] is true and so ∀V∃x .A is true too.

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

A1cii

Miscellaneous Properties of Unifiers

A1di

A ***substitution*** λ in a language L is a set of equations {xi == ti} such that each xi is unique, xi ≠ ti and xi does not occur in ti. (xi == ti is sometimes written as xi/ti (xi is replaced by ti), or ti/xi (ti replaces xi).

A substitution λ can be applied to P, where P may be a clause, literal or term; the application is written as Pλ and means that the substitutions indicated by λ are made to variables in P.

Usually λ will be *idempotent* (λ is fully evaluated); i.e. no xi occurs in any tj. Then (Xλ)λ = Xλ for any X.

If Pλ = Qλ and P and Q are both literals or both terms, then λ is a *unifier* of P and Q. Pλ is called a *ground instance* of P if it has no variables.

The unification algorithm for X,Y produces a *most general unifier* (mgu) of X,Y.
A mgu θ of X and Y is a unifier of X and Y, such that, for any other unifier λ of X and Y, ∃σ (Xθ) σ = X λ = Y λ. i.e. you can find σ to apply to Xθ that yields Xλ.

A1dii

Substitutions σ and θ can be *composed:* X(σ λ) is defined as (Xσ) λ.
If σ = {xi == ti} and λ = {yi == si}, then σλ = {xi == tiλ, yi == si}, where xi ≠ tiλ, xi does not occur in tiλ, and yi ≠ any xj. i.e. only those yi ≠ any xj are retained.)

e.g. θ = {x == f(y), z ==f (y)} unifies P(z,z) and P(x,f(y))
λ1 = {z == f(y), x == z} does not unify P(z,z) and P(x,f(y)) and is not idempotent;
another unifer is λ = {x==f(a), z==f(a), y==a} and λ = θ {y == a}

To *combine* two substitutions λ and σ just apply the unification algorithm to the unifiers λ and σ treated as equations.

e.g. σ ={x==f(y)} and λ ={x==f(a)} combine to give {x ==f(a), y==a}
but σλ = {x==f(y)} and λσ = {x==f(a)}.

Combination is symmetric: combine(λσ) = combine(σλ).
Note that combination and composition are not always the same:

e.g. if σ = {y==a} and λ = {x == f(y), z == f(y)}
combine(λσ) = combine(σλ) = {x ==f (a), z == f(a), y ==a }
λσ = {x == f(a), z == f(a), y == a} , but σλ = {x == f(y), y == a, z == f(y)}

but they are often the same: for instance,
when vars(λ) ∩ vars(σ) =∅ and no variable in vars(σ) occurs in any RHS of λ
(vars(σ) denotes the vars on LHS σ), then combine(σλ) =σλ .

About Subsumption: A1ei

Slides 6 discussed how using subsumed clauses leads to redundancy in a proof and introduced the Property Subfree (repeated below). 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.

Throughout this section assume that any factoring is combined with the resolution step that uses the factor. i.e. if $L1 \vee L2 \vee C$ is resolved with $\neg L3 \vee D$, where L1,L2,L3 unify with mgu θ and C and D are clauses, then the resolvent is $(C \vee D)\theta$, as if first is made a factor step between L1 and L2 and then a resolution step using $\neg L3$. This simplifies the proof. Also assume that by subsumption is always meant θsubsumption.

Property SubFree:
Let S be a set of unsatisfiable clauses such that none subsumes any other in S. Then, there is a refutation R from S such that for each clause Ck at depth $k \geq 0$ and used in R, Ck is not subsumed by any different clause that is in S or derived from S at depth $\leq k$.

The proof of Property SubFree uses this fact: 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.

Proof of Property SubFree: Let S be a set of unsatisfiable clauses such that all subsumed clauses have been removed and let R be some refutation using clauses in S with $m \geq 1$ steps. If R already possesses Property SubFree there is nothing to prove. Otherwise, let the first violation of Property SubFree occur in R at step $n \geq 1$. The proof uses induction on m-n.

Case m-n=0. The clause at step n (=m) is the empty clause. It is formed by resolving two facts D1 and D2. If a clause C subsumes D1 then C will resolve with D2 also to form the empty clause. Similarly if C subsumes D2. Hence D1 (D2) can be removed from its use in step m of the refutation. R will then possess Property SubFree as there are no more violations in R.

Induction step. (m-n>0). Let R1 at step n<m be derived from clauses D1 and D2 such that a clause C subsumes D1, where D1, D2 and C are all derived (or given) before step n. The Induction Hypothesis (IH) states that for any refutation of clauses from S of length m1 and such that the first violation of Property SubFree occurs at step k, where m1-k<m-n, a corresponding refutation satisfying Property SubFree can be found.

By the aforementioned fact either C subsumes R1 or C resolves with D1 to form R1' which subsumes R1. A new refutation of m steps or less is constructed from R as follows. Clauses in R at steps <n remain the same. Clause R1 is replaced by C if C subsumes R1, otherwise it is replaced by the resolvent R1' of C with D1. In both cases the replacement clause subsumes R1. After repeating such replacements for all clauses derived in step n, the resulting refutation R' has the same number of steps as R, albeit with some possible duplication of clauses. Moreover, the first violation of Property S is at step >n. Hence by the induction hypothesis a refutation can be found from R' that does not violate Property SubFree. In effect, the application of the induction hypothesis allows for new subsumptions by R1' to be propagated through the remainder of the refutation R'. In applying the hypothesis, some clauses may be made redundant (if they are no longer used), and duplicated clauses are removed.

You are encouraged to try to construct an example of a refutation that violates the Property and then to follow the construction to obtain a refutation that does satisfy it. A1eii