Seminar Report

Decidable Problems

Anna-Lena Rädler

Supervisor: Aart Middeldorp

3 February 2017

Abstract

Contents

1 Introduction

In the book Handbook of Practical Logic and Automated Reasoning [Har09] many different algorithms have been considered for verifying that a first-order formula is actually logically valid. But they don't always show when a formulas is invalid. Most algorithms keep on searching and will therefore never terminate.
Fortunately, there are certain procedures that work for special cases or certain classes of formulas.

Over the next pages the following algorithms and special cases for deciding the validity of a formula will be discussed. First the general information for what is needed to successfully implement a decision procedure, then the AE fragment and miniscoping where the **aedecide** procedure and the **miniscoping** procedure are the main parts. The latter procedure only works on a special class of formulas. The next topic are monadic formulas which are a result from miniscoping and prenexing. Their main procedure is the **wang** procedure. Monadic formulas also have an example which is quite old already namely the Aristotelian Syllogisms. The last topic will be the finite model property, where it is stated that the validity of all formulas of a certain class are decidable using the finite model property.

Unless not specifically cited, all sources are taken from John Harrisons book[Har09], to prevent citation redundancy.

2 The Decision Problem

There are different algorithms like Resolution or Tableaux that are used to verify if a first order formula is logically valid. But these algorithms usually don't tell us if a formula is invalid. So to get an algorithmic solution for verifying if a formula is logically not valid the following three naturally and closely connected problems have to be solved.

(1) Confirm that a logically valid formula is indeed valid (satisfiable), and never confirm an invalid (unsatisfiable) one.

(2) Confirm that a logically valid formula is indeed invalid (unsatisfiable), and never confirm a valid (satisfiable) one.

(3) Test whether a forumla is valid (satisfiable) or invalid (unsatisfiable).

Problem (1) can be solved using different procedures, like the previously mentioned Resolution and Tableaux but the problem with those is that they don't test whether a formula is valid or invalid, since they don't terminate every time. If one would try to prove an invalid formula the algorithm would keep on searching for ever.
But solutions for the same problem can be found for limited or modified forms of the same problem which is the goal of this paper.

3 The AE Fragment

For this chapter the Herbrand's theorem is needed. It states the following:

> The Skolemized, quantifier free form of a formula is unsatisfiable if and only if some finite conjunction of ground instances is propositionally unsatisfiable.

The only problem with this result is that the set of possible ground instances is usually infinite but there exists a special case when the Skolemized form only contains constants. If this is the case, then the number of ground instances is bounded. An example of this would be the Łoś Formula.

```
let los =
 <<(forall x y z. P(x,y) /\ P(y,z) ==> P(x,z)) /\
 (forall x y z. Q(x,y) /\ Q(y,z) ==> Q(x,z)) /\
 (forall x y. P(x,y) ==> P(y,x)) /\
 (forall x y. P(x,y) \/ Q(x,y))
 ==> (forall x y. P(x,y)) \/ (forall x y. Q(x,y))>>;;
```

Listing 1: Łoś Formula in `OCaml`.

```
<<(((~P(x,y) \/ ~P(y,z)) \/ P(x,z)) /\
   ((~Q(x,y) \/ ~Q(y,z)) \/ Q(x,z)) /\
   (~P(x,y) \/ P(y,x)) /\ (P(x,y) \/ Q(x,y))) /\
   ~P(c_x,c_y) /\ ~Q(c_x',c_y')>>;;
```

Listing 2: Skolemized Łoś formula.

When Skolemizing the negation of the Łoś, formula the resulting formula contains four constant symbols and three variables. Each of the three variable can be replaced by one of the four constants resulting in 64 ground instances. The Skolemized formula can be proved using the **davisputnam** procedure which finishes quickly after trying 45 of these possibilities.

The importance of a decision procedure that solves this problem is placing a bound on the number of ground instances. Instead of using the **davisputnam** procedure one could have just conjoined all ground instances and tested for propositional satisfiability once and for all. A procedure doing this is called **aedecide**.

```
let aedecide fm =
  let sgm = skolemize(Not fm) in
  let fvs = fv sfm
  and cnsts, funcs = partition (fun (_, ar) -> ar = 0)
  (functions sfm) in
  if funcs <> [] then failwith "Not␣decidable" else
  let consts = if cnsts = [] then ["x",0] else cnsts in
  let cntms = map (fun (c,_) -> Fn(c,[])) consts in
  let alltuples = groundtuples cntms [] 0 (length fvs) in
  let cjs = simcnf sfm in
  let grounds = map
        (fun tup -> image (subst (fpf fvs tup)))
        cjs) alltuples in
  not(dpll(unions grounds));;
```

Listing 3: The aedecide procedure.

This procedure always returns an answer, which means it shows, in any case, whether the input formula is valid or not. Termination could also be ensured quite easily for many generally theorem proving procedures. For example, the inner loop of the **davisputnam** procedure could be modified in such a way, that it returns true if the formula is valid and false when the set of ground instances is exhausted.

To make sure that the Skolemized form of the input formula has no nullary function symbols the following conditions have to be fulfilled:

- The input formula itself has no non nullary function symbols.

Skolemizing does not remove any function symbols. In fact Skolemizing only adds function symbols.

- There is no subformula of the form $\exists y.P[x, y]$, where x is a free or universally quantified variable.

This will always result in a Skolem function with at least x as an argument. For a sentence a sufficient condition is to make sure that all existential quantifiers occur before the universal ones, since then the problem with the subformula cannot happen.

- If all existential quantifiers occur before the universal ones the formula is said to be in the EA subset.

Because of quantifier and connective nesting this is easier to see when the input formula is transformed into prenex normal form. But this only considers the negation since nothing but testing for validity is considered.

- The input formula has to be of the opposite form, which means that it is in the AE subset or simply AE.

For all this the input formula is assumed to be in negation normal form, because this makes it a lot easier. By combining what in the past has been considered, one can as of now see that validity for AE formulas is decidable which automatically means that satisfiability for EA formulas is decidable.

To make this work the implementation must allow to Skolemize directly. This means that if prenex normal form transformations are performed first some tricks are needed in the order of the transformations. So if the negation normal form of the original formula is of the form $(\forall x.P(x)) \vee (\exists y.Q(y))$ the universal quantifier has to be pulled out first and the existential one follows:

$$(\forall x.P(x)) \vee (\exists y.Q(y)) \quad \rightarrow \quad \forall x.P(x) \vee \exists y.Q(y) \quad \rightarrow \quad \forall x.\exists y.P(x) \vee Q(y)$$

Luckily in the implementation of the **pullquants** procedure the subcases are ordered with the universal quantifier matches first, so this effect is already present.
But here one has to pay attention. This procedure has to be applied to the formula before it is negated because otherwise the exact opposite will happen.

$$(\forall x.P(x)) \vee (\exists y.Q(y)) \quad \rightarrow \quad \exists y.(\forall x.P(x)) \vee Qy) \quad \rightarrow \quad \exists y.\forall x.P(x) \vee Q(y)$$

```
let rec pullquants fm =
        match fm with
        And(Forall(x,p),Forall(y,q)) -> pullq(true,true) fm
                mk_forall mk_and x y p q
        | Or(Exists(x,p),Exists(y,q)) -> pullq(true,true) fm
                mk_exists mk_or x y p q
        | And(Forall(x,p),q) -> pullq(true,false) fm mk_forall
                mk_and x x p q
        | And(p,Forall(y,q)) -> pullq(false,true) fm mk_forall
                mk_and y y p q
        | Or(Forall(x,p),q) -> pullq(true,false) fm mk_forall
                mk_or x x p q
        | Or(p,Forall(y,q)) -> pullq(false,true) fm mk_forall
                mk_or y y p q
        | And(Exists(x,p),q) -> pullq(true,false) fm mk_exists
                mk_and x x p q
        | And(p,Exists(y,q)) -> pullq(false,true) fm mk_exists
                mk_and y y p q
        | Or(Exists(x,p),q) -> pullq(true,false) fm mk_exists
                mk_or x x p q
        | Or(p,Exists(y,q)) -> pullq(false,true) fm mk_exists
                mk_or y y p q
        | _ -> fm
and pullq(l,r) fm quant op x y p q =
        let z = variant x (fv fm) in
        let p' = if l then subst (x |=> Var z) p else p
        and q' = if r then subst (y |=> Var z) q else q in
        quant z (pullquants(op p' q'));;
```

Listing 4: The pullquants procedure.

4 Miniscoping

When Skolemizing first the problem of introducing quantifier nesting of an undesirable kind is usually avoided, but if the wrong kind of quantifier nesting is present from the beginning then the use of the previously mentioned **aedecide** procedure will be eliminated.
When applying prenex normal form transformation in reverse order one can massage really misshapen formulas into AE form. This means that the quantifiers have to be pushed into literal level instead of being pulled out. If a formula modified by doing this is the prenexed, the order of the quantifiers will be reversed.

$$\exists y.\forall x.P(y) \implies P(x) \tag{1}$$
$$\rightarrow \exists y.\forall x.\neg P(y) \vee P(x) \tag{2}$$
$$\rightarrow \exists y.\neg P(y) \vee (\forall x.P(x)) \tag{3}$$
$$\rightarrow (\exists y.\neg P(y)) \vee (\forall x.P(x)) \tag{4}$$
$$\rightarrow \neg(\forall y.P(y)) \vee (\forall x.P(x)) \tag{5}$$

```
let rec pushquant x p =
  if not (mem x (fv p)) then p else
  let djs = purednf(nnf p) in
  list_disj (map (separate x) djs);;
```

Listing 5: The pushquant procedure.

The number of formulas reducible to AE form is hard to find, since every valid formula has an AE equivalent, and so does every unsatisfiable one. To transform formulas into AE form the **miniscoping** procedure is introduced.

```
let rec miniscope fm =
  match fm with
        Not p -> Not(miniscope p)
        | And(p,q) -> And(miniscope p, miniscope q)
        | Or(p,q) -> Or(miniscope p, miniscope q)
        | Forall(x,p) -> Not(pushquant x (Not(miniscope p)))
        | Exists(x,p) -> pushquant x (miniscope p)
        | _ -> fm;;
```

Listing 6: The miniscope procedure.

As one can see the **miniscoping** procedure is a straight forward recursion. But there are two notable lines in this short procedure. Both lines considering the quantifiers are very interesting. They use the **pushquants** procedure which splits the scope of the quantifier into two ground, the one with the free variables and the one with the bound variables. By doing this the scope of the quantifier is reduced since only the free variables are in the scope. So these two line are actively responsible for reducing the scope of the quantifier which gives the **miniscoping** procedure its name.

5 Monadic Formulas

Monadic formulas are a class of formulas that will result in an AE formula after **miniscoping** followed by prenexing. They can have arbitrary quantifier nesting, but do not involve any function symbols and only use unary or monadic predicates, which means that they only have one argument, for example:

$$\exists x.P(x) \wedge Q(x)$$

If a monadic formula is miniscoped, the result will have the following property: the body of each quantifier $\forall x.\cdots$ or $\exists x.\cdots$ has (i) no other quantifiers, and (ii) no free variables other than x.
By incorporating **miniscoping** we extend the scope of the **aedecide** function to a broader class of problems which includes at least all monadic formulas. This procedure is called **wang** procedure, to honour Hao Wang who first implemented a theorem prover for this subset in 1960.
In principle the **wang** procedure will solve all monadic formulas, like the Pelletier Problem 20. This formula is trivial for the **wang** procedure.
But in practice, the simple miniscoping transformations used in the procedure can cause an explosion in the size of the formula, because if the formula has alternating quantifiers

```
let wang fm = aedecide(miniscope(nnf(simplify fm)));;
```

Listing 7: The wang procedure.

```
<<(forall x y. exists z. forall w. (P(x) /\ Q(y)
==> R(z) /\ U(w))
==>(exists x y. P(x) /\ Q(y)) ==> (exists z. R(z))>>;;
```

Listing 8: The Pelletier Problem 20.

then the formulas body is alternately transformed into DNF and CNF. A particularly bad example for this would be "Andrew's Challenge", which already blows up quite a bit when being transformed into NNF.

```
pnf(nnf(miniscope(nnf
  <<((exists x. forall y. p(x) <=> P(y)) <=>
  ((exists x. Q(x)) <=> (forall y. Q(y))) <=>
  ((exists x. forall y. Q(x) <=> Q(y)) <=>
  ((exists x. P(x)) <=> (forall y. P(y))))>>)));;
```

Listing 9: Andrew's Challenge in `OCaml`.

When this exact code is executed the resulting formulas is in AE, but is has 19 universal quantifiers followed by 10 existential ones. Which means that there are no fewer than 10^{19} ground instances of quite a large body.

5.1 Syllogisms

The book gives a nice example of monadic formulas called Syllogisms. They were already introduced by Aristotle in his Prior Analytics. Aristotelian syllogisms are constructed from three premises, each of them has to be of one of the following forms:

- A - all S are P
- E - no S are P
- I - some S are P
- O - some S are not P

They are also restricted to only include three terms, the subject S, the predicate P and a middle term M which occurs with either S or P. Aristotelian syllogisms are certain implications of the form if A and B then C, where A, B and C are called premises. This can be expressed using first order predicates:

$$\text{A: } \forall x.S(x) \implies P(x)$$
$$\text{E: } \forall x.S(x) \implies \neg P(x)$$
$$\text{I: } \exists x.S(x) \wedge P(x)$$
$$\text{O: } \exists x.S(x) \wedge \neg P(x)$$

An example for a Syllogism would be:

> All Humans are mortals and all Greeks are Human. Therefore all Greeks are mortal.

In this example the humans are the middle term M, the Greeks are the subject S and being mortal therefore must be the predicate P.
The above mentioned forms for the premises, A, E, I, and O, have already been used in Medieval schools as Mnemonics for the different syllogisms. For example the above mentioned example is of the form BARBARA because it only uses the A-form.

	I	II	III	IV
if	MP	PM	MP	PM
and	SM	SM	MS	MS
then	SP	SP	SP	SP

Using this table one has the ability to form 256 different syllogisms. But from those only 15 are valid according to the author. But in contrast according to Aristotelian syllogistic there are 24 valid ones. So why are there different numbers of valid syllogisms?
The valid syllogisms can be split into two groups [Col]. The ones that are valid according to the Aristotelian standpoint and the ones that are valid according to the Boolean standpoint. The 15 valid syllogisms are the one belonging into the latter category. They are always valid. The 9 syllogisms missing in the Boolean standpoint therefore are valid from the Aristotelian standpoint. This means they are only valid when the relevant term, the subject, exists.

For example the Darapti syllogism:

> All mammals have hair and all cats are mammals Therefore some cats have hair.

This is a valid statement. There are cats with and without hair. But if the cats are replaced by unicorns then the whole sentence would be invalid, because the relevant term, unicorns, does not exist.
But no matter from which standpoint the syllogisms are valid, the book's author managed to build a subset of the traditional Aristotelian syllogisms. And since there are only finitely many possible syllogisms, Aristotle's logic is decidable, that in fact means that the created subset is also decidable.

6 Finite Model Property

For the finite model property cardinality and the Löwenheim-Skolem theorem are needed. Cardinality means whether a formula p has a model M with domain D can depend only on the size of D. And the Löwenheim-Skolem theorem states that if a first order formula has a model of any finite cardinality, if has a model of any other infinite cardinality.
But more important, formulas can place a strong bound in the size of finite models even when only logic without equality is considered. Generally said, for syntactically restricted classes of formulas if often turns out, that satisfiability, like having a model at all, is equivalent to having a finite model.
A formula is said to have the finite model property for satisfiability precisely when it is satisfiable if and only if it is satisfiable on a finite model (respectively validity). There also is a systematic procedure for deciding the validity of all formulas with the finite model property for validity. Since in the book many procedures have been defined for verifying the validity of a formula that is indeed valid, like resolution as an example, and because of the finite model property there is a systematic procedure that verifies if the formula is not valid. One only has to enumerate larger and larger finite interpretations until one is found that does not hold. To turn those two procedures into a decision procedure they only need to be interleaved, and one of them will terminate successfully and therefore make the decision. In principle the termination for this procedure is guaranteed, but in practice the number of possible interpretations explodes dramatically as the size increases, which makes it not a very practical approach.

6.1 Small Model Property

The small model property is an instance of the finite model property. It is easier to see if a formulas has the small model property since it has to be of some definite size. If this is the case the class of formulas is said to have the small model property. A relatively easy example for one of these classes are monadic formulas. For those the following theorem exists.
If a formula p involves k distinct monadic predicates and none of higher arity and also involves no function symbols, then p has a model if and only if it has a model of size 2^k.
The small model property can be used to form a decision algorithm with a definite bound on its runtime, namely 2^k. But it's not a very practical one. If a monadic formula has to be decided, it just has to be tested in all interpretations of size 2^k.

6.2 Prefix Classes

Formulas can be ordered in prefix classes. Theorems can be used to show definitively that certain formulas have no AE equivalent. This can be done by showing that they do not have the corresponding instances of the finite model property.
Ackermann proved in 1928 that formulas of the form $\forall^n\exists\forall^m$ have the finite model property. Gödel proved the generalization of Ackermanns statement in 1932. He said that all formulas of the form $\forall^n\exists\exists\forall^m$ also have the finite model property. Gödel's set of prefixes

contains all cases when the decision problem can be solved using the finite model property. For any other prefix class the decision problem will not be solvable using the finite model property.
Until now only first order logic without equality has been considered. If equality is added the boundary between the decidable and undecidable prefix classes is slightly different. To get equality for first order logic the **eqaxiom** procedure has to be added. This procedure is purely universal which means that it does not change an AE formula. This means that if a formula was in AE and the **eqaxiom** procedure is added the formula stays in AE, this also means that it is decidable.
In 1932 Gödel asserted that his previously stated prefix class with added equality could also be decided using that same method he introduced for the non-equality case. It turns out that this was one of his rare mistakes, and Goldfarb in 1984 proved that this class is in fact undecidable with equality. This means that the only class that is decidable with and without added equality is the one that Ackermann proved, namely $\forall^n\exists\forall^m$.

7 Summary

To summarize one can see that in chapter "The Decision Problem" (2) that a decision algorithm always returns an answer, instead of either failing or keeping on running forever. In chapter "The AE Fragment" (3) it has been stated that any formula in the AE fragment is decidable. Here one can also see that there exists a special case of Skolemized formulas that has a bounded number of ground instances. In chapter "Miniscoping" (4) one can see that the scope of the quantifiers can be downsized by simly using the **miniscoping** procedure. In chapter "Monadic Formulas" (5) it has been shown that monadic formulas are a special class of formulas that can be transformed into the AE fragment. They also have the small model property, which has been shown in chapter "The Finite Model Property" (6). One can also see there that the only prefix class that is decidable with and without equality is $\forall^n\exists\forall^m$.

References

[Col] Categorical syllogsims. `http://rintintin.colorado.edu/~vancecd/phil1440/syllogisms.pdf`, Accessed on November 20, 2016.

[Har09] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, March 2009.