Model Finding Methods

for first-order logic

Jiří Vyskočil

2017

Interpretation and Model

- Let *interpretation* I be such that:
  - non-empty set of elements of D_I
  - for every functional symbol f of arity n we define a total function $f_I : D_I^n \rightarrow D_I$
  - for every predicate symbol p of arity m we define relation $p_I \in D_I^m$
    (or if you like p_I can be seen as a function $f_I : D_I^n \rightarrow$ {true, false})
- *Variable assignment e* for interpretation I is a mapping from variables to elements of D_I.
- For a given interpretation M and variable assignment *e* we can compute all free variables of a given formula φ, then compute (by induction) values of all sub-terms (→ terms), then compute values of all predicates, and finally compute the truth value of the whole formula φ.
- If φ is true then this is denoted **M,e ⊨ φ**

Evaluation of Truth Values

- **Variables:** Each variable x evaluates to $e(x)$ (variable assignment)
- **Functions:** Given terms $t_1,\ldots,t_n$ that have been evaluated to values $v_1,\ldots,v_n$ of the domain D_I, and a n-ary function symbol f, the term $f(t_1,\ldots, t_n)$ evaluates to $f(v_1,\ldots,v_n)$.
- **Atomic formulas:**
  - A formula $p(t_1,\ldots, t_n)$ is associated the value true or false depending on whether $p_I(v_1,\ldots,v_n)$ holds where values $v_1,\ldots,v_n$ are the evaluation of the terms $t_1,\ldots,t_n$.
  - A formula $t_1 = t_2$ is assigned true if t_1 and t_2 evaluate to the same object of D_I.
- **Logical connectives:** A formula in the form $\neg\,\phi$, $\phi\ \&\ \psi$, etc. is evaluated according to the truth table (propositional logic).
- **Existential quantifiers:** A formula $\exists x : \phi(x)$ is true according to I and e if there exists an evaluation e' of the variables that only differs from e regarding the evaluation of x and such that φ is true according to the interpretation I and the variable assignment e'. This formal definition captures the idea that $\exists x : \phi(x)$ is true iff there is a way to choose a value for x such that $\phi(x)$ is satisfied.
- **Universal quantifiers:** A formula $\forall x : \phi(x)$ is true iff the formula $\neg(\exists x : \neg\,\phi(x))$ is true. This captures the idea that $\forall x : \phi(x)$ is true if every possible choice of a value for x causes $\varphi(x)$ to be true.

Interpretation and Model

- If a formula φ evaluates to *true* under a given interpretation M and for all possible variable assignments, one says that:

  M satisfies φ

- this is denoted

  $M \models \varphi$

- A first-order structure that satisfies all formulae in a given theory is said to be a **model of the theory**.
- A formula is **logically valid** (or simply valid) if it is true in every interpretation. (this is similar to tautologies in propositional logic)
- A formula φ is a **logical consequence** of a formula ψ if every interpretation that makes ψ true also makes φ true.
  In this case one says that:

  φ is logically implied by ψ

Interpretation → Model - example

- **Let T be the following theory** (X,Y and Z are variables):

  $e \otimes X = X$ *% left identity*

  $inv(X) \otimes X = e$ *% left inverse*

  $(X \otimes Y) \otimes Z = X \otimes (Y \otimes Z)$ *% associativity*

  $a \otimes b \neq b \otimes a$ *% constants* a *and* b *do not commute*

- **Let *M* be the following interpretation of theory T:**

  $D_M = \{1,2,3,4,5,6\}$

e_M	1

a_M	2

b_M	3

X	$inv_M(X)$
1	1
2	2
3	3
4	5
5	4
6	6

$\otimes_M$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	1	4	3	6	5
3	3	5	1	6	2	4
4	4	6	2	5	1	3
5	5	3	6	1	4	2
6	6	4	5	2	3	1

Predicates = and ≠ are defined in a standard way.

Finding Models

Remark 1:

- Let $I \models \varphi$ where I is defined on set D.
  Let D' be a set and π is a bijection $D \leftrightarrow D'$, then we can construct an interpretation I' on D' such that holds

  $(\forall x_1, \ldots, x_m \in D')\ (p_{I'}(x_1, \ldots, x_m) = p_I(\pi^{-1}(x_1), \ldots, \pi^{-1}(x_m)))$

  $(\forall x_1, \ldots, x_n \in D')\ (f_{I'}(x_1, \ldots, x_n) = \pi(f_I(\pi^{-1}(x_1), \ldots, \pi^{-1}(x_n)))$

  Now, it also holds: $I' \models \varphi$
- I a I' are **isomorphic** in φ.
- **=>** The only important property for choosing D is its cardinality not the concrete content.

DPLL Like Finding Models

```
procedure search;
{
    cell = select_cell();
    top = last_value_to_consider();
    foreach i ∈ {1 ... top} {
        ok = assign_and_propagate(cell,i);
        if (ok) {
            search();
            undo_assignments();
        }
    }
}
```

Translation to SAT

- Another efficient way how to find models in first-order theories is a translation to SAT (Paradox system).

SAT literal encoding:

- For every predicate symbol p and for every its arguments $d_1,\ldots, d_{arity(p)}$ where $d_i \in D$, we introduce a propositional variable $\boldsymbol{p(d_1,\ldots, d_{arita(p)})}$ that represents: $p(d_1,\ldots, d_{arity(p)})$ = true.
- For every function symbol f, for every its arguments $d_1,\ldots, d_{arity(f)}$ where $d_i \in D$, and for every possible resulting value $d \in D$, we introduce a propositional variable $\boldsymbol{f(d_1,\ldots, d_{arita(f)}) = d}$ that represents: f applied on arguments $d_1,\ldots, d_{arity(f)}$ is equal to d.

Clause Flattening

Definition:

- A literal is shallow when it is in one of the following form:
  1. $p(x_1,\ldots, x_m)$ or $\neg p(x_1,\ldots, x_m)$
  2. $f(x_1,\ldots, x_n) = y$ or $f(x_1,\ldots, x_n) \neq y$
  3. $x = y$

Remark:

- If we would be able to translate all clauses and all its literals of an input theory to the previous form (all clauses contain only shallow literals) then we can use the previous SAT literal encoding and then we can use a SAT solver for finding a model of a particular size of set *D*.
- Such translation is called *flattening*.

Translation of Flattened Clauses to SAT

Algorithm:

- For every flattened first-order clause C and for every its possible instantiation defined by substitution σ : FV(C) $\rightarrow D$ we generate a propositional clause $\mathbf{C}\sigma$. The function FV gives us all free variables occurring in the clause. All created equalities of the form $d_1 = d_2$ where $d_1, d_2 \in D$ we can propagate by deletion the equality (if $d_1 \neq d_2$) or by deletion of the whole clause (if $d_1 = d_2$).
- For every function symbol f, for every instantiation of its arguments $d_1,\ldots, d_{arita(f)}$ where $d_i \in D$ and for every d, d' $\in D$ such that d $\neq$ d' we generate a propositional clause

  $$f(d_1,\ldots, d_{arita(f)}) \neq d \quad \vee \quad f(d_1,\ldots, d_{arita(f)}) \neq d'$$

  These clauses guarantee that, every function returns for the same arguments at most one resulting value.
- For every function symbol f and for every instantiation of its arguments $d_1,\ldots, d_{arita(f)}$ where $d_i \in D$ we generate a propositional clause

  $$f(d_1,\ldots, d_{arita(f)}) = 1 \quad \vee \ldots \vee \quad f(d_1,\ldots, d_{arita(f)}) = size\_of(D)$$

  These clauses guarantee that, the function f is total (it returns at least one resulting value for any input).

Clause Flattening

Elimination of non shallow literals:

There are only two cases where some literal is not shallow:

1. It contains at least one sub-term that is not variable.
2. It has the following form: $x \neq y$ where x and y are variables

Clause Flattening

Elimination of non shallow literals:

There are only two cases where some literal is not shallow:

1. It contains at least one sub-term that is not variable.
2. It has the following form: $x \neq y$ where x and y are variables

Case 1:

- Assume that some clause C contains a sub-term t that is not variable. Then we can apply the following transformation on C:

  $C[t] \Rightarrow x \neq t \vee C[x]$

  where x is a new fresh variable that does not occur anywhere in C yet.

- The transformation is logically correct, because C is substituted by $(\forall x)\ (x = t \Rightarrow C[x])$.

- If there are more occurrences of t in C then we substitute all of them by x.

Clause Flattening

Elimination of non shallow literals:

There are only two cases where some literal is not shallow:

1. It contains at least one sub-term that is not variable.
2. It has the following form: $x \neq y$ where x and y are variables

Case 2:

- Assume that some clause E contains a literal of the form $x \neq y$ (E has form $C \vee x \neq y$). Then we can apply the following transformation on E:

  $C[x,y] \quad \vee \quad x \neq y \quad \Rightarrow \quad C[x,x]$

- The correctness of the transformation is obvious.

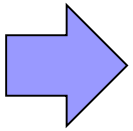
=> Complete Solution :

- By iterative application of the previous cases on an input clauses we will obtain the resulting clauses containing only shallow literals.

Reduction of Variables

- The number of instances that have to be generated for every clause is exponential to the number of different variables inside the clause. (more precisely s^k where k is the number of different variables and s is the size of D).
- Moreover the number can be even increased by addition of new variables during the flattening algorithm.
- One reasonable option, how to reduce the number of different variables inside the clause, is *splitting* of clauses:

Example:

$p(x,y) \vee q(x,z)$  $(p(x,y) \vee r(x)) \,\&\, (\neg r(x) \vee q(x,z))$

- We introduce a new predicate $r(x)$ for clause split.
- The previous transformation can decrease the number of free variables by 1.

Binary Split

Definition:

Given a clause $C[\mathbb{x}] \vee D[\mathbb{y}]$ where C, D are sub-clauses and $\mathbb{x}$, $\mathbb{y}$ are the sets of variables occurring in them.

Then C and D constitute a *proper binary split* of the given clause iff there exist at least one variable x in $\mathbb{x}$ such that $x \notin \mathbb{y}$, and at least one variable y in $\mathbb{y}$ such that $y \notin \mathbb{x}$.

The resulting two clauses after the split are

$$r(\mathbb{x} \cap \mathbb{y}) \vee C[\mathbb{x}]$$

&

$$\neg r(\mathbb{x} \cap \mathbb{y}) \vee D[\mathbb{y}]$$

where r is a new fresh symbol that does not occur anywhere in the problem yet.

Term Definitions

New Constants Definitions:

In cases where literals contain deep ground terms we can avoid introducing auxiliary variables, by introducing fresh constants as names for the terms, and substituting the terms for their names.

Example:

Flattening the clause $p(f(a, b), f(b, a))$ yields the clause:

$$a \neq x \vee b \neq y \vee f(x, y) \neq z \vee f(y, x) \neq w \vee p(z,w)$$

with 4 different variables that cannot be split.

however, if we first introduce new fresh constants $\mathbf{t_1}$ a $\mathbf{t_2}$, we obtain equivalent clauses: $\mathbf{t_1} = f(a, b)$ & $\mathbf{t_2} = f(b, a)$ & $p(\mathbf{t_1}, \mathbf{t_2})$

after flattening we have:

$$a \neq x \vee b \neq y \vee f(x, y) \neq z \vee \mathbf{t_1} = z \quad \&$$

$$a \neq x \vee b \neq y \vee f(y, x) \neq z \vee \mathbf{t_2} = z \quad \&$$

$$\mathbf{t_1} \neq x \vee \mathbf{t_2} \neq y \vee p(x,y)$$

The max. number of different variables in every clause is 3 only.

Static Symmetry Reduction

- Remark 1 tells us, that every model with more than one element has several isomorphic representations. (It can be constructed by permutations on D). This makes the problem of finding models harder for SAT solvers.
- A solution is an introduction of some canonical representation of model that represents all (or at least many) other isomorphic representations of that model.
- If we set one arbitrary value v to some constant c then if there exists some model then we can modify that model by Remark 1 to obtain a new model with the constant c set to the value v.

Constant Symmetries

- If we apply the previous idea iteratively then we can see that every new constant can have the same value as one of the previous constants or can have a completely new value (not the same).
- Consider that we will denote all constants by a_i. Now by addition of the following clauses (constraints) we can reduce the number of isomorphic models generated by SAT solvers:

$a_1=1$ &

$a_2=1 \vee a_2=2$ &

$a_3=1 \vee a_3=2 \vee a_3=3$ &

…

For all $1 < i \in D$ and for all $1 < d \leq i \in D$ we can even add:

$a_i=d \Rightarrow a_1=d-1 \vee a_2=d-1 \vee a_3=d-1 \vee \ldots \vee a_{i-1}=d-1$ that is:

$a_i \neq d \vee a_1=d-1 \vee a_2=d-1 \vee a_3=d-1 \vee \ldots \vee a_{i-1}=d-1$

Static Symmetry Reduction of Functions

- Constants are functions with arity 0.
- However, when there are function symbols of higher arity, this is no longer easy to do statically (at least we need to produce a big number of additional clauses/constraints).
- Nevertheless, we can remove some symmetries. Unary function case:

$$f(1)=1 \vee f(1)=2 \vee \ldots \vee f(1)=k+1 \quad \&$$

$$f(2)=1 \vee f(2)=2 \vee \ldots \vee f(2)=k+1 \vee f(2)=k+2 \quad \&$$

$$f(3)=1 \vee f(3)=2 \vee \ldots \vee f(3)=k+1 \vee f(3)=k+2 \vee f(3)=k+3 \quad \&$$

$$\ldots$$

where k is a number of all constants. If k=0, then we can introduce one arbitrary constant.

- For higher arities we can generalize the previous case by introduction of a new fresh unary function g with the following property $g(x)=f(x,\ldots,x)$.

References

- K. Claessen, N. Sörensson. **New Techniques that Improve MACE-style Finite Model Finding**. Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications; 2003.
- W. McCune. **Mace4 Reference Manual and Guide.** Mathematics and Computer Science Division. Technical Memorandum No. 264; 2003