Using SAT and SMT Solvers for Finite Model Finding with Sorts

Giles Reger   Martin Suda

School of Computer Science, University of Manchester, UK

Automated Reasoning Workshop 2016

Finite Models

- A finite models interpret symbols over a finite set of constants

- We consider a specific kind of model we call a DC-model
- In these models the domain of interpretation is n fresh domain constants
- All terms are interpreted as domain constants, which are interpreted as themselves
- If a FO formula has a model of size n it has a DC-model of size n

- Finite models can be useful in a range of applications
- Additionally, (in our experience) finite model finding can establish sat where other techniques (e.g. resolution) cannot

Finite Model Finding with a SAT solver

- A SAT-based finite-model finding approach was introduced by MACE [Mcc94] and extended by Paradox [CS03]

- The model finding idea for model size n is to introduce n domain constants, ground the problem with these and encode the result as a SAT problem. Then try increasing model sizes.

- Clauses (we assume clausification) must be *flattened* before grounding for the SAT encoding to work, i.e. no nested terms.
- Need to include *functionality* and *totality* in the encoding, e.g.

  $$f(d_1) \neq d_1 \vee f(d_1) \neq d_2 \qquad \text{and} \qquad f(d_1) = d_1 \vee f(d_1) = d_2$$

  i.e. $f(d_1)$ evaluates to at most one term and at least one term
- For efficiency we *break symmetries* by ordering ground terms

First Order Logic with Sorts

- A common extension to FOL is to add *sorts*
- Predicate and function symbols and quantifications then become *sorted*

- There exist translations from the sorted case to the unsorted (adding functions or predicates), but these add a lot of noise (unless sorts are *monotonic*)

- Alternatively, one can introduce a *Sorted Model* where each sort is interpreted over a separate set of constants. Importantly (see example below) different sorts may have different sizes (number of constants).

Organised Monkey Village

Each monkey has at least two bananas.

$$(\forall M : monkey)(\text{owns}(M, \text{b}_1(M)) \wedge \text{owns}(M, \text{b}_2(M)) \wedge \text{b}_1(M) \neq \text{b}_2(M))$$

$$(\forall M_1, M_2 : monkey)(\forall B : banana)$$
$$(\text{owns}(M_1, B) \wedge \text{owns}(M_2, B) \rightarrow M_1 = M_2)$$

$$(\forall T : tree)(\exists M_1, M_2, M_3 : monkey)$$
$$((\textstyle\bigwedge_{i=1}^{3} \text{sits}(M_i) = T) \wedge \text{distinct}(M_1, M_2, M_3))$$

$$(\forall M_1, M_2, M_3, M_4 : monkey)(\forall T : tree)$$
$$((\textstyle\bigwedge_{i=1}^{4} \text{sits}(M_i) = T) \Rightarrow \neg\text{distinct}(M_1, M_2, M_3, M_4))$$

$$(\forall M : monkey)(\text{partner}(M) \neq M \wedge \text{partner}(\text{partner}(M)) = M)$$

There must be at least twice as many bananas as monkeys

Organised Monkey Village

Every tree contains exactly three monkeys.

$$(\forall M : \mathit{monkey})(\mathrm{owns}(M, \mathrm{b}_1(M)) \wedge \mathrm{owns}(M, \mathrm{b}_2(M)) \wedge \mathrm{b}_1(M) \neq \mathrm{b}_2(M))$$

$$(\forall M_1, M_2 : \mathit{monkey})(\forall B : \mathit{banana}) \\ (\mathrm{owns}(M_1, B) \wedge \mathrm{owns}(M_2, B) \rightarrow M_1 = M_2)$$

$$(\forall T : \mathit{tree})(\exists M_1, M_2, M_3 : \mathit{monkey}) \\ ((\textstyle\bigwedge_{i=1}^{3} \mathrm{sits}(M_i) = T) \wedge \mathrm{distinct}(M_1, M_2, M_3))$$

$$(\forall M_1, M_2, M_3, M_4 : \mathit{monkey})(\forall T : \mathit{tree}) \\ ((\textstyle\bigwedge_{i=1}^{4} \mathrm{sits}(M_i) = T) \Rightarrow \neg \mathrm{distinct}(M_1, M_2, M_3, M_4))$$

$$(\forall M : \mathit{monkey})(\mathrm{partner}(M) \neq M \wedge \mathrm{partner}(\mathrm{partner}(M)) = M)$$

There must be exactly three times as many monkeys as trees

Organised Monkey Village

Each monkey has a unique partner.

$$(\forall M : monkey)(\text{owns}(M, \text{b}_1(M)) \wedge \text{owns}(M, \text{b}_2(M)) \wedge \text{b}_1(M) \neq \text{b}_2(M))$$

$$(\forall M_1, M_2 : monkey)(\forall B : banana) \\ (\text{owns}(M_1, B) \wedge \text{owns}(M_2, B) \rightarrow M_1 = M_2)$$

$$(\forall T : tree)(\exists M_1, M_2, M_3 : monkey) \\ ((\bigwedge_{i=1}^{3} \text{sits}(M_i) = T) \wedge \text{distinct}(M_1, M_2, M_3))$$

$$(\forall M_1, M_2, M_3, M_4 : monkey)(\forall T : tree) \\ ((\bigwedge_{i=1}^{4} \text{sits}(M_i) = T) \Rightarrow \neg\text{distinct}(M_1, M_2, M_3, M_4))$$

$$(\forall M : monkey)(\text{partner}(M) \neq M \wedge \text{partner}(\text{partner}(M)) = M)$$

There must be an even number of monkeys

Organised Monkey Village

$$(\forall M : \mathit{monkey})(\mathrm{owns}(M, \mathrm{b}_1(M)) \wedge \mathrm{owns}(M, \mathrm{b}_2(M)) \wedge \mathrm{b}_1(M) \neq \mathrm{b}_2(M))$$

$$(\forall M_1, M_2 : \mathit{monkey})(\forall B : \mathit{banana})$$
$$(\mathrm{owns}(M_1, B) \wedge \mathrm{owns}(M_2, B) \rightarrow M_1 = M_2)$$

$$(\forall T : \mathit{tree})(\exists M_1, M_2, M_3 : \mathit{monkey})$$
$$((\textstyle\bigwedge_{i=1}^{3} \mathrm{sits}(M_i) = T) \wedge \mathrm{distinct}(M_1, M_2, M_3))$$

$$(\forall M_1, M_2, M_3, M_4 : \mathit{monkey})(\forall T : \mathit{tree})$$
$$((\textstyle\bigwedge_{i=1}^{4} \mathrm{sits}(M_i) = T) \Rightarrow \neg\mathrm{distinct}(M_1, M_2, M_3, M_4))$$

$$(\forall M : \mathit{monkey})(\mathrm{partner}(M) \neq M \wedge \mathrm{partner}(\mathrm{partner}(M)) = M)$$

The ‘smallest’ model has 12 bananas, 6 monkeys and 2 trees

Adding Sorts to the Encoding and Search

- We can straightforwardly add sorts to the above encoding by introducing a set of domain constants *per sort* and using the relevant constants in the encoding
- The search must now consider a *domain size assignment* mapping each sort to its domain size
- A naive search could enumerate possible domain size assignments in a breadth-first manner. This will be finite-model-complete but highly inefficient

Using Constraints to Guide Search with an SMT Solver (1)

- Let n be the number of sorts and n_s be the size of sort s in the current assignment
- We extract constraints from failed proofs to guide the search
- We will extract a set of constraints $\mathcal{C}$ and ask a SMT solver to find a model for

$$\mathcal{C} \wedge k = \sum_{s=1}^{n} n_s$$

  starting with $k = n$ and increasing k by 1 whenever no model can be found i.e. we are going breadth-first

- The constraints will (at least) block previously attempted models

Using Constraints to Guide Search with an SMT Solver (2)

- To extract $\mathcal{C}$ we update the encoding with two extra labels:
  - $|s| > n_s$ stands for the size of s being *too small*
  - $|s| < n_s$ stands for the size of s being *too large*
- The encoding is updated accordingly
- totality becomes

$$\text{banana}_1(d_1) = d_1 \vee \text{banana}_1(d_1) = d_2 \vee |banana| > 2$$

- the grounding of a flattened clause would be

$$\text{owns}(d_3, d_1) \vee \text{banana}_1(d_3) \neq d_1 \vee |monkey| < 3$$

Using Constraints to Guide Search with an SMT Solver (3)

- Next we solve the resulting SAT problem under the assumption

$$\bigwedge_{s=1}^{n} \neg(|s| > n_s) \wedge \neg(|s| < n_s),$$

  i.e. that the current assignment is of the *"right size"*

- If no model is found, this technique can return a set $A_0 \subseteq A$ of assumptions *sufficient* for replaying the proof of unsatisfiability

- The clause $\neg A_0$ can be added directly to $\mathcal{C}$
- This rules out any new domain size assignment that could be shown to be unsatisfiable using part of the current proof

Finding a Model for the Monkey Village

- Run Vampire...

Experiments

	CVC4	Paradox	iProver	Vampire
FOF+CNF: sat	1181	1444	1348	**1503**
FOF+CNF: unsat	-	-	1337	**1628**

	CVC4	Vampire
UF: sat	764	**896**
UF: unsat	-	249

References

Koen Claessen, Ann Lillieström, and Nicholas Smallbone.

Sort it out with monotonicity - translating between many-sorted and unsorted first-order logic.

In *CADE-23*, pages 207–221, 2011.

Koen Claessen and Niklas Sörensson.

New techniques that improve MACE-style model finding.

In *CADE-19 Workshop: Model Computation - Principles, Algorithms and Applications*, 2003.

William Mccune.

A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems.

Technical report, Argonne National Laboratory,, 1994.

More Fun with Sorts

More can be done with sorts. For example:

- Merging sorts together to get *fewer sorts*
- Inferring *subsorts* and expanding them to get *more sorts*
- Detecting relationships *between sorts*, e.g. from injectivity
- Detecting *upper bounds* on sorts in order to establish unsatisfiability

All of this can be helped by detecting *monotonic sorts* [CLS11]

- (Roughly speaking) a sort s is monotonic for a formula φ if adding another domain constant to s in a model of φ produces another model for φ