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.

  \[ f(d_1) \neq d_1 \lor f(d_1) \neq d_2 \quad \text{and} \quad f(d_1) = d_1 \lor f(d_1) = d_2 \]

  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.

$$\forall M : monkey)(own(M, b_1(M)) \land own(M, b_2(M)) \land b_1(M) \neq b_2(M))$$

$$(\forall M_1, M_2 : monkey)(\forall B : banana)
(own(M_1, B) \land own(M_2, B) \implies M_1 = M_2)$$

$$(\forall T : tree)(\exists M_1, M_2, M_3 : monkey)
((\land_{i=1}^3 sit(M_i) = T) \land distinct(M_1, M_2, M_3))$$

$$(\forall M_1, M_2, M_3, M_4 : monkey)(\forall T : tree)
((\land_{i=1}^4 sit(M_i) = T) \implies \neg distinct(M_1, M_2, M_3, M_4))$$

$$(\forall M : monkey)(partner(M) \neq M \land partner(partner(M)) = M)$$

There must be at least twice as many bananas as monkeys.
Organised Monkey Village

Every tree contains exactly three monkeys.

\[(\forall M : monkey)(\text{owns}(M, b_1(M)) \land \text{owns}(M, b_2(M)) \land b_1(M) \neq b_2(M))\]

\[(\forall M_1, M_2 : monkey)(\forall B : banana)
\quad (\text{owns}(M_1, B) \land \text{owns}(M_2, B) \rightarrow M_1 = M_2)\]

\[(\forall T : tree)(\exists M_1, M_2, M_3 : monkey)
\quad (((\land_i^3 \text{sits}(M_i) = T) \land \text{distinct}(M_1, M_2, M_3)))\]

\[(\forall M_1, M_2, M_3, M_4 : monkey)(\forall T : tree)
\quad (((\land_i^4 \text{sits}(M_i) = T) \Rightarrow \neg \text{distinct}(M_1, M_2, M_3, M_4)))\]

\[(\forall M : monkey)(\text{partner}(M) \neq M \land \text{partner}(\text{partner}(M)) = M)\]

There must be exactly three times as many monkeys as trees.
Organised Monkey Village

Each monkey has a unique partner.

\[(\forall M : \text{monkey})(\text{owns}(M, b_1(M)) \land \text{owns}(M, b_2(M)) \land b_1(M) \neq b_2(M))\]

\[(\forall M_1, M_2 : \text{monkey})(\forall B : \text{banana})
\quad (\text{owns}(M_1, B) \land \text{owns}(M_2, B) \rightarrow M_1 = M_2)\]

\[(\forall T : \text{tree})(\exists M_1, M_2, M_3 : \text{monkey})
\quad ((\land_3 i=1 \text{sits}(M_i) = T) \land \text{distinct}(M_1, M_2, M_3))\]

\[(\forall M_1, M_2, M_3, M_4 : \text{monkey})(\forall T : \text{tree})
\quad ((\land_4 i=1 \text{sits}(M_i) = T) \Rightarrow \neg \text{distinct}(M_1, M_2, M_3, M_4))\]

\[(\forall M : \text{monkey})(\text{partner}(M) \neq M \land \text{partner}(\text{partner}(M)) = M)\]

There must be an even number of monkeys.
Organised Monkey Village

\[(\forall M : \text{monkey})(\text{owns}(M, b_1(M)) \land \text{owns}(M, b_2(M)) \land b_1(M) \neq b_2(M))\]

\[(\forall M_1, M_2 : \text{monkey})(\forall B : \text{banana})\]
\[\text{owns}(M_1, B) \land \text{owns}(M_2, B) \rightarrow M_1 = M_2\]

\[(\forall T : \text{tree})(\exists M_1, M_2, M_3 : \text{monkey})\]
\[\left((\bigwedge_{i=1}^{3} \text{sits}(M_i) = T) \land \text{distinct}(M_1, M_2, M_3)\right)\]

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

\[(\forall M : \text{monkey})(\text{partner}(M) \neq M \land \text{partner}(\text{partner}(M)) = M)\]

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 \textit{per sort} and using the relevant constants in the encoding.

▶ The search must now consider a \textit{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.
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 $C$ and ask a SMT solver to find a model for

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

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.
To extract $C$ 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:

![Math notation]

The grounding of a flattened clause would be

![Math notation]
Next we solve the resulting SAT problem under the assumption

\[ \bigwedge_{s=1}^{n} \neg(|s| > n_s) \land \neg(|s| < n_s), \]

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 \textit{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

<table>
<thead>
<tr>
<th></th>
<th>CVC4</th>
<th>Paradox</th>
<th>iProver</th>
<th>Vampire</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOF+CNF: sat</td>
<td>1181</td>
<td>1444</td>
<td>1348</td>
<td>1503</td>
</tr>
<tr>
<td>FOF+CNF: unsat</td>
<td>-</td>
<td>-</td>
<td>1337</td>
<td>1628</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CVC4</th>
<th>Vampire</th>
</tr>
</thead>
<tbody>
<tr>
<td>UF: sat</td>
<td>764</td>
<td>896</td>
</tr>
<tr>
<td>UF: unsat</td>
<td>-</td>
<td>249</td>
</tr>
</tbody>
</table>
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.

William Mccune.
A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems.
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 $\varphi$ if adding another domain constant to $s$ in a model of $\varphi$ produces another model for $\varphi$