The Uses of SAT Solvers in Vampire

Giles Reger and Martin Suda

School of Computer Science, University of Manchester

The 2nd Vampire Workshop

Introduction

In this talk we will:

- Talk about the different use of SAT solvers in Vampire
    1. Finite Model Building
    2. AVATAR
    3. Instance Generation
    4. Global Subsumption
- Talk about how they could be better!

Overview

Finite Model Building

- Newly added to Vampire this year
- Just implements existing ideas
- Useful for establishing non-theorems i.e. satisfiability checking

- *Idea:* For a domain size n create a ground problem that is satisfiable if the original problem has a finite model of size n.

- The ground literals can be (consistently) named/translated into SAT variables and the ground problem decided by a SAT solver

- We can just check for bigger and bigger values of n

Preparing the Problem

- **Definition Introduction.** This reduces the size of clauses produced by flattening. A clause $p(f(a,b), g(f(a,b)))$ becomes $p(t_1, t_2)$ and we introduce the definition clauses $t_1 = f(a,b)$ and $t_2 = g(t_1)$

- **Flattening.** This is necessary for the technique in general. A clause $p(f(a,b), g(f(a,b)))$ becomes

$$p(x_1, x_2) \vee x_1 \neq f(x_3, x_4) \vee x_2 \neq g(x_1) \vee x_3 \neq a \vee x_4 \neq b$$

- **Splitting.** This can reduce the number of variables in clauses (important later). The clause $p(x,y) \vee q(y,z)$ is transformed to the two clauses $p(x,y) \vee s(y)$ and $\neg s(y) \vee q(y,z)$.

The Constraints

- **Groundings.** For each (flattened) clause $C[\mathbf{x}]$ and each vector of domain constants $\mathbf{d}$ translate and add $C[\mathbf{d}]$
- **Functionality.** For each function symbol f with arity a, vector of domain constants $\mathbf{d}$ of length a and distinct domain constants d_1 and d_2 translate and add $f(\mathbf{d}) \neq d_1 \vee f(\mathbf{d}) \neq d_2$
- **Totality.** For each function symbol f with arity a and vector of domain constants $\mathbf{d}$ of length a translate and add $f(\mathbf{d}) = d_1 \vee \ldots \vee f(\mathbf{d}) = d_n$ for (all) the domain constants d_i
- Note the exponential nature of these constraint sets

Symmetry Breaking and Sort Inference

- **Symmetry Breaking.**
  - Any model will be symmetrical in ordering of domain constants
  - So the SAT solver will be checking the same model multiple times
  - We can (partly) break these symmetries by ordering ground terms
  - Pick and order n ground terms (include all constants at the front)
  - For term t_i and domain size n add the clauses

$$t_i \neq d_m \vee t_1 = d_{m-1} \vee \ldots \vee t_{i-1} = d_{m-1}$$

for $m \leq n$ and if $i \leq n$ add

$$t_i = d_1 \vee \ldots \vee t_i = d_i$$

- **Sort Inference.**
  - Separate constants and function positions into different distinct sorts
  - Under certain conditions we can detect a maximum size for a sort
  - This information can render certain constraints redundant

Importance of the SAT Solver

- The majority of time is spent inside the SAT solver
- Therefore, making the SAT solver faster can improve this method.
- **Variable Elimination.** As implemented in e.g. MiniSAT. Idea is to apply all resolutions on a variable to eliminate it. Only do this if it will reduce the size. Removes pure variables.
  - Can help a lot
  - Can make things worse

Anything Else?

- Deciding Non-Non-Theorems
  - This is a decision procedure for EPR i.e. we stop at n where n is the number of constants in the problem
  - The input can restrict the size of the domain, then we can detect the absence of a model i.e. $X = Y \vee X = Z$ means $n \leq 2$
- Incrementality?
  - Idea (from Paradox): use and update single SAT solver
  - Requires us to retract totality constraints
  - Pros: we only have to generate new stuff, we get learned clauses
  - Cons: we lose variable elimination

Overview

AVATAR

- A general architecture for proof search based on the idea of splitting
- Still relatively new, very exciting, and you will hear about it a lot
- Helps Vampire solve a lot of new problems
- Allows for exciting new extensions for theory reasoning
  - Combine with decision procedures i.e. use a SMT solver
  - See VampireZ3 in CASC as a proof of idea

Splitting: The Necessary Details

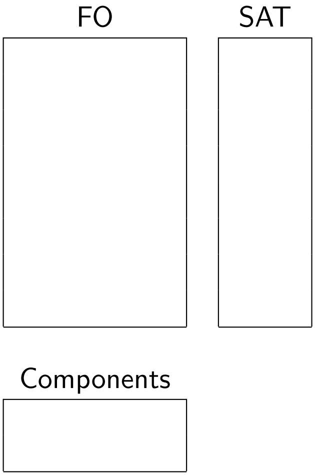
- *Motivation:* Reasoning with heavy/long clauses is expensive
- The set of clauses $S \cup (C_1 \vee \ldots \vee C_n)$ where C_i are minimal pairwise variable-disjoint components is satisfiable if all of $S \cup C_i$ are
- We call C_i a component and say C is splittable if $i > 1$
- In general, C_i is nicer than $C_1 \vee \ldots \vee C_n$
- Therefore, it suffices to explore each of $S \cup C_i$ separately
- To do this we need to
  1. Decide which C_i to assert/explore next
  2. Backtrack our decision if that branch is unsatisfiable
- In AVATAR we use a SAT solver to do this

AVATAR by Example

- Input:

  $p(a), \;\; q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ★ In FO use clauses with assertions
  - FO: Do FO proving
    - ★ Assertions must be preserved in inferences
  - Process refutation

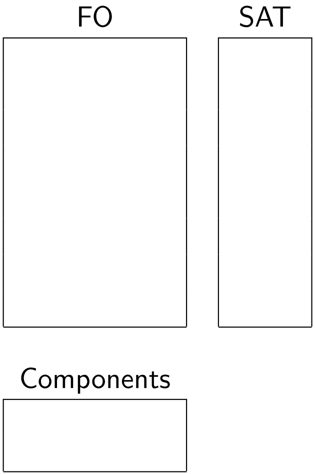


AVATAR by Example

- Input:

  $p(a), \quad q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ⋆ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ⋆ In FO use clauses with assertions
  - FO: Do FO proving
    - ⋆ Assertions must be preserved in inferences
  - Process refutation



AVATAR by Example

- Input:

  $p(a), \ q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ★ In FO use clauses with assertions
  - FO: Do FO proving
    - ★ Assertions must be preserved in inferences
  - Process refutation

FO

$p(a) \mid \{\}$

SAT

Components

AVATAR by Example

- Input:

  $p(a)$, $q(b)$, $\neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ★ In FO use clauses with assertions
  - FO: Do FO proving
    - ★ Assertions must be preserved in inferences
  - Process refutation

FO

$p(a) \mid \{\}$

$q(b) \mid \{\}$

SAT

Components

AVATAR by Example

- Input:

  $p(a), \;\; q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - In FO use clauses with assertions
  - FO: Do FO proving
    - Assertions must be preserved in inferences
  - Process refutation

FO	SAT
$p(a) \mid \{\}$ $q(b) \mid \{\}$	$1 \vee 2$

Components
$1 \mapsto \neg p(x)$
$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a), \; q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ★ In FO use clauses with assertions
  - FO: Do FO proving
    - ★ Assertions must be preserved in inferences
  - Process refutation

FO

$p(a) \mid \{\}$

$q(b) \mid \{\}$

SAT

$\underline{1} \vee 2$

Components

$1 \mapsto \neg p(x)$

$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a), \;\; q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ★ In FO use clauses with assertions
  - FO: Do FO proving
    - ★ Assertions must be preserved in inferences
  - Process refutation

FO	SAT
$p(a) \mid \{\}$ $q(b) \mid \{\}$ $\neg p(x) \mid \{1\}$	$\underline{1} \vee 2$

Components

$1 \mapsto \neg p(x)$

$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a),\ \ q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ★ In FO use clauses with assertions
  - FO: Do FO proving
    - ★ Assertions must be preserved in inferences
  - Process refutation

FO	SAT
$p(a) \mid \{\}$	$\underline{1} \vee 2$
$q(b) \mid \{\}$	
$\neg p(x) \mid \{1\}$	
$\bot \mid \{1\}$	

Components

$1 \mapsto \neg p(x)$

$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a), \;\; q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - In FO use clauses with assertions
  - FO: Do FO proving
    - Assertions must be preserved in inferences
  - Process refutation

FO	SAT
$p(a) \mid \{\}$	$\underline{1} \vee 2$
$q(b) \mid \{\}$	$\neg 1$
$\neg p(x) \mid \{1\}$	
$\bot \mid \{1\}$	

Components

$1 \mapsto \neg p(x)$
$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a), \;\; q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ⋆ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ⋆ In FO use clauses with assertions
  - FO: Do FO proving
    - ⋆ Assertions must be preserved in inferences
  - Process refutation

FO	SAT
$p(a) \mid \{\}$	$1 \vee \underline{2}$
$q(b) \mid \{\}$	$\underline{\neg 1}$
$\neg p(x) \mid \{1\}$	
$\bot \mid \{1\}$	

Components

$1 \mapsto \neg p(x)$
$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a), \quad q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ★ In FO use <u>clauses with assertions</u>
  - FO: Do FO proving
    - ★ Assertions must be preserved in inferences
  - Process refutation

FO

$p(a) \mid \{\}$

$q(b) \mid \{\}$

~~$\neg p(x) \mid \{1\}$~~

~~$\bot \mid \{1\}$~~

$\neg q(y) \mid \{2\}$

SAT

$1 \vee \underline{2}$

$\underline{\neg 1}$

Components

$1 \mapsto \neg p(x)$

$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a), \quad q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - In FO use clauses with assertions
  - FO: Do FO proving
    - Assertions must be preserved in inferences
  - Process refutation

FO

$p(a) \mid \{\}$

$q(b) \mid \{\}$

~~$\neg p(x) \mid \{1\}$~~

~~$\bot \mid \{1\}$~~

$\neg q(y) \mid \{2\}$

$\bot \mid \{2\}$

SAT

$1 \vee \underline{2}$

$\underline{\neg 1}$

Components

$1 \mapsto \neg p(x)$

$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a),\ \ q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - In FO use clauses with assertions
  - FO: Do FO proving
    - Assertions must be preserved in inferences
  - Process refutation

FO

$p(a) \mid \{\}$

$q(b) \mid \{\}$

~~$\neg p(x) \mid \{1\}$~~

~~$\bot \mid \{1\}$~~

$\neg q(y) \mid \{2\}$

$\bot \mid \{2\}$

SAT

$1 \vee \underline{2}$

$\underline{\neg 1}$

$\neg 2$

Components

$1 \mapsto \neg p(x)$

$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a),\quad q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - In FO use clauses with assertions
  - FO: Do FO proving
    - Assertions must be preserved in inferences
  - Process refutation

FO	SAT
$p(a) \mid \{\}$	$1 \vee 2$
$q(b) \mid \{\}$	$\neg 1$
~~$\neg p(x) \mid \{1\}$~~	$\neg 2$
~~$\bot \mid \{1\}$~~	
$\neg q(y) \mid \{2\}$	
$\bot \mid \{2\}$	

Components

$1 \mapsto \neg p(x)$

$2 \mapsto \neg q(y)$

AVATAR by Example

- Input:

  $p(a), \quad q(b), \neg p(x) \vee \neg q(y)$

- Repeat
  - FO: Process new clauses
    - ★ split clauses into components
  - SAT: Construct model
  - FO: Use model (do splitting)
    - ★ In FO use clauses with assertions
  - FO: Do FO proving
    - ★ Assertions must be preserved in inferences
  - Process refutation

- Refutation
  - From the SAT solver

FO	SAT
$p(a) \mid \{\}$	$1 \vee 2$
$q(b) \mid \{\}$	$\neg 1$
~~$\neg p(x) \mid \{1\}$~~	$\neg 2$
~~$\bot \mid \{1\}$~~	
$\neg q(y) \mid \{2\}$	
$\bot \mid \{2\}$	

Components

$1 \mapsto \neg p(x)$
$2 \mapsto \neg q(y)$

Varying the Architecture

- **Component Selection.**
  - What to do with ground literals?
  - What to do with unsplittable clauses?
- **What SAT solver to use, and how?**
  - Our own, MiniSAT, Lingeling
  - Setting various options
- **Minimizing the model.**
  - Do we need the whole model?
  - How does a partial model interact with splitting theory?

SAT Solver Effects

- What is clear:
  - The model produced by the SAT solver matters
  - Faster SAT solving can help
  - Incremental SAT solving can help
- What is unclear:
  - A lot...
  - How important the model is, what a nice model is
  - How important partial models are, what kind of partialness
  - How much information we should give the SAT solver
- Martin will say more today and on Thursday :)

Overview

Instance Generation

- *Observation:* By Hebrand Theorem, if a set of first-order clauses is unsatisfiable then there is a set of unsatisfiable ground instances that is also unsatisfiable
- The idea of Instance Generation is then as follows
  1. Given a set of first-order clauses S
  2. Produce ground abstraction $S\perp$ by mapping vars to fresh constant $\perp$
  3. If $S\perp$ is unsatisfiable then S is unsatisfiable
  4. Otherwise, attempt to refine the abstraction by adding clauses to S
  5. Goto 2
- Checking satisfiability of $S\perp$ can be done by a SAT solver

Refine the Abstraction?

- How can the abstraction be too general?

- Consider $S = \{p(f(x, a)), \neg p(f(b, y))\}$
- This gives $S\bot = \{p(f(\bot, a)), \neg p(f(b, \bot))\}$
- Which is SAT but S is unsatisfiable

- To refine the abstraction we add $p(f(b, a))$ and $\neg p(f(b, a))$

- Note that in the SAT solver $p(f(\bot, a))$ and $p(f(b, \bot))$ are just distinct variables

The InstGen rule

- This refinement is carried out by the InstGen rule:

$$\frac{C \vee L \qquad D \vee \overline{K}}{(C \vee L)\sigma \qquad (D \vee \overline{K})\sigma}$$

  where $\sigma = \mathtt{mgu}(L, K)$ and σ is a proper instantiator of L or K and both L and $\overline{K}$ are <u>selected</u>

- A literal is selected if it is appears in the model of the SAT solver
- This is based on the observation that the conflict that needs to be resolved by refinement is always between such literals

In Practice

- Instance Generation is applied as a saturation algorithm
- This means that we saturate (up to redundancy) the set of clauses with respect to the InstGen rule

- We can use a prolific constant from the problem in groundings

- We carry out restarts to reset the model periodically
- We use dismatching constraints to remove some redundant inferences

- We can combine with superposition by performing superposition proof search alongside this proof search and importing groundings of (unconditional) generated clauses into the SAT solver

Combination with AVATAR?

- One possible extension to this setup is to share the SAT solver
- Note that SAT variables are components in AVATAR and ground literals in Instance Generation but all ground literals are components
- Only get overlap if we use a constant from the problem for grounding
- Further idea, for component C in AVATAR add $[C] \rightarrow [C\gamma]$
- This connects non-ground parts of the AVATAR model with the Instance Generation model

Overview

Global Subsumption: the Ground Case

- This is a very effective simplification technique
- Let us consider the ground case first...

- Assume a set of first order clauses S
- Let S_{gr} be a set of ground clauses implied by S
  i.e. instances of clauses in S
- The ground clause $D \vee D'$ can be replaced by D in S if $S_{gr} \models D$
- This is sound as D follows from S and subsumes $D \vee D'$
- If D is empty then S_{gr} is unsatisfiable and so is S

Global Subsumption: the Non-Ground Case

- We can lift this to give the non-ground global subsumption rule:

$$\frac{C \vee C'}{C}$$

where $S_{gr} \models C\gamma$ for non-empty C' and injective substitution γ from variables in C to fresh constants

- For every generated clause C we
  1. Let $\gamma = [x_1 \mapsto c_1, \ldots x_n \mapsto c_n]$ for x_i in C and fresh c_i
  2. Add $C\gamma$ to S_{gr}
  3. Search for a minimal $C' \subset C$ such that $S_{gr} \models C'$

- We do not add more groundings to S_{gr} as we want this to be cheap

Example

- Take the following case:
  - $C = p(x, y) \vee r(x)$
  - $S = \{p(x, y) \vee r(x), p(x, x)\}$

- C cannot be reduced. Injectivity is important
  - If we do things wrong we can get $S_{gr} = \{p(a, b) \vee r(a), p(a, a)\}$
  - We check $\{p(a, a) \vee r(a), p(a, a), \neg p(a, a)\}$
  - We have $S_{gr} \models p(a, a)$ but $p(x, y)$ does not follow from S

- If we add $p(x, y)$ to S then C can be reduced
  - The correct grounding of S is $S_{gr} = \{p(a, b) \vee r(a), p(a, a), p(a, b)\}$
  - We check $\{p(a, b) \vee r(a), p(a, a), p(a, b), \neg p(a, b)\}$
  - C can be replaced by $p(x, y)$

SAT Solver Requirements

- As this a simplification technique we want it to be very quick
- Therefore, we only perform propagation in the SAT solver

- This means that we do not need the full power of the SAT solver
- One improvement would be to produce a restricted procedure that performs propagation only

Extending to combine with AVATAR?

- Currently only reason with unconditional clauses

- To reason with conditional clause $C \mid A$ we need to encode A in the SAT solver i.e. translate $A \rightarrow C\gamma$

- Then, when attempting to reduce $C \mid A$ we
  - Assert A for unconditional reduction
  - Assert AVATAR model for conditional reduction
    - Might need to extend A in reduced clause

- Further idea: use this method to attempt to reduce A

- Finally, we could share the SAT solver with AVATAR (or Instance Generation) but as noted above, we may want a restricted solver for Global Subsumption

Overview

1. Finite Model Building
2. AVATAR
3. Instance Generation
4. Global Subsumption
5. Other Ideas

Why the SAT Solver matters... and can we use this?

- In AVATAR and Instance Generation the model controls proof search
- *Idea: use Literal Selection to control the model generated*
- This requires a concept of nice model for each technique:
  - For AVATAR this might be about minimal change or minimality
  - For Instance Generation this might be about minimising the number of possible inferences or, conversely, to select more general inferences first i.e. those that make others redundant

Conclusions

- SAT solvers can provide powerful mechanisms for implementing effective techniques inside a first-order saturation prover
- But the way we use SAT solvers is not necessarily the same as the typical SAT usage
- Therefore, as well as improving the techniques themselves we can consider altering the SAT solver to improve performance