AVATAR: A SAT-based Architecture for First-Order Theorem-Provers

Marijn J.H. Heule
marijn@cs.utexas.edu

THE UNIVERSITY OF TEXAS AT AUSTIN

ACL2 Seminar, February 17, 2015

adaptation of a CAV'14 talk by Andrei Voronkov

AVATAR

Advanced
Vampire
Architecture for
Theories
And
Resolution

AVATAR

Definitions of *Avatar* from various dictionaries:

Advanced
Vampire
Architecture for
Theories
And
Resolution

AVATAR

Advanced
Vampire
Architecture for
Theories
And
Resolution

Definitions of *Avatar* from various dictionaries:

- Science Fiction: a hybrid creature, composed of human and alien DNA and remotely controlled by the mind of a genetically matched human being

AVATAR

Advanced
Vampire
Architecture for
Theories
And
Resolution

Definitions of *Avatar* from various dictionaries:

- Science Fiction: a hybrid creature, composed of human and alien DNA and remotely controlled by the mind of a genetically matched human being
- Hindu Mythology: the descent of a deity to the death in an incarnate form of some manifest shape; the incarnation of a god

AVATAR

Advanced
Vampire
Architecture for
Theories
And
Resolution

Definitions of *Avatar* from various dictionaries:

- Science Fiction: a hybrid creature, composed of human and alien DNA and remotely controlled by the mind of a genetically matched human being
- Hindu Mythology: the descent of a deity to the death in an incarnate form of some manifest shape; the incarnation of a god
- Automated Reasoning: a SAT solver embodied in a first-order theorem prover and in fact controlling its behavior

Summary

- Original motivation: problems having clauses containing propositional variables and other clauses that can split into components with disjoint sets of variables.
- Previously: splitting.
- New architecture: a first-order theorem-prover tightly integrated with a SAT or an SMT solver.
- Future: reasoning with both quantifiers and theories.

Context: Solve a Problem Abstraction using a SAT Solver

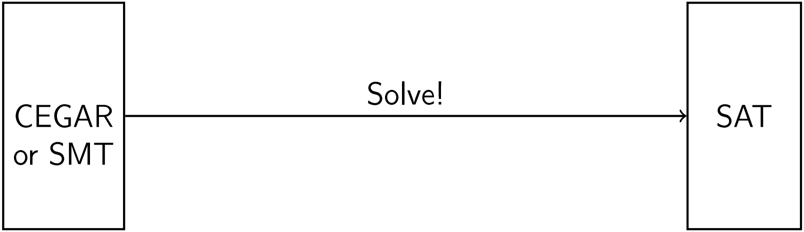
Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.

Context: Solve a Problem Abstraction using a SAT Solver

Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

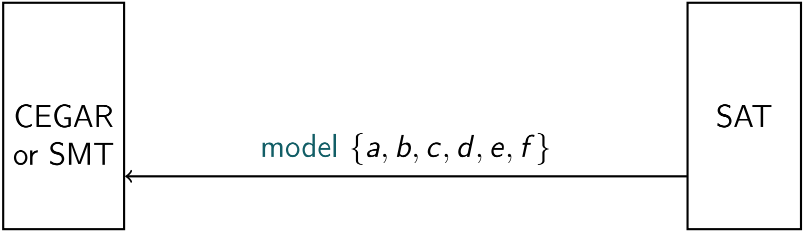
Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.



Context: Solve a Problem Abstraction using a SAT Solver

Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

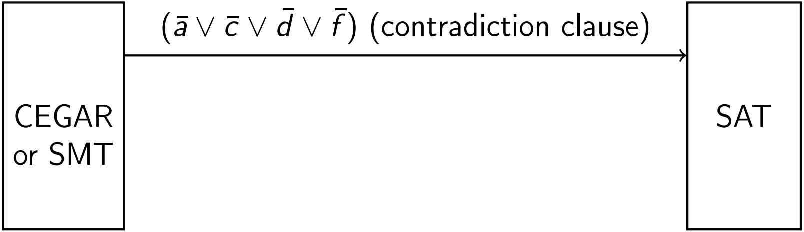
Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.



Context: Solve a Problem Abstraction using a SAT Solver

Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

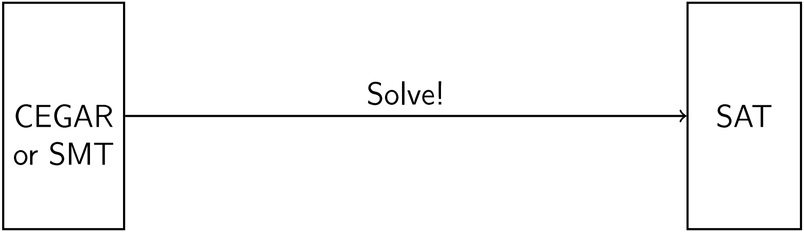
Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.



Context: Solve a Problem Abstraction using a SAT Solver

Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

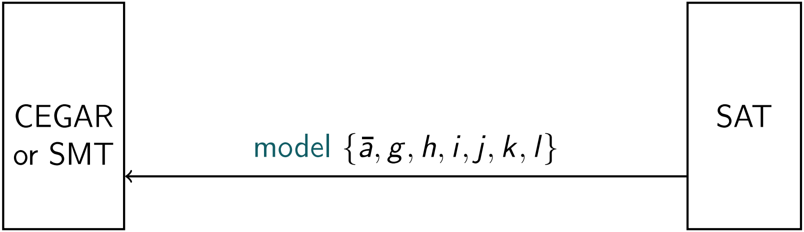
Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.



Context: Solve a Problem Abstraction using a SAT Solver

Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

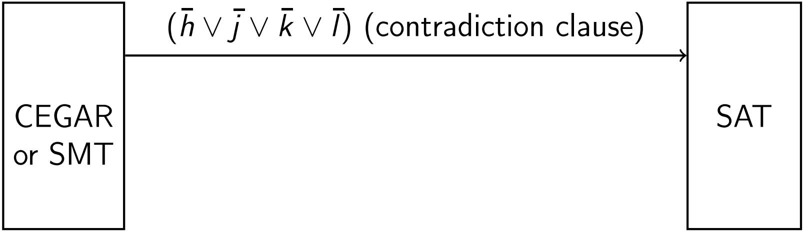
Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.



Context: Solve a Problem Abstraction using a SAT Solver

Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.



Context: Solve a Problem Abstraction using a SAT Solver

Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.

The loop terminates when either the SAT solver reports UNSAT or the model satisfies the original problem.

Context: Solve a Problem Abstraction using a SAT Solver

Counter-Example Guided Abstraction Refinement (CEGAR):
Only translate a subset of the constraints into SAT.

Satisfiability Modulo Theories (SMT):
Combine a SAT solver with theory solvers.

The loop terminates when either the SAT solver reports UNSAT or the model satisfies the original problem.

Can this architecture be used for first-order theorem provers?

Saturation Algorithms in First-Order Theorem-Provers

A formula F is saturated with respect to an inference system I if for every inference in I with premises in F the conclusion of the inferences is in F as well (or subsumed by a clause in F).

Saturation Algorithms in First-Order Theorem-Provers

A formula F is saturated with respect to an inference system I if for every inference in I with premises in F the conclusion of the inferences is in F as well (or subsumed by a clause in F).

Typically three kinds of inferences:

- Generation: add new clauses to the formula (resolution);
- Simplification: simplify clauses with existing clauses (self-subsumption);
- Deletion: remove clauses from the formula (subsumption).

Saturation Algorithms in First-Order Theorem-Provers

A formula F is saturated with respect to an inference system I if for every inference in I with premises in F the conclusion of the inferences is in F as well (or subsumed by a clause in F).

Typically three kinds of inferences:

- Generation: add new clauses to the formula (resolution);
- Simplification: simplify clauses with existing clauses (self-subsumption);
- Deletion: remove clauses from the formula (subsumption).

Possible outcomes of a saturation algorithms:

- if the empty clause is derived, then F is unsatisfiable;
- if saturation terminates, then F is satisfiable;
- if saturation runs forever, then F is satisfiable.

Saturation Algorithms in First-Order Theorem-Provers

A formula F is saturated with respect to an inference system I if for every inference in I with premises in F the conclusion of the inferences is in F as well (or subsumed by a clause in F).

Typically three kinds of inferences:

- Generation: add new clauses to the formula (resolution);
- Simplification: simplify clauses with existing clauses (self-subsumption);
- Deletion: remove clauses from the formula (subsumption).

Possible outcomes of a saturation algorithms:

- if the empty clause is derived, then F is unsatisfiable;
- if saturation terminates, then F is satisfiable;
- if saturation runs too long, then F is unknown.

FLoC Olympic Games

- CASC (FO solvers versus FO solvers)
- SAT (SAT solvers versus SAT solvers)
- SMT (SMT solvers versus SMT solvers)
- ...

FLoC Olympic Games

- CASC (FO solvers versus FO solvers)
- SAT (SAT solvers versus SAT solvers)
- SMT (SMT solvers versus SMT solvers)
- ...

Why not FO solvers versus SAT solvers ???

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
$\mid x$ (decide)

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
$\mid x$ (decide)
$\emptyset \mid x$ (unit propagation)

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

subsumption

$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
$\mid x$ (decide)
$\emptyset \mid x$ (unit propagation)

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$ (conflict clause)

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$ (conflict clause)
$\emptyset$ (unit propagation)

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$ (conflict clause)
$\emptyset$ (unit propagation)

SAT solver won!

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$ (conflict clause)
$\emptyset$ (unit propagation)

SAT solver won!

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$
(x) (resolution)
(y) (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$ (conflict clause)
$\emptyset$ (unit propagation)

SAT solver won!

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

subsumption

$(\bar{x} \vee \bar{y})$

(x) (resolution)

(y) (resolution)

SAT solver:

$(x \vee y)$

$(x \vee \bar{y})$

$(\bar{x} \vee y)$

$(\bar{x} \vee \bar{y})$

$(\bar{x})$ (conflict clause)

$\emptyset$ (unit propagation)

SAT solver won!

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(\bar{x} \vee \bar{y})$

(x) (resolution)

(y) (resolution)

SAT solver:

$(x \vee y)$

$(x \vee \bar{y})$

$(\bar{x} \vee y)$

$(\bar{x} \vee \bar{y})$

$(\bar{x})$ (conflict clause)

$\emptyset$ (unit propagation)

SAT solver won!

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$(\bar{x} \vee \bar{y})$
(x) (resolution)
(y) (resolution)
$(\bar{y})$ (resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$ (conflict clause)
$\emptyset$ (unit propagation)

SAT solver won!

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

subsumption

(x)	(resolution)
(y)	(resolution)
$(\bar{y})$	(resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$	(conflict clause)
$\emptyset$	(unit propagation)

SAT solver won!

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

(x)	(resolution)
(y)	(resolution)
$(\bar{y})$	(resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$	(conflict clause)
$\emptyset$	(unit propagation)

SAT solver won!

Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

(x)	(resolution)
(y)	(resolution)
$(\bar{y})$	(resolution)
$\emptyset$	(resolution)

SAT solver:

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x})$	(conflict clause)
$\emptyset$	(unit propagation)

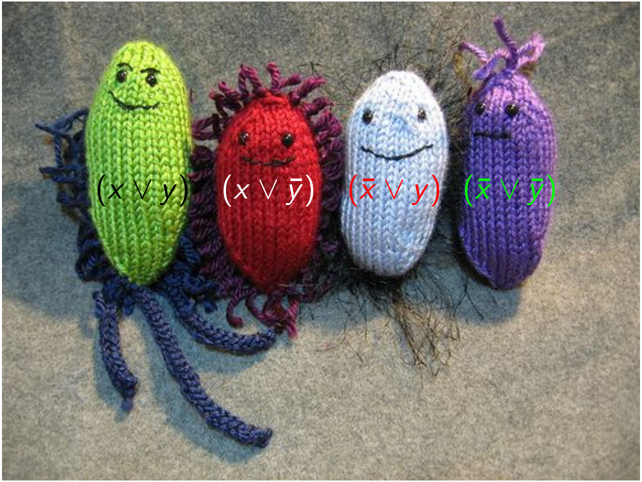
SAT solver won!

Search Space in Saturation Algorithms (1)

Illustrated using bacteria.

Search Space in Saturation Algorithms (1)

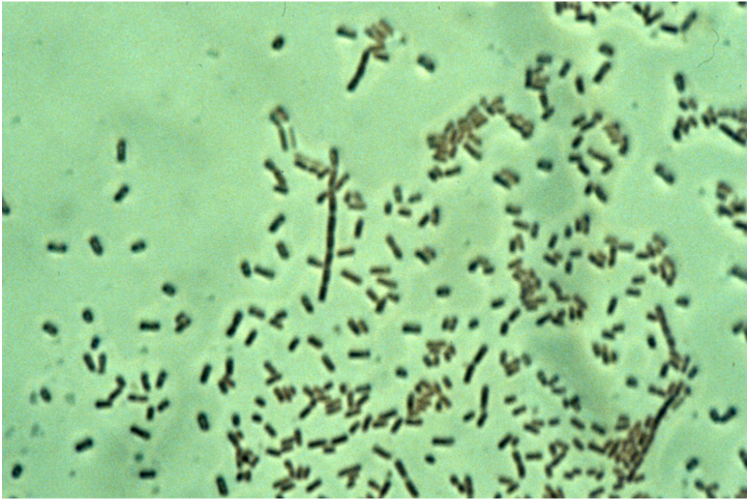
Illustrated using bacteria. In the beginning . . .



precisionnutrition.com

Search Space in Saturation Algorithms (2)

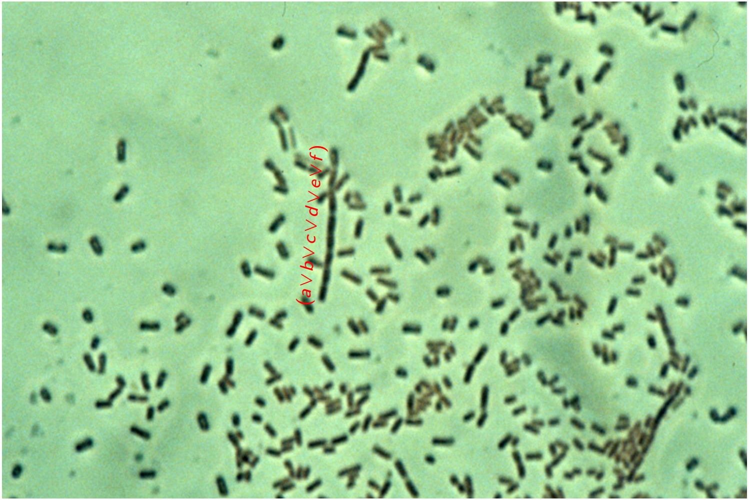
After a few steps . . .



www.nrcs.usda.gov

Search Space in Saturation Algorithms (2)

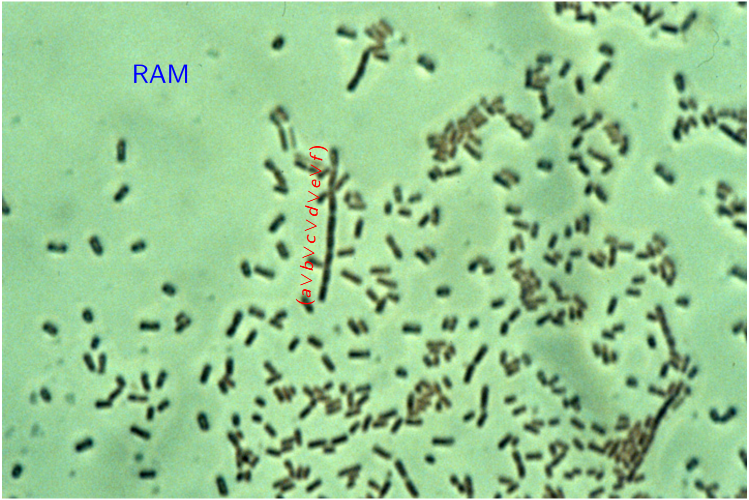
After a few steps ... and notice long clauses



www.nrcs.usda.gov

Search Space in Saturation Algorithms (2)

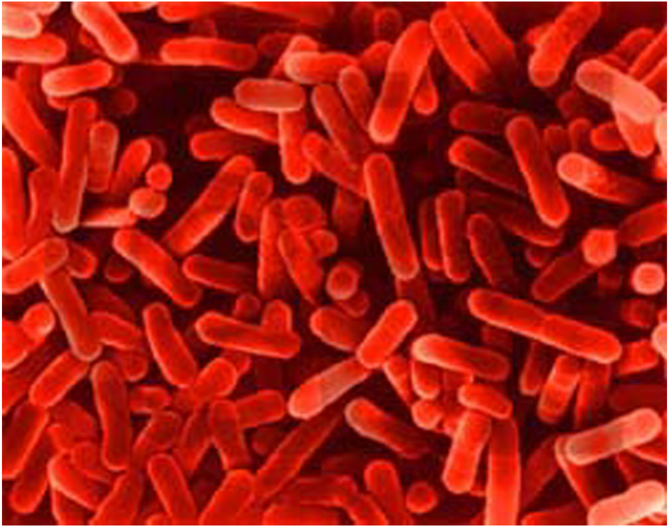
After a few steps . . . and notice long clauses



www.nrcs.usda.gov

Search Space in Saturation Algorithms (3)

After a few more steps ...



creepypasta.wikia.com

Reality of First-Order Theorem Proving

- Growing search spaces
- Repeated applications of algorithms whose complexity depends on clause sizes: resolution, superposition, demodulation, Knuth-Bendix order comparison, subsumption.
- Long clauses are a problem: produce even longer clauses; subsumption is NP-complete.

Long Clauses: Resolution

Example: resolving

$$p(x, f(y)) \vee p(f(x), y) \vee p(g(x, z), f(f(y))) \vee p(f(y), z) \vee$$
$$\bar{p}(g(z, z), g(y, f(x)) \vee p(f(a, x), g(z, g(y, z))) \vee \bar{p}(x, y)$$

against

$$\bar{p}(f(w), v) \vee p(f(v), w) \vee p(g(v, u), f(f(w))) \vee p(f(w), u) \vee$$
$$\bar{p}(g(u, u), g(w, f(v))) \vee p(f(a, v), g(u, g(w, u))) \vee \bar{p}(v, w)$$

Long Clauses: Resolution

Example: resolving

$$p(x, f(y)) \lor p(f(x), y) \lor p(g(x, z), f(f(y))) \lor p(f(y), z) \lor$$
$$\bar{p}(g(z, z), g(y, f(x)) \lor p(f(a, x), g(z, g(y, z))) \lor \bar{p}(x, y)$$

against

$$\bar{p}(f(w), v) \lor p(f(v), w) \lor p(g(v, u), f(f(w))) \lor p(f(w), u) \lor$$
$$\bar{p}(g(u, u), g(w, f(v))) \lor p(f(a, v), g(u, g(w, u))) \lor \bar{p}(v, w)$$

gives

$$p(f(f(w)), y) \lor p(g(f(w), z), f(f(y))) \lor p(f(y), z) \lor$$
$$\bar{p}(g(z, z), g(y, f(f(w))) \lor p(f(a, f(w)), g(z, g(y, z))) \lor$$
$$\bar{p}(f(w), y) \lor p(f(f(y)), w) \lor p(g(f(y), u), f(f(w))) \lor$$
$$p(f(w), u) \lor \bar{p}(g(u, u), g(w, f(f(y)))) \lor$$
$$p(f(a, f(y)), g(u, g(w, u))) \lor \bar{p}(f(y), w).$$

Long Clauses: Subsumption

Example: does

$$p(f(f(w)),y) \vee p(g(f(w),z),f(f(y))) \vee \bar{p}(f(w),y) \vee$$
$$\bar{p}(g(z,z),g(y,f(f(w)))) \vee p(f(a,f(w)),g(z,g(y,z))) \vee$$
$$p(f(y),z) \vee p(f(f(y),w) \vee p(g(f(y),u),f(f(w))) \vee$$
$$\bar{p}(g(u,u),g(w,f(f(y)))) \vee p(g(a,f(y)),g(u,g(w,u))) \vee$$
$$\bar{p}(f(y),w) \vee p(f(w),u)$$

subsume

$$p(g(f(y),u),f(f(g(x,y)))) \vee p(f(f(g(x,y))),y) \vee$$
$$p(f(y),z) \vee p(g(f(g(x,y)),z),f(f(y))) \vee p(f(g(x,y)),u) \vee$$
$$\bar{p}(g(z,z),g(y,f(f(g(x,y))))) \vee \bar{p}(f(g(x,y)),y) \vee$$
$$p(f(a,f(g(x,y))),g(z,g(y,z))) \vee p(f(f(y)),g(x,y)) \vee$$
$$p(g(a,f(y)),g(u,g(g(x,y),u))) \vee \bar{p}(f(y),g(x,y)) \vee$$
$$\bar{p}(g(u,u),g(g(x,y),f(f(y)))) \qquad ???$$

Long Clauses: Subsumption

Example: does

$$p(f(f(w)), y) \vee p(g(f(w), z), f(f(y))) \vee \bar{p}(f(w), y) \vee$$
$$\bar{p}(g(z, z), g(y, f(f(w)))) \vee p(f(a, f(w)), g(z, g(y, z))) \vee$$
$$p(f(y), z) \vee p(f(f(y), w) \vee p(g(f(y), u), f(f(w))) \vee$$
$$\bar{p}(g(u, u), g(w, f(f(y)))) \vee p(g(a, f(y)), g(u, g(w, u))) \vee$$
$$\bar{p}(f(y), w) \vee p(f(w), u)$$

subsume

$$p(g(f(y), u), f(f(g(x, y)))) \vee p(f(f(g(x, y))), y) \vee$$
$$p(f(y), z) \vee p(g(f(g(x, y)), z), f(f(y))) \vee p(f(g(x, y)), u) \vee$$
$$\bar{p}(g(z, z), g(y, f(f(g(x, y))))) \vee \bar{p}(f(g(x, y)), y) \vee$$
$$p(f(a, f(g(x, y))), g(z, g(y, z))) \vee p(f(f(y)), g(x, y)) \vee$$
$$p(g(a, f(y)), g(u, g(g(x, y), u))) \vee \bar{p}(f(y), g(x, y)) \vee$$
$$\bar{p}(g(u, u), g(g(x, y), f(f(y))))$$

???

Basis for DPLL

Consider the formula $F \cup \{C_1 \vee \cdots \vee C_n\}$, where $C_1 \vee \cdots \vee C_n$ is splittable.

Then $F \cup C_1 \vee \cdots \vee C_n$ is unsatisfiable is and only if each of

$$F \cup C_1$$
$$\ldots$$
$$F \cup C_n$$

is unsatisfiable too.

Basis for DPLL

Consider the formula $F \cup \{C_1 \vee \cdots \vee C_n\}$, where $C_1 \vee \cdots \vee C_n$ is splittable.

Then $F \cup C_1 \vee \cdots \vee C_n$ is unsatisfiable is and only if each of

$$F \cup C_1$$
$$\ldots$$
$$F \cup C_n$$

is unsatisfiable too.

Cannot be used in first-order logic:

- $\{p(x) \vee q(x), \bar{p}(a), \bar{q}(b)\}$ is satisfiable, while
- $\{p(x), \bar{p}(a), \bar{q}(b)\}$ and $\{q(x), \bar{p}(a), \bar{q}(b)\}$ are unsatisfiable.

Basis for DPLL

Consider the formula $F \cup \{ C_1 \vee \cdots \vee C_n \}$, where $C_1 \vee \cdots \vee C_n$ is splittable.

Then $F \cup C_1 \vee \cdots \vee C_n$ is unsatisfiable is and only if each of

$$F \cup C_1$$
$$\ldots$$
$$F \cup C_n$$

is unsatisfiable too.

Cannot be used in first-order logic:

- $\{p(x) \vee q(x), \bar{p}(a), \bar{q}(b)\}$ is satisfiable, while
- $\{p(x), \bar{p}(a), \bar{q}(b)\}$ and $\{q(x), \bar{p}(a), \bar{q}(b)\}$ are unsatisfiable.

Yet it can be used when $C_1 \vee \cdots \vee C_n$ have pairwise disjoint sets of variables.

Components, Splitting

Let $C_1, \ldots, C_n$ be clauses with disjoint sets of variables, $n \geq 2$.

The clause $D = C_1 \vee \cdots \vee C_n$ is splittable int $C_1, \ldots, C_n$.

If a clause is splittable, it has a maximal splitting, which can be found by the union-find algorithm.

Previous implementations:

- Splitting with backtracking (hard to implement, moderate improvement);
- Splitting without backtracking (rarely improves);

Splitting with Backtracking

Splitting with Backtracking

$$(x \vee y)$$
$$(x \vee \bar{y})$$
$$(\bar{x} \vee y)$$
$$(\bar{x} \vee \bar{y})$$

Splitting with Backtracking

split

$(x \vee y)$
$(x \vee \bar{y})$
$(\bar{x} \vee y)$
$(\bar{x} \vee \bar{y})$

$(\bar{x}) \mid \bar{x}$

Splitting with Backtracking

subsumption

$(x \vee y)$
$(x \vee \bar{y})$

$(\bar{x}) \mid \bar{x}$

Splitting with Backtracking

split

$(x \vee y)$
$(x \vee \bar{y})$

$(\bar{x}) \mid \bar{x}$

$(x) \mid x$

Splitting with Backtracking

subsumption

$(\bar{x}) \mid \bar{x}$

$(x) \mid x$

Splitting with Backtracking

resolution

$$(\bar{x}) \mid \bar{x}$$

$$(x) \mid x$$

$$\emptyset \mid x, \bar{x}$$

Splitting with Backtracking

$(x \vee y)$
$(x \vee \bar{y})$

backtrack

$(\bar{x}) \mid \bar{x}$

Splitting with Backtracking

split

$(x \vee y)$
$(x \vee \bar{y})$

$(\bar{x}) \mid \bar{x}$

$(\bar{y})$

Splitting with Backtracking

$(x \vee y)$

subsumption

$(\bar{x}) \mid \bar{x}$

$(\bar{y})$

Splitting with Backtracking

split $(x \vee y)$

$(\bar{x}) \mid \bar{x}$

$(x) \mid x$ $\quad$ $(\bar{y})$

Splitting with Backtracking

subsumption

$(\bar{x}) \mid \bar{x}$

$(x) \mid x$ $(\bar{y})$

Splitting with Backtracking

resolution

$(\bar{x}) \mid \bar{x}$

$(x) \mid x$ $(\bar{y})$

$\emptyset \mid x, \bar{x}$

Splitting with Backtracking

$$(\bar{x}) \mid \bar{x}$$

$$(x) \mid x \qquad (\bar{y})$$

$$\emptyset \mid x, \bar{x}$$

And so on ...

- Too many steps (for this example);
- Backtracking is expensive;
- Generally behaves well;
- Exploits too many branches ...

Clauses with Assertions

An new data-structure for rapid splitting with backtracking:
Assertion clauses $D \leftarrow A$ or $(C_1 \vee \cdots \vee C_n) \leftarrow C'_1, \ldots, C'_m$

All inference rules can be easily converted using assertion clauses:

$$\frac{D_1 \quad \ldots \quad D_k}{D}$$

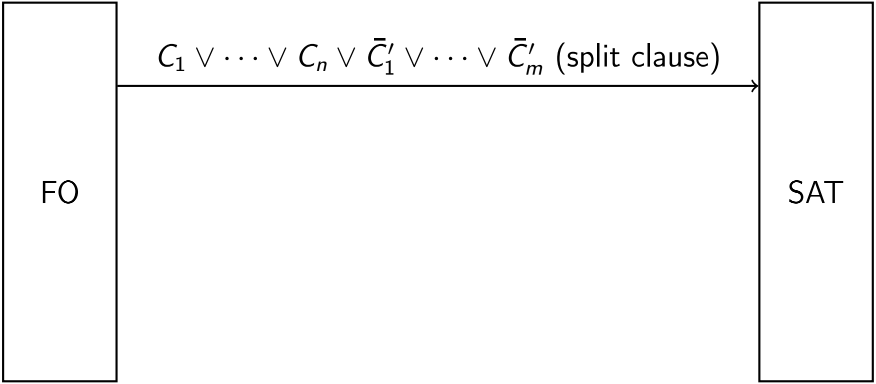
$$\frac{D_1 \leftarrow A_1 \quad \ldots \quad D_k \leftarrow A_k}{D \leftarrow A_1 \cup \cdots \cup A_k}$$

AVATAR

A SAT solver, which treats a component as a propositional variable.

AVATAR

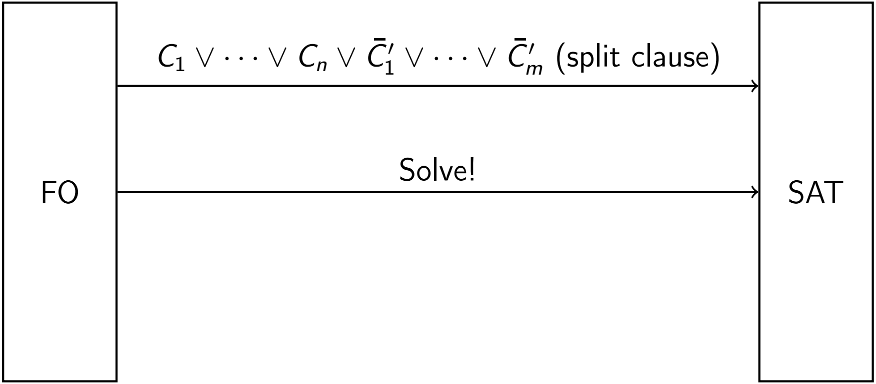
A SAT solver, which treats a component as a propositional variable.



Derives $C_1 \vee \cdots \vee C_n \mid C'_1, \ldots, C'_m$

AVATAR

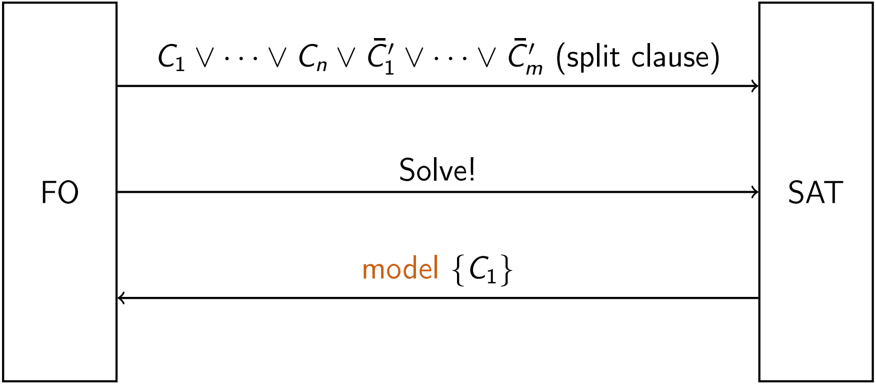
A SAT solver, which treats a component as a propositional variable.



Derives $C_1 \vee \cdots \vee C_n \mid C'_1, \ldots, C'_m$

AVATAR

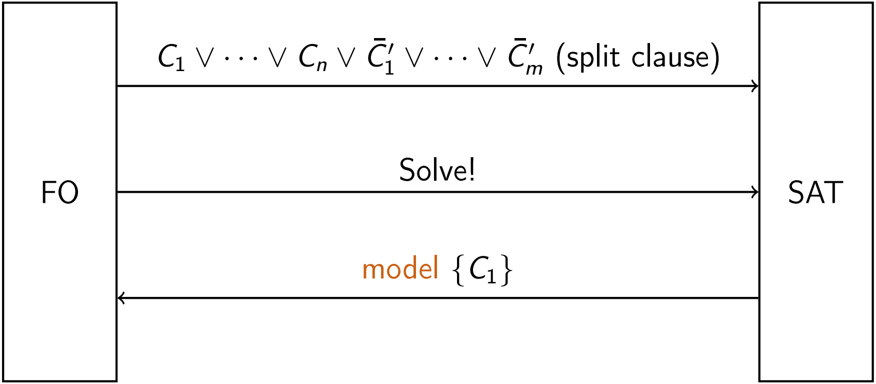
A SAT solver, which treats a component as a propositional variable.



Derives $C_1 \vee \cdots \vee C_n \mid C'_1, \ldots, C'_m$

AVATAR

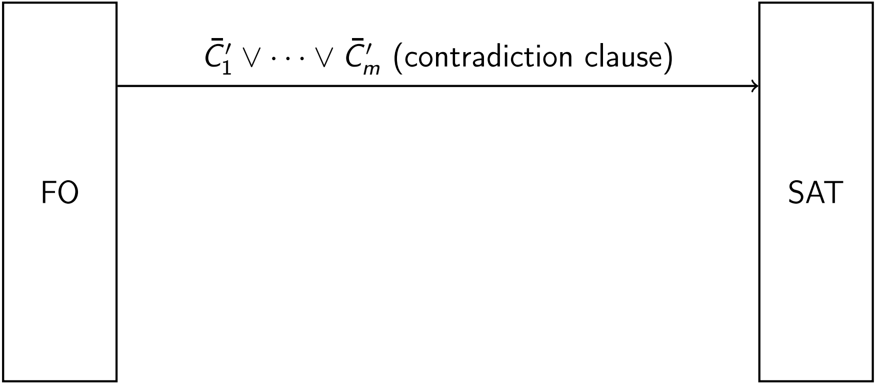
A SAT solver, which treats a component as a propositional variable.



Assert $C_1 \mid C_1$, analogue of backing if model changes

AVATAR

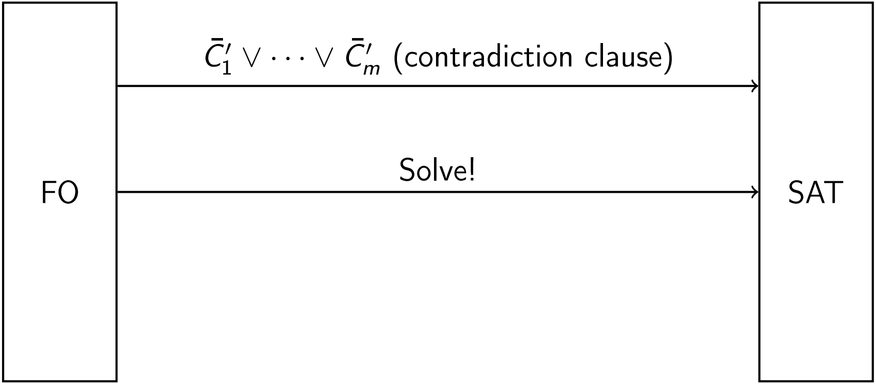
A SAT solver, which treats a component as a propositional variable.



Derives $\emptyset \mid C'_1, \ldots, C'_m$

AVATAR

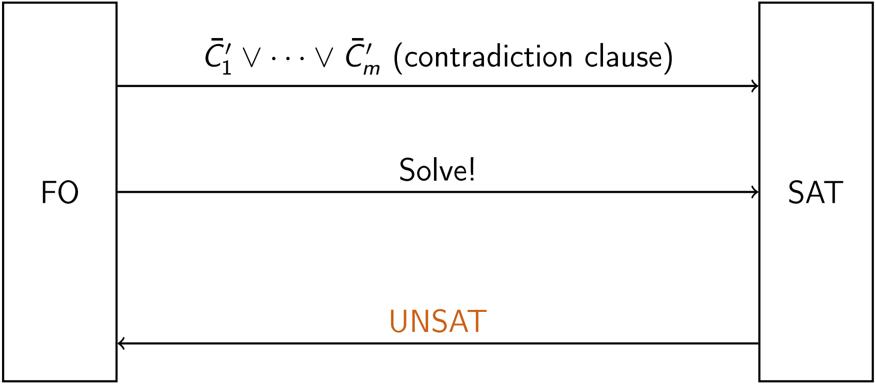
A SAT solver, which treats a component as a propositional variable.



Derives $\emptyset \mid C'_1, \ldots, C'_m$

AVATAR

A SAT solver, which treats a component as a propositional variable.



Derives $\emptyset \mid C'_1, \ldots, C'_m$

Problems

Implementing AVATAR heavily affect the saturation algorithm, redundancy and indexing.

- Clause deletion and undeletion via frozen clauses;
- Redundancy checking;
- Indexing with frozen clauses

Results

- Over 400 TPTP problems previously unsolved by any prover (including Vampire), probably unmatched since the TPTP appeared.
- About 5-10% increase in the number of problems solved by a single strategy.
- All splitting options and a lot of hard-to-maintain code removed from Vampire.

Results

- Over 400 TPTP problems previously unsolved by any prover (including Vampire), probably unmatched since the TPTP appeared.
- About 5-10% increase in the number of problems solved by a single strategy.
- All splitting options and a lot of hard-to-maintain code removed from Vampire.

CASC 2014 results of first-order theorems:

First-order Theorems	*Vampire* 2.6	ET 0.1	E 1.9	VanHElsi 1.0	CVC4 1.4–FOF	iProver 1.4	leanCoP 2.2	Prover9 1109a	Zipperpos 0.4–FOF	Muscadet 4.4	Princess 140704
Solved/400	375/400	339/400	321/400	310/400	215/400	216/400	158/400	95/400	73/400	32/400	134/400
Av. CPU Time	13.19	29.31	22.88	17.29	46.03	18.11	55.15	41.45	28.81	19.74	69.31
Solutions	372/400	339/400	321/400	310/400	215/400	214/400	158/400	95/400	73/400	30/400	0/400
µEfficiency	571	361	466	168	228	216	129	119	75	47	17
SOTAC	0.22	0.18	0.17	0.17	0.15	0.16	0.14	0.14	0.13	0.12	0.13
Core Usage	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.22
New Solved	5/6	5/6	0/6	0/6	0/6	6/6	0/6	0/6	0/6	0/6	0/6

Future Work

- SMT solver instead of SAT solver (already implemented)
- Arbitrary theory reasoning
- Many questions about AVATAR itself

AVATAR: A SAT-based Architecture for First-Order Theorem-Provers

Marijn J.H. Heule
marijn@cs.utexas.edu

THE UNIVERSITY OF TEXAS AT AUSTIN

ACL2 Seminar, February 17, 2015

adaptation of a CAV'14 talk by Andrei Voronkov