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

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

ACL2 Seminar, February 17, 2015

adaptation of a CAV’14 talk by Andrei Voronkov
**AVATAR**

Advanced Vampire Architecture for Theories And Resolution
Definitions of *Avatar* from various dictionaries:

- **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
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
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.
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.
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.
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.

\[(\bar{a} \lor \bar{c} \lor \bar{d} \lor \bar{f})\] (contradiction clause)
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.
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.

\((\overline{h} \lor \overline{j} \lor \overline{k} \lor \overline{l})\) (contradiction clause)
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.
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)
- ...
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:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]

SAT solver:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]

SAT solver:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[
\begin{align*}
(x \lor y) \\
(x \lor \bar{y}) \\
(\bar{x} \lor y) \\
(\bar{x} \lor \bar{y}) \\
(x) \quad \text{(resolution)}
\end{align*}
\]

SAT solver:

\[
\begin{align*}
(x \lor y) \\
(x \lor \bar{y}) \\
(\bar{x} \lor y) \\
(\bar{x} \lor \bar{y})
\end{align*}
\]
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
\[(x) \text{ (resolution)}\]

SAT solver:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
\[| \ x \text{ (decide)}\]
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
\[(x) \ (resolution)\]

SAT solver:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
\[\emptyset \mid x \ (decide)\]
\[\emptyset \mid x \ (unit \ propagation)\]
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[
\begin{align*}
\text{subsumption} \\
(\bar{x} \lor y) \\
(\bar{x} \lor \bar{y}) \\
(x) \quad \text{(resolution)}
\end{align*}
\]

SAT solver:

\[
\begin{align*}
(x \lor y) \\
(x \lor \bar{y}) \\
(\bar{x} \lor y) \\
(\bar{x} \lor \bar{y}) \\
\emptyset \mid x \quad \text{(decide)} \\
\emptyset \mid x \quad \text{(unit propagation)}
\end{align*}
\]
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[ (\bar{x} \lor y) \]
\[ (\bar{x} \lor \bar{y}) \]
\[ (x) \quad \text{(resolution)} \]

SAT solver:

\[ (x \lor y) \]
\[ (x \lor \bar{y}) \]
\[ (\bar{x} \lor y) \]
\[ (\bar{x} \lor \bar{y}) \]
\[ (\bar{x}) \quad \text{(conflict clause)} \]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[
\begin{align*}
(\overline{x} \lor y) \\
(\overline{x} \lor \overline{y}) \\
(x) \quad \text{(resolution)}
\end{align*}
\]

SAT solver:

\[
\begin{align*}
(x \lor y) \\
(x \lor \overline{y}) \\
(\overline{x} \lor y) \\
(\overline{x} \lor \overline{y}) \\
(\overline{x}) \quad \text{(conflict clause)} \\
\emptyset \quad \text{(unit propagation)}
\end{align*}
\]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[
\begin{align*}
(\bar{x} \lor y) \\
(\bar{x} \lor \bar{y}) \\
(x) \quad \text{(resolution)}
\end{align*}
\]

SAT solver:

\[
\begin{align*}
(x \lor y) \\
(x \lor \bar{y}) \\
(\bar{x} \lor y) \\
(\bar{x} \lor \bar{y}) \\
(\bar{x}) \quad \text{(conflict clause)} \\
\emptyset \quad \text{(unit propagation)}
\end{align*}
\]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[(\overline{x} \lor y)\]
\[(\overline{x} \lor \overline{y})\]
\[(x)\] (resolution)

SAT solver:

\[(x \lor y)\]
\[(x \lor \overline{y})\]
\[(\overline{x} \lor y)\]
\[(\overline{x} \lor \overline{y})\]
\[(\overline{x})\] (conflict clause)
\[\emptyset\] (unit propagation)

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
\[(x) \text{ (resolution)}\]
\[(y) \text{ (resolution)}\]

SAT solver:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
\[(\bar{x}) \text{ (conflict clause)}\]
\[\emptyset \text{ (unit propagation)}\]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[(\bar{x} \lor \bar{y})\]

\[(x) \quad \text{(resolution)}\]

\[(y) \quad \text{(resolution)}\]

SAT solver:

\[(x \lor y)\]

\[(x \lor \bar{y})\]

\[(\bar{x} \lor y)\]

\[(\bar{x} \lor \bar{y})\]

\[(\bar{x}) \quad \text{(conflict clause)}\]

\[\emptyset \quad \text{(unit propagation)}\]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

$$(\overline{x} \lor \overline{y})$$
$$(x) \quad \text{(resolution)}$$
$$(y) \quad \text{(resolution)}$$

SAT solver:

$$(x \lor y)$$
$$(x \lor \overline{y})$$
$$(\overline{x} \lor y)$$
$$(\overline{x} \lor \overline{y})$$

$$(\overline{x}) \quad \text{(conflict clause)}$$
$$\emptyset \quad \text{(unit propagation)}$$

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[(\overline{x} \lor \overline{y})\]
\[(x) \quad \text{(resolution)}\]
\[(y) \quad \text{(resolution)}\]
\[(\overline{y}) \quad \text{(resolution)}\]

SAT solver:

\[(x \lor y)\]
\[(x \lor \overline{y})\]
\[(\overline{x} \lor y)\]
\[(\overline{x} \lor \overline{y})\]
\[(\overline{x}) \quad \text{(conflict clause)}\]
\[\emptyset \quad \text{(unit propagation)}\]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

subsumption

\[(x) \quad \text{(resolution)}\]
\[(y) \quad \text{(resolution)}\]
\[(\bar{y}) \quad \text{(resolution)}\]

SAT solver:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
\[(\bar{x}) \quad \text{(conflict clause)}\]
\[\emptyset \quad \text{(unit propagation)}\]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

\[
(x) \quad \text{(resolution)}
\]
\[
(y) \quad \text{(resolution)}
\]
\[
(\bar{y}) \quad \text{(resolution)}
\]

SAT solver:

\[
(x \lor y)
\]
\[
(x \lor \bar{y})
\]
\[
(\bar{x} \lor y)
\]
\[
(\bar{x} \lor \bar{y})
\]
\[
(\bar{x}) \quad \text{(conflict clause)}
\]
\[
\emptyset \quad \text{(unit propagation)}
\]

SAT solver won!
Saturation Algorithms versus SAT (CDCL) solvers

Resolution prover:

Resolution:

\[(x)\] (resolution)
\[(y)\] (resolution)
\[(\bar{y})\] (resolution)
\[\emptyset\] (resolution)

SAT solver:

SAT Solver:

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]

\[(\bar{x})\] (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 . . .
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
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 . . .
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

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

against

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

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

subsume

\[ p(g(f(y), u), f(f(g(x, y)))) \lor p(f(f(g(x, y))), y) \lor \]
\[ p(f(y), z) \lor p(g(f(g(x, y)), z), f(f(y))) \lor p(f(g(x, y)), u) \lor \]
\[ \bar{p}(g(z, z), g(y, f(f(g(x, y)))) \lor \bar{p}(f(g(x, y)), y) \lor \]
\[ p(f(a, f(g(x, y))), g(z, g(y, z))) \lor p(f(f(y)), g(x, y)) \lor \]
\[ p(g(a, f(y)), g(u, g(g(x, y), u))) \lor \bar{p}(f(y), g(x, y)) \lor \]
\[ \bar{p}(g(u, u), g(g(x, y), f(f(y)))) \]

???
Long Clauses: Subsumption

Example: does

\[ p(f(f(w)), y) \lor p(g(f(w), z), f(f(y))) \lor \neg p(f(w), y) \lor \neg p(g(z, z), g(y, f(f(w)))) \lor p(f(a, f(w)), g(z, g(y, z))) \lor p(f(y), z) \lor p(f(f(y), w) \lor p(g(f(y), u), f(f(w))) \lor \neg p(g(u, u), g(w, f(f(y)))) \lor p(g(a, f(y)), g(u, g(w, u))) \lor \neg p(f(y), w) \lor p(f(w), u) \]

subsume

\[ p(g(f(y), u), f(f(g(x, y)))) \lor p(f(f(g(x, y))), y) \lor p(f(y), z) \lor p(g(f(g(x, y)), z), f(f(y))) \lor p(f(g(x, y)), u) \lor \neg p(g(z, z), g(y, f(f(g(x, y)))) \lor \neg p(f(g(x, y)), y) \lor p(f(a, f(g(x, y))), g(z, g(y, z))) \lor p(f(f(y)), g(x, y)) \lor p(g(a, f(y)), g(u, g(g(x, y), u))) \lor \neg p(f(y), g(x, y)) \lor \neg p(g(u, u), g(g(x, y), f(f(y)))) \]
Basis for DPLL

Consider the formula \( F \cup \{ C_1 \lor \cdots \lor C_n \} \), where \( C_1 \lor \cdots \lor C_n \) is splittable.

Then \( F \cup C_1 \lor \cdots \lor C_n \) is unsatisfiable is and only if each of

\[
\begin{align*}
F \cup C_1 \\
\vdots \\
F \cup C_n
\end{align*}
\]

is unsatisfiable too.
Basis for DPLL

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

Then $F \cup C_1 \lor \cdots \lor C_n$ 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:

- $\{p(x) \lor q(x), \bar{p}(a), \bar{q}(b)\}$ 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 $F \cup \{C_1 \lor \cdots \lor C_n\}$, where $C_1 \lor \cdots \lor C_n$ is splittable.

Then $F \cup C_1 \lor \cdots \lor C_n$ 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:

- {\(p(x) \lor q(x), \overline{p}(a), \overline{q}(b)\)} is satisfiable, while
- \{{\(p(x), \overline{p}(a), \overline{q}(b)\}} and \{{\(q(x), \overline{p}(a), \overline{q}(b)\}} are unsatisfiable.

Yet it can be used when $C_1 \lor \cdots \lor C_n$ 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 $D = C_1 \lor \cdots \lor C_n$ is splittable into $C_1, \ldots, C_n$.

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

\[(x \lor y)\]
\[(x \lor \bar{y})\]
\[(\bar{x} \lor y)\]
\[(\bar{x} \lor \bar{y})\]
Splitting with Backtracking

\[
\text{split} \\
(x \lor y) \\
(x \lor \bar{y}) \\
(\bar{x} \lor y) \\
(\bar{x} \lor \bar{y}) \\
(\bar{x}) | \bar{x}
\]
Splitting with Backtracking

\[(x \lor y) \quad (x \lor \overline{y})\]

subsumption

\[(\overline{x}) \mid \overline{x}\]
Splitting with Backtracking

\[
\text{split} \quad \begin{align*}
(x \lor y) \\
(x \lor \bar{y}) \\
(\bar{x}) \mid \bar{x} \\
(x) \mid x
\end{align*}
\]
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

\[(x \lor y) \]
\[(x \lor \bar{y}) \]

backtrack

\[(\bar{x}) \mid \bar{x} \]
Splitting with Backtracking

\[
\begin{align*}
\text{split} & \quad (x \lor y) \\
& \quad (x \lor \bar{y}) \\
& \quad (\bar{x}) \mid \bar{x} \\
& \quad (\bar{y})
\end{align*}
\]
Splitting with Backtracking

\[(x \lor y)\]

subsumption

\[(\bar{x}) \mid \bar{x}\]

\[(\bar{y})\]
Splitting with Backtracking

\[ (\overline{x} \vee y) \]

\[ \overline{x} \mid \overline{x} \]

\[ (x) \mid x \quad (\overline{y}) \]
Splitting with Backtracking

subsumption

\[(\bar{x}) | \bar{x} \]

\[(x) | x (\bar{y}) \]
Splitting with Backtracking

\[
\begin{array}{c}
\text{resolution} \\
\quad (\bar{x}) \mid \bar{x} \\
\quad (x) \mid x \\
\quad \emptyset \mid x, \bar{x}
\end{array}
\]
Splitting with Backtracking

\[
\begin{align*}
(x) & \mid x \\
\emptyset & \mid x, \bar{x} \\
(\bar{x}) & \mid \bar{x} \\
(\bar{y}) &
\end{align*}
\]

And so on . . .

- Too many steps (for this example);
- Backtracking is expensive;
- Generally behaves well;
- Exploits too many branches . . .
An new data-structure for rapid splitting with backtracking:

**Assertion clauses** \( D \leftarrow A \) or \( (C_1 \lor \cdots \lor C_n) \leftarrow C'_1, \ldots, C'_m \)

All inference rules can be easily converted using assertion clauses:

\[
\frac{D_1 \quad \cdots \quad D_k}{D} \\
D_1 \leftarrow A_1 \quad \cdots \quad D_k \leftarrow A_k \\
\frac{D \leftarrow \bigcup_{i=1}^{k} A_i}{D \leftarrow \bigcup_{i=1}^{k} A_i}
\]
AVATAR

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

FO

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

\[ C_1 \lor \cdots \lor C_n \lor \bar{C}_1' \lor \cdots \lor \bar{C}_m' \] (split clause)

Derives \( C_1 \lor \cdots \lor C_n \mid C_1', \ldots, C_m' \)
AVATAR

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

\[ C_1 \lor \cdots \lor C_n \lor \bar{C}_1' \lor \cdots \lor \bar{C}_m' \text{ (split clause)} \]

Derives \( C_1 \lor \cdots \lor C_n \mid C_1', \ldots, C_m' \)
A SAT solver, which treats a component as a propositional variable.

\[ C_1 \lor \cdots \lor C_n \lor \bar{C}_1' \lor \cdots \lor \bar{C}_m' \text{ (split clause)} \]

Derives \( C_1 \lor \cdots \lor C_n \mid C_1', \ldots, C_m' \)
A SAT solver, which treats a component as a propositional variable.

\[ C_1 \lor \cdots \lor C_n \lor \bar{C}'_1 \lor \cdots \lor \bar{C}'_m \] (split clause)

![Diagram of AVATAR process]

Assert \( C_1 \models C_1 \), analogue of backing if model changes
AVATAR

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

\[ \overline{C}_1' \lor \cdots \lor \overline{C}_m' \] (contradiction clause)

Derives \( \emptyset \mid C_1', \ldots, C_m' \)
A SAT solver, which treats a component as a propositional variable.

\[ \bar{C}_1' \lor \cdots \lor \bar{C}_m' \] (contradiction clause)

Derives \( \emptyset \mid C_1', \ldots, C_m' \)
AVATAR

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

\[ \bar{C}_1' \lor \cdots \lor \bar{C}_m' \text{ (contradiction clause)} \]

Derives \( \emptyset \models C'_1, \ldots, C'_m \)
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:

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

ACL2 Seminar, February 17, 2015

adaptation of a CAV’14 talk by Andrei Voronkov