Wedding Boolean Solvers with Superposition: a Societal Reform

Simon Cruanes

École polytechnique and INRIA, 23 Avenue d'Italie, 75013 Paris, France
https://who.rocq.inria.fr/Simon.Cruanes/

January 23rd, 2015

Summary

1. The Talented SAT and Superposition Solvers
2. Avatar: a Mighty Combination
3. Structural Induction
4. Conclusion

SAT solving: the Big Boolean Hammer

- SAT: boolean satisfiability
- the archetypical NP-complete problem

SAT solving: the Big Boolean Hammer

- SAT: boolean satisfiability
- the archetypical NP-complete problem
- but: good solvers exist (many breakthroughs, competition)

Chaff	(2001) first CDCL solver, 2-watch literals
Minisat	(2003) small, efficient, extensible free solver
Lingeling	
picosat	
...	

- gave rise to SMT solvers (alt-ergo, CVC4, Z3, yices...)
- encodings to SAT are good! (e.g. iProver, Satallax...)

Superposition: the King of Equality

In classical first-order theorem proving with $\simeq$, successful paradigm

- clausal calculus
  - literal: $s \simeq t$ or $s \not\simeq t$
  - clause: is a disjunction of literals $l_1 \vee \ldots \vee l_n$
  - empty clause means $\bot$
- saturation-based reasoning
  - state: set of clauses
  - *inference rules* deduce new clauses from current ones
  - new clauses are added to the set
  - $\rightarrow$ until fixpoint (**sat**) or $\bot$ (**unsat**)
  - might never terminate if problem is **sat**

Superposition: Example

Let's prove $(p \wedge a \simeq b \wedge f(a) \simeq c) \Rightarrow (p \wedge \exists x\ f(f(b)) \simeq f(x))$.
We take RPO with $p \succ f \succ a \succ b \succ c$ as ordering.

$$\dfrac{\dfrac{a \simeq b \qquad f(a) \simeq c}{f(b) \simeq c}\ \text{sup+} \qquad \dfrac{p \qquad \neg p \vee f(f(b)) \not\simeq f(x)}{f(f(b)) \not\simeq f(x)}\ \text{sup-, eq. res}}{\dfrac{f(c) \not\simeq f(x)}{\bot}\ \text{eq. res with } \{x \mapsto c\}}\ \text{sup-}$$

Superposition: Example

Let's prove $(p \land a \simeq b \land f(a) \simeq c) \Rightarrow (p \land \exists x\ f(f(b)) \simeq f(x))$.
We take RPO with $p \succ f \succ a \succ b \succ c$ as ordering.

$$\dfrac{\dfrac{a \simeq b \qquad f(a) \simeq c}{f(b) \simeq c}\ \text{sup+} \qquad \dfrac{p \qquad \neg p \lor f(f(b)) \not\simeq f(x)}{f(f(b)) \not\simeq f(x)}\ \text{sup-, eq. res}}{\dfrac{f(c) \not\simeq f(x)}{\bot}\ \text{eq. res with } \{x \mapsto c\}}\ \text{sup-}$$

unification and *ordering* are crucial

Superposition: Inference Rules

Superposition

$$\frac{C \vee s \simeq t \qquad D \vee u \dot{\simeq} v}{(C \vee D \vee u[t]_p \dot{\simeq} v)\sigma}$$

$\sigma = \text{mgu}(u|_p, s)$, ordering conditions

Equality Factoring

$$\frac{C \vee s \simeq s' \vee t \simeq t'}{(C \vee s' \not\simeq t' \vee t \simeq t')\sigma}$$

where $\sigma = \text{mgu}(s, t)$, ordering conditions

Equality Resolution

$$\frac{C \vee s \not\simeq t}{C\sigma}$$

where $\sigma = \text{mgu}(s, t)$, ordering conditions

Superposition: why it works

Superposition is **sound** and **complete** in theory.

In practice, needs many optimizations to work:

- redundancy criteria (remove trivial/useless clauses)
- simplification rules (infer + delete)
- implementation techniques (term indexing)

Need for Split

Problem with resolution/superposition: clauses *grow*.

Typically:

$$\frac{C \lor l \qquad D \lor \neg l}{C \lor D}$$

- $\rightarrow$ for non-unit clauses, conclusion has $m + n - 2$ literals
- $\rightarrow$ huge search space
- $\rightarrow$ heavy clauses (more indexing, memory, etc.)

Summary

1 The Talented SAT and Superposition Solvers

2 Avatar: a Mighty Combination

3 Structural Induction

4 Conclusion

Avatar: Split and Cut (1)

Often, clauses have independent components.

Components

- components make a partition of the clause
- no variable shared between components
- clause = boolean disjunction of its components

Avatar: Split and Cut (2)

Example

clause	$C \overset{\text{def}}{=} p(x) \lor q(y) \lor r(y, f(z)) \lor s$
components	$\begin{cases} p(x) \\ q(y) \lor r(y, f(z)) \\ s \end{cases}$
hence:	$C = (\forall x\ p(x)) \lor (\forall y\ \forall z\ q(y) \lor r(y, f(z))) \lor s$

C is actually a boolean clause!

Idea: box clauses (components) into boolean literals

Boxing

- $C \mapsto [\![C]\!]$: injection into **boolean atoms**
- $[\![C]\!]$ unique modulo alpha-renaming and AC of $\vee$
- $[\![\neg C]\!] \equiv \neg [\![C]\!]$ (if C has 1 literal)

Avatar: Split and Cut (4)

connect FO clauses and boolean atoms: the trail

Trail

- $C \leftarrow \overbrace{b_1 \sqcap b_2 \sqcap \ldots \sqcap b_n}^{\text{trail}}$
- means $(b_1 \sqcap b_2 \sqcap \ldots \sqcap b_n) \Rightarrow C$
- Theorem: $C \leftarrow [\![C]\!]$
  (proof: left to the reader)

Avatar: Split and Cut (4)

connect FO clauses and boolean atoms: the trail

Trail

- $C \leftarrow \overbrace{b_1 \sqcap b_2 \sqcap \ldots \sqcap b_n}^{\text{trail}}$
- means $(b_1 \sqcap b_2 \sqcap \ldots \sqcap b_n) \Rightarrow C$
- Theorem: $C \leftarrow [\![C]\!]$
  (proof: left to the reader)

Usual inferences inherit trails. Example:

$$\frac{C \vee l \leftarrow \Gamma_1 \qquad D \vee \neg l \leftarrow \Gamma_2}{C \vee D \leftarrow \Gamma_1 \sqcap \Gamma_2} \text{ resolution}$$

Avatar: Split and Cut (5)

Avatar keeps a set of boolean constraints $S_{\text{constraints}}$.

Avatar: Split and Cut (5)

Avatar keeps a set of boolean constraints $S_{\text{constraints}}$.

Splitting inference rule

If C has components $C_1, \ldots, C_n$ (with $n \geq 2$):

$$\frac{C_1 \vee \ldots \vee C_n \leftarrow \Gamma}{C_i \leftarrow [\![C_i]\!]} \text{split}(i),\ i \in \{1 \ldots n\}$$

Also add $\Gamma \Rightarrow_b \bigsqcup_{i=1}^{n} [\![C_i]\!]$ to $S_{\text{constraints}}$

Avatar: Split and Cut (5)

Avatar keeps a set of boolean constraints $S_{\text{constraints}}$.

Splitting inference rule

If C has components $C_1, \ldots, C_n$ (with $n \geq 2$):

$$\frac{C_1 \vee \ldots \vee C_n \leftarrow \Gamma}{C_i \leftarrow [\![C_i]\!]} \text{split}(i),\ i \in \{1 \ldots n\}$$

Also add $\Gamma \Rightarrow_b \bigsqcup_{i=1}^{n} [\![C_i]\!]$ to $S_{\text{constraints}}$

Bottom inference rule

When a clause $\bot \leftarrow b_1 \sqcap \ldots \sqcap b_n$ is found:

Add $\neg b_1 \sqcup \ldots \sqcup \neg b_n$ to $S_{\text{constraints}}$

Avatar: Split and Cut (6)

Avatar: the Proof Procedure

- regular superposition + SAT-solving on $S_{\text{constraints}}$
- **unsat** iff either one returns **unsat**
- pros:
  - for SAT problem, SAT-solver does all the work $\rightarrow$ fast
  - for ground problems, *unit-superposition*
  - superposition handles smaller clauses
  - resolution divided between (FO) prover and (SAT) solver
- also, can use current SAT interpretation to filter clauses.

Voronkov claims huge performance improvements in Vampire.

Let us re-examine the same problem:
$(p \wedge a \simeq b \wedge f(a) \simeq c) \Rightarrow (p \wedge \exists x\ f(f(b)) \simeq f(x))$.

$$\dfrac{\neg p \vee f(f(b)) \not\simeq f(x)}{\neg p \leftarrow \neg [\![p]\!]} \qquad \dfrac{\neg p \vee f(f(b)) \not\simeq f(x)}{f(f(b)) \not\simeq f(x) \leftarrow \neg [\![f(f(b)) \simeq f(x)]\!]}$$

$$\vdots \qquad\qquad \vdots$$

$$\pi_1 \qquad\qquad \pi_2$$

$$\dfrac{p \qquad \overset{\pi_1}{\vdots}}{\bot \leftarrow \neg [\![p]\!]}\ \text{sup-, eq. res} \qquad \dfrac{\dfrac{\dfrac{a \simeq b \qquad f(a) \simeq c}{f(b) \simeq c}\ \text{sup+} \qquad \overset{\pi_2}{\vdots}}{f(c) \not\simeq f(x) \leftarrow \neg [\![f(f(b)) \simeq f(x)]\!]}\ \text{sup-}}{\bot \leftarrow \neg [\![f(f(b)) \simeq f(x)]\!]}\ \text{eq. res}$$

$$S_{\text{constraints}} = \{\neg [\![p]\!] \sqcup \neg [\![f(f(b)) \simeq f(x)]\!], [\![p]\!], [\![f(f(b)) \simeq f(x)]\!]\}\ \textbf{unsat}$$

Summary

1 The Talented SAT and Superposition Solvers

2 Avatar: a Mighty Combination

3 Structural Induction

4 Conclusion

Our Goal

Poincaré: [l'induction est] «le raisonnement mathématique par excellence».

Herbrand universe calls for structural induction:

- powerful enough for data structures
- generalizes induction on naturals
- simpler than general Noetherian induction (uses subterm ordering $\triangleleft$)

Work inspired from [Kersani&Peltier, 2013].

General Principle

Refute the presence of a minimal model for a subset of all the clauses.

1. pick an *inductive constant* $\mathfrak{i}$

General Principle

Refute the presence of a minimal model for a subset of all the clauses.

1. pick an *inductive constant* $\mathfrak{i}$
2. pick a *cover set* $\kappa(\mathfrak{i})$
   e.g. for naturals, $\kappa(\mathfrak{i}) = \{0, s(\mathfrak{j})\}$ where $\mathfrak{j}$: nat is a fresh constant
   e.g. it can also be $\kappa(\mathfrak{i}) = \{0, s(0), s(s(\mathfrak{j}))\}$
   e.g. for trees, $\kappa(\mathfrak{i}) = \{\mathsf{E}, \mathsf{N}(\mathfrak{j}_1, t, \mathfrak{j}_2)\}$ with fresh t : term, $\mathfrak{j}_1, \mathfrak{j}_2$: tree

General Principle

Refute the presence of a minimal model for a subset of all the clauses.

1. pick an *inductive constant* $\mathfrak{i}$
2. pick a *cover set* $\kappa(\mathfrak{i})$
   e.g. for naturals, $\kappa(\mathfrak{i}) = \{0, s(\mathfrak{j})\}$ where $\mathfrak{j}$: nat is a fresh constant
   e.g. it can also be $\kappa(\mathfrak{i}) = \{0, s(0), s(s(\mathfrak{j}))\}$
   e.g. for trees, $\kappa(\mathfrak{i}) = \{\mathsf{E}, \mathsf{N}(\mathfrak{j}_1, t, \mathfrak{j}_2)\}$ with fresh t : term, $\mathfrak{j}_1, \mathfrak{j}_2$: tree
3. pick subset of ind. clauses, call it $S_{\text{min}}(\mathfrak{i})$

General Principle

Refute the presence of a minimal model for a subset of all the clauses.

1. pick an *inductive constant* $\mathfrak{i}$
2. pick a *cover set* $\kappa(\mathfrak{i})$
   e.g. for naturals, $\kappa(\mathfrak{i}) = \{0, s(\mathfrak{j})\}$ where $\mathfrak{j}$: nat is a fresh constant
   e.g. it can also be $\kappa(\mathfrak{i}) = \{0, s(0), s(s(\mathfrak{j}))\}$
   e.g. for trees, $\kappa(\mathfrak{i}) = \{\mathsf{E}, \mathsf{N}(\mathfrak{j}_1, t, \mathfrak{j}_2)\}$ with fresh t : term, $\mathfrak{j}_1, \mathfrak{j}_2$: tree
3. pick subset of ind. clauses, call it $S_{\text{min}}(\mathfrak{i})$
4. for every $t \in \kappa(\mathfrak{i})$
   (i) assert $\mathfrak{i} \simeq t \leftarrow [\![\mathfrak{i} \simeq t]\!]$
   (ii) assume the model is minimal for $S_{\text{min}}(\mathfrak{i})$ and $\mathfrak{i} \simeq t$
   (iii) seek contradiction

   also add $\bigoplus_{t \in \kappa(\mathfrak{i})} [\![\mathfrak{i} \simeq t]\!]$ to $S_{\text{constraints}}$ (where $\bigoplus$ is "xor")
5. no minimal model $\Rightarrow$ no model $\Rightarrow$ **unsat**

Problem with previous approach:

- consider *every* subset of clauses
- consider every $t \in \kappa(\mathfrak{i})$ for each subset

How to survive Combinatorial Explosions

Problem with previous approach:

- consider *every* subset of clauses
- consider every $t \in \kappa(\mathfrak{i})$ for each subset

$\rightarrow$ smells like combinatorial explosion!

How to survive Combinatorial Explosions

Problem with previous approach:

- consider *every* subset of clauses
- consider every $t \in \kappa(\mathfrak{i})$ for each subset

→ smells like combinatorial explosion!

→ some boolean solvers are *good* at this!

Meet QBF

Definition

A *quantified boolean formula* (or *QBF*) is defined as $Q_1 x_1 \; Q_2 x_2 \; \ldots \; Q_n x_n \; F$ where F is a boolean formula, $\{x_1, \ldots, x_n\}$ is the set of boolean variables in F, and every $Q_i \in \{\exists, \forall\}$.

- complexity: PSPACE-complete (expand **and/or** tree)
- but: benefits from advances in SAT
- SAT is QBF with $\exists$ only

Example

$\forall a \; \exists b \; \forall c \; ((a \sqcup b) \sqcap (c \sqcup \neg b))$ is a false QBF.

Clause Contexts

Problem

Induction	$\rightarrow$	second-order
clauses	$\rightarrow$	first-order

Clause Contexts

Problem

Induction	$\rightarrow$	second-order
clauses	$\rightarrow$	first-order

Solution

Use clause contexts

- a clause with a *hole* $\diamond$
- noted $C[\diamond]$
- can be *applied* to a term: $C[t] \stackrel{\text{def}}{=} [\diamond \mapsto t]C[\diamond]$

Example

- $\neg p[\diamond]$ to prove $\forall n\ p(n)$
- $n + s(\diamond) \not\simeq s(n + \diamond)$ to prove $\forall n\ \forall m\ n + s(m) \simeq s(n + m)$

Keep S_{input} and $S_{\text{min}}()$ Separate

S_{input} clauses deducible from input

- non-inductive clauses are the theory
- inductive clauses $\rightarrow$ find new contexts (heuristic)

$S_{\text{min}}()$ clauses deducible from induction hypothesis only

- induction hypothesis
- minimality assumptions
- saturate with inference rules

$\rightarrow$ do not mix them!

Keep S_{input} and $S_{\text{min}}()$ Separate (cont'd)

- Use a special *marker*, `input` , to annotate clauses from S_{input}

$\rightarrow$ remember, trails are inherited in inferences

- Redundancy criterion blocks interactions between $S_{\text{input}}$and $S_{\text{min}}()$

$$\frac{C \leftarrow \texttt{input}, [\![D[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!], \Gamma}{\top}$$

Example

- $(\neg p(n) \leftarrow \texttt{input}) \in S_{\text{input}}$ (provable)
- $(\neg p(n) \leftarrow [\![\neg p[\diamond] \in S_{\text{min}}(n)]\!]) \in S_{\text{min}}()$ (ind. hypothesis)

Express Induction Hypothesis

For inductive constant $\mathfrak{i}$, set of all contexts is $S_{\text{cand}}(\mathfrak{i})$

- $C[\mathfrak{i}] \leftarrow \overbrace{[\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!]}^{\text{in the subset?}} \sqcap \overbrace{[\![\text{init}(C[\diamond], \mathfrak{i})]\!]}^{\text{provable from } S_{\text{input}}?}$
- boolean valuation of atoms $[\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!]$ determine subset $S_{\text{min}}(\mathfrak{i})$
- $[\![\text{init}(C[\diamond], \mathfrak{i})]\!]$ added to QBF when $C[\mathfrak{i}]$ subsumed by some clause

Express Minimality

- model minimal for $S_{\text{min}}(\mathfrak{i}) \neq \emptyset$
  $\Rightarrow \exists C[\diamond] \in S_{\text{min}}(\mathfrak{i})$ with $\begin{cases} C[\mathfrak{i}] \text{ true} \\ C[\diamond] \text{ false on a smaller term } \mathfrak{j} \end{cases}$

Express Minimality

- model minimal for $S_{\text{min}}(\mathfrak{i}) \neq \emptyset$
  $\Rightarrow \exists C[\diamond] \in S_{\text{min}}(\mathfrak{i})$ with $\begin{cases} C[\mathfrak{i}] \text{ true} \\ C[\diamond] \text{ false on a smaller term } \mathfrak{j} \end{cases}$
- "winning" witness $C[\mathfrak{j}]$ annotated with $[\![\text{minimal}(C[\diamond], \mathfrak{i}, \mathfrak{j})]\!]$
- for each $C[\diamond]$ and $\mathfrak{j} \lhd t \in \kappa_{\downarrow}(\mathfrak{i})$ (subterm of inductive type):
  $\neg C[\mathfrak{j}] \leftarrow [\![\text{minimal}(C[\diamond], \mathfrak{i}, \mathfrak{j})]\!] \sqcap [\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!] \sqcap [\![\mathfrak{i} = t]\!]$
  (then reduced to CNF)

$\rightarrow$ **means**: "$C[\diamond]$ is the context for which $S_{\text{min}}(\mathfrak{i})$ is minimal"

$\rightarrow$ **means**: "$C[\diamond]$ is false on $\mathfrak{j}$, because it's smaller than $\mathfrak{i}$"

Express Minimality

- model minimal for $S_{\text{min}}(\mathfrak{i}) \neq \emptyset$
  $\Rightarrow \exists C[\diamond] \in S_{\text{min}}(\mathfrak{i})$ with $\begin{cases} C[\mathfrak{i}] \text{ true} \\ C[\diamond] \text{ false on a smaller term } \mathfrak{j} \end{cases}$
- "winning" witness $C[\mathfrak{j}]$ annotated with $[\![\text{minimal}(C[\diamond], \mathfrak{i}, \mathfrak{j})]\!]$
- for each $C[\diamond]$ and $\mathfrak{j} \triangleleft t \in \kappa_{\downarrow}(\mathfrak{i})$ (subterm of inductive type):
  $\neg C[\mathfrak{j}] \leftarrow [\![\text{minimal}(C[\diamond], \mathfrak{i}, \mathfrak{j})]\!] \sqcap [\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!] \sqcap [\![\mathfrak{i} = t]\!]$
  (then reduced to CNF)

$\rightarrow$ **means**: "$C[\diamond]$ is the context for which $S_{\text{min}}(\mathfrak{i})$ is minimal"

$\rightarrow$ **means**: "$C[\diamond]$ is false on $\mathfrak{j}$, because it's smaller than $\mathfrak{i}$"

- in QBF, disjunction that forces the *choice* of $C[\diamond]$ in $S_{\text{min}}(\mathfrak{i})$

What about QBF then?

QBF is needed to *enumerate* the characteristic function for $S_{\mathrm{min}}(\mathfrak{i})$ (ranges in $2^{S_{\mathrm{cand}}(\mathfrak{i})}$)

What about QBF then?

QBF is needed to *enumerate* the characteristic function for $S_{\text{min}}(\mathfrak{i})$ (ranges in $2^{S_{\text{cand}}(\mathfrak{i})}$)

Formula

$$
\begin{aligned}
F \stackrel{\text{def}}{=} \; & \exists_{a \in S_{\text{atoms}}} a \\
& \forall_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!] \\
& \exists_{t \in \kappa(\mathfrak{i})} [\![\mathfrak{i} = t]\!] \\
& \exists_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![\text{init}(C[\diamond], \mathfrak{i})]\!] \\
& \exists_{\mathfrak{j} \lhd t \in \kappa_{\downarrow}(\mathfrak{i}), C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![\text{minimal}(C[\diamond], \mathfrak{i}, \mathfrak{j})]\!] \\
& \left(\bigsqcap_{x \in S_{\text{constraints}}} x \right) \sqcap \left(\text{empty} \sqcup \bigsqcup_{t \in \kappa(\mathfrak{i})} \left\{ \begin{array}{l} [\![\mathfrak{i} = t]\!] \sqcap \\ \text{minimal}(t) \end{array} \right\} \right)
\end{aligned}
$$

What about QBF then?

QBF is needed to *enumerate* the characteristic function for $S_{\text{min}}(\mathfrak{i})$ (ranges in $2^{S_{\text{cand}}(\mathfrak{i})}$)

Formula

$$
\begin{aligned}
F \stackrel{\text{def}}{=}\; & \exists_{a \in S_{\text{atoms}}} a \\
& \forall_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!] \\
& \exists_{t \in \kappa(\mathfrak{i})} [\![\mathfrak{i} = t]\!] \\
& \exists_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![\text{init}(C[\diamond], \mathfrak{i})]\!] \\
& \exists_{\mathfrak{j} \lhd t \in \kappa_{\downarrow}(\mathfrak{i}), C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![\text{minimal}(C[\diamond], \mathfrak{i}, \mathfrak{j})]\!] \\
& \left(\bigsqcap_{x \in S_{\text{constraints}}} x \right) \sqcap \left(\text{empty} \sqcup \bigsqcup_{t \in \kappa(\mathfrak{i})} \left\{ \begin{array}{l} [\![\mathfrak{i} = t]\!] \sqcap \\ \text{minimal}(t) \end{array} \right\} \right) \\
\text{empty} \stackrel{\text{def}}{=}\; & \bigsqcap_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} \neg [\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!]
\end{aligned}
$$

What about QBF then?

QBF is needed to *enumerate* the characteristic function for $S_{\text{min}}(\mathfrak{i})$ (ranges in $2^{S_{\text{cand}}(\mathfrak{i})}$)

Formula

$$
\begin{aligned}
F &\stackrel{\text{def}}{=} \exists_{a \in S_{\text{atoms}}} a \\
&\quad \forall_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!] \\
&\quad \exists_{t \in \kappa(\mathfrak{i})} [\![\mathfrak{i} = t]\!] \\
&\quad \exists_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![\text{init}(C[\diamond], \mathfrak{i})]\!] \\
&\quad \exists_{\mathfrak{j} \lhd t \in \kappa_{\downarrow}(\mathfrak{i}), C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} [\![\text{minimal}(C[\diamond], \mathfrak{i}, \mathfrak{j})]\!] \\
&\quad \left(\bigsqcap_{x \in S_{\text{constraints}}} x \right) \sqcap \left(\text{empty} \sqcup \bigsqcup_{t \in \kappa(\mathfrak{i})} \left\{ \begin{array}{l} [\![\mathfrak{i} = t]\!] \sqcap \\ \text{minimal}(t) \end{array} \right\} \right) \\
\text{empty} &\stackrel{\text{def}}{=} \bigsqcap_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} \neg [\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!] \\
\text{minimal}(t) &\stackrel{\text{def}}{=} \bigsqcap_{\mathfrak{j} \lhd t} \bigsqcup_{C[\diamond] \in S_{\text{cand}}(\mathfrak{i})} \left(\begin{array}{l} [\![C[\diamond] \in S_{\text{min}}(\mathfrak{i})]\!] \sqcap \\ [\![\text{minimal}(C[\diamond], \mathfrak{i}, \mathfrak{j})]\!] \end{array} \right)
\end{aligned}
$$

Summary

1. The Talented SAT and Superposition Solvers
2. Avatar: a Mighty Combination
3. Structural Induction
4. Conclusion

Conclusion

- Avatar brings Superposition within splitting distance of SAT solvers

Conclusion

- Avatar brings Superposition within splitting distance of SAT solvers
- can be extended for structural induction, using QBF solvers
  (NP-complete not funny enough, let's use PSPACE-complete. . .)
- induction is **not** easy!

Conclusion

- Avatar brings Superposition within splitting distance of SAT solvers
- can be extended for structural induction, using QBF solvers
  (NP-complete not funny enough, let's use PSPACE-complete...)
- induction is **not** easy!
- prototype and paper: work in progress

Conclusion

- Avatar brings Superposition within splitting distance of SAT solvers
- can be extended for structural induction, using QBF solvers
  (NP-complete not funny enough, let's use PSPACE-complete. . .)
- induction is **not** easy!
- prototype and paper: work in progress

Questions?

Example

n^o	clause	constraint	source
1	$p(0, a)$		axiom
2	$\neg p(x, y) \vee p(s(x), f(y))$		axiom
3	$\neg p(\mathfrak{n}, x) \leftarrow \texttt{input}$		axiom
4	$\mathfrak{n} \simeq 0 \leftarrow [\![\mathfrak{n} \simeq 0]\!]$	$\begin{cases} [\![\mathfrak{n} \simeq 0]\!] \sqcup \\ [\![\mathfrak{n} \simeq s(\mathfrak{n}')]\!] \end{cases}$	split
5	$\mathfrak{n} \simeq s(\mathfrak{n}') \leftarrow [\![\mathfrak{n} \simeq s(\mathfrak{n}')]\!]$		split
6	$\bot \leftarrow \texttt{input} \sqcap [\![\mathfrak{n} \simeq 0]\!]$	$\neg [\![\mathfrak{n} \simeq 0]\!]$	sup (1,4)
7	$\neg p(s(\mathfrak{n}'), x) \leftarrow \begin{cases} [\![\mathfrak{n} \simeq s(\mathfrak{n}')]\!] \sqcap \\ [\![\mathrm{init}(C[\diamond], \mathfrak{n})]\!] \sqcap \\ [\![C[\diamond] \in S_{\mathrm{min}}(\mathfrak{n})]\!] \end{cases}$		hypothesis
8	$p(\mathfrak{n}', b) \leftarrow \begin{cases} [\![\mathfrak{n} \simeq s(\mathfrak{n}')]\!] \sqcap \\ [\![\mathrm{minimal}(C[\diamond], \mathfrak{n}, \mathfrak{n}')]\!] \sqcap \\ [\![C[\diamond] \in S_{\mathrm{min}}(\mathfrak{n})]\!] \end{cases}$		hypothesis

Example (cont'd)

n^o	clause	constraint
9	$p(s(\mathfrak{j}), f(b)) \leftarrow \begin{array}{c} [\![\mathfrak{n} \simeq s(\mathfrak{n}')]\!] \sqcap \\ [\![\mathsf{minimal}(C[\diamond], \mathfrak{n}, \mathfrak{n}')]\!] \sqcap \\ [\![C[\diamond] \in S_{\mathsf{min}}(\mathfrak{n})]\!] \end{array}$	
10	$\bot \leftarrow \begin{cases} [\![\mathfrak{n} \simeq s(\mathfrak{n}')]\!] \sqcap \\ [\![\mathsf{minimal}(C[\diamond], \mathfrak{n}, \mathfrak{n}')]\!] \sqcap \\ [\![\mathsf{init}(C[\diamond], \mathfrak{n})]\!] \sqcap \\ [\![C[\diamond] \in S_{\mathsf{min}}(\mathfrak{n})]\!] \end{cases}$	$\begin{array}{l} \neg[\![\mathfrak{n} \simeq s(\mathfrak{n}')]\!] \sqcup \\ \neg[\![\mathsf{minimal}(C[\diamond], \mathfrak{n}, \mathfrak{n}')]\!] \sqcup \\ \neg[\![\mathsf{init}(C[\diamond], \mathfrak{n})]\!] \sqcup \\ \neg[\![C[\diamond] \in S_{\mathsf{min}}(\mathfrak{n})]\!] \end{array}$