Superposition $\uplus$ Structural Induction

Simon Cruanes

Veridis, Inria Nancy
https://cedeela.fr/~simon/

17th of November, 2016

The Third Workshop on Automated Inductive Theorem-Proving, Vienna

Summary

Outline

This talk: mixing Superposition and Induction

- **Superposition**: state of the art for first-order classical reasoning (implemented in the best FO provers: E, Vampire, SPASS, . . .)
- **Induction**: cornerstone of many proof assistants, critical to reason about infinite structures

  (here, structural induction on datatypes)
- **Goal**: add inductive reasoning to first-order provers for inductions that are not too hard (possibly helped with lemmas).

  NOT about making the best inductive prover ever!

$\rightarrow$ mix of first-order and induction (and theories. . .) useful for, e.g., Sledgehammer, Why3.

Superposition in a Nutshell

the Superposition calculus:

- clausal (works on disjunctions of literals)
- refutational (goal: deduce $\bot$)
- equational (tailored for reasoning with equality)

Say we have only two elements a and b, on which p holds. Then prove $\forall x.\ p(x)$ by refuting $\exists c. \neg p(c)$:

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{\neg p(c) \qquad x \simeq a \lor x \simeq b}{\neg p(a) \lor c \simeq b}\,(\text{Sup})
      \qquad p(a)
    }{c \simeq b}\,(\text{Res})
    \qquad \neg p(c)
  }{\neg p(b)}\,(\text{Sup})
  \qquad p(b)
}{\bot}\,(\text{Res})
$$

(Note the *binding* of x to c using *unification*)

Superposition in a Nutshell

the Superposition calculus:

- clausal (works on disjunctions of literals)
- refutational (goal: deduce $\bot$)
- equational (tailored for reasoning with equality)

Say we have only two elements a and b, on which p holds. Then prove $\forall x.\ p(x)$ by refuting $\exists c. \neg p(c)$:

$$\frac{\neg p(c) \qquad x \simeq a \lor x \simeq b}{\neg p(a) \lor c \simeq b}\ (\text{Sup})$$

$$\frac{\neg p(a) \lor c \simeq b \qquad p(a)}{c \simeq b}\ (\text{Res})$$

$$(\text{Sup})\ \frac{c \simeq b \qquad \neg p(c)}{\neg p(b)}$$

$$(\text{Res})\ \frac{\neg p(b) \qquad p(b)}{\bot}$$

(Note the *binding* of x to c using *unification*)

Rules of Superposition

Superposition:
$$\frac{C \vee s \simeq t \qquad D \vee u\left[s_2\right]_p \dot{\simeq} v}{(C \vee D \vee u\left[t\right]_p \dot{\simeq} v)\sigma} \text{(Sup)}$$

where $s\sigma = s_2\sigma$, $\dot{\simeq} \in \{\simeq, \not\simeq\}$, $s\sigma \succ t\sigma$, $u\sigma \succ v\sigma$, $[\ldots]$

Equality Resolution:
$$\frac{C \vee s \not\simeq t}{C\sigma} \text{(EqRes)}$$

where $s\sigma = t\sigma$, $[\ldots]$

- σ is a substitution
- C, D are clauses (disjunctions of atoms)
- $u\,[t]_p$ puts t at position p in term u
- $\succ$ is on ordering on terms

Beyond Superposition: Clause Splitting With AVATAR

Induction will require case analysis.
However, Superposition not very good with boolean reasoning...
$\rightarrow$ use the AVATAR extension [Voronkov 2014] (here, modified a bit)

- delegate (some) reasoning to a SAT solver
- $\llbracket \cdot \rrbracket$: injective mapping to boolean literals
- $C \leftarrow \bigwedge_i b_i$ means clause C holds if lits b_i satisfied
- use atoms $\llbracket \mathfrak{n}_0 \simeq 0 \cdot \mathfrak{l}_0 \simeq \mathfrak{n}_0 :: \mathfrak{l}_1 \rrbracket$ to "select" branch in inductive proof

Example

Typical case analysis for induction:

$$\frac{\mathfrak{n}_0 \simeq 0 \vee \mathfrak{n}_0 \simeq s(\mathfrak{n}_1)}{\begin{array}{l} \mathfrak{n}_0 \simeq 0 \leftarrow \llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \\ \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \leftarrow \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket \\ \llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \vee \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket \end{array}} \text{(ASplit)}$$

Beyond Superposition: Clause Splitting With AVATAR

Induction will require case analysis.
However, Superposition not very good with boolean reasoning...
$\rightarrow$ use the AVATAR extension [Voronkov 2014] (here, modified a bit)

- delegate (some) reasoning to a SAT solver
- $\llbracket \cdot \rrbracket$: injective mapping to boolean literals
- $C \leftarrow \bigwedge_i b_i$ means clause C holds if lits b_i satisfied
- use atoms $\llbracket \mathfrak{n}_0 \simeq 0 \cdot \mathfrak{l}_0 \simeq \mathfrak{n}_0 :: \mathfrak{l}_1 \rrbracket$ to "select" branch in inductive proof

Example

Typical case analysis for induction:

$$\frac{\mathfrak{n}_0 \simeq 0 \vee \mathfrak{n}_0 \simeq s(\mathfrak{n}_1)}{\begin{array}{l} \mathfrak{n}_0 \simeq 0 \leftarrow \llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \\ \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \leftarrow \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket \\ \llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \vee \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket \end{array}} \text{(ASplit)}$$

Summary

Extended Logic

For inductive proving, we use an extended logic:

- FO + polymorphic types ($\sim$ TFF1)
- inductive datatypes
- recursive functions or rewriting rules
  (terminating confluent system)

Roughly corresponds to (an encoding of) TIP

TIP ("Tons of Inductive Problems")

- derivative/extension of SMT-LIB 2
- on github: https://tip-org.github.io/
- Dan Rosén, Nick Smallbone, Moa Johansson, Koen Claessen

Extended Logic

For inductive proving, we use an extended logic:

- FO + polymorphic types ($\sim$ TFF1)
- inductive datatypes
- recursive functions or rewriting rules
  (terminating confluent system)

Roughly corresponds to (an encoding of) TIP

TIP ("Tons of Inductive Problems")

- derivative/extension of SMT-LIB 2
- on github: `https://tip-org.github.io/`
- Dan Rosén, Nick Smallbone, Moa Johansson, Koen Claessen

Inductive Proof by Refutation

Example (Induction on Lists)

- assume $p([\,])$ and $\forall x\ l.\ p(l) \Rightarrow p(x :: l)$
- Prove p holds for all list
- by refutation:
  - assume $\exists \mathfrak{l}_0 : \text{list}.\ \neg p(\mathfrak{l}_0)$
  - coverset: $\mathfrak{l}_0 \in \{[\,], \mathfrak{t}_0 :: \mathfrak{l}_1\}$

$\rightarrow$ assert $(\mathfrak{l}_0 \simeq [\,]) \vee (\mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1)$ and deduce $\bot$ by case analysis

Inductive Proof by Refutation

Example (Induction on Lists)

- assume $p([\,])$ and $\forall x\ l.\ p(l) \Rightarrow p(x :: l)$
- Prove p holds for all list
- by refutation:
  - assume $\exists \mathfrak{l}_0 : \text{list}.\ \neg p(\mathfrak{l}_0)$
  - coverset: $\mathfrak{l}_0 \in \{[\,], \mathfrak{t}_0 :: \mathfrak{l}_1\}$

  $\rightarrow$ assert $(\mathfrak{l}_0 \simeq [\,]) \vee (\mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1)$ and deduce $\bot$ by case analysis

split (with AVATAR):

$$\frac{\mathfrak{l}_0 \simeq [\,] \vee \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1}{\mathfrak{l}_0 \simeq [\,] \leftarrow \llbracket \mathfrak{l}_0 \simeq [\,] \rrbracket \qquad \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket \qquad \llbracket \mathfrak{l}_0 \simeq [\,] \rrbracket \sqcup \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket}$$

Inductive Proof by Refutation

Example (Induction on Lists)

- assume $p([])$ and $\forall x\ l.\ p(l) \Rightarrow p(x :: l)$
- Prove p holds for all list
- by refutation:
  - assume $\exists \mathfrak{l}_0 : \text{list}.\ \neg p(\mathfrak{l}_0)$
  - coverset: $\mathfrak{l}_0 \in \{[], \mathfrak{t}_0 :: \mathfrak{l}_1\}$

  $\rightarrow$ assert $(\mathfrak{l}_0 \simeq []) \vee (\mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1)$ and deduce $\perp$ by case analysis

base case: easy

$$\dfrac{\dfrac{\dfrac{\boxed{\mathfrak{l}_0} \simeq [] \leftarrow [\![\mathfrak{l}_0 \simeq []]\!] \qquad \neg p(\boxed{\mathfrak{l}_0})}{\neg \boxed{p([])} \leftarrow [\![\mathfrak{l}_0 \simeq []]\!]}\ (\text{Sup}) \qquad \boxed{p([])}}{\perp \leftarrow [\![\mathfrak{l}_0 \simeq []]\!]}\ (\text{Res})}{\boxed{\neg [\![\mathfrak{l}_0 \simeq []]\!]}}\ (\text{A}\perp)$$

Inductive Proof by Refutation

Example (Induction on Lists)

- assume $p([])$ and $\forall x\ l.\ p(l) \Rightarrow p(x :: l)$
- Prove p holds for all list
- by refutation:
  - assume $\exists \mathfrak{l}_0 : \text{list}.\ \neg p(\mathfrak{l}_0)$
  - coverset: $\mathfrak{l}_0 \in \{[], \mathfrak{t}_0 :: \mathfrak{l}_1\}$

$\rightarrow$ assert $(\mathfrak{l}_0 \simeq []) \vee (\mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1)$ and deduce $\bot$ by case analysis

recursive case:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket \quad \neg p(\mathfrak{l}_0)}{\neg p(\mathfrak{t}_0 :: \mathfrak{l}_1) \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket}\,(\text{Sup}) \quad \neg p(l) \vee p(x :: l)}{\neg p(\mathfrak{l}_1) \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket}\,(\text{Res}) \quad p(\mathfrak{l}_1)}{\bot \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket}\,(\text{Res})}{\neg \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket}\,(\text{A}\bot)$$

Inductive Strengthening

Success, both $\boxed{\llbracket \mathfrak{l}_0 \simeq [\,] \rrbracket}$ and $\boxed{\llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket}$ are false!

$\rightarrow$ we used *inductive strengthening* to prove the recursive case.

Principle

- assume $\mathfrak{l}_0$ is a minimal counter-example to p

  (minimal w.r.t. subterm ordering $\lhd$)

  in other words: $\neg p(\mathfrak{l}_0)$ and $\forall x.\ x \lhd \mathfrak{l}_0 \Rightarrow p(x)$
- assert $p(\mathfrak{l}_1)$, since $\mathfrak{l}_1 \lhd \mathfrak{l}_0$ ($\simeq \mathfrak{t}_0 :: \mathfrak{l}_1$) and $\mathfrak{l}_0$ minimal
- theorem: $\exists$ model iff $\exists$ model with $\llbracket \mathfrak{l}_0 \rrbracket$ minimal
- Also works for nested induction (minimal tuple)

Success, both $\lfloor\!\lfloor \mathfrak{l}_0 \simeq [\,] \rfloor\!\rfloor$ and $\lfloor\!\lfloor \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rfloor\!\rfloor$ are false!

$\rightarrow$ we used *inductive strengthening* to prove the recursive case.

Principle

- assume $\mathfrak{l}_0$ is a minimal counter-example to p

  (minimal w.r.t. subterm ordering $\lhd$)

  in other words: $\neg p(\mathfrak{l}_0)$ and $\forall x.\ x \lhd \mathfrak{l}_0 \Rightarrow p(x)$
- assert $p(\mathfrak{l}_1)$, since $\mathfrak{l}_1 \lhd \mathfrak{l}_0$ $(\simeq \mathfrak{t}_0 :: \mathfrak{l}_1)$ and $\mathfrak{l}_0$ minimal
- theorem: $\exists$ model iff $\exists$ model with $[\![\mathfrak{l}_0]\!]$ minimal
- Also works for nested induction (minimal tuple)

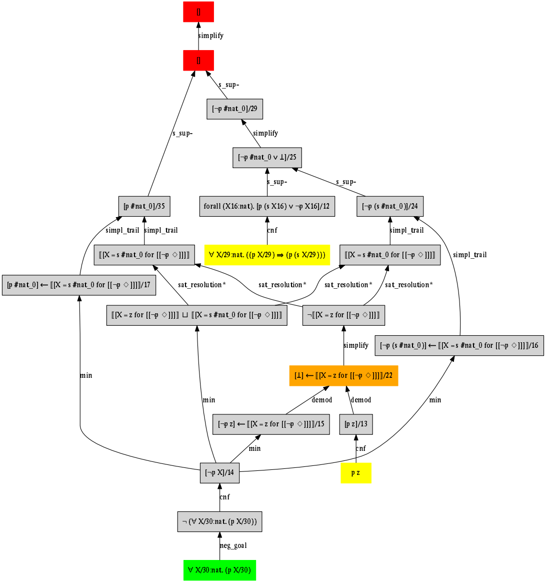
Summary

Implementation

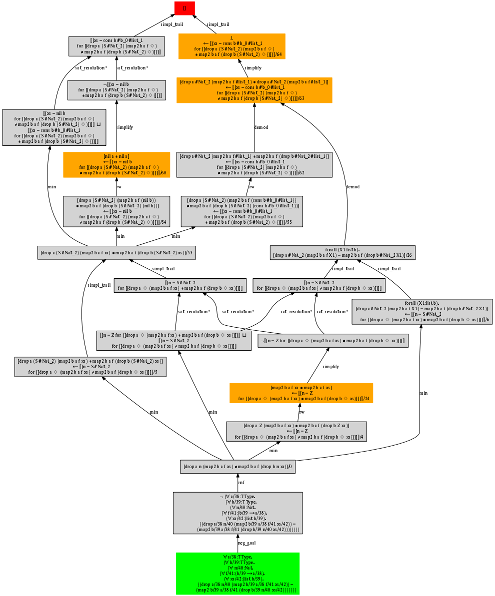
Zipperposition, a Superposition prover in OCaml

- not very good, but flexible and feature full
- follows the design of E, + typing, int arith, induction, and rewriting
- OCaml is expressive, reasonably fast, and safe
- BSD license, https://github.com/c-cube/zipperposition
- **demo:** a few proof graphs! (using graphviz)

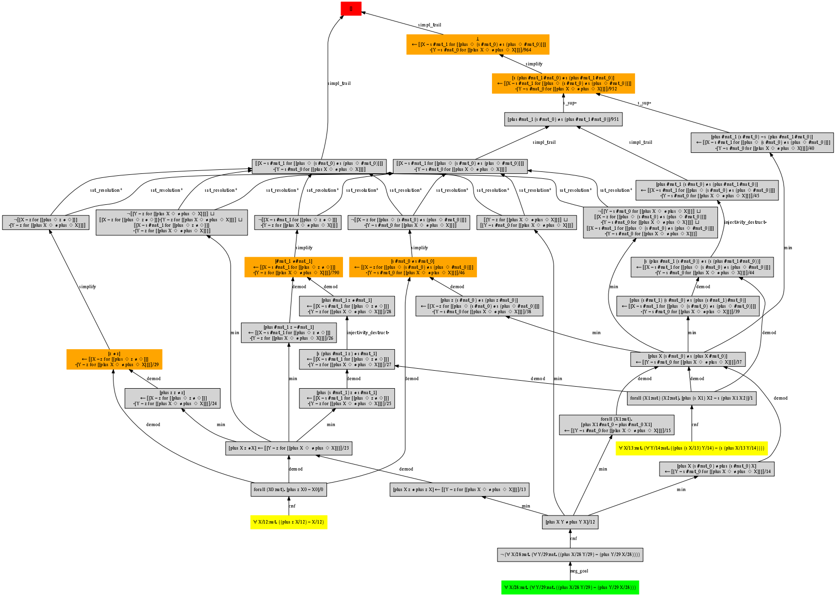
Trivial: $(p(0) \wedge \forall x.\ p(x) \Rightarrow p(s(x))) \Rightarrow \forall x.\ p(x)$



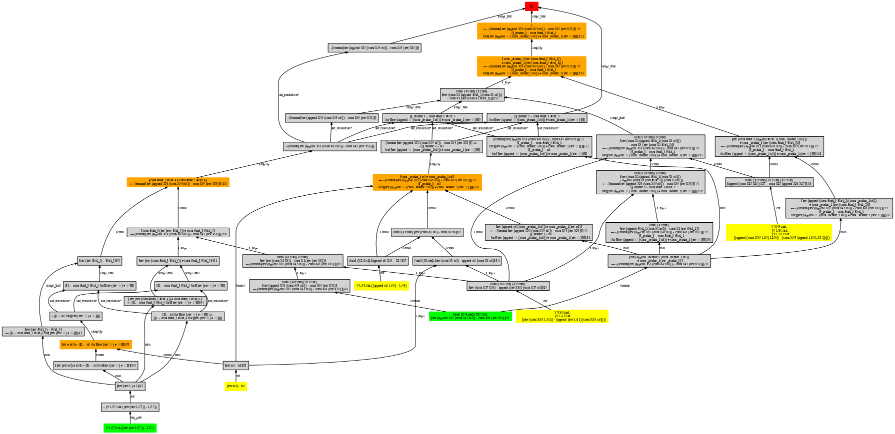
Isaplanner 12: $\mathrm{map}(f, \mathrm{drop}(n, l)) \simeq \mathrm{drop}(n, \mathrm{map}(f, l))$



$a + b \simeq b + a$



rev(rev(l)) = l (using a lemma)



Lemma in AVATAR

Inference Rule

introduce lemma F:

$$\frac{\top}{\begin{array}{rll} & F & \leftarrow \llbracket F \rrbracket \\ \land & \neg F & \leftarrow \neg \llbracket F \rrbracket \end{array}}$$

Lemma in AVATAR

Inference Rule

introduce lemma F (and reduce it to CNF) :

$$\frac{\top}{\begin{array}{rll} & \mathrm{cnf}(F) & \leftarrow [\![F]\!] \\ \wedge & \mathrm{cnf}(\neg F) & \leftarrow \neg [\![F]\!] \end{array}}$$

Typically, $\mathrm{cnf}(\neg F)$ will be refuted in a separate induction.

What is missing

Functional Induction

- we have function definitions (in TIP)
- those generate cover sets
- could use those cover sets for goals $\forall x_1, \ldots, x_n.\ P[f(x_1, \ldots, x_n)]$

Lemma Divination

- guessing lemmas is critical (not cut-free!)
- possibility: à la HipSpec
- possibility: generalize from "stuck" negative clauses (i.e., goals)
- haven't had the time to make it work yet.

Also, better handling of datatypes, etc.

What is missing

Functional Induction

- we have function definitions (in TIP)
- those generate cover sets
- could use those cover sets for goals $\forall x_1, \ldots, x_n.\ P[f(x_1, \ldots, x_n)]$

Lemma Divination

- guessing lemmas is critical (not cut-free!)
- possibility: à la HipSpec
- possibility: generalize from "stuck" negative clauses (i.e., goals)
- haven't had the time to make it work yet.

Also, better handling of datatypes, etc.

Conclusion

It is possible to extend a Superposition based prover to handle (structural) Induction.

- The point is to avoid losing all the progress in FO ATP in order to get induction!
- AVATAR is recommended, to handle case splitting
  - $\rightarrow$ can pursue many simultaneous proofs at the same time
  - $\rightarrow$ nested induction is no problem
- the tricky parts:
  - handle datatypes (maybe take inspiration from FOOL)
  - definitions: need rewriting OR good term orderings
  - divination of lemmas (as always)
- code on github (but beware of bugs, highly experimental)

Use QBF to quantify over subsets of all inductive clauses.
Constraint for constant $\mathfrak{i}$:

$$\begin{array}{rcl}
F_{\mathfrak{i}} & \stackrel{\text{def}}{=} & \exists_{a \in S_{\text{atoms}}} a \\
& & \forall_{C \in S_{\text{cand}}(\mathfrak{i})} [\![C \in S_{\min}(\mathfrak{i})]\!] \\
& & \exists_{t \in \mathfrak{i}} [\![\mathfrak{i} \simeq t]\!] \\
& & \exists_{C \in S_{\text{cand}}(\mathfrak{i})} [\![\text{init}(C, \mathfrak{i})]\!] \\
& & \exists_{t', t' \mathfrak{i}, C \in S_{\text{cand}}(\mathfrak{i})} [\![\text{minimal}(C, \mathfrak{i}, t')]\!] \\
& & \left(\bigsqcap_{x \in S_{\text{constraints}}} x \right) \sqcap \left(\text{empty} \sqcup \bigsqcup_{t \in \mathfrak{i}} [\![\mathfrak{i} \simeq t]\!] \sqcap \text{minimal}(t) \right) \\
\text{empty} & \stackrel{\text{def}}{=} & \bigsqcap_{C \in S_{\text{cand}}(\mathfrak{i})} \neg [\![C \in S_{\min}(\mathfrak{i})]\!] \\
\text{minimal}(t) & \stackrel{\text{def}}{=} & \bigsqcap_{t' \lhd t, t' \mathfrak{i}} \bigsqcup_{C \in S_{\text{cand}}(\mathfrak{i})} \left(\begin{array}{l} [\![C \in S_{\min}(\mathfrak{i})]\!] \sqcap \\ [\![\text{minimal}(C, \mathfrak{i}, t')]\!] \end{array} \right)
\end{array}$$

Splitting Clauses in AVATAR

Boxing Operation ($\sim$ Tseitin definitions)

First, we define boxing: $\llbracket\cdot\rrbracket$ (to be used on clause components)

- just *give a name* to a clause/formula
- for any x, $\llbracket x\rrbracket$ is a boolean literal
- $\llbracket\neg l\rrbracket = \neg\llbracket l\rrbracket$ if l ground atomic formula
- $\llbracket\forall x.\ F[x]\rrbracket = \llbracket\forall y.\ F[y]\rrbracket$

Example

clause	propositional clause (boxing)
$p \vee \neg q \vee \forall x.\ p(x)$	$\llbracket p\rrbracket \sqcup \neg\llbracket q\rrbracket \sqcup \llbracket p(x)\rrbracket$
$\forall x.\ \neg p(x) \vee \forall y\ z.\ q(y) \vee q(f(y,z))$	$\llbracket\neg p(x)\rrbracket \sqcup \llbracket q(y) \vee q(f(y,z))\rrbracket$
$\mathfrak{n}_0 \simeq 0 \vee \mathfrak{n}_0 \simeq s(\mathfrak{n}_1)$	$\llbracket\mathfrak{n}_0 \simeq 0\rrbracket \sqcup \llbracket\mathfrak{n}_0 \simeq s(\mathfrak{n}_1)\rrbracket$

Splitting Clauses in AVATAR

Boxing Operation ($\sim$ Tseitin definitions)

First, we define boxing: $\llbracket \cdot \rrbracket$ (to be used on clause components)

- just *give a name* to a clause/formula
- for any x, $\llbracket x \rrbracket$ is a boolean literal
- $\llbracket \neg l \rrbracket = \neg \llbracket l \rrbracket$ if l ground atomic formula
- $\llbracket \forall x.\ F[x] \rrbracket = \llbracket \forall y.\ F[y] \rrbracket$

Example

clause	propositional clause (boxing)
$p \vee \neg q \vee \forall x.\ p(x)$	$\llbracket p \rrbracket \sqcup \neg \llbracket q \rrbracket \sqcup \llbracket p(x) \rrbracket$
$\forall x.\ \neg p(x) \vee \forall y\ z.\ q(y) \vee q(f(y,z))$	$\llbracket \neg p(x) \rrbracket \sqcup \llbracket q(y) \vee q(f(y,z)) \rrbracket$
$\mathfrak{n}_0 \simeq 0 \vee \mathfrak{n}_0 \simeq s(\mathfrak{n}_1)$	$\llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \sqcup \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket$

Rules of AVATAR

A-clause

An **A-clause** is $C \leftarrow \Gamma$ where

- C is a clause (disjunction of literals)
- $\Gamma = \bigsqcap_{i=1}^{n} b_i$ with b_i boxes (propositional literals)

AVATAR Split

$$\frac{C_1 \vee \ldots \vee C_n \leftarrow \Gamma}{\bigwedge_{i=1}^{n} (C_i \leftarrow \llbracket C_i \rrbracket) \qquad \Gamma \Rightarrow (\bigsqcup_{i=1}^{n} \llbracket C_i \rrbracket)} \text{(ASplit)}$$

if $i \neq j \Rightarrow \text{vars}(C_i) \cap \text{vars}(C_j) = \emptyset$

AVATAR Absurd

$$\frac{\bot \leftarrow \bigsqcap_{i=1}^{n} b_i}{\bigsqcup_{i=1}^{n} \neg b_i} (\text{A}\bot)$$

- deduce clauses
- force ≥ 1 clause to be true (if Γ is)
- prune absurd branches

Rules of AVATAR

A-clause

An **A-clause** is $C \leftarrow \Gamma$ where

- C is a clause (disjunction of literals)
- $\Gamma = \prod_{i=1}^{n} b_i$ with b_i boxes (propositional literals)

AVATAR Split

$$\frac{C_1 \vee \ldots \vee C_n \leftarrow \Gamma}{\bigwedge_{i=1}^{n} (C_i \leftarrow \llbracket C_i \rrbracket) \qquad \Gamma \Rightarrow (\bigsqcup_{i=1}^{n} \llbracket C_i \rrbracket)} \text{ (ASplit)}$$

$$\text{if } i \neq j \Rightarrow \mathrm{vars}(C_i) \cap \mathrm{vars}(C_j) = \emptyset$$

AVATAR Absurd

$$\frac{\bot \leftarrow \prod_{i=1}^{n} b_i}{\bigsqcup_{i=1}^{n} \neg b_i} \text{ (A}\bot\text{)}$$

- deduce clauses
- force ≥ 1 clause to be true (if Γ is)
- prune absurd branches

Rules of AVATAR

A-clause

An **A-clause** is $C \leftarrow \Gamma$ where

- C is a clause (disjunction of literals)
- $\Gamma = \bigsqcap_{i=1}^{n} b_i$ with b_i boxes (propositional literals)

AVATAR Split

$$\frac{C_1 \vee \ldots \vee C_n \leftarrow \Gamma}{\boxed{\bigwedge_{i=1}^{n} (C_i \leftarrow \llbracket C_i \rrbracket)} \qquad \Gamma \Rightarrow (\bigsqcup_{i=1}^{n} \llbracket C_i \rrbracket)} \text{(ASplit)}$$

$$\text{if } i \neq j \Rightarrow \text{vars}(C_i) \cap \text{vars}(C_j) = \emptyset$$

AVATAR Absurd

$$\frac{\bot \leftarrow \bigsqcap_{i=1}^{n} b_i}{\bigsqcup_{i=1}^{n} \neg b_i} (\text{A}\bot)$$

- deduce clauses
- force ≥ 1 clause to be true (if Γ is)
- prune absurd branches

Rules of AVATAR

A-clause

An **A-clause** is $C \leftarrow \Gamma$ where

- C is a clause (disjunction of literals)
- $\Gamma = \bigsqcap_{i=1}^{n} b_i$ with b_i boxes (propositional literals)

AVATAR Split

$$\frac{C_1 \vee \ldots \vee C_n \leftarrow \Gamma}{\bigwedge_{i=1}^{n} (C_i \leftarrow \llbracket C_i \rrbracket) \qquad \boxed{\Gamma \Rightarrow (\bigsqcup_{i=1}^{n} \llbracket C_i \rrbracket)}} \text{(ASplit)}$$

$$\text{if } i \neq j \Rightarrow \text{vars}(C_i) \cap \text{vars}(C_j) = \emptyset$$

AVATAR Absurd

$$\frac{\bot \leftarrow \bigsqcap_{i=1}^{n} b_i}{\bigsqcup_{i=1}^{n} \neg b_i} \text{(A}\bot\text{)}$$

- deduce clauses
- force ≥ 1 clause to be true (if Γ is)
- prune absurd branches

Rules of AVATAR

A-clause

An **A-clause** is $C \leftarrow \Gamma$ where

- C is a clause (disjunction of literals)
- $\Gamma = \prod_{i=1}^{n} b_i$ with b_i boxes (propositional literals)

AVATAR Split

$$\frac{C_1 \vee \ldots \vee C_n \leftarrow \Gamma}{\bigwedge_{i=1}^{n} (C_i \leftarrow \llbracket C_i \rrbracket) \qquad \Gamma \Rightarrow (\bigsqcup_{i=1}^{n} \llbracket C_i \rrbracket)} \text{(ASplit)}$$

$$\text{if } i \neq j \Rightarrow \text{vars}(C_i) \cap \text{vars}(C_j) = \emptyset$$

AVATAR Absurd

$$\frac{\bot \leftarrow \prod_{i=1}^{n} b_i}{\boxed{\bigsqcup_{i=1}^{n} \neg b_i}} (\text{A}\bot)$$

- deduce clauses
- force ≥ 1 clause to be true (if Γ is)
- prune absurd branches