Superposition with Structural Induction

Simon Cruanes

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

29th of September, 2017

FroCoS 2017, Brasília

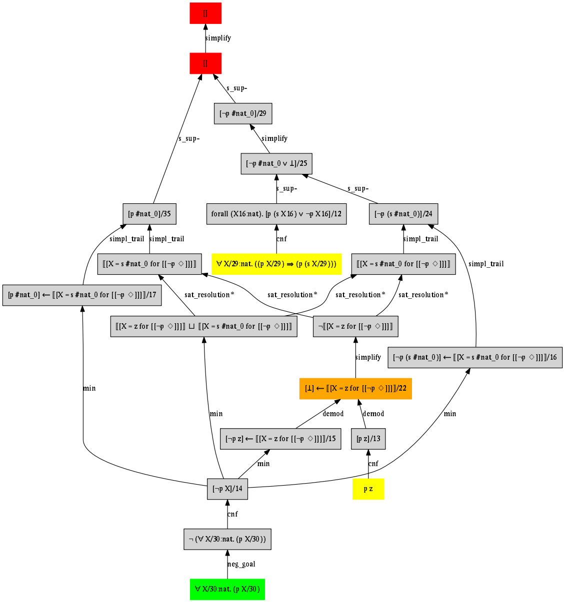
Summary

Outline

This talk: mixing Superposition and Induction

- mix of first-order and induction (and theories. . .) useful for, e.g., Sledgehammer, Why3.
- **Superposition**: state of the art for first-order classical reasoning (implemented in the best FO provers: E, SPASS, Vampire, . . .)
- **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 inductive proofs that are not too hard (possibly helped with lemmas)
  NOT about making the best inductive prover ever!

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



Extended Logic

For inductive proving, we use an extended logic:

- First order logic + equality + polymorphic types ($\sim$ TFF1)
- inductive datatypes (with standard model)
- recursive functions or rewriting rules
  (terminating confluent system)

$\triangleright$ Roughly corresponds to (an encoding of) TIP, or a fragment of SMT-LIB 2.6

(TIP: "Tons of Inductive Problems", Dan Rosén, Nick Smallbone, Moa Johansson, Koen Claessen)

Roadmap: Blocks to combine

Combination of many ingredients:

- a first-order superposition prover
- inference rules for inductive datatypes
- recursive functions (on datatypes)
- case analysis with AVATAR (simplified)
- structural induction schemes (explicit induction)
- heuristics that suggest inductive lemmas

Summary

Superposition in a Nutshell

the Superposition calculus:

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

Suppose we have only two elements a and b, with $p(a) \wedge p(b)$.
Then prove $\forall x.\ p(x)$ by refuting $\exists c. \neg p(c)$ (skolemized):

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg p(c) \qquad x \simeq a \vee x \simeq b}{\neg p(a) \vee c \simeq b} \qquad p(a)}{c \simeq b} \qquad \neg p(c)}{\neg p(b)} \qquad p(b)}{\bot}$$

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

Superposition in a Nutshell

the Superposition calculus:

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

Suppose we have only two elements a and b, with $p(a) \land p(b)$.
Then prove $\forall x.\ p(x)$ by refuting $\exists c. \neg p(c)$ (skolemized):

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

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

Beyond Superposition: Clause Splitting With AVATAR

Induction requires case analysis.

However, Superposition not very good with boolean reasoning...

$\Rightarrow$ use the AVATAR calculus [Voronkov 2014] (simplified)

Example

Typical case analysis for induction:

$$\frac{\mathfrak{n}_0 \simeq 0 \oplus \mathfrak{n}_0 \simeq s(\mathfrak{n}_1)}{\begin{array}{c} \mathfrak{n}_0 \simeq 0 \leftarrow \llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \\ \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \leftarrow \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket \\ \llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \oplus \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket \end{array}}$$

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

Beyond Superposition: Clause Splitting With AVATAR

Induction requires case analysis.

However, Superposition not very good with boolean reasoning...

$\Rightarrow$ use the AVATAR calculus [Voronkov 2014] (simplified)

Example

Typical case analysis for induction:

$$\frac{\mathfrak{n}_0 \simeq 0 \oplus \mathfrak{n}_0 \simeq s(\mathfrak{n}_1)}{\begin{array}{c} \mathfrak{n}_0 \simeq 0 \leftarrow \llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \\ \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \leftarrow \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket \\ \llbracket \mathfrak{n}_0 \simeq 0 \rrbracket \oplus \llbracket \mathfrak{n}_0 \simeq s(\mathfrak{n}_1) \rrbracket \end{array}}$$

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

Summary

Recursive Functions

```
(declare-datatype Nat ((z) (s Nat)))

(define-fun-rec fact ((x Nat)) Nat
  (let ((one (s z)))
    (if (leq x one)
      one
      (mult x (fact (pred x))))))
```

- Often, datatypes manipulated via *recursive functions*
- doesn't fit directly into Superposition
- we use rewriting ("Deduction Modulo") for unconditional expansion of defs
- not complete, but pragmatic
- **remark**: could do the same with carefully-chosen orderings (e.g. TKBO + subterm coeffs)

Recursive Functions

```
(declare-datatype Nat ((z) (s Nat)))

(define-fun-rec fact ((x Nat)) Nat
  (let ((one (s z)))
    (if (leq x one)
      one
      (mult x (fact (pred x))))))
```

- Often, datatypes manipulated via *recursive functions*
- doesn't fit directly into Superposition
- we use rewriting ("Deduction Modulo") for unconditional expansion of defs
- not complete, but pragmatic
- **remark**: could do the same with carefully-chosen orderings (e.g. TKBO + subterm coeffs)

Compiling to Rewriting

- compile away constructs of TIP into rewriting
- eliminate "ite", "match", λ-terms
- done before CNF

```
(declare-datatype Nat ((z) (s Nat)))
(define-fun-rec leq ((x Nat)(y Nat)) Bool
  (match x (case z true)
    (case (s x2)
      (match y (case z false)
        (case (s y2) (leq x2 y2))))))
```

$$\begin{aligned} \forall x.\ \mathrm{leq}(z, x) &\rightsquigarrow \top \\ \forall x.\ \mathrm{leq}(s(x), z) &\rightsquigarrow \bot \\ \forall x\ y.\ \mathrm{leq}(s(x), s(y)) &\rightsquigarrow \mathrm{leq}(x, y) \end{aligned}$$

```
(define-fun pred ((x Nat)) Nat
  (match x
    (case z z)
    (case (s x2) x2)))
(define-fun-rec fact ((x Nat)) Nat
  (let ((one (s z)))
    (if (leq x one)
      one
      (mult x (fact (pred x))))))
```

$$\begin{aligned} \mathrm{pred}(z) &\rightsquigarrow z \\ \forall x.\ \mathrm{pred}(s(x)) &\rightsquigarrow x \\ \forall x.\ \mathrm{fact}(x) &\rightsquigarrow \mathrm{f}(x, \mathrm{leq}(x, s(z))) \\ \forall x.\ \mathrm{f}(x, \top) &\rightsquigarrow s(z) \\ \forall x.\ \mathrm{f}(x, \bot) &\rightsquigarrow \mathrm{mult}(x, \\ &\qquad \mathrm{f}(\mathrm{pred}(x), \\ &\qquad\quad \mathrm{leq}(\mathrm{pred}(x), s(z)))) \end{aligned}$$

(where f is fresh)

Compiling to Rewriting

- compile away constructs of TIP into rewriting
- eliminate "ite", "match", λ-terms
- done before CNF

```
(declare-datatype Nat ((z) (s Nat)))
(define-fun-rec leq ((x Nat)(y Nat)) Bool
  (match x (case z true)
    (case (s x2)
      (match y (case z false)
        (case (s y2) (leq x2 y2))))))
```

$$
\begin{array}{rcl}
\forall x.\ \mathrm{leq}(z, x) & \rightsquigarrow & \top \\
\forall x.\ \mathrm{leq}(s(x), z) & \rightsquigarrow & \bot \\
\forall x\ y.\ \mathrm{leq}(s(x), s(y)) & \rightsquigarrow & \mathrm{leq}(x, y)
\end{array}
$$

```
(define-fun pred ((x Nat)) Nat
  (match x
    (case z z)
    (case (s x2) x2)))
(define-fun-rec fact ((x Nat)) Nat
  (let ((one (s z)))
    (if (leq x one)
      one
      (mult x (fact (pred x))))))
```

$$
\begin{array}{rcl}
\mathrm{pred}(z) & \rightsquigarrow & z \\
\forall x.\ \mathrm{pred}(s(x)) & \rightsquigarrow & x \\
\forall x.\ \mathrm{fact}(x) & \rightsquigarrow & \mathrm{f}(x, \mathrm{leq}(x, s(z))) \\
\forall x.\ \mathrm{f}(x, \top) & \rightsquigarrow & s(z) \\
\forall x.\ \mathrm{f}(x, \bot) & \rightsquigarrow & \mathrm{mult}(x, \\
 & & \quad \mathrm{f}(\mathrm{pred}(x), \\
 & & \quad\quad \mathrm{leq}(\mathrm{pred}(x), s(z)))
\end{array}
$$

(where f is fresh)

Summary

Structural Induction Schemes

We use the usual explicit structural induction rules on datatypes.

- $\cdot \lhd \cdot$ is the well-founded ordering (subterm through constructors)
- nat with $\{0, s\}$:

$$\forall P : \text{nat} \to \text{bool}.\ (P(0) \land \forall n : \text{nat}.\ P(n) \Rightarrow P(s(n))) \Rightarrow \forall n.\ P(n)$$

- list(α) with $\{[\,], (::)\}$:

$$\begin{array}{l} \forall \alpha.\ \forall P : \text{list}(\alpha) \to \text{bool}. \\ \quad \left(\begin{array}{l} P([\,]) \land \\ (\forall x : \alpha\ l : \text{list}(\alpha).\ P(l) \Rightarrow P(x :: l)) \end{array} \right) \\ \quad \Rightarrow \forall l.\ P(l) \end{array}$$

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 $\exists \mathfrak{l}_0 : \text{list}.\ \neg p(\mathfrak{l}_0)$ ($\mathfrak{l}_0$: minimal witness for $\neg p$)
  - 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

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 $\exists \mathfrak{l}_0 : \text{list}.\ \neg p(\mathfrak{l}_0)$ ($\mathfrak{l}_0$: minimal witness for $\neg p$)
  - 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

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 $\exists \mathfrak{l}_0 : \text{list}.\ \neg p(\mathfrak{l}_0)$ ($\mathfrak{l}_0$: minimal witness for $\neg p$)
  - 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

base case: easy

$$\dfrac{\dfrac{\dfrac{\mathfrak{l}_0 \simeq [\,] \leftarrow [\![\mathfrak{l}_0 \simeq [\,]]\!] \qquad \neg p(\mathfrak{l}_0)}{\neg\, p([\,]) \leftarrow [\![\mathfrak{l}_0 \simeq [\,]]\!]} \qquad p([\,])}{\bot \leftarrow [\![\mathfrak{l}_0 \simeq [\,]]\!]}}{\neg [\![\mathfrak{l}_0 \simeq [\,]]\!]}$$

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 $\exists \mathfrak{l}_0 : \text{list}.\ \neg p(\mathfrak{l}_0)$ ($\mathfrak{l}_0$: minimal witness for $\neg p$)
  - 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:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket \qquad \neg p(\mathfrak{l}_0)}{\neg p(\mathfrak{t}_0 :: \mathfrak{l}_1) \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket \qquad \neg p(l) \vee p(x :: l)}}{\neg\, p(\mathfrak{l}_1) \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket \qquad p(\mathfrak{l}_1)^{\dagger}}}{\bot \leftarrow \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket}}{\neg \llbracket \mathfrak{l}_0 \simeq \mathfrak{t}_0 :: \mathfrak{l}_1 \rrbracket}$$

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 $\forall x.\ p(x)$
  (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)

Inductive Strengthening

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 $\forall x.\ p(x)$

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

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

Combining several proofs with lemmas in AVATAR

- Each induction is done with a cut (on the ind. schema instance)
- All inductive proofs live in the same set of clauses
- prove the lemma F in one branch by refutation $(\neg\llbracket F \rrbracket)$
- use the lemma F in the other branch $(\llbracket F \rrbracket)$

Inference Rule

introduce lemma F:

$$\frac{\top}{\begin{array}{ccc} & F & \leftarrow \llbracket F \rrbracket \\ \wedge & \neg F & \leftarrow \neg\llbracket F \rrbracket \end{array}}$$

Combining several proofs with lemmas in AVATAR

- Each induction is done with a cut (on the ind. schema instance)
- All inductive proofs live in the same set of clauses
- prove the lemma F in one branch by refutation $(\neg\llbracket F \rrbracket)$
- use the lemma F in the other branch $(\llbracket F \rrbracket)$

Inference Rule

introduce lemma F:

$$\frac{\top}{\begin{array}{ccc} & F & \leftarrow \llbracket F \rrbracket \\ \wedge & \neg F & \leftarrow \neg\llbracket F \rrbracket \end{array}}$$

Combining several proofs with lemmas in AVATAR

- Each induction is done with a cut (on the ind. schema instance)
- All inductive proofs live in the same set of clauses
- prove the lemma F in one branch by refutation $(\neg [\![F]\!])$
- use the lemma F in the other branch $([\![F]\!])$

Inference Rule

introduce lemma F (and reduce it to CNF) :

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

$\mathrm{cnf}(\neg F) \leftarrow \neg [\![F]\!]$ is where the induction happens.

Example: Associativity of addition

- Let $F \stackrel{\text{def}}{=} \forall x\ y\ z : \text{nat}.\ x + (y + z) \simeq (x + y) + z$.
- induction on $\{x\}$, coverset $\mathrm{x}_0 = \{0, s(\mathrm{x}_1)\}$.
- clause set for this lemma:

$$
\begin{array}{rcl}
\forall x\ y\ z.\ x + (y + z) \simeq (x + y) + z & \leftarrow & [\![F]\!] \sqcap \\
0 + (\mathrm{y}_0 + \mathrm{z}_0) \not\simeq (0 + \mathrm{y}_0) + \mathrm{z}_0 & \leftarrow & \neg[\![F]\!] \sqcap [\![\mathrm{x}_0 \simeq 0]\!] \\
s(\mathrm{x}_1) + (\mathrm{y}_0 + \mathrm{z}_0) \not\simeq (s(\mathrm{x}_1) + \mathrm{y}_0) + \mathrm{z}_0 & \leftarrow & \neg[\![F]\!] \sqcap [\![\mathrm{x}_0 \simeq s(\mathrm{x}_1)]\!] \\
\forall y\ z.\ \mathrm{x}_1 + (y + z) \simeq (\mathrm{x}_1 + y) + z & \leftarrow & \neg[\![F]\!] \sqcap [\![\mathrm{x}_0 \simeq s(\mathrm{x}_1)]\!] \\
 & & [\![\mathrm{x}_0 \simeq 0]\!] \oplus [\![\mathrm{x}_0 \simeq s(\mathrm{x}_1)]\!]
\end{array}
$$

- lemma ready to be used
- (beginning of) refutation of the lemma
- case split

Example: Associativity of addition

- Let $F \stackrel{\text{def}}{=} \forall x\ y\ z : \text{nat}.\ x + (y + z) \simeq (x + y) + z$.
- induction on $\{x\}$, coverset $\mathrm{x}_0 = \{0, s(\mathrm{x}_1)\}$.
- clause set for this lemma:

$$
\begin{array}{rcl}
\forall x\ y\ z.\ x + (y + z) \simeq (x + y) + z & \leftarrow & \llbracket F \rrbracket \sqcap \\
0 + (\mathrm{y}_0 + \mathrm{z}_0) \not\simeq (0 + \mathrm{y}_0) + \mathrm{z}_0 & \leftarrow & \neg \llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq 0 \rrbracket \\
s(\mathrm{x}_1) + (\mathrm{y}_0 + \mathrm{z}_0) \not\simeq (s(\mathrm{x}_1) + \mathrm{y}_0) + \mathrm{z}_0 & \leftarrow & \neg \llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket \\
\forall y\ z.\ \mathrm{x}_1 + (y + z) \simeq (\mathrm{x}_1 + y) + z & \leftarrow & \neg \llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket \\
 & & \llbracket \mathrm{x}_0 \simeq 0 \rrbracket \oplus \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket
\end{array}
$$

- lemma ready to be used
- (beginning of) refutation of the lemma
- case split

Simultaneous Induction

- Example: $F \stackrel{\text{def}}{=} \forall x\ y\ z : \text{nat}.\ x \leq y \land y \leq z \Rightarrow x \leq z$
- $\{\forall x.\ 0 \leq x,\ \forall x.\ \neg(s(x) \leq 0),\ \forall x\ y.\ x \leq y \iff s(x) \leq s(y)\}$
- clause set for induction on $\{x, y, z\}$:

$$
\begin{array}{rcl}
\neg(x \leq y) \lor \neg(y \leq z) \lor (x \leq z) & \leftarrow & \llbracket F \rrbracket \\
0 \leq 0 & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq 0 \rrbracket \sqcap \llbracket \mathrm{y}_0 \simeq 0 \rrbracket \\
0 \leq 0 & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{y}_0 \simeq 0 \rrbracket \sqcap \llbracket \mathrm{z}_0 \simeq 0 \rrbracket \\
s(\mathrm{x}_1) \leq 0 & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket \sqcap \llbracket \mathrm{y}_0 \simeq 0 \rrbracket \\
s(\mathrm{x}_1) \leq s(\mathrm{y}_1) & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket \sqcap \llbracket \mathrm{y}_0 \simeq s(\mathrm{y}_1) \rrbracket \\
s(\mathrm{y}_1) \leq 0 & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{y}_0 \simeq s(\mathrm{y}_1) \rrbracket \sqcap \llbracket \mathrm{z}_0 \simeq 0 \rrbracket \\
s(\mathrm{y}_1) \leq s(\mathrm{z}_1) & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{y}_0 \simeq s(\mathrm{y}_1) \rrbracket \sqcap \llbracket \mathrm{z}_0 \simeq s(\mathrm{z}_1) \rrbracket \\
\neg(0 \leq 0) & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq 0 \rrbracket \sqcap \llbracket \mathrm{z}_0 \simeq 0 \rrbracket \\
\neg(s(\mathrm{x}_1) \leq 0) & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket \sqcap \llbracket \mathrm{z}_0 \simeq 0 \rrbracket \\
\neg(s(\mathrm{x}_1) \leq s(\mathrm{z}_1)) & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket \sqcap \llbracket \mathrm{z}_0 \simeq s(\mathrm{z}_1) \rrbracket \\
\neg(0 \leq s(\mathrm{z}_1)) & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq 0 \rrbracket \sqcap \llbracket \mathrm{z}_0 \simeq s(\mathrm{z}_1) \rrbracket \\
\neg(\mathrm{x}_1 \leq \mathrm{y}_1) \lor \neg(\mathrm{y}_1 \leq \mathrm{z}_1) \lor (\mathrm{x}_1 \leq \mathrm{z}_1) & \leftarrow & \neg\llbracket F \rrbracket \sqcap \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket \sqcap \llbracket \mathrm{y}_0 \simeq s(\mathrm{y}_1) \rrbracket \sqcap \llbracket \mathrm{z}_0 \simeq \\
 & & \llbracket \mathrm{x}_0 \simeq 0 \rrbracket \oplus \llbracket \mathrm{x}_0 \simeq s(\mathrm{x}_1) \rrbracket \\
 & & \llbracket \mathrm{y}_0 \simeq 0 \rrbracket \oplus \llbracket \mathrm{y}_0 \simeq s(\mathrm{y}_1) \rrbracket \\
 & & \llbracket \mathrm{z}_0 \simeq 0 \rrbracket \oplus \llbracket \mathrm{z}_0 \simeq s(\mathrm{z}_1) \rrbracket
\end{array}
$$

Selecting Induction Variables

Inspired from [Aubin 77]

- Intuition: do induction on arguments that block evaluation
- if two variables block the same term, do induction on both
- induction fails if a variable is both *active* and *invariant*

- in $\forall x\ y\ z.\ (x+y)+z \simeq x+(y+z)$, only x blocks evaluation
  $\Rightarrow$ do induction on x
- $\leq$ defined by
  $(0 \leq x) \rightsquigarrow \top, (s(x) \leq 0) \rightsquigarrow \bot, (s(x) \leq s(y)) \rightsquigarrow (x \leq y)$:
  In $\forall x\ y\ z.\ x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$, evaluation blocked by
  $\{\{x, y\}, \{y, z\}, \{x, z\}\}$
  $\Rightarrow$ do induction on $\{x, y, z\}$
- qrev with the rules
  $\text{qrev}([], x) \rightsquigarrow x, \text{qrev}(x :: y, z) \rightsquigarrow \text{qrev}(y, x :: z)$: the first position is primary, the second is an accumulator.
  $\Rightarrow$ given $F[\text{qrev}(x, t)]$, do induction on x, possibly generalize t

Selecting Induction Variables

Inspired from [Aubin 77]

- Intuition: do induction on arguments that block evaluation
- if two variables block the same term, do induction on both
- induction fails if a variable is both *active* and *invariant*

- in $\forall x\ y\ z.\ (x + y) + z \simeq x + (y + z)$, only x blocks evaluation
  $\Rightarrow$ do induction on x
- $\leq$ defined by
  $(0 \leq x) \rightsquigarrow \top, (s(x) \leq 0) \rightsquigarrow \bot, (s(x) \leq s(y)) \rightsquigarrow (x \leq y)$:
  In $\forall x\ y\ z.\ x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$, evaluation blocked by
  $\{\{x, y\}, \{y, z\}, \{x, z\}\}$
  $\Rightarrow$ do induction on $\{x, y, z\}$
- qrev with the rules
  $\mathrm{qrev}([], x) \rightsquigarrow x, \mathrm{qrev}(x :: y, z) \rightsquigarrow \mathrm{qrev}(y, x :: z)$: the first position is primary, the second is an accumulator.
  $\Rightarrow$ given $F[\mathrm{qrev}(x, t)]$, do induction on x, possibly generalize t

Selecting Induction Variables

Inspired from [Aubin 77]

- Intuition: do induction on arguments that block evaluation
- if two variables block the same term, do induction on both
- induction fails if a variable is both *active* and *invariant*

- in $\forall x\ y\ z.\ (x+y)+z \simeq x+(y+z)$, only x blocks evaluation
  $\Rightarrow$ do induction on x
- $\leq$ defined by
  $(0 \leq x) \rightsquigarrow \top, (s(x) \leq 0) \rightsquigarrow \bot, (s(x) \leq s(y)) \rightsquigarrow (x \leq y)$:
  In $\forall x\ y\ z.\ x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$, evaluation blocked by
  $\{\{x,y\},\{y,z\},\{x,z\}\}$
  $\Rightarrow$ do induction on $\{x,y,z\}$
- qrev with the rules
  $\mathrm{qrev}([], x) \rightsquigarrow x, \mathrm{qrev}(x :: y, z) \rightsquigarrow \mathrm{qrev}(y, x :: z)$: the first position is primary, the second is an accumulator.
  $\Rightarrow$ given $F[\mathrm{qrev}(x,t)]$, do induction on x, possibly generalize t

Selecting Induction Variables

Inspired from [Aubin 77]

- Intuition: do induction on arguments that block evaluation
- if two variables block the same term, do induction on both
- induction fails if a variable is both *active* and *invariant*

- in $\forall x\ y\ z.\ (x+y)+z \simeq x+(y+z)$, only x blocks evaluation
  $\Rightarrow$ do induction on x
- $\leq$ defined by
  $(0 \leq x) \rightsquigarrow \top, (s(x) \leq 0) \rightsquigarrow \bot, (s(x) \leq s(y)) \rightsquigarrow (x \leq y)$:
  In $\forall x\ y\ z.\ x \leq y \Rightarrow y \leq z \Rightarrow x \leq z$, evaluation blocked by
  $\{\{x,y\},\{y,z\},\{x,z\}\}$
  $\Rightarrow$ do induction on $\{x,y,z\}$
- qrev with the rules
  $\mathrm{qrev}([\,],x) \rightsquigarrow x, \mathrm{qrev}(x :: y, z) \rightsquigarrow \mathrm{qrev}(y, x :: z)$: the first position is primary, the second is an accumulator.
  $\Rightarrow$ given $F[\mathrm{qrev}(x,t)]$, do induction on x, possibly generalize t

Possible Extensions (future work)

Functional Induction

- we have function definitions (in TIP)
- recursive calls provide a well-founded scheme
- could use these schemes (done in other provers, e.g. ACL2)
  $\forall x_1, \ldots, x_n.\ P[f(x_1, \ldots, x_n)]$

Better Lemma Divination

- guessing lemmas is critical (not cut-free!)
- currently: generalize from goal clauses (heuristic)
- possibility: à la HipSpec / Hipster
- possibility: better generalization from "stuck" negative clauses (i.e., goals)

$\rightarrow$ lots of literature... a lot of work

Possible Extensions (future work)

Functional Induction

- we have function definitions (in TIP)
- recursive calls provide a well-founded scheme
- could use these schemes (done in other provers, e.g. ACL2)
  $\forall x_1, \ldots, x_n.\ P[f(x_1, \ldots, x_n)]$

Better Lemma Divination

- guessing lemmas is critical (not cut-free!)
- currently: generalize from goal clauses (heuristic)
- possibility: à la HipSpec / Hipster
- possibility: better generalization from "stuck" negative clauses (i.e., goals)

→ lots of literature. . . a lot of work

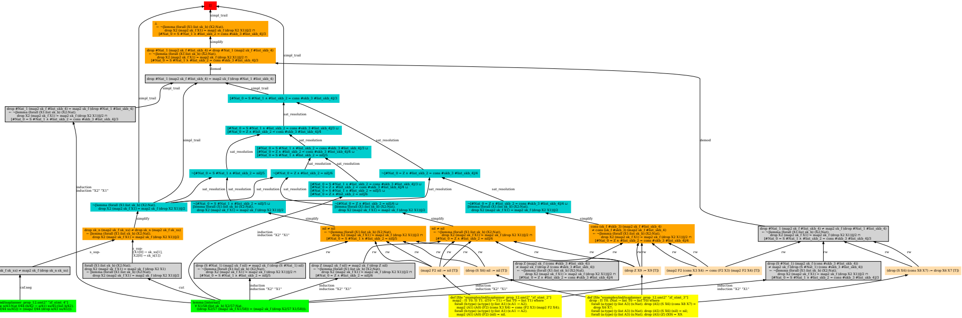
Summary

Implementation

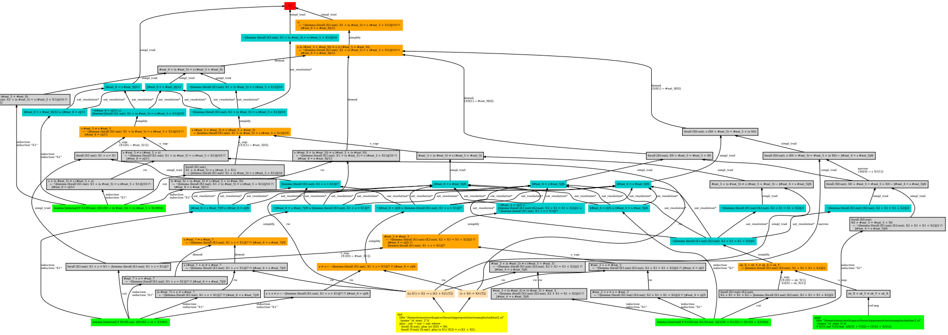
Zipperposition, a Superposition prover in OCaml

- not the best, but flexible and feature full
- follows the design of E, + polymorphic types, int arith, induction, and rewriting (on terms and literals: "Deduction Modulo")
- OCaml is expressive, reasonably fast, and safe (fewer bugs)
- BSD license, `https://github.com/c-cube/zipperposition`
- can output graphical proofs (using graphviz)

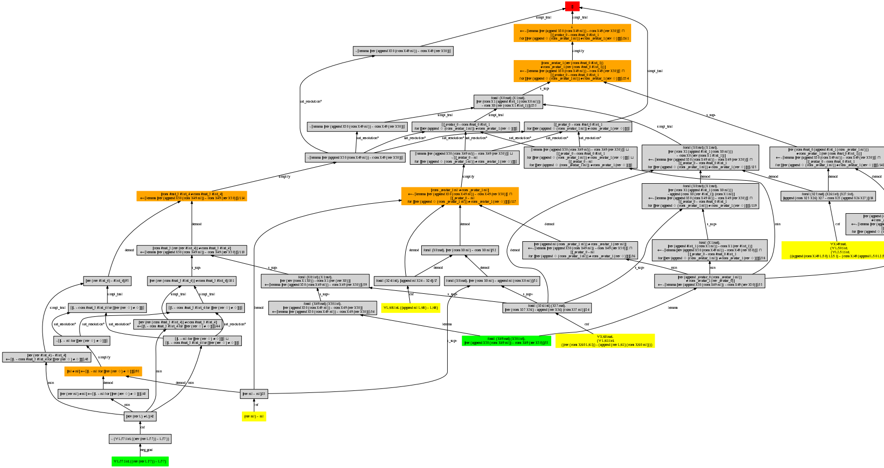
Isaplanner 12: $\mathrm{map}(f, \mathrm{drop}(n, l)) \simeq \mathrm{drop}(n, \mathrm{map}(f, l))$



$a + b \simeq b + a$



$\mathrm{rev}(\mathrm{rev}(l)) = l$ (with a user-provided lemma)



(note the intermediate lemma)

Some Benchmarks (30s timeout)

Isaplanner	unsat (/86)	time (s)
CVC4-gen	73	12.4
CVC4	67	1.6
Zipperposition	64	4.2

TIP	unsat (/484)	time (s)
CVC4-gen	160	27.7
Zipperposition	139	53.2
CVC4	138	8.2

TPTP (first-order)	solved	unsat	sat	time (s)
E	9802	8840	962	15,160
Zipperposition	5477	4865	612	14,445
CVC4	5282	5253	29	9283
prover9	3341	3341	0	4590

Conclusion

Pragmatic extension of 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 very useful to handle case splitting
  - $\rightarrow$ can pursue many simultaneous proofs at the same time
  - $\rightarrow$ failed induction attempts don't jeopardize the others
- the tricky parts:
  - handle datatypes (some progress in Vampire)
  - recursive functions: need rewriting **or** good term orderings
  - divination of lemmas (crazy heuristics?)
- many challenges also relevant for combination FO provers + ITPs! (e.g. recursive functions, datatypes)
- code at `https://github.com/c-cube/zipperposition`

Thank you for your attention!

Rules of Superposition

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

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

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

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

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

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

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

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

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

AVATAR Absurd

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

- 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

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

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

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

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

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

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

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

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)
- $\Gamma = \bigsqcap_{i=1}^{n} b_i$ with b_i boxes (propositional literals)

AVATAR Split

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

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

AVATAR Absurd

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

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