Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond

Simon Cruanes

École polytechnique and Inria
https://who.rocq.inria.fr/Simon.Cruanes/

September 10th, 2015

Foreword

In this thesis: techniques to have a program prove

$$\mathrm{len}(\mathrm{dup}(l)) \simeq 2 \cdot \mathrm{len}(l)$$

where

$$\begin{aligned}
\mathrm{dup}([]) &\simeq [] \\
\mathrm{dup}(x :: l) &\simeq x :: x :: \mathrm{dup}(l) \\
\mathrm{len}([]) &\simeq 0 \\
\mathrm{len}(x :: l) &\simeq 1 + \mathrm{len}(l)
\end{aligned}$$

. . . and more!

Summary

The Prevalence of Logic

Formal Logic: the art of precise reasoning.

- foundation of Mathematics
- theoretical Computer Science
- Philosophy
- formal methods in the industry (verifying planes, subways, CPUs. . .)
- . . .

A Case for Automated Theorem Proving

Logic revolves around theorems and proofs

Proof	: irrefutable argument following formal rules
Theorem	: claim (formula) backed by a proof
Conjecture	: claim not (yet) backed by a proof

Finding a proof of a conjecture is hard, but useful.

→ Automate it as much as possible: *Automated Theorem Proving*

A Case for Automated Theorem Proving

Logic revolves around theorems and proofs

Proof : irrefutable argument following formal rules
Theorem : claim (formula) backed by a proof
Conjecture : claim not (yet) backed by a proof

Finding a proof of a conjecture is hard, but useful.

→ Automate it as much as possible: *Automated Theorem Proving*

Classical First-Order Logic

Mathematical formulas with quantifiers.

Example (The Internet)

"Cats are cute, and Felix is a cat; therefore Felix is cute"

$$(\text{isa}(\text{Felix}, \text{cat}) \wedge (\forall x.\ \text{isa}(x, \text{cat}) \Rightarrow \text{cute}(x))) \Rightarrow \text{cute}(\text{Felix})$$

- $A \Rightarrow B$ means "if A then B"
- $A \wedge B$ means "A and B" (both are true)
- $A \vee B$ means "A or B" (at least one true)
- $\forall x.\ F$ means "for all x, F"
- $\exists x.\ F$ means "there exists an x such that F"
- $\neg F$ means "not F" (or "F is false")

Equational First-Order Logic

Equality

- extension of first-order logic:
  add predicate $x \simeq y$ ("x equals y")
  if $x \simeq y$, can replace x by y
- very useful theory for many problems

- Superposition (1990): proof system for first-order + equality
  state of the art (most major provers use it)
  → All our work is based on Superposition
- Goal of the thesis: extend Superposition beyond equality
  - theory of Linear Integer Arithmetic
  - Inductive reasoning
  - theory detection, polymorphism, ...

Equational First-Order Logic

Equality

- extension of first-order logic:
  add predicate $x \simeq y$ ("x equals y")
  if $x \simeq y$, can replace x by y
- very useful theory for many problems

- Superposition (1990): proof system for first-order + equality
  state of the art (most major provers use it)
  → All our work is based on Superposition
- Goal of the thesis: extend Superposition beyond equality
  - theory of Linear Integer Arithmetic
  - Inductive reasoning
  - theory detection, polymorphism, . . .

Example

If we learn that "cat" and "chat" are the same concept, we can substitute one for the other:

$$\dfrac{\dfrac{\text{isa}(\text{Felix}, \boxed{\text{chat}}) \qquad \boxed{\text{chat}} \simeq \text{cat}}{\boxed{\text{isa}(\text{Felix}, \text{cat})}} \text{(Sup)} \qquad \neg \boxed{\text{isa}(x, \text{cat})} \lor \text{cute}(x)}{\text{cute}(\text{Felix})} \text{(Res)}$$

Here we have superposition and resolution.

Note the *binding* of x to Felix using *unification*

Substitutions and Unification

Substitution

- noted σ
- maps *variables* to *terms*

Unification

- Crucial operation:
- unify terms s and t means finding σ such that $s\sigma = t\sigma$

Example

$\text{isa}(x, \text{cat})$ and $\text{isa}(\text{Felix}, \text{cat})$ unified by $\sigma = \{x \mapsto \text{Felix}\}$
$f(f(x,b),y)$ and $f(y,f(a,z))$ unified by $\sigma = \{x \mapsto a, y \mapsto f(a,b), z \mapsto b\}$

Rules of Superposition

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

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

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

(some details omitted)

Superposition: Main Loop

- refutational: to prove F, derive $\bot$ from $\neg F$.
- clausal calculus
  - literal: $s \simeq t$ or $s \not\simeq t$
  - clause: disjunction of literals $l_1 \vee l_2 \vee \ldots \vee l_n$
  - 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) or ∞-loop

A Word on Implementation

in ATP, **implementation** (writing actual programs) is important.

OCaml

We used OCaml (`https://ocaml.org`)

- functional language with **strong typing** → safe
- expressive, yet reasonably fast
- designed for theorem proving (ML)
- used in several other provers (iProver, Zenon, KRHyper. . .)

I wrote Logtk[1] and Zipperposition[2] over 3 years (free software).

[1] `https://www.rocq.inria.fr/deducteam/Logtk/`

[2] `https://www.rocq.inria.fr/deducteam/Zipperposition/`

Summary

Linear Integer Arithmetic?

Example

$$\forall x : \text{int}.\ (3 \mid x \land 2 \cdot x \leq 15 \land 5 \leq x) \Rightarrow x \simeq 6$$

Useful for

- program verification (loop indices, arrays, etc.)

  e.g., optimization that changes

  ```
  for (i=1; i≤10; i++) a[j+i]=a[j];
  ```

  into

  ```
  int n = a[j]; for (i=1; i≤10; i++) a[j+i] = n;
  ```

  requires proving $\forall i \in \mathbb{Z}.\ 1 \leq i \leq 10 \Rightarrow j \not\simeq j + i$
- indexed structures
- temporal logic (discrete time $\sim$ int)
- ...

Linear Integer Arithmetic?

Example

$$\forall x : \text{int.}\ (3 \mid x \wedge 2 \cdot x \leq 15 \wedge 5 \leq x) \Rightarrow x \simeq 6$$

Useful for

- program verification (loop indices, arrays, etc.)

  e.g., optimization that changes

  ```
  for (i=1; i≤10; i++) a[j+i]=a[j];
  ```

  into

  ```
  int n = a[j]; for (i=1; i≤10; i++) a[j+i] = n;
  ```

  requires proving $\forall i \in \mathbb{Z}.\ 1 \leq i \leq 10 \Rightarrow j \not\simeq j + i$
- indexed structures
- temporal logic (discrete time $\sim$ int)
- ...

Basics of LIA

- type int (represents $\mathbb{Z}$)
- symbols $0, 1, +$
- predicates $e_1 \leq e_2$, $n \mid e$ (n divides expr e)
- first-order terms (functions, variables...)

remarks

- $n \cdot t$ as a shortcut for $\sum_{i=1}^{n} t$, constants are $n \cdot \mathbf{1}$

→ no general product!

- in $n \mid e$, n must be a *constant* $(1, 2, 3, \ldots)$

  ($n \mid e$ means $\exists x.\ n \cdot x \simeq e$)
- $n \mid e$ always reduced to cases $n = d^e$ where d prime
- $e_1 - e_2$ not needed: $(e_1 - e_2 \leq e_3)$ transformed into $(e_1 \leq e_2 + e_3)$

Basics of LIA

- type int (represents $\mathbb{Z}$)
- symbols $0, 1, +$
- predicates $e_1 \le e_2$, $n \mid e$ (n divides expr e)
- first-order terms (functions, variables...)

remarks

- $n \cdot t$ as a shortcut for $\sum_{i=1}^{n} t$, constants are $n \cdot \mathbf{1}$

→ no general product!

- in $n \mid e$, n must be a *constant* $(1, 2, 3, \ldots)$

  ($n \mid e$ means $\exists x.\ n \cdot x \simeq e$)
- $n \mid e$ always reduced to cases $n = d^e$ where d prime
- $e_1 - e_2$ not needed: $(e_1 - e_2 \le e_3)$ transformed into $(e_1 \le e_2 + e_3)$

Superposition with LIA

Contribution: **first-order** inference system for Superposition + LIA
state of the art:

- combination with black-box solver (hierarchic sup)
- Superposition + linear $\mathbb{Q}$-arithmetic [Waldmann][Korovin Voronkov]

Example

$(16 \mid 2 \cdot a + b) \land (4 \mid c + \mathbf{1}) \land (b \simeq c)$ has no solution

$$16 \mid 2 \cdot a + b$$

Superposition with LIA

Contribution: **first-order** inference system for Superposition + LIA

Example

$(16 \mid 2 \cdot a + b) \land (4 \mid c + \mathbf{1}) \land (b \simeq c)$ has no solution

$$\dfrac{\dfrac{16 \mid 2 \cdot a + b}{2 \mid 2 \cdot a + b}\ (\text{CDiv})}{2 \mid b}$$

Superposition with LIA

Contribution: **first-order** inference system for Superposition + LIA

Example

$(16 \mid 2 \cdot a + b) \land (4 \mid c + \mathbf{1}) \land (b \simeq c)$ has no solution

$$\frac{16 \mid 2 \cdot a + b}{2 \mid b} \text{ (CDiv)}$$

Superposition with LIA

Contribution: **first-order** inference system for Superposition + LIA

Example

$(16 \mid 2 \cdot a + b) \land (4 \mid c + \mathbf{1}) \land (b \simeq c)$ has no solution

$$\dfrac{\dfrac{16 \mid 2 \cdot a + b}{2 \mid \boxed{b}}\,(\text{CDiv}) \qquad \boxed{b} \simeq c}{2 \mid c}\,(\text{CSup})$$

Superposition with LIA

Contribution: **first-order** inference system for Superposition + LIA

Example

$(16 \mid 2 \cdot a + b) \land (4 \mid c + \mathbf{1}) \land (b \simeq c)$ has no solution

$$\dfrac{\dfrac{\dfrac{\dfrac{16 \mid 2\cdot a + b}{2 \mid b}\,(\text{CDiv}) \qquad b \simeq c}{2 \mid c}\,(\text{CSup})}{4 \mid 2\cdot c} \qquad \dfrac{4 \mid c + \mathbf{1}}{4 \mid 2\cdot c + 2}}{\dfrac{4 \mid 2\cdot c + 2 - 2\cdot c}{4 \mid 2}}\,(\text{Chain}\mid)$$

Superposition with LIA

Contribution: **first-order** inference system for Superposition + LIA

Example

$(16 \mid 2 \cdot a + b) \wedge (4 \mid c + \mathbf{1}) \wedge (b \simeq c)$ has no solution

$$\dfrac{\dfrac{\dfrac{16 \mid 2 \cdot a + b}{2 \mid b}\,(\text{CDiv}) \qquad b \simeq c}{2 \mid c}\,(\text{CSup}) \qquad 4 \mid c + \mathbf{1}}{\dfrac{4 \mid 2}{\bot}}\,(\text{Chain}|)$$

LIA → deal with divisibility

Example

$a \simeq 2 \cdot b$ and $a \simeq 2 \cdot c + 1$ has no solution

$$\dfrac{\dfrac{\dfrac{\dfrac{a \simeq 2 \cdot b \qquad a \simeq 2 \cdot c + 1}{2 \cdot b \simeq 2 \cdot c + 1} \text{(Sup)}}{2 \mid 2 \cdot c - 2 \cdot b + 1} \text{(CDiv)}}{2 \mid 1}}{\bot}$$

LIA → case switch

Example

if $b \leq 4 \cdot a \leq b + 2$ and $4 \mid b + 3$, then $\bot$ ($[b, b+2]$ contains no multiple of 4):

$$\dfrac{\dfrac{\dfrac{b \leq 4 \cdot a \qquad 4 \cdot a \leq b + 2}{(4 \cdot a \simeq b) \lor (4 \cdot a \simeq b+1) \lor (4 \cdot a \simeq b+2)}\ (\text{CSwitch})}{(4 \mid b) \lor (4 \cdot a \simeq b+1) \lor (4 \cdot a \simeq b+2)}\ (\text{CDiv}) \qquad 4 \mid b + 3}{(4 \mid 3) \lor (4 \cdot a \simeq b+1) \lor (4 \cdot a \simeq b+2)}\ (\text{Chain}\mid)$$

$$\dfrac{\dfrac{(4 \mid 3) \lor (4 \cdot a \simeq b+1) \lor (4 \cdot a \simeq b+2)}{(4 \cdot a \simeq b+1) \lor (4 \cdot a \simeq b+2)}}{(4 \mid b + 1) \lor (4 \cdot a \simeq b+2)}\ (\text{CDiv})$$

$$\vdots$$

$$\dfrac{4 \cdot a \simeq b + 2}{4 \mid b + 2}\ (\text{CDiv})$$

$$\vdots$$

$$\dfrac{}{\bot}$$

Factoring Rules

Some *factoring* rules (sometimes) needed:

Example

$$\dfrac{\dfrac{(10 \le f(x)) \lor (11 \le f(y))}{(10 \le 11) \Rightarrow (10 \le f(y))}\ (\text{CFact}\le)}{\dfrac{\dfrac{10 \le f(y) \qquad\qquad f(a) \le 5}{10 \le 5}\ (\text{Chain}\le)}{\bot}}$$

with $\{x \mapsto y\}$, then $\{y \mapsto a\}$

note: outside of Presburger fragment (presence of function symbols)

Implementation

Incomplete (counter-ex by Uwe Waldmann); however

System implemented in Zipperposition

→ demonstrate practicability (many systems not implemented!)

→ difficulty: calculus is complex

→ also show it can be efficient

Participation at CASC J7

prover	solved/100	avg time (s)	μ-efficiency	SOTAC	new/50
Princess	81	20.3	291	0.22	35
Zip	80	6.5	626	0.27	44
CVC4	80	10	605	0.24	33
SPASS+T	75	6.8	314	0.18	30
Beagle	73	12.7	325	0.18	28

Implementation

Incomplete (counter-ex by Uwe Waldmann); however

System implemented in Zipperposition

→ demonstrate practicability (many systems not implemented!)

→ difficulty: calculus is complex

→ also show it can be efficient

Participation at CASC J7

prover	solved/100	avg time (s)	μ-efficiency	SOTAC	new/50
Princess	81	20.3	291	0.22	35
Zip	80	6.5	626	0.27	44
CVC4	80	10	605	0.24	33
SPASS+T	75	6.8	314	0.18	30
Beagle	73	12.7	325	0.18	28

Implementation Highlights

Difficulties

- rules are complex
- soundness concerns (overflows, implementation errors)
- efficiency concerns

Solutions

- use OCaml → high expressiveness and safety
- use Zarith/GMP for arbitrary-precision arithmetic
- power of abstraction to simplify code:
  - use canonical forms to represent arithmetic
  - simpler backtracking through iterators

Implementation Highlights: Linear Expressions

Seek canonical forms in representations:

```
type linexp = private {
    const : Z.t; (* ≥0 *)
    terms : (Z.t * term) list; (* sorted; each coeff >0 *)
}

val singleton : Z.t → term → linexp
val add : Z.t → term → linexp → linexp

val sum : linexp → linexp → linexp
val difference : linexp → linexp → linexp option
(* ... *)

type focused_linexp = private {
    term : term;
    coeff : Z.t; (* >0 *)
    rest : linexp;
}
val focus : term → linexp → focused_linexp option
val unfocus : focused_linexp → linexp
```

Implementation Highlights: Unification Algorithms

Unification is not trivial (backtracking):

Example

$$\frac{0 \le 2\cdot f(x) + f(y) + g(z) \qquad 3\cdot f(z) + h(z) \le 2}{h(z) \le 2 + g(z)}\ (\text{Chain}\le)$$

Unify $2\cdot f(x) + \underline{f(y)} + g(z)$ and $\underline{3\cdot f(z)} + h(z)$:

- unify $\underline{f(y)}$ and $\underline{3\cdot f(z)}$ with $\{y \mapsto z\}$
- unify $2\cdot f(x)$ and $\underline{f(y)}$ to obtain coefficient 3
- result: $\{x \mapsto y, y \mapsto z\}$
  common focused term: $\underline{3\cdot f(z)}$

(used in combination with term indexing)

Implementation Highlights: Unification Algorithms

Unification is not trivial (backtracking):

Example

$$\frac{0 \leq 2 \cdot f(x) + f(y) + g(z) \qquad 3 \cdot f(z) + h(z) \leq 2}{h(z) \leq 2 + g(z)} \text{(Chain}\leq\text{)}$$

Unify $2 \cdot f(x) + \underline{f(y)} + g(z)$ and $\underline{3 \cdot f(z)} + h(z)$:

- unify $\underline{f(y)}$ and $\underline{3 \cdot f(z)}$ with $\{y \mapsto z\}$
- unify $2 \cdot f(x)$ and $\underline{f(y)}$ to obtain coefficient 3
- result: $\{x \mapsto y, y \mapsto z\}$
  common focused term: $\underline{3 \cdot f(z)}$

(used in combination with term indexing)

Implementation Highlights: Unification Algorithms

Unification is not trivial (backtracking):

Example

$$\frac{0 \leq 2 \cdot f(x) + \boxed{f(y)} + g(z) \qquad \boxed{3 \cdot f(z)} + h(z) \leq 2}{h(z) \leq 2 + 3 \cdot g(z) + 6 \cdot f(x)} \text{ (Chain}\leq\text{)}$$

Unify $2 \cdot f(x) + \underline{f(y)} + g(z)$ and $\underline{3 \cdot f(z)} + h(z)$:

- unify $\underline{f(y)}$ and $\underline{3 \cdot f(z)}$ with $\{y \mapsto z\}$
- unify $2 \cdot f(x)$ and $\underline{f(y)}$ to obtain coefficient 3
- result: $\{y \mapsto z\}$

  $\underline{f(z)}$ on left, $\underline{3 \cdot f(z)}$ on right (multiply left literal by 3)

(used in combination with term indexing)

Implementation Highlights: Iterators

Backtracking is difficult → sought way of simplifying it

Sequence

We introduce novel monadic iterators

```
type α sequence = (α → unit) → unit

val return : α → α sequence
val (>>=) : α sequence → (α → β sequence) → β sequence
val map : (α → β) → α sequence → β sequence
val (@) : α sequence → α sequence → α sequence
val head : α sequence → α option
val cons : α → α sequence → α sequence
(* ... *)
```

An α sequence is a lazy list of α values

`return` and `>>=` form a backtracking monad

Implementation Highlights: Unification Algorithms (2)

previous example: unify $2 \cdot f(x) + \underline{f(y)} + g(z)$ and $\underline{3 \cdot f(z)} + h(z)$

```
let unify_ff σ f1 f2 =
  try
    (* unify focused terms *)
    let ρ1 = unify σ f1.term f2.term in
    (* extend unifier to subset of unfocused terms *)
    unify_self_f ρ1 f1
    >>= fun (new_f1, ρ2) →
    unify_self_f ρ2 f2
    >>= fun (new_f2, θ) →
    return (new_f1, new_f2, θ)
  with Fail → empty
```

```
val unify_self_f : subst → focused_linexp →
                   (focused_linexp * subst) sequence
val unify_ff : focused_linexp → focused_linexp →
               (focused_linexp * focused_linexp * subst) sequence
```

Summary of LIA

- calculus dealing with $\leq, \simeq, |$
  - purely deductive rules
  - seek canonical forms for literals (prime divisors, no $-$, etc.)
    $\rightarrow$ reflects in OCaml representation of linear expressions and literals
- variable elimination to avoid full ACU unification
  - specific unification algorithms with backtracking
    backtracking: iterators
  - can use regular terms and indexing
- competitive implementation (CASC J7 + benchmarks)

  (mere *plugin* to Zipperposition)

Summary

Induction: What is it about?

«Le raisonnement mathématique par excellence» (Poincaré)

Example

Assume $\forall x : \iota.\ \forall l : \text{list}.\ p(l) \Rightarrow p(x :: l)$ and $p([])$.

Then $\forall l : \text{list}.\ p(l)$ can be proved by induction on l

We focus on structural induction:

- powerful enough for data structures
- generalizes induction on natural numbers (« récurrence »)

→ deal with **non-linear arithmetic, data structures**

- simpler than general Nœtherian induction (uses subterm ordering $\lhd$)
- SoTA: provers dedicated to Induction (Spike, ACL2...)

Work inspired from [Kersani&Peltier, 2013].

Induction: What is it about?

«Le raisonnement mathématique par excellence» (Poincaré)

Example

Assume $\forall x : \iota.\ \forall l : \text{list}.\ p(l) \Rightarrow p(x :: l)$ and $p([])$.

Then $\forall l : \text{list}.\ p(l)$ can be proved by induction on l

We focus on structural induction:

- powerful enough for data structures
- generalizes induction on natural numbers (« récurrence »)

→ deal with **non-linear arithmetic, data structures**

- simpler than general Nœtherian induction
  (uses subterm ordering $\lhd$)
- SoTA: provers dedicated to Induction (Spike, ACL2...)

Work inspired from [Kersani&Peltier, 2013].

Induction: What is it about?

«Le raisonnement mathématique par excellence» (Poincaré)

Example

Assume $\forall x : \iota.\ \forall l : \text{list}.\ p(l) \Rightarrow p(x :: l)$ and $p([])$.

Then $\forall l : \text{list}.\ p(l)$ can be proved by induction on l

We focus on structural induction:

- powerful enough for data structures
- generalizes induction on natural numbers (« récurrence »)

→ deal with **non-linear arithmetic, data structures**

- simpler than general Nœtherian induction
  (uses subterm ordering $\lhd$)
- SoTA: provers dedicated to Induction (Spike, ACL2. . .)

Work inspired from [Kersani&Peltier, 2013].

Definitions

An inductive type is generated from set of constructors

Example

- nat has constructors $\{0 : \text{nat}, s : \text{nat} \to \text{nat}\}$
- list has constructors $\{[] : \text{list}, (::) : (\iota \times \text{list}) \to \text{list}\}$
- tree has $\{\text{E} : \text{tree}, \text{N} : (\text{tree} \times \iota \times \text{tree}) \to \text{tree}\}$

Any natural number is $s^k(0)$, any list is $t_1 :: t_2 :: \ldots :: t_k :: []$

Inductive theories (e.g., Peano axioms) defined on inductive types

$$0 + x \simeq x$$
$$s(x) + y \simeq s(x + y)$$
$$0 \times x \simeq 0$$
$$s(x) \times y \simeq y + x \times y$$

Definitions

An inductive type is generated from set of constructors

Example

- nat has constructors $\{0 : \text{nat}, s : \text{nat} \rightarrow \text{nat}\}$
- list has constructors $\{[\,] : \text{list}, (::) : (\iota \times \text{list}) \rightarrow \text{list}\}$
- tree has $\{\text{E} : \text{tree}, \text{N} : (\text{tree} \times \iota \times \text{tree}) \rightarrow \text{tree}\}$

Any natural number is $s^k(0)$, any list is $t_1 :: t_2 :: \ldots :: t_k :: [\,]$

Inductive theories (e.g., Peano axioms) defined on inductive types

$$
\begin{aligned}
0 + x &\simeq x \\
s(x) + y &\simeq s(x + y) \\
0 \times x &\simeq 0 \\
s(x) \times y &\simeq y + x \times y
\end{aligned}
$$

The Road Ahead

We will need 3 mechanisms:

1. reasoning by case (on the constructor)
2. using the induction hypothesis (in a refutation)
3. prove and use lemmas

We have 3 answers:

1. the AVATAR calculus [Voronkov 14]
2. inductive strengthening
3. an inference rule to introduce lemmas

The Road Ahead

We will need 3 mechanisms:

1. reasoning by case (on the constructor)
2. using the induction hypothesis (in a refutation)
3. prove and use lemmas

We have 3 answers:

1. the AVATAR calculus [Voronkov 14]
2. inductive strengthening
3. an inference rule to introduce lemmas

Prerequisite: AVATAR, not just a blue alien

Induction requires case analysis

Why AVATAR

- recent work [Voronkov 2014] to better handle *splitting* (efficient boolean disjunction)

→ *splitting* good for case analysis

- leverages powerful SAT-solvers
- makes clauses depend on propositional formulas

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)
$p \lor \neg q \lor \forall x.\ p(x)$	$\llbracket p \rrbracket \sqcup \neg \llbracket q \rrbracket \sqcup \llbracket p(x) \rrbracket$
$\forall x.\ \neg p(x) \lor \forall y\ z.\ q(y) \lor q(f(y,z))$	$\llbracket \neg p(x) \rrbracket \sqcup \llbracket q(y) \lor q(f(y,z)) \rrbracket$
$n_0 \simeq 0 \lor n_0 \simeq s(n_1)$	$\llbracket n_0 \simeq 0 \rrbracket \sqcup \llbracket n_0 \simeq s(n_1) \rrbracket$

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)
$p \vee \neg q \vee \forall x.\ p(x)$	$\llbracket p \rrbracket \sqcup \neg \llbracket q \rrbracket \sqcup \llbracket p(x) \rrbracket$
$\forall x.\ \neg p(x) \vee \forall y\ z.\ q(y) \vee q(f(y,z))$	$\llbracket \neg p(x) \rrbracket \sqcup \llbracket q(y) \vee q(f(y,z)) \rrbracket$
$n_0 \simeq 0 \vee n_0 \simeq s(n_1)$	$\llbracket n_0 \simeq 0 \rrbracket \sqcup \llbracket n_0 \simeq s(n_1) \rrbracket$

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

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

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

AVATAR Absurd

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

- 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

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

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

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

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

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

AVATAR Absurd

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

- 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

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

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

AVATAR Absurd

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

- 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

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

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

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 $\exists l_0 : \text{list}.\ \neg p(l_0)$
  - coverset: $l_0 \in \{[], t_0 :: l_1\}$

  → assert $(l_0 \simeq []) \lor (l_0 \simeq t_0 :: l_1)$ and deduce $\bot$ by case analysis

split:

$$\frac{l_0 \simeq [] \lor l_0 \simeq t_0 :: l_1}{l_0 \simeq [] \leftarrow \llbracket l_0 \simeq [] \rrbracket \quad l_0 \simeq t_0 :: l_1 \leftarrow \llbracket l_0 \simeq t_0 :: l_1 \rrbracket \quad \llbracket l_0 \simeq [] \rrbracket \sqcup \llbracket l_0 \simeq t_0 :: l_1 \rrbracket} \text{(ASplit)}$$

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 $\exists l_0 : \text{list}.\ \neg p(l_0)$
  - coverset: $l_0 \in \{[], t_0 :: l_1\}$

  → assert $(l_0 \simeq []) \vee (l_0 \simeq t_0 :: l_1)$ and deduce $\bot$ by case analysis

split:

$$\frac{l_0 \simeq [] \vee l_0 \simeq t_0 :: l_1}{l_0 \simeq [] \leftarrow \llbracket l_0 \simeq [] \rrbracket \quad l_0 \simeq t_0 :: l_1 \leftarrow \llbracket l_0 \simeq t_0 :: l_1 \rrbracket \quad \llbracket l_0 \simeq [] \rrbracket \sqcup \llbracket l_0 \simeq t_0 :: l_1 \rrbracket} \text{(ASplit)}$$

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 $\exists l_0 : \text{list}.\ \neg p(l_0)$
  - coverset: $l_0 \in \{[], t_0 :: l_1\}$

→ assert $(l_0 \simeq []) \vee (l_0 \simeq t_0 :: l_1)$ and deduce $\bot$ by case analysis

base case: easy

$$\dfrac{\dfrac{\dfrac{l_0 \simeq [] \leftarrow \llbracket l_0 \simeq [] \rrbracket \qquad \neg p(l_0)}{\neg p([]) \leftarrow \llbracket l_0 \simeq [] \rrbracket}\,(\text{Sup}) \qquad p([])}{\bot \leftarrow \llbracket l_0 \simeq [] \rrbracket}\,(\text{Res})}{\neg \llbracket l_0 \simeq [] \rrbracket}\,(\text{A}\bot)$$

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 $\exists l_0 : \text{list}.\ \neg p(l_0)$
  - coverset: $l_0 \in \{[], t_0 :: l_1\}$

→ assert $(l_0 \simeq []) \lor (l_0 \simeq t_0 :: l_1)$ and deduce $\bot$ by case analysis

recursive case:

$$\dfrac{\dfrac{l_0 \simeq t_0 :: l_1 \leftarrow \llbracket l_0 \simeq t_0 :: l_1 \rrbracket \qquad \neg p(l_0)}{\neg p(t_0 :: l_1) \leftarrow \llbracket l_0 \simeq t_0 :: l_1 \rrbracket}\ (\text{Sup}) \qquad \neg p(l) \lor p(x :: l)}{\neg p(l_1) \leftarrow \llbracket l_0 \simeq t_0 :: l_1 \rrbracket}\ (\text{Res})$$

$$\vdots$$

not enough!

Inductive Strengthening

Principle

- assume l_0 is a minimal counter-example to p
  (minimal w.r.t. subterm ordering $\lhd$)
  in other words: $\neg p(l_0)$ and $\forall l.\ l \lhd l_0 \Rightarrow p(l)$
- assert $p(l_1)$, since $l_1 \lhd l_0$ $(\simeq t_0 :: l_1)$ and l_0 minimal
- theorem: $\exists$ model iff $\exists$ model with $[\![l_0]\!]$ minimal

$$\dfrac{\dfrac{\dfrac{\dfrac{l_0 \simeq t_0 :: l_1 \leftarrow [\![l_0 \simeq t_0 :: l_1]\!] \qquad \neg p(l_0)}{\neg p(t_0 :: l_1) \leftarrow [\![l_0 \simeq t_0 :: l_1]\!]}\,(\text{Sup}) \qquad \neg p(l) \vee p(x :: l)}{\neg p(l_1) \leftarrow [\![l_0 \simeq t_0 :: l_1]\!]}\,(\text{Res}) \qquad p(l_1)}{\bot \leftarrow [\![l_0 \simeq t_0 :: l_1]\!]}\,(\text{Res})}{\neg [\![l_0 \simeq t_0 :: l_1]\!]}\,(\text{A}\bot)$$

Success, both $[\![l_0 \simeq [\,]]\!]$ and $[\![l_0 \simeq t_0 :: l_1]\!]$ are false!

Lemmas

lemmas sometimes required:

Example (Commutativity of +)

Assume $\forall x : \text{nat}.\ 0 + x \simeq x$ and $\forall x\ y : \text{nat}.\ s(x) + y \simeq s(x + y)$.
Proving $\forall x\ y : \text{nat}.\ x + y \simeq y + x$ requires:

- lemma $\forall x : \text{nat}.\ x + 0 \simeq x$
- lemma $\forall x\ y : \text{nat}.\ x + s(y) \simeq s(x + y)$

Lemma Intro Rule

introduce lemma F:

$$\frac{\top}{\begin{array}{ccc} & F & \leftarrow [\![F]\!] \\ \wedge & \neg F & \leftarrow \neg [\![F]\!] \end{array}}$$

Which lemmas to try: heuristics, theory detection (later)

Lemmas

lemmas sometimes required:

Example (Commutativity of +)

Assume $\forall x : \text{nat}.\ 0 + x \simeq x$ and $\forall x\ y : \text{nat}.\ s(x) + y \simeq s(x + y)$.
Proving $\forall x\ y : \text{nat}.\ x + y \simeq y + x$ requires:

- lemma $\forall x : \text{nat}.\ x + 0 \simeq x$
- lemma $\forall x\ y : \text{nat}.\ x + s(y) \simeq s(x + y)$

Lemma Intro Rule

introduce lemma F:

$$\frac{\top}{\begin{array}{ccc} & F & \leftarrow [\![F]\!] \\ \wedge & \neg F & \leftarrow \neg [\![F]\!] \end{array}}$$

Which lemmas to try: heuristics, theory detection (later)

Lemmas

lemmas sometimes required:

Example (Commutativity of +)

Assume $\forall x : \text{nat}.\ 0 + x \simeq x$ and $\forall x\ y : \text{nat}.\ s(x) + y \simeq s(x + y)$.
Proving $\forall x\ y : \text{nat}.\ x + y \simeq y + x$ requires:

- lemma $\forall x : \text{nat}.\ x + 0 \simeq x$
- lemma $\forall x\ y : \text{nat}.\ x + s(y) \simeq s(x + y)$

Lemma Intro Rule

introduce lemma F (and reduce it to CNF):

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

Which lemmas to try: heuristics, theory detection (later)

Summary of Induction

- extension of AVATAR with Inductive Strengthening
- able to prove and use lemmas
- compatible with Superposition + LIA, etc.

  e.g., prove $\mathrm{len}(\mathrm{dup}(l)) \simeq 2 \cdot \mathrm{len}(l)$
- implementation is only a prototype → hard to assess efficiency
- *further extension*: able to deal with multi-clauses properties

Limitations

- no nested induction (requires cut/lemma)
- mutually recursive types unsupported

  (e.g., a tree with a list of sub-trees)
- lemmas → mostly heuristic approach

Summary of Induction

- extension of AVATAR with Inductive Strengthening
- able to prove and use lemmas
- compatible with Superposition + LIA, etc.

  e.g., prove $\text{len}(\text{dup}(l)) \simeq 2 \cdot \text{len}(l)$
- implementation is only a prototype → hard to assess efficiency
- *further extension*: able to deal with multi-clauses properties

Example (A few other examples)

goal	notes
$x_1 + (x_2 + (\ldots + x_n)) \simeq x_n + (x_{n-1} + (\ldots + x_1))$	3 lemmas, 0.16s
$x \leq y \Rightarrow x + z \leq y + z$	2 lemmas, 0.7s
$l_1 @ (l_2 @ l_3) \simeq (l_1 @ l_2) @ l_3$	0.16s
$\text{len}(l_1 @ l_2) \simeq \text{len}(l_1) + \text{len}(l_2)$	0.15s

Summary

What is Theory Detection?

Mathematicians don't deduce blindly, unlike ATP.

Goals of Theory Detection

- finding what the problem is about:
  is it about groups? rings? lists?
- introduce lemmas for known theories
- introduce rewrite rules, decision procedures, heuristics, etc.

We will see:

- how axioms and theories are described
- how to use this information (lemmas. . .)
- how it works

Theory Detection

Meta-level: describe axioms and theories in a TPTP-like language:

Example (Group Theory)

```
axiom (associative F) <-
  holds (![X,Y,Z]: [F X (F Y Z) = F (F X Y) Z]).

axiom (left_identity {op=Mult, elem=E}) <-
  holds (![X]: [Mult E X = X]).

axiom (left_inverse {op=Mult, inverse=I, elem=E}) <-
  holds (![X]: [Mult (I X) X = E]).

theory (group {op=Mult, neutral=E, inverse=I}) <-
  axiom (associative Mult),
  axiom (left_inverse {op=Mult, inverse=I, elem=E}),
  axiom (left_identity {op=Mult, elem=E}).
```

ex: `theory (group {op=(+), neutral=0, inverse=(-)})` on $\mathbb{Z}$, or `theory (group {op=(×), neutral=1, inverse=(/)})` on $\mathbb{Q} \setminus \{0\}$

Applications

Example (Umangling Functional Relations)

Translate relational representations into functional ones
(better for Superposition; gain perf on some TPTP problems)

```
axiom (functional2 P) <-
  holds (![X,Y,Z]: [~ (P X Y Z), ~ (P X Y Z2), Z = Z2]).

axiom (total2 {pred=P, fun=F}) <-
  holds (![X,Y]: [P X Y (F X Y)]).

rewrite (![X,Y,Z]: [P X Y Z --> (Z = F X Y)]) <-
  axiom (functional2 P),
  axiom (total2 {pred=P, fun=F}).
```

Example of Application: Inductive Lemmas

Idea: suggest lemmas for known inductive theories

Represent Inductive Type at meta-level

Example: nat is inductive with constructors s and 0

$$\text{inductive}_{\langle \text{nat} \rangle} \ \{\!|\text{ty} = \text{nat}, \text{cstors} = [(\text{cstor}_{\langle \text{nat} \to \text{nat} \rangle} \ s), (\text{cstor}_{\langle \text{nat} \rangle} \ 0)]\,|\!\}$$

Example (Peano Numbers)

Lemma $\forall x \ y : \text{nat}. \ x + s(y) \simeq s(x+y)$:

```
theory (peano_add {succ=S, zero=Z, plus=P}) <-
  inductive @N {ty=@N, cstors=[(cstor _ S), (cstor _ Z)]}.
  holds (![X:N,Y:N]: [P (S X) Y = S (P X Y)]),
  holds (![X:N]: [P Z X = X]).

lemma (![X:N,Y:N]: [P X (S Y) = S (P X Y)]) <-
  theory (peano_add {succ=S, zero=_, plus=P}).
```

Meta Prover: the Inside

represents and infers properties *about* the problem

Meta-Level Reasoner

- higher-order language (∀,∃, application, extensible records, multisets, no λ, polymorphic types)

→ type inference and unification decidable (no λ)

- Horn clauses of the form $A \leftarrow B_1, \ldots, B_n$
- saturate by resolution (bottom-up prolog)
- scan FO clauses to detect instances of axioms, then saturate

Summary of Theory Detection

- Describe axioms, theories,. . . in HO terms
- Hook lemmas or rewrite rules to theories
- Plugins, e.g. for inductive types
- Implemented in Logtk, used in Zipperposition

Example (Inductive Lemma)

Zipperposition can prove inductively

$$double(n) \simeq n + n$$

with lemma $\forall x\ y.\ x + s(y) \simeq s(x + y)$, where

$$double(0) \simeq 0$$
$$double(s(n)) \simeq s(s(double(n)))$$

Summary

Overview

Linear Integer Arithmetic

- purely deductive system, using unification
- implementation: quite competitive (CASC, TPTP)

Structural Induction

- inductive strengthening
- introduce lemmas
- implementation: prototype only

But Also...

- Theory Detection [Burel&Cruanes 2013]
- polymorphism
- full implementation of a prover (Zipperposition)

Perspectives

Many possible extensions:

- combining LIA with hierarchic superposition
- achieve completeness on LIA
- thorough implementation of induction
- integration with proof assistants (Sledgehammer, Coq. . .)
- superdeduction and theory of (typed) sets (with David Delahaye)

Thank you for your attention.

Rules of Superposition

Superposition (Sup)

$$\frac{C \vee s \simeq t \qquad D \vee u\left[s_2\right]_p \circ v}{(C \vee D \vee u[t]_p \circ v)\sigma}$$

where $s\sigma = s_2\sigma$, $\circ \in \{\simeq, \not\simeq\}$

Equality Resolution (EqRes)

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

where $s\sigma = t\sigma$

Equality Factoring (EqFact)

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

where $s\sigma = t\sigma$

Superposition is only three rules (some details omitted)

- σ is a substitution
- C, D are clauses (disjunctions)
- $u[t]_p$ puts t at position p in term u

Rules of LIA

$$\frac{C \vee a \cdot t + u \simeq v \qquad C' \vee a' \cdot t + u' \sim v'}{C \vee C' \vee \varphi' \cdot u + \varphi \cdot v' \sim \varphi \cdot u' + \varphi' \cdot v} \text{ (CSup)}$$

$$\frac{C \vee a \cdot t + u \simeq v \vee a' \cdot t + u' \simeq v'}{C \vee \varphi \cdot u + \varphi' \cdot v' \not\simeq \varphi' \cdot u' + \varphi \cdot v \vee a' \cdot t + u' \simeq v'} \text{ (CFact}\simeq\simeq\text{)}$$

$$\frac{C \vee a \cdot t + u \ \sim \ a' \cdot t + v}{C \vee (a - a') \cdot t + u \ \sim \ v} \text{ (Canc)} \quad \text{and} \quad \frac{C \vee d^k \mid d^k \cdot t + u}{C \vee d^k \mid u} \text{ (Canc)}$$

$$\frac{C \vee v \leq a \cdot t + u \qquad C' \vee a' \cdot t + u' \leq v'}{C \vee C' \vee \varphi \cdot v + \varphi' \cdot u' \leq \varphi' \cdot v' + \varphi \cdot u} \text{ (Chain}\leq\text{)}$$

$$\frac{C \vee v \leq a \cdot t + u \qquad C' \vee a' \cdot t + u' \leq v'}{C \vee C' \vee \bigvee_{i=0}^{k} (\varphi \times a) \cdot t + \varphi \cdot u \simeq \varphi \cdot v + i \cdot \mathbf{1}} \text{ (CSwitch)}$$

$$\frac{C \vee \left\{ \begin{array}{ll} & a \cdot t + u \lesseqgtr v \\ \text{or} & a \cdot t + u \simeq v \end{array} \right\} \vee a' \cdot t + u' \lesseqgtr v'}{C \vee \varphi \cdot u + \varphi' \cdot v' + \mathbf{1} \lesseqgtr \varphi \cdot v + \varphi' \cdot u' \vee a' \cdot t + u' \lesseqgtr v'} \text{ (CFact}\leq\text{)}$$

where $\varphi \times a = \varphi' \times a' = \mathrm{lcm}(a, a')$

Rules of LIA (continued)

$$\frac{C \vee d^e \mid a \cdot t + u \qquad C' \vee d^{e+k} \mid a' \cdot t + u'}{C \vee C' \vee d^{e+k} \mid (\varphi \times d^k) \cdot u - \varphi' \cdot u'} \text{ (Chain|)}$$

where $\varphi \times (a \times d^k) = \varphi' \times a' = \operatorname{lcm}(a \times d^k, a') < d^{e+k}$

$$\frac{C \vee d^e \mid a' \cdot t + u' \vee d^{e+k} \mid a \cdot t + u}{C \vee d^{e+k} \nmid \varphi \cdot u - (d^k \times \varphi') \cdot u' \vee d^{e+k} \mid a \cdot t + u} \text{ (CFact||)}$$

where $\varphi \times a = \varphi' \times a' = \operatorname{lcm}(a, a')$, $\gcd(a', d^e) \cdot d^k \mid \gcd(a, d^{e+k})$

$$\frac{C \vee a \cdot t + u \simeq v \vee d^e \mid a' \cdot t + u'}{C \vee d^e \nmid \varphi \cdot v + \varphi' \cdot u' - \varphi \cdot u \vee d^e \mid a' \cdot t + u'} \text{ (CFact|}\simeq\text{)}$$

where $\gcd(a, d^e) \mid \gcd(a', d^e)$, $\varphi \cdot a = \varphi' \cdot a'$

$$\frac{C \vee a \cdot t + u \simeq v}{C \vee a \mid u - v} \text{ (CDiv)} \quad \text{and} \quad \frac{C \vee d^{k+k'} \mid (b \times d^k) \cdot t + u}{C \vee d^k \mid u} \text{ (CDiv)}$$

where $k \geq 1, k' \geq 1,\ a \geq 2,\ b \geq 1$

Variable Elimination

Let $C \stackrel{\text{def}}{=} C' \vee \bigvee_{i=1}^{k} l_i[x]$, then $C \equiv C' \vee \neg\left(\exists x.\ \bigwedge_{i=1}^{k} \neg l_i[x]\right)$, then

$$\text{elim}_x(C) = \bigcup_{n=1}^{\delta} \left\{ C' \vee G_{\infty}^{T}[-n] \right\} \cup \bigcup_{n=1}^{\delta} \bigcup_{j \in A} \left\{ C' \vee G^{T}[j-n] \right\}$$

where

$A \stackrel{\text{def}}{=} \{v_{e_m} - u_{e_m}\}_m \cup \{v_{a_i} - u_{a_i} + \mathbf{1}\}_i \cup \{v_{b_j} - u_{a_j}\}_j$

$$G_{-\infty}[x] = \begin{cases} \bot & \text{if } \{a_i[x']\}_i \cup \{f_n[x']\}_n \neq \emptyset \\ \bigwedge_{k,l}(n_{c_k} \mid u_{c_k} + x \wedge n_{d_l} \nmid u_{d_l} + x) \end{cases}$$

$$G_{\infty}[x] = \begin{cases} \bot & \text{if } \{a_i[x']\}_i \cup \{e_m[x']\}_m \neq \emptyset \\ \bigwedge_{k,l}(n_{c_k} \mid u_{c_k} + x \wedge n_{d_l} \nmid u_{d_l} + x) \end{cases}$$

$$a_i[x'] \stackrel{\text{def}}{=} x' + u_{a_i} \simeq v_{a_i}$$

$$b_j[x'] \stackrel{\text{def}}{=} x' + u_{b_j} \not\simeq v_{b_j}$$

$$c_k[x'] \stackrel{\text{def}}{=} n_{c_k} \mid x' + u_{c_k}$$

$$d_l[x'] \stackrel{\text{def}}{=} n_{d_l} \nmid x' + u_{d_l}$$

$$e_m[x'] \stackrel{\text{def}}{=} x' + u_{e_m} < v_{e_m}$$

$$f_n[x'] \stackrel{\text{def}}{=} u_{f_n} < x' + v_{f_n}$$

Incompleteness of LIA [Waldmann]

Assuming $a \succ b \succ c \succ d \succ e$, the clauses

$$7 \mid a \quad 7 \mid b$$
$$a \leq b \quad b \leq a + c$$
$$2 \cdot c + d \simeq e \vee 2 \cdot c + d \simeq e + 4 \vee e \leq d$$
$$d + 2 \simeq e \vee d + 4 \simeq e$$

- unsatisfiable: two last clauses imply $\bigvee_{i=1}^{4} c \simeq i$ (by case on the last clause)
- no $\{c \simeq i\}_{i=1}^{4}$ is generated, because of $\succ$

→ case switch between $a \leq b$ and $b \leq a + c$ not performed

→ refutation not reached

Experimental Evaluation of LIA on TPTP

Benchmarks from ARI,NUM,GEG,PUZ,SEV,SYN,SYO					
prover	unsat (/263)	%solved	unique	time (s)	avg time (s)
beagle	254	97	6	321	1.27
princess	251	95	0	229	0.91
zip	247	94	0	53	0.22

Benchmarks from DAT					
prover	unsat (/87)	%solved	unique	time (s)	avg time (s)
beagle	75	86	5	223	2.98
princess	60	69	1	326	5.44
zip	74	85	5	85	2.03

Benchmarks from SWV,SWW					
prover	unsat (/179)	%solved	unique	time (s)	avg time (s)
beagle	81	45	0	1432	17.6
princess	178	99	56	917	05.1
zip	52	29	0	1599	30.7

→ Decent performance overall

Unification Algorithms `unify_self_f`

```
let rec iter_self σ c t l m = match l with
| [] →
    return ({coeff=c; term=t; rest=m}, σ)
| (c2, t2) :: l2 →
    (* must merge, t = t2 *)
    if tσ = t2σ then iter_self σ (c + c2) t l2 m
    else (
      (* we can choose not to unify t and t2. *)
      iter_self σ c t l2 (add c2 t2 m) @
      try (* try to unify t and t2 *)
        let ρ = unify σ t t2 in
        let m2 = {m with terms=[]} in (* might have to merge *)
        iter_self ρ (c + c2) t (l2 @ m.terms) m2
      with Fail → empty
    )

let unify_self_f σ mf =
    let m = mf.rest in  (* unfocused part *)
    iter_self σ mf.coeff mf.term m.terms {m with terms=[]}
```

Splitting Clauses

In an inference

$$\frac{A_1 \lor \ldots \lor A_n \qquad B_1 \lor \ldots \lor B_m}{(A_2 \lor \ldots \lor A_n \lor B_2 \lor \ldots \lor B_m)\sigma} \text{ (Res) } \text{(where } A_1\sigma = (\neg B_1)\sigma\text{)}$$

conclusion has $m+n-2$ literals → **clauses grow**

big clauses → more memory, duplicated work. . .

Splitting Clauses (continued)

Idea: split a clause into *components* that share no variable

Example

clause	propositional clause
$p \vee \neg q \vee p(x)$	$p \vee \neg q \vee \forall x.\ p(x)$
$\neg p(x) \vee q(y) \vee q(f(y,z))$	$\forall x.\ \neg p(x) \vee \forall y\ z.\ q(y) \vee q(f(y,z))$
$x \leq 3 \vee 2 \cdot x \geq 8$	$\forall x.\ x \leq 3 \vee 2 \cdot x \geq 8$

$$\frac{p \vee \neg q \vee \forall x.\ p(x)}{p \leftarrow [\![p]\!] \mid \neg q \leftarrow \neg [\![q]\!] \mid \forall x.\ p(x) \leftarrow [\![p(x)]\!]} \text{ (ASplit)}$$

Choice between p, $\neg q$ and $\forall x.\ p(x)$ done by SAT-solver

→ three unit clauses instead of 1 ternary clause

Splitting Clauses (continued)

Idea: split a clause into *components* that share no variable

Example

clause	propositional clause
$p \lor \neg q \lor p(x)$	$p \lor \neg q \lor \forall x.\ p(x)$
$\neg p(x) \lor q(y) \lor q(f(y,z))$	$\forall x.\ \neg p(x) \lor \forall y\ z.\ q(y) \lor q(f(y,z))$
$x \leq 3 \lor 2 \cdot x \geq 8$	$\forall x.\ x \leq 3 \lor 2 \cdot x \geq 8$

$$\frac{p \lor \neg q \lor \forall x.\ p(x)}{p \leftarrow \llbracket p \rrbracket \mid \neg q \leftarrow \neg \llbracket q \rrbracket \mid \forall x.\ p(x) \leftarrow \llbracket p(x) \rrbracket} \text{(ASplit)}$$

Choice between p, $\neg q$ and $\forall x.\ p(x)$ done by SAT-solver

→ three unit clauses instead of 1 ternary clause

Multi-Clauses Induction

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

Polymorphism in Meta-Prover

informal definition	encoding
$[] \mathbin{@} l \simeq l$	$\forall \alpha.\ \forall l : \mathsf{list}(\alpha).\ \dot{\simeq}_\alpha\ [[]_{\langle\alpha\rangle} \mathbin{@}_{\langle\alpha\rangle} l, []_{\langle\alpha\rangle}]$
$(x :: l_1) \mathbin{@} l_2 \simeq x :: (l_1 \mathbin{@} l_2)$	$\forall \alpha.\ \forall x : \alpha.\ \forall l_1\ l_2 : \mathsf{list}(\alpha).$ $\dot{\simeq}_\alpha\ [(x ::_{\langle\alpha\rangle} l_1) \mathbin{@}_{\langle\alpha\rangle} l_2, x ::_{\langle\alpha\rangle} (l_1 \mathbin{@}_{\langle\alpha\rangle} l_2)]$
$\mathrm{rev}([]) \simeq []$	$\forall \alpha.\ \dot{\simeq}_\alpha\ [\mathrm{rev}_{\langle\alpha\rangle}([]_{\langle\alpha\rangle}), []_{\langle\alpha\rangle}]$
$\mathrm{rev}(x :: l) \simeq \mathrm{rev}(l) \mathbin{@} (x :: [])$	$\forall \alpha.\ \forall x : \alpha.\ \forall l : \mathsf{list}(\alpha).\ \dot{\simeq}_\alpha$ $\left[\begin{array}{l} \mathrm{rev}_{\langle\alpha\rangle}\ (x ::_{\langle\alpha\rangle} l), \\ (\mathrm{rev}_{\langle\alpha\rangle}\ l) \mathbin{@}_{\langle\alpha\rangle} (x ::_{\langle\alpha\rangle} []_{\langle\alpha\rangle}) \end{array} \right]$

HO Unit Resolution

Assume F ground fact (axiom instance, ...)

$$\frac{F \qquad A \leftarrow B_1, \ldots, B_n}{A\sigma \leftarrow B_2\sigma, \ldots, B_n\sigma}$$

where $B_1\sigma = F$

We assume *safe* Horn Clauses:

$$\text{freevars}(A) \subseteq \bigcup_{i=1}^{n} \text{freevars}(B_i)$$