Combining Superposition and Induction: a Practical Realization

Abdelkader Kersani and Nicolas Peltier

Laboratoire d'Informatique de Grenoble/CNRS
CAPP team - ASAP project (ANR-09-BLAN-0407-01)

FROCOS 2013 - September 2013 - Nancy

Introductory example

$$\text{length\_at\_least}(l, n) \Leftrightarrow n = 0 \vee \exists x, l', n' \, (l = cons(x, l') \wedge n = s(n') \wedge \text{length\_at\_least}(l', n'))$$

$$\text{nth}(x, l, n) \Leftrightarrow \exists l' \, l = cons(y, l') \wedge (n = s(0) \wedge x = y) \vee \exists n' \, (n = s(n') \wedge \text{nth}(x, l', n'))$$

Check that the following holds:
$\forall n \in \mathbb{N}, \forall l \, (\text{length\_at\_least}(l, n) \wedge n \neq 0 \Rightarrow \exists x \, \text{nth}(x, l, n))$

Introductory example (2)

- This problem cannot be stated in first-order logic ($n \in \mathbb{N}$)
- An inductive property of the form $\forall n, l \, \exists x \, \phi$
- Must combine:
  - Standard equational reasoning with unification to:
    1. Find the value of x (w.r.t. n, l)
    2. Check that it indeed fulfills the desired property
  - Inductive reasoning on n

Introductory example (3)

- Straightforward approach: use standard proof procedures for first-order logic together with explicit induction schemes

$$(\psi(0) \wedge \forall n\, \psi(n) \Rightarrow \psi(s(n))) \Rightarrow \forall n\, \psi(n)$$

for some "well-chosen" formula ψ

- Our approach: try to discover automatically such inductive lemmata, by detecting cycles in the search space

Plan of the talk

- The language
- A proof procedure: superposition + loop detection
- A cycle detection algorithm
- Experimentations

The language

Clausal (first-order) logic + a (unique) arithmetic parameter n

- Two sorts ι (standard terms) and ω (natural numbers), with $0 : \omega, s : \omega \to \omega$
- A special constant symbol n denoting a natural number
- Terms, (equational) literals and clauses are defined as usual
  do *not* contain the special symbol n
- n-clauses: constrained clauses of the form

$$[C \mid \mathcal{X}]$$

where:

- C is a clause
- $\mathcal{X}$ is of the form $\bigwedge_{i=1}^{k} n = t_i$, where $t_1, \ldots, t_k$ $(k \geq 0)$ are terms of sort ω

Semantics

- The special symbol n is interpreted as a term of the form $s^m(0)$ ($m \in \mathbb{N}$)
- 0 and s are interpreted as 0 and successor function
- The other symbols are interpreted as usual
- $[C \mid \bigwedge_{i=1}^{k} n = t_i]$ holds in I iff for every substitution σ such that $I(n) = t_i\sigma$, $C\sigma$ holds in I

The language (2)

Remarks:

- A strict extension of first-order logic
- The constant n does not occur in the clauses
  A formula of the form $f(n) = a$ must be written:

$$[f(x) = a \mid n = x]$$

- Extension to formulæ with several parameters

A theoretical limitation

Theorem

The set of satisfiable sets of n-clauses is neither recursively enumerable (of course !) nor co-recursively enumerable

Depart from:

- First-order logic (unsatisfiability is semi-decidable)
- Rewrite-based inductive theorem proving (non-provability is semi-decidable)

The language (3)

Proposition

Every (non-tautological) n-clause is equivalent to an n-clause of the form $[C \mid \top]$ or $[C \mid n = t]$

Proof: $\bigwedge_{i=1}^{k} n = t_i \Leftrightarrow n = t_1 \wedge \bigwedge_{i=2}^{k} t_1 = t_i$, thus

$$[C \mid \bigwedge_{i=1}^{k} n = t_i] \Leftrightarrow [C\sigma \mid n = t_1\sigma]$$

where $\sigma = \mathrm{mgu}(t_1, \ldots, t_k)$ and

$$[C \mid \bigwedge_{i=1}^{k} n = t_i] \Leftrightarrow \top$$

if $t_1, \ldots, t_k$ are not unifiable

Example

$$[f(x, y) = a \mid n = s(z) \wedge n = x \wedge n = y]$$

$$\longrightarrow [f(s(z), s(z)) = a \mid n = s(z)]$$

$$[f(x, y) = a \mid n = s(x) \wedge n = 0]$$

$$\longrightarrow \top$$

The language (4)

3 kinds of n-clauses:

1. Standard first-order clauses: express universal properties, not depending on the value of n
2. $[C \mid n = s^k(0)]$: expresses a property that holds only if n has some specific value $(n = k)$
3. $[C[x] \mid n = s^k(x)]$: expresses a property C that holds for $x = n - k$

The language (4)

3 kinds of n-clauses:

1. Standard first-order clauses: express universal properties, not depending on the value of n rank $\bot$
2. $[C \mid n = s^k(0)]$: expresses a property that holds only if n has some specific value $(n = k)$ no rank
3. $[C[x] \mid n = s^k(x)]$: expresses a property C that holds for $x = n - k$ rank k

$S[i]$ denotes the set of n-clauses of rank i in S

Proof Procedure: Constrained superposition calculus

Superposition:

$$\frac{[C \vee t \bowtie s \mid \mathcal{X}], [D \vee u = v \mid \mathcal{Y}]}{[C \vee D \vee t[v]_p \bowtie s \mid \mathcal{X} \wedge \mathcal{Y}]\sigma}$$

If $\bowtie \in \{=, \neq\}$, $\sigma = \mathrm{mgu}(u, t|_p)$, $u\sigma \not\leq v\sigma, t\sigma \not\leq s\sigma$, $t|_p$ is not a variable, $(t \bowtie s)\sigma \not< C\sigma$, $(u = v)\sigma \not< D\sigma$.

Proof Procedure: Constrained superposition calculus (2)

Reflection:

$$\frac{[C \vee t \neq s \mid \mathcal{X}]}{[C \mid \mathcal{X}]\sigma}$$

If $\sigma = \text{mgu}(t, s)$, $(t \neq s)\sigma \not< C\sigma$

Factorisation:

$$\frac{[C \vee t = s \vee u = v \mid \mathcal{X}]}{[C \vee s \neq v \vee t = s \mid \mathcal{X}]\sigma}$$

If $\sigma = \text{mgu}(t, u)$, $t\sigma \not< s\sigma$, $u\sigma \not< v\sigma$, $(t = s)\sigma \not< C\sigma$.

Proof Procedure

Remarks:

- The parameter n is abstracted away from the clauses: $f(n) = a \longrightarrow [f(x) = a \mid n = x]$
- Allows for a lazy instantiation of this parameter: $[f(x) = a \mid n = x], f(0) \neq a \vdash [\Box \mid n = 0]$
- "Weakly" complete: if $S \models n \neq k$ (for some $k \in \mathbb{N}$) then $S \vdash [\Box \mid n = k]$ (modulo subsumption)
- Not complete: no contradiction is derived in finite time (almost never terminates)

A trivial example

Prove the following:

$$p(0) \wedge \forall x\, p(x) \Rightarrow p(s(x)) \models \forall n \in \mathbb{N}\, p(n)$$

A trivial example (2)

Use the superposition calculus:

1	$p(0) = \texttt{true}$	
2	$p(x) \neq \texttt{true} \vee p(s(x)) = \texttt{true}$	
3	$[p(x) \neq \texttt{true} \mid n = x]$	
4	$[\square \mid n = 0]$	(superposition, 1, 3)
5	$[p(x) \neq \texttt{true} \mid n = s(x)]$	(superposition, 2, 3)
6	$[\square \mid n = s(0)]$	(superposition, 1, 5)
...	...	
	$[\square \mid n = s^k(0)]$	

Uncomplete calculus

If S is unsatisfiable, we have:

$$\forall k \in \mathbb{N}\, S \vdash n \neq k$$

but not:

$$S \vdash \forall k \in \mathbb{N}\, n \neq k$$

$$(\equiv \bot)$$

A trivial example (3)

A "cycle" in the search space:
Clause 5 : $[p(x) \neq \texttt{true} \mid n = s(x)]$ is almost identical to Clause 3 : $[p(x) \neq \texttt{true} \mid n = x]$, up to a translation on n.
Clause 3 $\equiv p(n)$
Clause 5 $\equiv p(n-1)$
Idea: detect those cycles and use them to prune the search space

Cycle Detection

First step: Formalize the notion of translation

- $S{\downarrow_i} \equiv S\{n \leftarrow n - i\}$
- $[C \mid n = t] \longrightarrow [C \mid n - i = t]$

First step: Formalize the notion of translation

- $S{\downarrow_i} \equiv S\{n \leftarrow n - i\}$
- $[C \mid n = t] \longrightarrow [C \mid n - i = t]$

Cycle Detection

First step: Formalize the notion of translation

- $S{\downarrow_i} \equiv S\{n \leftarrow n - i\}$
- $[C \mid n = t] \longrightarrow [C \mid n = t + i]$

Cycle Detection

First step: Formalize the notion of translation

- $S{\downarrow_i} \equiv S\{n \leftarrow n - i\}$
- $[C \mid n = t] \longrightarrow [C \mid n = s^i(t)]$

Cycle Detection

Second step:

Cycle Detection Rule

If there exists $S_{ind} \subseteq S$ such that:

1. $S_{ind} \models n \neq l$, for every $l \in [i, i+j[$
2. and $S_{ind} \models S_{ind}{\downarrow_j}$,

then $S \models n < i$ (i.e. $S \models [\square \mid n = s^i(x)]$)

Proof: by "descente infinie"

Cycle Detection

In practice:

- S is the whole search space (set of generated n-clauses)
- $S_{ind} \subseteq S$
- Decidable conditions are needed

Cycle Detection

In practice:

- S is the whole search space (set of generated n-clauses)
- $S_{ind} \subseteq S$
- Decidable conditions are needed
  - Condition 1: Check that $[\Box \mid n = l]$ has been derived from S_{ind}
  - Condition 2: Check that some set of n-clauses S_{loop} has been derived from S_{ind}, with $S_{loop} = S_{ind}{\downarrow_j}$

Cycle Detection

In practice:

- S is the whole search space (set of generated n-clauses)
- $S_{ind} \subseteq S$
- Decidable conditions are needed
  - Condition 1: Check that $[\Box \mid n = l]$ has been derived from S_{ind}
  - Condition 2: Check that some set of n-clauses S_{loop} has been derived from S_{ind}, with S_{loop} subsumes $S_{ind}\downarrow_j$

Cycle Detection

In practice:

- S is the whole search space (set of generated n-clauses)
- $S_{ind} \subseteq S$
- Decidable conditions are needed
  - Condition 1: Check that $[\square \mid n = l]$ has been derived from S_{ind}
  - Condition 2: Check that some set of n-clauses S_{loop} has been derived from S_{ind}, with S_{loop} subsumes $S_{ind}{\downarrow_j}$ (or any decidable entailment relation)

Cycle Detection

In practice:

- S is the whole search space (set of generated n-clauses)
- $S_{ind} \subseteq S$
- Decidable conditions are needed
  - Condition 1: Check that $[\Box \mid n = l]$ has been derived from S_{ind}
  - Condition 2: Check that some set of n-clauses S_{loop} has been derived from S_{ind}, with S_{loop} subsumes $S_{ind}\downarrow_j$ (or any decidable entailment relation)
  - A further restriction: assume that all n-clauses in S_{ind} have the same rank i (or $\perp$)

Example (continued)

1	$p(0) = \texttt{true}$	
2	$p(x) \neq \texttt{true} \vee p(s(x)) = \texttt{true}$	
3	$[p(x) \neq \texttt{true} \mid n = x]$	
4	$[\square \mid n = 0]$	(superposition, 1, 3)
5	$[p(x) \neq \texttt{true} \mid n = s(x)]$	(superposition, 2, 3)
6	$[\square \mid n = s(0)]$	(superposition, 1, 5)
...	...	
	$[\square \mid n = s^k(0)]$	

- $S_{ind} = \{1, 2, 3\}$, $S_{loop} = \{1, 2, 5\}$, $i = 0$, $j = 1$
- $[\square \mid n = x]$ can be derived
- Unsatisfiability is detected

Cycle Detection in Practice

How to generate effectively the numbers i, j and the sets S_{ind}, S_{loop} ?

An algorithm to compute S_{ind}, S_{loop} (for fixed i, j)

Properties:

- Sound: the computed sets S_{ind}, S_{loop} satisfy the desired property
- Complete: if some sets S_{ind}, S_{loop} satisfy the desired property, then the algorithm succeeds (but not necessarily with output S_{ind}, S_{loop})
- Efficient: polynomial w.r.t. the size of the set S
- Based on a greatest fixpoint computation

The Algorithm

$S_0 \leftarrow \{n \neq k, k \in [i, i+j[\}$
$S_{ind} \leftarrow S[i]$
if $S_{ind} \not\models S_0$ **then**
  return `false`
end if
$S_{loop} \leftarrow \{\mathcal{D} \in S[i+j] \mid S_{ind} \vdash \{\mathcal{D}\}\}$
while $\exists \mathcal{C} \in S_{ind} \mid S_{loop} \not\supseteq \{\mathcal{C}\downarrow_j\}$ **do**
  $S_{ind} \leftarrow S_{ind} \setminus \{\mathcal{C}\}$
  if $S_{ind} \not\models S_0$ **then**
    return `false`
  end if
  Remove from S_{loop} all the n-clauses $\mathcal{D}$ s.t. $S_{ind} \not\models \{\mathcal{D}\}$
end while
return `true`

Implemented in Prover9

Use n-clauses to model *schemata of formulæ*

- Formulæ depending on some parameter n
- Constructed using special connectives $\bigvee_{i=a}^{b} \phi$ and $\bigwedge_{i=a}^{b} \phi$

Example: n-bit adder

$$Sum_i(p, q, c, r) \stackrel{\text{def}}{=} r_i \Leftrightarrow (p_i \oplus q_i) \oplus c_i$$

$$Carry_i(p, q, c) \stackrel{\text{def}}{=} c_{i+1} \Leftrightarrow (p_i \wedge q_i) \vee (c_i \wedge p_i) \vee (c_i \wedge q_i)$$

$$Adder(p, q, c, r) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} Sum_i(p, q, c, r) \wedge \bigwedge_{i=1}^{n} Carry_i(p, q, c) \wedge \neg c_1$$

Translation into clausal form (1)

$$\bigvee_{i=0}^{n} \phi \longrightarrow p(n)$$

with:

$$p(0) \Leftrightarrow \phi\{i \to 0\}$$

$$\forall x\, p(x+1) \Leftrightarrow \phi\{i \to x+1\} \vee p(x)$$

$$\bigvee_{i=a}^{n+b} \phi \longrightarrow \bigvee_{i=0}^{n} (\phi \wedge q_i) \vee \phi\{i \to n+1\} \vee \ldots \vee \phi\{i \to n+b\}$$

with:

$$\neg q(0) \wedge \ldots \wedge \neg q(a-1) \wedge q(a)$$

$$\forall x\, q(x) \to q(s(x))$$

Translation into clausal form (E)

Eliminate terms of the form $s(t)$ where t is not a variable:

$$p_{s(t)} \longrightarrow p'_t$$

with:

$$\forall x\, p_{s(x)} \Leftrightarrow p_x$$

Experimentations

Example	Time	# of calls to Cycle_2	# clauses
Ripple-carry adder ($A + 0 = A$)	0.48	336	33833
Ripple-carry adder (commutativity)	0.03	102	2003
Ripple-carry adder (associativity)	0.09	207	10154
Unicity of the result (ripple-carry)	0.7	150	50901
Carry-propagate adder (commutativity)	0.02	14	1980
Carry-propagate adder (associativity)	0.01	20	3972
Equivalence between the ripple-carry and the carry-propagate adders	0.03	14	1980
Totality of $<$ $(n_1 \geq n_2 \vee n_1 < n_2)$	0.01	47	185

Summary

- A technique to combine superposition calculus and inductive theorem proving
- Automated discovery of (some) inductive invariants
- Completeness can be ensured in some cases (CADE), e.g. if the formulæ contain no non-arithmetic variable (schemata of propositional formulæ)
- An implementation based on Prover9

Future Work

- Incremental loop detection
- Heuristics to "guess" the values of i and j or to trigger the application of the loop detection rule
- Improve the implementation, more experimentations