

Improvements in Formula Generalization

by using a **Decision Procedure**

Markus Aderhold

Fachgebiet Programmiermethodik
Technische Universität Darmstadt

Deduction and Decision Procedures
Dagstuhl-Seminar 07401, October 5, 2007



Outline

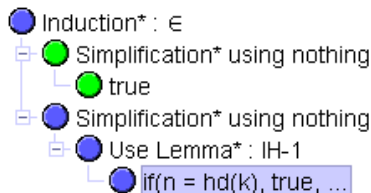
- 1 Motivation
- 2 Generalization Heuristics
 - Inverse Substitution
 - Inverse Weakening
 - Inverse Functionality
- 3 Evaluation



An Inductive Proof

lemma $n \in \text{isort}(k) \rightarrow n \in k \Leftarrow$

$\forall n : \mathbb{N}, k : \text{list}[\mathbb{N}]$ **if** $n \in \text{isort}(k)$ **then** $n \in k$ **else true end**



$$\overbrace{\{k \neq \varepsilon\}}^H, \overbrace{\{\text{if}(n' \in \text{tl}(k), \text{true}, n' \notin \text{isort}(\text{tl}(k)))\}}^{IH} \mapsto$$

$$\text{if}(n = \text{hd}(k), \text{true}, \text{if}(n \in \text{tl}(k), \text{true}, \text{if}(n \in \text{isort}(\text{tl}(k)), \text{true}, \dots)))$$

Problem: cannot be proved by another induction

Solution: generalize sequent and prove *generalization*



Contribution

Develop improved generalization heuristics:

- identify **promising** generalization candidates
- compute generalizations **more directly**
(reduce number of experimental attempts)
- get **more general (reusable)** generalizations
- avoid **over-generalizations**
(sparingly use a **disprover**)



Prerequisites

The theorem prover provides:

- recursion information $\mathcal{RP}(f)$ for each procedure f
- a **disprover** (to detect over-generalizations)
- an **induction heuristic** (nice to have, but not necessary)

Our approach does *not* require:

- **user interaction** or guidance
- a particular strategy to select **induction variables**
- a specific **rewriting strategy**



About \checkmark eriFun

\checkmark eriFun is

- a semi-automated **inductive theorem prover** for universal first-order formulas $\phi = \forall x_1 : \tau_1, \dots, x_n : \tau_n b$,
- designed for **program verification**.

Representation of verification problems:

- functional **programming language** \mathcal{L}
- **sequents** $\langle H, IH \mapsto goal \rangle$



A Simple \mathcal{L} -Program

Example

```

structure bool <= true, false
structure  $\mathbb{N}$  <= 0,  $^+ (^- : \mathbb{N})$ 
procedure [infix]  $>(x, y : \mathbb{N}) : \text{bool} <=$ 
if  $x = 0$ 
  then false
  else if  $y = 0$ 
    then true
    else  $^-(x) > ^-(y)$ 
  end
end

```

recursion position sets of $>$: $\mathcal{RP}(>) = \{\{1\}, \{2\}\}$



A Simple \mathcal{L} -Program (cont.)

Example

```

structure list[@A] <=  $\varepsilon$ , [infix] :: (hd : @A, tl : list[@A])
procedure insert(n :  $\mathbb{N}$ , k : list[ $\mathbb{N}$ ]) : list[ $\mathbb{N}$ ] <=
if k =  $\varepsilon$ 
  then n ::  $\varepsilon$ 
  else if n > hd(k)
    then hd(k) :: insert(n, tl(k))
    else n :: k
  end
end

procedure [infix]  $\in$ (x : @A, k : list[@A]) : bool <= ...
procedure isort(k : list[ $\mathbb{N}$ ]) : list[ $\mathbb{N}$ ] <= ...

```

recursion position sets of *insert*: $\mathcal{RP}(\textit{insert}) = \{\{2\}\}$



The Disprover

Problem

- given $\phi = \forall x_1 : \tau_1, \dots, x_n : \tau_n b$, find a **substitution** σ such that $\text{eval}_P(\sigma(b)) = \text{false}$
(interpreter $\text{eval}_P : \mathcal{T}(\Sigma(P)) \mapsto \mathcal{T}(\Sigma(P)^c)$)
- disproving ϕ is **semi-decidable**, but not decidable

Solution

- have predicate symbols $\doteq : @A \times @A$ and $\triangleright : \mathbb{N} \times \mathbb{N}$
- to disprove ϕ , solve $b \doteq \text{false}$
- **Phase I**: eliminate **type variables**, **if**, **=**, and **procedure function symbols**
- **Phase II**: decide if there is a substitution that satisfies the remaining literals



The Generalization Algorithm

Algorithm *generalize*($\langle H, IH \mapsto goal \rangle$)

$\langle H', IH' \mapsto goal' \rangle := \text{eliminateSelectors}(\langle H, \emptyset \mapsto goal \rangle)$

$\varphi := \text{s-eval}_P(\bigwedge H' \rightarrow goal')$

repeat

$\varphi := \text{s-eval}_P(\text{heuristicallyApplyGeneralizationRule}(\varphi))$

until no rule heuristically applicable

return φ

Generalization rules:

- **Inverse Substitution**: remove subterms
- **Inverse Weakening**: remove conditions
- **Inverse Functionality**: remove function symbols



Inverse Substitution

$$\frac{\varphi[x_1, \dots, x_n]}{\varphi[t_1, \dots, t_n]} \quad (\textit{substitution})$$

$$\frac{\varphi[t_1, \dots, t_n]}{\varphi[x_1, \dots, x_n]} \quad (\textit{inverse substitution})$$

Which subterms t_i of φ to generalize?

- selector elimination: e. g. $tl(k) \rightsquigarrow l$
- common non-variable subterms: e. g. $isort(l) \rightsquigarrow l'$
- generalize apart variable occurrences: e. g. $x \rightsquigarrow x'$



Selector Elimination

- **destructor style induction** for $\forall k : list[\mathbb{N}] \varphi[k]$:
 - prove $\langle \{k = \varepsilon\}, \emptyset \mapsto \varphi[k] \rangle$
 - prove $\langle \{k \neq \varepsilon\}, \{\varphi[tl(k)]\} \mapsto \varphi[k] \rangle$: reduce $\varphi[k]$ to $\varphi[tl(k)]$
- many **selector calls** $hd(k)$ and $tl(k)$
- **elimination**: replace k with $m :: l$, and simplify

Example

$$\langle \{m :: l \neq \varepsilon\}, \dots \mapsto \\ n \neq hd(m :: l) \wedge n \notin tl(m :: l) \wedge n \notin isort(tl(m :: l)) \rightarrow \\ n \notin insert(hd(m :: l), isort(tl(m :: l))) \rangle$$


Selector Elimination

- **destructor style induction** for $\forall k : list[\mathbb{N}] \varphi[k]$:
 - prove $\langle \{k = \varepsilon\}, \emptyset \mapsto \varphi[k] \rangle$
 - prove $\langle \{k \neq \varepsilon\}, \{\varphi[tl(k)]\} \mapsto \varphi[k] \rangle$: reduce $\varphi[k]$ to $\varphi[tl(k)]$
- many **selector calls** $hd(k)$ and $tl(k)$
- **elimination**: replace k with $m :: l$, and simplify
- **only if** $H \vdash k \neq \varepsilon$ \Rightarrow “safe”
and k does **not occur in a recursion position** \Rightarrow “useful”

Definition

t_i occurs in a **recursion position** in $f(t_1, \dots, t_n)$ if there is some $l \in \mathcal{RP}(f)$ such that $i \in l$.

$$insert(t_1, t_2) \quad \mathcal{RP}(insert) = \{\{2\}\}$$



Common Non-Variable Subterms

Idea: Replace some **generalizable** subterms t of φ that occur in **recursion positions** with fresh variables x_t .

Definition

A term t is **generalizable** if:

- t has a **recursively defined** type,
- t is not a **variable**, and
- t contains neither **constructors** nor **selectors**.

Example

$$\varphi = n \neq m \wedge n \notin l \wedge n \notin \text{isort}(l) \rightarrow n \notin \text{insert}(m, \text{isort}(l))$$

$$\varphi' = n \neq m \wedge n \notin l \wedge n \notin l' \rightarrow n \notin \text{insert}(m, l')$$

$$\mathcal{RP}(\epsilon) = \{\{2\}\} = \mathcal{RP}(\text{insert})$$



Generating Proposals

For each procedure call $f(t_1, \dots, t_n)$ of φ and each $I \in \mathcal{RP}(f)$, propose $S := \{t_i \mid i \in I\}$ if S seems **suitable**.

Definition

Proposal S is called **suitable** if each $t_i \in S$ is **generalizable** and if some $t_i \in S$ occurs at least **twice** in φ .

For **equations**: $t_i \in S$ needs to occur

- **on the other side** as well, or
- **more than once** on one side.

Example

$$\varphi = x + y * z = y * z + x$$

$$\varphi' = x + n = n + x$$

Example

$$\varphi = (y * z + x) - y * z = x$$

$$\varphi' = (n + x) - n = x$$



Proposals for Recursive Calls

If $S = \{f(t_1, \dots, t_n)\}$

and $f(t_1, \dots, t_n)$ “looks like a recursive call of f ”,

then simultaneously propose **all recursive calls of f** in φ .

Example

- suppose $qsort(smaller(hd(k), tl(k)))$ occurs in φ
- after selector elimination: $qsort(smaller(m, l))$
- proposal $S = \{qsort(smaller(m, l))\}$
- additionally propose
 $S' := \{qsort(smaller(m, l)), qsort(larger(m, l))\}$

↪ fewer iterations, less time (speed-up factor about 3)



Selecting a Proposal

- Sort the list of proposals: Prefer proposals S that
 - pass the **induction test**,
 - were **suggested more often**,
 - generalize **more occurrences** of subterms.
- Accept the **first proposal** that is not rejected by the **disprover**.

Induction Test

Check if the **induction heuristic** suggests an induction on at least one of the **fresh variables** x_t , $t \in S$.



Generalizing Apart Variables

Idea: Separate variable occurrences in **recursion positions** from occurrences in **non-recursion positions**.

Example

$$\varphi = x + (x + x) = (x + x) + x$$

$$\varphi' = x' + (x + x) = (x' + x) + x$$

Find:

- procedure f and a set $I \in \mathcal{RP}(f)$,
- variable v , indices $i \in I$ and $j \notin I$,
- procedure calls

$$f(t_1, \dots, t_{i-1}, v, t_{i+1}, \dots, t_n) \text{ and}$$

$$f(t'_1, \dots, t'_{j-1}, v, t'_{j+1}, \dots, t'_n) \text{ in } \varphi$$

$$f = +, I = \{\{1\}\}$$

$$v = x, i = 1, j = 2$$

$$x + x$$

$$x + x$$



Generalizing Apart Variables

- try **two alternatives** ①/② to separate variable occurrences
- accept the **first “useful” generalization**
(useful = some syntactic checks + not **disproved**)

action ($g \neq f$)

replace $t_i = v$ with v' in $f(t_1, \dots, t_n)$

replace $t_i = v$ with v' in $g(t_1, \dots, t_m)$

recurse into t_i in $f(t_1, \dots, t_n)$

recurse into t_i in $g(t_1, \dots, t_m)$

①

②

if rec. pos.

if rec. pos.

no

if rec. pos.

if rec. pos.

if rec. pos.

yes

if rec. pos.

- $x' * (x + x) = x' * x + x' * x$

(first attempt, using ①)

- $(x' + x) - x = x'$

(first attempt, using ①)

- $|k' \langle \rangle k| = |k'| + |k|$

(first attempt, using ②)



Inverse Weakening

Idea: Remove unnecessary conditions. $\frac{B \rightarrow A}{A}$ (*inv. weak.*)

Example (Insertionsort)

$$\varphi = n \neq m \wedge n \notin l \wedge n \notin l' \rightarrow n \notin \text{insert}(m, l')$$

$$\varphi' = n \neq m \wedge n \notin l' \rightarrow n \notin \text{insert}(m, l')$$

Example (Mergesort)

$$\varphi = k \neq \varepsilon \wedge \text{tl}(k) \neq \varepsilon \wedge \text{ordered}(l_1) \wedge \text{ordered}(l_2) \rightarrow \text{ordered}(\text{merge}(l_1, l_2))$$

$$\varphi' = \text{ordered}(l_1) \wedge \text{ordered}(l_2) \rightarrow \text{ordered}(\text{merge}(l_1, l_2))$$



Inverse Weakening

Definition

Variable v occurs **insignificantly** in ℓ if

(i) $\mathcal{V}(\ell) = \{v\}$ and (ii) no procedure is used in ℓ .

Definition

Condition B in $\mathit{if}\{B, A_1, A_2\}$ is deemed **probably unnecessary** if all occurrences of some $v \in \mathcal{V}(B)$ are *insignificant*.

B is **possibly unnecessary** if ← use **disprover**

- some $v \in \mathcal{V}(B)$ occurs only *insignificantly* in *other* literals,
- neither A_1 nor A_2 is “*false*”.

If B is **probably or possibly unnecessary**, replace ***if***-expression with $\mathit{if}\{A_1, A_2, \mathit{false}\}$, i. e., $A_1 \wedge A_2$.



Inverse Functionality

$$\frac{f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n)}{t_1 = t'_1 \wedge \dots \wedge t_j = t'_j \wedge \dots \wedge t_n = t'_n} \quad (\text{inverse functionality})$$

Side conditions:

- $t_i = t'_i$ for all $i \neq j$,
- equation does not occur as **condition** in φ , and
- resulting formula not **disproved**

Example

$$\varphi = |\text{rev}(\text{rev}(k))| = |k|$$

$$\varphi' = \text{rev}(\text{rev}(k)) = k$$



Evaluation

- integrated in `veriFun`
- assessment of **quality**:
 - success rate: **see below**
 - rate of proposed over-generalizations
(to be found by the **disprover**): **< 12%**
- successfully solves **many “naturally occurring” examples**:
 - Ireland/Bundy (“Productive Use of Failure”): **32 examples**
(5 examples for *inverse functionality* for implications,
13 examples for *inverse replacement* and *lemma discovery*)
 - Aubin (PhD Thesis): **5 examples**
 - more than **15 examples** in other case studies
(correctness properties of common sorting algorithms, properties
of common arithmetic or list operations, e. g., monotonicity of +
and *, associativity of \cap and \cup)



Related Work

	ACL2	Aubin	INKA	IsaPlanner	CLAM	RRL	PVS	veriFun
minimal common subterms	✓							✓
maximal common subterms		✓	✓	✓	✓	✓		✓
selector elimination	●							✓
generalizing apart variables		✓	✓		✓			✓
inverse weakening	●		●			●/✓		✓
inverse functionality		✓	✓	✓				✓
inverse replacement		✓	✓		✓	✓		
disprover		●	✓		✓	✓		✓

✓ : supported¹

● : partially supported

– : not supported

¹implemented and documented



Conclusion

- improved, goal-directed heuristics to generalize formulas:
 - inverse substitution: no commitment to *minimal* or *maximal* subterms; selector elimination; generalizing apart variable occurrences with *few(er) attempts*; use recursion information
 - inverse weakening: more *liberal* \rightsquigarrow better generalizations
 - inverse functionality: less over-generalizations
- disprover rarely has to reject an over-generalization
- flexible approach: other verification systems can benefit from the improvements
- find out more: CADE 2007, Disproving WS @ IJCAR 2006



Quicksort

Example

```
procedure qsort(k : list[ $\mathbb{N}$ ]) : list[ $\mathbb{N}$ ] <=
```

```
if k =  $\varepsilon$ 
```

```
  then  $\varepsilon$ 
```

```
  else qsort(smaller(hd(k), tl(k))) <>
```

```
    hd(k) :: qsort(larger(hd(k), tl(k)))
```

```
end
```

```
procedure smaller(n :  $\mathbb{N}$ , k : list[ $\mathbb{N}$ ]) : list[ $\mathbb{N}$ ] <= ...
```

```
procedure larger(n :  $\mathbb{N}$ , k : list[ $\mathbb{N}$ ]) : list[ $\mathbb{N}$ ] <= ...
```

$$\mathcal{RP}(\text{qsort}) = \{\{1\}\}$$

$$\mathcal{RP}(\text{smaller}) = \mathcal{RP}(\text{larger}) = \{\{2\}\}$$



Quicksort: *smaller*

Example

```
procedure smaller( $n : \mathbb{N}$ ,  $k : \text{list}[\mathbb{N}]$ ) :  $\text{list}[\mathbb{N}] \leq =$   
if  $k = \varepsilon$   
  then  $\varepsilon$   
  else if  $\text{hd}(k) > n$   
    then smaller( $n$ ,  $\text{tl}(k)$ )  
    else  $\text{hd}(k) :: \text{smaller}(n, \text{tl}(k))$   
  end  
end
```

$$\mathcal{RP}(\text{smaller}) = \{\{2\}\}$$

