Correctness of Functional Programs

A simple setting

Michael R. Hansen

mrh@imm.dtu.dk

Informatics and Mathematical Modelling

Technical University of Denmark

Overview

Today:

- Verification of functional programs
  Simple setting
  - terminating programs
  - set-theoretic interpretation of types
  - inductively defined datatypes
  - structural induction
  - well-founded induction

  which covers a wide range of interesting programs

Next week

- Test exam

Example: the merge function

Merge two ordered lists:

```
fun merge(xs,[]) = xs
  | merge([],ys) = ys
  | merge(x::xs,y::ys) =
    case Int.compare(x,y) of
           EQUAL => x::y::merge(xs,ys)
         | LESS => x::merge(xs,y::ys)
         | GREATER => y::merge(x::xs,ys)
```

Correctness: what does it involve?

Example: the `merge` function

Merge two ordered lists:

```
    fun merge(xs,[]) = xs
      | merge([],ys) = ys
      | merge(x::xs,y::ys) =
        case Int.compare(x,y) of
              EQUAL => x::y::merge(xs,ys)
            | LESS => x::merge(xs,y::ys)
            | GREATER => y::merge(x::xs,ys)
```

Correctness: what does it involve?

- termination proof
- $\forall xs, ys : \alpha list.$

$$ordered(xs) \wedge ordered(ys) \Rightarrow ordered(merge(xs, ys)) \qquad (M)$$

- is more needed?

Reasoning about program expressions

Are the following meaningful?

- From $e_1 = e_1 + e_2$ we conclude $e_2 = 0$
- $e_3 + e_3 = 2e_3$

Not necessarily in the presence of non-termination and side effects:

- Given fun f(x) = f(x) + 1.
- Let e_3 be given by (x:= !x+1; !x +3)

In domain theory special partially ordered sets are introduced to deal with non-termination. Dana Scott late 1960s

In Hoare Logic one can reason about imperative programs

Floyd, Hoare late 1960s

Simple setting

- terminating programs
- applicative (pure functional) programs
- set-theoretical interpretation of types

Supports valid arguments based on

- simple equational reasoning
- inductive reasoning

Simple setting

- terminating programs
- applicative (pure functional) programs
- set-theoretical interpretation of types

Supports valid arguments based on

- simple equational reasoning
- inductive reasoning

Certain datatypes are excluded, e.g.

```
datatype A = C of A -> A
```

as no set A is isomorphic to $A \rightarrow A$.

Can be dealt with in domain theory

Example: iterative factorial function

We prove $\forall n \in \mathbb{N} \forall p \in \mathbb{N}. facti(n, p) = n! \cdot p$, where

```
fun facti(0,p) = p                     (* Case 1 *)
  | facti(n,p) = facti(n-1,n*p)        (* Case 2 *)
```

using the following well-known induction rule for natural numbers

1. $P(0)$ base case
2. $\forall n.(P(n) \Rightarrow P(n+1))$ inductive step

$$\frac{P(0) \qquad \forall n.(P(n) \Rightarrow P(n+1))}{\forall n.P(n)}$$

Example: iterative factorial function

We prove $\forall n \in \mathbb{N} \forall p \in \mathbb{N}.facti(n, p) = n! \cdot p$, where

```
fun facti(0,p) = p                    (* Case 1 *)
  | facti(n,p) = facti(n-1,n*p)  (* Case 2 *)
```

using the following well-known induction rule for natural numbers

1.	$P(0)$	base case
2.	$\forall n.(P(n) \Rightarrow P(n+1))$	inductive step
	$\forall n.P(n)$	

What is $P(n)$?

Example: iterative factorial function

We prove $\forall n \in \mathbb{N} \forall p \in \mathbb{N}. facti(n, p) = n! \cdot p$, where

```
fun facti(0,p) = p                    (* Case 1 *)
  | facti(n,p) = facti(n-1,n*p)  (* Case 2 *)
```

using the following well-known induction rule for natural numbers

1.	$P(0)$	base case
2.	$\forall n.(P(n) \Rightarrow P(n+1))$	inductive step
	$\forall n.P(n)$	

What is $P(n)$?

Base case. We must prove $\forall p \in \mathbb{N}. facti(0, p) = 0! \cdot p$. Trivial.

Example: iterative factorial function

We prove $\forall n \in \mathbb{N} \forall p \in \mathbb{N}.\mathit{facti}(n,p) = n! \cdot p$, where

```
fun facti(0,p) = p                  (* Case 1 *)
  | facti(n,p) = facti(n-1,n*p)     (* Case 2 *)
```

using the following well-known induction rule for natural numbers

$$\frac{\begin{array}{lll} 1. & P(0) & \text{base case} \\ 2. & \forall n.(P(n) \Rightarrow P(n+1)) & \text{inductive step} \end{array}}{\forall n.P(n)}$$

What is $P(n)$?

Base case. We must prove $\forall p \in \mathbb{N}.\mathit{facti}(0,p) = 0! \cdot p$. Trivial.

Inductive step. Consider arbitrary $n \in \mathbb{N}$. We must establish

$$\underbrace{\underbrace{\forall p \in \mathbb{N}.\mathit{facti}(n,p) = n! \cdot p}_{\text{induction hypothesis}}}_{P(n)} \Rightarrow \underbrace{\forall p \in \mathbb{N}.\mathit{facti}(n+1,p) = (n+1)! \cdot p}_{P(n+1)}$$

Example cont'd

Assume the induction hypothesis:

$$\forall p' \in \mathbb{N}.\mathit{facti}(n, p') = n! \cdot p' \qquad (\mathrm{Ind.hyp.})$$

Consider arbitrary $p \in \mathbb{N}$.

$$\begin{array}{ll} & \mathit{facti}(n+1, p) \\ = & \mathit{facti}(n, (n+1) \cdot p) \qquad \mathrm{Case}\ 2, \mathrm{as}\ n+1 \neq 0 \\ = & n! \cdot (n+1) \cdot p \qquad \mathrm{Ind.hyp.}, p' \mapsto (n+1) \cdot p \\ = & (n+1)! \cdot p \end{array}$$

which establishes the inductive step.

Hence $\forall n \in \mathbb{N} \forall p \in \mathbb{N}.\mathit{facti}(n, p) = n! \cdot p$, by the induction rule.

Structural induction over lists

The declaration

```
datatype 'a list = Nil | :: of 'a * 'a list
```

denotes an inductive definition of lists (of type 'a)

- [] is a list
- if x is an element and xs is a list, then $x :: xs$ is a list
- lists can be generated by above rules only

The following structural induction rule is therefore sound:

1.	$P([])$	base case
2.	$\forall xs.\forall x(P(xs) \Rightarrow P(x :: xs))$	inductive step

$$\frac{\begin{array}{ll} 1.\ P([]) & \text{base case} \\ 2.\ \forall xs.\forall x(P(xs) \Rightarrow P(x :: xs)) & \text{inductive step} \end{array}}{\forall xs.P(xs)}$$

Example

```
fun [] @ ys = ys | (x::xs) @ ys = x::(xs @ ys);
fun len [] = 0 | len (_::xs) = 1+len xs;
```

We prove: $\forall xs.len(xs@ys) = len(xs) + len(ys)$

Base case: $len([]@ys) = len(ys) = 0 + len(ys) = len([]) + len(ys)$

Inductive step:

$$\begin{array}{lll} & len((x :: xs)@ys) & \\ = & len(x :: (xs@ys)) & def.append \\ = & 1 + len(xs@ys) & def.len \\ = & 1 + (len(xs) + len(ys)) & ind.hyp. \\ = & (1 + len(xs)) + len(ys) & arith. \\ = & len(x :: xs) + len(ys) & def.len \end{array}$$

Exercises

Prove

- `rev(xs @ ys) = rev(ys) @ rev(xs)`

where

```
fun rev []       = []                  (* Case 1 *)
  | rev (x::xs) = rev xs @ [x]        (* Case 2 *)
```

Exercises

Prove

- `rev(xs @ ys) = rev(ys) @ rev(xs)`

where

```
fun rev []       = []                       (* Case 1 *)
  | rev (x::xs) = rev xs @ [x]              (* Case 2 *)
```

Did you (need to) prove

- `xs @ [] = xs`
- `xs @ (ys @ zs) = (xs @ ys) @ zs ?`

Exercises

Prove

- `rev(xs @ ys) = rev(ys) @ rev(xs)`

where

```
fun rev []       = []                     (* Case 1 *)
  | rev (x::xs) = rev xs @ [x]            (* Case 2 *)
```

Did you (need to) prove

- `xs @ [] = xs`
- `xs @ (ys @ zs) = (xs @ ys) @ zs ?`

Can you prove correctness of merge using previous ind. rule?

Well-founded relation

Let R be a binary relation on A, i.e. $R \subseteq A \times A$.

An element $m \in X \subseteq A$ is *minimal* in X if for no element $x \in X$: xRm.

A binary relation $R \subseteq A \times A$ is *well-founded* if

- every non-empty subset $X \subseteq A$ has a minimal element.

Equivalent formulation:

- A contains no countable infinite descending chains: i.e there is no infinite sequence $x_0, x_1, x_2, \ldots$ of elements of A such that $x_{i+1} R x_i$.

Well-founded induction

Given irreflexive, well-founded R on X.

Principle of well-founded induction:

$$\frac{\overbrace{\forall y \in X.((\forall x \in X.xRy \Rightarrow P(x)) \Rightarrow P(y))}^{\text{premise}}}{\forall y \in X.P(y)}$$

Well-founded induction

Given irreflexive, well-founded R on X.

Principle of well-founded induction:

$$\frac{\overbrace{\forall y \in X.((\forall x \in X.xRy \Rightarrow P(x)) \Rightarrow P(y))}^{\text{premise}}}{\forall y \in X.P(y)}$$

Principle is sound because:

Suppose premise is true and $E = \{e \in X \mid \neg P(e)\} \neq \emptyset$.

We derive a contradiction as follows:

- E has a minimal element $m \in E$ since R is well-founded and $\neg P(m)$
- xRm implies $x \notin E$, i.e. $P(x)$ holds, since m is minimal in E
- $P(m)$ by the premise as $\forall x \in X.xRm \Rightarrow P(x)$

Examples

The previous induction principles are specializations:

- Natural numbers, with $a <_s b$ iff $b = a + 1$.
- Lists, with $xs \prec_{tl} ys$ iff $xs = tail(ys)$.

A few other examples:

- List, with prefix ordering $\prec$
- List, with lexicographical ordering $\prec_L$
- Trees, with sub-tree ordering
- $\mathbb{N} \times \mathbb{N}$ with $(n, p) <' (n', p')$ iff $n + 1 = n'$
- $List \times List$ with $(xs, ys) <'' (xs', ys')$ iff $length(xs) + length(ys) < length(xs') + length(ys')$

Exercise

- Redo proof for facti using well-founded induction using the relation $(n, p) <' (n', p')$ iff $n + 1 = n'$ on $\mathbb{N} \times \mathbb{N}$.

  Notice that ind. hyp. can be simplified.

- Consider a proof for merge (property M) using well-founded induction on the basis of: $List \times List$ with $(xs, ys) <'' (xs', ys')$ iff $length(xs) + length(ys) < length(xs') + length(ys')$

  Formulate the main proof steps. (You do not need to complete them.)

Structural induction on Trees

Inductive definition of binary trees:

```
datatype 'a tree = Lf | Br of 'a * 'a tree * 'a tre
```

and an associated induction rule:

1. $P(Lf)$ base case
2. $\forall t_1, t_2. \forall n. (P(t_1) \wedge P(t_2) \Rightarrow P(Br(n, t1, t2))$ inductive step

$$\frac{1. \; P(Lf) \qquad 2. \; \forall t_1, t_2. \forall n. (P(t_1) \wedge P(t_2) \Rightarrow P(Br(n, t1, t2))}{\forall t. P(t)}$$

Example

```
fun count Lf                  = 0
  | count(Br(_,t1,t2)) = 1 + count t1 + count t2;

fun depth Lf = 0
  | depth(Br(n,t1,t2)) =
       1+ Int.max(depth t1, depth t2)
```

Property: for every binary tree t:

$$\mathrm{count}(t) \leq 2^{\mathrm{depth}(t)} - 1$$

Example cont'd: proof

- by structural induction over trees

Base case:

$$\mathrm{count}(\mathrm{Lf}) = 0 = 2^{\mathrm{depth}(\mathrm{Lf})} - 1$$

Inductive step:

$$\begin{array}{lll}
 & \mathrm{count}(\mathrm{Br}(n, t_1, t_2)) & \\
= & 1 + \mathrm{count}(t_1) + \mathrm{count}(t_2) & \text{def. count} \\
\leq & 1 + (2^{\mathrm{depth}(t_1)} - 1) + (2^{\mathrm{depth}(t_2)} - 1) & \text{ind.hyp.} \\
\leq & 2^{\max(\mathrm{depth}(t_1), \mathrm{depth}(t_2))} + 2^{\max(\mathrm{depth}(t_1), \mathrm{depth}(t_2))} - 1 & \text{arith.} \\
= & 2^{1+\max(\mathrm{depth}(t_1), \mathrm{depth}(t_2)} - 1 & \text{arith.} \\
= & 2^{\mathrm{depth}(\mathrm{Br}(n, t_1, t_2))} - 1 & \text{def. depth}
\end{array}$$

Exercise

```
fun postorder Lf              = []
  | postorder(Br(n,t1,t2)) =
          postorder t1 @ postorder t2 @ [n];

fun preorder Lf              = []
  | preorder(Br(n,t1,t2)) =
          n :: preorder t1 @ preorder t2;

fun reflect Lf              = Lf
  | reflect(Br(n,t1,t2)) =
          Br(n, reflect t2, reflect t1);
```

Prove: for every binary tree t:

$$\mathrm{postorder}(\mathrm{reflect}(t)) = \mathrm{rev}(\mathrm{preorder}(t))$$