CS5860 Iteration and Other Forms of Induction

Thur. Oct. 27, 2011

PLAN & ANNOUNCEMENTS

1. See homework exercises due Nov 7 for grading - see Web page
2. Need projects settled by Nov 4, due by end of study period.
3. Other forms of induction - logical form, some examples
4. Iteration and loops
5. Loop invariants and proofs of program properties

Forms of Induction

1. From base larger than 0

Define $y \leq x$ as $Eq(y,x) \vee y < x$ Recall def of $y < x$ last lect.

Induction base b

$$(A(b) \;\&\; \forall x.(b \leq x \Rightarrow (A(x) \Rightarrow A(S(x)))) \Rightarrow \forall x(b \leq x \Rightarrow A(x))$$

The Stamps Problem uses base 8. Here is the Nuprl form (see supplemental material)

$\vdash \forall n:\mathbb{N}.\ n \geq 8 \Rightarrow \exists i,j:\mathbb{N}.\ n = (i*3 + j*5)$.

by $\lambda(n.\ ind_8(\_\_;\ n,\ ind\text{-}hyp.\ \_\_\_\_\_))$

base case $\vdash \exists i,j:\mathbb{N}.\ (8 = i*3 + j*5)$ by $\langle \_, \langle \_, \_ \rangle \rangle$

$\vdash \langle 1, \langle 1, arith \rangle \rangle$

$n:\mathbb{N}$, $n > 8$, ind_hyp: $\exists i,j:\mathbb{N}.\ (n-1) = i*3 + j*5$

$\vdash \exists i,j:\mathbb{N}.\ n = i*3 + j*5$ by spread(ind_hyp; __)

$i:\mathbb{N}$, $j:\mathbb{N}$, $n-1 = i*3 + j*5$, $j = 0$ $\vdash \exists i,j:\mathbb{N}.\ n = i*3 + j*5$ by $\langle i-3, 2 \rangle$

" " " $j \neq 0$ $\vdash$ " " " " by $\langle i+2, j-1 \rangle$

CS 5860 Iteration and Other Forms of Induction

Thur Oct 27, 2011

Forms of Induction

2. Multiple base cases, e.g. two

$$(A(0) \;\&\; A(1) \;\&\; \forall x.\, (x \geq 2 \Rightarrow (A(x) \Rightarrow A(S(x))))) \Rightarrow \forall x.\, A(x)$$

3. Complete_induction (course of values)

$$\forall x.\, (\forall y.\, (y < x \Rightarrow P(y)) \Rightarrow P(x)) \Rightarrow \forall x.\, P(x)$$

To show $P(x)$ we get to use $P(0), P(1), \cdots, P(x-1)$.

Why do we know $P(0)$? There is no base case.

Notice that for $x=0$ the hypothesis is

$$\forall y.\, (y < 0 \Rightarrow P(y)) \Rightarrow P(0)$$

and there are no $y < 0$ to use, so we must have a proof of $P(0)$ from no hypotheses, well a trivial one, evidence $*$.

A very good exercise is to show

$$\text{Induction} \iff \text{Complete\_induction}$$

This is not simple but very constructive with interesting computational content.

4. Least Number Principle (Boolean Logic)^{o}

$$\exists x.\, P(x) \Rightarrow \exists y.\, (P(y) \;\&\; \forall z.\, (z < y \Rightarrow \sim P(z)))$$

How to make this programmable? Here is one way.

$$\forall x.\, (P(x) \vee \neg P(x)) \Rightarrow [\, \exists y\, (y < x \;\&\; P(x) \;\&\; \forall z.\, (z < y \Rightarrow \sim P(z))) \,] \vee [\, \forall y\, (y < x \Rightarrow \sim P(y)) \,] \quad \text{for any } x.$$

o superscript means Boolean Logic, possibly not programmable

CS 5860 Iteration and Other Forms of Induction

Thur. Oct 27, 2011

Forms of Induction

5. Infinite Descent

$\forall x.\ (P(x) \Rightarrow \exists y\ (y < x \ \&\ P(y))) \Rightarrow \forall x.\ \sim P(x)$

If we assume $P(x)$ and can find a smaller value such that $P(y)$ then $P(x)$ cannot be known for any x, e.g. $\forall x.\ \sim P(x)$.

6. Efficient Induction

We can define an approximation to division, say $x \div 3$ as in quotient and remainder example

so $3 \div 3 = 1$, $4 \div 3 = 1$, $5 \div 3 = 1$, $6 \div 3 = 2$

$rm(4,3) = 1$, $rm(5,3) = 2$, $rm(6,3) = 0$

$x \div 3$ is the largest q $3 * q \leq x$

$P(0) \ \&\ \forall x.(P(x \div 2) \Rightarrow P(x)) \Rightarrow \forall x.\ P(x)$

See the instance

$P(0) \ \&\ \forall x.(P(x \div 4) \Rightarrow P(x)) \Rightarrow \forall x.\ P(x)$

for the "fast integer square root" example, posted as supplemental material.

CS 5860 Iteration and Other Forms of Induction

Thur Oct 27, 2011

Iteration and Tail Recursion

We noted in the last lecture that iteration is also a realizer for induction. It can also be presented as a special kind of recursion called tail recursion. We wrote the iterative form for addition.

Compare these definitions of the factorial function,

$n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots 1$.

$$fact(x) = \text{if } x = 0 \text{ then } 1 \text{ else } x * fact(x-1).$$

This takes linear space to evaluate.

$$it\text{-}fact(p, x, m) = \text{if } x > m \text{ then } p \text{ else } it\text{-}fact(x*p, x+1, m)$$

$$fact2(x) = it\text{-}fact(1, 1, x).$$

We can write this as a loop.

```
p := 1;
for x = 1 to m do p := p*x od
```

```
p := 1; x := 1    { p = 1*2*...*x }
while x ≤ m do p := p*x; x := x+1 od
{ p = 1*2*...*x for 1 ≤ x ≤ m }
```

CS 5860 Iteration and Other Forms of Induction

Thur Oct 27, 2011

We can reason about while loops using this rule known as the Hoare While Rule. It is like induction expressed as iteration.

In(x)
{ In(x) => Inv(x) }
while b(x) do
  { Inv(x) }
  x := f(x)
  { Inv(x) }
od
{ ~b(x) & Inv(x) }
{ (~b(x) & Inv(x)) => Out(x) }

Inv(x) & b(x) => Inv(f(x))

We can express much of this in FOL, except for termination of the loop.

$$\forall x \; (b(x) \vee \sim b(x)) \;\&\; (In(x) \Rightarrow Inv(x)) \;\&\; Inv(x) \,\&\, b(x) \Rightarrow Inv(f(x)) \;\&\; \sim b(x) \,\&\, Inv(x) \Rightarrow Out(x))$$

All this implies that if the loop terminates and the input x satisfies In(x), then the result satisfies Out(x). This is a partial correctness claim.

Note: These are wonderful projects applying these ideas to reasoning about abstract programs (called program schemes) in FOL.

Appendix A

Derivation of a Fast Integer Square Root Algorithm

by Christoph Kreitz

A.1 Deriving a Linear Algorithm

The standard approach to proving $\forall n\,\exists r\; r^2{\leq}n \wedge n{<}(r{+}1)^2$ is induction on n, which will lead to the following two proof goals

Base Case: prove $\exists r\; r^2{\leq}0 \wedge 0{<}(r{+}1)^2$

Induction Step: prove $\exists r\; r^2{\leq}n{+}1 \wedge n{+}1{<}(r{+}1)^2$ assuming $\exists r_n\; r^2{\leq}n \wedge n{<}(r_n{+}1)^2$.

The base case can be solved by choosing $r = 0$ and using standard arithmetical reasoning to prove the resulting proof obligation $0^2{\leq}0 \wedge 0{<}(0{+}1)^2$.

In the induction step, one has to analyze the root r_n. If $(r_n{+}1)^2{\leq}n{+}1$, then choosing $r = r_n{+}1$ will solve the goal. Again, the proof obligation $(r_n{+}1)^2{\leq}n{+}1 \wedge n{+}1{<}((r_n{+}1){+}1)^2$ can be shown by standard arithmetical reasoning. $(r_n{+}1)^2 > n{+}1$, then one has to choose $r = r_n$ and prove $r_n^2{\leq}n{+}1 \wedge n{+}1{<}(r_n{+}1)^2$ using standard arithmetical reasoning.

Figure A.1 shows the trace of a formal proof in the Nuprl system [40, 10] that uses exactly this line of argument. It initiates the induction by applying the library theorem

`NatInd` $\quad \forall P{:}\mathbb{N}{\to}\mathbb{P}.\;\; (P(0) \wedge (\forall i{:}\mathbb{N}^+.\;\; P(i\text{-}1) \Rightarrow P(i))) \Rightarrow (\forall i{:}\mathbb{N}.\;\; P(i))$

The base case is solved by assigning 0 to the existentially quantified variable and using Nuprl's autotactic (trivial standard reasoning) to deal with the remaining proof obligation. In the step case (from $i{-}1$ to i) it analyzes the root r for $i{-}1$, introduces a case distinction on $(r{+}1)^2{\leq}i$ and then assigns either r or $r{+}1$, again using Nuprl's autotactic on the rest of the proof.

Nuprl is capable of extracting an algorithm from the formal proof, which then may be run within Nuprl's computation environment or be exported to other programming systems. The algorithm is represented in Nuprl's extended lambda calculus.

Depending on the formalization of the existential quantifier there are two kinds of algorithms that may be extracted. In the standard formalization, where $\exists$ is represented as a (dependent) product type, the algorithm – shown on the left* – computes both the integer square root r of a given natural number n and a proof term, which verifies that r is in fact the integer square root of n. If $\exists$ is represented as a set type, this verification information is dropped during extraction and the algorithm – shown on the right – only performs the computation of the integer square root.

*The place holders pf_k represent the actual proof terms that are irrelevant for the computation.

```
∀n:ℕ.  ∃r:ℕ.  r²≤n<(r+1)²
BY  allR
  n:ℕ
  ⊢ ∃r:ℕ.  r²≤n<(r+1)²
  BY  NatInd 1
  .....basecase.....
      ⊢ ∃r:ℕ.  r²≤0<(r+1)²
  √  BY  existsR ⌈0⌉ THEN Auto
  .....upcase.....
      i:ℕ+,  r:ℕ,  r²≤i-1<(r+1)²
      ⊢ ∃r:ℕ.  r²≤i<(r+1)²
      BY  Decide ⌈(r̲+1)²≤i⌉ THEN Auto
      .....Case 1.....
          i:ℕ+,  r:ℕ,  r²≤i-1<(r+1)²,  (r+1)²≤i
          ⊢ ∃r:ℕ.  r²≤i<(r+1)²
      √  BY  existsR ⌈r̲+1⌉ THEN Auto'
      .....Case 2.....
          i:ℕ+,  r:ℕ,  r²≤i-1<(r+1)²,  ¬((r+1)²≤i)
          ⊢ ∃r:ℕ.  r²≤i<(r+1)²
      √  BY  existsR ⌈r̲⌉ THEN Auto
```

Figure A.1: Proof of the Specification Theorem using Standard Induction.

```
let rec sqrt i
= if i=0 then <0,pf_0>
  else let <r,pf_{i-1}>  = sqrt (i-1)
       in
         if (r̲+1)²≤n  then
<r̲+1,pf_i>
         else <r̲,pf_i'>
```

```
let rec sqrt i
= if i=0 then 0
  else let r = sqrt (i-1)
       in
         if (r̲+1)²≤n  then r̲+1
         else r̲
```

Using standard conversion mechanisms, Nuprl can then transform the algorithm into any programming language that supports recursive definition and export it to the corresponding programming environment. As this makes little sense for algorithms containing proof terms, we only convert the algorithm on the right. A conversion into SML, for instance, yields the following program.

```
fun sqrt n = if n=0 then 0
             else let val r = sqrt (n-1)
                  in
                    if n<(r̲+1)^2  then r̲
                    else r̲+1
                  end
```

A.2 Deriving an Algorithm that runs in $\mathcal{O}(\sqrt{n})$

Due to the use of standard induction on the input variable, the algorithm derived in the previous section is linear in the size of the input n, which is reduced by 1 in each step. Obviously, this is not the most efficient way to compute an integer square root. In the following we will derive more efficient algorithms by proving $\forall n\,\exists r\; r^2 \leq n \wedge n < (r+1)^2$ in a different way. These proof, however, will have to rely on more complex induction schemes to ensure a more efficient computation.

See Web page for this derivation.