ETH zürich

P. Müller, D. Basin and R. Jung

Formal Methods and Functional Programming

Session Sheet 8: Induction

Background: Induction Schemas

Mathematical Induction:

- The rule for *weak mathematical induction* is given by:

$$\frac{\underline{\Gamma} \vdash \underline{P}(0) \quad \underline{\Gamma}, \underline{P}(\underline{n}) \vdash \underline{P}(\underline{n}+1)}{\underline{\Gamma} \vdash \forall \underline{n} \cdot \underline{P}(\underline{n})} \quad \text{where } \underline{n} \text{ not free in } \underline{\Gamma}$$

- The rule for *strong mathematical induction* is given by:

$$\frac{\underline{\Gamma}, \forall \underline{m} < \underline{n} \cdot \underline{P}(\underline{m}) \vdash \underline{P}(\underline{n})}{\underline{\Gamma} \vdash \forall \underline{n} \cdot \underline{P}(\underline{n})} \quad \begin{array}{l}\text{where } \underline{n} \text{ not free in } \underline{\Gamma} \\ \text{and } \underline{m} \text{ not free in } \underline{P}(\underline{n})\end{array}$$

Structural Induction:

- The rule for *weak structural induction* over lists is given by:

$$\frac{\underline{\Gamma} \vdash \underline{P}([\,]) \quad \underline{\Gamma}, \underline{P}(\underline{xs}) \vdash \underline{P}(\underline{x} : \underline{xs})}{\underline{\Gamma} \vdash \forall \underline{xs} \cdot \underline{P}(\underline{xs})} \quad \text{where } \underline{x}, \underline{xs} \text{ not free in } \underline{\Gamma}$$

- The rule for *strong structural induction* over lists is given by:

$$\frac{\underline{\Gamma}, \forall \underline{ys} \sqsubset \underline{xs} \cdot \underline{P}(\underline{ys}) \vdash \underline{P}(\underline{xs})}{\underline{\Gamma} \vdash \forall \underline{xs} \cdot \underline{P}(\underline{xs})} \quad \begin{array}{l}\text{where } \underline{xs} \text{ not free in } \underline{\Gamma} \\ \text{and } \underline{ys} \text{ not free in } \underline{P}(\underline{xs})\end{array}$$

The subterm relation for lists, denoted by $\sqsubset$, is defined as follows:

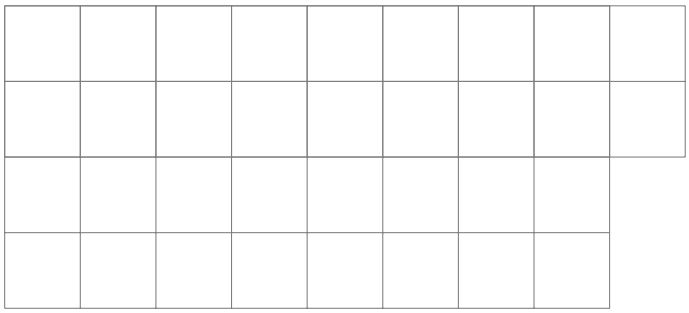
$$\forall xs, ys \cdot xs \sqsubset ys \iff (\exists x \cdot ys = x : xs) \vee (\exists zs \cdot xs \sqsubset zs \wedge zs \sqsubset ys)$$

Assignment 1 (Prime Divisor)

An integer p is *prime*, which we write $prime(p)$, if and only if it is only divisible by itself and by 1. Prove that all integers $n \geq 2$ are divisible by a prime number.

Assignment 2 (Splitting a Chocolate Bar)

Consider a chocolate bar consisting of n squares arranged in a rectangular pattern:



Task: We want to split the bar into small squares. Assuming we only can cut the bar along a line, how many cuts do we need?

Task: Prove that we can split the bar into small squares with $n-1$ cuts along the lines. More precisely, for an integer k, let $C(k)$ be the property "$k-1$ cuts along the lines are sufficient to split into small squares a chocolate bar containing k squares". Prove $C(k)$ for all $k \geq 1$, with a *strong* induction.

Task: *Any* proof by strong induction can be done with a weak induction. To see this, prove that we can split the bar into small squares with $n-1$ cuts along the lines, this time with a *weak* induction.

Assignment 3 (Run-Length Encoding)

The background of this assignment is a simple run-length encoding scheme[1]. In our case, the input data is encoded as a list of natural numbers[2] of even length. The encoded representation has the form $n_1 : v_1 : n_2 : v_2 : \ldots :$ `[]`, where each pair $n_i : v_i$ denotes, that the input data contained n_i consecutive occurrences of v_i. For example, the input `1:1:1:5:5:5:5:[]` will be encoded as `3:1:4:5:[]`.

[1] `http://en.wikipedia.org/wiki/Run-length_encoding`
[2] We include zero in the natural numbers.

The function `dec` decodes run-length encoded data represented as a list of natural numbers. The function `rep` n v creates a list

```
dec []       = []                          -- (D1)
dec [n]      = []                          -- (D2)
dec (n:v:xs) = rep n v ++ dec xs           -- (D3)

rep 0 v = []                               -- (R1)
rep n v = v:(rep (n-1) v)                  -- (R2)
```

The function `srclen` computes the length of the source data from the encoded representation.

```
srclen []       = 0                        -- (S1)
srclen [n]      = 0                        -- (S2)
srclen (n:v:xs) = n + srclen xs            -- (S3)
```

Note: The pathological cases (D2) and (S2) are only there to make the corresponding functions total.

Lemmas: You may use the two following lemmas without proving them:

(L1) $\forall xs, ys \cdot$ `length (`xs `++` ys`) = length` xs `+ length` ys

(L2) $\forall x, y \cdot$ `length (rep` x y`) = x`

Task: Prove that the length computed by `srclen` corresponds to the length of the decoded data, i.e.,

$$\forall xs \cdot \texttt{length (dec } xs\texttt{) = srclen } xs$$

Session 8 - Induction

Notation wie $\forall n \geq 2 \ldots$ nicht mehr erlaubt. Stattdessen $\forall n.\ n \geq 2 \Rightarrow \ldots$

① Let $P(n) \equiv \exists d.\ prime(d) \wedge d \mid n$ — bemerke $P(n)$ und $\exists d. \ldots$ sowie $\forall n \geq 2$

Prove ~~$\forall n \geq 2.\ P(n)$~~ $\forall n.\ n \geq 2 \Rightarrow P(n)$ by strong induction on n

So for any $n \in \mathbb{N}$ prove $P(n)$ assuming IH: $\forall j.\ 2 \leq j < n \Rightarrow P(j)$

Case distinction on n (von mir, um mit späteren Kochrezept konform zu sein)

Case $prime(n)$: Then $n = 1 \cdot n$, therefore is divided by some prime, so $P(n)$

Case $\neg prime(n)$: So these is some $d \in \mathbb{N}$ w/ $1 < d < n$ and $d \mid n$. Since $2 \leq d < n$ we have $P(d)$ per IH, so there is some prime d' w/ $d' \mid d$.

Therefore $d' \mid n$, so $P(n)$ holds

Notice we're not distinguishing Base & Step Case ?
(I would've done that)
→ Case $n = 2$ ist in $prime(n)$ enthalten

②

[a] Important: We can only cut one piece at a time

→ $n-1$ cuts (Intuition: mit jedem Cut gewinnen wir 1 Stück)

[b] Let $C(k)$ be property "need $k-1$ cuts to cut of rectangle consisting of k squares"

Prove ~~$\forall k \geq 1.\ C(k)$~~ $\forall k.\ k \geq 1 \Rightarrow C(k)$ by strong induction on k (wieder: IH via Cases)

For arbitrary $k \geq 1$, prove $C(k)$ assuming IH $\forall j\ 1 \leq j < k \Rightarrow C(j)$

Case distinct. on k

Case $k = 1$: We only have 1 square, so 0 cuts necessary

Case $k > 1$: Can split rectangle into 2 parts w/ 1 cut to obtain two chocol. bars w/ $k_1 \geq 1$ and $k_2 \geq 1$ squares, where $k_1 + k_2 = k$. Since $k_1 < k$, $k_2 < k$ we have $C(k_1)$, $C(k_2)$ per IH meaning we need

$(k_1 - 1) + (k_2 - 1)$ cuts to cut up the smaller chocolate bars into squares.

We therefore need $1 + (k_1 - 1) + (k_2 - 1) = k - 1$ cuts to cut the original chocolate bar, so $C(k)$ holds

← Pattern, das normalerweise verwendet wird

c Idea: "Push" strong post (~~$\forall 1 \leq k \leq n$~~ $\forall k.\ 1 \leq k \leq n$) into property since we need to gain $C(k_1)$, $C(k_2)$ from it for any k_1, k_2 w/ $k_1 + k_2 = k$, $1 \leq k_1 < n$, $1 \leq k_2 < n$

Let $P(k) \equiv \forall j. \left(1 \leq j \leq k \Rightarrow C(j)\right)$

Prove ~~$\forall k \geq 1. P(k)$~~ $\forall k. k \geq 1 \Rightarrow P(k)$ by weak induction on k

Base Case: Prove $P(1) \equiv \forall j.\ 1 \leq j \leq 1 \Rightarrow C(1)$

So prove $C(1)$ holds. We need $1 - 1 = 0$ cuts to split up a chocolate bar w/ 1 square. So $P(1)$ holds

Step Case: Let $k \geq 1$ be arbitrary but fixed. Prove $P(k+1)$ assuming IH: $P(k)$ holds.

So show $P(k+1) \equiv \forall j.\ 1 \leq j \leq k+1 \Rightarrow C(j)$

Let j w/ $1 \leq j \leq k+1$ be arbitrary but fixed.

Case distinction on j to prove $P(k+1)$

Case $1 \leq j \leq k$: We immediately get $C(j)$ from $P(k)$

Case $j = k+1$: Same as before, cut chocolate bar into 2 pieces. We get $k_1 + k_2 = k+1$, $1 \leq k_1 \leq k$, $1 \leq k_2 \leq k$, so from $P(k)$ (w/ $j = k_1$, $j = k_2$) we know $C(k_1)$, $C(k_2)$, so need $(k_1 - 1) + (k_2 - 1)$

cuts for the parts. Overall $1+(k_1-1)+(k_2-1)$ $= (k+1)-1$ cuts, so $C(j)$ holds.

→ nicht mehr so exakt wie frühere Induktion

→ aber trotzdem muss Beweisstruktur sauber sein (case dist. ankündigen, ...), alles relevante ausgeschrieben ☺

→ spätes primär structural induction, da gibt es wieder Kochrezepte

③ Run-length encoding

→ normal bisschen Haskell, aber auch hier paar Sachen anders (wird aber nicht vorkommen)

→ D2, S2 machen die Funktionen total (funktionieren auf jedem Input)

→ Wie beweisen? (Weak) structural induction würde im Step Case "$y :: a$, $ys :: [a]$ arbitrary but fixed, prove $P(y:ys)$ assuming $P(ys)$" machen.

Wir müssen aber wissen, wie ys aussieht, um D2, D3 zu unterscheiden → strong structural induction

Let $P(xs) \equiv$ length (dec xs) = srclen xs → erneut ∀xs. ..., dass xs :: [Mat] implizit

Prove $\forall xs.\ P(xs)$ by strong structural induction on xs

Show $P(xs)$ for arbitrary $xs :: [Mat]$, assuming IH: $\forall ys \sqsubset xs.\ P(ys)$ ← sublist

Case distinction on xs: ← note the ≡ here

Case $xs \equiv []$:

length (dec [])

= length [] (by D1)

$= 0$ (by length.1)

$=$ srclen [] (by S1)

Case xs ≡ [n] for some n :: Nat : Analogous to prev. case

↑ neu, aber nicht übertreiben

Case xs ≡ (n : v : zs) for some n, v :: Nat, zs :: [Nat]

↑ IH ist jz oben einmal festgelegt, wird bei structural induction praktisch

```
  length (dec (n:v:zs))
= length (rep n v ++ dec zs)                (D3)
= length (rep n v) + length (dec zs)        (L1)
= n + length (dec zs)                       (L2)
= n + srclen zs                             (IH)
= srclen (n:v:zs)                           (S3)
```

↑ hier brauchen wir den „strong" Aspekt der strong structural induction