ETH zürich

P. Müller and D. Basin

Formal Methods and Functional Programming

Session Sheet 9: 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:

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

- 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.

Solution. We define $P(n) \equiv \exists d.\, prime(d) \wedge d \mid n$. We prove $\forall n \geq 2.\, P(n)$ with a strong induction.

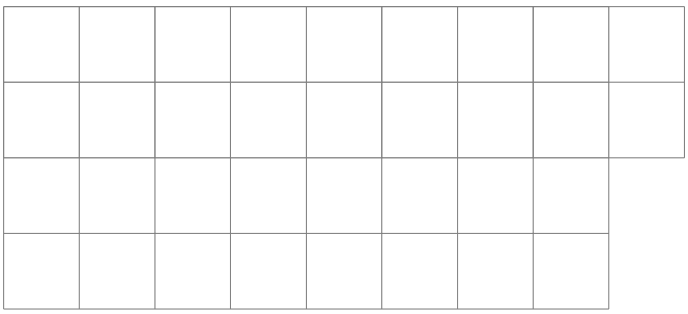
Let n be an integer such that $n \geq 2$. We assume $P(k)$ for all integers k such that $2 \leq k < n$, and we prove $P(n)$, by distinguishing two cases:

Case 1: $prime(n)$, i.e. n is a prime number. Then $n = n \times 1$, thus n is a prime divisor of n, which concludes the case.

Case 2: $\neg prime(n)$, i.e. n is not a prime number. Then, by definition, there must exist an integer d such that $1 < d < n$, and n is divisible by d. Since $2 \leq d < n$, we know that $P(d)$ holds, and thus d has a prime divisor d'. Since d' divides d and d divides n, d' divides n. Moreover, d' is prime, which concludes the case.

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?

Solution. $n - 1$.

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.

Solution. We prove $\forall k \geq 1.\, C(k)$ by strong induction.

Let $k \geq 1$ be arbitrary, and let us assume $C(j)$ for all $1 \leq j < k$. Our goal is to prove $C(k)$. To do this, we consider two cases:

Case 1: $k = 1$. In this case, our chocolate bar is already split into a small square, so we need 0 cut. Moreover, $k - 1 = 1 - 1 = 0$, which concludes the case.

Case 2: $k > 1$. In this case, the chocolate bar can be split into two pieces, with one cut. We thus obtain two chocolate bars with $k_1 \geq 1$ and $k_2 \geq 1$ squares, and we know that $k_1 + k_2 = k$. It follows that $k_1 < k$ and $k_2 < k$, and thus, $C(k_1)$ and $C(k_2)$ are true: We can cut the two smaller bars into small squares with $(k_1 - 1) + (k_2 - 1)$ cuts. Therefore, we can cut the original chocolate bar into small squares with $1 + (k_1 - 1) + (k_2 - 1) = (k_1 + k_2) - 1 = k - 1$ cuts, which concludes the proof.

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.

Solution. In the previous task, we used our induction hypothesis to obtain $C(k_1)$ and $C(k_2)$, where k_1 and k_2 are strictly smaller than k. With a weak induction, we will have to prove $P(k)$ from $P(k-1)$ (assuming $k \geq 2$), where P is the property we will prove by weak induction. Thus, to mimic the previous proof, we need to be able to deduce $C(k_1)$ and $C(k_2)$ from $P(k-1)$. The key idea is to put all $C(j)$ into $P(k)$, for $j \leq k$, as follows:

Let $P(k) \equiv \forall j.\, 1 \leq j \leq k \Rightarrow C(j)$ be our induction hypothesis, which we prove for all $k \geq 1$ by weak induction.

Base case: Let us prove $P(1)$, i.e. $\forall j.\, 1 \leq j \leq 1 \Rightarrow C(j)$. To do this, let j be an arbitrary integer, and let us assume that $1 \leq j \leq 1$. j must be 1, so we need to prove $C(1)$. We need $1 - 1 = 0$ cut to split into small squares a chocolate bar containing 1 square, which concludes the proof.

Induction step: Let $k \geq 1$ be arbitrary. We assume $P(k)$, and we have to prove $P(k+1)$, i.e., $\forall j.\, 1 \leq j \leq k+1 \Rightarrow C(j)$.

Let j be an arbitrary integer such that $1 \leq j \leq k+1$. We distinguish two cases:

Case 1: $1 \leq j \leq k$. In this case, we get $C(j)$ from $P(k)$, which proves the case.

Case 2: $j = k+1$. In this case, we do the same proof as in the strong induction: the chocolate bar can be split into two pieces, with one cut. We thus obtain two chocolate bars with $k_1 \geq 1$ and $k_2 \geq 1$ squares, and we know that $k_1 + k_2 = k + 1$. It follows that $1 \leq k_1 \leq k$ and $1 \leq k_2 \leq k$. Thus, from $P(k)$ (with $j = k_1$ and $j = k_2$), we know that $C(k_1)$ and $C(k_2)$ hold, i.e. we can cut the two smaller bars into small squares with $(k_1 - 1) + (k_2 - 1)$ cuts. Therefore, we can cut the original chocolate bar into small squares with $1 + (k_1 - 1) + (k_2 - 1) = (k_1 + k_2) - 1 = (k + 1) - 1$ cuts, which proves $C(j)$, and thus concludes the case.

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:[]`.

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.,

$\forall xs \cdot$ `length (dec` xs`) = srclen` xs

[1] `http://en.wikipedia.org/wiki/Run-length_encoding`

[2] We include zero in the natural numbers.

Solution. We observe that weak structural induction would fail in the case $xs \equiv$ (x:y:zs). Therefore, we do the proof by strong structural induction.

We define $P(xs) \equiv$ `length (dec` xs`) = srclen` xs and prove $\forall xs \cdot P(xs)$ by strong structural induction on xs. Thus, we have to show that $P(xs)$ holds for some arbitrary xs :: [`Nat`] and may assume $\forall ys \sqsubset xs \cdot P(ys)$ as our induction hypothesis. We proceed by a case analysis on xs:

- **Case** $xs \equiv$ `[]`:

$$\texttt{length (dec [])} = \texttt{length []} = 0 = \texttt{srclen []} \qquad \text{(D1,S1)}$$

- **Case** $xs \equiv$ [n] for some n :: `Nat`: Analogous to the previous case.
- **Case** $xs \equiv$ (n:v:zs) for some n, v :: `Nat` and zs :: `[Nat]`:

$$\begin{aligned}
& \texttt{length (dec (}n\texttt{:}v\texttt{:}zs\texttt{))} & \\
&= \texttt{length (rep } n\ v \texttt{ ++ dec } zs\texttt{)} & \text{(D3)} \\
&= \texttt{length (rep } n\ v\texttt{) + length (dec } zs\texttt{)} & \text{(L1)} \\
&= n \texttt{ + length (dec } zs\texttt{)} & \text{(L2)} \\
&= n \texttt{ + srclen } zs & \text{(IH)} \\
&= \texttt{srclen (}n\texttt{:}v\texttt{:}zs\texttt{)} & \text{(S3)}
\end{aligned}$$