Recursive definitions

- A recursive definition of a set begins with a *basis statement* that specifies one or more elements in the set. The *recursive part* of the definition involves one or more operations that can be applied to elements already known to be in the set, so as to produce new elements of the set.

Example: let $AnBn$ be the language over $\Sigma = \{a, b\}$ defined as $AnBn = \{a^n b^n \mid n \in \mathbb{N}\}$. Its recursive definition is

1. $\Lambda \in AnBn$
2. For every $x \in AnBn$, $axb \in AnBn$.

Example: recursive definition of PAL over $\Sigma = \{a, b\}$

1. $\Lambda, a, b \in PAL$
2. For every $x \in PAL$, $axa \in PAL$ and $bxb \in PAL$.

Recursive definitions of functions

Example 1: factorial function $n! = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 1$

$$f(0) = 1; \text{ for every } n \in \mathbb{N}, \; f(n+1) = (n+1) \cdot f(n)$$

Different notation

$$f(n) = \begin{cases} 1 & n = 0 \\ nf(n-1) & \text{otherwise} \end{cases}$$

Example 2

The function $f(n) = 2n + 1$ for natural numbers n can be defined recursively

$$f(0) = 1; \text{ for every } n \in \mathbb{N}, \; f(n+1) = f(n) + 2$$

Example: $EXPR$ is a language of legal algebraic expressions involving the identifier a, the binary operations $+$ and $*$, and parentheses,i.e.,

$$\Sigma = \{a, +, *, (,)\}.$$

Some of the strings in the language are

$$a,\ \ a + a * a,\ \ \text{and}\ (a + a * (a + a)).$$

Its recursive definition:

These are just strings, you don't need to compute them

1. $a \in EXPR$
2. $\forall x, y \in EXPR$, $x + y$ and $x * y \in EXPR$
3. $\forall x \in EXPR$, $(x) \in EXPR$

Example: a recursive definition of a set of languages over $\{a, b\}$. We denote by $\mathcal{F}$ the subset of $2^{\{a,b\}^*}$ (the set of languages over $\{a, b\}$) defined as follows:

set of all subsets of {a,b}*

1. $\emptyset$, $\{\Lambda\}$, $\{a\}$, and $\{b\}$ are elements of $\mathcal{F}$
2. $\forall L_1, L_2 \in \mathcal{F}$, $L_1 \cup L_2 \in \mathcal{F}$
3. $\forall L_1, L_2 \in \mathcal{F}$, $L_1 L_2 \in F$

Given language *L,* typical questions that one can ask are to (dis)prove that:

- string *x* belongs to *L*
- all (some) *x* in *L* have some property
- another language is a subset of L, etc.
- L* can contain certain strings
- recursive definition of L is …

(Example: compiler + source code.)

1. $a \in EXPR$
2. $\forall x, y \in EXPR$, $x + y$ and $x * y \in EXPR$
3. $\forall x \in EXPR$, $(x) \in EXPR$

Suppose we want to prove that every string $x \in EXPR$ satisfies the statement $P(x)$.

The principle of **structural induction** uses recursive definition of language. It says that in order to show that $P(x)$ is true for every $x \in EXPR$, it is sufficient to show:

Induction hypothesis

1. $P(a)$ is true
2. $\forall x, y \in EXPR$, $\boxed{\text{if } P(x) \text{ and } P(y) \text{ are true,}}$ then $P(x + y)$ and $P(x * y)$ are true.
3. $\forall x \in EXPR$, $\boxed{\text{if } P(x) \text{ is true,}}$ then $P((x))$ is true.

operations on *x*, *y* generate strings for which P() is true length of *x*

Claim. *For every element x of $EXPR$, $|x|$ is odd.*

Proof.

- **Basis step.** We wish to show that $|a|$ is odd. This is true because $|a| = 1$.
- **Induction hypothesis.** $x, y \in EXPR$, and $|x|$,$|y|$ are odd.
- **Statement to be proved in the induction step.**

$$|x + y|,\ |x * y|,\ \text{and } |(x)| \text{ are odd.}$$

length of *x*

Claim. *For every element x of $EXPR$, $|x|$ is odd.*

Proof.

Proof of induction step.

- The lengths
$$|x+y| \text{ and } |x*y|$$
are both $|x|+|y|+1$, because the symbols of $x+y$ include those in x, those in y, and the additional "operator" symbol.
- The length $|(x)|$ is $|x|+2$, because two parentheses have been added to the symbols of x.
- The first number is odd because the induction hypothesis implies that it is the sum of two odd numbers plus 1, and the second number is odd because the induction hypothesis implies that it is an odd number plus 2.

$\square$

Sometimes stronger statements are easier to prove

$EXPR$ is a language of legal algebraic expressions.

1. $a \in EXPR$
2. $\forall x, y \in EXPR$, $x + y$ and $x * y \in EXPR$
3. $\forall x \in EXPR$, $(x) \in EXPR$

Prove that no string in $EXPR$ can contain the substring $++$.

No problem with basis step, hypothesis, and concluding about $x*y$, and (x). Proving that $x+y$ doesn't contain $++$ is more difficult. Neither x nor y contains $++$ as a substring, but if x ended with $+$ or y started with $+$, then $++$ would occur in the concatenation.

Stronger statement: $\forall x \in EXPR$, x doesn't begin or end with $+$ **and** doesn't contain $++$. Complete proof at home.

Defining functions on sets defined recursively

- For $\Sigma = \{a, b\}$, we have $\{a, b\}^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$
- Recursive definition of set $S = \{a, b\}^*$ is

$$\Lambda \in S; \ \forall x \in S \ xa, xb \in S$$

- Example: let us consider the **reverse function** $r : \Sigma^* \to \Sigma^*$ that is defined recursively by referring to the recursive definition of Σ^*

$$r(x) = \begin{cases} \Lambda & x = \Lambda \\ ar(y) & x = ya, \ y \in \Sigma^* \\ br(y) & x = yb, \ y \in \Sigma^* \end{cases}$$

output of *r* if input of *r* is …

Reverse function *r(x)* reverses input string *x*. Example: *x = abac, r(x) = caba*

To illustrate the close relationship between the recursive definition of *{a, b}**, the recursive definition of *r*, and the principle of structural induction, we prove the following fact about the reverse function.

Claim. $\forall x, y \in \{a, b\}^*$, $r(xy) = r(y)r(x)$.

Note that we have two variables x, and y, i.e., "$\forall x, y$" is translated to "$\forall x$, and $\forall y$". In such cases the quantifiers are nested; we can write the statement in the form $\forall y\ P(y)$, where $P(y)$ is itself a quantified statement, $\forall x(...)$, and so we can attempt to use structural induction on y.

We prove $\forall y \in \Sigma^*$, $P(y)$ is true, where $P(y)$ is the statement "$\forall x \in \Sigma^*$, $r(xy) = r(y)r(x)$".

Proof. We prove $\forall y \in \Sigma^*$, $P(y)$ is true, where $P(y)$ is the statement "$\forall x \in \Sigma^*$, $r(xy) = r(y)r(x)$".

i.e., y is an empty string

Basic step. We need to prove the statement "$\forall x \in \Sigma^*$, $r(x\Lambda) = r(\Lambda)r(x)$." (complete at home)

Induction hypothesis. $y \in \Sigma^*$, and $\forall x \in \Sigma^*$, $r(xy) = r(y)r(x)$.

Statement to be proved in induction step. $\forall x \in \Sigma^*$, $r(x(ya)) = r(ya)r(x)$, and $r(x(yb)) = r(yb)r(x)$.

$$\begin{aligned}
r(x(ya)) &= r((xy)a) && \text{concatenation is associative} \\
&= a(r(xy)) && \text{by the definition of } r \\
&= a(r(y)r(x)) && \text{by the induction hypothesis} \\
&= (ar(y))r(x) && \text{concatenation is associative} \\
&= r(ya)r(x) && \text{by the definition of } r.
\end{aligned}$$

The second part of the proof is similar. Complete it in the assignment! □

Claim. *Prove that for every non-empty language $L \subseteq \{a, b\}^*$,*

$$if \quad L^2 \subseteq L, \quad then \quad LL^* \subseteq L.$$

Proof sketch. If $x \in LL^*$ then $x = x_1x_2$, where $x_1 \in L$, and $x_2 \in L^*$.

- if $x_2 = \Lambda$ - trivial; if $x = \Lambda$ - trivial
- otherwise $\exists y_1, \cdots, y_k \in L$ $s.t.$ $x = y_1y_2 \cdots y_k$, where $\forall y_i \neq \Lambda, 1 \leq i \leq k$.
- We prove by induction on k (the number of strings y_i)
  - Basis: k=1 - trivial
  - Hypothesis: the statement is true for $k - 1$
  - Induction: $x = y_1y_2 \cdots y_k = zy_k$, where $z = y_1y_2 \cdots y_{k-1}$. By the hypothesis $z \in L$, then $zy_k \in L^2$, and then $x \in L$, because $L^2 \subseteq L$.

You can also prove it by induction on L^k without splitting into substrings.