Session 48: Recursively Defined Sets and Structures

- Recursive definitions of sets
- Natural Numbers and Strings
- Recursively defined functions
- Well-formed Formulae

Recursively Defined Sets and Structures

Recursion can be also used to define sets

Recursive definitions of sets have two parts:

- The **basis step** specifies an initial collection of elements.
- The **recursive step** gives the rules for forming new elements in the set from those already known to be in the set.

- Only elements generated in the basis and recursive steps belong to the set (**exclusion rule**)

Examples

Recursive definition of the set of natural numbers **N:**

BASIS STEP: 0 ∊ **N.**

RECURSIVE STEP: If *n* is in **N**, then *n + 1* is in **N**.

$\mathbb{N} = \{0\}, \mathbb{N} = \{0,1\}, \mathbb{N} = \{0,1,2\}, \ldots$

A subset of Integers *S3*:

BASIS STEP: 3 ∊ S3.

RECURSIVE STEP: If *x* ∊ *S3* and *y* ∊ *S3*, then *x + y* is in *S3.*

$S_3 = \{3\}, S_3 = \{3,6\}, S_3 = \{3,6,9,12\}, \ldots$

Strings

Definition: The set Σ^* of *strings* over the alphabet Σ:

BASIS STEP: $\lambda \in \Sigma^*$ (λ is the empty string)

RECURSIVE STEP: If w is in Σ^* and x is in Σ, then $wx \in \Sigma^*$.

The alphabet Σ is a finite set, e.g.

- $\Sigma = \{0, 1\}$
- $\Sigma = \{a, b, \ldots., z\}$

Examples

If Σ = {0, 1}, the strings in in Σ* are the set of all bit strings:

$\Sigma^* = \{\lambda\}, \Sigma^* = \{\lambda, 0, 1\}, \Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11\}, \ldots$

If Σ = {*a*, *b*}, showing that *aab* is in Σ*:

since $\lambda \in \Sigma^*$, $a \in \Sigma$ we have $a \in \Sigma^*$

since $a \in \Sigma^*$, $a \in \Sigma$ $\quad aa \in \Sigma^*$

$aa \in \Sigma^*$, $b \in \Sigma$ $\quad aab \in \Sigma^*$

Recursively defined functions on recursively defined sets

We can define functions by recursion on recursively defined sets

Example: Give a recursive definition of $l(w)$, the **length of the string** w.

Base Step : $l(\lambda) = 0$

Recursive Step : for $w \in \Sigma^*$, $x \in \Sigma$ $\quad l(wx) = l(w) + 1$

String Concatenation

Definition: The **concatenation** of two strings w_1 and w_2, denoted by $w_1 \cdot w_2$, is defined recursively as follows.

BASIS STEP: If $w \in \Sigma^*$, then $w \cdot \lambda = w$.

RECURSIVE STEP: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

- Often $w_1 \cdot w_2$ is written as $w_1 w_2$.

Example: $\Sigma = \{0,1\}$, Σ^* all finite bitstrings, $w \in \Sigma^*$

Base Step: $w \cdot \lambda = w$

Recursive Step: $w \cdot (\lambda 0) = (w \cdot \lambda) 0 = w0$ e.g. $01 \cdot (\lambda 0) = (01 \cdot \lambda) 0 = 010$

Well-Formed Formulae in Propositional Logic

Definition: The set of **well-formed formulae** in propositional logic involving **T**, **F**, propositional variables, and operators from the set $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ is recursively defined as

BASIS STEP: **T**, **F** and *s*, where *s* is a propositional variable, are well-formed formulae.

RECURSIVE STEP: If *E* and *F* are well formed formulae, then $(\neg E)$, $(E \wedge F)$, $(E \vee F)$, $(E \rightarrow F)$, $(E \leftrightarrow F)$, are well-formed formulae.

Examples: $((p \vee q) \rightarrow (q \wedge F))$ well-formed

$p \wedge q$ not well-formed

$p \wedge q \wedge r$ "

$p q \wedge$ "

Summary

- Recursive definitions of sets
- Natural Numbers and Strings
- Recursively defined functions
- Well-formed Formulae