Formal Language and Automata Theory

Course outlines

- # Introduction:
  - **Mathematical preliminaries:**
    - ✪sets, relations, functions,sequences, graphs, trees, proof by induction, definition by induction (recursion).
  - **Basics of formal languages:**
    - ✪alphabet, word, sentence, concatenation ,union, iteration [= Kleene star], language, infinity of languages, finite representations of languages
- # PART I: Finite Automata and Regular Sets
  - **DFA,NFA,regular expressions and their equivalence**
  - **limitation of FAs;**
  - **Closure properties of FAs,**
  - **Optimization of FAs**

course outline (cont'd)

- **PART II: Pushdown Automata and Context Free Languages**
  - CFGs and CFLs; normal forms of CFG
  - Limitation of CFG; PDAs and their variations,
  - closure properties of CFLs
  - Equivalence of pda and CFGs; deterministic PDAs
  - parsing (Early or CYK's algorithms)
- **PART III: Turing Machines and Effective Computability**
  - Turing machine [& its variations] and Equivalence models
  - Universal TMs
  - Decidable and undecidable problems (Recursive sets and recursively enumerable sets)
  - Problems reductions ; Some undecidable problems

Goals of the course

- **understand the foundation of computation**
- **make precise the meaning of the following terms:**
  - **[formal] languages, problems, Programs, machines, computations**
  - ***computable* {languages, problems, sets, functions}**
- **understand various models of machines and their relative power : FA, PDAs, LA (linear bounded automata), TMs, [register machines, RAMs,...]**
- **study various representations of languages in finite ways via grammars: RGs, CFGs, CSGs, general PSGs**

Chapter 1 Introduction

Mathematical preliminaries (reviews)

- sets (skipped)
- functions (skipped)
- relations
- induction
- Recursive definitions

Sets

- Basic structure upon which all other (discrete and continuous) structures are built.
- a set is a collection of objects.
   - an object is anything of interest, maybe itself a set.
- Definition 1.
   - A set is a collection of objects.
   - The objects is a set are called the elements or members of the set.
   - If x is a memebr of a set S, we say S contains x.
   - notation: $x \in S$ vs $x \notin S$
- Ex: In 1,2,3,4,5, the collection of 1,3 5 is a set.

Set description

- **How to describe a set:?**

1. List all its member.

- the set of all positive odd integer >10 = ?
- The set all decimal digits = ?
- the set of all upper case English letters = ?
- The set of all nonnegative integers   = ?

2. Set builder notation:

- P(x) : a property (or a statement or a proposition) about objects.
- e.g., P(x) = " x > 0 and x is odd"
- then {x | P(x) } is the set of objects satisfying property P.
- P(3) is true => 3 $\in$ {x | P(x)}
- P(2) is false => 2 $\notin$ {x | P(x)}

Set predicates

Definition 2.

- Two sets S1, S2 are equal iff they have the same elements
- $S1 = S2$ iff $\forall x\ (x \in S1 \Leftrightarrow x \in S2)$
- Ex: $\{1,3,5\} = \{1,5,3\} = \{1,1,3,3,5\}$

- Null set $=\{\} = \emptyset =_{def}$ the collection of no objects.

Def 3': [empty set] for-all x $x \notin \emptyset$.

Def 3. [subset]

- $A \subseteq B$ iff all elements of A are elements of B.
- $A \subseteq B \Leftrightarrow$ for-all x $(x \in A \Rightarrow x \in B)$).

- Def 3'': $A \subset B =_{def} A \subseteq B \wedge A \neq B$.
- Exercise : Show that: 1. For all set A $(\emptyset \subseteq A)$
  - 2. $(A \subseteq B \wedge B \subseteq A) \Leftrightarrow (A = B)$ 3. $A \subseteq \emptyset \Rightarrow A = \emptyset$

Size or cardinality of a set

Def. 4

- | A | = the size(cardinality) of A = # of distinct elements of A.

- **Ex:**
  - |{1,3,3,5}| = ?
  - |{}| = ?
  - | the set of binary digits } | = ?
  - |N| = ? ; |Z| = ? ; | {2i | i in N} = ?
  - |R| = ?
- **Def. 5.**
  - A set A is finite iff |A| is a natural number ; o/w it is infinite.
  - Two sets are of the same size (cardinality) iff there is a 1-1 & onto mapping between them.

countability of sets

- Exercise: Show that
  - 1. |N| = |Z| = | Q | = {4,5,6,...}
  - 2. |R| = | [0, 1) |
  - 3. |N| $\neq$ |R|
- Def.
  - A set A is said to be denumerable iff |A| = |N|.
  - A set is countable (or enumerable) iff either |A| = n for some n in N or |A| = |N|.
- By exercise 3,
  - R is not countable.
  - Q and Z is countable.

The power set

Def 6.

- If A is a set, then the collection of all subsets of A is also a set, called the poser set of A and is denoted as P(A) or 2^A.

- Ex:
  - P({0,1,2}) = ?
  - P({}) = ?
  - |P({1,2,..., n})| = ?
- Order of elements in a set are indistinguishable. But sometimes we need to distinguish between (1,3,4) and (3,4,1) --> ordered n-tuples

More about cardinality

Theorem: for any set A, $|A| \neq |2^A|$.

Pf: (1) The case that A is finite is trivial since $|2^A| = 2^{|A|} > |A|$ and there is no bijection b/t two finite sets with different sizes.

(2) assume $|A| = |2^A|$, i.e., there is a bijection $f: A \to 2^A$.

Let $D = \{x \text{ in } A \mid x \notin f(x)\}$. ==>

1. D is a subset of A; Hence

2. $\exists y$ in A s.t. $f(y) = D$.

Problem: Is $y \in D$?

if yes (i.e., $y \in D$) ==> $y \notin f(y) = D$, a contradiction

if no (i.e., $y \notin D$) ==> $y \in f(y) = D$, a contradiction too.

So the assumption is false, i.e., there is no bijection b/t A and 2^A.

Note: Many proofs of impossibility results about computations used arguments similar to this.

Cartesian Products

Def. 7 [n-tuple]

- If a1,a2,...,an (n > 0) are n objects, then "(a1,a2,...,an)" is a new object, called an (ordered) n-tuple [with a_i as the ith elements.
- Any orderd 2-tuple is called a pair.
- (a1,a2,...,am) = (b1,b2,...,bn) iff
  - m = n and for i = 1,..,n $a_i = b_i$.

Def. 8: [Cartesian product]

A x B $=_{def}$ {(a,b) | a in A /\ b in B }

A1 x A2 x ...x An =def {(a1,...,an) | ai in Ai }.

Ex: A = {1,2}, B = {a,b,c} , C = {0,1}

1. A x B = ? ; 2. B x A = ?
3. A x {} = ? ;4. A x B x C = ?

Set operations

- union, intersection, difference , complement,
- Definition.

1. $A \cup B = \{x \mid x \text{ in } A \text{ or } x \text{ in } B\}$
2. $A \cap B = \{x \mid x \text{ in } A \text{ and } x \text{ in } B\}$
3. $A - B = \{x \mid x \text{ in } A \text{ but } x \text{ not in } B\}$
4. $\sim A = U - A$
5. If $A \cap B = \{\} \Rightarrow$ call A and B disjoint.

Set identites

- Identity laws: A ? ? = A
- Domination law: U ? ? = U; {} ?? = {}
- Idempotent law: A ? A = A ;
- complementation: ~~A = A
- commutative : A ? B = B ? A
- Associative: A ? (B ? C) = (A ? B) ? C
- Distributive: A ? (B ? C) = ?
- DeMoregan laws: ~(A ? B) = ~A ? ~B

Note: Any set of objects satisfying all the above laws is called a Boolean algebra.

Prove set equality

1. Show that $\sim(A \cup B) = \sim A \cap \sim B$ by show that
   - 1. $\sim(A \cup B) \subseteq \sim A \cap \sim B$
   - 2. $\sim A \cap \sim B \subseteq \sim(A \cup B)$

- pf: (By definition) Let x be any element in $\sim(A \cup B)$ ...

2. show (1) by using set builder and logical equivalence.

3. Show distributive law by using membership table.

4. show $\sim(A \cup (B \cap C)) = (\sim C \cap \sim B) \cup \sim A$ by set identities.

Functions

- Def. 1 [functions]   A, B: two sets

1. a function f from A to B is a set of pairs (x, y) in AxB s.t., for each x in A there is at most one y in B s.t. (x,y) in f.
2. if (x,y) in f, we write f(x) = y.
3. f :A ->B means f is a function from A to B.

Def. 2. If f:A -> B ==>

1. A: the domain of f;   B: the codomain of f

if f(a)=b =>

2. b is the image of a;   3. a is the preimage of b
4. range(f) = {y | ∃x s.t. f(x) = y} = f(A).
5. preimage(f) = {x | $ y s.t. f(x) = y } = f^{-1}(B).
6. f is total iff f^{-1}(B) = A.

Types of functions

- **Def 4.**

f: A x B; S: a subset of A,

T: a subset of B

1. $f(S) =_{def} \{y \mid \exists\, x \text{ in } S \text{ s.t. } f(x) = y\}$
2. $f^{-1}(T) =_{def} \{x \mid \exists\, y \text{ in } T \text{ s.t. } f(x) = y\}$

Def. [1-1, onto, injection, surjection, bijection]

f: A -> B.

- f is 1-1 (an injection) iff f(x)=(fy) => x = y.
- f is onto (surjective, a surjection) iff f(A) = B
- f is 1-1 & onto <=> f is bijective (a bijection, 1-1 correspondence)

Relations

- **A, B: two sets**
  - AxB (Cartesian Product of A and B) is the set of all ordered pairs {<a,b> | a $\in$ A and b $\in$ B }.
  - Examples:

    A= {1,2,3}, B= {4,5,6} => AxB = ?

- **A1,A2,…,An (n > 0): n sets**
  - A1xA2x...xAn = {<a1,a2,...,an> | ai $\in$ Ai }.
  - Example:

    1. A1={1,2},A2={a,b},A3={x,y} ==> |A1xA2xA3| = ?
    2. A1= {}, A2={a,b} => A1xA2 = ?

Binary relations

- **Binary relation:**
  - A,B: two sets
  - A binary relation R between A and B is any subset of AxB.
  - Example:

If A={1,2,3}, B ={4,5,6}, then which of the following is a binary relation between A and B ?

R1 = {<1,4>, <1,5>, <2,6> }

R2 = {}

R3 = {1,2,3,4}

R4 = {<1,2>, <3,4>, <5,6> }

Terminology about binary relations

- R: a binary relation between A and B (I.e., a subset of AxB), then
  - The domain of R:

    dom(R) = {x ∈ A | ∃ y ∈ B s.t. <x,y> ∈ R}
  - The range of R:

    range(R) ={y ∈ B, | ∃ x ∈ A, s.t., <x,y> ∈ R}
  - <x,y> ∈ R is usually written as x R y.
- If A = B, then R is simply called a relation over(on) A.
- An n-tuple relation R among A1,A2,...,An is any subset of A1xA2...xAn, n is called the arity of R
- If A1=A2=...=An=A => R is called an n-tuple relation (on A),.

Operations on relations (and functions)

- $R \subseteq AxB$; $S \subseteq B \times C$: two relations
- composition of R and S:
  - $R \cdot S$ = {<a,c> | there is b in B s.t., <a,b> in R and <b,c> in S }.
- Identity relation: I_A = {<a,a> | a in A }
- Converse relation: R^{-1} = {<b,a> | <a,b> in R }
- f:A -> B; g: B->C: two functions, then

  $g \cdot f$:A->C defined by $g \cdot f(x) = g(f(x))$.
- Note: function and relation compositions are associative, I.e., for any function or relation f,g,h,

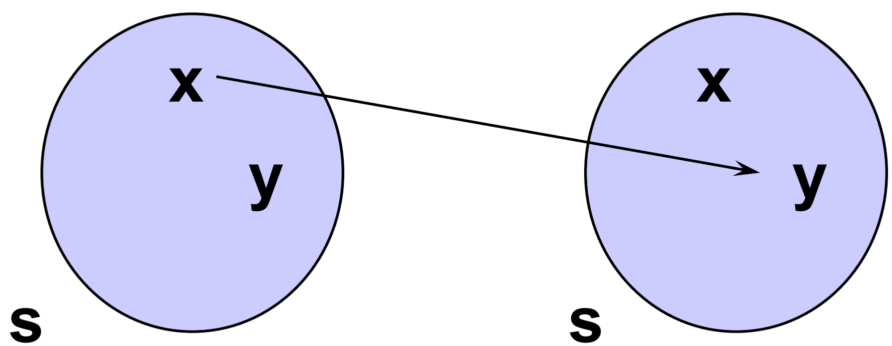
  $f \cdot (g \cdot h) = (f \cdot g) \cdot h$

Properties of binary relations

- R: A binary relation on S,

1. R is *reflexive* iff for all x in S, x R x.
2. R is *irreflexive* iff for all x in S, not x R x.
3. R is *symmetric* iff for all x, y in S, xRy => yRx.
4. R is *asymmetric* iff for all x,y in S, xRy => not yRx.
5. R is *antisymmetric* iff for all x,y in S, xRy and yRx => x=y.
6. R is *transitive* iff for all x,y,z in S, xRy and yRz => xRz.

Graph realization of a binary relation and its properties.



rule: if xRy then draw an arc from x on left S to y on right S.

Examples

- The relation ≤ on the set of natural numbers N.

  What properties does ≤ satisfy ?
  - ref. irref, or neither ?
  - symmetric, asymmetric or antisymmetric ?
  - transitive ?
- The relation > on the set of natural numbers N.
- The divide | relation on integers N ?
  - x | y iff x divides y. (eg. 2 | 10, but not 3 | 10)

  What properties do > and | satisfy ?
- The BROTHER relation on males (or on all people)
  - (x,y) ∈ BROTHER iff x is y's brother.
- The ANCESTOR relation on a family.
  - (x,y) ∈ ANCESTOR if x is an ancestor of y.

What properties does BROTHER(ANCESTOR) have?

Properties of relations

- R: a binary relation on S

1. R is a *preorder* iff it is ref. and trans.
2. R is a *partial order* (p.o.) iff R is ref.,trans. and antisym. (usually written as ≤).
3. R is a *strict portial order* (s.p.o) iff it is irref. and transitive.
   - usually written <.
4. R is a *total (or linear) order* iff it is a partial order and every two element are *comparable* (i.e., for all x,y either xRy or yRx.)
5. R is an *equivalence relation* iff it is ref. sym. and trans.

- If R is a preorder (resp. po or spo) then (S,R) is called a *preorder set* (resp. *poset*, *strict poset*).
- What order set do (N, <) , (N, ≤) and (N, |) belong to ?

Properties of ordered set

- (S, ≤): a poset, X: a subset of S.

1. b in X is the *least (or called minimum) element* of X iff b ≤ x for all x in X.
2. b in X is the *greatest (or called maxmum or largest) element* of X iff X ≤ b for all x in X.

- Least element and greatest element, if existing, is unigue for any subset X of a poset (S, ≤)

pf: let x, y be least elements of X.

Then, x≤ y and y≤ x. So by antisym. of ≤, x = y.

3. X ia a chain iff (X,R) is a linear order(, i.e., for all x, y in X, either x≤y or y≤x) .
4. b in S is a *lower bound* (resp., *upper bound*) of X iff b ≤ x (resp., x ≤ b) for all x in X.
   - Note: b may or may not belong to X.

Properties of oredered sets

- (S, ≤) : a poset, X: a nonempty subset of S.

5. b in X is ***minimal*** in X iff there is no element less than it.
   - i.e., there is no x in X, s.t., (x < b),
   - or "for all x, x ≤ b => x =b."
6. b in X is a ***maximal*** element of X iff there is no element greater then it.
   - i.e., there is no x in X, s.t., (b < x),
   - or "for all x, b ≤ x => x=b."

- Note:

1. Every maximum element is maximal, but not the converse in general.
2. Maximal and minimal are not unique in general.

well-founded set and minimum conditions

- (S,≤) : a poset (偏序集).

1. ≤ is said to be ***well-founded*** **(良基性)** iff there is no infinite descending sequence. (i.e., there is no infinite sequence x1,x2,x3,.... s.t., x1 > x2 > x3 >...).
   - Note: x > y means y < x (i.e., y≤ x and y = x)
   - if ≤ is well-founded => (S,≤) is called a well-founded set.
2. (S,≤) is said to satisfy *the* ***minimal condition*** iff every nonempty subset of S has a minimal element.

- (S, ≤): a total ordered set (全序集).

3. ≤ is said to be a **well-ordering(良序)** iff every nonempty subset of S has a least element.
   - If ≤ is well ordered, then (S,≤) is called a well-ordered set.

Examples of ordered sets

- Among the relations $(N,\leq)$, $(N,\geq)$, $(N, |)$, $(Z, \leq)$, $(Z,\geq)$, $(Z,|)$ and $(R, \leq)$,

1. Which are well-founded ?
2. Which are well-ordered ?

Equivalence of well-foundness and minimal condition

- $(S,\leq)$ is well-founded (w.f.) iff it satisfies the minimal conditions (m.c.).

pf: scheme: (1) not w.f => not m.c. (2) not m.c. => not w.f.

(1) Let x1,x2,... be any infinite sequence s.t. x1 > x2 > x3 >... .

Now let X={x1,x2,...}. X obviously has no minimal element.

S thus does not satisfy m.c.

(2) Let X be any nonempty subset of S w/o minimal elements. Now

(*) choose arbitrarily an element a1 from X and let

$X1 = \{x \mid x \in X \text{ and } a1 > x \}$ (i.e. the set of all elements in X < a1).

Since a1 is not minimal, X1 is nonempty and has also no minimal element.

We can then repeat the procedure (*) infinitely to find a2, X2, a3, X3,... and obtain an infinite descending sequence a1 > a2 > a3 > ...

Hence S is not w.f.

A general proof scheme to show the infinity of a set

- Let P be a property of sets and C $=_{def}$ { X | P(X) holds }. If there exists a function f : C → C satisfying the property: forall set X $\in$ C (i.e, P(X) holds),
   - 1. f(X) is a nonempty proper subset of X,
   - (i.e., f(X) $\neq \emptyset$, f(X) $\neq$ X and f(X) $\subset$ X), and
   - 2. f preserves property P
   - (I.e., P(X) implies P(f(X), or X$\in$C => f(X)$\in$C),

  then all sets X with property P are infinite .

Pf: Let $X_0, X_1, \ldots, X_k, \ldots$ be an infinite sequence of sets with X_0 = X and $X_{k+1} =_{def} f(X_k) = f(f(X_{k-1})) = \ldots = f^{k+1}(X)$.
Since X = X_0 has property P and f preserves P, by induction on k, all X_k has property P. And by (1) $X_k \subset f(X_k) = X_{k+1}$ for all k.
Now the sequence X_0-X_1, X_1-X_2, X_2-X_3,…. is an infinite sequence of nonempty and disjoint subsets of X. X thus is infinite.

Variants of Inductions

- **Mathematical Induction:**
  - To prove a property P(n) holds for all natural number $n \in N$, it suffices to show that
    - (1) P(0) holds --- (base step) and
    - (2) For all $n \in N$, p(n) => p(n+1) --- (induction step)
    - ✪P(n) in (2) is called *induction hypothesis* (h.p.)

$$\frac{P(0),\ \forall n\ (P(n) \Rightarrow P(n+1))}{\forall n\ P(n)}\ \text{MI1}$$

= $\forall$m. m<n → P(m) // Ind. Hyp.

$$\frac{P(0),\ \forall n\ (\ \underline{(P(0) \wedge \ldots p(n-1))}\ ) \Rightarrow P(n)\ )}{\forall n\ P(n)}\ \text{MI2}$$

Well-order Induction

- **Well-order induction:**
  - (S,≤) a well-ordered set; P(x): a property about S.
  - To show that P(x) holds for all x∈S, it suffices to show
    - (1) $P(x_{min})$ holds where x_{min} is the least element of S. --- (base step)
    - (2) for all x∈S, if (for all y∈S y < x => P(y)) then p(x) ---(ind. step)

    ✪ (1) is a special case of (2) [i.e., (2) implies (1)]

    ✪ (for all y in S y < x => P(y)) in (2) is called the ind. hyp. of (2).

- $$\frac{P(X_{min}),\quad \forall y\ [\ (\forall x.\ x< y \Rightarrow P(x)\ )\Rightarrow P(y)}{\forall x\ P(x)}\ \mathrm{WI}$$

Variants of inductions

- **Well-founded induction (WI):**
  - (S, ≤) a well-founded set. P(x) a property about S.
  - WI says that to prove that P(x) holds for all x in S, it suffices to show that
    - (1) P(x) holds for all minimal elements x in S --- base step, and
    - (2) for all y in S, (for all z in S z < y => P(z)) => p(y) ---ind. step
    - ✪ (1) has already been implied by (2)
    - ✪ (for all z in S z < y => P(z)) in (2) is the ind. hyp. of the proof.

  - forall minimal x, P(x), $\forall$y [($\forall$z, z<y =>P(x)) => P(y)]

  -- WI

  $\forall$x P(x)

- **Facts: w.f. Ind. => well-ordered ind. => math ind.**
  - (I.e., If w.f ind. is true, then so is well-ordered ind. and if well-ordered ind. is true , then so is math. ind.)

Correctness of WI

- (S,≤) : a well-founded set.   P(x) : a property about S.

 Then P(x) holds for all x ∈ S, if

 (1) P(x) holds for all minimal elements x ∈ S --- base step, and

 (2) for all y ∈ S, *(for all z ∈ S z < y => P(z))* => p(y) ---ind. Step

pf: Suppose WI is incorrect. Then there must exist (S,≤) satisfying (1)(2) but

 the set NP = { x | x ∈ S and P(x) is not true } is not empty. (*)

==> Let xm be any minimal element of NP.

case (1): xm is minimal in S. --> impossible! (violating (1))

 since if xm is minimal in S, by (1), P(xm) holds and xm ∉ NP.

case (2): xm is not minimal in S. --> still impossible!

 xm not minimal in S ==> L = {y | y ∈ S /\ y < xm } is not empty.

 ==> L ∩ NP = { } (o/w xm would not be minimal in NP.)

 ==> (ind. hyp. holds for xm , i.e., for all z ∈ S, z < xm => p(z) is true)

 ==> (by (2).) p(xm) holds ==> xm ∉ NP.

case(1) and (2) imply NP has no minimal element and hence is empty, which contradicts with (*). Hence the assumption that WI is incorrect is wrong.

Definition by induction (or recursion)

- **Consider the following ways of defining a set.**

1. the set of even numbers Even = {0,2,4,...}:
  - Initial rule : $0 \in$ Even.
  - closure rule: if $x \in$ Even then $x + 2 \in$ Even.

2. The set of strings Σ^+ over an alphabets Σ = {a,b,...,z}
  - Initial: if $x \in \Sigma$, then "x" $\in \Sigma^+$.
  - closure: If $x \in \Sigma$ and "α" $\in \Sigma^+$, then "xα" $\in \Sigma^+$.

3. The set (Z^*) of integer lists.
  - Initial: [ ] is a (integer) list,
  - closure: If x is an integer and [L] is a list, then [x L] is a list.
    - e.g., [ ], [4], [3 4], [2 3 4], [5 2 3 4]

- Problem: All definitions well-defined? What's wrong?

Problems about recursive definition

- The above definitions are incomplete in that there are multiple sets satisfy each definition
- Example:
  - Let Ni = {0,2,4,6,...} U { 2i+1, 2i+3, ...}.
  - Then {0,2,4,6,...} and N_i (i > 0) all satisfy Def. 1.
- Among {0,2,4,6,...} and N_i (i > 0) , which one is our intended set ?
- How to overcome the incompleteness ?
- Relationship between {0,2,4,...} and the collection of sets satisfying the definitions?
  - {0,2,4,...} is the least set among all sets.
  - {0,2,4,...} is equal to the intersection of all sets.
  - Every other set contains some elements which are not grounded in the sense that they have no proof (or derivation).

General form of inductively defining a set (or domain)

- Ω : a set, Init: a subset of Ω
  F: a set of functions/rules on Ω,
- we define a subset Δ of Ω as follows:

1. Initialization: Every element of Init is an element of Δ.
   (or simply Init $\subseteq \Delta$)
2. closure: If f:Ω^n->Ω in F and $t_1,\ldots,t_n$ are members of Δ,
   then so is $f(t_1,\ldots,t_n)$
3. plus one of the following 3 phrases.
   3.1 Δ is the least subset of Ω with the above two properties.
   3.2 Δ is the intersection of all subsets of Ω with property 1,2.
   3.3 Only elements of Ω obtainable by a finite number of applications of rules 1 and 2 are elements of Δ.

How to derive an object from an inductive defiintion

- Ω : a set, Init: a subset of Ω
  F: a set of functions on Ω,

Given Init and F, an object $x \in \Omega$ is said to be derivable from Init and F if there is a finite sequence $x_1, x_2, \ldots x_n$ of objects of Ω such that

1. $x = x_n$,
2. for all $1 \le k \le n$, either
   2.1 $x_k \in$ Init --- (x_k is an axiom) or
   2.2 $x_k = f(t_1,\ldots,t_n)$ where $f \in F$ and $\{t_1,\ldots,t_n\} \subseteq \{x_1,\ldots,x_{k-1}\}$.
   --- (x_k is got from closure rule)

The sequence is called a derivation(or deduction,proof) of x.

Examples

- Ω = Z, Init: = {2} , F = { x5, add5, +}
- Then 54 is derivable since

1. 2, --- axiom
2. 7, --- add5(2)
3. 9, --- +(2,7)
4. 45, ---- x5(9)
5. 54 ---- +(9,45)

```
                      =
                      2
         =         =  ----+5
         2         2   7
=    -----+5   --------+5
2        7         9
----------+    ---------x5
  9,               45
--------------------------- +
         54
```
derivation (proof ;deduction) tree for 54

is a derivation of 54. The length of the derivation is 5.

Define functions on recursively defined domains

- Once a domain Δ is defined inductively. We can define functions on the domain according to the following recursive scheme:
- A function $v : \Delta \rightarrow C$, can be defined as follows:
  - basis case: specify the value $v(t)$ of t for each primitive object t in Init.
  - recursive case: specify the value $v(f(t_1,t_2,\ldots,t_n))$ of every compound object $f(t_1,t_2,\ldots,t_n)$ in terms of the values $v(t_1),v(t_2),\ldots,v(t_n)$ of smaller composing objects $t_1,\ldots,t_n$, for all $f \in F$ and $t_1,\ldots,t_n \in \Delta$.

Example

- Consider the following functions defined on integer lists:

  `sum : integer List(denoted Z*)→ Z`

  with `sum(L)` $=_{def}$ sum of all integers in L.

- We can define sum by recursion as follows:
  - Basis case: specify the value `sum(L)` of L for each primitive L in Init = { [] }.
  - ==> basis: `sum([]) = 0.`
  - Recursive case: specify the value `sum(f(t1,...,tn))` of element `f(t1,...,tn)` for each f ∈ F and $t_1,\ldots,t_n \in \Delta = Z^+$.
  - ==> Recursion: where F = { f_x | f_x([L']) = [x L'], x ∈Z }
  - For any list of integers L of the form [x L'],

    `sum(L) = sum([x L’]) = x + sum([L’])`

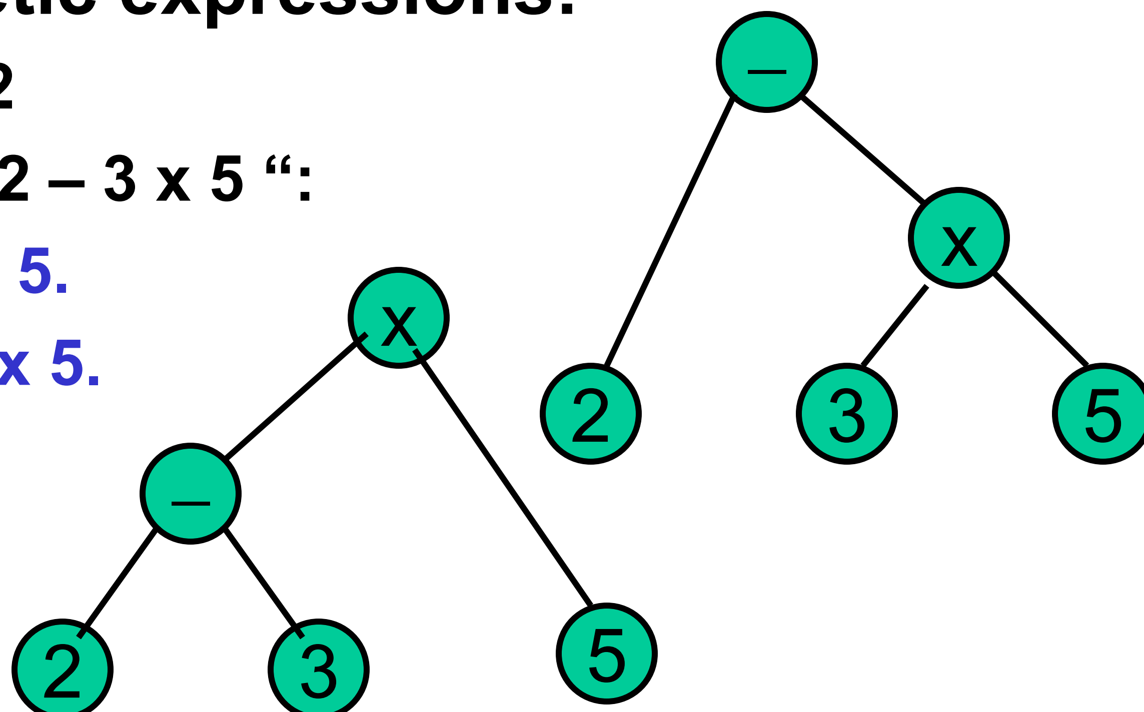
```
Ex: sum([4,3,2])=4+sum([3,2])=4+3+sum([2])=4+3+2+sum([])
  = 4 + 3 +2 + 0 = 9.
```

Define functions on recursively defined domains

- **More example:**
  - **#a : $\Sigma^+ \rightarrow$ N with #a(x) $=_{def}$ number of a's in string x.**
- **Now we can define #a as follows:**
  - **Basis case: specify the value #a("x") of "x" for each x in Σ.**
    ==> #a("a") = 1 ; #a("y") = 0 if y $\neq$ a.
  - **Recursive case: specify the value #a("dz") of element "dz" for each d $\in\Sigma$ and "z"$\in\Sigma^+$.**
  - **==> for any d $\in$ Σ and "z" $\in$ Σ^+ ,**
  - **then #a("dy") = 1 + #a("y") if d = a and**
  - **#a("dy") = #a("y") if d $\neq$ a**
- But are such kind of definitions well defined?
  - **A sufficient condition: If t*he recursively defined domain is***
  - ***not ambiguous (i.e., multi-defined)I.e.,* there is only one way to form (or derive) each element in the domain.**

Example of a multi-defined(ambiguous) Domain

- **Arithmetic Expression (AExp ⊆ Σ*) :**
  - **Init:** Constants a,b,c,… and variables x,y,z,… are arithmetic expressions
  - **Closure**: If α and β are arithmetic expressions then so are α+β, α–β, –α, α x β .
- **ambiguous arithmetic expressions:**
  - 2 – 3 x 5 ;   x – y + 2
  - two ways to form " 2 – 3 x 5 ":
  - 1. 2, 3, 2-3, 5, 2-3 x 5.
  - 2. 2, 3, 5, 3x5, 2- 3x 5.



Problem for ambiguous defintions

- Define the fucntion val : AExp → Z as follows:
- Basis case:
  - val(c) = c where c is an integer constant.
  - val(x) = 0 where x is a variable.
- Recursion : where α and β are expressions
  - val(α+β) = val(α) + val(β)
  - val(α-β) = val(α) – val(β)
  - val(α * β) = val(α) * Val(β)
- Then there are two possible values for 2 – 3 * 5.
  - val(2 - 3 * 5) =val(2) – val(3 * 5) = 2 – [val(3) * val(5)] = 2 – 15 = -13.
  - val(2 – 3 * 5) = val(2-3) * val(5) = [val(2) – val(3)] * 5
  - = -1 * 5 = -5.
- As a result, the function val is not well-defined !!

Structural induction

- Δ: an inductively defined domain

  P(x): a property on Δ.
- To show that P(x) holds for all x in Δ, it suffices to show
  - Basis step: P(x) holds for all x in Init.
  - Ind. step: $\underline{P(t_1),\ldots,P(t_n)}$ => $P(f(t_1,\ldots,t_n))$ for all f in F,

    and for all $t_1,\ldots,t_n$ in Δ.
- Example: show $P(x) \equiv \#a(x) \geq 0$ holds for all x.
  - Basis step: $x \in$ Init = {"a","b","c",...})
    - ✪ x = "a" => #a(x) = 1 $\geq$ 0.
    - ✪ x $\neq$ "a" => #a(x) = 0 $\geq$0
  - Ind. step: x = "dy" where d is any element in Σ and "y" is any element in Σ^+.

    By ind. hyp. #a("y") $\geq$ 0. hence
    - ✪ if d = a => #a("dy") = 1 + #a("y") $\geq$0
    - ✪ if d$\neq$ a => #a("dy") = #a("y") $\geq$ 0.

More example:

- Define the set of labeled binary trees as follows:
- Σ: a set of labels = {a,b,c,..}
- Γ = Σ U {(,)}, Γ* = the set of strings over Γ.
- T_Σ is a subset of Γ* defined inductively as follows:
  - Init: () is a tree. // no more written as "()"
  - closure: if x is a label, and L and R are trees,

    then (x L R) is a tree.
- Example / counterexample:
  - (), (a ()()), ((a) (b) ()) .
- For tree T, let lf(T) = #of '(' ,lb(T) =# of labels, and e(T) = number of empty subtrees "()" in T. All can be defined inductively as follows:
  - basis: lf(()) = 1; lb(()) = 0; e(()) = 1.
  - recursive: lf((x L R)) = 1+ lf(L) + lf(R);
  - lb((x L R)) = 1 + lb(L) +lb(R) ; e((x L R)) = e(L) + e(R).

More example(cont'd)

- Use structural ind. to prove properties of trees.
- Show that for all tree T in T_Σ : $P(T) \equiv_{def}$

  $lf(T) = lb(T) + e(T)$

  holds for all tree T.

   Basis step[T = ()] : lf(()) = 1, lb(())=0, e(())=1 => P(()) holds.

   ind. step[T= (x L R) where x: any label, L, R: any trees] :

  assume (ind.hyp.:) lf(L) = lb(L) + e(L) and

  lf(R) = lb(R) + e(R). Then

  lf((x L R)) = 1 + lf(L) + lf(R) = 1 +lb(L) +lb(R) + e(L) +e(R)

  e((x L R)) = e(L) +e(R)

  lb((x L R)) =1 + lb(L) + lb(R)

  ==> lf((X L R)) = lb((X L R)) + e((X L R)).

Exercise

- Let Z^* be the domain of integer lists as defined inductively at slide 36. Let *append* : $Z^* \times Z^* \to Z^*$ be the binary function on Z^* such that, given x and y, append(x,y) will form a list equal to the concatenation of x and y. For instance, append([1], [2 3]) = [1 2 3] and append([2 3 4],[2 1]) = [2 3 4 2 1].

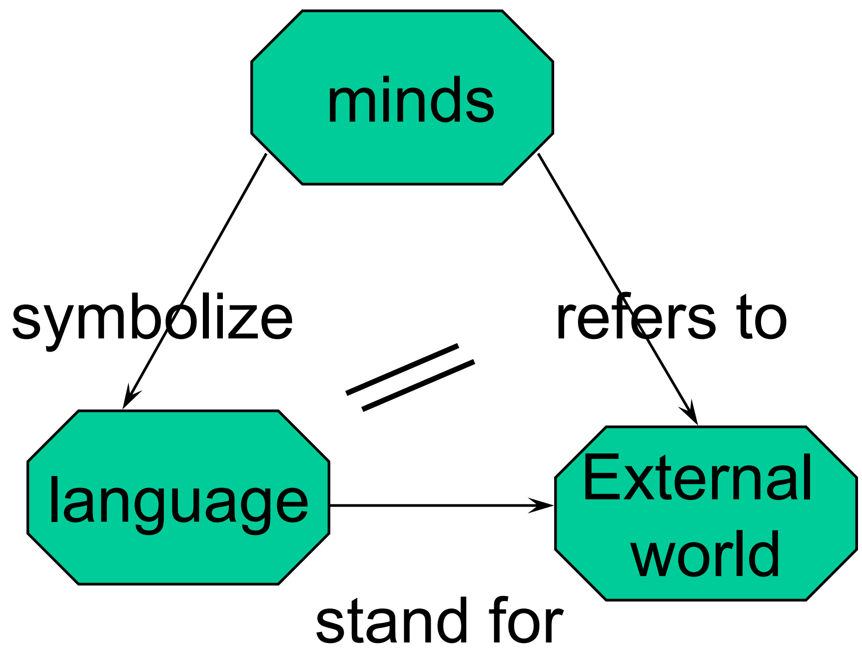
1. Give an inductive definition of append(x,y) according to (the structure of) its first argument x.
2. Prove by induction that for all integer lists x and y,

   sum(x) + sum(y) = sum(append(x,y)),

   where the function sum was defined at slide 42.

2. Basics of formal languages

What is a language ?



The meaning triangle:

Different levels of language analysis

- phonetic and phonological analysis(語音與音韻分析)
  - determine how words are related to sounds that realize them; required for speech understanding.
  - Phonetics concerns itself with the production, transmission, and perception of the *physical* phenomena(phones) which are abstracted in the mind to constitute these speech sounds or signs.
  - Phonology concerns itself with systems of phonemes (音位), abstract *cognitive* units of speech sound or sign which distinguish the words of a language.
  - Ex: k in 'kill' and 'skill' are two phones [k],[g] but same phoneme /k/; book(單數) → books (多數)
- morphological analysis: (詞彙分析;構詞學)
  - determine how words are formed from more basic meaning units called "morphemes". (詞 素)
  - morpheme: primitive unit of meaning in a language.
    - ✪eg: friendly = friend + ly; luckily = lucky + ly

Levels of language analysis

- **syntax analysis: (語法分析)**
  - determine how words can be put together to form correct sentences.
  - determine what structure role each word plays in the sentence.
  - determine what phrases are subparts of what other parts.
  - ex: John saw his friend with a telescope
  - => S[NP[noun:'John'] // one more result not listed!
  - VP[verb: 'saw',
  - NP[NP[possessivePronoun:'his', noun: 'friend']]
  - PP[prep: 'with', NP [art: 'a' noun:'telescope]]]]
- **Semantics analysis : (語意分析)**
  - determine what words mean and how these meanings combine in sentence to form sentence meanings. context independent.
  - Possible analysis result of the previous example:
  - person(j), person(f), name(j,'John'), time(t), friend(f,j) //?
  - see(j, f, t), before(t, now), possess(f, te, t).

Levels of language analysis

- Pragmatic analysis: (語用分析)
  - studies the ways in which context contributes to meaning
  - concern how sentences are used in different situation and how use affects the interpretation of sentences.
  - ex: Would you mind opening the door?
  - John saw his friend with a telescope .
- Discourse analysis (篇章或對話分析) ,...
  - 代名詞解析，文句的銜接與連貫等。
  - Ex: Wang has a friend. John saw him (Wang or Wang's friend?) with a telescope yesterday.
- World knowledge,...
- Languages (including natural and programming languages) contains many facets, each an active research domain of AI, linguistics, psychology, philosophy, cognitive science and mathematics.

What are formal languages

- In the study of formal languages we care about only the well-formedness/membership, but not the meaning of sentences in a language.

Ex1: Our usual decimal language of positive numbers ?

- **Problem: Which of the following are well-formed [representation of] numbers:**
  (1) 128 (2) 0023 (3) 44ac (4) 3327
- **Let L be the set of all well-formed [representations of] numbers. ==> 123, 3327 in L but 0023, 44ac not in L.**
- **So according to the view of FL, The usual decimal language of positive numbers (i.e., L) is just the set :**
- **{ x | x is a finite sequence of digits w/t leading zeros }.**
- **Note: FL don't care about that string '134' corresponds to the (abstract) positive number whose binary representation is 10000000 –It's the job of semantics.**

Definition 2.1

- An alphabet Σ (or vocabulary; 字母集) is a finite set.
  - Ex: decimal_alphabet = {0,1,2,3,4,5,6,7,8,9}
  - binary_digit = {0,1}; Hexidecimal-alphabet = {0,..,9,A,..,F}
  - alphabet-of-English-sentences = {a, word, good, luckily,...}
  - alphabet-of-English-words = {a,...,z,A,...,Z}
- Elements of an alphabet are called *letters* or *symbols*
- *A string (or word or sentence) over Σ is a finite sequence of elements of Σ.*
  - Ex: if Σ = {a,b} then "aabaa" is a string over Σ of length 5.
  - Note: A *string* $x = x_0 x_1 \ldots x_{n-1}$ *of length n is in fact viewed as a function*
  - $x: [0..n) \rightarrow \Sigma$ *such that* $x(k) = x_k$ *for k in* $[0,n)$.
- The length of a string x, denoted |x|, is the number of symbols in x. ex: |abbaa| = 5.
- There is a unique string of length 0, called the null string or empty string, and is denoted by ε (or λ)

Definition 2.1 (cont'd)

- $\Sigma^* =_{def}$ the set of all strings over Σ.
  - Ex: $\{a,b\}^* = \{\varepsilon,a,b,aa,ab,ba,bb,aaa,...\}$
  - $\{a\}^* = \{\varepsilon,a,aa,aaa,aaaa,...\} = \{a^n \mid n \geq 0\}$.
  - $\{\}^*$ = ? ($\{\}$ or $\{\varepsilon\}$ or ε ?)
- Note the difference b/t sets and strings:
  - $\{a,b\} = \{b,a\}$ but $ab \neq ba$.
  - $\{a,a,b\} = \{a,b\}$ but $aab \neq ab$
- So what's a (formal) language ?
- *A **language** over Σ is a set of strings over Σ (i.e., a subset of Σ^*).* Ex: let $\Sigma = \{0,...,9\}$ then all the followings are languages over Σ.
  - 1. $\{\varepsilon\}$ 2. $\{\}$ 3. $\{0,...,9\} = \Sigma$ 4. $\{x \mid x \in \Sigma^*$ and has no leading 0s$\}$ 5. $\Sigma^5 = \{x \mid |x| = 5\}$ 6. $\Sigma^* = \{x \mid |x|$ is finite $\}$

Examples of practical formal languages

Ex: Let Δ be the set of all ASCII codes.

- a C program is simply a finite string over Δ satisfying all syntax rules of C.
- C-language $=_{def}$ { x | x is a well-formed C program over Δ }.
- PASCAL-language = {x | x is a well-formed PASCAL program over Δ }.

Similarly, let ENG-DIC = The set of all English lexicons

= { John, Mary, is, are, a, an, good, bad, boys, girl,..}

- an English sentence is simply a string over ENG-DIC
- ==> English $=_{def}$ {x | x is a legal English sentence over END-DIC} ==>
- 1.John is a good boy . $\in$ English.
- 2. |John is a good boy . | = ?

issues about formal languages

- **Why need formal languages?**
  - for specification (specifying programs, meanings etc.)
  - i.e., basic tools for communications b/t people and machines.
  - although FL does not provide all needed theoretical framework for subsequent (semantic processing...) processing, it indeed provides a necessary start, w/t which subsequent processing would be impossible -- first level of abstraction.
  - Many basic problems [about computation] can be investigated at this level.
- **How to specify(or represent) a language ?**
  - Notes: All useful natural or programming languages contain infinite number of strings (or programs and sentences)

How to specify a language

- principles: 1. must be precise and no ambiguity among users of the language: 2. efficient for machine processing
- tools:
- 1. traditional mathematical notations:
  - ✪ A = {x | |x| < 3 and x $\in$ {a,b}} = {e,a,b,aa,ab,ba,bb}
  - ✪ problem: in general not machine understandable.
- 2. via programs (or machines) :
- P: a program; L(P) $=_{def}$ {x | P return 'ok' on input string x}
  - ✪ precise, no ambiguity, machine understandable.
  - ✪ hard to understand for human users !!
- 3. via grammars: (easy for human to understand)
  - ✪Ex: noun := book | boy | jirl | John | Mary
  - ✪ art := a | an | the ; prep := on | under | of | ...
  - ✪ adj := good | bad | smart | ...
  - ✪ NP := noun | art noun | NP PP | ...
  - ✪ PP := prep NP ==> 'the man on the bridge' $\in$ PP.

Non-enumerability of languages

- Recall that a set is denumerable if it is countably infinite. (i.e., A set T is *denumerable* if there is a 1-1 and onto mapping b/t T and {0,1,...})
- Exercises: If Σ is finite and nonempty, then
  - 1. Σ^* is denumerable (i.e., $|\Sigma^*| = |N|$)
  - 2. 2^{Σ^*} (ie., the set of all languages over Σ) is uncountable.
  - pf: Since $|2^{\Sigma^*}| \neq |\Sigma^*| = |N|$, hence $|2^{\Sigma^*}|$ is not countable

Operations on strings

- **string concatenations:**
  - x,y: two strings ==> x·y is a new string with y appended to the tail of x. i.e., x·y is the function :
  - z : [0, len(x)+len(y)) → Σ such that
  - z(n) = x(n) for $0 \le n <$ len(x) and
  - z(len(x)+n) = y(n) for $0 \le n <$ len(y).
  - Some properties of · :

    1. ASSOC: (xy)z = x(yz) ; 2. Identity: εx = xε = x.

    3. |xy| = |x| + |y|.
- **conventions and abbreviations:**
  - Σ: for alphabet ; a,b,c: for symbols;
  - x,y,z: for strings; A,B,C: for languages;
  - x^5 for xxxxx; $x^1 = x$; $x^0 = \varepsilon$.
  - #a(x) $=_{def}$ number of a's in x. ==> #a(aabbcca) = 3.

Operations on languages (i.e, string sets)

1. usual set operations:

- Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
  - Ex: {a,ab} U { ab, aab} = {a,ab,aab}
- intersection: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- complements in Σ^*: $\sim A =_{def} \Sigma^* - A = \{x \mid x \text{ not } \in A\}$
- ex: $\sim\{x \mid |x| \text{ is even}\} = \{x \mid |x| \text{ is odd}\}$.

2. Set concatenations:

$A \cdot B =_{def} \{xy \mid x \in A \text{ and } y \in B\}$.

- Ex: {b,ba} {a,ab} = {ba,bab,baa,baab}.

3. Powers of A: A^n ($n \geq 0$) is defined inductively:

1. $A^0 = \{\varepsilon\}$; $A^{n+1} = A \cdot A^n = A \cdot A \cdot \ldots \cdot A$. ----- n A's

Operations on languages (cont'd)

Ex: Let $A = \{ab, abb\}$. Then

- 1. $A^0 = ?$ 2. $A^1 = ?$ 3. $A^2 = ?$ 4. $|A^4| = ?$
- 5. Hence $\{a,b,c\}^n = \{x \in \{a,b,c\}^* \mid |x| = n\}$ and
- $A^n = \{ x_1 x_2 \ldots x_n \mid x_1, \ldots, x_n \in A \}$

5. Asterate (or star) A^* of A is the union of all finite powers of A:

$$A^* =_{def} \bigcup_{k \geq 0} A^K = A^0 \cup A \cup A^2 \cup A^3 \cup \ldots$$

$$= \{x_1 x_2 \ldots x_n \mid n \geq 0 \text{ and } x_i \in A \text{ for } 1 \geq i \geq n\}$$

notes:

1. n can be 0 ==> $\varepsilon \in A^*$. ==> $\varepsilon \in \{\}^*$.
2. If $A = \Sigma$ ==> $A^* = \Sigma^*$ = the set of all finite strings over Σ.

Properties of languages operations

6. $A^+ =_{def}$ the set of all nonzero powers of A

$=_{def} \bigcup_{k\geq1} A^k = A \cup A^2 \cup A^3 \cup \ldots = A\,A^*$.

Properties of languages operations

1. associative: $\cup$, $\cap$, $\cdot$:

   $A\cup(B\cup C) = (A\cup B)\cup C$; $A\cap(B\cap C) = (A\cap B)\cap C$;

   $A(BC) = (AB)C$

2. commutative : $\cup$,$\cap$:
3. Identities:
   - 1. $A \cup \{\} = \{\}\cup A = A$; 2. $A\cap \Sigma^* = \Sigma^*\cap A = A$;
   - 3. $\{\varepsilon\}A = A\{\varepsilon\} = A$.
4. Annihilator: $A\{\} = \{\}A = \{\}$.

Properties of languages operations (cont'd)

5. Distribution laws:

- $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$; $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- $A(B \cup C) = AB \cup AC$; $\underline{A(B \cap C) = AB \cap AC \ \ (x)}$
- Ex: Let A = {a,ab}, B = {b}, C = {ε}
- ==> $A(B \cap C)$ = ? AB = ? AC = ?
- ==> $A(B \cap C)$ ____ $AB \cap AC$.
- Exercise: show that $A(B \cup C) = AB \cup AC$.

6. De Morgan Laws: $\sim(A \cup B)$ = ? $\sim(A \cap B)$ = ?

7. Properties about A*:

- 1. $A^*A^* = A^*$; 2. $A^{**} = A^*$; 3. $A^* = \{\varepsilon\} \cup AA^*$
- 4. $AA^* = A^*A = A^+$. 5. $\{\}^* = \{\varepsilon\}$.
- Exercises: Prove 1~5. (hint: direct from the definition of A*)

A language for specifying languages

- In the term: 'a language for specifying languages', the former language is called a *metalanguage* while the later languages are called *target languages*.
  - So in the C language reference manual, the BNF form is a meta language and C itself is the target language.
- Σ: an alphabet ; $\Delta = \Sigma \cup \{ +, *, e, \emptyset, \cdot,), (\}$;
- $E = \Delta \cup \{\sim, \cap, -\}$
- 1. The set of all regular expressions is a subset of Δ^* which can be defined inductively as follows:
  - Basis: 1. e, $\emptyset$ are regular expressions
  - 2. Every symbol a in Σ is a regular expression.
  - Induction: If α and β are regular expressions then so are
  - $(\alpha+\beta)$, $(\alpha\cdot\beta)$, α^*.

Regular expressions

- Examples:
- legal reg. expr. : e, (a+b)*, ((a +(b·c))+(e·b)*)
- illegal reg. expr: (ab), a + b, ((a + Σ)) + d, where d $\notin$ Σ.
- illegal formally but legal if treated as abbreviations:
- ab --> (a·b) ; a+b --> (a+b);
- a + bc* --> (a + (b·c*))

- **Extended regular expressions (EREGs):**
- **EREGs are strings over E and can be defined inductively as follows:**
  - Basis: 1. e, ∅ are EREGs
  - 2. Every symbol a in Σ is an EREG.
  - Induction: If α and β are EREGs then so are
  - (α+β), (α·β), α*, (~α), (α∩β), (α-β)

Languages represented by regular expressions

- [Extended] regular expressions provides a finite way to specify infinite languages.
- Definition: for each EREG (and hence also REG) α, the language (over Σ) specified (or denoted or represented) by α, written $L(\alpha)$, is defined inductively as follows:
  - Basis: $L(e) = \{\varepsilon\}$; $L(\emptyset) = \{\}$;
  - for each $a \in \Sigma$, $L(a) = \{a\}$.
  - Induction: assume $L(\alpha)$ and $L(\beta)$ have been defined for EREG α and β. Then
  - $L(\alpha+\beta) = L(\alpha) \cup L(\beta)$; $L(\alpha\beta) = L(\alpha) L(\beta)$; $L(\alpha^*) = L(\alpha)^*$;
  - $L(\sim\alpha) = \Sigma^* - L(\alpha)$; $L(\alpha - \beta) = L(\alpha) - L(\beta)$; $L(\alpha \cap \beta) = L(\alpha) \cap L(\beta)$.
- Definition: a language A is said to be ***regular*** if $A = L(\alpha)$ for some regular expression α.

Examples:

- Let $\Sigma = \{a,b\}$. Then
  - $L(a(a+b)^*) = \{x \mid x$ begins with a $\} = \{a,aa,ab,aaa,aab,aba,...\}$
  - $L(\sim(a(a+b)^*)) = \{x \mid x$ does not begin with a$\}$
  - $= \{x \mid x$ begins with b $\} \cup \{\varepsilon\} = L(e + b(a+b)^*)$.
- Regular expressions and Extended regular expressions give us finite ways to specify infinite languages. But the following questions need to be answered before we can be satisfied with such tools.
  - 1. Are EREG or REGs already adequate ?
  - (i.e, For every $A \subseteq \Sigma^*$, there is an expression α s.t., $L(\alpha) = A$?) ==> ans: _____.
  - 2. For every expression α, is there any [fast] machine that can determine if $x \in L(\alpha)$ for any input string x ?
  - Ans: ________

IS EREG more expressive than REG ?

- **L1, L2: two [meta] languages;**
  - we say L1 is at least as expressive as L2 if L(L2) $=_{def}$ {A | there is an expression a in L2 s.t. A = L(a) } is a subset of L(L1).
  - L1 is said to be equivalent to L2 in expressive power iff both are at least as expressive as the other.
- **Problem:**
  - EREG is at least as expressive as REG since L(REG) is a subset of L(EREG) (why?)
  - But does the converse hold ? (i.e, Is it true that for each EREG α there is a REG β s.t., L(α) = L(β) ?
  - ans: _____.