CS 316 3/17/2014.

Another way to making an argument smaller for recursion:
(cdr e1) where e1 is a suitably transformed version of e.

Ex ssort on Asn. 5
(ssort L) ⇒ a list obtained by sorting the element of L into ascending order.
L: List of real numbers e
(ssort '(3 7 1 0 8 9 5)) ⇒ (0 1 3 5 7 8 9).

In this case, let e1 = (Min-first e).
= e with the first occurence of the ~~least~~ least element moved to the ~~first~~.

e1 = (0 3 7 1 8 9 5)
(cdr e1)

Sometimes you should use different recursive strategies for different argument values.

Ex Merge-lists. on Asn 5 (Asn = Assigned ϕ, Assignment).
(merge-lists L1 L2) ⇒ two lists of real members in ascending order
a List of members in ascending order obtained by merging L1 and L2

(merge-lists '(2 7 9 12) '(0 1 8 10 15)) ⇒ (0 1 2 7 8 9 10 12 15)
L1 L2

strategy 1: compute (merge-lists L1 L2) from (merge-lists (cdr L1) L2).
strategy 2: compute (merge-lists L1 L2) from (merge-lists L1 (cdr L2)).
strategy 1 is appropiate of (car L1) < (car L2)
strategy 2 is appropiate of (car L1) > (cad L2)
Either strategy works of (car L1) = (car L2)

Another Example unrepeated-elts (from Asn. 5)
(unrepeated-elts '(C A D D A B B B A C)) ⇒ (C A D A B C)
~~strategy~~ strategy 1: compute (~~unrepeated~~ unrepeated L) from (unrepeated elts (cdr L))
strategy 2: — — — (unrepeated (cddr L))

L = (CA DD . .) ~~target~~ strategy 1

L = 0(AADD . . .) strategy 2

L = (A A A — —) storge 3

Programming paradigm.

Already discussed:

Imperative programming

object-oriented imperative programming

Functional programming

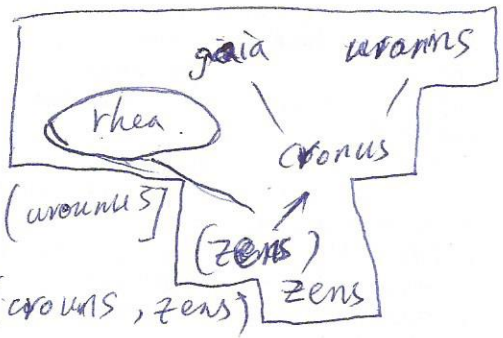
Aother Large. established paradigm in logic programming

In logic programming we write collecting of facts and inference rules.

A logic pragrm is used by writing queries which the logic programming system will find answers to by deduction from the facts and inference rules.

Prolog (developed in the early 1970s by colmerauser) is the best known logic programming Language.

Simple Example:



parent (uranus, cronus)

parent (cronus, zens)

Facts

parent (gaia, crouns)

parent (rhea, zeus)

inference rule.

grand parent (person1, person2)

:- parent (X, person2)

parent (person1, X).

the above is a very simple logic program (consisting of 4 facts clauses and 1 rule clause).

Question that use their program: ?- parent (gaia, X)

X = cronus; ⟶ "try to find another answ

false = no more answers.

?- parent(X, zeus)

X = rhea;

X = cronus;

false;

?- grandparent(Y, zeus);

Y = gaia;

Y = uranus;

false;

Syntax of programming Languages.

You will receive exercises soon — check email regularly (by Friday).

Syntax = form without regard to meaning

Semantics = meaning

syntax rules have the property that a compiler can check whether or not they are violated — it is never necessary to execute the program to determine whether a syntax rule is violated.

Analogy with English

The dog barked loudly.

|| A valid sentence. == syntactically valid has correct grammtic form & semantically valid ("meaningful")

The night barked loudly.

|| syntactically valid* — Has the same form/syntax as the above valid sentence

|| semantically invalid ("meaningless" — because. nights cannot bark).

* if can be obtained from that sentence by replacing one name with another.

The dog loudly. → syntically invalid — don't have correct grammatical form as it has no v

For a program in a language such as C++ or Java, a ~~syntical~~ syntactically valid program (roughly speaking), a piece of text that can be obtained from some valid program by making changes of the following kinds (zero or more times).

1. Remove a declaration (of a variable, function etc.)
2. Replace one identifier with another
3. Replace a constant in literal (e.g. "dog" or 37) with another literal of the same kind.

Ex: A syntactically valid C++ programm that is not a valid C++ progr

```
int g()
{
    int x=7;
    return x[3]/((x-7)/(3-3));
}
```

This is syntactically valid because it can be obtained from the following valid program by making changes of the above kinds:

Exercise