AI SMPS 2023: Lists and Tuples

Version 0.5 (prepared by S. Baskaran)

A quick reference for list and tuple operators used in the algorithms.

In the assignments and final exam, answers to short-answer-type questions depend on the sequence in which values are added, read and removed from lists and tuples. Therefore, it is important to understand the representation and operations on lists and tuples.

OPERATORS AND EXPRESSIONS

▷	▷ a right pointing triangle starts a line comment
_	▷ an underscore, a don't care value, a wild card
=	▷ equality-test operator
←	▷ assignment operator
:	▷ list constructor, a.k.a, **cons** operator
++	▷ list concatenation operator
null	▷ null value
head LIST	▷ returns the head of a list
tail LIST	▷ returns the tail of a list
take n LIST	▷ returns at most n elements from a list
first TUPLE	▷ returns the first element of a tuple
second TUPLE	▷ returns the second element of a tuple
third TUPLE	▷ returns the third element of a tuple

EXPRESSION$_1$ = EXPRESSION$_2$	▷ equality test
EXPRESSION$_1$ **is null**	▷ **is** test
EXPRESSION$_1$ **is not empty**	▷ **is** test
PATTERN ← EXPRESSION	▷ assignment

In what follows, all equality tests (expr$_1$ = expr$_2$) evaluate to true.

LIST OPERATIONS

LIST$_2$ ← ELEMENT : LIST$_1$ ▷ list representation
LIST ← HEAD : TAIL ▷ components of a list

[] ▷ an empty list
3 : 2 : 1 : [] ▷ a three element list
[3, 2, 1] ▷ a list in shorthand notation

[3, 2, 1] = 3 : [2, 1] = 3 : 2 : [1] = 3 : 2 : 1 : []

[] **is empty** = TRUE
[1] **is empty** = FALSE

[1] = 1 : []
1 = **head** [1] = **head** 1 : []
[] = **tail** [1] = **tail** 1 : []

(**tail** [1]) **is empty** = TRUE

3 = **head** [3, 2, 1] = **head** 3 : 2 : 1 : []
[2, 1] = **tail** [3, 2, 1] = **tail** 3 : 2 : 1 : []
2 = **head tail** [3, 2, 1] = **head tail** 3 : 2 : 1 : []
[1] = **tail tail** [3, 2, 1] = **tail tail** 3 : 2 : 1 : []
1 = **head tail tail** [3, 2, 1] = **head tail tail** 3 : 2 : 1 : []

head tail tail 3 : 2 : 1 : [] ▷ **head**, **tail** are right associative
= **head** (**tail** (**tail** (3 : 2 : 1 : [])))
= **head** (**tail** (2 : 1 : []))
= **head** (1 : [])
= 1

[o, u, t] = **take** 3 [o, u, t, r, u, n]
[a, t] = **take** 3 [a, t]
[a] = **take** 3 [a]
[] = **take** 3 []

LIST$_3$ = LIST$_1$ ++ LIST$_2$
[] = [] ++ []
LIST = LIST ++ [] = [] ++ LIST
[o, u, t, r, u, n] = [o, u, t] ++ [r, u, n]
[r, u, n, o, u, t] = [r, u, n] ++ [o, u, t]

[r, o, u, t] = (**head** [r, u, n]) : [o, u, t]
[n, u, t] = **tail tail** [r, u, n] ++ **tail** [o, u, t]
[n, u, t] = (**tail tail** [r, u, n]) ++ (**tail** [o, u, t])

a ← **head** [3, 2, 1] ▷ a ← 3;
b ← **tail** [3, 2, 1] ▷ b ← [2, 1];

a : b ← [3, 2, 1] ▷ a ← 3; b ← [2, 1];
a : b ← 3 : 2 : 1 : [] ▷ a ← 3; b ← [2, 1];

a : b : c ← [3, 2, 1] ▷ a ← 3; b ← 2; c ← [1];
a : b : c ← 3 : 2 : 1 : [] ▷ a ← 3; b ← 2; c ← [1];

a : _ : c ← [3, 2, 1] ▷ a ← 3; c ← [1];
a : _ : c ← 3 : 2 : 1 : [] ▷ a ← 3; c ← [1];

TUPLE OPERATIONS

(101, "Oumuamua", 400m) ▷ a 3-tuple
(101, 102) ▷ a 2-tuple

101 = **first** (101, 102)
102 = **second** (101, 102)

pair ← (101, 102)
101 = **first** pair = **first** (101, 102)
102 = **second** pair = **second** (101, 102)

a ← **first** pair ▷ a ← 101;
b ← **second** pair ▷ b ← 102;
(a, b) ← pair ▷ a ← 101; b ← 102;
(a, b) ← (101, 102) ▷ a ← 101; b ← 102;

a ← **first** pair ▷ a ← 101;
(a, _) ← pair ▷ a ← 101;

b ← **second** pair ▷ b ← 102;
(_, b) ← pair ▷ b ← 102;

400m = **third** (101, "Oumuamua", 400m)
c ← **third** (101, "Oumuamua", 400m) ▷ c ← 400m;
(_, _, c) ← (101, "Oumuamua", 400m) ▷ c ← 400m;

101 = **head second** (1, [101, 102, 103], **null**)
[102, 103] = **tail second** (1, [101, 102, 103], **null**)

(a, h : t, c) ← (1, [101, 102, 103], **null**)
▷ a ← 1;
▷ h ← 101;
▷ t ← [102, 103];
▷ c ← **null**;

Done. You are ready, go, finish your work.