CS2104 Programming Language Concepts

Tutorial 03, Lecture 2 and 3

Please, first finish with your teaching assistent the previous tutorial.

1 Record Arity and Record Construction

A record is described by label, features, and fields. For a tuple, the set of its features is uniquely identified by its width. Please explain why!
For a general record this is different. Therefore, Oz provides an operation that returns the *arity* of a record: the arity is defined as its features. The arity is returned as a sorted list, where integers go first in increasing order, followed by atoms in increasing order.
For example, `{Arity r(b:1 a:2 3:3)}` returns `[3 a b]`.
Can you think of a good reason why the arity is sorted?
To construct a record without giving fields, you can use `MakeRecord` which takes a label and a list of features (which does not need to be sorted). For example, `{MakeRecord r [b a 3]}` constructs a record with label `r` and features `b`, `a`, and `3`.
Try some examples!

2 Pairs and Pairlists

The system provides syntactic sugar for pairs similar to syntactic sugar for cons. The statement `X = a # b` abbreviates `X = ´#´(a b)`, which constructs a tuple with label `´#´` (why quotes?) and fields `a` and `b`.
Pairs are often used as elements of lists. These lists are called pairlists. For example, the following list is a pairlist: `[a#1 b#2 c#3]`.
There is a catch: both pairs and cons are constructed with infix operators (`#` and `|`). Try to find out which one binds tighter. The concept of tighter binding is known to you from school with `+` and `*`, where `*` binds tighter than `+`. How to find out?

3 Zip and UnZip

Two important functions that convert pairlists to pairs of lists and vice versa are `Zip` and `UnZip`.
Implement a function `Zip` that takes a pair `Xs#Ys` of two lists `Xs` and `Ys` and returns a pairlist, where the first field of each pair is taken from `Xs` and the second from `Ys`. For example,

```
{Zip [a b c]#[1 2 3]}
```

returns the pairlist [a#1 b#2 c#3]. Give an implementation of Zip where you can assume that both lists have the same length.
The function UnZip does the inverse, for example

```
{UnZip [a#1 b#2 c#3]}
```

returns [a b c]#[1 2 3]. Give a specification and implementation of UnZip.

4 Arithmetic Expressions: Grammar

Develop a grammar for simple arithmetic expressions that involve only: natural numbers, addition, and multiplication. What are the tokens of this language? Does your grammar capture that multiplication binds tighter than addition?

5 Arithmetic Expressions: Evaluation

Suppose that you are given an arithmetic expression described by a tree constructed from tuples as follows:

- An integer is described by a tuple int(N) where N is an integer.
- An addition is described by a tuple add(X Y) where both X and Y are arithmetic expressions.
- A multiplication is described by a tuple mul(X Y) where both X and Y are arithmetic expressions.

Implement a function Eval that takes an arithmetic expression and returns its value.
For example, add(int(1) mul(int(3) int(4))) is an arithmetic expression and its evaluation returns 13.

6 Evaluation with Variables

This assignment is voluntary and left for you to be done at home.
Suppose that now also variables are allowed in arithmetic expressions:

- A variable is described by a tuple var(A) where A is an atom giving the *variable name*.

An *environment* is a record that has label env and has for each variable name a feature that has an integer value.
How can you evaluate these expressions with respect to an environment? Give a specification and an implementation.
Which property must an environment fulfill such that evaluation actually works?