McMaster University
Department of Computing and Software
Musa Al-hassy

# COMP SCI 3EA3 — Software Specification and Correctness

January 20, 2017

## Exercise 3.1 — Learning about C

Implement the following BNF types in C; the first is done for you.

1. Integer Trees,

$$Tree_{\mathbb{Z}} \ ::= \ \mathbb{Z} \mid Tree_{\mathbb{Z}} \triangledown Tree_{\mathbb{Z}}$$

**Sample Solution:** The union of possible options is represented in C using the `union` keyword, but we need to somehow indicate what data lives in what section so we introduce tags to keep track of this. —compare with disjoint sums versus unions in maths.

```
struct int_tree
{
  enum { isLeaf , isFork } tag;
  union
  {
    int Leaf; // if tag == isLeaf we use only this value

    // if tag == isFork we use this datatype
    struct { struct int_tree *left , *right; } Fork;
  } Contents;
}; // int_tree

typedef struct int_tree IntTree;
```

More on trees in C can be found at
`https://www.cs.bham.ac.uk/~hxt/2013/c-programming-language/trees-in-cplusplus.pdf`
For example, $0\triangledown(1\triangledown 2)$ –the tree with left child 0, and right child being a tree itself with left child 1 and right child 2— can be represented by `Fork( MkLeaf(0) , Fork( MkLeaf(1), MkLeaf(2) ) )` where these *constructors* are defined by

```
IntTree* MkLeaf(int n)
{
  IntTree* result = malloc(sizeof(IntTree));
  *result = (IntTree) { isLeaf , { .Leaf = n} };
  return result;
}

IntTree* Fork(IntTree* l, IntTree* r)
{
  IntTree* result = malloc(sizeof(IntTree));
  *result = (IntTree){ isFork , { .Fork.left = l , .Fork.right = r} };
  return result;
}
```

( Simply put, if you use non-pointer parameters then things wont work. )

Exercise: write a method to show a tree.

2. Integer Stacks $\quad Stack_{\mathbb{Z}} \ ::= \ \epsilon \mid \mathbb{Z} \triangleright Stack_{\mathbb{Z}}$
3. Extended Integers $\quad Maybe_{\mathbb{Z}} \ ::= \ \mathbb{Z} \mid \bot$
4. Integer-Natural Pairs $\quad Pair_{\mathbb{Z},\mathbb{N}} \ ::= \ \mathbb{Z} \sim \mathbb{N}$
5. Integer-Natural Sum $\quad Sum_{\mathbb{Z},\mathbb{N}} \ ::= \ \mathbb{Z} \mid \mathbb{N}$
6. Booleans $\quad \mathbb{B} \ ::= \ true \mid false$
7. Naturals Below $n : \mathbb{N}$ $\quad Fin_n \ ::= \ 0 \mid 1 \mid 2 \mid \cdots \mid n-1 \qquad$ Try $n = 0, 1, 2$ ;)

Above $\epsilon, \bot, true, false, 0, 1, \ldots, n-1$ are zero-ary function symbols! While $\sim$ and $\triangleright$ are binary function symbols.

**Exercise 3.1 — Algebra**

In ACSL, formalise the inductive principles for the *expressions/terms* of the types in the previous question. The first one is done for you.

**Sample Solution:**

> (Aside:
>
> A *Magma* is a pair $(S, \oplus)$ consisting of a type $S$ and a binary operation over that type, $\oplus : S \times S \to S$.
>
> An expression over this type is either an element of the carrier $S$ or an application of the binary operation to existing expressions, that is
>
> $$\mathit{MagmaExpr}_S \ ::= \ S \mid \mathit{MagmaExpr} \oplus \mathit{MagmaExpr}$$
>
> Notice that this is just $\mathit{Tree}_{\mathbb{Z}}$, upto renaming!
>
> That is, the type of *expressions over a magma* is nothing more than *Binary Tree.*
>
> )

The induction principle is: for any predicate $P : \mathit{Tree}_{\mathbb{Z}} \to \mathbb{B}$,

$$(\forall s : \mathbb{Z} \bullet P\ s) \wedge (\forall l, r : \mathit{Tree}_{\mathbb{Z}} \bullet P\ l \wedge P\ r \Rightarrow P(l \nabla r)) \implies (\forall t : \mathit{Tree}_{\mathbb{Z}} \bullet P\ t)$$

Roughly put, an element can only belong to one datatype and so we need to declare a way to *embed* our carrier's elements as elements of the associated expressions, this is done via the operation `MkLeaf`.

```
/*@ axiomatic TreeProperties
{
  logic IntTree MkLeaf(integer n);
  logic IntTree Fork(IntTree l, IntTree r);
  logic boolean p(IntTree x);

  predicate LeavesHaveP    = \forall integer x     ; p( MkLeaf (x) )              ;
  predicate ForksHaveP     = \forall IntTree l , r ; p(l) && p(r) ==> p(Fork(l,r)) ;
  predicate AllTreesHaveP  = \forall IntTree t     ; p(t)                          ;

  lemma TreeInd: LeavesHaveP && ForksHaveP ==> AllTreesHaveP;
}
*/
```

Exercise: can the final lemma be proven by Frama-C? Why or Why not?

**Exercise 3.3 — Specify a C function and prove its implementation correct**

1. A "celebrity" $x$ in a room of people named $0..n-1$ is a person who is in the room *such that* everyone else knows them and they do not know anyone else.

   Formalize this property in ACSL.
2. Can there ever be multiple celebrities in a room?
3. Specify the function

   ```
   int findCelebrity( int** knows, int n)
   ```

   that *assuming* there is celebrity, finds it using `knows[0..n-1 , 0..n-1]` relationship:

   `knows[i][j]` $\equiv$ person $i$ knows person $j$
4. Add loop specifications to aid verification of your implementation.
5. Write a `main` function as "driver" program to be able to run some tests, and to use value analysis.

**Exercise 3.4** — **Puzzle**

What is wrong with the following incorrect program? Add appropriate ACSL specfications to find the error.

```
// this program calcuates x to the power of n
int arsac(int x, unsigned int n)
{
  int a, i, t
 ; a = x
 ; i = 0
 ; OhMy:
    ; a = a * x
    ; i = i + 1
    ; t = n - 1
 ; if (i < t) goto OhMy
 ; return a
 ;
}
```

**Exercise 3.5** — CalcCheck

Log onto Avenue and try proving the theorems and solving the puzzles and problems taken from last term's comp sci 2dm3 class with Professor Wolfram Kahl, whose tool CalcCheck we will use to obtain immediate feedback on our proofs.

"Avenue → Contents → Exercises (CS2DM3 Prof Kahl)"

1. Familiarise yourself with the CalcCheck tool by reading the contents titled "CalcCheck". There will be "grand tutorial" on the tool on **Monday January 23 by Curtis D'Alves** in-place of lecture.
2. **Read**: chapters 5 —*Calculational Logic: Part 1*— of the course text.
3. Syntax: **formalise** the puzzles of "2DM3 Exercise 3.2".
4. Operators: add proofs for all theorems in "2DM3 Exercise 5.1".

Note that this exercises section is essentially tantamount to simply

"prove all theorems encountered in chapter 5 of the course text"

however, using the tool increases **confidence in one's own proofs!**