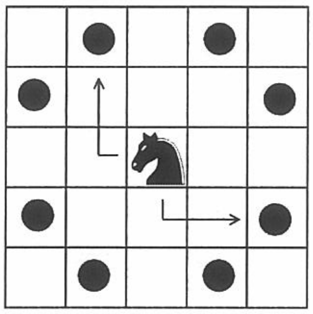
Program Verification (192114300)

Exam 24 June 2014, 13:45–17:15

- This exam contains 5 exercises, for which at most 90 points are awarded. The final mark equals the number of points obtained divided by 10, plus one.
- During the exam, the use of all materials is allowed.

Exercise 1 (*25 points*)

The *knight* in chess moves by shifting one field in one direction, then two fields in a perpendicular direction.



Design a graph representation of a chess board that allows an easy representation of knight moves.

a. *(5 points)* Give the type graph of your representation. Full points will only be awarded for a solution in that takes advantage of the symmetry of the board.

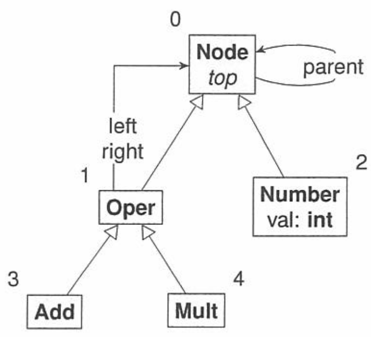
b. *(4 points)* Give a graph representation of a 2×2-board. (Obviously there are no valid moves on such a board.)

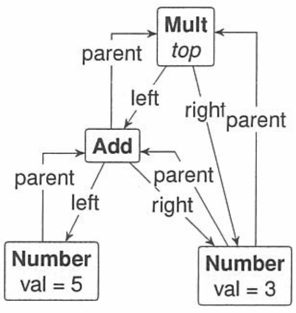
c. *(6 points)* Give one or more rules to represent a move. Full points will only be awarded for a solution in which all moves are applications of a *single* rule.

d. *(5 points)* Draw the entire state space resulting from a 3×3-board, in which the knight starts in a corner field. You do not have to draw the precise graphs of the individual states, but do make clear (by annotating your arrows) which transition corresponds to what (type of) move.

Exercise 2 (*20 points*)

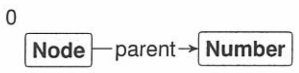
Consider the following type graph, which can be used to represent syntax trees of arithmetic expressions. An exemple graph is also given; note that subtrees may be shared.



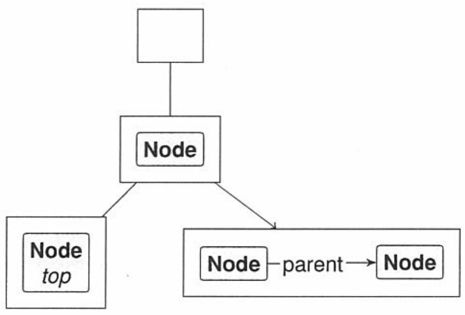


The numbers in the type graph are node identities that will be used in the exercise below.

a. *(5 points)* Is the following graph well-typed? If so, give a typing morphism that maps the graph nodes to the type nodes, and show that the morphism is correct; if not, argue that there does not exist such a morphism.

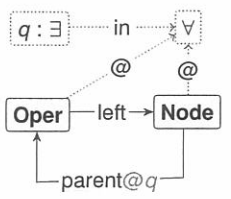


b. *(5 points)* What property does the following multi-graph nested structure encode? Formulate the property in natural language and in first order logic. *Hint: the branching from the second to the third level represents disjunction; the bottom right morphism maps the node in the source graph to the right hand node in the target graph.*



c. *(5 points)* What property does the structure above represent if you leave off its top level? Formulate your answer in natural language and in first-order logic. How does this relate to the answer of the previous question?

d. *(5 points)* What property does the following single-graph nested structure encode? Formulate the property in natural language, as a multi-graph nested structure and in first order logic.



Exercise 3: Hoare Logic (*15 points*)

a. *(3 points)* Consider the statement $\texttt{pick}(S_1, S_2)$ that randomly chooses to execute either S_1 or S_2 (but not both). Its semantics is described by the following operational semantics rule:

$$\frac{(S_1, s) \to s' \vee (S_2, s) \to s'}{(\texttt{pick}(S_1, S_2), s) \to s'}$$

Give a Hoare logic rule for this statement.

b. *(3 points)* Consider Dijkstra's guarded command statement $b_1 \to S_1 \,\Box\, b_2 \to S_2$, where b_1 and b_2 are conditional expressions, and S_1 and S_2 are statements (NB We use the binary guarded command, not the N-ary version). This statement randomly executes *one* of the branches for which the conditional expression evaluates to true. If none of the conditional expressions evaluates to true, it behaves as a Skip statement.

Give an operational semantics rule to describe the behaviour of this statement.

c. *(3 points)* Give an appropriate Hoare logic rule for this statement.

d. *(3 points)* Give an argument based on the semantics of the guarded command statement why your rule is correct.

e. *(3 points)* Derive that your Hoare logic rule for the guarded command statement is correct from your rule for $\texttt{pick}(S_1, S_2)$ and the existing Hoare logic rules.

Exercise 4: Separation Logic (*20 points*)

Consider the class `Queue`, implemented using an underlying linked list structure.

```
class Queue {
    // pred queue<alpha> = ...

    Node head;
    Node tail;
    Node sentinel;

    // empty list
    boolean empty() {
        return head == sentinel;
    }

    // dequeue head node
    int deq(int i) {
        int res = head.val;
        head = head.next;
        return res;
    }

    // enqueue new element
    void enq(int i) {
        Node n = new Node();
        n.val = i;
        n.next = sentinel;
        if (tail == sentinel) {
            head = n;
        }
        tail = n;
    }

    // checks whether element is stored in the queue
    boolean search(int x) {
        boolean res = false;
        Node n = head;
        while (n[redacted] != sentinel & !res) {
            if (n.val == x) {
                res = true;
            } else {
                n = n.next;
            }
        }
        return res;
    }
}

class Node {
    int val;
    Node next;
}
```

The `Queue` class maintains pointers to the first and the last element of the queue, therefore a special `sentinel` node is used to mark the end of the list.

a. *(4 points)* Specify an abstract predicate `queue` that captures that `Queue` represents a sequence α.
b. *(3 points)* Write a specification for method `empty` using this abstract predicate.
c. *(3 points)* Write a specification for method `deq` using this abstract predicate.

d. *(3 points)* Write a specification for method `enq` using this abstract predicate.

e. *(7 points)* Write a specification for method `search` and prove that the implementation respects its specification.

Exercise 5: Multithreading *(15 + 5 bonus points)*

Suppose we have the following concurrent program (using the classical parallel composition operator $||$):

$$x := x + n \,||\, n := n + 1$$

a. *(6 points)* First we assume that the assignments are executed atomically. Give a specification for this program and prove its correctness using the Owicki-Gries method.

b. *(3 points)* Now we move to a more realistic setting, where the assignments are not executed atomically. Explain why in that case you cannot prove this program correct using permission-based separation logic.

c. *(6 points)* To make the execution of the statements atomic again, we add locks to the program.

$$(\texttt{lock}\ u; x := x + n; \texttt{unlock}\ u;) \,||\, (\texttt{lock}\ u; n := n + 1; \texttt{unlock}\ u;)$$

Use permission-based separation logic to show that this program is free of data races.

d. *(5 bonus points)* Use the history-based extension of permission-based separation logic to prove what the possible final values of `x` are.