Exam in

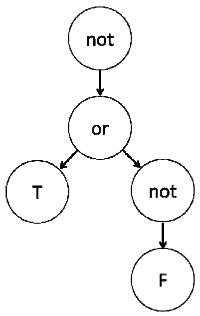
Declarative Languages

Course code:	D7012E
Time:	4 hours, 09:00-13:00
Number of assignments:	6
Total number of points:	30
Date of exam:	2014-12-18
Teacher:	Fredrik Bengtsson, tel. 0920492431, 0738166670
Allowed aiding equipment:	Dictionary

Good luck!

Assignment 1 (6p)

a (2p): Declare a data type `BTree` in Haskell that is able to represent logical formulae. The formulae should contain the binary operations "*and*" and "*or*", the unary operation "*not*", as well as the constant truth-values "*T*" and "*F*". The data type should represent the formulae in a tree-like fashion (sometimes called parse-tree or abstract syntax tree). The constructors corresponding to the formula alternatives should be called *And, Or, Not, T* and *F*. An example of such an expression is *"not (T or (not F))"*, and the corresponding syntax-tree would look like below:



b (1p): State, using your data type from (a), the Haskell expression that would correspond to the above tree.

c (3p): Declare a function `beval` that takes a `BTree` as argument, evaluates the logical formula represented by the `BTree`, and returns its truth-value.

Assignment 2 (6p)

a (3p), Write a predicate, `close(+L1, +L2, -N)`, in prolog, that computes the "closeness" of two lists. If the lists contains identical elements, `N` is 0. For each element that differs between the lists, `N` is increased by 1. A diff is defined as follows: element at position *k* in list 1 differs from element at position k in list 2, for any *k*.

b (3p), Consider the following predicates in prolog:

```
test(N,M):-
  !,
  member(N,[1,2]),
  member(M,[3,4]).

testo(N,M,K,J):-
  test(N,M),
  !,
  test(K,J).
```

Now, consider the following goal:

```
testo(N,M,K,J).
```

How many different solutions will be found when backtracking over this goal?

Assignment 3 (6p)

Consider the following standard Haskell-functions:

```
foldr :: (a -> b -> b) -> b -> [a] -> b
map :: (a -> b) -> [a] -> [b]
```

a (3p), Implement the function `map` with the help of `foldr`. (Your implementation should not use recursion directly, but use `foldr`.)

b (3p), Implement a function

```
split :: [a] -> ([a],[a])
```

that takes a list, `L`, and returns a pair of lists containing the even and odd elements of the list `L`, respectively. Use `foldr` and do not type a recursive implementation directly.

Assignment 4 (3p)

Consider the Haskell Monad class:

```
class  Monad m  where
    (>>=)           :: m a -> (a -> m b) -> m b
    (>>)            :: m a -> m b -> m b
    return          :: a -> m a
    fail            :: String -> m a
```

Given an instance of the monad class, `m`, implement an operator

```
(<=<) :: (b->m c) -> (a->m b) -> (a->m c)
```

that returns a new function where the two monadic functions are sequenced "backwards", such that the resulting function from expression

```
a <=< b
```

is a composed function that sequences `b` first and `a` next.

Assignment 5 (6p)

a (2p), In prolog, implement a predicate, `picknumber(-N,M)`, that computes, via backtracking, the sequence of numbers 2,3,…,M assigned to N.

b (2p), In prolog, implement a predicate, `intFactor(+N, -F, -R)`, that computes the <u>smallest</u> integer factor, F, of N. R is the rest when the factor N is removed. (That is R is rest of N/F, all integers.) Use pickNumber from (a),

c (2p), In prolog, implement a predicate, `primeFactor(+N, -L)`, that computes all the prime factors in N, returned in the list L. Use `intFactor` from (b).

Assignment 6 (3p)

What is the logical equivalent of the following programs:

a (1p)

```
p :- c.
p :- a,b.
```

b (1p)

```
p :- a,!,b.
p :- c.
```

c (1p)

```
p :- c.
p :- a,!,b.
```

State a logical (boolean) expression equivalent to `p`.