Isabelle/HOL Exercises
Trees, Inductive Data Types

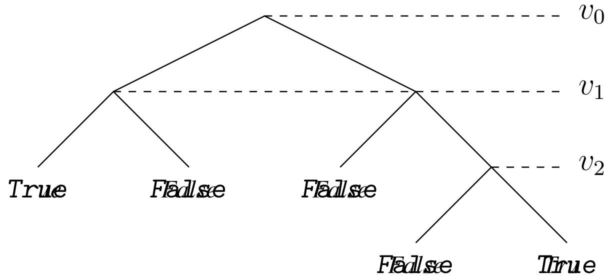
Binary Decision Diagrams

Boolean functions (in finitely many variables) can be represented by so-called *binary decision diagrams* (BDDs), which are given by the following data type:

datatype *bdd = Leaf bool | Branch bdd bdd*

A constructor *Branch b1 b2* that is i steps away from the root of the tree corresponds to a case distinction based on the value of the variable v_i. If the value of v_i is *False*, the left subtree *b1* is evaluated, otherwise the right subtree *b2* is evaluated. The following figure shows a Boolean function and the corresponding BDD.

v_0	v_1	v_2	$f(v_0, v_1, v_2)$
False	*False*	$*$	*True*
False	*True*	$*$	*False*
True	*False*	$*$	*False*
True	*True*	*False*	*False*
True	*True*	*True*	*True*



Exercise 1: Define a function

consts *eval :: "(nat ⇒ bool) ⇒ nat ⇒ bdd ⇒ bool"*

that evaluates a BDD under a given variable assignment, beginning at a variable with a given index.

Exercise 2: Define two functions

consts
bdd_unop :: "(bool ⇒ bool) ⇒ bdd ⇒ bdd"
bdd_binop :: "(bool ⇒ bool ⇒ bool) ⇒ bdd ⇒ bdd ⇒ bdd"

for the application of unary and binary operators to BDDs, and prove their correctness.

Now use *bdd_unop* and *bdd_binop* to define

consts
bdd_and :: "bdd ⇒ bdd ⇒ bdd"

```
  bdd_or :: "bdd ⇒ bdd ⇒ bdd"
  bdd_not :: "bdd ⇒ bdd"
  bdd_xor :: "bdd ⇒ bdd ⇒ bdd"
```

and show correctness.

Finally, define a function

```
consts bdd_var :: "nat ⇒ bdd"
```

to create a BDD that evaluates to `True` if and only if the variable with the given index evaluates to `True`. Again prove a suitable correctness theorem.

Hint: If a lemma cannot be proven by induction because in the inductive step a different value is used for a (non-induction) variable than in the induction hypothesis, it may be necessary to strengthen the lemma by universal quantification over that variable (cf. Section 3.2 in the Tutorial on Isabelle/HOL).

Example: instead of

```
lemma "P (b::bdd) x"
apply (induct b)
```

Strengthening:

```
lemma "∀x. P (b::bdd) x"
apply (induct b)
```

Exercise 3: Recall the following data type of propositional formulae (cf. the exercise on “Representation of Propositional Formulae by Polynomials”)

```
datatype form = T | Var nat | And form form | Xor form form
```

together with the evaluation function `evalf`:

```
constdefs
  xor :: "bool ⇒ bool ⇒ bool"
  "xor x y ≡ (x ∧ ¬ y) ∨ (¬ x ∧ y)"

consts
  evalf :: "(nat ⇒ bool) ⇒ form ⇒ bool"
primrec
  "evalf e T = True"
  "evalf e (Var i) = e i"
  "evalf e (And f1 f2) = (evalf e f1 ∧ evalf e f2)"
  "evalf e (Xor f1 f2) = xor (evalf e f1) (evalf e f2)"
```

Define a function

```
consts mk_bdd :: "form ⇒ bdd"
```

that transforms a propositional formula of type `form` into a BDD. Prove the correctness theorem

theorem *mk_bdd_correct: "eval e 0 (mk_bdd f) = evalf e f"*