# Implementation description “satisfiable”

By Luka Hale and Christian Konig 2013/09/09

## *Contradiction*

The contradiction method makes use of the recursive isContradiction method and supplies it with all possible proposition values using the predefined allVals method. The isContradiction method returns True if the given function returns False for all given proposition values by using the predefined eval function.

```
-- logical contradiction
contradiction :: Form -> Bool
contradiction x = isContradiction (allVals x) x

isContradiction :: [Valuation] -> Form -> Bool
isContradiction [] y = True;
isContradiction (x:xs) y = (not (eval x y)) && (isContradiction xs y)
```

## *Tautology*

The tautology method makes use of the recursive isTautology method and supplies it with all possible proposition values using the predefined allVals method. The isTautology method returns True if the given function returns True for all given proposition values by using the predefined eval function.

```
-- logical tautology
tautology :: Form -> Bool
tautology x = isTautology (allVals x) x

isTautology :: [Valuation] -> Form -> Bool
isTautology [] y = True
isTautology (x:xs) y = (eval x y) && (isTautology xs y)
```

## *Entailment*

The entails method makes use of the recursive isEntailsment method and supplies it with all possible proposition values for the first function using the predefined allVals method. The isEntailment method returns True if the second function evaluates true for all proposition values for which the first function returns True as well, while using the predefined eval function.

```
--logical entailment  @TODO
entails :: Form -> Form -> Bool
entails x y                    = isEntailment x y (allVals x)

isEntailment :: Form -> Form -> [Valuation] -> Bool
isEntailment x y [] = True
isEntailment x y (z:zs) = (((((eval z x) == True && (eval z y) == True) || ((eval z x) == False))) && isEntailment x y zs)
```

## *Equivalence*

The tautology method makes use of the recursive isEquiv method and supplies it with all possible proposition values for the first function using the predefined allVals method. The isEquiv method returns True if the two given functions return the same result for all given proposition values by using the predefined eval function.

```
-- logical equivalence
equiv :: Form -> Form -> Bool
equiv x y = isEquiv x y (allVals x)

isEquiv :: Form -> Form -> [Valuation] -> Bool
isEquiv x y [] = True
isEquiv x y (z:zs) = ((eval z x) == (eval z y)) && (isEquiv x y zs)
```

Time spent: 4 h