Domain should be declare and give it a name

(declare-sort D)

Declare a constants (a,b,c, pidocchio) and specify the domain D

(declare-const c D)

Declare a binary function f

(declare-fun f (D D) d )

Declare function g unary

(declare-fun g (D) D)

Declare predicate R binary but is a predicate

(declare-fun R (D D) Bool)

P unary predicate

(declare-fun P (D) Bool)

(declare-const p Bool)

P unary predicate and c constant

P(c) is wrong Z3 use (P c)

R binary predicate c,d constant

R (c,d) wrong z3 use (R c d)

How to write down formulas

Use assert

^ and

V or

→ =>

¬ not

P(c) ^ Rc,d)

(and (P c) (R c d))

Quantifiers

∃ x<formula>

(exists ((xD)) <formula>)

∃ x ∃ y <formula>

(exists ((x D) (y D)) <formula>)

Universal quantifiers

∀x <formula>

(forall ((x D)) <formula>)

∀ x ∀ y ∀ z <formula>

(forall ((x D) (y D)(z D)) <formula>)

* List of declaration:
* Sort (domain)
* Constants
* Functions
* Predicate
* List of assertion
* (check-sat)
* (get-model)

(get-value ( <list of terms of formulas> )) !it is more useful than (get-model)!

(get-value ((K1 Italy) (K2 Italy) (K3 Italy)))

The answer will be:

(K1 Italy) = true

(K2 Italy) = false

(K3 Italy) = false

## EXERCISE 1

(1) No one buys the Times unless he is highly educated. No illiterate person

can read. No one who cannot read is very educated. Pinocchio is illiterate.

Prove that: ’Pinocchio does not buy the Times’. [Hint: see the course notes,

example 22 p.57.]

; declare Domain

(declare-sort D)

; declar constant Pinocchio

(declare-const p D)

; declare unary predicate Compra

(declare-fun C (D) Bool)

; declare unary predicate Legge

(declare-fun L (D) Bool)

; declare unary predicate Analfabeta

(declare-fun A (D) Bool)

; declare unary predicate Intelligent

(declare-fun I (D) Bool)

; assert the formulas

(assert (forall ((x D)) (=> (C x) (I x))))

(assert (forall ((x D)) (not (and (A x) (L x)))))

(assert (forall ((x D)) (=> (not (L x)) (not (I x)))))

(assert (A p))

(assert (C p))

(check-sat)

## EXERCISE 2

Show that 3 colors are sufficient to color the part of the European map that includes the following sets of countries: Italy, Austria, Switzerland, France, Germany. [Hint: use universal formulae (not just propositional logic!), see the course notes, example 23 p.58.]

; declare Domain

(declare-sort D)

(declare-const italy D)

(declare-const austria D)

(declare-const switzerland D)

(declare-const france D)

(declare-const germany D)

(declare-fun R (D D) Bool)

(declare-fun K1 (D) Bool)

(declare-fun K2 (D) Bool)

(declare-fun K3 (D) Bool)

(assert (forall ((x D) (y D))( => (R x y) (not(and (K1 x) (K1 y))))))

(assert (forall ((x D) (y D))( => (R x y) (not(and (K2 x) (K2 y))))))

(assert (forall ((x D) (y D))( => (R x y) (not(and (K3 x) (K3 y))))))

;exactly one out of three

(assert (forall ((x D)) (and (or (K1 x) (K2 x) (K3 x)) (not (and (K1 x)(K2 x)))(not (and (K1 x)(K3 x)))(not (and (K2 x)(K3 x))))))

(assert(forall ((x D)) (K1 x)))

(assert(forall ((x D)) (K2 x)))

(assert(forall ((x D)) (K3 x)))

(assert(forall ((x D)) (not (and(K1 x) (K2 x)))))

(assert(forall ((x D)) (not (and(K1 x) (K3 x)))))

(assert(forall ((x D)) (not (and(K2 x) (K3 x)))))

(assert (R italy switzerland))

(assert (R italy france))

(assert (R italy austria))

(assert (R france switzerland))

(assert (R france germany))

(assert (R germany switzerland))

(assert (R germany austria))

(assert (R switzerland austria))

(check-sat)

----------------

; declare Domain

(declare-sort D)

(declare-const italy D)

(declare-const austria D)

(declare-const switzerland D)

(declare-const france D)

(declare-const germany D)

(declare-fun R (D D) Bool)

(declare-fun K1 (D) Bool)

(declare-fun K2 (D) Bool)

(declare-fun K3 (D) Bool)

(assert (forall ((x D) (y D))( => (R x y) (not(and (K1 x) (K1 y))))))

(assert (forall ((x D) (y D))( => (R x y) (not(and (K2 x) (K2 y))))))

(assert (forall ((x D) (y D))( => (R x y) (not(and (K3 x) (K3 y))))))

;exactly one out of three

(assert (forall ((x D)) (and (or (K1 x) (K2 x) (K3 x)) (not (and (K1 x)(K2 x)))(not (and (K1 x)(K3 x)))(not (and (K2 x)(K3 x))))))

(assert (R italy switzerland))

(assert (R italy france))

(assert (R italy austria))

(assert (R france switzerland))

(assert (R france germany))

(assert (R germany switzerland))

(assert (R germany austria))

(assert (R switzerland austria))

(check-sat)

(get-value ((K1 italy) (K2 italy)(K3 italy)))

(get-value ((K1 austria) (K2 austria)(K3 austria)))

(get-value ((K1 switzerland) (K2 switzerland)(K3 switzerland)))

(get-value ((K1 france) (K2 france)(K3 france)))

(get-value ((K1 germany) (K2 germany)(K3 germany)))

------------------------

; declare Domain

(declare-sort D)

(declare-const a D)

(declare-const b D)

(declare-const c D)

(declare-fun C (D) Bool)

(declare-fun F (D) Bool)

(declare-fun L (D) Bool)

; assert everyone among a,b,c is either a C or a F (but not both)

(assert (and(or(C a) (F a)) (not (and(C a) (F a)))))

(assert (and(or(C b) (F b)) (not (and(C b) (F b)))))

(assert (and(or(C c) (F c)) (not (and(C c) (F c)))))

;at least one L between a b c

(assert(or (L a)(L b)(L c)))

; at most one werewolf

(assert(not(and (L a)(L b))))

(assert(not(and (L a)(L c))))

(assert(not(and (L b)(L c))))

; statement 1

(assert (and (=> (C a)(L c))(=> (F a) (not(L c)))))

; statement 2

(assert(and (=> (C b)(not(L b)))(=> (F b)(L b))))

; statement 3

(assert(=>(C c)(or ((and (F a)(F b))(and (F a)(F c))(and (F b)(F c))))))

(assert(=>(F c)(not(or ((and (F a)(F b))(and (F a)(F c))(and (F b)(F c)))))))

------------------------

## EXERCISE 3

(declare-sort D)

(declare-const a D)

(declare-const b D)

(declare-const c D)

(declare-fun C (D) Bool)

(declare-fun F (D) Bool)

(declare-fun L (D) Bool)

(assert(and(or (C a) (F a))(not(and (C a)(F a)))))

(assert(and(or (C b) (F b))(not(and (C b)(F b)))))

(assert(and(or (C c) (F c))(not(and (C c)(F c)))))

(assert(or (L a) (L b)(L c)))

(assert(and (not(and (L a)(L b)))(not(and (L a)(L c)))(not(and (L c)(L b)))))

(assert(and (=> (C a)(L c))(=> (F a)(not(L c)))))

(assert(and (=> (C b)(not(L b)))(=> (F b)(not(L b)))))

(assert(=>(C c)(or (and (F a)(F b))(and(F a)(F c))(and(F c)(F b)))))

(assert(=>(F c)(not(or (and (F a)(F b))(and(F a)(F c))(and(F c)(F b))))))

(check-sat)

(get-model)