11-02-15

EX1

Alphabet:

Sa(x), C(x), B(x), incommand(x,y), onboard(x,y), tb(x,y), tl(x,y), L(x), T(x), ts(x,y), Se(x), O(x)

Axioms:

Forall x. C(x) implies Sa(x) \\ ISA

Forall x,y. incommand(x,y) implies C(x) and B(y) \\ typing

Forall x,y. incommand(x,y) implies onboard(x,y) \\ subset

Forall x. C(x) implies #{y|incommand(x,y)}<=1 \\ multiplicity

Forall y. B(y) implies 1<=#{x|incommand(x,y)}<=1 \\ multiplicity

Forall x,y. onboard(x,y) implies Sa(x) and B(y) \\ typing

Forall x. Sa(x) implies #{y|onboard(x,y)}<=1 \\ multiplicity

Forall y. B(y) implies 1<=#{x|onboard(x,y)} \\ multiplicity

Forall x,y. tb(x,y) implies B(x) and T(y) \\ typing

Forall y. T(y) implies 1<=#{x|tb(x,y)}<=1 \\ multiplicity

Forall x,x’y. tb(x,y) and tb(x’,y) implies x=x’ \\ key

Forall x,y. tl(x,y) implies T(x) and L(x) \\ typing

Forall x. T(x) implies 1<=#{y|tl(x,y)}<=1 \\ multiplicity

Forall x,y. ts(x,y) implies T(x) and Se(y) \\ typing

Forall x. T(x) implies 1<=#{y|ts(x,y)}<=1 \\ multiplicity

Forall x. O(x) implies Se(x) \\ ISA

EX2

1. The instantiation is not complete because there are two ISA in this UML diagram. So we need to complete this instantiation using the following procedure:

Given a UML diagram T

Iold = 0, Inew = =

While (Iold and Inew are different) do

For each (forall x. A(x) implies B(x) in T) do

For each a in A^Inew do

B^Inew = B^Inew union {a}

Similarly for each subset constraints forall x,y. P(x,y) implies R(x,y)

I=Inew

Return I

I = (Obj^I,Sa^I, C^I, B^I, incommand^I, onboard^I, tb^I, tl^I, L^I, T^I, ts^I, Se^I, O^I)

I0:

Sa^I = {Dustin, Lubber, Rusty}

C^I={Alice,Beth}

B^I = {Consitution,Enterprise}

incommand^I = {(Alice, Constitution), (Beth, Enterprise)}

onboard^I = {(Dustin, Constitution), (Lubber, Constitution), (Rusty, Constitution)}

tb^I = {}

tl^I = {}

L^I = {}

T^I = {}

ts^I ={}

Se^I = {Mediterran}

O^I = {Altantic}

We have ISAs so we are going to take instance of subclass and put in its superclass. The same also for subset relation

I1:

Sa^I = {Dustin, Lubber, Rusty, Alice,Beth }

C^I={Alice,Beth}

B^I = {Consitution, Enterprise}

incommand^I = {(Alice, Constitution), (Beth, Enterprise)}

onboard^I = {(Dustin, Constitution), (Lubber, Constitution), (Rusty, Constitution), (Alice, Constitution), (Beth, Enterprise)}

tb^I = {}

tl^I = {}

L^I = {}

T^I = {}

ts^I ={}

Se^I = {Mediterran, Atlantic}

O^I = {Altantic}

Repeat until we have In+1 = In

I2:

Sa^I = {Dustin, Lubber, Rusty, Alice,Beth }

C^I={Alice,Beth}

B^I = {Consitution, Enterprise}

incommand^I = {(Alice, Constitution), (Beth, Enterprise)}

onboard^I = {(Dustin, Constitution), (Lubber, Constitution), (Rusty, Constitution), (Alice, Constitution), (Beth, Enterprise)}

tb^I = {}

tl^I = {}

L^I = {}

T^I = {}

ts^I ={}

Se^I = {Mediterran, Atlantic}

O^I = {Altantic}

I2 = I1 so the instantiation is complete

Now we are going to check if this instantiation is correct, so if I|=T. This means that we want to verify if all axioms in T are evaluate true over this interpretation I.

Each instance of Commander is also instance of Sailor, ok. For each instance of Commander there is at most one Boat incommand, true. Each Boat has one and only one Commander incommand, correct. Each instance of incommand is also instance of onboard. Each instance of Sailor have at most one Boat onboard, and each Boat has at least one Sailor onboard, ok. Finally each instance of Ocean is also instance of Sea.

The instantiation is correct.

1. q(x)<- C(x) and Exists y. incommand(x,y) and B(y) and Exists z. onboard(z,y) and S(z) and not C(z)

q(x): {Alice}

1. q(x) <- B(x) and forall y. onboard(y,x) implies C(y)

q(x):{Enterprise}

1. q(x) <- B(x) and forall y. onboard(y,x) and S(y) implies not C(y)

q(x):{}

EX3

Model checking a formula phi over transition system T = <S,Ra,Pi> (S is set of states, Ra set of transitions, Pi a mapping function from a set of proposition P to a subset of S) means that we want to verify if the initial state of T is in the extension of phi over T. With model checking we return a subset of S in which each element satisfies phi. To compute this set we are going to use the labelling algorithm that consists in labelling states of T with predicates that are true in them. The extensions of least fixpoint and greatest fixpoint are computed using the approximates theorem Tarksi-Knaster.

νX.muY.((a and<next>X) or (not b and<next>Y))

We are going to find the greatest fixpoint (GFP) because of the presence of vX

[X0] = {1,2,3,4}

[X1] = [mu Y (a and <next>X0) or (not b and <next>Y)]

We are going to find the least fixpoint (LFP) because of the presence of muY

[Y00] = {}

[Y01] = [(a and <next>X0) or (not b and <next>Y00)] = ([a] intersect PreE(next,[X0])) union ([not b] intersect PreE(next,[Y00])) = ({2,4} intersect {1,2,3,4}) union ({1,2} intersect {}) = {2,4} union {} = {2,4}

[Y02] = [(a and <next>X0) or (not b and <next>Y01)] = ([a] intersect PreE(next,[X0])) union ([not b] intersect PreE(next,[Y01])) = ({2,4} intersect {1,2,3,4}) union ({1,2} intersect {1,3,4}) = {2,4} union {1} = {1,2,4}

[Y03] = [(a and <next>X0) or (not b and <next>Y02)] = ([a] intersect PreE(next,[X0])) union ([not b] intersect PreE(next,[Y02])) = ({2,4} intersect {1,2,3,4}) union ({1,2} intersect {1,3,4}) = {2,4} union {1} = {1,2,4}

[Y03] = [Y02] - - > found LFP

[X1] = {1,2,4}

[X2] = [mu Y (a and <next>X1) or (not b and <next>Y)]

We are going to find the least fixpoint (LFP) because of the presence of muY

[Y10] = {}

[Y11] = [(a and <next>X1) or (not b and <next>Y10)] = ([a] intersect PreE(next,[X1])) union ([not b] intersect PreE(next,[Y10])) = ({2,4} intersect {1,3,4}) union ({1,2} intersect {}) = {4} union {} = {4}

[Y12] = [(a and <next>X1) or (not b and <next>Y11)] = ([a] intersect PreE(next,[X1])) union ([not b] intersect PreE(next,[Y11])) = ({2,4} intersect {1,3,4}) union ({1,2} intersect {3,4}) = {4} union {} = {4}

[Y12] = [Y11] - - > found LFP

[X2] = {4}

[X3] = [mu Y (a and <next>X2) or (not b and <next>Y)]

We are going to find the least fixpoint (LFP) because of the presence of muY

[Y20] = {}

[Y21] = [(a and <next>X2) or (not b and <next>Y20)] = ([a] intersect PreE(next,[X2])) union ([not b] intersect PreE(next,[Y20])) = ({2,4} intersect {3,4}) union ({1,2} intersect {}) = {4} union {} = {4}

[Y22] = [(a and <next>X2) or (not b and <next>Y21)] = ([a] intersect PreE(next,[X2])) union ([not b] intersect PreE(next,[Y21])) = ({2,4} intersect {3,4}) union ({1,2} intersect {3,4}) = {4} union {} = {4}

[Y22] = [Y21] - - > found LFP

[X3] = {4}

[X3] = [X2] - - > found GFP

[phi] = {4}

It is 1 in [phi]? No, so phi is not satisfied by this transition system

Now we are going to do model checking of CTL formula. Given a Kripke model (KM) = <S,I,R,AP,L> (S is set of states, I set of initial states, R set of transitions, AP set of atomic proposition and L labelling function) and a CTL formula phi we want to verify if KM,s |= phi where s is state of S. With model checking we return a subset of S in which each state satisfies phi. To compute it we need to exploit the syntactic structure of CTL formula, in particular, we translate each sub formula of CTL into mu calculus formula and then we apply the labelling algorithm to find their extensions.

EF (a and EX (AG b))

alpha = AG b = vX b and [next] X

beta = EX alpha

gamma = a and beta

delta = EF gamma

[alpha] = [vX b and [next] X]

We are going to find the greatest fixpoint (GFP) because of the presence of vX

[X0] = {1,2,3,4}

[X1] = [b and [next] X0] = [b] intersect PreA(next,[X0]) = {3,4} intersect {1,2,3,4} = {3,4}

[X2] = [b and [next] X1] = [b] intersect PreA(next,[X1]) = {3,4} intersect {2,4} = {4}

[X3] = [b and [next] X2] = [b] intersect PreA(next,[X2]) = {3,4} intersect {4} = {4}

[X3] = [X2] - - > found GFP

[alpha] = {4}

[beta] = [EX alpha] = [<next> alpha] = PreE(next,[alpha]) = {3,4}

[gamma] = [a and beta] = [a] intersect [beta] = {2,4} intersect {3,4} = {4}

[delta] = [EF gamma] = [muX gamma or <next>X]

We are going to find the least fixpoint (LFP) because of the presence of muX

[X0] = {}

[X1] = [gamma or <next> X0] = [gamma] union PreE(next,[X0]) = {4} union {} = {4}

[X2] = [gamma or <next> X1] = [gamma] union PreE(next,[X1]) = {4} union {3,4} = {3,4}

[X3] = [gamma or <next> X2] = [gamma] union PreE(next,[X2]) = {4} union {2,3,4} = {2,3,4}

[X4] = [gamma or <next> X3] = [gamma] union PreE(next,[X3]) = {4} union {1,2,3,4} = {1,2,3,4}

[X5] = [gamma or <next> X4] = [gamma] union PreE(next,[X4]) = {4} union {1,2,3,4} = {1,2,3,4}

[X5] = [X4] - - > found LFP

[delta] = {1,2,3,4}

It is 1 in [delta] ? YES, so the CTL formula is true over this transition system

EX4

Check if the Hoare triple {P} while g do S {Q} is correct cannot be done automatic, ut we need inference rule.

Assuming to have a candidate invariant I, we show

1. P implies I
2. {I and g} S {I} = {I and g} implies wp(S,I)
3. {I and not g} implies {Q}

If candidate I respect these operations is called invariant.

If we have a candidate these operations are done automatically, otherwise it could be difficult to manage with it. The problem of check if the Hoare triple is correct or not is undecidable because there is no sound or complete technique to do this or to find all possible invariants. The only thing that we can do is to check if candidate invariant is invariant or not.

Candidate: i+j <=10

{i=0 AND j=10} while(i<10) do {j=j-1; i:= i+1} {j=0}

* P implies I

{i=0 and j=10} implies {i+j<=10}

Yes, because in this case i+j=10 and it is always <=10

* {I and not g} implies {Q}

{i+j<=10 and i>=10} implies {j=0}

Ok, because if the i+j should be <=10 and i>=10 means that i=10 and j=0

* Now we are going to compute wp(S,I)

{i+j<=10} is wp

j=j-1

{i+1+j <=10}

i=i+1

{i+j<=10}

{I and g} implies wp(S,I)

{i+j<=10 and i<10} implies {i+j<=10}

The condition is satisfied because for hypothesis we have i+j <=10 and also for

thesis so the condition is always true

i+j<=10 is invariant

EX5

q1() :- edge(r,b), edge(b,g), edge(g,r)

q2() :- edge(i,f), edge(f,v), edge(v,i), edge(i,a), edge(a,v), edge(a,s), edge(s,i).

We want to check if q1() is contained in q2(), this means that forall x. q1(x) implies q2(x) is valid. Validity means that forall I,alpha |= forall x. q1(x) implies q2(x).

In FOL validity is undecidable, but with conjunctive queries we can make queries satisfiable because we can transform into databases.

We need 3 steps:

1. Freeze free variables, i.e. substitute free variable with fresh constant

In this case we don’t need this operation because our queries are already Boolean so we don’t substitute anything

1. Build canonical database corresponding of q1

Iq1 = (delta^I1,e^I1,c^I1)

delta^I1 = {r,b,g}

e^I1 = {(r,b),(b,g),(g,r)}

c^I1 = {}

1. Check if q2 is true over this databases, i.e. find an assignment forall free variables

Iq2 = (delta^I2,e^I2,c^I2)

delta^I2 = {i,f,v,a,s}

e^I2 = {(i,f), (f,v), (v,i), (i,a), (a,v), (a,s), (s,i)}

c^I2 = {}

alpha(i) = r

alpha(f) = b

alpha(v) = g

alpha(a) = b

alpha(s) = g

This is a satisfying assignment

Now we want to find a homomorphism. A homomorphism is a mapping between two interpretations, between elements of two domains h:delta^I implies delta^J such that:

1. h(c^I) = c^J
2. (x,y) in e^I then (h(x),h(y)) in e^J

Find a homomorphism means that we want to guess it, in such way that it respect these two properties. In particular there is a theorem that says that if we have an assignment alpha, that it is a satisfying assignment, we can transform alpha in two homomorphism between two canonical interpretation, Iq1|= Iq2 iff Iq2 implies Iq1

The first property is satisfied because we don’t have constant.

We are going to check the second.

(i,f) in e^I2 then (h(i),h(f)) is in e^I1? Yes, (r,b) is in e^I1

(f,v) in e^I2 then (h(f),h(v)) is in e^I1? Yes, (b,g) is in e^I1

(v,i) in e^I2 then (h(v),h(i)) is in e^I1? Yes, (g,r) is in e^I1

(i,a) in e^I2 then (h(i),h(a)) is in e^I1? Yes, (r,b) is in e^I1

(a,v) in e^I2 then (h(a),h(v)) is in e^I1? Yes, (b,g) is in e^I1

(a,s) in e^I2 then (h(a),h(s)) is in e^I1? Yes, (b,g) is in e^I1

(s,i) in e^I2 then (h(s),h(i)) is in e^I1? Yes, (g,r) is in e^I1