9-06-14

EX1

Alphabet:

A(x), B(x), Rab(x,y), C(x), Rcb(x,y) Rac(x,y)

Axioms:

Forall x,y. Rab(x,y) implies A(x) and B(y) \\ typing

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

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

Forall x,y. Rac(x,y) implies A(x) and C(y) \\ typing

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

Forall x. A(x) implies 1<=#{y|Rac(x,y)} \\ multi

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

Forall x,y. Rac(x,y) implies Rab(x,y) \\ subset

Forall x,y Rcb(x,y) implies Rab(x,y) \\ subset

Forall x. A(x) implies 1<=#{y|Rab(x,y)} after refinements \\ multi

Forall y. B(y) implies 1<=#{x|Rab(x,y)}<=1 after refinements \\ multi

EX2

Model checking a closed mu calculus formula phi over transition systems T = <S,Ra,Pi> (S set of states, Ra is set of transitions such that action a is in A and Pi is mapping function from a set of proposition P to a subset of S) means to verify if the initial state of T (in S) is in the extension of phi over T, given a valuation V. With Model checking we return the subset of S, and we compute it using labelling algorithm, that consist in labelling states of T with predicates that are true in them. The extensions of least fixpoint and greatest fixpoint are computed using the Tarski-Knaster approximates theorem.

vXmuY ((a and <next> X ) or <next> Y)

We have vX so we are going to find the greatest fixpoint (GFP)

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

[X1] = [muY((a and <next> X0) or <next>Y)]

mX identify that now we are going to find the least fixpoint (LFP)

[Y00] = {}

[Y01]=[((a and <next> X0) or <next>Y00)] =(([a] intersect PreE(next,[X0])) union PreE(next,[Y00])) = (({2,4} intersect {1,2,3,4}) union {}) = {2,4}

[Y02] = [((a and <next>X0) or <next>Y01)] = (([a] intersect PreE(next,[X0])) union PreE(next,[Y01])) = (({2,4} intersect {1,2,3,4}) union {1,2,3,4}) = {1,2,3,4}

[Y03] = [((a and <next>X0) or <next>Y01)] = (([a] intersect PreE(next,[X0])) union PreE(next,[Y02])) = (({2,4} intersect {1,2,3,4}) union {1,2,3,4}) = {1,2,3,4}

[Y03] = [Y02] so, {1,2,3,4} is LFP

[X1] = {1,2,3,4}

[X1]=[X0] so, {1,2,3,4} is GFP

Is T |= phi? YES because 1 belong to [phi] = {1,2,3,4}, so phi is true in transition system

Now, to we need to do model checking with CTL formula. So, given a formula a CTL formula phi and a Kripke model (KM) = <S,I,R,AP,L> where S is set of states, I is set of initial states, R is set of transitions, AP is set of atomic preposition and L is mapping function L:S->2^AP; we want to check if KM,s |= phi where s is state of S. Model checking return a set of states, subset of S, in which each state satisfies phi. To compute this set we need to exploit syntactic structure of CTL formula, in particular we translate each CTL sub formula in mu formula and then we apply the labelling algorithm to find their extensions.

AG(EXa implies AXAXEGa)

alpha = EG a

beta = AX alpha

gamma = AX beta

delta = EXa

sigma = delta implies gamma

tau = AG sigma

[alpha] = [vX a and <next> X]

There is vX so we want to find the greatest fixpoint (GFP), so we will have that

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

[X1] = [a and <next> X0] = [a] intersect PreE(next,[X0]) = {2,4} intersect {1,2,3,4} = {2,4}

[X2] = [a and <next> X1] = [a] intersect PreE(next,[X1]) = {2,4} intersect {1,2,3,4} = {2,4}

[X2] = [X1] - - > {2,4} is GFP

[beta] = [AX alpha] = [[next] alpha] = PreA(next,[alpha]) = {1,3,4}

[gamma] = [AX beta] = [next] beta] = PreA(next, [beta]) = {3,4}

[delta] = [EX a] = [<next> a] = PreE(next,[a]) = {1,3,4}

[sigma] = [delta implies gamma] = [not delta or gamma] = [not delta] union [gamma] = {2} union {3,4} = {2,3,4}

[tau] = [AG sigma] = [vX sigma and [next] X]

We have vX se we need to compute the GFP

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

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

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

[X2] = [X1] -- > {3,4} is GFP

[tau] = {3,4}

Is 1 in [tau] ? NO, so the CTL formula is False in this transition system

EX 3

1. q() <- Exists x. Student(x) and Exists y. Exam(y) and passed(x,y) \\ IT is a CQ
2. q() <- Exists x. Student(x) and Exists y. (Exam(y) and passed(x,y)) and forall z passed(x,z) and Exam(z) implies z = y \\ IT is not a CQ
3. q() <- Exists x. Student(x) and Exists y,y’. Exam(y) and Exam(y’) and passed(x,y) and passed(x,y’) and y noteq y’ \\ IT is a CQ
4. q() <- Exists x. Student(x) and not Exists y. Exam(y) and passed(x,y) \\ IT is a CQ
5. q() <- Exists x. Student(x) and forall y. Exam(y) and passed(x,y) \\ IT is not a CQ
6. q() <- Exists x,y. Student(x) and Student(y) and (forall z. passed(y,z) and Exam (z) implies passed(x,z))

EX4

EX5

wp(d,Q) = {s|forall s’. ((d,s)->s’)->s’ |=Q}

All the states s such that the execution of program d in that state s, gives s’ that satisfies post-condition Q. So, wp gives us the minimum condition such that you will achieve Q by executing d. Since we don’t have while instruction we can compute wp automatically , starting from below and going backward.

{x>0 and y>=0 and y=0} or {x>0 and y+1=2y} = {x>0 and y=0} or {x>0 and y=1} = {x>0 and y=0 | false} = {x>0 and y=0 } is wp

x:=y+1

{x>0 and y>=0 and y=0} or {x>0 and x=2y}

if(x>0 & y>=0) then{

{y=0}

x:=y-x;

*{x=x-y}={y=0}*

y:=x-y

*{x=y}*

}

else if (x>0) then

*{x-y=y}={x=2y}*

x:=x-y

*{x=y}*

*{x=y}*