22/01/2014

EX 1

Alphabet:

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

Axioms:

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

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

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

Forall x C(x) implies not D(x) \\ disjunction

Forall x A(x) implies C(x) or D(x) \\ complete

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

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

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

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

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

EX2

Model checking a closed mu calculus formula phi over a transition system T = < S,Ra,Pi> means to verify if the initial state of T(in S) is in the extension of phi over T, given a valuation (S is set of states, Ra is set of transitions and Pi is a mapping function from a set of proposition P to a subset of S). We can compute it with labelling algorithm, that consist of labelling the states of T with predicates that are true in them. The extension of least fixpoint and greatest fixpoint are computed by applying Tarski-Knaster approximation theorem.

muXvY (a or [next]X) and [next]Y

First we have mu X so we understand that we want to compute least fixpoint (LFP)

[X0]= 0

[X1] =[ vY a or [next]X0) and [next]Y]

We have vY so we want to find a greatest fixpoint(GFP)

[Y00] = {1,2,3,4}

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

[Y02] = [a or [next]X0) and [next]Y01] = {2,4} intersect {3,4} = {4}

[Y02] = [Y01] 🡺 {4} is GFP

[X1] = {4}

[X2] =[ vY a or [next]X1) and [next]Y]

We have vY so we need to fin GFP

[Y10] = {1,2,3,4}

[Y11] = [a or [next]X1 and [next] Y10] = ({2,4} union PreA(next,[X1]) )intersect PreA(next,[Y10]) = ({2,4} union {3,4}) intersect {1,3,4} = {3,4}

[Y12] == [a or [next]X1 and [next] Y11] = ({2,4} union PreA(next,[X1]) )intersect PreA(next,[Y11]) = {2,3,4} intersect {3,4} = {3,4}

[Y12]= [Y11] we found GFP

[X2] = {3,4}

[X3] =[ vY a or [next]X1) and [next]Y]

We have vY so we need to fin GFP

[Y20] = {1,2,3,4}

[Y21] = [a or [next]X2 and [next] Y20] = ({2,4} union PreA(next,[X2]) )intersect PreA(next,[Y20]) = ({2,4} union {3,4}) intersect {1,3,4} = {3,4}

[Y22] = [a or [next]X2 and [next] Y21] = ({2,4} union PreA(next,[X2]) )intersect PreA(next,[Y21]) = ({2,4} union {3,4}) intersect {3,4} = {3,4}

[Y22] = [Y21] we found GFP

[X3] = {3,4}

[X3] = [X2] we found LFP

It Is T |= phi? No, because 1 not in the {3,4}

Model checking CTL formula we need to consider Kripke model <S,I,R,AP,L> build over T (S is set of states, I is set of initial states, R is set of transition, AP set of atomic proposition and L labelling function L:S->2^AP). Give a Kripke model and CTL formula phi, model checking means that KM,s |= phi where s is state of S and it return a set of states, a subset of S such that they satisfy phi. To compute this set of states we exploit syntactic structures of CTL formula, translating its sub formula into mu calculus formula and then applying to each mu calculus formula the labelling algorithm to find their extensions. CTL formula are interpreted as over branching time structures, that it say over trees instead a linear time structure as for LTL.

AFAG a

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

beta =AF alpha

[apha] = [vX a and [next]X]

We want to find the greatest fixpoint

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

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

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

[X2] = [X1] GFP is found

[beta] = [AF alpha] = [mu X alpha or [next] X]

We are going to find least fixpoint

[X0] = 0

[X1] = [alpha or [next] X0] = [alpha] union PreA(next,[X0]) = {4} union {} = {4}

[X2] = [alpha or [next] X1] = [alpha] union PreA(next,[X1]) = {4} union {3,4} = {3,4}

[X3] = [alpha or [next] X2] = [alpha] union PreA(next,[X2]) = {4} union {3,4} = {3,4}

[X3] = [X2] so LFP is found

Is is T|= phi? No because 1 is not in {3,4}, so CTL formula is false

EX5

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

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

At the end of program x=100. We star from the end as you can se below

{y = -50} is the wp

x:= y+50

{y>0 and y = 0} or {y<=0 and y = -50} 🡪 the first is false so {y<=0 and y = -50} is ok

if (y>0) then

x:=y+100

{y+100+y=100}= {y=0}

else x:=y+200

{y+200+y = 100} = {y = -50}

{x+y = 100}

x:=x+y

{x:=100