09-07-14

EX1

Alphabet:

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

Axioms:

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

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

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

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

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

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

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

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

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

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

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

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

EX2

Model checking a closed mu calculus formula phi over transition system T = <S,Ra,Pi> (S is set of states, Ra is set of transition such that action a is in A and Pi 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. When we apply model checking we return a subset of S in which each state satisfy phi. To compute it we need to apply 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 applying the Tarski-Knaster approximates theorem.

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

We have vX and we understand that we want to compute greatest fixpoint (GFP)

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

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

mu Y identify that we are going to compute least fixpoint (LFP) so

[Y00] = {}

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

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

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

[Y04] = [(a and <next> X0) or <next> Y03] = ([a] intersect PreE(next,[X0])) union PreE(next,[Y03]) = ({2} intersect {1,2,3,4}) union {1,2,3,4} = {1,2,3,4}

[Y04] = [Y03] - - > {1,2,3,4} is LFP

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

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

So the entire formula [phi] = {1,2,3,4} but it is 1 is in {1,2,3,4}? YES, so phi is true in this transition system

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

AG(EX(EX a or EXEX a))

alpha = EX a

beta = EX alpha

gamma = alpha or beta

delta = EX gamma

sigma = AG delta

[alpha] = [<next> a] = PreE(next,[a]) = {1,3}

[beta] =[EX alpha] = [<next> alpha] = PreE(next,[alpha]) = {2,3,4}

[gamma] = [alpha or beta] = [alpha] union [beta] = {1,2,3,4}

[delta] =[EX gamma] = [<next> gamma] = PreE(next,[gamma]) = {1,2,3,4}

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

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

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

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

[X2] = [delta and [next] X1] = [delta] intersect PreA(next,[X1]) = {1,2,3,4} intersect {1} = {1}

[X3] = [delta and [next] X2] = [delta] intersect PreA(next,[X2]) ={1,2,3,4} intersect {} = {}

[X4] = [delta and [next] X3] = [delta] intersect PreA(next,[X3]) ={1,2,3,4} intersect {} = {}

[X4] = [X3] {} is GFP

[sigma] ={}

So the entire formula [sigma] = {}, it is 1 is in {}? NO, so sigma is false in this transition system

EX3

1. q() <- Exists x. Person(x) and Exists y. Appetizer(y) and likes(x,y) and Exists z. MainCourse(z) and likes(x,z) \\ It is a CQ
2. q() <- Exists x. Person(x) and Exists y, y’. Appetizer(y) and likes(x,y) and Appetizer(y’) and likes(x,y’) and y noteq y’ and Exists z. MainCourse(z) and likes(x,z) \\ It is a CQ
3. q() <- Exists x. Person(x) and Exists z. MainCourse(z) and likes(x,z) and (forall y. MainCourse(y) and likes(x,y) implies y= z) \\ IT is not a CQ
4. q() <- Exists x. Person(x) and Forall y. Appetizer(y) and likes(x,y) \\ IT is not a CQ
5. q() <- Exists x,y. Person(x) and Person(y) and Forall z. (Appetizer(z) and likes(y,z)) implies likes(x,z)

EX4

We exclude White because it is not in Person. Z variable is not a person so it should be white but neither Smith o Brown o null1 like White, so I exclude them. Remain only null2 and Green, but I cannot return null2. So the result should be Green.

Q(x): {Green}

EX5

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

All states s such that the execution of the program d in state s give a state s’ that satisfies the post-condition Q. So, the wp gives you the minimum condition such that you will achieve Q by executing d. Since there is no while instruction we can compute wp automatically starting from below and going backwards.

{y=100 } or {x!=0 90-y+y = 100} = {y=100} or {false} = {y=100} is wp

x:= 90-y

[{ y>0 and y = 100 and } or (y<=0 and y = 90}] or [x!= 0 and x=100] = [{y=100 } or {false} [x!= 0 and x+y=100] = {y= 100} or {x!=0 and x+y= 100}

if (x=0) then {

{ y>0 and y-x = 100} or (y<=0 and y-x = 90}

if (y>0) then

{y-x = 100}

x:= y-x

{x+y = 100}

{10-x+y = 100} = {y-x=90}

else x:=10-x

{x+y = 100}

}

{x+y = 100}

x:=x+y

{x=100}

y:=10+y

{x=100}