RECAP EX.

3 EXERCISES FOR EACH TYPE OF EXERCISE

EX1

16-12-15B

Alphabet:

Sa(x), C(x), Sh(x), FB(x), WB(x), B(x), onboard(x,y), incommand(x,y)

Axioms:

Forall x. WB(x) implies Sh(x) \\ ISA

Forall x. WB(x) implies FB(x) \\ ISA

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

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

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

Forall x. B(x) implies Sh(x) or FB(x) \\ complete

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

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

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

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

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

16-12-16B

Alphabet:

T(x), Al(x), Ar(x), SE(x), bonus(x,y), isContained(x,y), recordedBy(x,y)

Axiom:

Forall x. SE(x) implies Al(x) \\ ISA

Forall x,y. bonus(x,y) implies T(x) and SE(y) \\ typing

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

Forall x,y. bonus(x,y) implies isContained(x,y) \\ subset

Forall x,y. isContained(x,y) implies T(x) and A(y) \\ typing

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

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

Forall x,y. recordedBy(x,y) implies Al(x) and Ar(y) \\ typing

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

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

21-12-2017B

Alphabet:

C(x), BC(x), contract(x,y,z), S(x), P(x,) cost(x,y,z,k), provides(x,y), real(x)

Axioms:

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

Forall x,y,z. contract(x,y,z) implies C(x) and S(y) and P(z) \\ typing

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

Forall x,y,z,k cost(x,y,z,k) implies contract (x,y,z) and real(k) \\ typing

Forall x,y,z. contract(x,y,z) implies 1<=#{k|cost(x,y,z,k)}<=1 \\ multiplicity

Forall x,y. provides(x,y) and P(x) and S(y) \\ typing

Forall x. P(x) implies 1<=#{y|provides(x,y)}<=10 \\ multiplicity

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

EX2

16-12-15B

1. The instantiation is not complete because we have an ISA in UML diagram (T), so we need to apply this following procedure

Iold = 0, Inew = I

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}

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

I=Inew

Return I

I = (Obj^I, C^I, Sa^I, B^I, Sh^I, WB^I, FS^I, onboard^I, incommand^I)

I0:

C^I = {Alice,George}

Sa^I={Dustin,Lubber, Rusty}

B^I = {}

Sh^I={Constitution}

WB^I={Essex}

FS^I ={}

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

incommand^I = {(Alice,Constitution), (George,Essex)}

I1:

C^I = {Alice,George}

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

B^I = { }

Sh^I={Constitution, Essex }

WB^I={Essex}

FS^I ={ Essex }

I2:

C^I = {Alice,George}

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

B^I = { Constitution, Essex }

Sh^I={Constitution, Essex }

WB^I={Essex}

FS^I ={ Essex }

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

incommand^I = {(Alice,Constitution), (George,Essex)}

I3 = I2 so the instantiation is complete

Now we need to check if the instantiation is correct, in particular if I3|=T. This means that Interpretation I3 is correct if all axiom in T are evaluated true in the interpretation.

Each instance of C should be incommand with at most one instance of Sh, it is ok. Each instance of Sh should have at one and only one instance of C in the association incommand, it is ok. Each instance of WB is also instance of FB and Sh. Each instance of FB is also instance of B, the same for FB. These are true. Each instance of incommand mush e instance also of onboard. So the instantiation is correct

1. q(x)<- Sa(x) and Exists y. onboard(x,y) and FB(y)

q(x): {George}

1. q()<- Exists x. B(x) and Exists. y,y’,y’’ onboard(y,x) and Sa(y) and onboard(y’,x) and Sa(y’) and onboard(y’’,x) and S(y’’) and y noteq y’ and y’ noteq y’’ and y noteq y’’

q()<- : true

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

q(x) : {}

16-12-16B

1. The instantiation is not complete because we have an ISA so we need to complete it following procedure

Iold = 0, Inew = I

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}

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

I=Inew

Return I

I = (Obj^I, T^I, Al^I, SE^I, Ar^I, isContained^I, bonus^I,recordedBy^I)

I0:

T^I = {t1,t2,t3,t4,t5,t6}

Al^I ={a1,a2,a3}

SE^I = {s1,s2}

Ar^I = {bt,rs}

isContained^I = {(t1,a1),(t2,a1),(t3,a1),(t1,a2),(t4,a2),(t5,a2),(t5,a3)}

bonus^I = {(t5,s1),(t6,s2)}

recordedBy^I = {(a1,bt),(a2,bt),(a3,rs),(s1,rs),(s2,bt)}

I1:

T^I = {t1,t2,t3,t4,t5,t6}

Al^I ={a1,a2,a3, s1,s2}

SE^I = {s1,s2}

Ar^I = {bt,rs}

isContained^I = {(t1,a1),(t2,a1),(t3,a1),(t1,a2),(t4,a2),(t5,a2),(t5,a3)}

bonus^I = {(t5,s1),(t6,s2)}

recordedBy^I = {(a1,bt),(a2,bt),(a3,rs),(s1,rs),(s2,bt)}

I2 = I1

The instantiation is complete

Now we need to check if the instantiation is correct, so if I2|=T. This means that I2|=T if all axioms in T are evaluated true in the interpretation I2.

Each instance of SE is also instance of Al, ok. Each instance of Al is recordedBy one and only one instance of Ar, ok. Each instance of Ar have at least one instance of Al in the association recordedBy, ok. Each instance of bonus is also instance of isContained. Each instance of T is a bonus in at most one SE. Each instance of T isContained in At least one instance of Al,ok and finally, each instance of Al has at least one T.

The instantiation is correct.

1. q(x)<- T(x) and Exists y. Al(y) and isContained(x,y) and Exists z. SE(z) and isContained (x,z) and z noteq y and Exists k. recordedBy(y,k) and recordedBy(z,k)

q(x) : {t5}

1. q(x) <- Ar(x) and forall y. recordedBy(x,y) and Al(y) implies not SE(y)

q(x) : {}

1. q() <- Exists x. T(x) and forall y. SE(y) implies isContained(x,y)

q(): false

21-12-2017B

1. The instantiation is not complete because we have an ISA so we need to complete it following procedure

Iold = 0, Inew = I

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}

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

I=Inew

Return I

I = (Obj^I, C^I, BC^I, S^I, P^I, provides^I, contract^I, cost^I)

I0:

C^I = {c1,c2,c3,c4}

BC^I = {b1,b2,b3}

S^I = {s1,s2,s3}

P^I = {p1,p2}

provides^I = {(p1,s1),(p1,s2),(p1,s3),(p2,s2)}

contract^I /cost^I = {(c1,s1,p1,90),(c1,s2,p1,80),(c1,s3,p1,50),(b2,s1,p2,170),(b2,s2,p2,100)}

I1:

C^I = {c1,c2,c3,c4, b1,b2,b3}

BC^I = {b1,b2,b3}

S^I = {s1,s2,s3}

P^I = {p1,p2}

provides^I = {(p1,s1),(p1,s2),(p1,s3),(p2,s2)}

contract^I /cost^I = {(c1,s1,p1,90),(c1,s2,p1,80),(c1,s3,p1,50),(b2,s1,p2,170),(b2,s2,p2,100)}

I2 = I1

The instantiation is complete

Now we need to check if the instantiation is correct, so if I2|=T. This means that I2|=T if all axioms in T are evaluated true in the interpretation I2.

Each instance of BC is also instance of C, ok. Each instance of P provides at least 1 and at most 10 instance of S, ok. For each pairs (instance of C, instance of S) there is one and only one instance of P in contract. There is only one cost for each instance for each triple of contract. Ok

The instantiation is correct.

1. q() <- Forall x,y. P(x) and S(y) and (Exists z. contract(z,y,x)) implies not provides(x,y)

q(): false

1. q(x) <- C(x) and Exists y,z. contract(x,y,z) and S(y) implies provides (p2,y)

q(x):{c1,b2}

1. q(x)<- C(x) and forall z. P(z) implies (Exists y. S(y) and contract(x,y,z))

q(x):{}

EX3

16-12-15B

Model checking a closed mu formula phi over transition system T = <S,Ra,Pi> (S set of states, Ra set of transitions, Pi mapping function from a set of atomic 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 satisfied phi. To compute it we need to apply the labelling algorithm, that consists in labelling each state of T with predicates that are true in them. The extensions of least fixpoint and greatest fixpoint are computed using Tarski-Knaster approximates theorem.

vX.muY.((a and [next]X) or (<next>Y))

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

[X0] = {0,1,2,3,4}

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

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

[Y00] = {}

[Y01] = [(a and [next] X0) or <next> Y00] = ([a] intersect PreA(next,[X0])) union PreE(next,[Y00]) = ({0,1,4} intersect {0,1,2,3,4}) union {} = {0,1,4}

[Y02] = [(a and [next] X0) or <next> Y01] = ([a] intersect PreA(next,[X0])) union PreE(next,[Y01]) = ({0,1,4} intersect {0,1,2,3,4}) union {0,3,4} = {0,1,3,4}

[Y03] = [(a and [next] X0) or <next> Y02] = ([a] intersect PreA(next,[X0])) union PreE(next,[Y02]) = ({0,1,4} intersect {0,1,2,3,4}) union {0,3,4} = {0,1,3,4}

[Y04] = [Y03] - - > found LFP

[X1] = {0,1,3,4}

[X2] = [mu Y (a and [next] X1) or <next> Y]

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

[Y10] = {}

[Y11] = [(a and [next] X1) or <next> Y10] = ([a] intersect PreA(next,[X1])) union PreE(next,[Y10]) = ({0,1,4} intersect {3,4}) union {} = {4}

[Y12] = [(a and [next] X1) or <next> Y11] = ([a] intersect PreA(next,[X1])) union PreE(next,[Y11]) = ({0,1,4} intersect {3,4}) union {3} = {3,4}

[Y13] = [(a and [next] X1) or <next> Y12] = ([a] intersect PreA(next,[X1])) union PreE(next,[Y12]) = ({0,1,4} intersect {3,4}) union {0,3} = {0,3,4}

[Y14] = [(a and [next] X1) or <next> Y13] = ([a] intersect PreA(next,[X1])) union PreE(next,[Y13]) = ({0,1,4} intersect {3,4}) union {0,3} = {0,3,4}

[Y14] = [Y13] - - > found LFP

[X2] = {0,3,4}

[X3] = [mu Y (a and [next] X2) or <next> Y]

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

[Y20] = {}

[Y21] = [(a and [next] X1) or <next> Y20] = ([a] intersect PreA(next,[X1])) union PreE(next,[Y20]) = ({0,1,4} intersect {3,4}) union {} = {4}

[Y22] = [(a and [next] X1) or <next> Y21] = ([a] intersect PreA(next,[X1])) union PreE(next,[Y21]) = ({0,1,4} intersect {3,4}) union {3} = {3,4}

[Y23] = [(a and [next] X1) or <next> Y22] = ([a] intersect PreA(next,[X1])) union PreE(next,[Y22]) = ({0,1,4} intersect {3,4}) union {0,3} = {0,3,4}

[Y24] = [(a and [next] X1) or <next> Y23] = ([a] intersect PreA(next,[X1])) union PreE(next,[Y23]) = ({0,1,4} intersect {3,4}) union {0,3,4} = {0,3,4}

[Y24] = [Y23] - - > found LFP

[X3] = {0,3,4}

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

It is 0 in {0,3,4} ? YES, phi formula is satisfied by this transition system

Now we want to do model checking with CTL formula. Given 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 propositions, L labelling function) and a CTL formula phi we want to check that KM,s |=phi where s is state of S. With model checking we return a subset of S, where each element satisfied phi. To compute it we need to exploit the syntactic structure of CTL formula, in particular, we transform CTL sub formula into mu formula and apply labelling algorithm to find their extensions.

AG(EX(a implies EF not a))

alpha = EF not a = muX not a or <next>X

beta = a implies alpha

gamma = EX beta

delta = AG gamma

[alpha] = [muX not a or <next>X]

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

[X0] = {}

[X1] = [not a or <next>X0] = [not a] union PreE(next,[X0]) = {2,3} union {} = {2,3}

[X2] = [not a or <next>X1] = [not a] union PreE(next,[X1]) = {2,3} union {0,1,2} = {0,1,2,3}

[X3] = [not a or <next>X2] = [not a] union PreE(next,[X2]) = {2,3} union {0,1,2,4} = {0,1,2,3,4}

[X4] = [not a or <next>X3] = [not a] union PreE(next,[X3]) = {2,3} union {0,1,2,4} = {0,1,2,3,4}

[X4] = [X3] - - > found LFP

[alpha] = {0,1,2,3,4}

[beta] = [a implies alpha] = [not a or alpha] = [not a] union [alpha] = {2,3} union {0,1,2,3,4} = {0,1,2,3,4}

[gamma] = [EX beta] = [<next> beta] = PreE(next,[beta]) = {0,1,2,3,4}

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

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

[X0] = {0,1,2,3,4}

[X1] = [gamma and [next] X] = [gamma] intersect PreA(next,[X0]) = {0,1,2,3,4} intersect {0,1,2,3,4} = {0,1,2,3,4}

[X1] = [X0] - - ->found GFP

[delta] = {0,1,2,3,4}

It is 0 in [delta]? YES, CTL formula is true in this transition system

16-12-16B

Model checking a closed mu formula phi over transition system T = <S,Ra,Pi> (S set of states, Ra set of transitions, Pi mapping function from a set of atomic 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 satisfied phi. To compute it we need to apply the labelling algorithm, that consists in labelling each state of T with predicates that are true in them. The extensions of least fixpoint and greatest fixpoint are computed using Tarski-Knaster approximates theorem.

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

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

[X0] = {0,1,2,3,4}

[X1] = [muY.( not a and <next> X0) or ([next]Y)]

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

[Y00] = {}

[Y01] = [(not a and <next> X0) or [next]Y00] = ([not a] intersect PreE(next,[X0])) union PreA(next,[Y00]) = ({0,3,4} intersect {0,1,2,3,4}) union {} = {0,3,4}

[Y02] = [(not a and <next> X0) or [next]Y01] = ([not a] intersect PreE(next,[X0])) union PreA(next,[Y01]) = ({0,3,4} intersect {0,1,2,3,4}) union {3,4} = {0,3,4}

[Y03] = [(not a and <next> X0) or [next]Y02] = ([not a] intersect PreE(next,[X0])) union PreA(next,[Y02]) = ({0,3,4} intersect {0,1,2,3,4}) union {3,4} = {0,3,4}

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

[X1] = {0,3,4}

[X2] = [muY.( not a and <next> X1) or ([next]Y)]

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

[Y10] = {}

[Y11] = [(not a and <next> X1) or [next]Y10] = ([not a] intersect PreE(next,[X1])) union PreA(next,[Y10]) = ({0,3,4} intersect {0,2,3,4}) union {} = {0,3,4}

[Y12] = [(not a and <next> X1) or [next]Y11] = ([not a] intersect PreE(next,[X1])) union PreA(next,[Y11]) = ({0,3,4} intersect {0,2,3,4}) union {3,4} = {0,3,4}

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

[X2] = {0,3,4}

[X2] = [X1] - - > foung GFP

[phi] = {0,3,4}

It is 0 in [phi]? YES, phi is satisfied by this transition system

Now we want to do model checking with CTL formula. Given 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 propositions, L labelling function) and a CTL formula phi we want to check that KM,s |=phi where s is state of S. With model checking we return a subset of S, where each element satisfied phi. To compute it we need to exploit the syntactic structure of CTL formula, in particular, we transform CTL sub formula into mu formula and apply labelling algorithm to find their extensions.

EF(AG(a implies EXAX not a))

alpha = AX not a = [[next] not a]

beta = EX alpha

gamma = a implies beta

delta = AG gamma

sigma = EF delta

[alpha] = [[next] not a] = PreA(next,[not a]) = PreA(next,{1,2}) = {1}

[beta] = [EX alpha] = [<next> alpha] = PreE(next,[alpha]) = {0}

[gamma] = [a implies beta] = [not a or beta] = [not a] union [beta] = {1,2} union {0} = {0,1,2}

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

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

[X0] = {0,1,2,3,4}

[X1] = [gamma and [next] X0] = [gamma] intersect PreA(next,[X0]) = {0,1,2} intersect {0,1,2,3,4} = {0,1,2}

[X2] = [gamma and [next] X1] = [gamma] intersect PreA(next,[X1]) = {0,1,2} intersect {1,2,4} = {1,2}

[X3] = [gamma and [next] X2] = [gamma] intersect PreA(next,[X2]) = {0,1,2} intersect {1} = {1}

[X4] = [gamma and [next] X3] = [gamma] intersect PreA(next,[X3]) = {0,1,2} intersect {} = {}

[X5] = [gamma and [next] X4] = [gamma] intersect PreA(next,[X4]) = {0,1,2} intersect {} = {}

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

[delta] = {}

[sigma] = [muX delta or <next> X]

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

[X0] = {}

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

[X1] = [X0]

[sigma ]= {}

It is 0 in [sigma]? NO, CTL formula is false in this transition system

21-12-2017B

Model checking a closed mu formula phi over transition system T = <S,Ra,Pi> (S set of states, Ra set of transitions, Pi mapping function from a set of atomic 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 satisfied phi. To compute it we need to apply the labelling algorithm, that consists in labelling each state of T with predicates that are true in them. The extensions of least fixpoint and greatest fixpoint are computed using Tarski-Knaster approximates theorem.

νX.muY.((b and [next]X) or (a and〈next〉Y))

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

[X0] = {0,1,2,3,4}

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

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

[Y00] = {}

[Y01] = [(b and [next]X0) or (a and<next>Y00)] = ([b] intersect PreA(next,[X0])) union ([a] intersect PreE(next,[Y00])) = ({3,4} intersect {0,1,2,3,4}) union ({1,2} intersect {}= {3,4} union {} = {3,4}

[Y02] = [(b and [next]X0) or (a and<next>Y01)] = ([b] intersect PreA(next,[X0])) union ([a] intersect PreE(next,[Y01])) = ({3,4} intersect {0,1,2,3,4}) union ({1,2} intersect {0,3,4}= {3,4} union {} = {3,4}

[Y02] = [Y01] - - > found LFP

[X1] = {3,4}

[X2] = [muY.(b and [next]X1) or (a and<next>Y)

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

[Y10] = {}

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

[Y12] = [(b and [next]X1) or (a and<next>Y11)] = ([b] intersect PreA(next,[X1])) union ([a] intersect PreE(next,[Y11])) = ({3,4} intersect {3}) union ({1,2} intersect {0}= {3} union {} = {3}

[X2] = {3}

[X3] = [muY.(b and [next]X2) or (a and<next>Y)

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

[Y20] = {}

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

[Y21] = [Y20]

[X3] = {}

[X4] = [muY.(b and [next]X3) or (a and<next>Y)

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

[Y30] = {}

[Y31] = [(b and [next]X3) or (a and<next>Y30)] = ([b] intersect PreA(next,[X3])) union ([a] intersect PreE(next,[Y30])) = ({3,4} intersect {}) union ({1,2} intersect {}= {}

[Y31] = [Y30]

[X4] = {}

[X4] = [X3]

[phi] = {}

It is 0 in [phi]?NO, phi is not satisfied by this transition system

Now we want to do model checking with CTL formula. Given 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 propositions, L labelling function) and a CTL formula phi we want to check that KM,s |=phi where s is state of S. With model checking we return a subset of S, where each element satisfied phi. To compute it we need to exploit the syntactic structure of CTL formula, in particular, we transform CTL sub formula into mu formula and apply labelling algorithm to find their extensions.

EF (AG (a implies EX AX not a))

alpha = AX not a = [next] not a

beta = EX alpha

gamma = a implies beta

delta = AG gamma

sigma = EF delta

[alpha] = [[next] not a] = PreA(next,[not a]) = PreA(next,{0,3,4}) = {3,4}

[beta] = [EX alpha] = [<next> alpha] = PreE(next,[alpha]) = {0,3,4}

[gamma] = [ a implies beta] = [not a or beta] = [not a ] union [beta] = {0,3,4} union {0,3,4} = {0,3,4}

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

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

[X0] = {0,1,2,3,4}

[X1] = [gamma and [next] X0] = [gamma] intersect PreA(next,[X0]) = {0,3,4} intersect {0,1,2,3,4} = {0,3,4}

[X2] = [gamma and [next] X1] = [gamma] intersect PreA(next,[X1]) = {0,3,4} intersect {3,4} = {3,4}

[X3] = [gamma and [next] X2] = [gamma] intersect PreA(next,[X2]) = {0,3,4} intersect {3} = {3}

[X4] =[gamma and [next] X3] = [gamma] intersect PreA(next,[X3]) = {0,3,4} intersect {} = {}

[X5] =[gamma and [next] X4] = [gamma] intersect PreA(next,[X4]) = {0,3,4} intersect {} = {}

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

[delta] = {}

[sigma] = [EF delta] = [mu X delta or <next> X]

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

[X0] = {}

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

[X1] = [X0]

[sigma] = {}

It is 0 in [sigma]? NO, the CTL formula is false in this transition system

EX4 – BISIMULATION

16-12-15B

Two states of transition systems are bisimular if they have the same behaviour. Thi means that:

* Locally look undistinguishable
* Every action done in one state can also done in the other state

A binary relation R is bisimulation iff (s,t) in R implies that:

* s is final iff f is final
* For every action a:
  + If s action a s’ then exists t’. that t action a t’ in R
  + If t action a t’ then exists s’. that s action a s’ in R

A state s0 of transition system S is bisimular to a state t0 of transition system T iff there is a bisimulation relation between s0 and t0

Algorithm to compute bisumulation is:

1. R = SxT
2. R’ = R- {(s,t) such that not (s final in S noteq t final in T)
3. While (R noteq R’) do

R=R’

R’’ = R’ – {(s,t) such that exists s’a. s action a s’ and not exists t’. t action a

t’ and (s’,t’) in R}– {(s,t) such that exists t’a. t action a t’ and not exists s’. s

action a s’ and (s’,t’) in R}

So we compute it.

1. Assuming that all state of transition S are equal to the states of transition T

R0 = {(s1,t1),(s1,t2),(s2,t1),(s2,t2),(s3,t1),(s3,t2)}

1. Remove all pairs of states in which one is final and other not

R1 = {(s1,t1),(s2,t2),(s3,t1)}

1. Repeat iteratively this operation of removing pair in which one state so and action and the other one cannot copy

R2 = {(s1,t1),(s2,t2)}

* t1 can do a and go to t2 but s3 cannot do a, so I remove it.

R3 = {(s1,t1)}

* t2 can do c and go to t1 and s2 can do c and go to s3 but (s3,t1) is not in R2, so I remove (s2,t2).

R4 = {}

* t1 can do a and go to t2 and s1 can do a and go to s2 but (s2,t2) is not in R3 so I remove (s1,t1)

R5 = {}

R5 = R4

S and T are not bisimular because (s1,t1) is not in R5

16-12-16B

Two states of transition systems are bisimular if they have the same behaviour. This means that:

* locally look undistinguishable
* Every action done in one state can be done also in the other state

A binary relation R is a bisimulation iff (s,t) in R implies:

* s is final iff t is final
* For every action a
  + If s action a s’ then exists t’. t action a t’ in R
  + If t action a t’ then exists s’. s action a s’ in R

A state s0 of transition system S is bisimular to a state t0 of transition system T iff there exists a bisimulation between s0 and t0

Algorithm for bisimulation

1. R = SxT
2. R’ = R-{(s,t) such that not (s final in S noteq t final in T)
3. While (R noteq R’) do

R = R’

R’’ = R’ – {(s,t) such that exists s’,a. s action a s’ then not exists t’. t action a t’ and (s’,t’) in R}{(s,t) such that exists t’,a. t action a t’ then not exists s’. s action a s’ and (s’,t’) in R}

So we compute it

1. Assuming that all states transition system S are equal to all states of transition system T

R0 = {(t1,q1),(t1,q2),(t1,q3), (t2,q1),(t2,q2),(t2,q3)}

1. Remove all pairs in which one is final and the other one is not final

R1 = {(t1,q1),(t2,q2),(t2,q3)}

1. Iteratively repeat to remove all pairs in which one state can do an action an the other one cannot copy in the relation

R2 = {(t1,q1) }

* t2 can do b and go to t1 and q2 can do b and go to q3 but (t1,q3) is not in R1 so I remove (t2,q2)
* t2 can do b and go to t1 and q3 cannot do b so I remove it

R3 = {}

* t1 can do a and go to t2 and q1 can do a and go to q2 but (t2,q2) is not in R2 so I remove (t1,q1)

R4 = {}

R4 = R3

S and T are not bisimular because (t1,q1) is not in R4

4-02-16

Two states of transition systems are bisimular if they have the same behaviour. This means that:

* locally look undistinguishable
* Every action done in one state can be done also in the other state

A binary relation R is a bisimulation iff (s,t) in R implies:

* s is final iff t is final
* For every action a
  + If s action a s’ then exists t’. t action a t’ in R
  + If t action a t’ then exists s’. s action a s’ in R

A state s0 of transition system S is bisimular to a state t0 of transition system T iff there exists a bisimulation between s0 and t0

Algorithm for bisimulation

1. R = SxT
2. R’ = R-{(s,t) such that not (s final in S noteq t final in T)
3. While (R noteq R’) do

R = R’

R’’ = R’ – {(s,t) such that exists s’,a. s action a s’ then not exists t’. t action a t’ and (s’,t’) in R}{(s,t) such that exists t’,a. t action a t’ then not exists s’. s action a s’ and (s’,t’) in R}

So we compute it.

1. Assuming that all states of transition S are equal to states of transitions T

R0 = {(t1,q1),(t1,q2),(t1,q3),(t1,q4),(t1,q5), (t2,q1),(t2,q2),(t2,q3),(t2,q4),(t2,q5)}

1. Remove all pairs in which one is final and the other one is not final

R1 = {(t1,q1),(t1,q4),(t1,q5), (t2,q2),(t2,q3)}

1. Remove iteratively all pairs that one can do an action and the other one cannot copy it.

R2 = {(t1,q1), (t2,q2),(t2,q3)}

* t1 can do a and go to t2 and q4 can do a and go to q5 but (t2, q5) is not in R1 so I remove (t1,q4)
* t2 can do b and go to t1 and q5 cannot do b so I remove (t1,q5)

R3 = {(t1,q1),(t2,q2)}

* t2 can do b and go to t1 and q3 can do b and go to q4 but (t1,q4) is not in R2 so I remove (t2,q3)

R4 = {(t2,q2)}

* t1 can do a and go to t2 and q1 can do a and go to q3 but (t2,q3) is not in R3 so I remove (t1,q1)

R5 = {}

* t2 can do b and go to t1 and q2 can do b and go to q1 but (t1,q1) is not in R4 so I remove( t2,q2)

R6 = {}

R6 = R5

S and T are not bisimular because (t1,q1) is not in R6

EX5 – INCOMPLETE DATABASE

16-12-16B

q()←lives(x,y),incountry(y,z)

q():true

q(x,z)←lives(x,y),incountry(y,z)

q(x,z):(mary,JP)

we cannot return null values because we are not consider them as a output of conjunctive queries

4-02-16

q(x)←contains(x,y),genre(y,z)

q(x):{cd1,cd2,cd3,cd4}

q(x,z)←contains(x,y), genre(y,z)

q(x,z): {(cd2,blues), (cd3,rock)}

17-06-16

q(x)←Employee(x),Manages(x,y)

q(x) :{ Smith, Brown }

EX5 - CONJUNCTIVE QUEIRIES

16-12-17B

q1(a,b) :- e(a,y), e(x,y), e(x,b)

q2(a,b) :- e(a,y), e(x,y), e(x,z), e(w,z), e(w,b)

To check if q1 contained in q2 means that we want to verify that forall x. q1(x) implies q2(x) is valid. Validity means that forall I,alpha |= forall x. q1(x) implies q2(x) where I is interpretation and alpha assignment.

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

We need 3 steps to check if q1 is contained in q2

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

q1(c1,c2) = e(c1,y), e(x,y), e(x,c2)

q2(c1,c2) = e(c1,y), e(x,y), e(x,z), e(w,z), e(w,c2)

1. Build canonical database correponding of q1

Iq1 = {delta^I1, e^I1, c^I1}

delta^I1= {c1,c2,y,x}

e^I1 = {(c1,y),(x,y),(x,c2)}

c^I1 = {c1,c2}

1. Verify if q2 is true over database of q1. So means that we want to verify if there is an assignment for all free variable.

alpha(y) = y \\ because we have (c1,y) so it the only possibility is to consider y

alpha(x)=x \\ because we have (x,y) so it the only possibility is to consider x

alpha(z)=c2 \\ because we have (x,z) so it the only possibility is to consider c2

alpha(w)=x \\ because we have (w,z) so it the only possibility is to consider x

This is a satisfying assignment alpha

Iq2 ={delta^I2,e^I2,c^I2}

delta^I2= {y,x,c1,c2,z,w}

e^I2={(c1,y), (x,y), (x,z), (w,z), (w,c2)}

c^I2 = {c1,c2}

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

* h(c^I) = c^J
* (x,y) in e^I then (h(x),h(y)) in e^J

Now we need to guess a homomorphism that respect these two properties. Remember that there is a theorem that says that if we have an assignment alpha, a satisfying assignment, we can transform alpha in 2 homomorphism between canonical databases Iq1|=Iq2 iff h:Iq2 implies Iq1

The first property is satisfied because:

h(c1) in c^I2 = c1 in c^I1

h(c2) in c^I2 = c2 in c^I1

For the second property we have:

(c1,y) in e^I2 then (h(c1),h(y)) in e^I1? OK, because (c1,y) in e^I1

(x,y) in e^I2 then (h(x),h(y)) in e^I1? OK, because (x,y) in e^I1

(x,z) in e^I2 then (h(x),h(z)) in e^I1? OK, because (x,c2) in e^I1

(w,z) in e^I2 then (h(w),h(z)) in e^I1? OK, because (x,c2) in e^I1

(w,c2) in e^I2 then (h(w),h(c2)) in e^I1? OK, because (x,c2) in e^I1

25-01-17

q1(x) :- edge(x,y), edge(y,y), edge(x,z), edge(y,z), edge(z,y).

q2(x) :- edge(x,y), edge(y,z), edge(x,v), edge(v,z), edge(v,y).

We want to check if q1 is contained in q2, this means that we want to verify if forall x q1(x) implies q2(x) is valid. Validity meast that forall I,alpha |= forall x. q1(x) implies q2(x) where I interpretation and alpha assignment.

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

We need 3 steps:

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

q1(c) = edge(c,y), edge(y,y), edge(c,z), edge(y,z), edge(z,y)

q2(c) = edge(c,y), edge(y,z), edge(c,v), edge(v,z), edge(v,y)

1. Build canonical database of q1 because now q1 is Boolean, this because we substitute free variable have fresh variable

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

delta^I1 = {c,y,z}

e^I1 = {(c,y), (y,y), (c,z), (y,z), (z,y)}

c^I1 = c

1. Check if q2 is true over database of q1, so means to find assignment forall free variable

alpha(x) = x

alpha(y) =y

alpha(z) = z

alpha(v) =y

This is a satisfying assignment

Canonical database of q2:

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

delta^I2 = {c,y,z,v}

e^I2 = {(c,y), (y,z), (c,v), (v,z), (v,y)}

c^I2 = c

Now we are going 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:

* h(c^I) = c^J
* (x,y) in e^I then (h(x),h(y)) in e^J

We need to guess a homomorphism that respect these two properties. Remember that there is theorem that says that if we have an assignment alpha, a satisfying assignment, we can transform alpha in two databases such that Iq1|=Iq2 iff Iq2 implies Iq1.

The first property is satisfied, we need to check the second:

(c,y) in e^I2 then (h(c),h(y)) in e^I1? YES, (c,y) is in e^I1

(y,z) in e^I2 then (h(y),h(z)) in e^I1? YES, (y,z) is in e^I1

(c,v) in e^I2 then (h(c),h(v)) in e^I1? YES, (c,y) is in e^I1

(v,z) in e^I2 then (h(v),h(z)) in e^I1? YES, (y,z) is in e^I1

(v,y) in e^I2 then (h(v),h(y)) in e^I1? YES, (y,y) is in e^I1

09-07-15

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

q2() :- edge(x,y), edge(y,z), edge(z,x), edge(z,v), edge(v,w), edge(w,z)

Check if q1 is contained in q2, this consist in that forall x. q1(x) implies q2(x) is valid. Validity means that forall I,alpha |= forall x. q1(x) implies q2(x) where I interpretation and alpha assignment.

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

We need 3 steps:

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

In this case we have already Boolean queries so we don’t do this step

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 database of q1, so means that we want to find an assignment forall free variables

alpha(x) = r

alpha(y) = b

alpha(z) = g

alpha(v) = r

alpha(w) = b

Iq2 = (delta^I1, e^I1, c^I1)

delta^I2 ={x,y,z,v,w}

e^I2 = {(x,y), (y,z), (z,x), (z,v), (v,w), (w,z)}

c^I2 = {}

This is a satisfying assignment

Now we want to find a homomorphism. A homomorphism is a mapping between two interpretation, element 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

We guess homomorphism that respect these two properties. Remember that there is a theorem that says that if we have an assignment alpha, a satisfying assignment, we can transform alpha in two homomorphism between two canonical databases such that Iq1|=Iq2 iff Iq2 implies Iq1

The first property is satisfied, we need to check the second:

(x,y) in e^I2 then (h(x),h(y)) in e^I1? YES, because (r,b) is in e^I1

(y,z) in e^I2 then (h(y),h(z)) in e^I1? YES, because (b,g) is in e^I1

(z,x) in e^I2 then (h(z),h(x)) in e^I1? YES, because (g,r) is in e^I1

(z,v) in e^I2 then (h(z),h(v)) in e^I1? YES, because (g,r) is in e^I1

(v,w) in e^I2 then (h(v),h(w)) in e^I1? YES, because (r,b) is in e^I1

(w,z) in e^I2 then (h(w),h(z)) in e^I1? YES, because (b,g) is in e^I1

EX4 - HOARE TRIPLE

25-01-17

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

Assuming to have a candidate invariant I, we show that:

* P implies I
* {I and g} S {I} = {I and g} implies wp(S,I)
* {I and not g} implies {Q}

If the candidate respect all these operations it is called invariant

If we know the candidate these operations are automatic, otherwise it is difficult to manage with. The problem of checking if Hoare triple is true or not is undecidable because there is no sound or complete technique that do this or find all possible invariants. We have only sound technique to determine if that candidate is invariant or not.

Candidate: i+j =9

{i=0 AND j=9} while(i<10) do (i:= i+1; j=j-1) {j<0}

* P implies I

{i=0 and j=9} implies {i+j=9}

Ok because if i=0 and j=9, the sum is 9

* {I and not g} implies {Q}

{i+j=9 and i>=10} implies {j<0}

Ok because if we need to have i+j =9 and i>=10 j should be surely < 0

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

We need to compute wp(S,I)

{i+1+j-1=9} = {i+j=9} is wp(S,I)

i=i+1

{i+j-1=9}

j=j-1

{i+j=9}

{i+j=9 and i<10} implies {i+j=9}

YES because the hypothesis is also the thesis, this means that condition is satisfied

i+j = 9 is invariant

21-12-17B

Check if Hoare triple {P} while g o S {Q} is correct cannot be done automatically but we need inference rule.

Assuming to have a candidate invariant I, we show that:

* P implies I
* {I and g} S {I} = {I and g}implies wp(S,I)
* {I and not g} implies Q

If the candidate respect all these conditions that it is called invariant.

If we have the candidate these operation can be done in automatic way, otherwise It is difficult to manage with it. The problem of check if Hoare triple is true or not is undecidable because there is no sound or complete technique to do this or to find all possible invariants. The only thing we can do is to check if the candidate invariant is invariant or not.

Candidate: {x>=0 and y>=0 and x+y = 31}

{x= 31 and y= 0}while(x>0) do (x=x−1; y:= y+1){y= 31}

* P implies I

{x= 31 and y= 0} implies {x>=0 and y>=0 and x+y = 31}

Ok

* {I and not g} implies {Q}

{ x>=0 and y>=0 and x+y = 31 and x<=0} implies {y=31}

{x=0 and y>=0 and x+y=31} implies {y=31}

Ok

* Wp(S,I)

{ x>=1 and y>=-1 and x+y = 31} is wp

x=x-1

{ x>=0 and y+1>=0 and x+y = 30}

y=y+1

{ x>=0 and y>=0 and x+y = 31}

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

{x>=0 and y>=0 and x+y = 31 and x>0} implies{ x>=1 and y>=-1 and x+y = 31}

{x>0 and y>= 0 and x+y=31} implies {x>=1 and y>=-1 and x+y = 31}

{x>0 and y>=0} implies {x>=1 and y>=-1}

The condition is always satisfied

{ x>=0 and y>=0 and x+y = 31} is invariant

09-07-15

Check if Hoare triple {P} while g o S {Q} is correct cannot be done automatically but we need inference rule.

Assuming to have a candidate invariant I, we show that:

* P implies I
* {I and g} S {I} = {I and g}implies wp(S,I)
* {I and not g} implies Q

If the candidate respect all these conditions that it is called invariant.

If we have the candidate these operation can be done in automatic way, otherwise It is difficult to manage with it. The problem of check if Hoare triple is true or not is undecidable because there is no sound or complete technique to do this or to find all possible invariants. The only thing we can do is to check if the candidate invariant is invariant or not.

Candidate: i<=10

{i=0} while(i<10) do i:= i+1 {i=10}

* P implies I

{i=0} implies i<=10 ok

* {I and not g} implies {Q}

{i<=10 and i>=10} implies {i=10}

{i=10} implies {i=10} ok

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

{i<=9} is wp

i=i+1

{i<=10}

{i<=10 and i<10} implies {i<=9}

{i<10} implies {i<=9}

YES, the condition is satisfied

i<=10 is invariant

EX4 – WEAKEST PRECONDITION

11-06-15

wp(d,Q) = {s|forall s’ such that ((d,s) implies s’) implies s’|=Q}

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

{y>=48 and y+2+y = 100} = {y> =48 and y=49} = {y=49} is wp

x := y + 2;

[{x<50 and y<0 and x=50} or {x<50 and y>=0 and x=100}] or {x>=50 and x+y = 100} = [false or false] or {x>=50 and x+y = 100} = false or {x>=50 and x+y = 100} = {x>=50 and x+y = 100}

if (x < 50) then {

{y<0 and x=50} or {y>=0 and x=100}

if (y < 0) then

{x=50}

x := 2\*x;

{x=100}

else y := y\*y

{x=100}

}

{x+y = 100}

else x := x + y;

{x=100}

{x=100}

y := y\*y

{x=100}

18-02-14

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

All state s such that the execution of program d in state s give us s’ in which the post-condition Q is satisfied. The wp give us the minimum condition that we will achieve Q by executing d. Since there is no while instruction we can compute weakest precondition automatically starting from below and going backward.

{y>0 and y+1+y = 0} or {y = -100} = {y>0 and y = -1/2} or {y=-100} = false or {y=-100} = {y=-100} is wp

x := y + 1;

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

if (y > 0) then

{x+y = 0}

x := x + y

{x=0}

{y=-100}

else x := y + 100;

{x=0}

{x+y = y} = {x=0}

x := x + y;

{x=y}

22-01-15

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

All state s such that the execution of program d in state s give us s’ in which the post-condition Q is satisfied. The wp give us the minimum condition that we will achieve Q by executing d. Since there is no while instruction we can compute weakest precondition automatically starting from below and going backward.

{50+y<=50 and 50+y+y=0} or {50+y>50 and 50+y-y =0} = {y<=0 and y=-25} or {y>0 and 50 =0} = {y<=0 and y=-25} or false = {y<=0 and y=-25} is wp

x := 50 + y;

[{x>50 and y>0 and x-y=0} or {x>50 and y<=0 and x=0}] or {x<=50 and x+y =0} = [{x>50 and y>0 and x-y=0} or false] or {x<=50 and x+y =0} = {x>50 and y>0 and x-y=0} or {x<=50 and x+y =0}

if (x > 50) then {

{y>0 and x-y=0} or {y<=0 and x=0}

if (y > 0) then

{x-y = 0}

x := x - y;

{x=0}

{x=0}

else y := -y

{x=0}

}

{x+y=0}

else x := x + y;

{x=0}

{x=0}

y := y + 50

{x=0}

EX4 – EVALUATION SEMANTICS AND TRANSITION SEMANTICS

while (x<10) do x := x + 5

1. Evaluation semantics

Given a program d and a memory state s we want to compute the memory state s’ obtained by executing d in s.

(d,s) implies s’

S0:x=1

(while (x<10) do x:=x+5, S0) implies sf

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

(x=x+5,S0) implies s1 and (while (x<10) do x:=x+5, s1) implies sf

S1 : x = 6

(while (x<10) do x:=x+5, S0) implies sf

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

(x=x+5,S0) implies S1 and (while (x<10) do x:=x+5, S1) implies sf

(while (x<10) do x:=x+5, S1) implies sf

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

(x=x+5,S0) implies S1 and (x=x+5,S1) implies s2 and (while (x<10) do x:=x+5, s2) implies sf

S2 : x=11

(while (x<10) do x:=x+5, S1) implies sf

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

(x=x+5,S0) implies S1 and (x=x+5,S1) implies S2 and (while (x<10) do x:=x+5, S2) implies sf

(while (x<10) do x:=x+5, S2) implies sf

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

(x=x+5,S0) implies S1 and (x=x+5,S1) implies S2 and (while (x<10) do x:=x+5, S2) implies S3

S3: x=11

S3 = S2 stop

1. Transition semantics

Given a program d and a memory state s compute the memory state s’ and the program d’ that remains to be executed obtained by executing a single step of d in s.

S0:x=1

(while (x<10) do x:=x+5, S0) implies (d’,s’)

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

(x:=x+5,S0) implies (epsilon,s1)

S1:x=6

(while (x<10) do x:=x+5, S0) implies (d’,s’)

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

(x:=x+5,S0) implies (epsilon,S1)

d’ = (epsilon, while (x<10) do x:=x+5)

s’ = S1

(epsilon,while (x<10) do x:=x+5, S1) implies (d’’,s’’)

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

(epsilon,S1) is true

(while (x<10) do x:=x+5, S1) implies (d’’,s’’)

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

(x:=x+5,S1) implies (epsilon,s2)

S2: x=11

d’’ =(epsilon, while (x<10) do x:=x+5)

s’’ = S2

(epsilon,while (x<10) do x:=x+5, S2) implies (d’’’,s’’’)

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

(epsilon,S2) true

(while (x<10) do x:=x+5, S2) implies (d’’’,s’’’)

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

(while (x<10) do x:=x+5, S2) true

This while program is terminate because while condition is false, and considering ruler for terminating it is one of them, so ok. And for epsilon, also in that case it is an expression for terminantion.

EX5 – TABLEAUX

21-12-17B

We want to check validity of formula phi, this means that if forall interpretation I, I|=phi. Tableaux is a method for proving in a mechanical way if a formula is satisfiable or not. This means that to check the validity we must prove that if I|= phi forall possible interpretations and this is a NP-complete problem.

Check satisfiability of a closed formula means that exists an interpretation I such that I|= phi. So,we need to transform our problem in a problem of satisfiability by negating the formula and check if it is satisfiable or not.

A formula is satisfiable is all branches are open but in our case we need to have all branch closed because we star with not phi, so if all branches of out tableaux are closed our formula is logically valid.

Immagine che contiene testo

Descrizione generata automaticamente

The formula is valid

18-01-18

­We want to check if formula phi is valid, so if forall interpretation I, I|=phi. Tableaux is a method for proving, in a mechanical way, if a formula is satisfiable or not. This means that we check validity if I|=phi forall interpretation I and this is a NP-complete problem.

Check satisfiability of a closed formula means that exists an interpretation such that I|=phi. So, we are going to transform our problem of validity in a problem of satisfiability by negating the formula phi, so it became not phi, and check if it is satisfiable or not.

A formula is satisfiable if its tableaux contains at least one open branch but in our case the formula in tableaux start with not phi so if we have all branches that are closed this means that our formula is logically valid.

Immagine che contiene testo

Descrizione generata automaticamente

­­17-06-19

We want to check the validity of a formula phi, this means that if forall interpretation I, I|=phi. Tableaux is a method for proving, in a mechanical way, if a formula is satisfiable or not. In this way to check validity we must prove that if I|=phi for all possible interpretation, and this is a NP-complete problem.

Check satisfiability a closed formula phi means that exists an interpretation such that I|=phi, so we are going to transform our problem of validity in a problem of satisfiability by negating formula phi and check if it is satisfiable.

A formula is satisfiable if there is at least one open branch in the tableaux. In our case to check validity we mush have all branches closed in the tableaux because we have not phi, and so in this case we can prove that formula is logically valid

­Immagine che contiene testo

Descrizione generata automaticamente

EX6 – LTL MODEL CHECKING

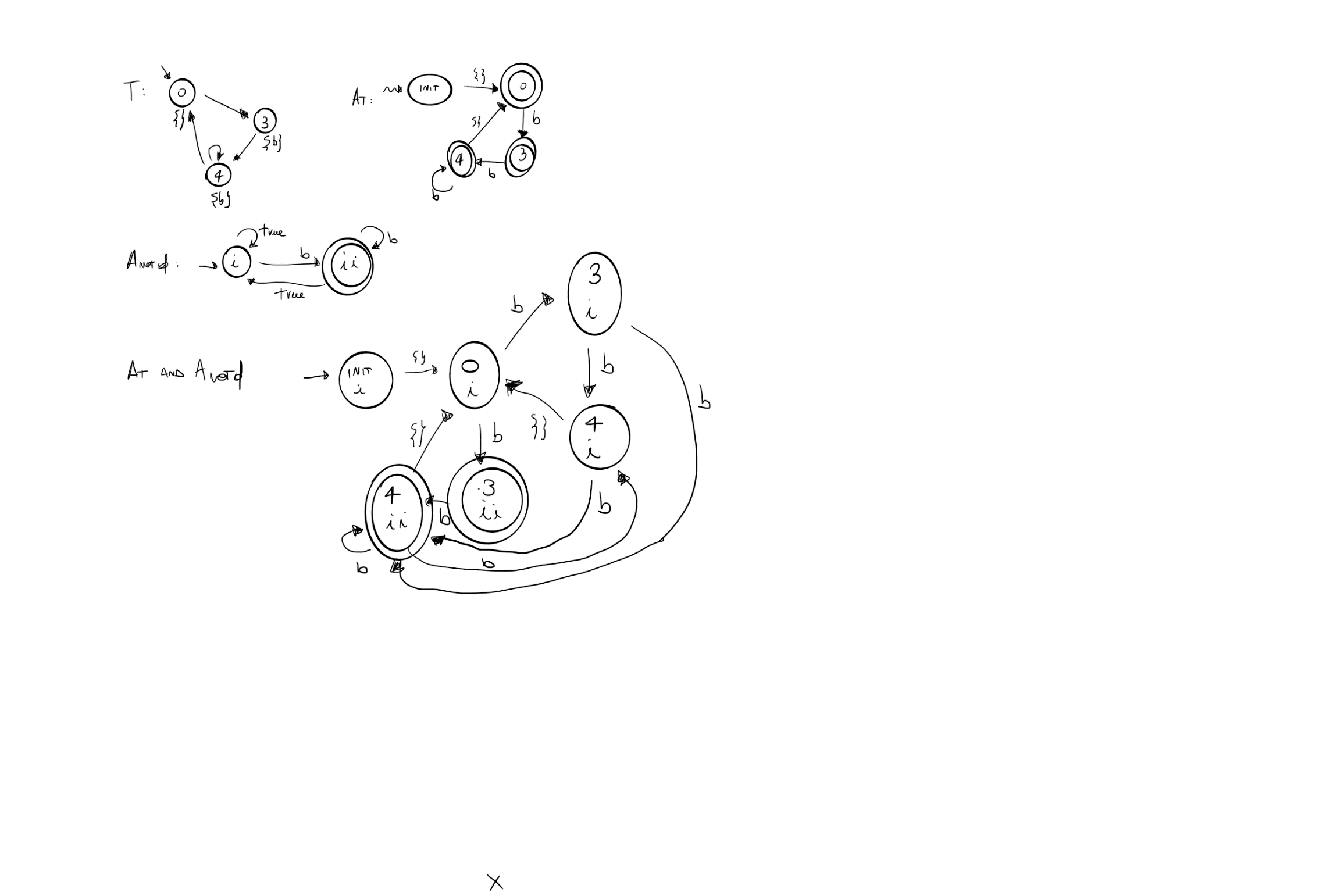
21-12-17B

In LTL we cannot transform formula into mu calculus as CTL. We cannot even exploit in NFA and DFA because they work on finite states, while LTL is evaluate on infinite languages and has infinite traces. What we can do to translate LTL formula in NBA because with NBA we can go to a finite state infinite often.

To model check LTL formula phi over a transition system we need to prove L(T) subseteq L(phi) iff L(T) intersection L(not phi) = {}. We transform T in automata At and notphi in automata Anotphi, so we obtain that we want to check the nonemptyness so L(At and Anotphi) = {}. In this way we can prove that phi is satisfied by T, so we need to check if the new automata accept at least a word (so if there is a trace that start from initial state and go to final state infinite often).

To prove non emptiness we need to solve that if At and Anotphi,init |= sigma, where sigma is:

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



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

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

[X0] = {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}

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

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

[Y00] = {}

[Y01] = [(final and <next>X0) or <next>Y00] = ([final] intersect PreE(next,[X0])) union PreE(next,[Y00]) = ({(3,ii),(4,ii)} intersect {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}) union {} = {(3,ii),(4,ii)}

[Y02] =[(final and <next>X0) or <next>Y01] = ([final] intersect PreE(next,[X0])) union PreE(next,[Y01]) = ({(3,ii),(4,ii)} intersect {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}) union {(0,i),(3,i),(4,i),(3,ii),(4,ii)} = {(0,i),(3,i),(4,i),(3,ii),(4,ii)}

[Y03] =[(final and <next>X0) or <next>Y02] = ([final] intersect PreE(next,[X0])) union PreE(next,[Y02]) = ({(3,ii),(4,ii)} intersect {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}) union {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)} = {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}

[Y04] =[(final and <next>X0) or <next>Y03] = ([final] intersect PreE(next,[X0])) union PreE(next,[Y03]) = ({(3,ii),(4,ii)} intersect {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}) union {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)} = {(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}

[Y04] = [Y03] - - > found LFP

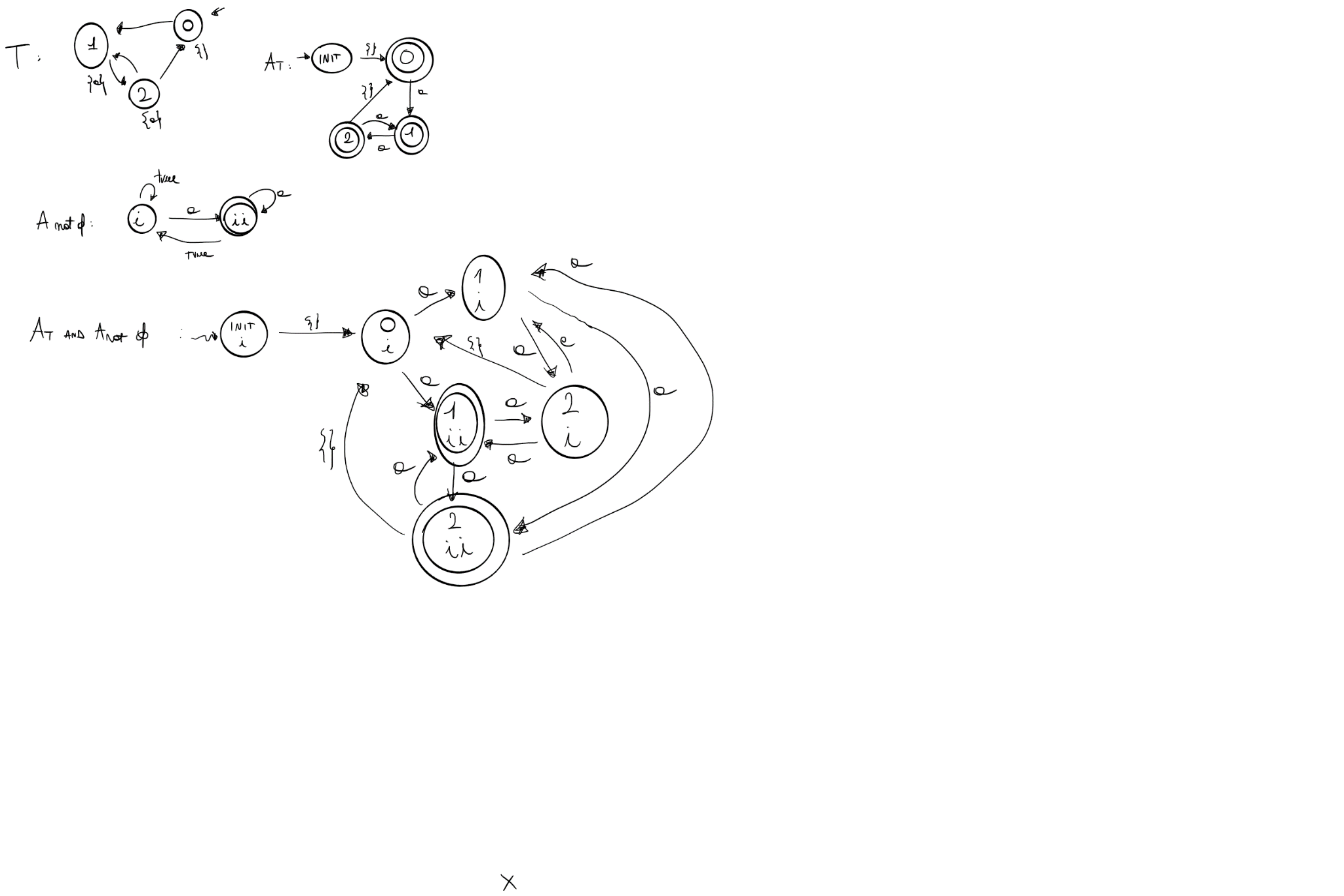
[X1] ={(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}

[X1] = [X0] - - > found GFP

It is (init,i) in ={(init,i),(0,i),(3,i),(4,i),(3,ii),(4,ii)}? YES, to LTL formula is satisfied by this transition system

17-06-19

In LTL we cannot translate formula into mu calculus as CTL. We cannot even exploit NFA or DFA because they work with finite state, while LTL is evaluated on infinite states and has infinite traces. What we can do is to use NBA because NBA accept to go to finite state infinite often. To do model checking with LTL formula phi over a transition system T we need to prove that L(T) subseteq L(phi) iff L(T) intersect L(notphi) = {}. So we translate transition system T in an automata At and also notphi in automata Anotphi and we check the nonemptyness L(At and Anotphi) = {}. In this way, we verify if the LTL formula phi is satisfied by transition system T, in particular if At and Anotphi, init |= sigma= vXmuY (final and <next>X) or <next>Y.



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

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

[X0] = {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}

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

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

[Y00] = {}

[Y01] = [(final and <next>X0) or <next>Y00] = ([final] intersect PreE(next,[X0])) union PreE(next,[Y00]) = ({(1,ii),(2,ii)} intersect {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}) union {} = {(1,ii),(2,ii)}

[Y02] = [(final and <next>X0) or <next>Y01] = ([final] intersect PreE(next,[X0])) union PreE(next,[Y01]) = ({(1,ii),(2,ii)} intersect {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}) union {(0,i),(1,i),(2,i),(1,ii),(2,ii)} = {(0,i),(1,i),(2,i),(1,ii),(2,ii)}

[Y03] = [(final and <next>X0) or <next>Y02] = ([final] intersect PreE(next,[X0])) union PreE(next,[Y02]) = ({(1,ii),(2,ii)} intersect {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}) union {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)} = {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}

[Y04] = [(final and <next>X0) or <next>Y03] = ([final] intersect PreE(next,[X0])) union PreE(next,[Y03]) = ({(1,ii),(2,ii)} intersect {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}) union {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)} = {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}

[Y04] = [Y03] - - > found LFP

[X1] = {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}

[X1] = [X2] - - > found GFP

It is (init,i) in {(init,i),(0,i),(1,i),(2,i),(1,ii),(2,ii)}? YES, so the LTL formula is satisfied by T.