Ex. FOL – Verify if formula phi is valid

Check if the formula above, phi, is valid means to check if for all interpretations I, I |= phi. But Tableaux method is a method for proving, in a mechanical way, if a given formula is satisfiable or not. This means that to check validity we must prove if I |= phi for all possible interpretations and this is a NP-complete problem.

Check satisfiability means to find an interpretation I that satisfies a closed formula phi, i.e. check if there is a I such that I |= phi. So, we need to transform this problem in a satisfiable problem, and we do this by negating the formula phi and by check if it unsatisfiable.

If the formula is satisfiable then there exists an open branch in the tableaux. To test formula for validity we start with not phi. If in the tableaux of it there are only closed branches, means that phi is logically valid otherwise not. A branch is closed if it contains two opposite literals since every branch is attempted to find an interpretation I that satisfies phi.

Ex. CONJUNCTIVE QUERIES – Contained and Homomorphism

First, we want to check if q1(x) subseteq q2(x) so this means that we want to check if the q1(x) implies q2(x) is valid. Valid means that Forall I,alpha |= Forall x q1(x) implies q2(x).

We know that in FOL validity is undecidable but with conjunctive queries we can make it satisfiable because we can transform queries in databases.

We need 3 steps to verify it q1 is contained in q2

1) Freeze queries, i.e. substitute free variables with fresh constant
2) Build database (Iq1) for corresponding to q1
3) Check that q1() subseteq q2() iff Iq1 |= Iq2. So, check if q2 is true over database of q1, verify if there is an assignment for all free variables

After that we need to check the homomorphism. Homomorphism is a mapping between two interpretations, between the elements of 2 domains h= delta^I -> delta^J such that

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

Where e is the predicate

Find a homomorphism is to guess a mapping and show that it respects these two properties. But remember theorem that if you have an assignment alpha, which is a satisfying assignment, you can transform alpha in 2 homomorphism between two canonical interpretations Iq1 |= q2 iff h: Iq2 -> Iq1

Ex. CTL – Model Checking

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

Ex. MU CALCULUS

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

Ex. LTL – Model Checking

In LTL we cannot do model checking by translating formulas in mu calculus as for CTL. We cannot even exploit in NFA or DFA properties because they are finite and work on finite states, while LTL is evaluated on infinite languages and has infinite traces. We can instead translate LTL formulas into NBA like NFA, with difference that while NFA accept when they go into finite state, NBA accept only if they go to a finite state infinite often. To model check the formula over transition system T, we must prove L(T) subseteq L(phi) iff L(T) intersection L(not phi) = {}. If we translate T into At and not phi into Anotphi, we can check nonemptiness by showing that L(At and Anotphi) = {}. This is a great solution because using NBA, intersect is easy; instead, the complement is very difficult. In this way we can prove that phi is satisfied over T, so check is this new automata accept at leas a word (a trace of transition system that satisfies not phi such that from the initial state goes to a final state and loops there). To prove nonemptiness we need to check if At and Anotphi, init |= sigma := vXmuY (final and <next> X) or <next> Y

Ex. TRANSITION SYSTEMS – Bisimulation

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

- Locally two states look undistinguishable
- Every action that can be done on one of them can also be done on the other

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

- s is final iff t is final
- For all action a
    - If s -> s' then exists t'. t->t' and (s',t') in R
    - If t->t' then exists s'. s-> s' and (s',t') 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 initial states s0 and t0.

The algorithm used to compute bisimulation is:

1) R = SxT
2) R' = R – {(s,t) such that not(s in Final in S eq t in Final in T)}
3) While (R noteq R')

    R = R'

    R'' = R' – ({(s,t) such that exist s',a. s action a(->) s' and not exists

        t'. t action a t' and (s',t') in R'}

        {(s,t) such that exist t', a. t action a t' and not exists s'.

        s action a s' and (s',t') in R'})

In a few words:

1) First approximation we assume that every states of S is equal to every state of T
2) Remove the states that are final in one or not final in the other
3) Now enter in the loop and we iteratively remove those pairs where one can do action and other cannot copy remaining in the relation. This algorithm is based on computing iteratively fixpoint approximates for the greatest fixpoint (GFP). When Rn=R_n+1 it means that GFP has been reached.

Ex. HOARE – weakest precondition

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

All the states s such that execution the program d in that 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 we do not have "while" in this program we can compute wp automatically, starting from below and going backward.

{P}S{Q} = P -> wp(S,Q) = P => W

If the wp is false it means that there is no stat from P can produce Q executing S so that P not => wp

Ex. HOARE – triple and while condition

Check whether a Hoare triple {P} while g do S {Q} is correct cannot be done automatically, we have inference rule.

We assume to have a candidate invariant I, and we need to show

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

If I satisfy the 3 premises above it is called invariant.

If the invariant I is given these operations are automatic, otherwise it could be difficult to manage with. The problem of checking if a triple with while is true or not is undecidable, there is not a sound and complete technique to do this or to generate all possible invariants for the problem. We have only a sound technique to check whether a candidate is an invariant or not.

If it is not the case, we cannot say that the triple is false but at the same time we are not sure that an invariant exists.

Ex. UML – instantiations

To complete instantiations of UML diagram T, we use this procedure:

I_old = 0; I_new= I

While (I_new and I_old are different) do

    I_old = I_new

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

        For each (a in A^{I_new}) do

            B^{I_new} = B^{I_new} union {a}

    Similar for each subset constraints forall x,y. P(x,y) -> R(x,y) in T

I = I_new

Return I

In a few words, we do this procedure for ISA and subset constraints and it return the resulting completed instantiation.

To check if the instantiation I is correct we need to verify if I |= T, this means that checking if if all axioms in T evaluate to true in the interpretation I.