Homework 4

Morgan Ciliv

March 2017

1 Resolution

First, we convert all of the clauses to disjunctive form (Note: sentence 3 is converted to sentences 3 and 4 below).

1. $A \vee B$
2. $\neg B \vee C$
3. $\neg C \vee D$
4. $\neg A \vee D$

Sentences numbered 2, 3, and 4 above are horn clauses, but number 1, $A \vee B$, is not because it has more than one positive literal and is a disjunction. Because of these facts, number 1 is by definition not a horn clause. Since all of the clauses must be horn clauses in order to use modus ponens, we cannot derive D using generalized modus ponens by also noting that to derive the truth-value of D, each of the sentences must be used.

Below I show how D can be derived using resolution.

First I negate D. Since D is negated and each of the sentences must be true, $\neg C$ must be true and $\neg D$ must be true due to sentences 3 and 4. This makes literals A false in sentence 1 and B false in sentence 2. Therefore, B must be true in sentence 1 and $\neg B$ must be true in sentence 2 which brings us to a contradiction because it is impossible to have a truth-value assignment in classical logic such that B and $\neg B$ are both true.

2 Unification

Note: Most general unifier is returned in each of the evaluations of the unification algorithm.

2.1 knows(x, y) knows(mother(z), y)

1. $Unify(knows(x, y), knows(mother(z), y), \{\})$
   (a) Both Terms: $Unify(\{x, y\}, \{mother(z), y\}, Unify(knows, knows, \{\}))$
      i. We have to do the internal call first: $Unify(knows, knows, \{\})$
      ii. knows=knows, returning u (*i.e.*, $\{\}$)
   (b) So this outer call is now: $Unify(\{x, y\}, \{mother(z), y\}, Unify(knows, knows, \{\}))$
   (c) Both List: $Unify(y, y, Unify(x, mother(z), \{\})$
      i. We have to do the internal call first: $Unify(x, mother(z),)$
      ii. x is a variable, so we call $Unify - Var(x, mother(z), \{\})$
      iii. Which returns $\{\} + \{x/mother(z)\}$, *i.e.*, $\{x/mother(z)\}$
   (d) So this call is really: $Unify(y, y, \{x/mother(z)\})$
   (e) y is variable, so we call $Unify - Var(y, y, \{x/mother(z)\})$
   (f) y = y, returning {x/mother(z)}, which is propagated out and returned.

2.2 knows(x, y) knows(mother(x), x)

1. $Unify(knows(x, y), knows(mother(x), y), \{\})$
   (a) Both Terms: $Unify(\{x, y\}, \{mother(x), y\}, Unify(knows, knows, \{\}))$
      i. We have to do the internal call first: $Unify(knows, knows, \{\})$
      ii. knows=knows, returning u (*i.e.*, $\{\}$)
   (b) So this outer call is now: $Unify(\{x, y\}, \{mother(z), y\}, Unify(knows, knows, \{\}))$
   (c) Both List: $Unify(y, y, Unify(x, mother(x), \{\})$
      i. We have to do the internal call first: $Unify(x, mother(x),)$
      ii. x is a variable, so we call $Unify - Var(x, mother(x), \{\})$
      iii. Mother(x) is a structure containing x, so to prevent cycles, we return FAIL, which is propagated out and returned.

2.3 knows(x, y) knows(mother(z), w)

1. $Unify(knows(x, y), knows(mother(z), w), \{\})$
   (a) Both Terms: $Unify(\{x, y\}, \{mother(z), w\}, Unify(knows, knows, \{\}))$
      i. We have to do the internal call first: $Unify(knows, knows, \{\})$
      ii. knows=knows, returning u (*i.e.*, $\{\}$)
   (b) So this outer call is now: $Unify(\{x, y\}, \{mother(z), w\}, Unify(knows, knows, \{\}))$
   (c) Both List: $Unify(y, w, Unify(x, mother(z), \{\})$
      i. We have to do the internal call first: $Unify(x, mother(z),)$
      ii. x is a variable, so we call $Unify - Var(x, mother(z), \{\})$

iii. Which returns $\{\} + \{x/mother(z)\}, i.e., \{x/mother(z)\}$

(d) So this call is really: $Unify(y, w, \{x/mother(z)\})$

(e) y is variable, so we call $Unify - Var(y, w, \{x/mother(z)\})$

(f) returning {x/mother(z), y/w}, which is propagated out and returned.

2.4 knows(x, x) knows(mother(y), y)

1. $Unify(knows(x, x), knows(mother(y), y), \{\})$

   (a) Both Terms: $Unify(\{x, x\}, \{mother(y), y\}, Unify(knows, knows, \{\}))$

      i. We have to do the internal call first: $Unify(knows, knows, \{\})$

      ii. knows=knows, returning u $(i.e., \{\})$

   (b) So this outer call is now: $Unify(\{x, x\}, \{mother(y), y\}, Unify(knows, knows, \{\}))$

   (c) Both List: $Unify(x, y, Unify(x, mother(y), \{\})$

      i. We have to do the internal call first: $Unify(x, mother(y),)$

      ii. x is a variable, so we call $Unify - Var(x, mother(y), \{\})$

      iii. Which returns $\{\} + \{x/mother(y)\}, i.e., \{x/mother(y)\}$

   (d) So this call is really: $Unify(x, y, \{x/mother(y)\})$

   (e) y is variable, so we call $Unify - Var(x, y, \{x/mother(y)\})$

   (f) Var is unified with some val in u; therefore, return Unify(mother(y), y, {x/mother(y)})

      i. Therefore, the new call is evaluated: Unify(mother(y), y, {x/mother(y)})

      ii. y is a variable, so we call $Unify - Var(y, mother(y), \{\})$

      iii. Mother(y) is a structure containing y, so to prevent cycles, we return FAIL, which is propagated out and returned.

3 Looking Back - Spare Tire Problem

3.1 Uninformed Search

I would cast this as a search problem by

- The states could the list of information about the places of the objects in the world. Instead of the actions being stored here, the actions would be in the successor function below which determines which of these states the world becomes.
- My successor function would be to choose from all possible actions and to check to whether that action meets all its preconditions. The successor state is then determined by the changes that the actions make on the state of the world.

- My goal test would be a check whether the current state is equivalent to the fact that the spare tire is removed from the trunk , the flat tire is removed from the axel, and the spare tire is put on the axel.
- PDDL is superior because it doesn't consume as much space. In uninformed search, there are numerous states which will probably have to be searched and each of those states contains information on the state of the world. This isn't as good as PDDL where the information is kept in "one place" and altered and evaluated to determine a solution.

3.2 Constraint Satisfaction

I would cast this as a constraint satisfaction problem by

- The variables would be the list of actions to take such that a corresponding object will be assigned to it.
- The values would be the corresponding assignments of actions and objects to those actions.
- The constraints would be the preconditions for each of the actions. The constraints/preconditions must be met in order to proceed.
- The advantage of PDDL would be that you could analyze the problem in a depth first manner instead of scanning for the one with the most constraining variable and least constraining value. PDDL aligns with the problem more. Overall, I tried making the problem stated in a similar way, yet constraint satisfaction doesn't align with this problem as well as the optimization goal of constraint satisfaction doesn't apply as well to this type of problem, so there is extra calculation that might not be the most well suited for this problem.

4 Data Analysis

4.1 PDDL Description of the Problem

Note: Ana = analyze vis = visualize

Actions:	ssh(newComp)	exp(expName)	send(file, comp)	ana(exp.data)	vis(exp.csv)
Preconds:	None	At(Workstat)	Exists(file)	Exists(exp.data) At(SuperComp)	Exists(exp.csv) At(Workstat)
Delete List	At(prevComp)				
Add List	At(newComp)	Exists(exp.data)		Exists(exp.csv)	

4.2 Solution that connects initial state to goal state with a sequence of legal actions

[ssh(Workstat),
exp(experiment_4),
ssh(AMU workstation),
send(experiment_4.data, supercomp),
ssh(supercomp),
analyze(experiment_4.data),
send(experiment_4.csv, workstat),
ssh(workstat),
visualize(experiment_4.csv)]