Künstliche Intelligenz 1 – WS 2017/18

Michael Kohlhase
Informatik, FAU Erlangen-Nürnberg
For Course Purposes Only

February 6, 2018

Contents

1 Assignment 1 (Rational Agents) – Given Oct. 26., Due Nov. 03.

Problem 1.1 Explain the difference between agent function and agent program. How many agent programs can there be for a given agent function? 30pt

Solution: The function specifies the input-output relation (outside view). The program implements the function (inside view).

The function takes the full sequence of percepts as arguments. The program uses the internal state to avoid that.

There are either none or infinitely many programs for a function.

Problem 1.2 For each of the following agents, develop a PEAS description of the task environment. 40pt

1. Robot soccer player
2. Internet book-shop agent (that is: an agent for book shops that stocks up on books depending on demand)
3. Autonomous Mars rover
4. Mathematical theorem prover
5. First-person shooter (Counterstrike, Unreal Tournament etc.)

Characterize the environments of these agents according to the properties discussed in the lecture. Where not "obvious", justify your choice with a short sentence.

Choose suitable designs for the agents.

Problem 1.3 Suggest agent programs for the following indeterministic variants of the vacuum cleaner example. 30pt

1. In one out of four cases the suck action fails and deposits dirt on the floor. And the dirt sensor is wrong in one out of ten cases.
2. In each time step a square has a 10 % chance of becoming dirty again.

2 Assignment 2 (Prolog) – Given Nov. 02., Due Nov. 10.

This exercise sheet is supposed to get you started with Prolog. The SWI ProLog interpreter can be downloaded from `http://www.swi-prolog.org/`. The SWI manual is available at `http://www.swi-prolog.org/pldoc/`.

The lecture notes explain the basics of ProLog, in particular handling lists. If there are any commands missing that are strictly necessary to solve these exercises, use the course forum to let us know.

Problem 2.1 (Basic `ProLog` Functions)

Your task is to implement the functions listed below in `ProLog`. Note that many of them are built-in, but we ask you create your own functions. 30pt

- a function removing multiple occurrances of elements in a list

```
?- removeDuplicates([1,1,1,1,2,2,3,4,1,2,7],A).
A = [1, 2, 3, 4, 7].
```

- a function reversing a list

```
?- myReverse([1,2,3,4,2,5],R).
R = [5, 2, 4, 3, 2, 1].
```

- a function outputting all the permutations of the elements in a list

```
?- myPermutations([1,2,3],Z).
Z = [1, 2, 3] ;
Z = [2, 1, 3] ;
Z = [2, 3, 1] ;
Z = [1, 3, 2] ;
Z = [3, 1, 2] ;
Z = [3, 2, 1].
```

Feel free to implement any helper functions.

Solution:

```
% remove duplicates function
delete(_,[],[]).
delete(X,[X|T],R) :- delete(X,T,R).
delete(X,[H|T],[H|R]) :- not(X=H), delete(X,T,R).
removeDuplicates([],[]).
removeDuplicates([H|T],[H|R]) :- delete(H,T,S), removeDuplicates(S,R).

% reverse function
preReverse([],X,X).
preReverse([X|Y],Z,W) :- preReverse(Y,[X|Z],W).
myReverse(A,R) :- preReverse(A,[],R).

% permute function
takeout(X,[X|T],T).
takeout(X,[H|T1],[H|T2]) :- takeout(X,T1,T2).
```

```
myPermutations([],[]).
myPermutations([X|Y],Z) :- myPermutations(Y,W), takeout(X,Z,W).
```

Problem 2.2 Program a predicate for addition, multiplication and exponentiation in unary representation. The number 3 in unary representation is the `ProLog` term `s(s(s(o)))`, i.e. application of the arbitrary function `s` to an arbitrary value `o` iterated three times. Note that `ProLog` does not allow you to program (binary) functions,so you must come up with a three-place predicate. 30pt

You should use `add(?X,?Y,?Z)` to mean $X+Y=Z$ and program the recursive equations $X+0=X$ (base case) and $X+f(Y)=f(X+Y)$.

Solution:

```
uadd(X,o,X).
uadd(X,s(Y),s(Z)) :- add(X,Y,Z).
```

The problems for multiplication and exponentiation are quite similar

```
umult(_,o,o).
umult(X,s(Y),Z) :- umult(X,Y,W), uadd(X,W,Z).
uexpt(_,o,s(o)).
uexpt(X,s(Y),Z) :- uexpt(X,Y,W), umult(X,W,Z).
```

Problem 2.3 (Binary Tree)

40pt

1. Think of a way of representing binary trees in `ProLog`
2. Write a `ProLog` function `construct` that constructs a binary search tree out of a list of (distinct) numbers.
3. Write a `ProLog` function `count_leaves` that given a binary tree returns the number of leaves. Use it to test how cost efficient your previous function is (in terms of the structure that you obtain). What can you observe for larger lists of numbers?
4. Write a `ProLog` function `symmetric` that checks whether a binary tree is symmetrical.

Solution:

```
add(X,nil,tree(X,nil,nil)).
add(X,tree(Root,L,R),tree(Root,L1,R)) :- X @< Root, add(X,L,L1).
add(X,tree(Root,L,R),tree(Root,L,R1)) :- X @> Root, add(X,R,R1).

construct(L,T) :- construct(L,T,nil).

construct([],T,T).
construct([N|Ns],T,T0) :- add(N,T0,T1), construct(Ns,T,T1).

count_leaves(nil,0).
count_leaves(tree(_,nil,nil),1) :- !.
count_leaves(tree(_,L,R),N) :- count_leaves(L,NL), count_leaves(R,NR), N is NL+NR.

symmetric(nil).
symmetric(t(_,L,R)) :- mirror(L,R).
```

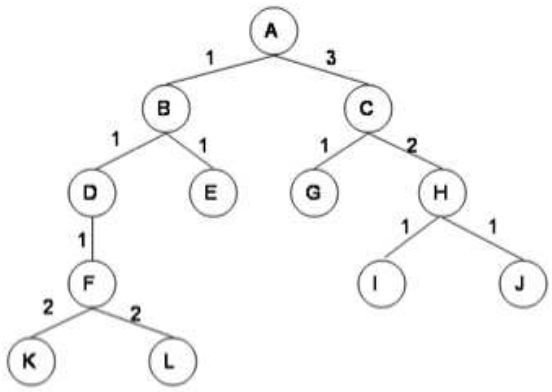
```
mirror(nil,nil).
mirror(t(_,L1,R1),t(_,L2,R2)) :- mirror(L1,R2), mirror(R1,L2).

%A few tests
test1(X):- construct([5,2,4,1,3],Y), count_leaves(Y,X).
% X=2
test2(X):- construct([6,10,5,2,9,4,8,1,3,7],Y), count_leaves(Y,X).
% X=3
symmetric(tree(1,tree(2,nil,nil),tree(2,nil,nil))).
% true.
symmetric(tree(1,tree(3,nil,nil),tree(2,nil,nil))).
% false.
```

By looking at the number of leaves we can have an idea how balanced the binary tree is. The bigger the number of leaves the more balanced the tree is, therefor the more efficient is you representation.

(note: maybe this question is a bit out of scope so don't cut too many points.)

3 Assignment 3 (Tree Search) – Given Nov. 09., Due Nov. 17.

Problem 3.1 (Search Strategy Comparison on Tree Search)
Consider the tree shown below. The numbers on the arcs are the arc lengths. 20pt



Assume that the nodes are expanded in alphabetical order when no other order is specified by the search, and that the goal is state G. No visited or expanded lists are used. What order would the states be expanded by each type of search? Stop when you expand G. Write only the sequence of states expanded by each search.

Search Type	Sequence of States
Breadth First	
Depth First	
Iterative Deepening (step size 1)	
Uniform Cost	

Problem 3.2 (Tree Search in ProLog)

80pt

Implement the tree search algorithms presented in the lectures in ProLog, meaning:

1. BFS
2. DFS
3. Iterative Deepening with variable step size
4. Uniform cost search

Use the following implementation of trees:

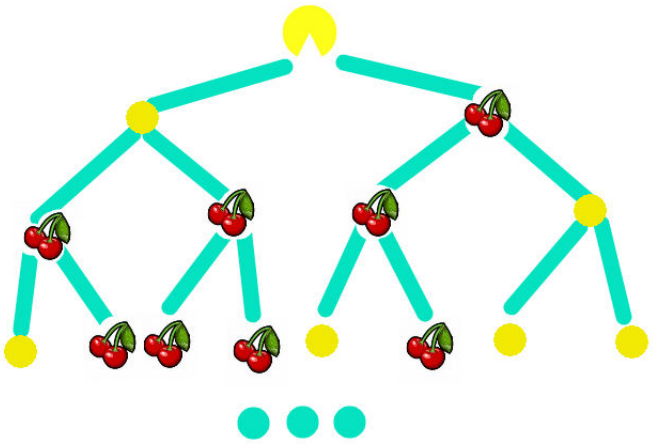
```
subtrees([]). % The empty list is a valid list of subtrees
istree(tree(Value,Children)) :- string(Value),subtrees(Children).
% A (valid) tree is a pair of some value/label
```

```
% (represented as a string) and a valid list of subtrees
subtrees([(Cost,T)|Rest]) :- number(Cost),istree(T), subtrees(Rest).
% A non-empty list is a valid list of subtrees, if it consists of pairs (Cost,T), where T
% is a valid tree and Cost is a number representing the step cost. i.e. the simple binary
% tree with root A, two children B, C and step costs A->B=2 and A->C=3 would be
%represented as: tree("A",[tree(2,("B",[])),tree(3,("C",[]))])
```

Solution: `https://swish.swi-prolog.org/p/Tree%20search.swinb`

4 Assignment 4 (A* and Adversarial Search I) – Given Nov. 18., Due Nov. 24.

Problem 4.1 (Binary Pacman)
Consider the Pacman game played on a perfect binary tree (full binary tree in which all leaves are at the same depth) similar to the picture below. 30pt



In each node, there is the possibility of having Cherries or not. The goal of the game is to reach one of the leaves with the maximum number of collected Cherries. A valid move from a node is only to the chidren of that node.

To solve this problem, define:

- a cost function
- 2 admissible heuristics $h_1(n)$ and $h_2(n)$ which are different from $h^*(n)$ (the true cost from n to goal), and not constant functions

that help find the solutions with most Cherries first, when using A^*. Give a short argument why your heuristics are admissible.

Prove that your choice of functions find the solutions with most Cherries first.

Provide a non-trivial example of the game with 5 levels and 15 Cherries randomly assigned to nodes (meaning don't put them all in the leaves, or all on 1 path, etc.) and show the order in which nodes are explored by A^* using each heuristic.

Solution: Since we want solutions with most Cherries first, and all goal states are equally far from the initial state, a good cost function could be $c(n) = \textit{number of nodes without Cherries}$.

Since the heuristics must be admissible, they have to be smaller than the real cost to the goal. But in order to know the real cost to the goal, we have to know the best solution of that subtree. Also, constant functions are not allowed.

To get around this, 2 heuristics could be $h_1(n) = \begin{cases} 0 & \text{if } \textit{this node has a Cherry} \\ ,1 & \text{else} \end{cases}$ and $h_2(n) = h_1(n) + \begin{cases} 1 & \text{if } \textit{both children of this node are } 1 \end{cases}$. Those heuristics are admissible because we add 1 every time we are forced to add 1 for a certain path, which cannot be avoided.

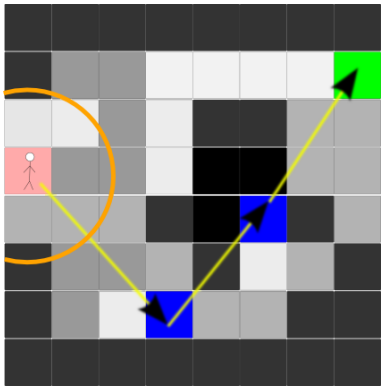
Assume that A^* with those heuristics doesn't find the solution with most Cherries first. All solutions explore the same numbers of nodes (hight of the binary tree). Assume that the best solution has cost c1, while our solution has cost $c2 > c1$. This means that at some step we explored a node with cost n and heuristic 1, before we explored a node with cost n and heuristic 0, but since we are using A^* this cannot happen. Contradiction.

Problem 4.2 (Let's play games!)

You are playing an RPG game where you have to control a character on a map. The map is internally represented as a matrix D of integer numbers between 0 and 10; the number represents how difficult is for your character to pass (therefore, if $D_{i,j} = 0$ then your character will pass really fast, while if $D_{i,j} = 10$ your character might not pass at all). Consider that the game assigns to your character a certain visibility radius r (that is, you cannot "see" further away than r units). You now want to move your character from one cell of the map to another. 50pt

- The programmer of the game decided to use A* for finding the shortest path (time-wise) between the two points. Design a heuristic that would work for the given problem.
- While playing the game, you observe that it consumes a lot of memory and is very slow while your character is moving. How would you improve the A* algorithm or the heuristic it uses in order to become more efficient?
- Consider that the programmer chose a really bad heuristic for the pathfinding algorithm, which takes your character through slow terrain. However, you are a skilled player and you already know the map by heart. You decide to use the "waypoint" feature of the game (defining a set of points the character has to reach in the order you input them in order to optimize the path). Design a heuristic function that would make use of the waypoints you set and/or of the lines connecting them.

For example, in the picture below, we encoded the difficulty of the terain in grayscale (white is equivalent to 0, up to black, which is equivalent to 10). The starting position is marked in red, the goal position in green. You already have set two waypoints marked in blue, which are connected by three lines in yellow. Your visible area is delimited by the orange circle around your starting position.



Note: The problem does not have a fixed solution, but rather is meant to make you think of varieties of A* and different heuristics. You are expected to give a rigurous description of your **assumptions** while solving it. You are not required to give precise formulas for the heuristics, but remember that they can make your solution more clear.

Solution:

Problem 4.3 (Adversarial Search)

20pt

Which of the following games can be handled by the appproaches to adversarial search discussed in the lecture (Section 6.1)? Explain.

1. 2-player Poker
2. Backgammon
3. Basketball
4. Cephalopod (for game rules check out `http://www.marksteeregames.com/Cephalopod_rules.pdf`)
5. Chinese Checkers (Halma)
6. Rock-Paper-Scissors

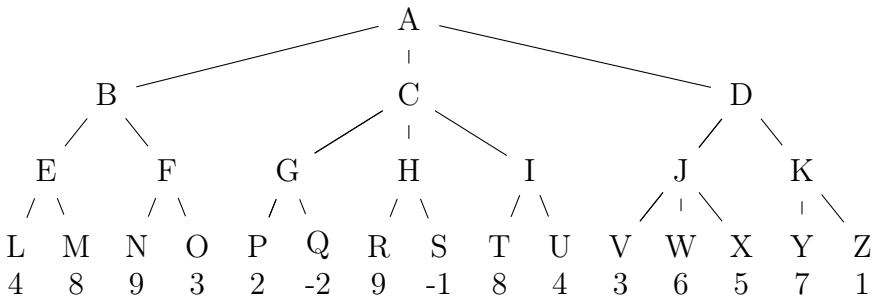
Solution:

1. (2-player Poker): No: Non-deterministic, incomplete information
2. (Backgammon) No: Non-deterministic
3. (Basketball) No: Too many players, non-deterministic, not discrete
4. Cephalopod Yes: Two-player turn-taking zero-sum game; deterministic (even though dice are used); full-information; finite state space, finite number of moves, ends after finite number of steps.
5. (Chinese Checkers): No: Too many players
6. (Rock-Paper-Scissors): No: Simultaneous moves

5 Assignment 5 (Adversarial Search II - MiniMax) – Given Nov. 23., Due Dec. 01.

Problem 5.1 (Game Tree)

Consider the following game tree. Assume it is the maximizing player's turn to move. The values at the leaves are the static evaluation function values of the states at each of those nodes. 30pt



10 pt

1. Label each non-leaf node with its minimax value. See above — 5 pt
2. Which move would be selected by Max? — 10 pt
3. List the nodes that the alpha-beta algorithm would prune (i.e., not visit). Assume children of a node are visited left-to-right. — 5 pt
4. In general (i.e., not just for the tree shown above), if we traverse a game tree by visiting children in right-to-left order instead of left-to-right, can this result in a change to
   (a) the minimax value computed at the root?
   (b) The number of nodes pruned by the alpha-beta algorithm?

Solution:

1. A:8, B:8, C:2, D:6, E:8, F:9, G:2, H:9, I:8, J:6, K:7
2. B
3. OHRSITUKYZ
4. (a) no, (b) yes

Problem 5.2 (Tic-Tac-Toe in Prolog)

70pt

Look at the prolog implementation of the game tic-tac-toe at `https://swish.swi-prolog.org/p/Tic-Tac-Toe.swinb` (original at `https://courses.cs.washington.edu/courses/cse341/03sp/slides/PrologEx/tictactoe.pl.txt`).

You can play a game by querying for playo and can watch the AI play against itself by typing selfgame.

The current AI strategy is implemented via the predicate orespond at lines 61ff. You'll notice that the primary rule for choosing a move is

```
orespond(Board,Newboard) :-
  move(Board, o, Newboard),
  not(x_can_win_in_one(Newboard)).
```

which basically translates to "pick the first valid move that doesn't result in x winning in their next move".

Replace orespond by a predicate implementing the MiniMax algorithm.

Solution:

6 Assignment 6 (Constraint Satisfaction Problems) – Given Dec. 14., Due Dec. 22.

Problem 6.1 Assume a CSP with a ternary constraint 20pt

$$(x_1, x_2, x_3) \in C \subseteq D_1 \times D_2 \times D_3.$$

Show how this constraint can be replaced with binary constraints by introducing additional variables over appropriate domains.

Solution: Additional variable x_4 with domain $D_4 = D_1 \times D_2$.

Constraints: $(x_4, x_3) \in C$, $x_1 = \pi_1(x_4)$, $x_2 = \pi_2(x_4)$ where $\pi_i : D_4 \to D_i$ is the projection.

I deducted points if the solution was not formally precise. In particular it is necessary to mention explicitly what new variables, domains and constraints are introduced.

(In this way all constraints can be reduced to binary ones. Whether that is a good idea is another question: The universes grow exponentially, but on the other hand there may be more highly optimized implementations for the binary case available.)

Problem 6.2 (50 Queens)

Formalize the *50 Queens Problem* as a constraint network. Hint: You do not have to write down all the constraints explicitly, but it has to be clear what the exact constraints are. 30pt

Problem 6.3 (CSP: Greater (or Lesser) Chain)

Consider the general less-than chain below, which we interpret as a CSP: Each of the N variables X_i has the domain $\{1, \ldots, M\}$. The constraints between adjacent variables X_i and X_{i+1} require that $X_i < X_{i+1}$. 50pt

$$X_1 < X_2 < X_3 < \cdots < X_N$$

1. For now, assume $N = M = 5$.
   (a) How many solutions does the CSP have?
   (b) What will the domain of X_1 be after enforcing the consistency of only the arc $X_1 \to X_2$?
   (c) What will the domain of X_1 be after enforcing the consistency of only the arcs $X_2 \to X_3$ and (then) $X_1 \to X_2$?
   (d) What will the domain of X_1 be after fully enforcing arc consistency?
2. Now consider the general case for arbitrary N and M.
   (a) What is the minimum number of arcs (big-O is ok) which must be processed by AC-3 (the algorithm which enforces arc consistency) on this graph before arc consistency is established?
   (b) Imagine you wish to construct a similar family of CSPs which forces one of the two following types of solutions: either all values must be ascending or all values must be descending, from left to right. For example, if $M = N = 3$, there would

be exactly two solutions: $\{1, 2, 3\}$ and $\{3, 2, 1\}$. Explain how to formulate this variant. Your answer should include a constraint graph and precise statements of variables and constraints.

Solution:

1. $N = M = 5$.
   (a) Just one: 1, 2, 3 ...
   (b) $\{1, 2, 3, 4\}$
   (c) $\{1, 2, 3\}$
   (d) $\{1\}$
2. (a) $O(MN)$. You will have to process many arcs multiple times, one for each domain value.
   (b) Several good answers. One is have ternary constraints on each adjacent triple that the triple should be either an increasing or decreasing triple. The overlap between triples enforces that the choice be global. Another is to introduce a global variable indicating ascent or descent and have ternary constraints between adjacent nodes and the global one, allowing, for example, triples like $\langle 1, 2, < \rangle$ or $\langle 2, 1, > \rangle$.

---

7 Assignment 7 (Propositional Logic) – Given Dec. 21., Due Jan. 12.

Problem 7.1 (Soundness of a Calculus)
We have a calculus for propositional logic that consist of the axioms 20pt

$$P \vee \neg P \qquad \text{and} \qquad P \wedge Q \Rightarrow Q \wedge P$$

and the inference rule (called "modus tollens" by the way)

$$\frac{P \Rightarrow Q \quad \neg Q}{\neg P}\, MT$$

Show that this calculus is sound.

Solution: Let $\mathcal{C}$ be the given calculus. We need to show that if $\mathcal{H} \vdash_{\mathcal{C}} \varphi$, then $\mathcal{H} \models \varphi$.

So, assume $\mathcal{H} \vdash_{\mathcal{C}} \varphi$ for some formula φ and some set of axioms $\mathcal{H}$. Proof by induction on $\mathcal{C}$:

- (Base case 1) $\varphi \in \mathcal{H}$, then trivially $\mathcal{H} \models \varphi$.
- (Base case 2) $\varphi = P \vee \neg P$ for some formula P. Then φ is a tautology (proof trivial) and thus $\mathcal{H} \models \varphi$.
- (Base case 3) $\varphi = P \wedge Q \Rightarrow Q \wedge P$ for formulas P, Q. Again φ is a tautology and thus $\mathcal{H} \models \varphi$.
- (Induction step) $\varphi = \neg P$ for some formula P and there are formulas $P \Rightarrow Q$ and $\neg Q$ such that $\mathcal{H} \vdash_{\mathcal{C}} P \Rightarrow Q$ and $\mathcal{H} \vdash_{\mathcal{C}} \neg Q$. By induction hypothesis we may assume $\mathcal{H} \models P \Rightarrow Q$ and $\mathcal{H} \models \neg Q$. Naturally, $P \Rightarrow Q$ is equivalent to $\neg Q \Rightarrow \neg P$, so by modus ponens we have $\mathcal{H} \models \neg P$.

Problem 7.2 (Natural Deduction)

20pt

Prove the following formula using only the rules of the Natural Deduction calculus.

$$(\mathbf{A} \Rightarrow \mathbf{B} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{A} \wedge \mathbf{B} \Rightarrow \mathbf{C}$$

Specify the rules applied at each step.

We will now consider a formulation of propositional logic, which we will call PL_{NQ} (**P**redicate **L**ogic with **N**o **Q**uantifiers). The idea is to use very elaborate names for propositional logic: `ProLog` terms, which encode atomic formulae.

Use `ProLog` for Talking/Programming about Logics

- **Idea**: We will use PLNQ (prop. logic where prop. variables are ADT terms)

- represent the ADT as facts of the form

```
constant(mia).
pred(love,2).
pred(run,1).
fun(father,1)
```

  this licenses ProLog terms like run(mia). and love(mia,father(mia)).

- represent propositional connectives as ProLog operators, which we declare with the following declarations.

```
:- op(900,yfx,<>). % equivalence
:- op(900,yfx,>). % implication
:- op(850,yfx,\/). % disjunction
:- op(800,yfx,\&). % conjunction
:- op(750,fx,~). % negation
```

  The first argument of **op** is the operator precedence, the second the fixity. This licenses ProLog terms like X > Y. and ~(X).

- Use the ProLog built-in predicate =.. to deconstruct terms: a literal f(a,b)=..Z binds Z to the list [f,a,b], i.e. the first element of the list is the function/predicate symbol, followed by the arguments.

Problem 7.3 Write a simple syntax checker that checks arities in function application and complex formulae by writing a predicate wff/1[1]. 10pt

Solution:

```
term(X):-constant(X).
term(X):-X=..[Y|R],length(R,N),fun(Y,N),termlist(R).
termlist([]).
termlist([X|R]):-term(X),termlist(R).
wffatom(X):-X=..[Y|R],length(R,N),pred(Y,N),termlist(R).
wff(X):-wffatom(X).
wff(X):-X=..[~,Y],wff(Y).
wff(X):-X=..[Y,Z,W],member(Y,[<>,>,\/,&]),wff(Z),wff(W).
```

Problem 7.4 Remember that we call a set $\mathcal{H}$ of atomic formulae in PLNQ a Herbrand model; it induces a valuation ν for PLNQ by $\nu(A) := \mathsf{T}$, iff $A \in \mathcal{H}$. 10pt

Write a couple of example Herbrand models (sets of atomic formulae), using a binary model/2[2] relation, given by ProLog facts like the following

```
model(3,[love(peter,mary),hate(mary,peter)]).
```

Check well-formedness of the models, using the predicate wff/1 from Problem 7.3.

Solution:

[1] the /1 is the notation for a unary predicate.

[2] the first parameter just denotes the number of the model.

```
wfm(N):-model(N,L),wffatomlist(L).
wffatomlist([]).
wffatomlist([X|R]):-wffatom(X),wffatomlist(R).
```

Problem 7.5 Write a simple evaluator for closed formulae 15pt

```
evaluate(love(peter,mary) \& hate(mary,peter),3)
```

should succeed. evaluate should fail if the input is not valid or ill-formed.

Solution:

```
evaluate(X,N) :- wff(X),model(N,L),eval(X,L).
eval(X,L) :- member(X,L).
eval(~(X),L) :- \+ eval(X,L).
eval(X & Y,L) :- eval(X,L),eval(Y,L).
eval(X \/ _,L) :- eval(X,L).
eval(_ \/ Y,L) :- eval(Y,L).
eval(X > Y,L) :- eval(Y,L); not(eval(X,L)).
eval(X <> Y,L) :- eval(X,L),eval(Y,L).
eval(X <> Y,L) :- \+ eval(X,L), \+ eval(Y,L).
```

Problem 7.6 Write a translator predicate that translates away all logical connectives except & and ~. 10pt

Solution:

```
naonly(X,X) :- wffatom(X).
naonly(~(X),~(Z)) :- naonly(X,Z).
naonly(X & Y, Z & W) :- naonly(X,Z), naonly(Y,W).
naonly(X \/ Y,~(~(Z) & ~(W))) :- naonly(X,Z), naonly(Y,W).
naonly(X > Y,~(Z & ~(W))) :- naonly(X,Z), naonly(Y,W).
naonly(X <> Y, (Z > W) & (W > Z)) :- naonly(X,Z),naonly(Y,W).
```

If we want to improve the output quality (less negations), we can introduce a line

```
naonly(~(~(X)),Z) :- naonly(X,Z).
```

before the negation treatment in line 2 (otherwise it is useless; why?)

Problem 7.7 Revise the evaluator to a tableau theorem prover/model generation procedure for closed formulae that only contain the connectives & and ~. 15pt

Solution: We first need two special term constructors that associate truth values with terms. We take **true**(X) and false(X) to mean that the formula represented by X carries the label T or F. We say that a labeled formula is a literal, iff it is a labeled atom.

```
literal(true(X)) :- wffatom(X).
literal(false(X)) :- wffatom(X).
```

We develop the model generator along the lines of the evaluator from the last handout, recursing over the structure of the term in question. Like the evaluator, we will make use of `ProLog`'s backtracking feature to construct the models by depth-first search.

```
clash(List) :- member(A,List),member(~A,List).
tabl(F,[F]) :- literal(F).
tabl(true(A&B),H) :- tabl(true(A),K), tabl(true(B),L),
                     append(K,L,P),list_to_set(P,H),\+clash(H).
tabl(false(A&_,H)) :- tabl(false(A),H).
tabl(false(_&B,H)) :- tabl(false(B),H).
tabl(true(~A,H)) :- tabl(false(A),H).
tabl(false(~A,H)) :- tabl(true(A),H).
```

Extend the model generator from the last problem to one that works on arbitrary closed formulae and takes closed world knowledge into account. We will use the bits and pieces that we have developed so far. The first step is to convert the world knowledge into a conjunction of formulae in ~,& form that can be fed to the tableau (we append the formula X at the end.)

```
mg(X,N,H) :- wk(N,L), toconj(L,I),naonly(X,Y),tab(I&Y,H).
toconj([X],X).
toconj([X|L],Y&C) :- naonly(X,Y),toconj(L,C).
```

```
(List("holidays","new year").map("happy " + _)).mkString(" and a ")
```

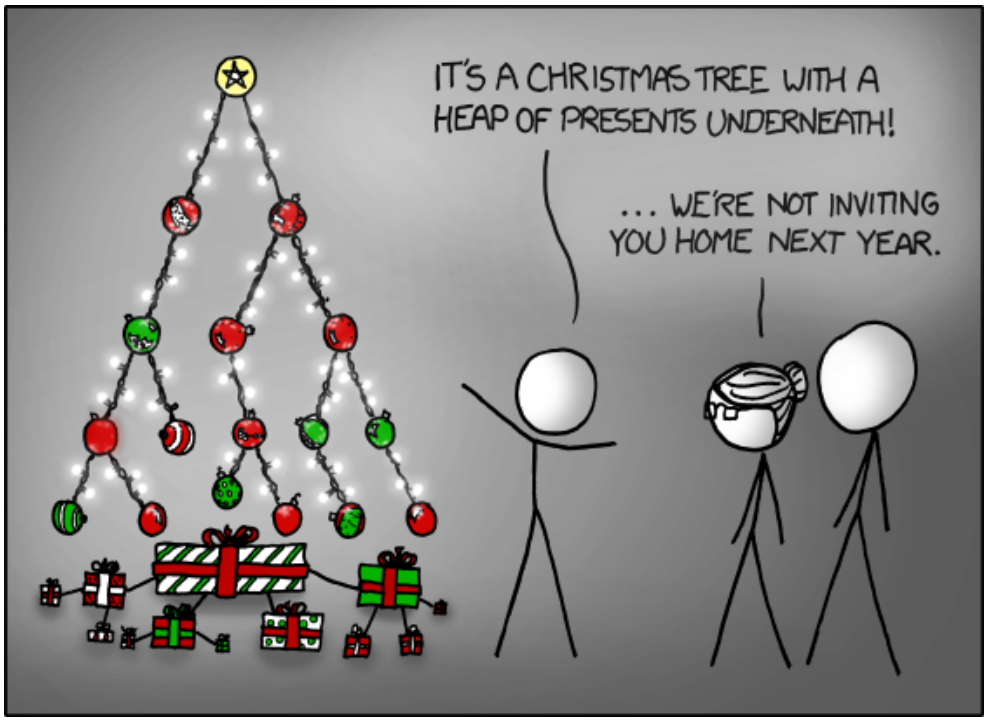


Figure 1: `https://xkcd.com/835/`

8 Assignment 8 (Propositional Logic) – Given Dec. 11., Due Jan. 19.

Problem 8.1 (The NOR Connective)
Show that all logical binary connectives $(\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow)$ can be expressed by the $\downarrow$ (*nor*) connective, which is (or, rather, can be) defined as $\mathbf{A} \downarrow \mathbf{B} := \neg(\mathbf{A} \vee \mathbf{B})$. Rewrite $\mathbf{P} \vee \neg \mathbf{P}$ (tertium non datur) into an expression containing only $\downarrow$ as a logical connective. 40pt

Solution: $P \vee \neg P = \neg\neg(P \vee \neg P) = \neg(P \downarrow \neg P) = (P \downarrow (P \downarrow P)) \downarrow (P \downarrow (P \downarrow P))$

Problem 8.2 (Calculi Comparison)
Prove (or disprove) the validity of the following formulae in i) Natural Deduction ii) Tableau and iii) Resolution: 60pt

1. $(P \wedge Q) \Rightarrow (P \vee Q)$
2. $((A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C)) \Rightarrow C$
3. $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$

Can you identify any advantages or disadvantage of the calculi, and in which situations?

Solution:

ND 1.

(1)	1	$(P \wedge Q)$	Assumption
(2)	1	P	$\wedge E_\ell$ (on 1)
(3)	1	$(P \vee Q)$	$\vee I_\ell$ (on 2)
(4)		$(P \wedge Q) \Rightarrow (P \vee Q)$	$\Rightarrow I$ (on 1 and 3)

2.

(1)	1	$(A \vee B) \wedge ((A \Rightarrow C) \wedge (B \Rightarrow C))$	Assumption
(2)	1	$(A \vee B)$	$\wedge E_\ell$ (on 1)
(3)	1	$(A \Rightarrow C) \wedge (B \Rightarrow C)$	$\wedge E_r$ (on 1)
(4)	1	$(A \Rightarrow C)$	$\wedge E_\ell$ (on 3)
(5)	1	$(B \Rightarrow C)$	$\wedge E_r$ (on 3)
(6)	1,6	A	Assumption
(7)	1,6	C	$\Rightarrow E$ (on 4 and 6)
(8)	1,8	B	Assumption
(9)	1,8	C	$\Rightarrow E$ (on 5 and 8)
(10)	1	C	$\vee E$ (on 2, 7 and 9)
(11)		$((A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C)) \Rightarrow C$	$\Rightarrow I$ (on 1 and 10)

3.

(1)		$(P \vee \neg P)$	TND
(2)	2	P	Assumption
(3)	2,3	$(P \Rightarrow Q) \Rightarrow P)$	Assumption
(4)	2	$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	$\Rightarrow I$ (on 3 and 2)
(5)	5	$\neg P$	Assumption
(6)	5,6	$(P \Rightarrow Q) \Rightarrow P)$	Assumption
(7)	5,6,7	P	Assumption
(8)	5,6,7	F	FI (on 5 and 7)
(9)	5,6,7	Q	FE (on 8)
(10)	5,6	$P \Rightarrow Q$	$\Rightarrow -I$ (on 7 and 9)
(11)	5,6	P	$\Rightarrow E$ (on 6 and 10)
(12)	5	$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	$\Rightarrow$ I (on 6 and 11)
(13)		$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$	$\vee E$ (on 1, 4 and 12)

Tableau 1.

(1)	$(P \wedge Q) \Rightarrow (P \vee Q)^F$	
(2)	$(P \wedge Q)^T$	(from 1)
(3)	$(P \vee Q)^F$	(from 1)
(4)	P^T	(from 2)
(5)	Q^T	(from 2)
(6)	P^F	(from 3)

2.

(1)	$((A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C)) \Rightarrow C^F$	
(2)	$(A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C)^T$	(from 1)
(3)	C^F	(from 1)
(4)	$(A \vee B)^T$	(from 2)
(5)	$(A \Rightarrow C)^T$	(from 2)
(6)	$(B \Rightarrow C)^T$	(from 2)
(7)	A^T | A^F | C^T (split on 5) ; B^T | B^F | C^T (split on 6)	(split on 4)

3.

(1)	$((P \Rightarrow Q) \Rightarrow P) \Rightarrow P^F$	
(2)	$(P \Rightarrow Q) \Rightarrow P)^T$	(from 1)
(3)	P^F	(from 1)
(4)	$P \Rightarrow Q^F$ | P^T ; (5) | P^T (from 4)	(split on 2)

Resolution 1. $(P \wedge Q) \Rightarrow (P \vee Q)$: We negate and build a CNF:

$$\begin{aligned} &(P \wedge Q) \wedge \neg(P \vee Q) \\ \equiv &P \wedge Q \wedge \neg P \wedge \neg Q \end{aligned}$$

yielding clauses $\{P^T\}, \{Q^T\}, \{P^F\}, \{Q^F\}$

2. $((A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C)) \Rightarrow C$: We negate and build a CNF:

$$\begin{aligned} &((A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C)) \wedge \neg C \\ \equiv &(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee C) \wedge \neg C \end{aligned}$$

yielding clauses $\{A^T, B^T\}, \{A^F, C^T\}, \{B^F, C^T\}, \{C^F\}$.
Resolving yields:

$$\begin{aligned}
\{A^F, C^T\} + \{C^F\} &\Longrightarrow \{A^F\} \\
\{B^F, C^T\} + \{C^F\} &\Longrightarrow \{B^F\} \\
\{\ \mathrm{A}^T, B^T\} + \{A^F\} &\Longrightarrow \{B^T\} \\
\{\ \mathrm{B}^T\} + \{B^F\} &\Longrightarrow \{\}
\end{aligned}$$

3. $((P \Rightarrow Q) \Rightarrow P) \Rightarrow P$: We negate and build a CNF:

$$\begin{aligned}
&((P \Rightarrow Q) \Rightarrow P) \wedge \neg P \\
\equiv&(\neg(P \Rightarrow Q) \vee P) \wedge \neg P \\
\equiv&((P \wedge \neg Q) \vee P) \wedge \neg P \\
\equiv&((P \vee P) \wedge (\neg Q \vee P) \wedge \neg P
\end{aligned}$$

yielding clauses $\{P^T\}, \{Q^F, P^T\}, \{P^F\}$.

---

9 Assignment 9 (First-order Logic) – Given Jan. 18., Due Jan. 26.

Problem 9.1 (Free variables)

Let $H \in \Sigma_1^p$ and $f \in \Sigma_2^f$. For each of the following strings of symbols, state whether they are syntactically correct first-order formulae. If they are, state the free and bound variables. If they are not, explain why: 30pt

1. $\exists X.(H(Y) \Rightarrow \forall Y.\neg H(f(X,Y)))$
2. $(H(Y) \Rightarrow \exists Y.H(f(Y)))$
3. $\forall X.(H(X) \wedge \exists Y.H(f(X,Z)))$
4. $H(X)$
5. $\forall X.\forall Y.H(X,Y)$

Solution:

1. X is bound, Y is both free and bound.
2. Not well-formed - f is binary.
3. X is bound, Z is free.
4. X is free.
5. Not well-formed - H is unary.

Problem 9.2 (First-Order Semantics)

Let $=\in \Sigma_2^p$, $P \in \Sigma_1^p$ and $+ \in \Sigma_2^f$. We use the semantics of first-order logic without equality. 40pt

- Prove or disprove the following formulae semantically, using value functions and without using a proof calculus. If a formula is not valid, give a model in which it is false.
- Which formulae change their validity if we allow for empty models, and why?

1. $\forall X.\forall Y X + Y = X + Y$
2. $\exists X.(P(X) \Rightarrow \forall Y.P(Y))$
3. $P(Y) \Rightarrow \exists X.P(X)$

Solution: Let φ be any value function.

1. Not valid - Counter-model: Let $\mathcal{I}_\varphi(=)$ be the empty relation on an arbitrary domain.

2. Valid:

$$\begin{aligned}
&\mathcal{I}_\varphi(\exists X.(P(X) \Rightarrow \forall Y.P(Y))) = \top \\
\Leftrightarrow &\text{There is some } a \in \mathcal{D}_\mathcal{I} \text{ s.t. } \mathcal{I}_\varphi((P(a) \Rightarrow \forall Y.P(Y))) = \top \\
\Leftrightarrow &\text{There is some } a \in \mathcal{D}_\mathcal{I} \text{ s.t. } \mathcal{I}_\varphi(\neg(P(a) \wedge \neg\forall Y.P(Y))) = \top \\
\Leftrightarrow &\text{There is some } a \in \mathcal{D}_\mathcal{I} \text{ s.t. } \mathcal{I}_\varphi(P(a) \wedge \neg\forall Y.P(Y)) = \bot \\
\Leftrightarrow &\text{There is some } a \in \mathcal{D}_\mathcal{I} \text{ s.t. } \mathcal{I}_\varphi(P(a)) = \bot \text{ or } \mathcal{I}_\varphi(\neg\forall Y.P(Y)) = \bot \\
\Leftrightarrow &\text{There is some } a \in \mathcal{D}_\mathcal{I} \text{ s.t. } \mathcal{I}_\varphi(P(a)) = \bot \text{ or } \mathcal{I}_\varphi(\forall Y.P(Y)) = \top \\
\Leftrightarrow &\text{There is some } a \in \mathcal{D}_\mathcal{I} \text{ s.t. } \mathcal{I}_\varphi(P(a)) = \bot \text{ or for all } b \in \mathcal{D}_\mathcal{I} : \mathcal{I}_\varphi(P(b)) = \top
\end{aligned}$$

3. Valid:

$$\begin{aligned}
&\mathcal{I}_\varphi(P(Y) \Rightarrow \exists X.P(X)) = \top \\
\Leftrightarrow &\mathcal{I}_\varphi(\neg(P(Y) \wedge \neg\exists X.P(X))) = \top \\
\Leftrightarrow &\mathcal{I}_\varphi(P(Y) \wedge \neg\exists X.P(X)) = \bot \\
\Leftrightarrow &\mathcal{I}_\varphi(P(Y)) = \bot \text{ or } \mathcal{I}_\varphi(\neg\exists X.P(X)) = \bot \\
\Leftrightarrow &\mathcal{I}_\varphi(P(Y)) = \bot \text{ or } \mathcal{I}_\varphi(\exists X.P(X)) = \top \\
\Leftrightarrow &\mathcal{I}_\varphi(P)(\varphi(Y)) = \bot \text{ or there is some } a \in \mathcal{D}_\mathcal{I} \text{ s.t. } \mathcal{I}_\varphi(P(a)) = \top
\end{aligned}$$

Problem 9.3 (Variable Assignments)

Let φ_1, φ_2 be two variable assignments and $\mathbf{A}$ a first-order formula. Prove: If $\varphi_1(X) = \varphi_2(X)$ for all variables $X \in \mathsf{free}(\mathbf{A})$, then $\mathcal{I}_{\varphi_1}(\mathbf{A}) = \mathcal{I}_{\varphi_2}(\mathbf{A})$. Use structural induction on the defintion of a value function. 30pt

Solution: Proof by induction:

- $\mathbf{A} = X$ for some variable X, then by assumption $\varphi_1(X) = \varphi_2(X)$ and hence $\mathcal{I}_{\varphi_1}(\mathbf{A}) = \mathcal{I}_{\varphi_2}(\mathbf{A})$.
- $\mathbf{A} = \top$ or $\mathbf{A} = \bot$, then $\mathcal{I}_{\varphi_1}(\mathbf{A}) = \mathcal{I}_{\varphi_2}(\mathbf{A}) = \top$ (respectively $\bot$).
- $\mathbf{A} = f(t_1, ..., t_n)$ for some n-ary function symbol and the claim holds (by induction hypothesis) for $t_1, ..., t_n$. Then trivially $\mathcal{I}_{\varphi_1}(\mathbf{A}) = \mathcal{I}_{\varphi_2}(\mathbf{A})$.
- $\mathbf{A} = \neg\mathbf{A}'$ and the claim holds by induction hypothesis for $\mathbf{A}'$. Then $\mathcal{I}_{\varphi_1}(\mathbf{A}') = \mathcal{I}_{\varphi_2}(\mathbf{A}')$ and hence $\mathcal{I}_{\varphi_1}(\mathbf{A}) = \mathcal{I}_{\varphi_2}(\mathbf{A})$.
- $\mathbf{A} = (\mathbf{A}_1 \wedge \mathbf{A}_2)$ and the claim holds by induction hypothesis for $\mathbf{A}_1$ and $\mathbf{A}_2$. Then $\mathcal{I}_{\varphi_1}(\mathbf{A}) = \top$ iff $\mathcal{I}_{\varphi_1}(\mathbf{A}_1) = \top$ and $\mathcal{I}_{\varphi_1}(\mathbf{A}_2) = \top$, which by IH is the case iff $\mathcal{I}_{\varphi_2}(\mathbf{A}_1) = \top$ and $\mathcal{I}_{\varphi_2}(\mathbf{A}_2) = \top$, which is the case iff $\mathcal{I}_{\varphi_2}(\mathbf{A}) = \top$.
- $\mathbf{A} = \forall X.\mathbf{A}'$ and by induction hypothesis the claim holds for $\mathbf{A}'$. Then $\mathcal{I}_{\varphi_1}(\mathbf{A}) = \top$ iff for all $a \in \mathcal{D}_\mathcal{I} : \mathcal{I}_{\varphi_1}(\mathbf{A}'[\frac{a}{X}]) = \top$, which is the case iff for all $a \in \mathcal{D}_\mathcal{I} : \mathcal{I}_{\varphi_2}(\mathbf{A}'[\frac{a}{X}]) = \top$, which is the case iff $\mathcal{I}_{\varphi_2}(\mathbf{A}) = \top$.

---

10 Assignment 10 (First-order Logic Proving) – Given Jan. 25., Due Feb. 2.

Problem 10.1 (Natural Deduction)
Let $\leq, \in \Sigma_2^p$, $P \in \Sigma_1^p$, $+ \in \Sigma_2^f$ and $- \in \Sigma_1^f$. We use the semantics and natural deduction of first-order logic with equality. 40pt

Prove the following formulae in Natural Deduction:

1. $\forall X.(X \leq -X \Rightarrow \exists Y.X \leq Y)$
2. $\exists X.(P(X) \Rightarrow \forall Y.P(Y))$

(Hint: The second formula requires the law of the excluded middle and is somewhat elaborate. Try proving the lemma $\neg\forall Y.P(Y) \Rightarrow \exists Y.\neg P(Y)$ first and use that in your actual proof)

Solution:

1.

1(Assumption)^1	$X \leq -X$
2$\exists$-Introduction	$\exists Y.X \leq Y$
3 $\Rightarrow$ -Introduction^1	$X \leq -X \Rightarrow \exists Y.X \leq Y$
4$\forall$-Introduction	$\forall X.(X \leq -X \Rightarrow \exists Y.X \leq Y)$

2. We start with the lemma $\neg\forall Y.P(Y) \Rightarrow \exists Y.\neg P(Y)$:

(Assumption)^1	$\neg\forall Y.P(Y)$
(Assumption)^2	$\neg\exists Y.\neg P(Y)$
(Assumption)^3	$\neg P(Y)$
$\exists$-Introduction	$\exists Y \neg P(Y)$
$\bot$-Introduction	$\bot$
$\neg$-introduction^3	$\neg\neg P(Y)$
$\neg$-Elimination	$P(Y)$
$\forall$-Introduction	$\forall Y.P(Y)$
$\bot$-Introduction	$\bot$
$\neg$-introduction^2	$\neg\neg\exists Y.\neg P(Y)$
$\neg$-Elimination	$\exists Y.\neg P(Y)$
$\Rightarrow$ -Introduction^1	$\neg\forall Y.P(Y) \Rightarrow \exists Y.\neg P(Y)$

Now for the actual proof, using the above lemma:

(TND)	$\forall Y.P(Y) \vee \neg\forall Y.P(Y)$
(Assumption)^1	$\forall Y.P(Y)$
(Assumption)^2	$P(X)$
$\Rightarrow$ -Introduction^2	$P(X) \Rightarrow \forall Y.P(Y)$
$\exists$-Introduction	$\exists X.(P(X) \Rightarrow \forall Y.P(Y))$
(Assumption)^1	$\neg\forall Y.P(Y)$
(Lemma)	$\neg\forall Y.P(Y) \Rightarrow \exists Y.\neg P(Y)$
$\Rightarrow$ -Elimination	$\exists Y.\neg P(Y)$
$\exists$-Elimination	$\neg P(c)$
(Assumption)^2	$P(c)$
$\bot$-Introduction	$\bot$
$\bot$-Elimination	$\forall Y.P(Y)$
$\Rightarrow$ -Introduction^2	$P(c) \Rightarrow \forall Y.P(Y)$
$\exists$-Introduction	$\exists X.(P(X) \Rightarrow \forall Y.P(Y))$
$\vee$ -Elimination^1	$\exists X.(P(X) \Rightarrow \forall Y.P(Y))$

Problem 10.2 (First-Order Resolution)

Prove the following formula using resolution. 30pt

$P \in \Sigma_1^p, R \in \Sigma_2^p, a, b \in \Sigma_0^f$

$$\exists X.\forall Y.\exists Z.\exists W.\,((\neg P(Z) \wedge \neg R(b,a)) \vee \neg R(a,b) \vee R(W,a) \vee (P(Y) \wedge R(X,b)))$$

Solution: We negate:

$$\forall X.\exists Y.\forall Z.\forall W.(P(Z) \vee R(b,a)) \wedge R(a,b) \wedge \neg R(W,a) \wedge (\neg P(Y) \vee \neg R(X,b))$$

We skolemize:

$$(P(Z) \vee R(b,a)) \wedge R(a,b) \wedge \neg R(W,a) \wedge (\neg P(f_Y(X)) \vee \neg R(X,b))$$

This yields the clauses $\{P(Z)^T, R(b,a)^T\}, \{R(a,b)^T\}, \{R(W,a)^F\}, \{P(f_Y(X))^F, R(X,b)^F\}$. We resolve:

$$\{P(Z)^T, R(b,a)^T\} + \{R(W,a)^F\}[b/W] \Longrightarrow \{P(Z)^T\}$$
$$\{R(a,b)^T\} + \{P(f_Y(X))^F, R(X,b)^F\}[a/X] \Longrightarrow \{P(f_Y(a))^F\}$$
$$\{P(Z)^T\}[f_Y(a)/Z] + \{P(f_Y(a))^F\} \Longrightarrow \{\}$$

Problem 10.3 (First-Order Tableaux)

Prove the following formula using the tableaux calculus. 30pt

$P \in \Sigma_1^p$

$$\exists X.\,(P(X) \Rightarrow \forall Y.P(Y))$$

Solution:

(1)	$\exists X.(P(X) \Rightarrow \forall Y.P(Y))^F$	
(2)	$P(V_X) \Rightarrow \forall Y.P(Y)^F$	(from 1)
(3)	$P(V_X)^T$	(from 2)
(4)	$\forall Y.P(Y)^F$	(from 2)
(5)	$P(c_Y)^F$	(from 4)
(6)	$\bot[c_Y/V_X]$	

11 Assignment 11 (Planning) – Given Feb. 1.., Due Feb. 1.

Problem 11.1 (STRIPS)
Encode the following problems as STRIPS planning tasks (Definition 15.3.2 in the slides): 40pt

1. *Generalized Die Hard Problem:* You are given a water spout and two jugs, one holding p and one holding q gallons, where $p < q$ and p and q are relatively prime. Starting with both jugs empty, the goal is to have exactly k gallons in one of the jugs. You can only fill the jugs from the spout fully.

2. *Generalized Seven Bridges of Königsberg*: Given a river with several islands and various bridges between the islands and the two sides of the river, the goal is to find a path which crosses every bridge exactly once.

Solution:

1. $$P = \{Jug_p(n) \mid 0 \leq n \leq p,\ n \in \mathbb{N}\} \cup \{Jug_q(n) \mid 0 \leq n \leq q,\ n \in \mathbb{N}\}$$
$$I = \{Jug_p(0), Jug_q(0)\}$$
$$G = \{Jug_p(k)\}$$

   (After all, if we manage to get k liters in jug q, we can just empty p and fill it with the contents of q; that's just two added steps, but it makes G definable)

   A contains:

   (a) $Empty_p : \mathsf{pre} = \{\}, \mathsf{add} = \{Jug_p(0)\}, \mathsf{del} = \{Jug_p(n) \mid 1 \leq n \leq p\}$

   (b) $Empty_q : \mathsf{pre} = \{\}, \mathsf{add} = \{Jug_q(0)\}, \mathsf{del} = \{Jug_q(n) \mid 1 \leq n \leq q\}$

   (c) $FillUp_p : \mathsf{pre} = \{\}, \mathsf{add} = \{Jug_p(p)\}, \mathsf{del} = \{Jug_p(n) \mid 0 \leq n < p\}$

   (d) $FillUp_q : \mathsf{pre} = \{\}, \mathsf{add} = \{Jug_q(q)\}, \mathsf{del} = \{Jug_q(n) \mid 0 \leq n < q\}$

   (e) For all x, y with $0 \leq x \leq p,\ 0 \leq y \leq q$, and with $m = \mathsf{min}(x + y, p)$:
   $Fillp_{x,y}$:
   $$\mathsf{pre} = \{Jug_p(x), Jug_q(y)\},$$
   $$\mathsf{add} = \{Jug_p(m), Jug_q(m - x)\},$$
   $$\mathsf{del} = \{Jug_p(z) \mid z \neq m\} \cup \{Jug_q(z) \mid z \neq m - x\}$$
   and with $m = \mathsf{min}(x + y, q)$:
   $Fillq_{x,y}$:
   $$\mathsf{pre} = \{Jug_p(x), Jug_q(y)\},$$
   $$\mathsf{add} = \{Jug_p(m - y), Jug_q(m)\},$$
   $$\mathsf{del} = \{Jug_p(z) \mid z \neq m - y\} \cup \{Jug_q(z) \mid z \neq m\}$$

2. Let I be the set of places, B a set of bridges and $from, to : B \to I$ and $i_0 \notin I$ a starting point.

$$P = \{IsAt(i) \mid i \in I\} \cup \{Visited(b) \mid b \in B\} \cup \{NotVisited(b) \mid b \in B\} \cup \{IsAt(i_0)\}$$

$$I = \{IsAt(i_0)\} \cup \{NotVisited(b) \mid b \in B\}$$

$$G = \{Visited(b) \mid b \in B\}$$

A contains:

(a) For every $i \in I$:

$$startAt(i) : \mathsf{pre} = \{IsAt(i_0)\}, \mathsf{add} = \{IsAt(i)\}, \mathsf{del} = \{isAt(i_0)\}$$

(b) For every $b \in B$:
$GoVia(b)$:

$$\mathsf{pre} = \{NotVisited(b), IsAt(from(b))\},$$

$$\mathsf{add} = \{IsAt(to(b)), Visited(b)\},$$

$$\mathsf{del} = \{isAt(from(b)), NotVisited(b)\}$$

Problem 11.2 Consider the following problem. A bicycle has a front wheel and a back wheel installed and both wheels have a flat tire. A robot needs to repair the bicycle. The room also contains a tire pump and a box with all the other equipment needed by the robot to repair a bicycle. The robot can repair a wheel with the help of the box and the tire pump when the robot and the three objects are at the same position. The bicycle is repaired when the robot has done a final overall check which requires both tires to be repaired and to be installed on the bicycle again. For this check, the box is also needed at the same position as the bicycle and the robot. 40pt

The exercise is to model this problem as a STRIPS planning task. In doing so, assume the following framework. The robot is currently at position "A", the bicycle is at position "B", and the "Frontwheel" and the "Backwheel" are installed on the "Bicycle". The "Box" is at position "C" and the "Pump" at position "D". The actions available for the robot are:

- "*Go*" from one position to another. The four possible positions A, B, C, and D are connected in such a way that the robot can reach every other place in one "*Go*".
- "*Push*" an object from one place to another. The bicycle is not pushable and the wheels are only pushable if not installed on the bicycle; obviously the robot cannot push itself; every other object is always pushable. "*Push*" moves both the object and the robot.
- "*Remove*" a wheel from the bike.
- "*RepairWheel*" to fix a wheel with a flat tire.
- "*InstallWheel*" to put a wheel back on the bike.

- "$FinalCheck$" to make sure that not only the two wheels are repaired and installed but also the rest of the bike is in good condition. The box is needed at the same position for this.

(a) Write a STRIPS formalization of the initial state and goal descriptions.

(b) Write a STRIPS formalization of the six actions. In doing so, please make use of "object variables", i.e., write the actions up in a parametrized way. State, for each parameter, by which objects it can be instantiated.

In both (a) and (b), make use of only the following predicates:

- $At(x, y)$: To indicate that object $x \in \{Robot, Bicycle, Frontwheel, Backwheel, Box, Pump\}$ is at position $y \in \{Bicycle, A, B, C, D\}$.
- $Pushable(x)$: To indicate that object $x \in \{Robot, Bicycle, Frontwheel, Backwheel, Box, Pump\}$ can be pushed.
- $Repaired(x)$: To indicate that object $x \in \{Robot, Bicycle, Frontwheel, Backwheel, Box, Pump\}$ is repaired.
- $FlatTire(x)$: To indicate that object $x \in \{Robot, Bicycle, Frontwheel, Backwheel, Box, Pump\}$ has a flat tire.

Solution:

(a) Initial state description:

$$\begin{aligned} I \;=\; & \{At(Robot, A), At(Bicycle, B), At(Frontwheel, Bicycle), \\ & At(Backwheel, Bicycle), At(Box, C), At(Pump, D), Pushable(Box), \\ & Pushable(Pump), FlatTire(Frontwheel), FlatTire(Backwheel)\} \end{aligned}$$

Goal description: $G = \{Repaired(Bicycle)\}$

(b) Action descriptions:

- $Go(x, y) =$

$$\begin{aligned} pre &: \{At(Robot, x)\} \\ add &: \{At(Robot, y)\} \\ del &: \{At(Robot, x)\} \end{aligned}$$

for all $x, y \in \{A, B, C, D\}$.

- $Push(x, y, z) =$

$$\begin{aligned} pre &: \{At(Robot, y), At(x, y), Pushable(x)\} \\ add &: \{At(Robot, z), At(x, z)\} \\ del &: \{At(Robot, y), At(x, y)\} \end{aligned}$$

for all $x \in \{Robot, Bicycle, Wheel, Box, Pump\}$, $y, z \in \{A, B, C, D\}$.

- $Remove(x, y) =$

  $pre : \{At(Robot, x), At(Bicycle, x), At(y, Bicycle)\}$
  $add : \{At(y, x), Pushable(y)\}$
  $del : \{At(y, Bicycle)\}$

  for all $x \in \{A, B, C, D\}$, $y \in \{Frontwheel, Backwheel\}$.

  [Note that the facts Pushable(Frontwheel) and Pushable(Backwheel) can also be initially given and never deleted because of the way the action "Push" is defined.]

- $RepairWheel(x, y) =$

  $pre : \{At(Robot, x), At(y, x), FlatTire(y), At(Box, x), At(Pump, x)\}$
  $add : \{Repaired(y)\}$
  $del : \{FlatTire(y)\}$

  for all $x \in \{A, B, C, D\}$, $y \in \{Frontwheel, Backwheel\}$.

- $InstallWheel(x, y) =$

  $pre : \{At(Robot, x), At(Bicycle, x), At(y, x), Repaired(y), Pushable(y)\}$
  $add : \{At(y, Bicycle)\}$
  $del : \{At(y, x), Pushable(y)\}$

  for all $x \in \{A, B, C, D\}$, $y \in \{Frontwheel, Backwheel\}$.

- $FinalCheck(x) =$

  $pre : \{At(Robot, x), At(Bicycle, x), At(Box, x),$
  $At(Frontwheel, Bicycle), At(Backwheel, Bicycle),$
  $Repaired(Frontwheel), Repaired(Backwheel)\}$
  $add : \{Repaired(Bicycle)\}$
  $del : \{\}$

  for all $x \in \{A, B, C, D\}$.
