Künstliche Intelligenz 1 – WS 2018/19

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

February 5, 2019

Contents

1 Assignment 1 (Prolog) – Given Oct. 26., Due Nov. 02.

This exercise sheet is supposed to get you started with Prolog. The SWI ProLog interpreter can be downloaded from `http://www.swi-prolog.org/`. An online interpreter is also avilable at `https://swish.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.

Please hand in your solutions as prolog files that we can run and check.

Here and in general: *Write comments!* They make it a lot easier for your tutors to give points!

Problem 1.1 (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. 40pt

- 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].
  ```

  Hint: You may want to implement a helper method delete(X,LS,RS), that removes all instances of X in LS and returns the result in RS.

- 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].
  ```

  This one is rather tricky. Try it out for yourself *first*. Ask yourself why this is seemingly difficult. Then consider the following hint:

  Hint: Start with a helper predicate takeout(X,LSA,LSB) that is true iff LSB is the result of removing the first occurence of X from LSA. How does this allow you to define the notion of *permutation* recursively?

- The zip function takes two lists with lengths that differ at most by 1, and outputs a list of lists containing one element from the first list and the element with the same index from the other list, possibly followed by a one-element list with the left-over argument.

Create a ProLog predicate with 3 arguments: the first two would be the two lists you want to zip, and the third one would be the result. For instance:

```
?− zip([1,2,3],[4,5,6],L).
L = [[1, 4], [2, 5], [3, 6]].

?− zip([1,2],[3,4,5],L).
L = [[1, 3], [2, 4], [5]].
```

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).

%zip
zip([L],[],[[L]]).
zip([],[L],[[L]]).
zip([A],[B],[[A,B]]).
zip([H1|T1],[H2|T2],L) :− zip(T1,T2,T), append([[H1,H2]],T,L).
```

Problem 1.2 (Binary Tree)

A binary tree of (in this case) natural numbers is inductively defined as a triple tree(n,t1,t2), where n is a natural number and t1 and t2 are themselves binary trees. 60pt

A *leaf* is a terminal node, i.e. a tree of the form tree(n,nil,nil). An example tree in prolog would be:

```
tree(1,tree(2,nil,nil),tree(2,nil,nil))
```

1. Write a ProLog function construct that constructs a binary tree out of a list of (distinct) numbers, such that for every subtree tree(n,t1,t2), all values in t1 are smaller than n and all values in t2 are larger than n. E.g. the list $(5, 2, 4, 1, 3)$ would yield tree(5, tree(2, tree(1, nil, nil), tree(4, tree(3, nil, nil), nil)), nil).
2. 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?

3. 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.)

2 Assignment 2 (PEAS) – Given Nov. 02., Due Nov. 09.

Problem 2.1 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 2.2 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 (first slide of Section 6.4, slide nr. 69). Where not "obvious", justify your choice with a short sentence.

Choose suitable designs for the agents.

Problem 2.3 (Complexity of Loops)

For each of the following program fragments compute the worst-case time complexity (as a function of n). You can assume that ⟪body⟫ is executed in constant time. 30pt

Hint: Write nested summations to express the number of times each of the loop bodies is executed.

1.

```
i, j := 0
while i ≤ n − 1
    i := i + 1
    while j ≤ 10
        j := j + 1
        ⟪body⟫
    end
end
```

2.

```
i := 0
while i ≤ n − 1
    i := i + 1
    ⟪body⟫
end
i := 0
while i ≤ n − 1
    i := i + 1
    j := i
    while j ≤ n − 1
       j := j + 1
       ⟪body⟫
  end
end
```

Solution: We determine the complexities by systematic counting:

1. The outer loop is executed n times and the inner loop is executed 10 times for each execution of the outer loop. So ⟪body⟫ is executed 10 n times.

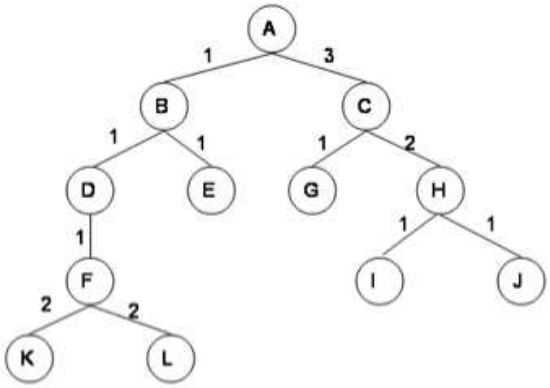
   Another way to think about it the number of times the loop body is executed is given by $\sum_{i=0}^{n-1} (\sum_{j=1}^{10} 1) = \sum_{i=0}^{n-1} 10 = 10\ n$ Thu the time complexity is in $\mathcal{O}(n)$.

2. The first loop has asymptotic time complexity $\mathcal{O}(n)$.. For the pair of nested loops, the loop body is executed $\sum_{i=0}^{n-1} (\sum_{j=0}^{n-1} 1) = \sum_{i=0}^{n-1} n - 1 = (n)n - 1\text{div}2$ So the second set of loops has asymptotic time complexity in $\mathcal{O}(n^2)$. Thus the overall asymptotic time complexity of the entire program segment is in $\mathcal{O}(n^2)$.

---

3 Assignment 3 (Tree Search) – Given Nov. 09., Due Nov. 23.

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	

Solution:

Search Type	List of States
Breadth First	ABCDEG
Depth First	ABDFKLECG
Iterative Deepening	AABCABDECG
Uniform Cost	ABDECFG

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

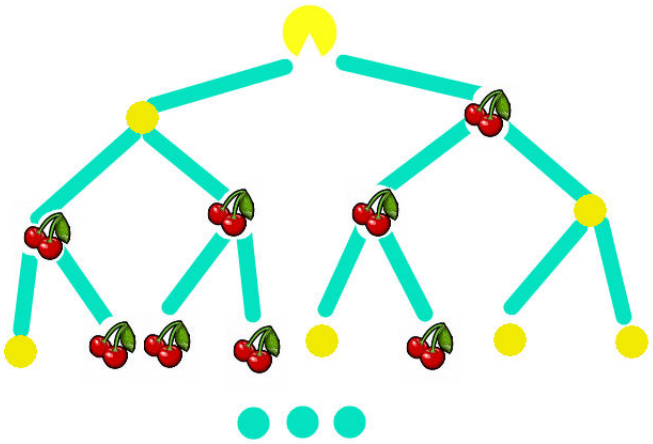
Use the following implementation of trees:

```
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",[(2,tree("B",[])),(3,tree("C",[]))])
```

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

4 Assignment 4 (A*) – Given Nov. 16., Due Nov. 23.

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. 50pt



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 $c(n) =$ *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.

5 Assignment 5 (Adversarial Search I) – Given Nov. 23., Due Nov. 30.

Problem 5.1 (Adversarial Search)

30pt

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

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 an optimal algorithm.

Solution:

Problem 5.3 (Kalah Framework)

Familiarize yourself with the Kalah framework at 0pt

`https://github.com/KWARC/Kalah-Framework`.

Over the course of the next weeks, implement your own agent as teams of maximally 3 students per team. The precise deadline will be announced next week. You can already start implementing your agent – e.g. by implementing methods that simulate a move, or compute the full game tree...

We follow the rules described at the introduction section of `https://en.wikipedia.org/wiki/Kalah` with variable numbers of houses and seeds.

The team with the best agent receives an additional(!) 100 points, the 2nd team 90 points, the 3rd 80 etc.

Please read the instructions and rules specified in the readme of the git repository carefully!

Solution:

6 Assignment 6 (Kalah Tournament) – Given Nov. 30., Due Dec. 09.

Problem 6.1 (Kalah Tournament)

Download the Kalah framework at `https://github.com/KWARC/Kalah-Framework`. Implement your own agent as teams of maximally 3 students per team. 100pt

We follow the rules described at the introduction section of `https://en.wikipedia.org/wiki/Kalah` with variable numbers of houses and seeds.

The team with the best agent receives an additional 100 points, the 2nd team 90 points, the 3rd 80 etc.

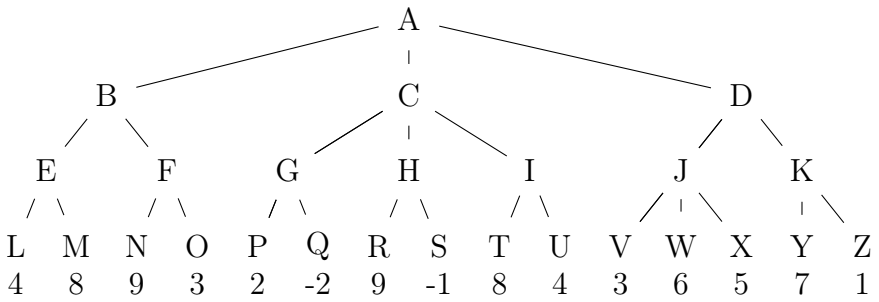
Please read the instructions and rules specified in the readme of the git repository carefully!

Solution:

7 Assignment 7 (Adversarial Search and CSPs) – Given Dec. 07., Due Dec. 14.

Problem 7.1 (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. 50pt



20 pt

1. Label each non-leaf node with its minimax value. See above — 5 pt
2. Which move would be selected by Max? — 15 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. — 10 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 7.2 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 7.3 (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

8 Assignment 8 (CSPs and Propositional Logic) – Given Dec. 14., Due Dec. 21.

Problem 8.1 (Acyclic Constraint Graphs)

Consider the acyclic constraint graph algorithm AcyclicCG from the lecture. 30pt

Prove that this algorithm will always find a solution (assuming one exists) without ever having to backtrack.

Solution: Assume an acyclic constraint graph given, with variable order $v_1 < \ldots < v_n$. Assume a solution exists, but the algorithm needs to backtrack. Then there exists an index k with partial assignment a where the algorithm assigns a value $a(v_k) = x$ such that all extensions of a violate a constraint. Since v_k is arc-consistent relative to all following variables, the assignment $a(v_k) = x$ already violates a constraint. By construction, v_k has a parent v_ℓ with $\ell < k$ such that v_ℓ is arc-consistent relative to v_k. By forward-checking, once v_ℓ is assigned a value, all values in the domain of v_k that violate a constraint with the partial assignment up to v_ℓ (so in particular the value x) are removed, contradiction.

Problem 8.2 (Scheduling CS Classes)

You are in charge of scheduling for computer science classes that meet Mondays, Wednesdays and Fridays. There are 5 classes that meet on these days and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time. The classes are: 40pt

- Class 1 - Intro to Programming: meets from 8:00-9:00am
- Class 2 - Intro to Artificial Intelligence: meets from 8:30-9:30am
- Class 3 - Natural Language Processing: meets from 9:00-10:00am
- Class 4 - Computer Vision: meets from 9:00-10:00am
- Class 5 - Machine Learning: meets from 9:30-10:30am

The professors are:

- Professor A, who is available to teach Classes 3 and 4.
- Professor B, who is available to teach Classes 2, 3, 4, and 5.
- Professor C, who is available to teach Classes 1, 2, 3, 4, 5.

1. Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.

2. Give the constraint graph associated with your CSP

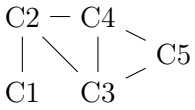
3. Show the domains of the variables after running arc-consistency on this initial graph (after having already enforced any unary constraints).

Solution:

1.

Variables	Domains
C1	C
C2	B,C
C3	A,B,C
C4	A,B,C
C5	B,C

Constraints: C1 $\neq$ C2, C2 $\neq$ C3 , C3 $\neq$ C4, C4 $\neq$ C5, C2 $\neq$ C4, C3 $\neq$ C5

2. 

3.

Variable	Domain
C1	C
C2	B
C3	A,C
C4	A,C
C5	B,C

Note that C5 cannot possibly be C, but arc consistency does not rule it out.

4. C1 = C, C2 = B, C3 = C, C4 = A, C5 = B. One other solution is possible (where C3 and C4 are switched).

Problem 8.3 (PL Semantics)

30pt

Prove whether the following formulae are valid. If not, give an assignment that serves as counterexample.

Use the definition of *interpretation* and *assignments* for your proofs (i.e. specifically no truth tables!)

- $A \Rightarrow (B \Rightarrow A)$
- $(A \wedge B) \Rightarrow (A \wedge C)$

Solution:

- $A \Rightarrow (B \Rightarrow A)$ is valid:

  For any assignment φ:

$$\begin{aligned}\mathcal{I}_\varphi(A \Rightarrow (B \Rightarrow A)) &= \mathcal{I}_\varphi(\neg(A \wedge \neg\neg(B \wedge \neg A)) \\ &= \top \text{ iff } \mathcal{I}_\varphi(A \wedge \neg\neg(B \wedge \neg A)) = \bot \\ &\text{ iff not both } \varphi(A) = \top \text{ and } \mathcal{I}_\varphi(\neg\neg(B \wedge \neg A)) = \top \\ &\text{The latter is the case iff } \mathcal{I}_\varphi(B \wedge \neg A) = \top \\ &\text{ iff } \varphi(B) = \top \text{ and } \varphi(A) = \bot\end{aligned}$$

  So for the formula is false iff both $\mathcal{I}_\varphi(A) = \top$ and $\mathcal{I}_\varphi(A) = \bot$.

- $(A \wedge B) \Rightarrow (A \wedge C)$: Not valid. Counterexample: $\varphi(A) = \varphi(B) = \top$, $\varphi(C) = \bot$.

---

9 Assignment 9 (Propositional Logic and Natural Deduction) – Given Dec. 21., Due Jan. 11.

Problem 9.1 (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. 20pt

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 9.2 (Natural Deduction)

Prove (or disprove) the validity of the following formulae in Natural Deduction: 30pt

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

Solution:

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)

	(1)		$(P \vee \neg P)$	TND
3.	(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)

Problem 9.3 (Completeness)

Assume we have a logic $\mathcal{L}$ that satisfies the contradiction theorem: 50pt

Theorem 2.12: $\mathcal{H} \models \mathbf{A}$ iff $\mathcal{H} \cup \{\neg \mathbf{A}\}$ is unsatisfiable

with a proof calculus $\mathcal{C}$ that satisfies the deduction theorem:

$$\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A} \text{ iff } \mathcal{H} \cup \{\neg \mathbf{A}\} \vdash_{\mathcal{C}} \bot$$

where $\mathcal{H}$ is any set of formulae and $\mathbf{A}$ a single formula.

1. Prove, that the following two properties are equivalent:
   - *Completeness:* For any set of formulae $\mathcal{H}$: $\mathcal{H} \models \mathbf{A}$ iff $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$
   - *Model Existence:* For any set of formulae $\mathcal{H}$: $\mathcal{H}$ has a model iff $\mathcal{H} \nvdash_{\mathcal{C}} \bot$
2. Assume our logic is complete. Prove the compactness theorem, which states:

   A (possibly infinite) set of formule $\mathcal{H}$ has a model if and only if every *finite* subset of $\mathcal{H}$ has a model.

Hint: Proofs in a calculus are always of finite length.

Solution:

1. Let $\mathcal{H}$ be any set of formulae and $\mathbf{A}$ any formula. Consider the following diagram:

$$\begin{array}{ccccc} & & \text{Completeness} & & \\ & \mathcal{H} \models \mathbf{A} & \Leftrightarrow & \mathcal{H} \vdash_{\mathcal{C}} \mathbf{A} & \\ \text{Contradiction} & \Updownarrow & & \Updownarrow & \text{Deduction} \\ & \mathcal{H} \cup \{\neg \mathbf{A}\} \text{ is unsatisfiable} & \Leftrightarrow & \mathcal{H} \cup \{\neg \mathbf{A}\} \vdash_{\mathcal{C}} \bot & \\ & & \text{Model Existence} & & \end{array}$$

   More detailed:

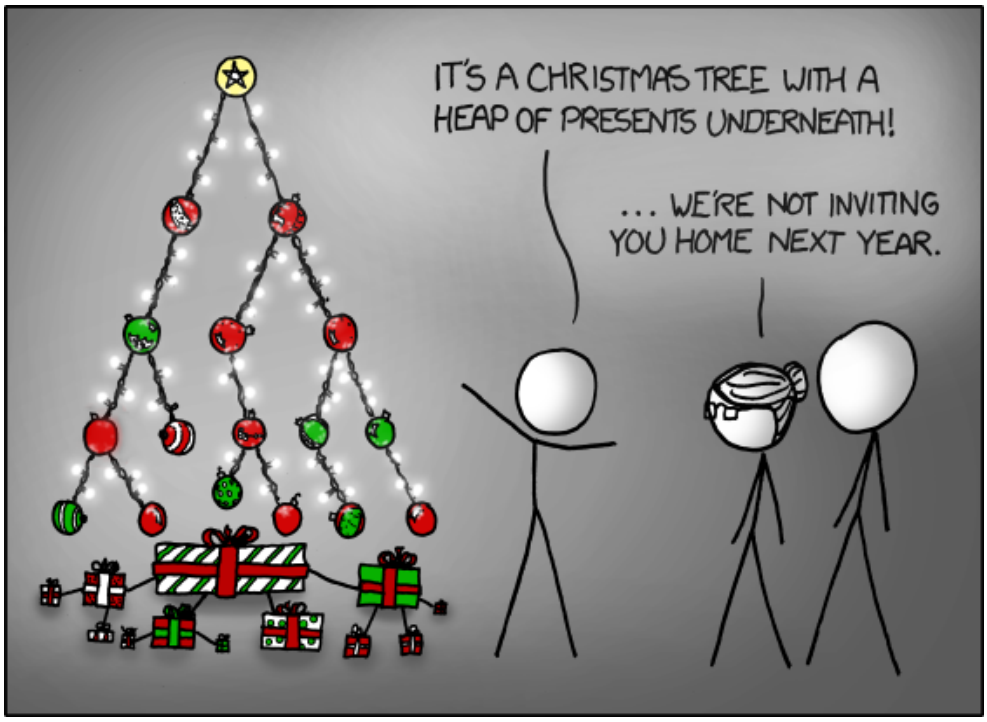
Assume $\mathcal{C}$ complete. Assume $\mathcal{H} \vdash_{\mathcal{C}} \bot$, then by completeness $\mathcal{H} \models \bot$, i.e. every model implies a contradiction in itself, hence no model can exist. Hence if $\mathcal{H}$ has a model, then $\mathcal{H} \nvdash_{\mathcal{C}} \bot$.

Conversely, if $\mathcal{H}$ has no model, then trivially for any proposition $\mathbf{A}$ we have $\mathcal{H} \models \mathbf{A}$ (since "all models satisfy $\mathbf{A}$"), so in particular $\mathcal{H} \models \bot$. By completeness we get $\vdash_{\mathcal{C}} \bot$.

Now assume model existence holds. Then $\mathcal{H} \models \mathbf{A}$ if and only if (by the contradiction theorem) $\mathcal{H} \cup \{\neg \mathbf{A}\}$ is unsatisfiable, i.e. has no model. By model existence, this is the case if and only if $\mathcal{H} \cup \{\neg \mathbf{A}\} \vdash_{\mathcal{C}} \bot$, and hence if and only if (by the deduction theorem) $\mathcal{H} \vdash_{\mathcal{C}} \mathbf{A}$.

2. Any model of $\mathcal{H}$ is in particular a model of any finite subset of $\mathcal{H}$.

   Conversely, assume $\mathcal{H}$ has *no* model. By completeness/model existence we have $\mathcal{H} \vdash_{\mathcal{C}} \bot$. Proofs in $\mathcal{C}$ are finite, so the set P of all premises in the proof of $\bot$ is a finite subset of $\mathcal{H}$ with $P \vdash_{\mathcal{C}} \bot$, so by completeness/model existence P has no model.



Happy Holidays!

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

10 Assignment 10 (Propositional Logic Calculi) – Given Jan. 11., Due Jan. 18.

Problem 10.1 (Calculi Comparison)

Prove (or disprove) the validity of the following formulae in i) Natural Deduction ii) Tableau and iii) Resolution iv) semantically, by rewriting the formulae using equivalences until you arrive at an obvious tautology. 70pt

For Natural Deduction, you can skip the ones you proved in your last assignment.

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:

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

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

Problem 10.2 (DNF)

A formula is in **conjunctive normal form (CNF)**, if it is of the form $\bigwedge_i \bigvee_j (\neg) A_{ij}$. As you know, CNF formulae are needed for resolution. 30pt

A formula is in **disjunctive normal form (DNF)**, if it is of the form $\bigvee_i \bigwedge_j (\neg) A_{ij}$.

Prove, that the problem of transforming a given formula into disjunctive normal form is NP hard.

Hint:

1. Remember that the problem of finding an assignment for an arbitrary formula is NP hard (SAT Problem).
2. Any problem that *subsumes* the SAT problem is correspondingly NP hard as well ("subsumes" in the sense of: The SAT problem can be reduced to the given problem in polynomial time).
3. Why is it easy to find an assignment for a formula in DNF?

Solution: Given a formula in DNF, a valid assignment only needs to satisfy any of the conjunctive clauses. A conjunctive clause $\bigwedge_i L_i$ is satisfiable iff it does not contain two literals of the forms A and $\neg A$. We can wlog assume that an algorithm that computes a DNF for a given formula does not produce clauses that contain two such literals (this point is contestable, if this assumption increases the complexity class!).

If a conjunctive clause is satisfiable, the literals in the clause yield a valid partial assignment that can be arbitrarily extended on the variables not occuring in the clause. In particular, to solve a DNF we only need to consider the first clause, whose length is bounded by the number of propositional variables in the original formula, hence this can be easily checked in polynomial time.

---

11 Assignment 11 (First Order Logic) – Given Jan. 18., Due Jan. 25.

Problem 11.1 (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 $\mathcal{I}_{\varphi_2}(\mathbf{A}) = \top$.

Problem 11.2 (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. 30pt

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.

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

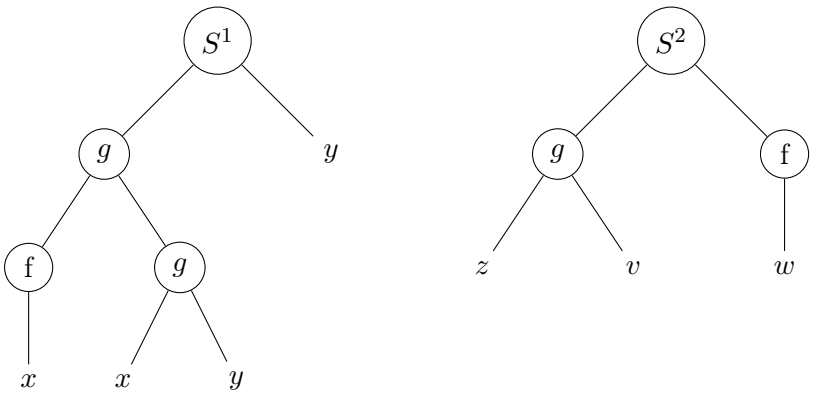
12 Assignment 12 (First Order Logic and Planning) – Given Jan. 25., Due Feb. 01.

Problem 12.1 (Unification)
Decide whether (and how or why not) the following pairs of terms are unifiable. 20pt
$S_1 \in \Sigma_2^p, S_2 \in \Sigma_3^p, f \in \Sigma_1^f, g \in \Sigma_2^f, c \in \Sigma_0^f$

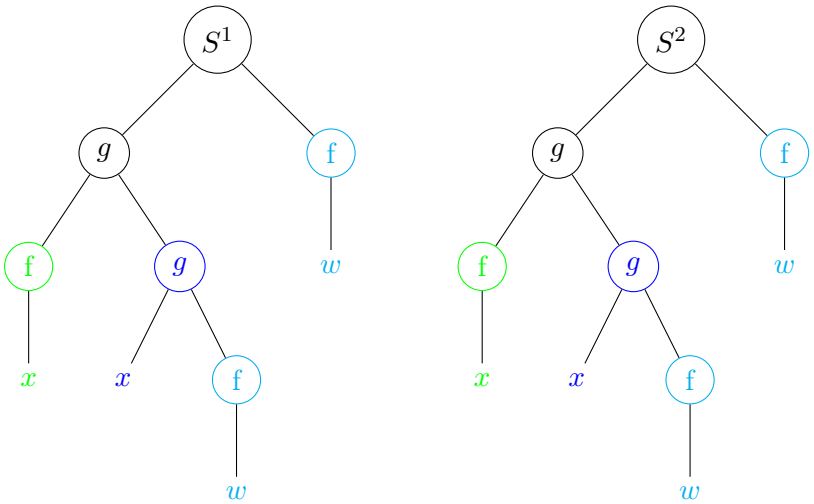
1. $S_1(g(f(x), g(x, y)), y)$ and $S_1(g(z, v), f(w))$
2. $S_2(g(f(x), g(x, u)), f(y), z)$ and $S_2(g(g(g(u, v), f(w)), f(c)), f(g(u, v)), f(c))$

Solution:

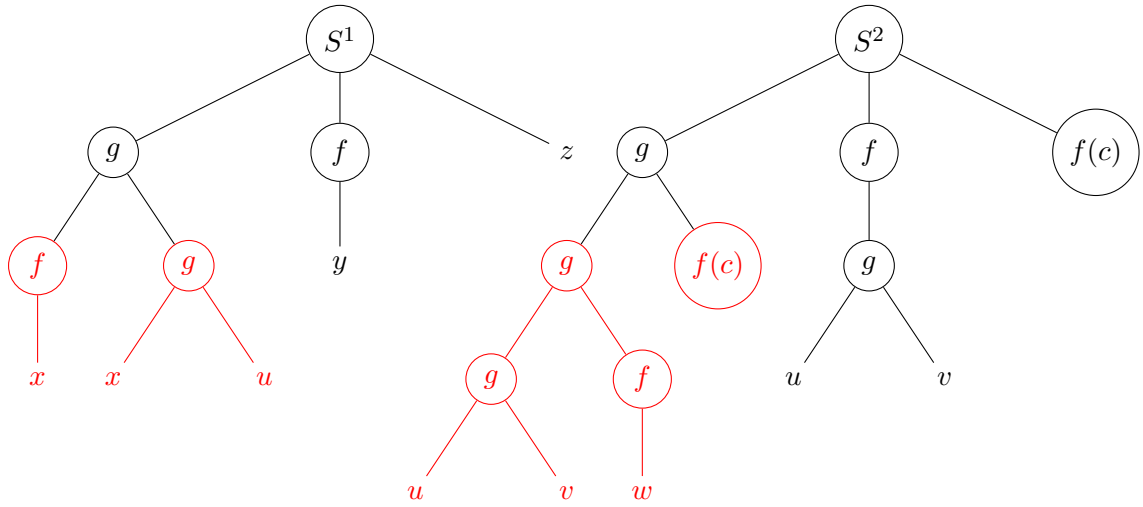
1. The term trees look like this:



Obviously, we need to perform the following substitutions to make the two trees equal:



2. The term trees look like this:



Obviously, the red subtrees can't be unified.

Problem 12.2 (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 12.3 (First-Order Tableaux)
Prove the following formula using the first-order free variable tableaux calculus. We have 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)	$\bot[c_Y/V_X]$	

Problem 12.4 (STRIPS) 30pt

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.

We use the following predicates:

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

The intial state is $I = \{Jug_p(0), Jug_q(0)\}$ and the goal state $G = Jug_p(k)$.

Give the pre, add and del lists for the following actions:

- $Empty_p/Empty_q$: Empties jug p/q completely
- $FillUp_p/FillUp_q$: Fill up jug p/q fully
- For all x, y with $0 \leq x \leq p,\ 0 \leq y \leq q$:
  $Fillp_{x,y}/Fillq_{x,y}$: pour the contents of jug q/p into jug p/q until the former is empty or the latter is full.

Solution:

- $Empty_p$: $\mathsf{pre} = \{\}, \mathsf{add} = \{Jug_p(0)\}, \mathsf{del} = \{Jug_p(n) \mid 1 \leq n \leq p\}$
- $Empty_q$: $\mathsf{pre} = \{\}, \mathsf{add} = \{Jug_q(0)\}, \mathsf{del} = \{Jug_q(n) \mid 1 \leq n \leq q\}$
- $FillUp_p$: $\mathsf{pre} = \{\}, \mathsf{add} = \{Jug_p(p)\}, \mathsf{del} = \{Jug_p(n) \mid 0 \leq n < p\}$
- $FillUp_q$: $\mathsf{pre} = \{\}, \mathsf{add} = \{Jug_q(q)\}, \mathsf{del} = \{Jug_q(n) \mid 0 \leq n < q\}$
- 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 $m = \mathsf{min}(x + y, q)$:
  $Fillq_{x,y}$:

$$\mathsf{pre} = \{Jug_p(x), Jug_q(y)\},$$
$$\mathsf{add} = \{Jug_p(m - y), Jug_q(m)\},$$
$$\mathsf{del} = \{Jug_p(z) \mid z \neq m - y\} \cup \{Jug_q(z) \mid z \neq m\}$$
