Problem Set 2 Solutions
CSE 473

Krzysztof Gajos
kgajos@cs.washington.edu

May 4, 2004

1 Abalone

Sorry, I will fill this one later.

2 CSP

There are a number of ways to solve this problem in a reasonable way. Here is my suggestion:

For each quarter of your ugrad career, we can have three variables (assuming that you take three classes per quarter). The domain of each variable contains all available courses. That is, if you started school in the Fall of 2000, the first three variables could be `au00-1`, `au00-2` and `au00-3`. To each of these variables you can assign any course from the course catalog. Now, assuming that you have the right vocabulary, you can propose the following kinds of constraints:

- you have to take at least 10 CSE courses (this is an n-ary constraint that applies to all viariables)
- you have to take CSE 473 (again, an n-ary constraint that makes sure that at least one variable is set to CSE 473
- you cannot take the same class twice (n-ary constraint)
- you have to tak CSE 143 before you take CSE 473 (this one is tricky to represent: you could implement it as a very complex n-ary constraint where you analyze the assignments of classes to quarters, and reject any assignments where it is impossible to take CSE 143 before AI; alternatively you can install $O(n^2)$ binary constraints between every pair of variables – this is the theoretically cleaner but practially less efficient approach)

Another way to solve this problem is to make each course a variable, and the domain of each variable will be the list of quarters while you are in school or the "null" quarter (when this quarter is selected as the value for a subject, then you will never take the subject). This formulation, although slightly less intuitive to some, will make some constraints easier. For example, it is easier now to express the constraint that says that CSE 473 has to be taken after CSE 143 (now it is a binary constraint between the two variables representing these subjects). However, you need to

add a new type of constraint that says that each quarter you need to take at least two but no more than four classes. This, would most easily be expressed as an ugly n-ary constraint.

You can add furter constraints, such as that you need to take at least one class from Dan Weld, at least one TA-ed by Tessa and at least two whose code is a prime number.

3 AIMA 7.8

Answers: valid, satisfiable, valid

4 Fred

First, convert everything to CNF. I further chose to use the set notation we discussed in class. Attempting to show that what we know entails Fred being rich, we will try to show the contrary and derive contradiction. In this case, all I do is pick out unit clauses and keep eliminating clauses and literals as follows:

{(¬R,H) (R,¬H) (R,L) (¬H,F) (¬L,F) (¬F,S) (¬R)}

set R = false

{ (¬H) (L) (¬H,F) (¬L,F) (¬F,S) }

set L = true

{ (¬H) (¬H,F) (F) (¬F,S) }

set H = false

{ (F) (¬F,S) }

set F = true

{ (S) }

set S = true

{ }

You could have used resolution or anything else you want. As you see, we do not get a contradiction. Which means that what we know about Fred does not entail that he is rich but it does not entail the oposite either! All it says that there exists a consistent world, in which Fred is not rich but it does not say that Fred can never be rich.

We use the same methods to show that Fred is smelly:

{(¬R,H) (R,¬H) (R,L) (¬H,F) (¬L,F) (¬F,S) (¬S)}

set S = false

{(¬R,H) (R,¬H) (R,L) (¬H,F) (¬L,F) (¬F) }

set F = false

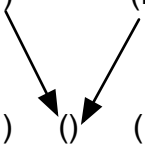
{(¬R,H) (R,¬H) (R,L) (¬H) (¬L) }

set L = false

{(¬R,H) (R,¬H) (R) (¬H) }

set H = false

{(¬R) (R) }



resolve

{(¬R) () (R) }

Notice that this time we finish by performing resolution on two unit clauses and we end up with an empty clause, i.e. contradiction! Excellent, even if Fred is not guaranteed to be rich, we know he is smelly. An important thing to observe her is that resolution does not eliminate the clauses that participate in it! It only produces new clauses.

5 FOL

(a)

one predicate needs explaining: fools(fooler, foolee, time)

(b)

$\forall_x isPolitician(x)\,(\exists_p \forall_t fools(x,p,t) \wedge \forall_p \exists_t fools(x,p,t) \wedge \exists_{p,t} \neg fools(x,p,t))$

(c)

one predicate needing explaining: tookClass(student, nameOfClass, academicQuarter)

(d)

$\exists_x isStudent(x) \wedge tookClass(x, \text{“}AI''\text{, “}sp04'')$

(e)

one unusal predicate: isScoreInClass(variable, nameOfClass, academicQuarter)

(f)

$\forall_{q,o} \exists_a isAcademicQtr(q) \wedge isScoreInClass(o, \text{“}OS'', q) \wedge isScoreInClass(a, \text{“}AI'', q) \wedge (a > o)$