Informatics 2D: Tutorial 4

Satisfiability and First-Order Logic

Week 5

1 DPLL algorithm

The DPLL algorithm consists of the following steps:

- Convert proposition to CNF
- Loop through the following until a satisfying assignment is found or none is possible:
  - Loop through the following simplifications until the formula can't be simplified anymore:
    - Pure literal heuristic.
    - Unit Clause heuristic.
  - Select a variable and branch the search space into a formula where the variable is true and a formula where the variable is false. (This means that you try the algorithm recursively upon these new formulae, with a satisfying assignment for one of the new formula being a satisfying assignment for the original).

Your lecture notes and R&N chapter 7 section 6 describe the steps in more detail.

Question: Use the DPLL algorithm to show whether the following propositional formulae is satisfiable:

$S_{1,1} \wedge (S_{1,1} \Leftrightarrow W_{1,2} \vee W_{1,1} \vee W_{2,1}) \wedge \neg((W_{1,2} \wedge P_{1,2}) \vee (W_{2,1} \wedge P_{2,1})) \wedge \neg P_{1,1} \wedge \neg((W_{1,1} \wedge W_{2,1}) \vee (W_{1,1} \wedge W_{1,2}))$

Answers

The proposition consists of the following conjuncts:

- $S_{1,1}$
- $\neg P_{1,1}$

- $S_{1,1} \Leftrightarrow W_{1,2} \vee W_{1,1} \vee W_{2,1}$
- $\neg((W_{1,2} \wedge P_{1,2}) \vee (W_{2,1} \wedge P_{2,1}))$
- $\neg((W_{1,1} \wedge W_{2,1}) \vee (W_{1,1} \wedge W_{1,2}))$

Converting each into CNF:

$S_{1,1} \Leftrightarrow W_{1,2} \vee W_{1,1} \vee W_{2,1}$

- $S_{1,1} \Rightarrow W_{1,2} \vee W_{1,1} \vee W_{2,1} \wedge W_{1,2} \vee W_{1,1} \vee W_{2,1} \Rightarrow S_{1,1}$
- $(\neg S_{1,1} \vee W_{1,2} \vee W_{1,1} \vee W_{2,1}) \wedge ((\neg W_{1,2} \wedge \neg W_{1,1} \wedge \neg W_{2,1}) \vee S_{1,1})$
- $(\neg S_{1,1} \vee W_{1,2} \vee W_{1,1} \vee W_{2,1}) \wedge (\neg W_{1,2} \vee S_{1,1}) \wedge (\neg W_{1,1} \vee S_{1,1}) \wedge (\neg W_{2,1} \vee S_{1,1})$

$\neg((W_{1,2} \wedge P_{1,2}) \vee (W_{2,1} \wedge P_{2,1}))$

- $\neg(W_{1,2} \wedge P_{1,2}) \wedge \neg(W_{2,1} \wedge P_{2,1})$
- $(\neg W_{1,2} \vee \neg P_{1,2}) \wedge (\neg W_{2,1} \vee \neg P_{2,1})$

$\neg((W_{1,1} \wedge W_{2,1}) \vee (W_{1,1} \wedge W_{1,2}))$

- $(\neg W_{1,1} \vee \neg W_{2,1}) \wedge (\neg W_{1,1} \vee \neg W_{1,2})$

So the CNF formula has the following conjuncts:

- $S_{1,1}$
- $\neg P_{1,1}$
- $\neg S_{1,1} \vee W_{1,2} \vee W_{1,1} \vee W_{2,1}$
- $\neg W_{1,2} \vee S_{1,1}$
- $\neg W_{1,1} \vee S_{1,1}$
- $\neg W_{2,1} \vee S_{1,1}$
- $\neg W_{1,2} \vee \neg P_{1,2}$
- $\neg W_{2,1} \vee \neg P_{2,1}$
- $\neg W_{1,1} \vee \neg W_{2,1}$
- $\neg W_{1,1} \vee \neg W_{1,2}$

Which has pure literals $\neg P_{1,1}$, $\neg P_{1,2}$ and $\neg P_{2,1}$ and the unit clauses with literals $S_{1,1}$ and $\neg P_{1,1}$.

So we assign $S_{1,1}$ to true and $P_{1,1}$, $P_{1,2}$ and $P_{2,1}$ to false which, using early termination, leaves us with:

- $W_{1,2} \vee W_{1,1} \vee W_{2,1}$
- $\neg W_{1,1} \vee \neg W_{2,1}$
- $\neg W_{1,1} \vee \neg W_{1,2}$

There are no more simplifications to make so we must pick a literal and branch on it, choosing $W_{1,2}$ and assigning it to true leaves:

- $\neg W_{1,1} \vee \neg W_{2,1}$
- $\neg W_{1,1}$

Assigning $W_{1,1}$ to false, since $\neg W_{1,1}$ is a unit clause literal, satisfies the remaining clauses.

This gives us an assignment of $S_{1,1}$ and $W_{1,2}$ to true and $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $W_{1,1}$, and to false. Note that we have not assigned $W_{2,1}$ to either true or false, since in either case we have a satisfying assignment.

2 First-Order Logic

Part 1: Represent the following sentences in first-order logic. You will have to define a vocabulary (which should be consistent between sentences).

1. Some students took French in spring 2001.
2. Every student who takes French passes it.
3. Only one student took Greek in spring 2001.
4. The best score in Greek is always higher than the best score in French.
5. There is a male barber who shaves all the men who do not shave themselves.

Part 2: Write down a first-order logic sentence such that every world in which it is true contains exactly one object.

2.1 Answer

Part 1: An example vocabulary might be:

- $French$ - a constant denoting the subject French
- $Greek$ - a constant denoting the subject Greek

- $student/1$ - a unary relation, $student(x)$ iff constant x is a student (unnecessary if the $took$ relation is defined to only apply to students)
- $took/3$ - a ternary relation, $took(x, y, t)$ iff x took the subject y during time interval t (there are alternatives to having time as an argument to the $took$ relation. You could represent this as a course event, e.g. $course(e) \wedge during(e, t) \wedge took(x, e) \wedge subject(y, e)$)
- $pass/2$ - a binary relation, $pass(x, y)$ iff x passes the subject y
- $Spring2001$ - a constant denoting the time interval spring 2001
- $bestScore/2$ - a binary function which identifies the best score in a subject during a given time interval.
- $greaterThan/2$ - a binary relation which has the same meaning as $>$
- $equals/2$ - a binary relation which has the same meaning as $=$ (note that it's acceptable to assume that we are using first-order logic with equality, provided that the student knows what this means).
- $numOfStudents/2$ - a binary function which identifies the number of students in taking a subject during a given time interval.
- $barber/1$ - a unary relation, $barber(x)$ iff x is a barber.
- $shaves/2$ - a binary relation, $shaves(x, y)$ iff x shaves y.

Given this vocabulary you can represent the sentences as follows:

1. $\exists x.student(x) \wedge took(x, French, Spring2001)$
2. $\forall x, t.student(x) \wedge took(x, French, t) \Rightarrow pass(x, French)$ (this is slightly vague, since a student could fail and then pass on the second attempt)
3. $equals(numOfStudents(Greek, Spring2001), 1)$ (the challenge here is to represent the cardinality of the set of students taking Greek during spring 2001). An alternative is $\exists x.student(x) \wedge took(x, Greek, Spring2001) \wedge (\forall y.took(y, Greek, Spring2001) \Rightarrow x = y)$. This formula asserts that there exists a student who took Greek in spring 2001 and if there is anything else which took Greek in spring 2001 then it must be this student. So it would be impossible to satisfy this sentence if less than one student took Greek in spring 2001, since we have asserted the existence of at least one student with this property. But also impossible to satisfy it if more than one student took Greek in spring 2001, since in that case there would be a y such that $took(y, Greek, Spring2001) \wedge y \neq x$.
4. $\forall t.greaterThan(bestScore(Greek, t), bestScore(French, t))$
5. $\exists x.barber(x) \wedge \forall y.\neg shaves(y, y) \Rightarrow shaves(x, y)$ - (almost) Russell's paradox, there is no barber with this property as if there was then it would be possible to prove that the

$shaves(Barber, Barber)$ and $\neg shaves(Barber, Barber)$ (N.B. Russell's paradox is that there is a barber who shaves all *and only* the men who do not shaves themselves - this is an equivalence, $\Leftrightarrow$, rather than an implication)

Part 2: One possible sentence is $\forall x.P(x) \wedge \neg\exists x.x \neq A \wedge P(A)$; which means that for all objects property P holds and there are no objects not equal to object A such that property P holds. So if this sentence is true then there can only be one object, A, in the domain of interpretation. If there were any other objects then they would have to have property P and not have property P in order to satisfy this sentence, which is impossible. A simpler alternative is $\exists x\forall y.x = y$; which means that there exists an object such that all other objects are equivalent to this object, so there is only one unique object in the domain.

3 Most General Unifier (MGU)

The most general unifier (MGU) is the least constrained substitution that makes two clauses unify with each other. What is the MGU for each pair of clauses below? If there is no MGU, explain why.

The Unify algorithm in figure 1 (also in R&N Section 9.2, p.328.)

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
    y, a variable, constant, list, or compound
    θ, the substitution built up so far (optional, defaults to empty)

  if θ = failure **then return** failure
  else if $x = y$ **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  else return failure

---

function UNIFY-VAR(var, x, θ) **returns** a substitution
  inputs: var, a variable
    x, any expression
    θ, the substitution built up so far

  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ

Figure 1: Unification Algorithm.

1. p(A, B, B) and p(x, y, z)
2. q(y, g(A, B)) and q(g(x, x), y)
3. older(father(y), y) and older(father(x), John)
4. knows(father(y), y) and knows(x, x)

Note that, constants are upper case (e.g. A, B) and variables are lower case (e.g. x, y, z).

Answer

1. x/A, y/B, z/B
2. Unification fails. Start with the (partial) substitution y/g(x,x), then add x/A to get y/g(x,x), x/A. At this point, one clause is q(g(A, A), g(A, B)), and the other q(g(A, A), g(A, A)). Since q(A, A) cannot be unified with q(A,B) unification fails.
3. x/John, y/John
4. Unification fails. Start with (partial) substitution x/father(y). One clause is now knows(father(y),y), and the other knows(father(y),father(y)). Unification fails here because we can't unify y and father(y), due to the occurs check.