CSC110 Fall 2021 Assignment 2: Logic, Constraints, and Nested Data

Hung, Shou-Yi (Ray), Scott Cui

October 3, 2021

Part 1: Predicate Logic

1. 1. If D_1 is the set of all Natural Numbers, then it's possible to only satisfy one statement. In this case, statement 1 will be satisfied, but statement 2 will not. Because for every number you pick in the Natural Numbers set, you can always find a number that is greater than it just by adding 1. But for the number 1 (or 0 depending on how you start the Natural Numbers), you cannot find a number that is smaller than it (Statement 2).
   2. If D_2 is the set of all Integers, including negatives, then it's possible to satisfy both. In this case, statement 1 will be satisfied because you can always find a number (y) greater than any number (x) you pick from the set. You can also always find a number (x) that is smaller than any number (y) you pick from the set.
   3. If D_3 is a set with only 1 element, then it's possible to falsify both statements. Pick $D_3 = 1$, then for statement 1 and 2, we cannot find a y or x that is greater than or lesser than it. So both statements automatically becomes false.

2. 1. $P(x) : |x| >= 10$
   2. $Q(x) : |x| >= 5$

   For Statement 3, $\forall x \in S$, condition P is always not satisfied and condition Q will only sometimes be satisfied. Which makes it false.

   However, for Statement 4, $\forall x \in S$, condition P will still not be satisfied always, but since this is an implication statement, if the hypothesis is false, then the whole implication statement will be true regardless of what $Q(x)$ is .

   Therefore, in the two predicates I've listed, one of the statements will be true and the other will be false.

3. Complete this part in the provided `a2_part1.py` starter file. Do **not** include your solution in this file.

4. Complete this part in the provided `a2_part1.py` starter file. Do **not** include your solution in this file.

Part 2: Conditional Execution

Complete this part in the provided `a2_part2.py` starter file. Do **not** include your solution in this file.

Part 3: Generating a Timetable

1. Complete this part in the provided `a2_part3.py` starter file. Do **not** include your solution in this file.

2. (a) *IMPORTANT DEFINITIONS/NOTATION* (don't change this text!)

   We define the following sets:

   - C: the set of all possible courses
   - S: the set of all possible sections
   - M: the set of all possible meeting times
   - SC: the set of all possible schedules

   We also define the following notation for expressions involving the elements of these sets:

   - The first three (courses/sections/meeting times) are represented as tuples (as described in the assignment handout), and you can use the indexing operation on these values. For example, you could translate "every section term is in $\{'F', 'S', 'Y'\}$" into predicate logic as the statement:

   $$\forall s \in S,\ s[1] \in \{'F', 'S', 'Y'\}$$

   - The start and end times of a meeting time can be compared chronologically using the standard $<$, $\leq$, $>$, and $\geq$ operators.
   - For a section $s \in S$, $s[2]$ represents a tuple of meeting times. You may use standard set operations and quantifiers for these tuples (pretend they are sets). For example, we can say:
     - $\forall s \in S,\ s[2] \subseteq M$
     - $\forall s \in S,\ \forall m \in s[2],\ m[1] < m[2]$
   - Finally, for a schedule $sc \in SC$, you can use the notation $sc.sections$ to refer to a set of all sections in that schedule. You can use quantifiers with that set of schedules as well, e.g. $\forall s \in sc.sections,\ ...$

   Predicate for meeting times conflicting:

   $$\begin{aligned} MeetingTimesConflict(m_1, m_2) : (m_1[0] = m_2[0]) \wedge \\ ((m_1[2] > m_2[1] \wedge m_2[2] > m_1[1]) \vee (m_2[2] > m_1[1] \wedge m_1[2] > m_2[1])), \\ \text{where } m_1, m_2 \in M \end{aligned}$$

   Predicate for sections conflicting:

   $$\begin{aligned} SectionsConflict(s_1, s_2) : (s_1[1] = s_2[1] \vee (s_1[1] \in \{'Y'\} \vee s_2[2] \in \{'Y'\})) \wedge (\exists x \in s_1[2], \exists y \in s_2[2], \\ MeetingTimesConflict(x, y)), \\ \text{where } s_1, s_2 \in S \end{aligned}$$

   Predicate for valid schedule:

   $$\begin{aligned} IsValidSchedule(sc) : \forall x \in sc.sections, \forall y \in sc.sections : x \neq y, \neg SectionsConflict(x, y), \\ \text{where } sc \in SC \end{aligned}$$

   (b) Complete this part in the provided `a2_part3.py` starter file. Do **not** include your solution in this file.

3. (a) You may use all notation from question 2(a). Note that a course $c \in C$ is a tuple, and $c[2]$ is a set of sections, and so can be quantified over: $\forall s \in c[2], ...$

   Predicate for section-schedule compatibility:

   $$\begin{aligned} IsCompatibleSection(sc, s) : \forall x \in sc.sections, \neg SectionsConflict(x, s) \\ \text{where } sc \in SC, s \in S \end{aligned}$$

Predicate for course-schedule compatibility:

$$IsCompatibleCourse(sc, c) : \exists x \in c[2], IsCompatibleSection(sc, x)$$
$$\text{where } sc \in SC, c \in C$$

(b) Complete this part in the provided `a2_part3.py` starter file. Do **not** include your solution in this file.

Part 4: Processing Raw Data

Complete this part in the provided `a2_part4.py` starter file. Do **not** include your solution in this file.