Group 3: Downtown Dining

Amy Hong (19YH94)

Eric Lam (19EWHL1)

Lucy Woloszczuk (19LEW3)

Wesley Lam (19WESW)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

October 30, 2021

Abstract

In this project, logic will be used to model the selection of an ideal restaurant among a party that includes 2-8 members. This model will use a priority system to account for both the dietary restrictions(higher priority) and the preferences(lower priority) for each group member.

Propositions

Party size: N - The number of people in the party, N can contain values 2-8
Time: t - The hour that the party needs to eat, can contain values 0 - 23
Weekday: d - The weekday that the party needs to eat, can contain values 1-7
Allowable - $A(r)$: The set of all restaurants that are allowed.

Person (restrictions, preferences): Represents each person in the group students
Restrictions (dietary, price): The list of conditions that a restaurant must satisfy

Dietary: all restrictions related to food - can be vegetarian, dairy-free, halal

- V_p: is true if a person is vegetarian, p represents the person
- D_p: is true if a person is lactose intolerant
- H_p: is true if a person requires halal food
- V_r: is true if a restaurant has vegetarian options, r represents the restaurant
- D_r: is true if a restaurant has dairy-free options
- H_r: is true if a restaurant has halal options

Price: Preferred price sign displayed by Google ($, $$, $$$, $$$$)

- P_{rj} : is true if the dollar sign on Google matches the person's preferred price, where j represents the number of dollar signs

Rating: How many stars does the person require the restaurant to have on Google - can be 1, 2, 3, 4, 5

- R_{pk}: is true if the number of stars is greater than or equal to k, where k is the number of stars

Seating: Does the person want patio seating?

- S_p: is true if patio seating is wanted, false if not

Location: The location of the person or restaurant

- L_{pd}: Is true if the person is located at location "d", where "d" is the distance from the starting location.

Price: Price sign displayed by google ($, $$, $$$, $$$$)

- P_{rk}: is true if the restaurant bill per person is lower than j, where j is the bill price per person

Rating: How many stars the restaurant has on Google - can be 1, 2, 3, 4, 5

- R_{rk}: is true if the number of stars is equal to k, where k is the number of stars

Seating: Is patio seating available?

- S_r: is true if patio seating is available, false if not

Location: The location of the restaurant is on Princess

- L_{ra}: Is true if the restaurant is located at location "a", where "a" is the address.

Open: Is the restaurant considered open?

- O_r: Is true if the restaurant is currently open.

Open Time: Is the restaurant open during the hour where the party eats?

- OT_{rt}: Is true if the restaurant is open at time t (The hour that the party needs to eat)

Open Day: Is the restaurant open during the weekday where the party eats?

- OD_{rd}: Is true if the restaurant is open on the day, d (The day that the party needs to eat)

Weather: The current weather conditions

- *Sunny*: Is true if it is sunny
- *Raining*: Is true if it is raining
- *Snowing*: Is true if it is snowing

Constraints

Rating Constraints:

- Each restaurant can only have one rating at a time
  k = 1 = 1 star, k = 2 = 2 star, ..,
  $\forall r(R_{r,k} \rightarrow \neg R_{r,k+1} \wedge \neg R_{r,k+2}... \wedge \neg R_{r,k-1}...)$

Price Constraints:

- Each restaurant can only have one price at a time
  k = 1 = $, k = 2 = $$, k = 3 = $$$, k = 4 = $$$$
  $\forall r(P_{r,k} \rightarrow \neg P_{r,k+1} \wedge \neg P_{r,k+2}... \wedge \neg P_{r,k-1}...)$

Party Size Constraints:

- The restaurant must have enough seats for the given party size, N
  Restaurant(seating) $\rightarrow$ ($\neg$restaurant(seating, 0) $\vee$ $\neg$restaurant(seating, 1) $\wedge$ $\neg$restaurant(seating, 2) $\vee$... $\neg$restaurant(seating, (N-1)) $\wedge$ (restaurant(seating, N) $\vee$ restaurant(seating, N+ 1) $\vee$... $\vee$ restaurant(seating, N + (8-n))) for all available seating x

Location Constraints:

- Each restaurant has exactly one unique address a
  $\exists a L_{r,a} \wedge \forall b(L_{r,b} \rightarrow a = b)$
- Each restaurant must be located on n Princess St. where n is the address number
  $L_{r,a} \rightarrow a =$ "n Princess St."

Opening constraints:

- Restaurants must be open on the day and hour the party eats.
  $\forall$r $(O_r \rightarrow (OD_{rd} \wedge OT_{rt}))$

Dietary Constraints:

- All allowable restaurants must not violate any dietary constraints. For example, if a person in the group is vegetarian all allowable restaurants must be vegetarian.
  $\forall r \exists p(A(r) \rightarrow ((V_p \rightarrow V_r) \wedge (D_p \rightarrow D_r) \wedge (H_p \rightarrow H_r))$

Weather Constraints:

- If it is sunny outside, it cannot be raining
  $Sunny \rightarrow \neg Raining$
- If it is raining or snowing, patio seating is not available
  $(Raining \vee Snowing) \rightarrow \neg S_r$

Model Exploration

List all the ways that you have explored your model – not only the final version, but intermediate versions as well. See (C3) in the project description for ideas.

One of our Service propositions wasn't coming up in any solutions unless specifically required to true (it instead defaults to False). We tried implementing different workarounds but currently stumped.

```python
@constraint.at_least_one(E)
@proposition(E)
class Service:

    def __init__(self, data):
        self.data = data

    def __repr__(self):
        return f"{self.data}"


# dine-in, take-out, delivery
e = Service("Eat-in")
t = Service("Take-out")
u = Service("Delivery")


# Build an example full theory for your setting and return it.
#
#  There should be at least 10 variables, and a sufficiently large formula to describe it (>50 operators)
#  This restriction is fairly minimal, and if there is any concern, reach out to the teaching staff to cla
#  what the expectations are.
def solution():
    # E.add_constraint(v | g | d | h | ~h | ~v | ~g | ~d)
    # E.add_constraint(~(~e & ~t & ~u))
    E.add_constraint((e & (i | o)) | (~e & ~i & ~o))  # eat-in means there must be either indoor or outdoo
    E.add_constraint(f >> t)  # fast-food restaurants have take-out

    # TODO: code for loop to add constraints

    # E.add_constraint(r3 >> (r3 | r4 | r5))
    # E.add_constraint(r3 & p2)

    return E
```

(0.1)

We also began thinking about how we could eliminate restaurants from our solution based on certain constraints. Our idea was that we could automatically create a proposition corresponding to each restaurant in a Google Sheet. We have yet to have our program actually take restaurants from this sheet.

```
for i in range(10):
    exec("m" + str(i) + " = BasicProposition('k" + str(i) + "')")
    print(eval("m" + str(i)))

print(m1)
```

To start with, we have been experimenting with the code above in order to get a better sense of how this could be implemented and arbitrarily scaled as necessary. The above code will successfully create 10 BasicPropositions (labelled k0-k9), that currently exist beyond the scope of the loop in which they were created, and which will require more experimentation to ascertain stability.

We also created our own displaySolution function so that we could more clearly see which propositions are true. This function returns a list of True propositions for easier readability.

```
def displaySolution():
    if not T.satisfiable():
        return
    result = []
    lis = T.solve()
    for key in lis.keys():
        if lis.get(key):
            result.append(key.__repr__())
    result.sort()
    return result
```

Because we chose a project that doesn't come with a strictly-defined ruleset, we spent a significant amount of time discussing what we wanted to be able to accomplish, and so when we get those things fully implemented, we will be able to explore the model more deeply at that time.

First-Order Extension

We plan on using this section to show the correctness of our model, and to look at some more complex concepts that it might not be possible to show that our model will accomplish.
For example, it might be interesting to show that if there are only three places that meets everyone's needs, and we have specific times set aside where we're all available to go, there might be only one permutation that actually works, because of the hours of operation of those three establishments. Obviously, this is something our model should be able to handle, and so it should be something we could prove in Jape.
Another example might be that we shouldn't need to worry if the meat is kosher at a vegan restaurant. So, the model should be able to get by without needing information about the types of meat served at that restaurant, even though that is something the model would otherwise be expected to check, if we specified

that we needed the food to be kosher.

Request Feedback

1. In our project proposal, we were told that our project idea lacked complexity. We have since added more propositions and constraints. Is our project complex enough?
2. How should we go about implementing our Jape proofs?
3. What features would you want from a model similar to ours? Are there specific accommodations that you would want to see?