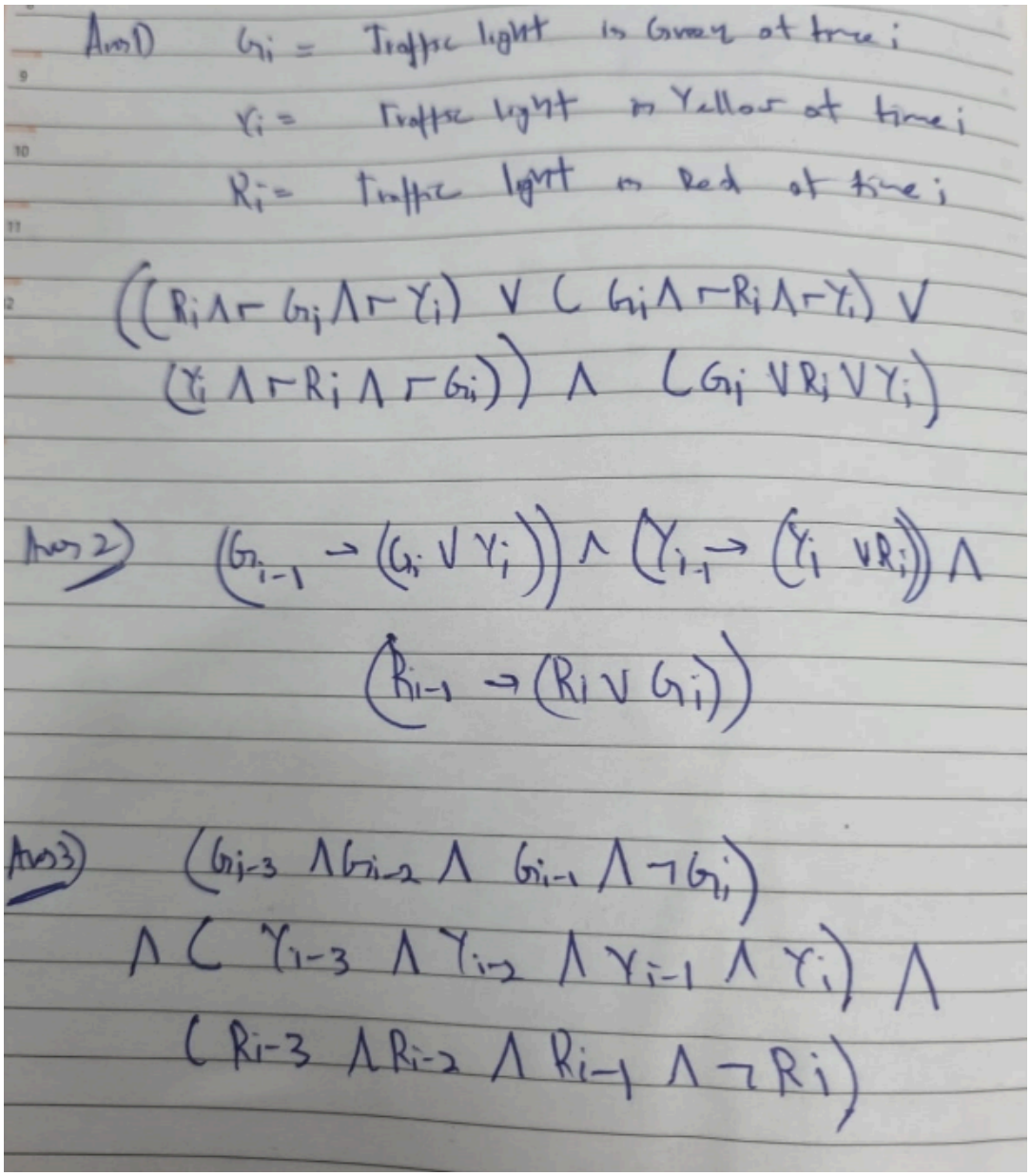
AI ASSIGNMENT

Knowledge Representation, Reasoning and Planning

Keshav Chhabra (2022247)

THEORY

Part 1



Ans1) G_i = Traffic light is Green at time i

Y_i = Traffic light is Yellow at time i

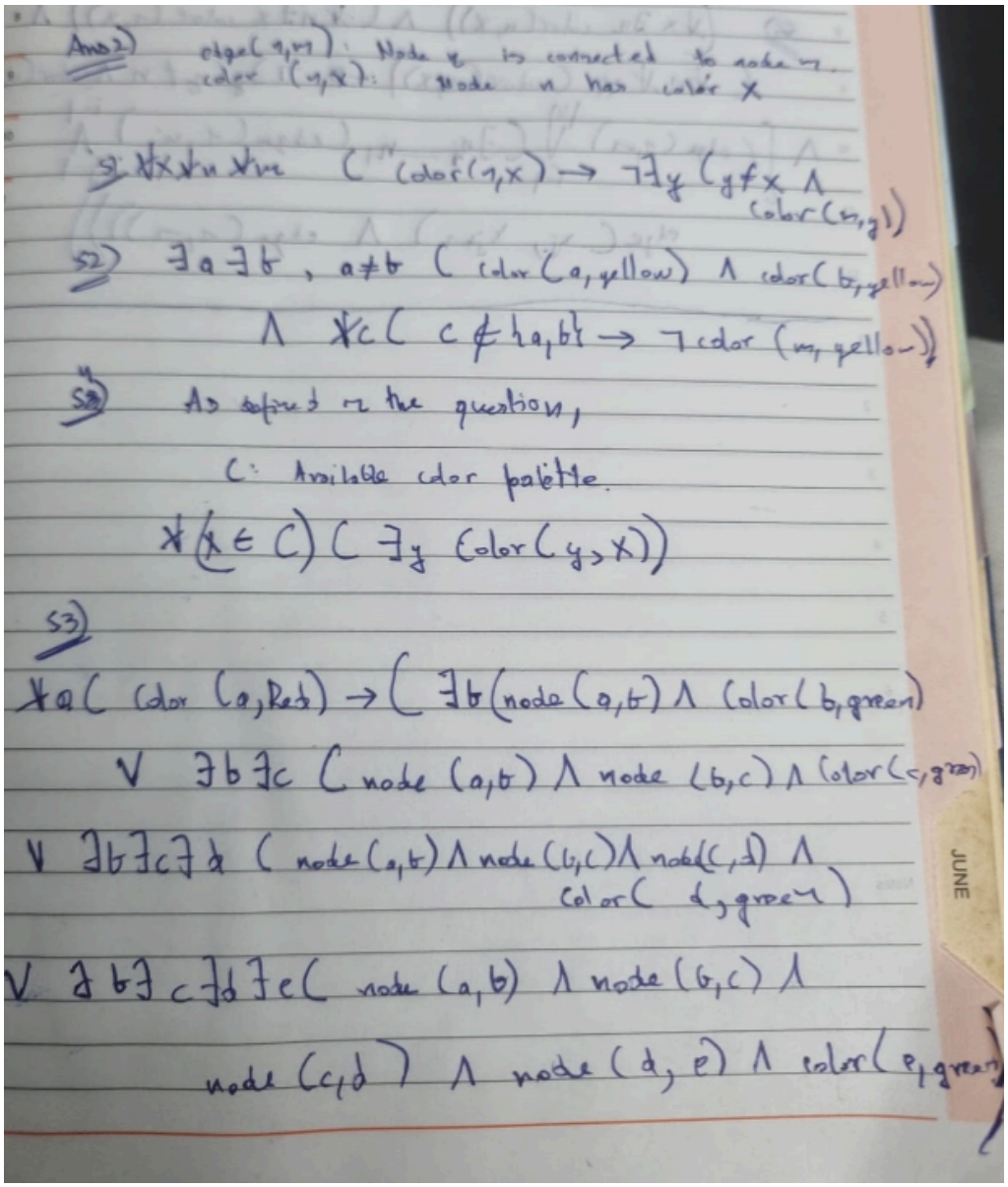
R_i = Traffic light is Red at time i

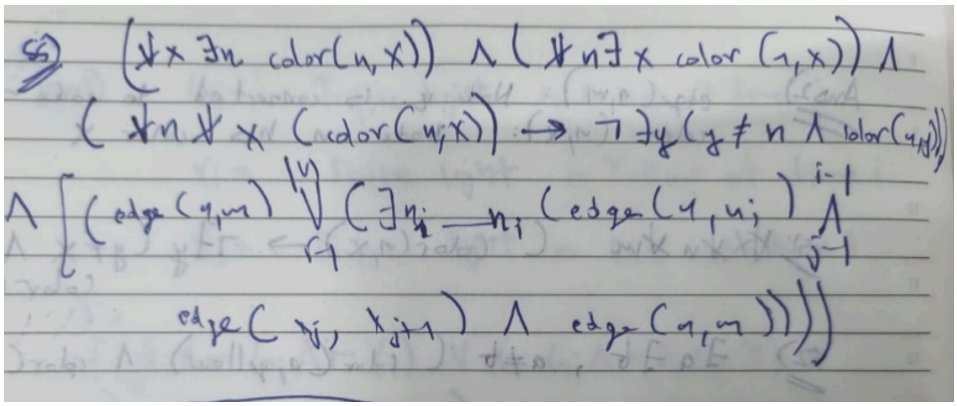
$$((R_i \wedge \neg G_i \wedge \neg Y_i) \vee (G_i \wedge \neg R_i \wedge \neg Y_i) \vee (Y_i \wedge \neg R_i \wedge \neg G_i)) \wedge (G_i \vee R_i \vee Y_i)$$

Ans 2) $$(G_{i-1} \rightarrow (G_i \vee Y_i)) \wedge (Y_{i-1} \rightarrow (Y_i \vee R_i)) \wedge (R_{i-1} \rightarrow (R_i \vee G_i))$$

Ans3) $$(G_{i-3} \wedge G_{i-2} \wedge G_{i-1} \wedge \neg G_i) \wedge (Y_{i-3} \wedge Y_{i-2} \wedge Y_{i-1} \wedge Y_i) \wedge (R_{i-3} \wedge R_{i-2} \wedge R_{i-1} \wedge \neg R_i)$$

Part 2


Ans 2)
edge(q,r): Node q is connected to node r.
color(r,x): Node r has color x
S1) ∀x∀r∀c (color(r,x) → ¬∃y (y≠x ∧ color(r,y))
S2) ∃a∃b, a≠b (color(a,yellow) ∧ color(b,yellow)
∧ ∀c(c ∉ {a,b} → ¬color(r,yellow))
S3) As defined in the question,
C: Available color palette.
∀(x ∈ C)(∃y color(y,x))
S3)
∀a(color(a,Red) → (∃b(node(a,b) ∧ color(b,green)
∨ ∃b∃c (node(a,b) ∧ node(b,c) ∧ color(c,green)
∨ ∃b∃c∃d (node(a,b) ∧ node(b,c) ∧ node(c,d) ∧
color(d,green)
∨ ∃b∃c∃d∃e(node(a,b) ∧ node(b,c) ∧
node(c,d) ∧ node(d,e) ∧ color(e,green)
JUNE



Part 3

Propositional Logic (PL) Representation:

- R : Represents "can read."
- L : Represents "is literate."
- D : Represents "is a dolphin."
- I : Represents "is intelligent."

First-Order Logic (FOL) Representation:

• R(x): "x can read."
• L(x): "x is literate."
• D(x): "x is a dolphin."
• I(x): "x is intelligent."
a) PL: $R \rightarrow L$
FOL: $\forall x\,(R(x) \rightarrow L(x))$
b) PL:$D \rightarrow \neg L$
FOL: $\forall x\,(D(x) \rightarrow \neg L(x))$
c) PL: This statement cannot be fully captured in propositional logic, as it requires quantifiers and variables.
FOL: $\exists x\,(D(x) \wedge I(x))$
d) PL: This statement cannot be fully captured in propositional logic, as it requires quantifiers and variables.
FOL: $\exists x\,(I(x) \wedge \neg R(x))$
e) PL: This statement cannot be fully captured in propositional logic, as it requires quantifiers and variables.
FOL: $\exists x(D(x) \wedge I(x) \wedge R(x)) \wedge \forall y((D(y) \wedge I(y) \wedge R(y)) \rightarrow \neg L(y))$
Proof using Resolution
For the first two statements eliminate the implications and for the third statements break into two statements instead
of x take a skolem constant K:
• S1: $\neg R(x) \vee L(x)$
• S2: $\neg D(x) \vee \neg L(x)$
• S3: D(K)
• S4: I(K)
To prove the 4th statement we need to add its negation in the KB and arrive at a contradiction:
• Add negated goal S5: $\neg I(x) \vee R(x)$
• Resolve S4 with S5 and take x = K, giving S6: R(K)
• Resolve S6 with S1 and take x = K, giving S7: L(K)
2
• Resolve S7 with S2 and take x = K, giving S8: ¬D(K)
• Now resolving S8 with S3 gives us the empty clause.
Arriving at empty clause signifies a contradiction (i.e. unsatisfiability) and hence the KB entails the query.
Now we add statements S5: I(L) and S6: ¬R(L) for skolem constant L.

To prove the 5th statement we will prove the conjunction of 2 statements separately:

(i) $\forall y((D(y) \land I(y) \land R(y)) \rightarrow \neg L(y))$

To prove this we will add its negation $\exists x\,(D(x) \land I(x) \land R(x) \land L(x))$ and arrive at a contradiction:

- For skolem constant Z add S7: D(Z), S8: I(Z), S9: R(Z) and S10: L(Z).
- Resolve S9 with S1 and take x = K, giving S10: L(K).
- Resolve S7 with S2 and take x = K, giving S11: ¬L(K).
- Now resolving S10 with S11 gives us the empty clause.

(ii) $\exists x(D(x) \land I(x) \land R(x))$

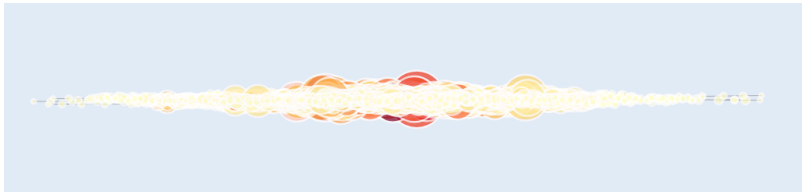
To prove this we will add its negation and arrive at a contradiction:

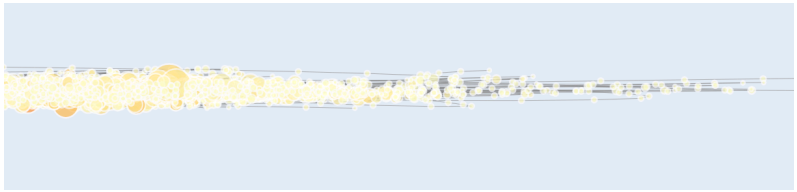
- Add negated goal S7: $\neg D(x) \lor \neg I(x) \lor \neg R(x)$
- Resolve S3 with S2 and S7 and take x = K, giving S8: ¬L(K) and S9: $\neg I(K) \lor \neg R(K)$.
- Resolve S4 with S9 and take x = K, giving S10: ¬R(K).
- But here we cannot arrive at an empty clause and hence this statement cannot be proved using resolution.

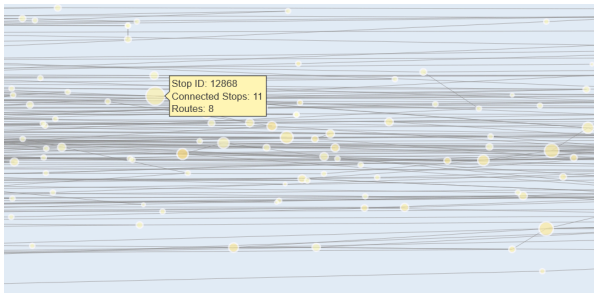
Thus the fifth statement cannot be proved using resolution refutation.

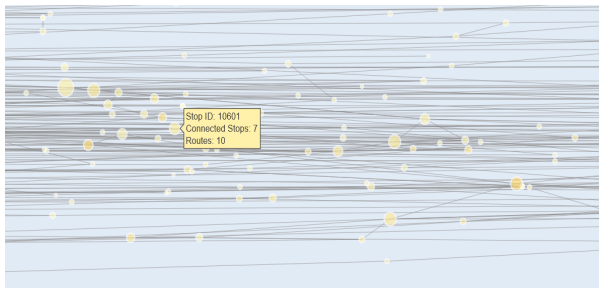
COMPUTATIONAL

REPRESENTATION OF THE KNOWLEDGE BASE USING GRAPH









Using the query as (2573, 1177)

- **Brute Force Approach**

Execution Time: 0.001017 seconds

Memory Usage: 0.000148 MB

- **FOL Library-Based Reasoning**

Execution Time: 0.001405 seconds

Memory Usage: 0.036464 MB

Comparison of steps

• **Brute Force Approach**

The brute-force approach is straightforward, with only basic filtering logic.

Total steps: O(n) in terms of route iterations, where n is the number of routes.

Minimal intermediate steps beyond route checking.

• **FOL Library-Based Reasoning**

The FOL approach introduces intermediate steps like fact insertion and predicate definition.

Total steps: Significantly more than brute force due to setting up the knowledge base and defining rules,

though it enables complex reasoning.

Intermediate steps include fact insertion, predicate evaluation, and handling Datalog's rule-based filtering.

```
Forward Chaining Execution Time: 0.007931 seconds
Forward Chaining Memory Usage: 0.039167 MB
Test forward_chaining (22540, 2573, 4686, 1):  Pass
Forward Chaining Execution Time: 0.017357 seconds
Forward Chaining Memory Usage: 0.168207 MB
Test forward_chaining (951, 340, 300, 1):  Pass
```

```
Backward Chaining Execution Time: 0.013264 seconds
Backward Chaining Memory Usage: 0.168207 MB
Test backward_chaining (22540, 2573, 4686, 1):  Pass
Backward Chaining Execution Time: 0.022931 seconds
Backward Chaining Memory Usage: 0.168207 MB
Test backward chaining (951, 340, 300, 1):  Pass
```

```
PDDL Planning Execution Time: 81.295234 seconds
PDDL Planning Memory Usage: 80.242249 MB
Test pddl_planning (22540, 2573, 4686, 1):  Pass
Terms initialized: DirectRoute, RouteHasStop, OptimalRoute
Executing PDDL Planning...
PDDL Planning Execution Time: 46.446092 seconds
PDDL Planning Memory Usage: 161.840588 MB
```

Forward Chaining:

1. **Start from the beginning:** Begin your search at the start_stop_id.
2. **Identify relevant routes:** Look for routes that include both the start_stop_id and the stop_id_to_include .
3. **Find connecting routes:** Determine which of these routes also include the end_stop_id .
4. **Add to the solution:** Add these connecting routes to your final solution.

Backward Chaining:

1. **Start from the end:** Begin your search at the end_stop_id .
2. **Identify relevant routes:** Look for routes that include both the end_stop_id . and the stop_id_to_include
3. **Find connecting routes:** Determine which of these routes also include the start_stop_id.
4. **Add to the solution:** Add these connecting routes to your final solution.

Comparison:

Both forward and backward chaining involve similar steps, but they differ in their approach:

- **Forward chaining:** Data-driven, explores all paths from the start.
- **Backward chaining:** Goal-driven, works backward from the end goal.

While both methods can be effective, backward chaining can be more efficient in certain scenarios. When constraints are applied, backward chaining can stop searching as soon as a valid path is found, potentially saving processing time. Forward chaining, on the other hand, may continue to explore additional routes that ultimately don't meet the constraints.

PDDL Planning: it is an advanced technique to build the route taking into account possible combinations of the routes and transfers

1. **Declarative Approach**

- Separates domain knowledge (actions, predicates) from problem instances
- Actions: board_route and transfer_route
- Predicates DirectRoute , ValidPath
- Leverages constraint satisfaction
- Prunes invalid paths based on predicates

- Space complexity: O(T * R)
- where: T = max_transfers + 1 R = number of routes