GRP 27: Curriculum Satisfiability

Luke Gannon (22LFG1)

Kareem Yakubu (21KOY)

Will Wang (21WZW1)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

December 8, 2023

Contents

Abstract

This project addresses the intricate task of academic scheduling, incorporating elements such as professors, courses, classrooms, and academic programs. Our objective is to create a schedule adhering to a range of constraints, including professor qualifications, course prerequisites, and program requirements.

We utilize propositional logic to model the scheduling process, where each component is represented as a proposition. These propositions are amalgamated into a logical theory, which is then solved using a SAT solver to produce a feasible schedule.

The project explores the effects of various constraints on the scheduling process. It investigates how professor qualifications influence their course allocations and the impact of course prerequisites on the scheduling timeline. Furthermore, it ensures that academic program requirements are met by appropriately scheduling all necessary courses.

A critical aspect of this exploration is recognizing the computational limits posed by SAT solvers, especially when dealing with models that have a high number of variables or complex constraint permutations. This acknowledgment is vital, as it underscores the challenges and practical considerations in applying such models to real-world scheduling problems.

Through this project, we aim to provide insights into the challenges and complexities of academic scheduling and demonstrate the utility of propositional logic and SAT solvers in solving complex scheduling issues, while also considering the computational constraints inherent in such approaches.

Propositions

In this section, we present a comprehensive list of propositions devised for the academic scheduling model. These propositions encapsulate various elements of the scheduling process, including course assignments, professor qualifications, and program requirements. While all these propositions contribute to the theoretical framework of our model, it's important to note that not all were employed in the final implementation, as detailed later in the Model Exploration section.

1. **CourseAssigned**$_{crsdt}(Course, Room, Term, Day, Time)$:

   This proposition represents whether a course c is assigned to a room r at a specific term s, day d, and time t.

   - Example: "Course: 'CS101' is assigned to Room: '101' at Time: '10:00' on Day: 'Monday' in Term: 'Spring 2022'."

2. **CoursePrerequisite**$_{c1c2}$:

   This proposition represents whether a course c1 has a prerequisite course c2.

   - Example: "Course: 'CS102' has Prerequisite: 'CS101'."

3. **ProfessorAssigned**$_{pcsdt}$:

   This proposition represents whether a professor p is assigned to teach a course c during a term s, day d, and time t.

   - Example: "Professor: 'Dr. Jane Smith' is assigned to Course: 'CS101' at Time: '10:00' on Day: 'Monday' in Term: 'Spring 2022'."

4. **ProfessorQualified**$_{pc}$:

   This proposition represents whether a professor p is qualified to teach a course c.

   - Example: "Professor: 'Dr. Jane Smith' is qualified to teach Course: 'CS101'."

5. **ProgramReqCourse**$_{pc}$:

   This proposition represents whether a course c is a required course for a program p.

   - Example: "Course: 'CS101' is required for Program: 'Computer Science' in Year: '1'."

6. **ProgramCanComplete**$_{pt}$:

   This proposition represents whether a program p can be completed in a specific term t.

   - Example: "Program: 'Computer Science' can be completed in Term: 'Spring 2022'."

7. **CourseFullYear**$_{c}$:

   This proposition represents whether a course c is a course that is scheduled throughout the entire year.

- Example: "Program: 'Math 111' is schedule for 6 credits over 2 terms"

Constraints

List of constraints that relate to the model and their (English) interpretation.

1. **A professor cannot be assigned to teach two courses at the same time.**
   $\neg(ProfessorAssigned_{pc1sdt} \wedge ProfessorAssigned_{pc2sdt})$
   For every pair of distinct courses c1 and c2 it is not the case that a professor is assigned to both of them during the same time.

2. **Professors can only teach courses for which they're qualified.**
   $\neg(\neg ProfessorQualified_{pc} \wedge ProfessorAssigned_{pcsdt})$
   It is not the case where a professor p is not qualified to teach a course c, and the professor is assigned to teach the course c.

3. **2 Professors cannot be assigned to the same course during the same time.**
   $\neg(ProfessorAssigned_{p1csdt} \wedge ProfessorAssigned_{p2csdt})$
   For every pair of distinct professors p1, and p2. It is not the case that p1 and p2 both teach a course c during term s, day d, and time t.

4. **Prerequisites are scheduled in a term before the course that requires them.**
   $CoursePrerequisite_{c1c2} \implies (CourseAssigned_{c1r(2)dt} \wedge CourseAssigned_{c2r(1)dt})$
   For every pair of distinct courses c1 and c2. If c2 is a prerequisite of c1, then c1 is assigned during term 2, and c2 is assigned during term 1.

5. **There are at most 2 lectures per course.**
   $CourseAssigned_{crsd1t1} \wedge CourseAssigned_{crsd2t2}$
   For every pair of distinct time periods (d1, t1) and (d2, t2), a course c is assigned during d1, t1 and d2, t2.

6. **A Program can only be "completed" during a Term if all the required Courses can be scheduled during that term.**
   $(ProgramReqCourse_{pc} \wedge CourseAssigned_{crsdt}) \implies ProgramCanComplete_{ps}$
   For each course c and each term s, if c is a required course by a program p, and c is assigned during s, then the p can be completed during s.

7. **The proposition ProgramCanComplete must be true.**
   $ProgramCanComplete_{pt}$
   For each term t.

8. **Two courses cannot be assigned to the same room during the same term, day, and time.**
   $\neg(CourseAssigned_{c1rsdt} \wedge CourseAssigned_{c2rsdt})$
   For every pair of distinct courses c1 and c2, it is not the case where c1 is assigned to a room r during term s, day d, and time t, and c2 is also assigned to r during term s, day d, and time t.

9. **A full-year course must be scheduled for both terms**
   $CourseFullYear_c \implies (CourseAssigned_{cr(1)dt} \land CourseAssigned_{cr(2)dt})$

   If a course c is a full-year course, then c must be assigned during terms 1 and 2.

10. **If two programs share a required course, that course must not conflict with other required courses in either program.**
    $(ProgramReqCourse_{p1c1} \land ProgramReqCourse_{p2c1} \land ProgramReqCourse_{p2c2}) \implies \neg(CourseAssigned_{c1rsdt} \land CourseAssigned_{c2rsdt})$

    For each pair of distinct programs p1, and p2, and for each course c1, and c2, if a course c1 is required by both p1 and p2, and a course c2 is required by p2, then it is not the case where c1 is assigned to a room r during term s, day, d and time t and, c2 is assigned to a room r, during term s, day d, and time t.

11. **A course cannot have multiple lectures in a day**
    $CourseAssigned_{crsdt1} \implies \neg CourseAssigned_{crsdt2}$

    For each pair of distinct times t1, and t2, it is not the case where a course c is assigned to a room r during a term s, day d, and time t1, and also a time t2.

Model Exploration

Initial Focus on Individual Student Enrollment

Our project initially concentrated on modeling the academic scheduling process from an individual student's perspective. The primary goal was to ascertain whether a student could feasibly schedule all their required courses within a year. This approach allowed us to focus on the core elements of course availability and student course load, laying the foundation for our scheduling model. It highlighted the challenges students face in aligning course offerings with their academic plans, providing us with valuable insights into the practical aspects of academic scheduling.

Shift to University Perspective

Following valuable feedback, we pivoted our model to reflect the university's viewpoint. This shift marked a significant expansion of our project's scope, moving from individual student scheduling to managing the scheduling needs of multiple academic programs over a typical four-year degree span. This broader perspective required us to consider a more complex array of variables, such as program-specific course requirements and the scheduling of courses across different academic years. The transition to this university-centric approach allowed us to address the larger-scale challenges of academic scheduling, such as resource allocation and program coherence.

Incorporating Time Slots, Professors, and Classrooms

As our model evolved, we began to integrate additional critical factors into our scheduling algorithm, namely time slots, professors, and classrooms. This

development was crucial in transforming our model into a more comprehensive representation of the academic scheduling process. Including time slots in the model brought in the dimension of temporal scheduling, necessitating the alignment of courses with available time periods. The inclusion of professors added another layer, requiring us to consider their qualifications and availability. Finally, incorporating classroom availability presented logistical challenges, ensuring that courses were assigned to appropriate venues. Each addition significantly increased the complexity of our model but was essential for creating a realistic and functional academic schedule.

Redefining Course Selection: Required vs. Electives

In the early stages of our model, electives were included alongside required courses. However, we soon realized the practicality of focusing solely on required courses. Electives, by their nature, offer flexibility and choice to students, making their scheduling less predictable and more individualized. By concentrating on required courses, which have more structured scheduling and are mandatory for program completion, we were able to streamline our model. This approach mirrored the priority of academic institutions to ensure that essential courses for each program are available and scheduled in a manner that enables all students to complete their programs on time.

Prerequisite Scheduling Constraints

One of the critical aspects we introduced to our model was the scheduling of course prerequisites. We recognized the importance of ensuring that prerequisite courses are scheduled before the courses that require them. This logical sequence is fundamental to academic progression and was a crucial factor in our model. Implementing this constraint required careful consideration of course dependencies and their impact on the overall schedule. It was a challenging yet essential addition that greatly enhanced the realism and functionality of our model.
Our prerequisite constraints were too restrictive as they worked with scheduling prerequisites and their corresponding courses in all possible time slots across two terms. We modified the constraints so that a prerequisite is scheduled in any time slot in the term before its corresponding course.

```
# Ensure that prerequisites are scheduled in a term before the course that requires them (IFF the course is the same level as the course that requires it)
for prop in tqdm(course_prerequisite_props, desc="Adding course constraints (1/4)"):
        course = prop.course
        prerequisite = prop.prerequisite
        course_level = int(course.split()[1][0])  # Extract the course level from the course code
        prerequisite_level = int(prerequisite.split()[1][0])  # Extract the course level from the prerequisite code

        if course_level == prerequisite_level:  # Check if the course level matches the prerequisite level
            prereq_in_prev_term = [CourseAssigned(prerequisite, room, "T-1", day, time) for room in CLASSROOMS for day in DAYS for time in TIMESLOTS]
            course_in_next_term = [CourseAssigned(course, room, "T-2", day, time) for room in CLASSROOMS for day in DAYS for time in TIMESLOTS]
            # Add constraint for the entire term, not each room and day
            constraint.add_implies_all(E, course_in_next_term, prereq_in_prev_term)
```

Independent Year Level Scheduling

Our model initially approached scheduling with a cumulative perspective, considering the impact of course offerings across all year levels. However, to better reflect the student experience and simplify our model, we shifted to viewing each year level independently. This change meant that courses for a particular year were scheduled without regard to their implications for other years. This

approach aligns with how students typically plan their schedules, focusing on the immediate next year's requirements. It also simplified our model, making it more manageable and efficient, as it reduced the interdependencies and complexities involved in multi-year scheduling.

Handling Shared Required Courses

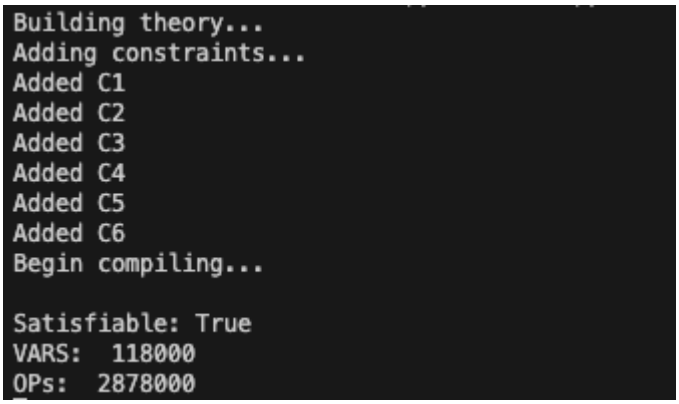
A significant complexity in our model arose from the realization that some programs share required courses. For example, multiple computing programs might require a foundational course like CISC 204. This shared requirement introduced a new dimension to our scheduling model, necessitating careful coordination to avoid conflicts. We implemented a constraint ensuring that if two programs share a required course, it must be scheduled in a way that accommodates both programs. Additionally, we allowed courses exclusive to a single program to conflict with non-required courses from others. This nuanced approach to shared courses was vital in accurately reflecting the complexities of real-world academic scheduling.

Optimizing for Computational Efficiency

As our model grew in sophistication, we faced significant challenges regarding computational efficiency, especially when managing a high number of constraints and propositions. Initially, our model, with fewer constraints, yielded a manageable number of scheduling blocks. However, as we introduced more constraints to more accurately reflect real-world scheduling scenarios, we encountered issues with unwieldy output and excessive computational demands. This situation necessitated a critical phase of optimization, focusing on implementing more effective constraints to reduce computational load without compromising the model's accuracy. Striking a balance between the model's complexity and computational feasibility was essential in developing a practical and reliable scheduling tool.

A particular test utilizing 4 courses, 8 classrooms, 5 days, and 8 time slots over 2 terms highlighted this challenge, returning an overwhelming 2241 scheduling blocks. Recognizing this as impractical, we identified a need to implement additional, more efficient constraints.

Our initial approach to managing classroom assignments, for instance, involved placing constraints on every possible combination of courses and classrooms. This method, while comprehensive, placed considerable strain on the solver, particularly as the number of combinations increased with larger datasets.



To address this, we revised our strategy, focusing on a more targeted approach. We created lists for each course being scheduled in a classroom and applied a constraint to allow at most one assignment from each list. This new implementation was significantly more efficient, reducing the computational load while maintaining the integrity and accuracy of the scheduling model.

```
# Ensure that a course can't be assigned to a classroom that already has a course scheduled at a specific day and time.
    # assignments_by_block = defaultdict(list)
    # for assignment in course_assigned_props:
    #     block = (assignment.room, assignment.term, assignment.day, assignment.time)
    #     assignments_by_block[block].append(assignment)
    # for assignments in tqdm(assignments_by_block.values(), desc="Adding classroom constraints"):
    #     for assignment1, assignment2 in combinations(assignments, 2):
    #         E.add_constraint(~assignment1 | ~assignment2)
```

```
# Classroom Constraints:
    # Ensure no two courses are scheduled in the same classroom at the same timeslot
    for room, day, time, term in product(CLASSROOMS, DAYS, TIMESLOTS, TERMS):
        # Create a list of propositions for each course being scheduled in this classroom at this timeslot
        course_assignments = [CourseAssigned(course, room, term, day, time) for course in COURSES]
        # Add a constraint that at most one of these propositions can be true
        constraint.add_at_most_one(E, course_assignments)
```

Through these optimization efforts, we successfully enhanced our model's performance, making it both more efficient and more reflective of real-world scheduling challenges.

Full-Year Course Constraints

Incorporating full-year courses like MATH 111, which span two terms, presented another layer of complexity. We addressed this by adding a constraint that ensures full-year courses are scheduled in both terms of the academic year. However, we encountered an issue where the model scheduled these courses in every available time slot, leading to unrealistic schedules. To resolve this, we refined our constraints to limit the frequency of course sessions: a course could have at most two lectures per week and no more than one lecture per day. This adjustment helped us model the typical structure of full-year courses more accurately.

Troubleshooting and Refining Constraints

Our journey with the scheduling model was marked by continuous refinement, especially in the realm of constraints. We encountered a significant challenge when our model returned false for a schedule that should have been feasible. The root cause was traced to our constraint handling for classroom assignments. Our initial approach placed a constraint on every possible pair of courses, leading to an overwhelming number of computations and placing immense strain on the solver. We resolved this by strategically limiting constraints to only those courses that shared a time slot and classroom. This fine-tuning not only fixed the error but also optimized the solver's performance, enhancing the overall efficiency of our model.

Evolving Display Functions

As our model evolved, so did our methods for displaying the output. Initially, we relied on basic tables for data presentation, which quickly proved inadequate for handling large volumes of data. We then transitioned to using the pandas library, which allowed us to create more sophisticated pivot tables. These tables were then rendered as HTML tables or exported to Excel files for better visualization and user interaction. We further enhanced the user experience by generating separate tables for different terms, and aligning course blocks with relevant information like time and room. This evolution in our display functions significantly improved the usability and readability of our model's output.

```
Satisfiable: True
VARS:  504
OPs:  2309100

Building Professor Schedule Encoding...
Begin compiling the encoding...

Satisfiable: True
VARS:  8
OPs:  24
  Term        Day        Time Room             Course
0  T-2    Tuesday  11AM-1230PM   A5  CISC 124 (Prof: P1)
1  T-1    Tuesday  11AM-1230PM   A2  CISC 121 (Prof: P1)
2  T-2     Monday    8AM-930AM   A2  MATH 120 (Prof: P2)
3  T-1  Wednesday  11AM-1230PM   A5  MATH 120 (Prof: P2)
4  T-2  Wednesday    8AM-930AM   A4  MATH 120 (Prof: P2)
5  T-1  Wednesday   930AM-11AM   A1  CISC 121 (Prof: P1)
6  T-1     Monday  11AM-1230PM   A6  MATH 120 (Prof: P2)
7  T-2     Monday   930AM-11AM   A1  CISC 124 (Prof: P1)
```

Term 1 Schedule

	Day	Friday	Monday	Thursday	Tuesday	Wednesday
Time	**Room**					
11AM-1230PM	**A1**	CISC 121 (Room: A1)			MATH 120 (Room: A1)	
	A4					MATH 120 (Room: A4)
330PM-5PM	**A3**		MATH 120 (Room: A3)			
	A6	MATH 120 (Room: A6)				
	A7	MATH 121 (Room: A7)				
5PM-630PM	**A2**				MATH 121 (Room: A2)	
630PM-8PM	**A2**		MATH 121 (Room: A2)			
	A3					CISC 121 (Room: A3)
	A5			CISC 121 (Room: A5)	CISC 121 (Room: A5)	MATH 121 (Room: A5)
930AM-11AM	**A1**			MATH 121 (Room: A1)		
	A4		CISC 121 (Room: A4)	MATH 120 (Room: A4)		

Term 2 Schedule

	Day	Friday	Monday	Thursday	Tuesday	Wednesday
Time	**Room**					
1230PM-2PM	**A2**			MATH 121 (Room: A2)		
2PM-330PM	**A2**	MATH 121 (Room: A2)				
	A3					CISC 124 (Room: A3)
	A5				MATH 121 (Room: A5)	
330PM-5PM	**A1**	MATH 120 (Room: A1)				
5PM-630PM	**A2**			MATH 120 (Room: A2)		
	A3			CISC 124 (Room: A3)		
630PM-8PM	**A2**				MATH 120 (Room: A2)	
	A4	CISC 124 (Room: A4)				
	A7		MATH 120 (Room: A7)			
8AM-930AM	**A2**		CISC 124 (Room: A2)			
930AM-11AM	**A3**		MATH 121 (Room: A3)		CISC 124 (Room: A3)	MATH 120 (Room: A3)
	A4					MATH 121 (Room: A4)

Streamlining Propositions and Constraints

In our pursuit of creating an efficient and practical scheduling model, we recognized the importance of streamlining our propositions and constraints. Throughout the development process, we identified several elements that were either redundant or unnecessary, leading to their elimination from the model. This process of streamlining greatly enhanced both the computational efficiency and clarity of our model.

One such example was the ProgramCanComplete proposition. Initially included to ensure that each program could be feasibly completed within the given constraints, we found that this proposition was naturally satisfied by other constraints in the model, specifically the constraint ensuring that there be at least one lecture per course. This realization led us to remove the ProgramCanComplete proposition, as it did not add any additional functional value to the model.

Similarly, the ClassroomAssigned constraint was identified as duplicative. Initially, it was designed to ensure that courses were appropriately assigned to classrooms. However, upon closer examination, we realized that its function was essentially covered by the CourseAssigned proposition. This overlap made the ClassroomAssigned constraint redundant, leading to its removal from the model.

This process of critically evaluating and streamlining our propositions and constraints not only made our model more efficient in terms of computational resources but also simplified its overall structure. By removing unnecessary elements, we made the model more transparent and easier to work with. This approach was crucial in ensuring that our model remained sharply focused and effective in addressing the core challenges of academic scheduling, without being bogged down by superfluous or redundant components.

Further Exploration

We noticed that sometimes a group of courses is required, and a student must pick a certain amount of credits out of the group. Constraints involving group courses would result in scheduling courses so that all courses in the group do not conflict with other required courses, or at least enough courses for the required amount of credits.

Another characteristic of the courses we thought of was courses with different sections. Courses with 2 sections would have to be scheduled so that at least one section can taken by a student. Different sections would also lift some restrictions on constraints such as the constraint that if two programs share a required course, that course must not conflict with other required courses in either program. Having 2 sections would allow one section to conflict with some of its required programs as the other section can be scheduled to fit them.

We also explored more professor-related aspects such as having multiple professors teach a course with multiple sections or having professors teach multiple courses in a term. These constraints would need courses to also be scheduled in a way to accommodate a professor's schedule .

First-Order Extension

Introduction to First-Order Logic

Extending our model to a predicate logic setting, we would include propositions to specify the types of our objects:

1. P(x): x is a professor.
2. C(x): x is a course.
3. R(x): x is a room.
4. T(x): x is a term
5. Pg(x): x is a program.

Domains of Discourse:

1. Professors: A list of professor names.
2. Courses: A list of courses.
3. Rooms: A list of rooms.
4. Terms:"T-1" or "T-2" to represent the first or second term.
5. Programs: List of programs.
6. Days: List of days Monday - Sunday.
7. Time-slots: List of 90-minute time slots.

Predicates:

1. ProfessorAssigned(p,c,s,d,t): True if a professor p is assigned to teach a course c during a term s, day d, and time t.
2. ProfessorQualified(p,c): True if a professor p is qualified to teach a course c.
3. CourseAssigned(c,r,s,d,t): True if a course c is assigned to a room r at a specific term s, day d, and time t.
4. CoursePrerequisite(c1,c2): True if a course c1 has a prerequisite course c2.
5. ProgramReqCourse(p,c): True if a course c is a required course for a program p.
6. ProgramCanComplete(p,t): True if a program p can be completed in a term t.
7. CourseFullYear(c): True if a course c is scheduled throughout the entire year.

Constraints:

Some constraints would be easier to represent in a predicate setting. To avoid receptiveness we will assume that the objects are always the right type. So there is no need to state P(x), C(y)...

1. **A professor cannot be assigned to teach two courses at the same time.**

   $\forall p \forall c1 \forall c2 \forall s \forall d \forall t((ProfessorAssigned(p, c1, s, d, t) \wedge ProfessorAssigned(p, c2, s, d, t)) \implies (c1 = c2))$

   If a professor is assigned to a course c1, and a course c2 during the same time, then c1 is the same course as c2.

2. **Professors can only teach courses for which they're qualified.**

   $\forall p \forall c \neg(\neg ProfessorQualified(p, c) \wedge \exists s \exists d \exists t ProfessorAssigned(p, c, s, d, t))$

   For every professor p and course c, it is not the case where p is not qualified to teach c, and there exists a term s, day d, and time t where p is assigned to teach c.

3. **Prerequisites are scheduled in a term before the course that requires them.**

   $\forall c1 \forall c2(CoursePrerequisite(c1, c2) \implies (\exists r \exists d \exists t CourseAssigned(c1, r, 2, d, t) \wedge \exists r \exists d \exists t CourseAssigned(c2, r, 1, d, t))$

   For all courses c1 and c2 such that c2 is a prerequisite to c1, c1 is assigned during term 2 and c2 is assigned during term 1.

4. **A classroom cannot be assigned to two courses at the same time.**

   $\forall r \forall c1 \forall c2 \forall s \forall d \forall t((CourseAssigned(c1, r, s, d, t) \wedge CourseAssigned(c2, r, s, d, t)) \implies (c1 = c2))$

   For all classrooms r, terms s, days d, and time t, if a course c1 and a course c2 are both assigned to r during s,d, and t, then c1 and c1 are the same course.

5. **A course cannot have multiple lectures per day.**

   $\forall c \forall r \forall s \forall d(\exists t1(CourseAssigned(c, r, s, d, t1) \implies \neg \exists t2 CourseAssigned(c, r, s, d, t2))$

   For all courses c rooms r, terms s, and days d, if there exists a time t1 such that c is assigned to r during s d and t1, then it is .

6. **The proposition ProgramCanComplete must be true.**

   $\forall p \forall t ProgramCanComplete(p, t)$

   For all programs p and for all terms t, p must be completable.

7. **If two programs share a required course, that course must not conflict with other required courses in either program.**

   $\forall p1 \forall p2 \forall c1((ProgramReqCourse(p1, c1) \wedge ProgramReqCourse(p2, c1)) \implies \forall c2 \forall r \forall s \forall d \forall t \neg(CourseAssigned(c1, r, s, d, t) \wedge CourseAssigned(c2, r, s, d, t) \wedge ProgramReqCourse(p2, c2)))$

   For every course c1 if it is required by programs p1 and p2, then it is not the case where c1 is assigned at the same time as a course c2 that is required by the program p2.

8. **A Program can only be "completed" during a Term if all the required Courses can be scheduled during that term.**

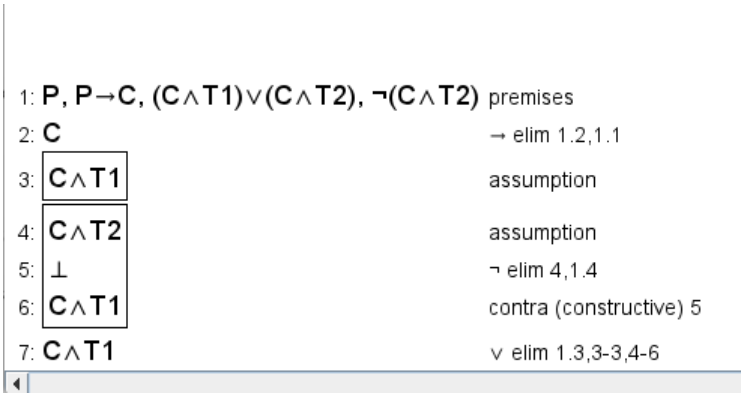
   $\forall p \forall s(\forall c \exists r \exists d \exists t(ProgramReqCourse(p,c) \wedge CourseAssigned(c,r,s,d,t)) \implies ProgramCanComplete(p,s))$

   For all programs p and terms s, for all courses c such that c is required by p, and there exists a room r, day d, and time t that c is assigned, then the program can be completed.

Jape Proofs

Proof 1:

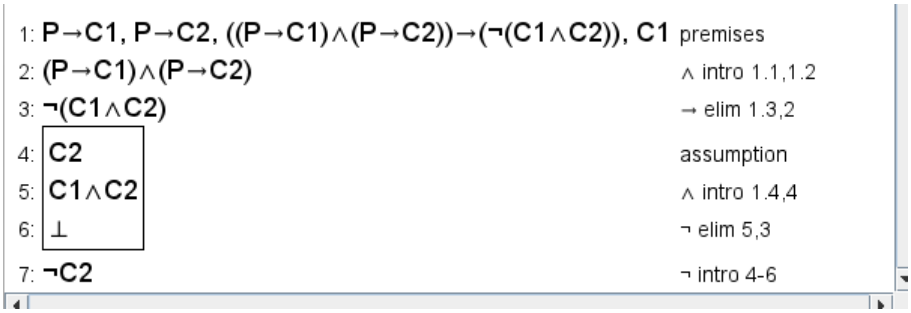
If a program is required by a course and the course is not scheduled in term 2, then it must be scheduled in term 1



We can interpret that a program P exists, and P requires a course C. The course C can be scheduled during term 1 T1, or term 2 T2. It C is not scheduled during T2 then it must be scheduled during T1

Proof 2:

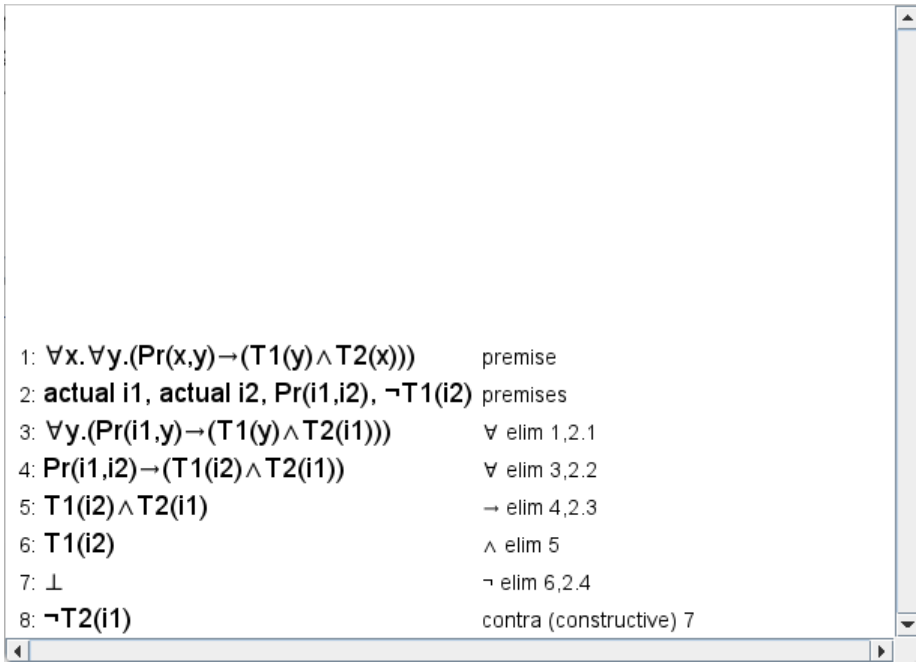
If 2 courses are required for a program and 1 course is scheduled for a certain time, the other course cannot be scheduled for that time



We interpret C1 as course 1 is scheduled for a certain time, and C2 as course 2 is scheduled for the same time. If C1 is a required program of P and C2 is also a required program of P, then it is not that C1 and C2 are both true. If we have C1, course 1 is scheduled for that time, then that means C2 is not true

Proof 3:

A course cannot be scheduled unless its prerequisites are scheduled before it.



We interpret it as, if a course y is a prerequisite of a course x, then y is scheduled in term 1, and x is scheduled in term 2. So if a course i2 is a prerequisite of i1, and it is not scheduled in term 1, then i1 cannot be scheduled in term 2.

Conclusion

We deeply explored a university course scheduling logical model to try and illustrate how propositional and predicate logic can be applied to solve real-world problems. Taking into account many of the nuanced attributes of courses, we translated them into propositions and constraints. We encountered difficulties whilst trying to scale our model, stemming from the sheer quantity of propositions combined with our limited computing power. The complexity of the model quickly surpassed our expectations, exceeding millions of operations at some points. Nevertheless, we successfully modelled three full years' worth of university courses, keeping the schedule free of classroom and time conflicts and accounting for the realistic limitations of professor availability. Had we been able to further develop this project, we would have liked to explore constraints

that would have made the scheduling even more complex - such as student sections, classroom capacity and more; we were unfortunately unable to do this due to the lack of processing power at our disposal. With our chosen propositions and constrains, we built a elegant and sophisticated logical model to solve our course planning senerio.

Term 1 Schedule

	Day	Monday	Tuesday	Wednesday
Time	Room			
11am-12:30pm	A1		CISC 335 (Prof: P1)	
	A2	CISC 204 (Prof: P3)	CISC 327 (Prof: P8)	
	A3	CISC 326 (Prof: P9)	MATH 121 (Prof: P11)	CISC 235 (Prof: P7)
	A4	PHIL 259 (Prof: P4)	CISC 324 (Prof: P9)	CISC 221 (Prof: P2)
	A5	MATH 121 (Prof: P11)	MATH 110 (Prof: P10)	
12:30-2pm	A2	CISC 324 (Prof: P9)	CISC 365 (Prof: P8)	CISC 365 (Prof: P8)
	A3		CISC 102 (Prof: P6)	MATH 111 (Prof: P2)
	A4	CISC 223 (Prof: P3)	CISC 326 (Prof: P9)	MATH 110 (Prof: P10)
	A5	CISC 235 (Prof: P7)	COMM 200 (Prof: P5)	CISC 360 (Prof: P6)
8-9:30am	A1		CISC 220 (Prof: P7)	
	A2	CISC 121 (Prof: P1)	CISC 203 (Prof: P10)	CISC 102 (Prof: P6)
	A3	CISC 124 (Prof: P4)	PHIL 259 (Prof: P4)	CISC 325 (Prof: P11)
	A4	CISC 325 (Prof: P11)	MATH 112 (Prof: P12)	CISC 335 (Prof: P1)
	A5	COMM 200 (Prof: P5)	CISC 360 (Prof: P6)	
9:30-11am	A1	CISC 221 (Prof: P2)		
	A2		CISC 124 (Prof: P4)	CISC 322 (Prof: P5)
	A3	MATH 120 (Prof: P12)	MATH 120 (Prof: P12)	CISC 121 (Prof: P1)
	A4	CISC 327 (Prof: P8)	CISC 223 (Prof: P3)	
	A5		MATH 111 (Prof: P2)	

Term 2 Schedule

	Day	Monday	Tuesday	Wednesday
Time	Room			
11am-12:30pm	A1	MATH 121 (Prof: P11)		CISC 220 (Prof: P7)
	A2		CISC 221 (Prof: P2)	MATH 120 (Prof: P12)
	A3			CISC 327 (Prof: P8)
	A4	MATH 111 (Prof: P2)	PHIL 259 (Prof: P4)	
	A5	CISC 327 (Prof: P8)	CISC 324 (Prof: P9)	CISC 121 (Prof: P1)
12:30-2pm	A1	PHIL 259 (Prof: P4)	CISC 322 (Prof: P5)	MATH 112 (Prof: P12)
	A2	COMM 200 (Prof: P5)		
	A3			MATH 111 (Prof: P2)
	A4	MATH 110 (Prof: P10)	CISC 220 (Prof: P7)	
	A5	CISC 121 (Prof: P1)	MATH 112 (Prof: P12)	COMM 200 (Prof: P5)
8-9:30am	A1	CISC 102 (Prof: P6)	CISC 365 (Prof: P8)	CISC 322 (Prof: P5)
	A2		MATH 121 (Prof: P11)	
	A3	CISC 325 (Prof: P11)	CISC 203 (Prof: P10)	CISC 326 (Prof: P9)
	A4		CISC 204 (Prof: P3)	CISC 360 (Prof: P6)
	A5	CISC 365 (Prof: P8)	MATH 120 (Prof: P12)	
9:30-11am	A1	CISC 221 (Prof: P2)	MATH 110 (Prof: P10)	
	A3		CISC 360 (Prof: P6)	CISC 325 (Prof: P11)
	A4	CISC 203 (Prof: P10)		CISC 204 (Prof: P3)
	A5	CISC 326 (Prof: P9)		CISC 102 (Prof: P6)

3 Years of Courses for a Program Scheduled