STUDENT NUMBER: 101063025
MSc ARTIFICIAL INTELLIGENCE

SATISFIABILITY MODULO THEORIES AND BOUNDED MODEL CHECKING for AI PLANNING

ABSTRACT

This project explores the application of Satisfiability Modulo Theories (SMT) and Bounded Model Checking (BMC) in the context of AI planning. It particularly focuses on an AI-driven optimising taxi system within a simulated urban environment with intelligent agents (taxis). The intelligent agents (taxis) operate under various constraints such as dynamic obstacle management and traffic rules. The primary objective of the project is to ensure effective route planning, collision avoidance, optimisation of taxi operators, and traffic regulations compliance.

The system implemented using the Python Programming language, integrates the Z3 SMT solver to address complex logical constraints and implement BMC for formal verification of safety properties. Through a series of carefully designed experiments, the simulation evaluates the effectiveness, and efficiency of various route optimization algorithms such as A*, Depth-First Search, and Breadth-First Search. In addition, the experiments focus on collision avoidance, system scalability, traffic compliance, dynamic roadblocks and the performance of route optimization algorithms.

The results reflect the effectiveness of integrating SMT and BMC in enhancing AI planning systems. Particularly for such a system, SMT and BMC are implemented to ensure the systems' safety and reliability, in addition to providing insights into the efficiency, scalability, and robustness of urban transportation models. The findings highlight the importance of selecting appropriate planning algorithms based on environmental complexity and fleet size. This project contributes to the development of advanced and reliable automated planning/urban transportation systems by providing insights into the practical application of formal verification techniques in real-time, dynamic environments.

TABLE OF CONTENTS

Introduction

Artificial Intelligence (AI) has significantly advanced in recent years, leading to substantial improvements in automated systems and intelligent decision-making processes. The increasing complexity of urban environments and demand for efficient transportation systems have driven significant advancements in AI planning and verification techniques. Autonomous systems, such as self-driving taxis, are becoming integral components of modern cities, requiring sophisticated algorithms to manage their operations effectively. Two main companies are known to be testing self-driving vehicles currently: Waymo owned by google and Cruise (approval granted in 2022) owned by General Motors in the United States, (The Taxi Centre, 2023). Both companies have established a fleet of fully automated, driverless vehicles currently being evaluated for their capability to safely pick up and drop off passengers within designated areas. Cruise's operation consists of 30 electric vehicles within San Francisco operating between 10am and 6pm while Waymo's operation is in testing phase in limited areas in Phoenix Arizona.

The complexity of AI planning arises from the need to generate, optimize, and verify plans in dynamic and uncertain environments. This requires the development of sophisticated algorithms with reasoning capabilities about the states and actions of the system while considering various constraints such as time, resources and external factors. Through a series of experiments, this report evaluates the performance of different AI planning algorithms, including A*, Depth First Search, and Breadth-First Search, under different environmental conditions. The experiments test the system's ability to avoid collisions, adapt to static obstacles, comply with traffic rules, and scale efficiently with increasing optional loads. Traditional planning methods provide foundational frameworks; however, they are limited in handling the complexities of real-world scenarios which has led to the exploration of more advanced techniques that include Satisfiability Modulo Theories (SMT) and Bounded Model Checking (BMC).

Researchers have in many cases turned to Satisfiability Modulo Theories (SMT) and Bounde Model Checking (BMC) to address these limitations. SMT extends the classical Boolean satisfiability problem (SAT) by incorporating theories that govern the behaviour of non-Boolean elements, such as real numbers, arrays, and integers. Such rich mathematical theories allow for the expression of more complex and realistic constraints. BMC is a verification technique responsible for checking the correctness of models within a specified bounded number of steps. Due to BMC's ability to provide counterexamples when properties are violated, this makes it a powerful tool for identifying potential issues in system designs.

This report takes a shallow history in BMC, SMT, SAT and AI Planning. It introduces the concept of AI planning giving a brief history of the module and key facts including planning terminology, planning algorithms (including constraint planning and Machine Learning) and Planning Domain Languages utilized in AI planning. Section 2 then dives a little deeper into Boolean satisfiability problem (SAT) and highlights it's limitations. This creates the bridge to introduce the SMT module in section 3 as the preferred technique to address some of the limitations of SAT. The report broadens the module and presents four popular SMT solvers with images of their architecture, i.e., Z3, CVC, MATHSAT and Yices.

The focal point of this report is the comprehensive study of the application of SMT and BMC in planning, and verification with a focus on a simulated taxi dispatch system. Built using Python programming language, the simulation models a dynamic urban environment with intelligent taxis operating autonomously. The aim of the system is to optimize taxi operations by maximising revenue and ensuring traffic regulations compliance. Z3 SMT solver (developed by Microsoft) has been implemented as the preferred solver to achieve these objectives due to its ability in handling complex logical constraints. BMC has also been employed for formal verification of safety properties.

Through various experiments, the report evaluates the system's performance under different environmental conditions. This includes the performance of various AI planning algorithms such as A*, Depth-First Search and Breadth First Search. The experiments are designed to test the system's ability to adapt to static obstacles, compliance with traffic rules, collision avoidance, and scalability with increasing operational loads. The

evaluations provide an insight into how different planning algorithms and verification techniques can be applied to enhance the reliability of autonomous systems in real-world scenarios.

The findings of this report contribute to a wider field of AI planning by showcasing the potential of formal verification techniques in tackling some of the most pressing challenges in autonomous decision making and intelligent system design.

SECTION 1

1.1 AI Planning

What is AI planning or planning in AI?

Also known as scheduling or automated planning is an extension of Artificial Intelligence that focuses on the development of sequences or strategies of action to attain specified goals. It is essentially used for execution by autonomous robots, intelligent agents and unmanned/automatic vehicles. AI planning involves determines a sequence of actions for a system described in a declarative manner to achieve its objectives while maximizing overall performance matrix, (klu.ai, 2023).

A basic planning problem typically involves an initial world description, a partial goal description, and a set of actions (or operators) which transform one partial world state into another. The problem can be made more complex by incorporating elements like temporal constraints, uncertainty, or the need to optimize specific properties. The goal is to find a sequence of actions or plan that transitions from initial state to goal state.

1.2 Planning Terminology

An AI planning system generates a plan as a solution to a specified problem, which is defined by an initial state and goal state, (klu.ai, 2023). The initial state describes the current world, while the goal state represents the desired outcome after the plan's execution. The world in which planning occurs is termed the application domain. Goals can often be broken down into simpler subgoals. Plans are composed of operator schemata, which define actions in terms of their preconditions and effects. These schemata describe possible actions and can be instantiated by replacing variables with specific constants. The term "operator" can refer to both the schemata and the instances of these actions. Actions directly executable by the planner are called primitive actions or primitives. This framework helps structure the planning process and ensures that the system can systematically approach complex problems.

In AI planning, Strips operators originating from the early Strips planning program are fundamental concepts that describe actions using three elements: preconditions, add-lists, and delete-lists. Preconditions are facts that must hold for an action to be executed. When an operator's preconditions are met in a given state, the operator can be applied, resulting in a new state formed by removing all elements in the delete-list and then adding those in the add-list. Despite newer planning systems evolving beyond this model, the terminology of Strips operators remains standard in AI planning literature.

A plan is an organised sequence of operators that, when executed in order, transforms the initial state into a goal state. A plan is considered a solution if it is applicable in the initial state and

achieves the goal upon execution. This applicability is assessed through temporal projection, where the sequence of operator applications is simulated to ensure the goal is met in the final state. AI planners typically input a set of operator schemata and a problem defined by an initial state and a goal. The planner's task, known as plan generation or synthesis is to produce a plan that satisfies the goal. Planners can be domain-independent, meaning they work across various application domains, (klu.ai, 2023).

Planners differ in how they define their search space. Early planners (pre-1975) focused on state-space planning, where points in the search space represent world states at different times and solutions are sequences of operators transitioning from the initial state to the goal state. Late planners (post-1975) emphasized partial plan space, where points represent partially elaborated plans that evolve through transformations, leading to a complete plan that achieves the goal. Action-ordering approaches which define plans by the temporal ordering of actions, have become prevalent due to their effectiveness in handling complex domains.

1.3 AI Planning Algorithms.

In the domain of Artificial Intelligence, planning algorithms are employed to formulate strategies or action plans for execution by intelligent agents, autonomous robots and unmanned vehicles. These solutions are intricate and need to be identified and optimised in multidimensional space.
Various types of planning algorithms are utilized in AI, including:

1.3.1 Forward chaining state space search

Explain by (www.javatpoint.com, n.d.), can also be referred to as forward reasoning or forward deduction and is a form of reasoning method that begins with atomic statements in the knowledge base and uses inference rules. These include Modus Ponens, in a forward direction to derive additional information until the desired goal is reached.

1.3.2 Backward chaining search

Also termed as backward deduction or backward reasoning approach starts with goal and works in reverse linking rules to identify known facts that support the achievement of that goal, according to (www.javatpoint.com, n.d.). It is identified as a top-down approach and relies on Modus ponens inference rule. The main is divided into sub-goals to validate the facts as true. This approach is termed as goal driven since the selection and application of rules is determined by the list of goals. It primarily employs a depth-first search strategy to establish proof and has been utilised in automated theorem proving tools, game theory, proof assistants, inference engines and a variety of other AI applications.

1.3.3 Partial-order planning

A partial-order planner maintains a flexible sequence of actions, only establishing a specific order between actions when necessary. Although sometimes referred to as a non-linear planner, this term is somewhat misleading since these planners often result in a linear plan. (Partial-Order Planning, n.d.).

1.3.4 Classical planning

Classical AI planning primarily focuses on generating plans to achieve predefined goals in environments where most relevant external conditions are known and remain stable, ensuring that the success of the plan is not impacted by changes in the outside world, (GeeksforGeeks, 2024). Classical Planning involves an agent leveraging the structure of a problem to develop intricate action plans. The agent typically engages in three key tasks: Planning, where the agent formulates a plan after understanding the problem; Acting, where it decides and executes the appropriate actions; and Learning, where the agent gains new knowledge from the outcomes of its actions.

Planning Domain Definition Language is used to represent actions within a planning problem through a unified action schema. PDDL is a formal language designed for specifying planning and scheduling problems in artificial intelligence. It standardises how elements in automated planning systems are expressed, allowing for the definition of a planning domain that includes a set of actions, each with associated preconditions and effects. These actions serve as fundamental components for creating plans. PDDL also defines the problem within the domain by outlining the initial state and the desired goal state to be achieved, facilitating the automated generation of solutions. PDDL describes four essential elements in a search problem:

1. **Initial state**: which represents each state as a conjunction of ground functionless atoms.
2. **Actions**: defined by a set of action schemas that implicitly define the ACTION() and RESULT() functions.
3. **Result**: The outcome achieved through the set of actions executed by the agent.
4. **Goal**: similar to a precondition, and is expressed as a conjunction of literals, where each literal can be either positive or negative. The goal defines the desired end state that the planning process aims to achieve, guiding the selection and application of actions within the planning domain.

However, many planners do not fully support all elements of any version of PDDL, and they often have unique quirks that can lead to incorrect interpretation of PDDL constructs or require slight syntax variations that deviate from the official specification. For example, some planners implicitly assume that all arguments to an action must be distinct, while others may require action preconditions or effects to be expressed as conjunctions, even when dealing with a single condition or none. Additionally, most planners ignore the ‘: requirements’ section of the domain definition but may still fail to parse a domain if this section is missing or contains unrecognised keywords. To avoid issues, it’s advisable to use the simplest constructs necessary and always consult the planner’s documentation, (GeeksforGeeks, 2024).

1.3.5 Hierarchical planning

Described as a problem-solving approach where complex tasks are divided into smaller manageable sub-tasks. It arranges the sub-tasks within a hierarchy structure, with each level of the hierarchy representing a different degree of abstraction, (Singh, 2024). This approach allows AI systems to address specific actions step-by-step while keeping the

overall objective in focus. The key components of hierarchical planning include task decomposition, abstraction levels, refinement of tasks, temporal constraints, and resource allocation. These elements enable efficient problem-solving by ensuring tasks are executed in the correct order, with appropriate resource management. Hierarchical planning techniques include Hierarchical Task Network (HTN) planning, Hierarchical Reinforcement Learning (HRL), and Hierarchical State Space Search, each of which facilitates the breakdown of tasks and decision-making processes across different domains.

Applications of hierarchical planning are widespread in fields such as autonomous driving and robotics. In autonomous driving, it enables the vehicle to navigate complex environments by driving the process into long-term route planning, behavioural decisions, and real-time motion control. In robotics, hierarchical planning helps in manipulating objects and coordinating multi-robot systems for complex tasks, improving efficiency by focusing on manageable subtasks. The advantages of hierarchical planning include scalability, adaptability, reusability and modularity, which enhance the AI system's ability to handle large-scale problems and adapt to dynamic environments. This structure also promotes interpretability, making it easier to analyse and refine planning processes. (Singh, 2024).

1.3.6 Constraint planning

(Nareyek et al., 2005), describes constraint-based planners as defined by their approach to framing the planning problem through explicit constraints. The concept of incorporating constraints into planning first emerged with the MOLGEN planner. Contrary to traditional methods which rely on linear inequalities or propositional clauses, constraint-based planners utilize propagation techniques, a common solution method in Constraint Satisfaction Problems (CSPs). Constraint based planners are categorized into three distinct approaches.

1. Planning with Constraint Posting which utilizes Constraint Programming (CP) to solve subproblems within the planning process.
2. Planning with Maximal Graphs which constructs a large CSP that encompasses all possible planning options up to a certain plan size. It also demonstrates full interaction between value-based and structure-based aspects of the problem. However, this approach does not scale well for larger problems.
3. Completely Capturing Planning with CP which involves expressing the entire planning problem within the CP framework and requires extended CP frameworks to accommodate various possible graph structures. Like the maximal graphs approach, it exhibits full interaction between value-based and structure-based aspects of the problem.

More recent planners have been developed to build on this idea including systems like CPlan, parcPLAN, Descartes, GP-CSP, MACBeth, EUROPA, CPplan, Jussi Rintanen and Hartmut Jungholt, EXCALIBUR agent's planning system and the method of Eric Jacopin and Jacques Penon. However, nearly all planning systems could be considered constraint-based since even traditional total-order planners can be interpreted as using propagation techniques to eliminate infeasible options by spreading state information.

1.3.7 Machine learning

Mostly utilised in automated planning (AP) which refers to the process of generating a sequence of actions (a plan) that will achieve a specific goal from a given starting state. Machine Learning (ML) has been proposed as a solution to overcome the challenges in automated planning by automating the process of knowledge acquisition which is crucial for effective planning. Machine Learning techniques can be used to automatically define planning action models and search control knowledge, which can enhance the scalability and robustness of planning systems. One example where ML contributes to planning (AP) is in automatic learning of planning action models. Action models define the effects of actions within a planning domain and accurately specifying these models manually can be difficult and error prone. ML techniques can learn these models by analysing examples of successful action sequences (plans) in fully or partially observable environments, with deterministic or stochastic effects, (Jiménez et al., 2012).

For instance, in fully observable and deterministic environments, systems like LIVE and EXPO learn action models by monitoring the execution of actions and updating the models by monitoring the execution of actions and updating the model based on observed outcomes. In environments where actions have probabilistic effects or where the state is partially observable, more sophisticated ML techniques such as Stochastic Logic Programs or Inductive Logic Programming (ILP) are used to learn complex action models that account for uncertainty.

Learning of search control knowledge is another critical area in planning where ML techniques can be applied. Search control knowledge guides the planner in selecting the most promising actions during the planning process. Search control knowledge can significantly improve the efficiency of planning algorithms by reducing the size of the search space and focusing the search on more promising paths. ML techniques can learn this knowledge from past planning experiences using approaches like generalised policies, heuristic functions and macro-actions.
Macro-actions can combine frequently occurring action sequences into a single action, reducing the depth of the search tree. Learning heuristics involves creating functions that estimate the cost of achieving goals from a given state, which can guide the planner more effectively than domain-independent heuristics. Generalised policies provide guidelines that map planning contexts to preferred actions which can be used directly during the planning process to make decisions.

Integrating Machine Learning techniques into Automated Planning provides a powerful approach to overcoming some of the most significant challenges in the field. ML enhances the scalability, robustness and efficiency of planning systems by automating the acquisition of action models and search control knowledge, making them more suitable for complex, real-world applications, (Jiménez et al., 2012).

Planning algorithms are applied in a wide range of fields, such as cybersecurity, dialog systems, transportation, logistics and many more.

1.4 Planning domain modelling languages

Planning domain modelling languages, such as the Planning Domain Definition Language (PDDL) are crucial in defining and standardizing the problems that AI planning systems address. PDDL was introduced in 1998 and has become the most prominent language for this purpose, evolving with each AI planning competition. Its standardization has facilitated more reusable

and comparable research, though it may lack the expressive power of some domain-specific languages. PDDL is designed to specify a wide range of planning and scheduling problems, which are then processed by planning software. The planner, using appropriate algorithms, generates a solution plan, typically ordered, although the exact format of the output is not dictated by PDDL itself, (klu.ai, 2023).

SECTION 2

2.1 The Boolean satisfiability problem (SAT)

SAT involves determining whether a Boolean logic formula can be satisfied, (Hořeňovský, 2018). Specifically, a formula is considered satisfiable if at least one assignment of true or false values to its variables that makes the formula evaluate as true. If no such assignment exists, the formula is deemed unsatisfiable. SAT is particularly intriguing because a variant of it was the first problem to be proven NP-complete. This means that many other computational problems can be efficiently translated into SAT, solved as SAT problems and then the solutions can be converted back into solutions for the original problems. For instance, the widely discussed dependency management problem is NP-complete and can be translated into a SAT problem, just as SAT can be translated back into a dependency management scenario.

Previously, according to (Rintanen, 2014), NP-complete problems were regarded as virtually unsolvable with the phrase "It is N-complete, don't bother trying to solve it", except in the simplest cases. This reflected the believe that such problems were practically unsolvable. However, the attitude has shifted to "It is NP-complete, you might as well solve it," due to the significant advancements on SAT solving since the mid-1990s have changed this perspective making these problems more approachable. These breakthroughs have led to major developments in state space search, which are now being applied in the creation of intelligent systems. The impact of these advancements is beginning to extend into other fields as well, including probabilistic reasoning and machine learning. SAT has found numerous industrial applications with more emerging over time. Research into extensions of SAT is now a vibrant area within automated reasoning and AI. Many significant problems in AI and computer science are NP-complete, particularly those involving combinatorial challenges and finding optimal solutions.

Applications of SAT in Computer Science span various domains, including:

- **Reachability problems** such as model-checking in Computer Aided Verification for sequential circuits and software, and planning in Artificial Intelligence. In addition to discrete event systems diagnosis for system behaviour analysis.
- **Integrated circuits** applications consisting of automatic test pattern generation (ATPG), equivalence checking, logic synthesis and fault diagnosis.
- **In biology and language**, SAT is used for haplotype inference, computing evolutionary tree measures and the construction of phylogenetic trees.

Today, SAT typically refers to CNF-SAT, a Boolean satisfiability problem where formulas are expressed in conjunction normal form (CNF). In this form, the entire formula is a conjunction (AND) of clauses and each clause is a disjunction (OR) of literals. Here are some examples; (A V B) (B V C), (A V B) C, A V B and A C. In practice, there are two methods for passing a formula to a SAT solver: through the semi-standard DIMacs file format or by using the SAT solver as a library. While using a solver as a library, like MiniSat for C++ is often preferred in real-world applications for its integration flexibility, the Dimacs format is useful for quickly prototyping applications and testing different solvers' performance on specific problems, (Rintanen, 2014).

SECTION 3

3.1 SATIFIABILITY MODULO THEORIES

What Satisfiability Modulo Theories?
Satisfiability Modulo Theories (SMT) is a framework in mathematical logic and computer science which extends the classical Boolean satisfiability problem (SAT). A satisfiability problem (SAT) is a problem where the challenge lies in establishing whether there exists an interpretation that can satisfy a given formula. In various domains of computer science, such as verification of software and hardware, numerous critical problems can be translated into the task of verifying the satisfiability of a formula within a specific logic framework. The language for SAT differs from that of SMT where SAT use Boolean logic and SMT use first-order logic, (De Moura and Bjørner, 2011).

SMT operates within the framework of first-order logic (Barrett and Tinelli, 2018), but incorporates theories that define the behaviour of non-Boolean elements within the formulas. These theories allow the SMT solver to reason about formulas in a more structured and theory specific manner by constraining the interpretation of function and predicate symbols. The ability to handle a wide range of theories is what gives SMT flexibility proving it to be suitable for complex problems that cannot be easily expressed in pure propositional logic.

A central aspect of SMT is its grounding in decision procedures for first-order logic, which have been refined and adapted to work efficiently within specific theories. These procedures form the backbone of SMT solvers when combined with the powerful techniques of SMT solvers, (Barret and Tinelli, 2018). The development of SMT-LIB, a standard language and benchmark suite for SMT solvers has played a crucial role in advancing research and ensuring the reliability of these solvers across different applications.

3.2 Application of SMT in Model Checking

The use of Satisfiability Modulo Theories (SMT) has greatly enhanced model checking, a technique for verifying system correctness. In model checking, systems are represented as state transition systems, with each state described through logical formulas. SMT solvers are employed to assess the satisfiability of these formulas under various conditions, allowing for error detection or property verification within the system. SMT is especially valuable in bounded model checking (BMC), where the aim is to identify counterexamples to system correctness within a defined depth. By encoding the transition system and the properties to be verified as SMT formulas, solvers can efficiently explore the state space. Similarly, in k-induction methods, SMT solvers verify system invariants by checking if specific conditions hold across all possible states, (Barrett and Tinelli, n.d).

3.3 First-order logic

According to (Barret, 2008), first-order logic also known as first-order predicate calculus or first-order functional calculus is a type of logic where the predicate of a sentence can only refer to a single subject. first-order syntax or formulas consist of logical symbols and parameters.
Logical symbols include;

- Quantifiers ∀ and ∃
- Propositional connectives ¬,
- Variables x, y, z, ...and
- Relational symbols R, P, Q

Parameters include constants, predicates, functions and equality symbols.
A first order language must first define its parameters. Each function symbol and predicate has an associated arity, a natural number indicating the number of arguments it takes. Constant symbols can be considered as functions with an arity of 0. The initial key concept in defining well-formed formulas is that of terms explained as expressions which denote objects. When applying the concept of terms, prefix notations are used to avoid ambiguity.

3.4 SMT Solvers

SMT solvers are pivotal in verification technology as they offer a means to reason about systems modelled at higher abstraction levels than Boolean logic, which is the limitation of SAT solvers, (Barret, 2008). The primary reasoning behind the development of SMT solvers is to create verification engines that maintain the automation and efficiency of SAT solvers while natively handling more complex expressions and theories. Unlike SAT solvers which use Boolean logic, SMT solvers use first order logic which includes Boolean operations and extends to more complex expressions involving constants, functions and predicate symbols. Expressions in first order logic are constructed using logical symbols such as propositional connectives and quantifiers as well as parameters like predicate and function symbols.

The semantics of SMT solvers are based on first-order logic, where the truth of a formula is determined by models or structures. A signature defines the set of non-logical symbols while a model provides a domain and mapping of these symbols to elements and relations within the domain. The SMT framework consists of various theories which include the theory of equality, arrays, inductive data types, reals and integers. Each theory has selective properties that define how formulas are evaluated and verified which enable SMT solvers to handle a wide range of verification tasks effectively.

Theory solvers are also an integral component of SMT solvers and are responsible for determining the satisfiability of sets of literals. Countable algorithmic approaches are employed including congruence closure for equality reasoning and Shostak's method for combining decision procedures. The Nelson-Oppen method is also utilised for combining theories with disjoint signatures. These algorithms are essential for managing complex formulas involving multiple theories and ensuring efficient decision-making within SMT solvers. SMT solvers are widely used in various verification scenarios, from software verification to hardware design and analysis. Practical examples demonstrate how SMT solvers handle formulas across different theories and the role of decision procedures in verifying complex systems. The combination of different theories allows SMT solvers to be highly adaptable and effective in addressing diverse verification challenges, (Barret, 2008).

SMT solvers are increasingly becoming the preferred tool for a growing range of verification applications and in this section, we introduce different SMT solvers, mostly focusing on the popular ones. The most recommended SMT solvers include Z3, Yices, MathSAT, and CVC5 which have been recognised at competitions.

3.3.1 Z3 solver

According to (Wintersteiger, n.d.), Z3 is an efficient SMT solver developed by Microsoft, and it possesses specialised algorithms to solve background theories. It is an efficient SMT solver widely used in software analysis, symbolic execution tools and verification. We can go in-depth and introduce the key capabilities of Z3Py.

- Constraint Solving: Z3Py allows users to define and solve complex systems of constraints involving various types of variables which include integers, Booleans and real numbers. The **'solve'** function is central to this capability enabling the resolution of constraints that involve arithmetic, logical operators and more.
- Z3 supports a range of data types including bit-vectors, real numbers and integers. With such versatility, Z3 can handle different kinds of mathematical operations from basic arithmetic to more complex polynomial constraints.
- It can simplify mathematical and logical expressions resulting into an easier interpretation and analysis of formulas. Functions such as '**simplify'** are utilised to minimize expressions to their simplest form, facilitating more efficient problem solving.
- A wide array of Boolean logic and operations which are essential for building and solving logical constraints within a problem are supported by Z3Py. These include **'And', 'Not', 'Or', 'Implies'**, among others.
- Z3 provides a more advanced Solver API beyond the simple **'solve'** command that allows for exploring different scopes of constraints, backtracking and, incremental solving which is useful in applications where the problem set evolves over time.
- Various arithmetic operations are handled by Z3Py therefore supporting both real and integer numbers. It can solve nonlinear polynomial constraints and work with large numbers with the inclusion of irrational algebraic numbers.
- To allow precise modelling of computations performed by CPUs and in low-level programming languages, Z3 supports machine arithmetic through bit-vectors. It handles signed and unsigned bit-vectors and provides a variety of bit-wise operations.
- Following the solving of a set of constraints, Z3 can provide models to satisfy these constraints. This feature is essential to understand the possible solutions to a problem, allowing for deeper analysis and verification.
- Z3Py allows the definition of uninterpreted functions and constants which are utilised in modelling problems where some functions or variables do not have a predefined interpretation. Due to this flexibility, Z3 is enabled to handle a wide range of abstract problems.
- Python's list of comprehensions can be leveraged by Z3Py to create and manipulate lists of variables and expressions efficiently. It avails functions to create vectors of variables which can simplify the management of large sets of constraints.
- Z3Py can be implemented locally done by including in python scripts, making it available for offline use which allows for integration into different development workflows and environments.
- Application in Real-World Problems: it can be extended to solve practical problems such as kinematic equations, optimization of package installations and puzzles like the Eight Queens problem and Sudoku. The provided examples amplify and illustrate the power and versatility of Z3Py in handling real world scenarios.

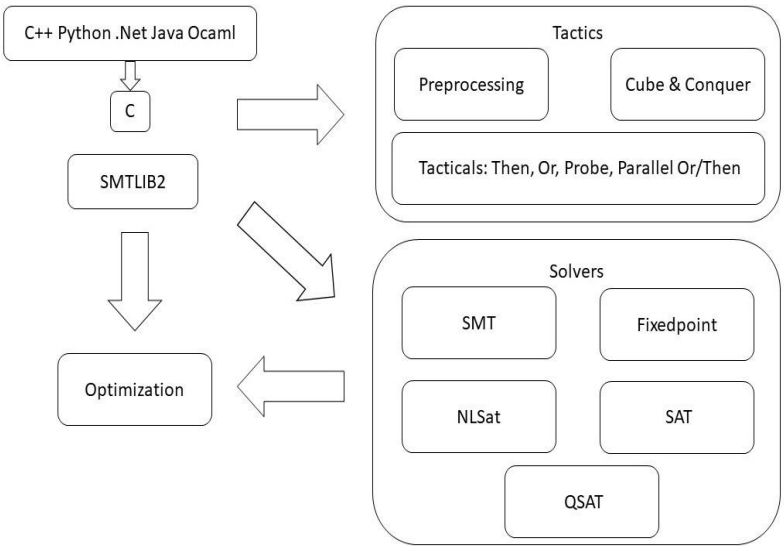


Figure 1 shows the overall architecture of Z3, by (Wintersteiger, n.d.).

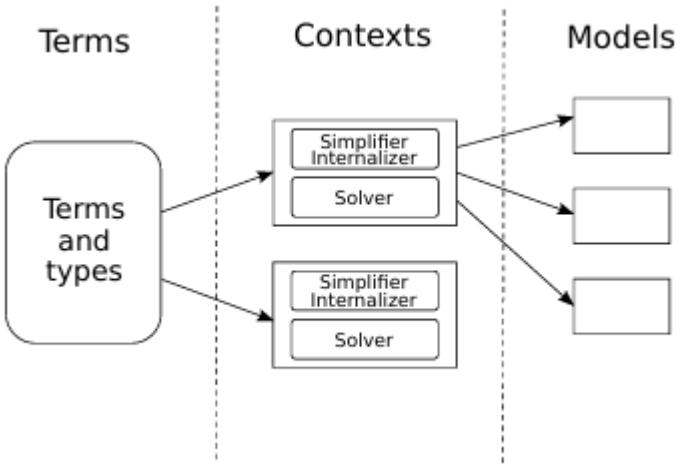
3.3.2 Yices

Yices is a lineage of formal verification tools originating from the decision procedures developed by Shostak at SRI in the 1980s which have evolved into current time SMT solvers, Dutertre, 2014). Released in 2006, Yices has undergone significant changes culminating in the re-implementation of Yices 2 aimed to improve modularity, performance and usability. Yices 2 simplifies the logic of Yices 1 by removing complex type constructs. It has the capability to declare scalar types and new uninterpreted types. Yices supports primitive types like integers, bitvectors, Booleans and reals. The system encourages the use of subtyping which enables arithmetic terms of integer and real types to be mixed contrary to SMT-LIB 2.0 which treats these as disjoint types. Yice's logic is compatible with arithmetic and bitvector logics as per SMT-LIB 2.0, with extensions to support tuples and more general function-update operations making Yices 2 flexible for various formal verification applications.

Yices 2 uses a Boolean satisfiability solver and specialised solver for four main theories: uninterpreted functions with equalities, bitvectors, arrays and linear arithmetic. Solvers can be selected based on problem's requirements or combined with extensions for theory solver interaction. The uninterpreted function solver depends on a congruence closure algorithm while the arithmetic solver employs the Simplex method with additional specialised solvers for integer and real difference logic fragments. A "bit-blasting" approach is employed by the bitvector solver converting constraints to a Boolean SAT problem and the array solver uses classical axioms for array updates, (Dutertre, 2014).

The system architecture of Yices consists of three main modules for handling terms and types, contexts and models. A comprehensive API that supports various operations including types, pretty printing and constructing terms is responsible for managing the global database of terms and types. Yices system prioritises memory efficiency achieved through compact data structures and hash-consing for subterm sharing. A central data structure within Yices 2 i.e., Contexts, stores assertions to be checked for satisfiability allowing for dynamic configuration and solver selection, (Dutertre, 2014).

Figure 2 shows the Yices architecture by (Dutertre, 2014).



3.3.3 MATHSAT

MATHSAT is a sophisticated SMT solver developed through a collaboration between FBK-IRST and the University of Trento over the past decade. According to (Cinatti et al, 2013), MATHSAT5 is the latest iteration of this tool, and it introduces significant advancements and novel features compared to its predecessor, MATHSAT4. MATHSAT5 is designed to incorporate a wide range of SMT-LIB theories and their combinations thereby providing essential functionalities such as unsatisfiable cores, AllSMT and interpolation. However, MATHSAT5 is limited in its support for quantifiers.

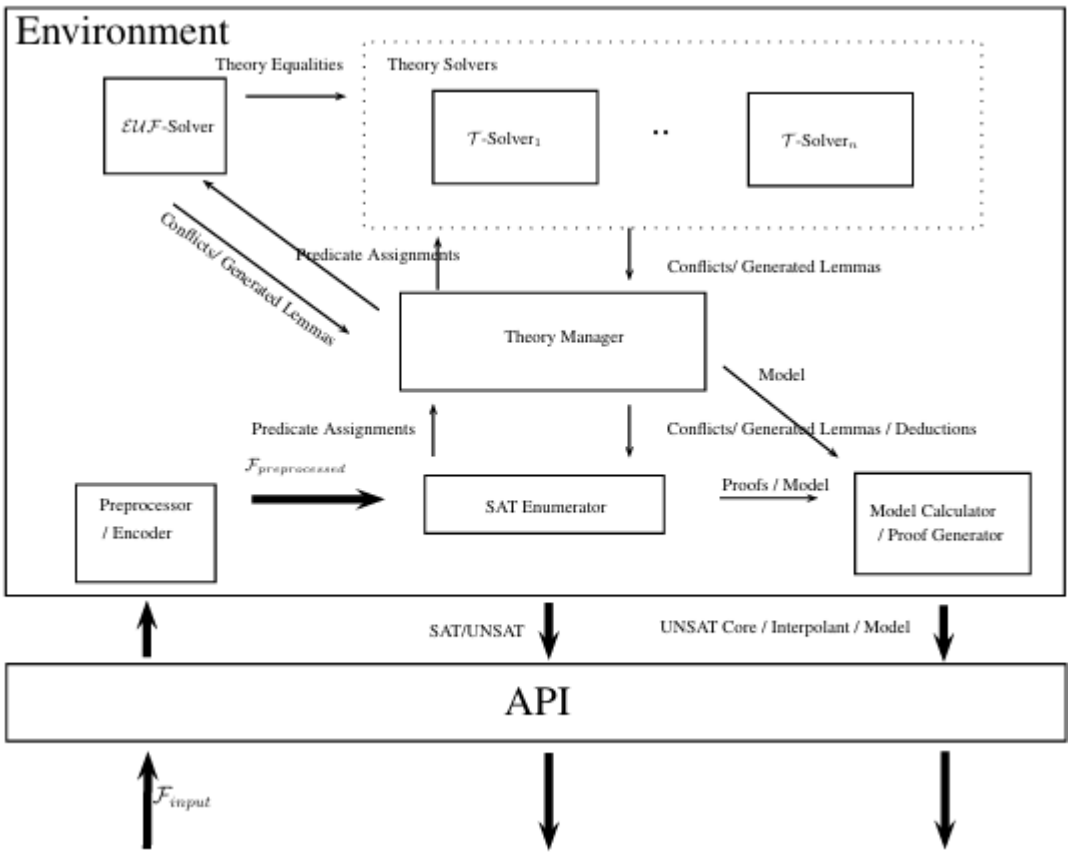


Figure 3 shows the MATHSAT5 architecture, by (Cinatti et al, 2013).

(Cinatti et al, 2013) describes the MATHSAT5 architecture as designed to facilitate efficient SMT solving through a coordinated environment that manages various solver components, including constraint encoding, preprocessing, theory solvers and SAT engine. The environment acts as the central hub, handling memory garbage collection, and coordinating the interaction between components.
The preprocessor normalises formulas and inline constants to simplify the input before processing. The constraint encoder converts inputs formulas into CNF and handles constructs not directly supported by the core components such as term-level if then-else structures.

At the core of MATHSAT5 are the SAT engine and theory solvers, which work together using the lazy/DPLL(T) approach. The SAT engine can either be native MINIST-style solver or a third-party pluggable SAT solver. The native SAT engine supports Boolean formula simplifications like variable elimination, subsumed Clause Removal, and Backwards Subsumption, all adapted for correctness in SMT contexts.
The theory manager serves as the interface between the SAT engine and individual theory solvers, ensuring modular integration and allowing for the easy addition or removal of solvers. The architecture also includes a model and proof generator that produces models for satisfiable formulas and refutation proofs for unsatisfiable ones, combining Boolean and theory-specific reasoning.

This modular and flexible design enables MATHSAT5 to support a wide range of theories and functionalities while maintaining efficiency and adaptability. MATHSAT5 is available for use and has been adopted in various internal projects as well as by several industrial partners, reflecting its utility and effectiveness in formal verification and other related applications.

3.3.4 Corporate Validity Checker (CVC)

According to (Barbosa et al., 2022) Cvc5 builds on the architecture of its predecessor cvc4, while introducing significant enhancements. It supports a wide range of theories including all SMT-LIB standards and several non-standard theories like separation logic, sequences, relations and finite sets. The tool also introduces higher-order reasoning and syntax-guided synthesis (SyGuS). With a mostly new development team, cvc5 features updated documentation, easier installation and improved APIs. It remains an open-source under the 3-clause BSD license and is compatible with major platforms like Linux, macOS and Windows. The name change reflects its evolution and modified capabilities since cvc4.

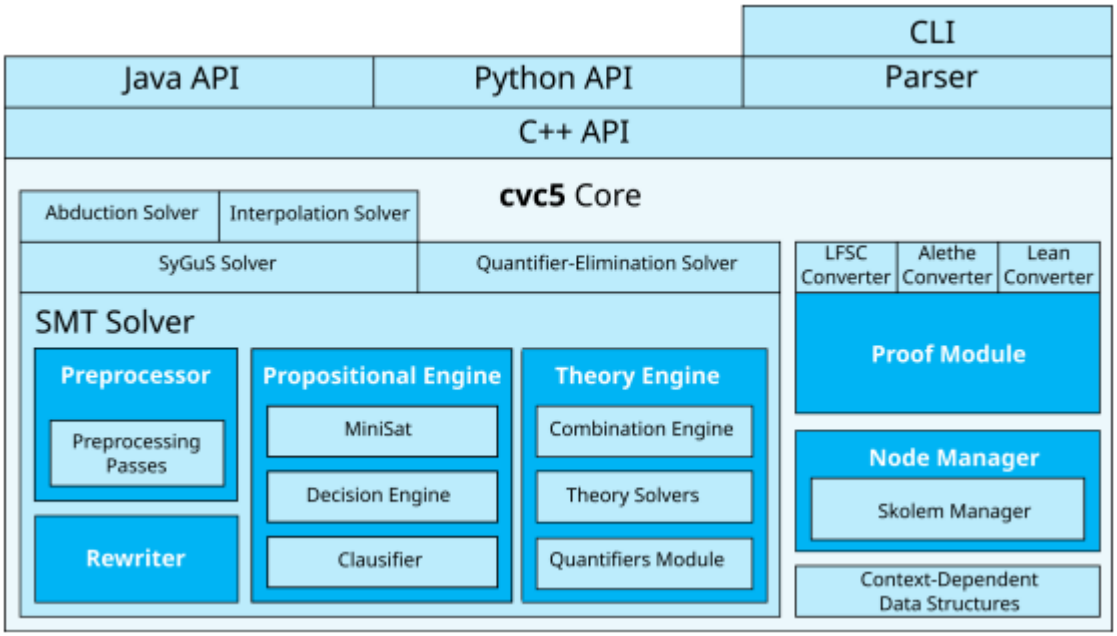


Figure 4 shows the architecture of CVC5, by (Barbosa et al., 2022).

Architecture of cvc5

The architecture of cvc5 revolves around its SMT solver, which utilises the CDCL(T) framework and a customised MiniSat propositional solver. Its key components include the Rewriter, Preprocessor, Propositional Engine, Theory Engine, and various additional solvers for advanced features like abduction, interpolation, syntax-guided synthesis (SyGuS), and quantifier elimination. These modules extend cvc5 beyond standard satisfiability checking. The system provides a C++ API, a command-line interface supporting various input languages and can output formal proofs in multiple formats, enhancing its usability for diverse applications, (Barbosa et al., 2022).

Preprocessor: Before any satisfiability check, cvc5 applies a series of transformations to each input formula. These include [1]required normalization passes, [2]optional simplification passes and [3]optional reduction passes, that transform the formula from one logic to another like converting non-linear integer arithmetic to bit-vector problems. Cvc5 implements 34 preprocessing passes, which are self-contained and can be modified without affecting the SMT solver engine.

Rewriter: It converts terms into semantically equivalent normal forms during solving. It categorises rewritten rules into required and optional types and maintains a cache to avoid redundancy. In addition, it simplifies the implementation of other components by ensuring all terms are normalised. In specific contexts like SyGuS, an extended Rewriter is used to apply more sophisticated rewriting rules. Automated improvements to the Rewriter are facilitated through a workflow that detects and suggests new rewrite rule candidates using the SyGuS solver.

Propositional Engine: The core CDCL(T) engine handles the Boolean abstraction of input formulas and generates satisfying assignments. It consists of a Classifier which converts Boolean abstractions into Conjunctive Normal Form (CNF), and a customized MiniSat SAT solver. Enhancements to MiniSat include resolution proofs, native support, and a Decision Engine for customised decision Heuristics. The Propositional Engine interacts with the Theory Engine by asserting theory literals during solving process and adjusting its efforts based on model completeness.

Proof Module: Developed to replace the incomplete system of CVC4, the cvc5 Proof Module ensures minimal overhead during proof production while maintaining detailed proofs for efficient checking. It supports both eager and lazy proof production and can emit proofs in various formats, including LFSC, Lean 4, Isabelle/HOL, and Coq.

Node Manager: In cvc5, formulas and terms are represented as nodes within a directed acyclic graph managed by the Node Manager. It uses hash consing for memory efficiency and manages Skolem symbols introduced during solving. The Node Manager also handles reference counting and reuses constants where possible to optimize performance.

Context-Dependent Data Structures: cvc5 optimizes multiple satisfiability checks by using context-dependant data structures that automatically save and restore state changes. These structures improve memory management and overall system performance through efficient context-level memory allocation.

SECTION 4

4.1 BOUNDED MODEL CHECKING

Model checking is a verification technique that involves verifying properties of state transition systems by exploring their state transition graphs introduced around 18 years ago, (Clarke et al, 2001), which makes that about 41 years at the present time (2024). These properties are expressed using temporal logic, which allows reasoning about the ordering of events over time. Temporal logic is powerful because it enables assertions about the future states of a system without directly introducing time. Model checking gained popularity in the industry during the late 1990s due to its rich specification language and high degree of automation, leading to its adoption by various CAD companies.

Initially, model checking used explicit representations of state transition graphs and efficient graph traversal techniques. However, these methods were limited by the state explosion problem where the number of system states grows exponentially with the number of components, making it difficult to manage systems with more than a million states. This limitation rendered the technique unsuitable for most industrial applications, particularly in hardware design.

In the 1990s, symbolic model checking emerged as a solution to the state explosion problem, utilizing Binary Decision Diagrams (BDDs) to perform breath-first searches of state spaces. BDDs allowed for an order of magnitude increase in the size of designs that could be verified, such as the Futurebus+ Cache Consistency Protocol. Despite this improvement, BDD-based model checkers still struggled with larger designs, limiting their industrial application. Enhancements in BDDs, abstraction and compositional reasoning improved the capacity of these tools, but they remained somewhat fragile when handling typical industrial designs, (Clarke et al, 2001).

Bounded Model Checking (BMC) involves creating a Boolean formula that is satisfied if a finite sequence of state transitions in a system can reach specific states of interest. Recently, bounded model checking using satisfiability solving has shown promise particularly for safety and liveness properties. The process examines all path segments of a given length, k, and continues with larger k if no satisfying path is found. SAT solvers like PROVER, SATO, and GRASP handle these formulas, offering advantages such as not requiring exponential space and quickly checking large designs. Unlike BDD-based model checking, which uses memory-intensive

breath-first search, BMC can find minimal-length paths, aiding in understanding generated examples. Additionally, SAT tools typically require less manual intervention compared to BDDs. This method can effectively check invariants and detect counterexamples with less manual intervention than BDD-based methods, making it more attractive for industrial use.

However, BMC is not without its limitations. (Clarke et al, 2001) describes It as generally incomplete, meaning it cannot always guarantee a true or false outcome for every specification. Especially as the propositional formula grows with each time step, hindering the ability to find long counterexamples. Despite these drawbacks, BMC is valuable complement to existing verification techniques, particularly in finding bugs or confirming correctness in cases where other methods may fail. Prior to BMC, SAT-based decision procedure had been applied in hardware verification, specification logics, railway control systems, and AI planning.

SECTION 5

5.1 SMT AND BMC IMPLEMENTATION

The program's goal is to simulate an autonomous taxi system within a dynamic urban environment. The program manages taxis navigating a city grid, handling obstacles, and traffic constraints. It ensures efficient fare pickups and drop-offs while avoiding collisions and optimizing routes based on real-time conditions. The system incorporates advanced algorithms for decision making, including probabilistic reasoning, model checking for safety, and dynamic pathfinding. The simulation visualizes taxi movements, traffic flow and parked cars, aiming to enhance understanding and development of autonomous vehicle systems in complex, real- world scenarios.

5.2 The Project Design

The project is a detailed simulation framework designed to model the operations of autonomous taxis within a dynamic, grid-based urban environment. The primary focus is on the navigation of taxis through a city grid, where they provide transportation services while avoiding obstacles, optimizing efficiency and maintaining safety. The simulation incorporates elements of artificial intelligence, specifically pathfinding algorithms such as Breadth-First Search (BFS) and utilises model checking techniques to ensure that the taxis operate safely and efficiently.

5.3 Aims and Objectives

Aim

The main aim of the project is to simulate and optimize the operations of autonomous taxis within a city grid. These taxis are tasked with picking up and dropping off passengers at various locations while avoiding collisions and navigating blocked nodes. The simulation is designed to maximize revenue for each taxi while ensuring safety by adhering to constraints such as maintaining a minimum safe distance between taxis and avoiding blocked nodes.

Objectives

1. **Route Optimisation**

To develop an efficient path-planning algorithm using the A* algorithm that allows taxis to navigate from their current location to their destination while considering factors such as roadblocks and speed limits. The algorithm should minimise travel time and maximize the profitability of each taxi.

2. **Collision Avoidance**
   To implement a robust collision avoidance mechanism using Z3 solver, ensuring that no two taxis occupy the same space at the same time. This involves formulating constraints that prevent taxis from colliding with each other while navigating the urban grid.

3. **Dynamic Obstacle Management**
   To incorporate dynamic elements such as roadblocks into simulation, testing the taxis' ability to reroute and adapt to changing conditions in real-time. This also includes visually representing blocked streets and junctions, allowing for clear analysis of traffic flow disruptions.

4. **Performance Analysis**
   To evaluate the performance of the taxi fleet under various scenarios, including multiple roadblocks and varying numbers of taxis. The simulation should provide insights into how these factors influence the efficiency, safety, and profitability of taxi operations.

By achieving these objectives, the program will offer valuable insights into optimising urban taxi operations, with potential applications in traffic management and autonomous vehicle development.

5.4 System Components

1. **Grid-Based City Model**

   The city is modelled as a grid, with each cell representing a node (either a junction or a street point). These nodes can be free, blocked, or occupied by a taxi. Blocked nodes are either permanently inaccessible due to obstacles or temporarily blocked due to traffic conditions.

2. **Autonomous Taxis:**

   Taxis in the simulation are autonomous agents that navigate the grid to transport passengers. Each taxi operates using pathfinding algorithm typically A*, to determine the shortest path to its destination. The taxis must dynamically adjust their routes if they encounter blocked nodes or need to avoid other taxis.

3. **Pathfinding and Navigation:**

   The A* algorithm calculates the optimal path for each taxi, considering the status of nodes (whether blocked or free). The algorithm ensures that taxis avoid paths that might lead to collisions or blocked nodes. When a taxi detects another taxi whin the unsafe distance or encounters a blocked node, it either reroutes or stops, depending on the situation.

4. **Model Checking for Safety:**

The simulation employs Bounded Model Checking (BMC) to predict potential collisions or safety violations over a set number of future time steps (N). BMC evaluates whether any taxi will move into a blocked node. If a potential safety violation is detected, the system takes corrective actions to avoid the issue.

5. **Revenue and Performance Metrics:**

   The performance of the taxis is measured based on total revenue generated, influenced by the number of successful fare pickups and drop-offs. The system also tracks and reports possible collisions, which impact the overall safety and efficiency of the simulation.

6. **Visualisation:**

   The simulation is visually represented using Pygame, with real-time displays of the grid, nodes, and taxis. Blocked nodes are highlighted, and taxis are shown moving through the grid, providing a clear view of the simulation's performance and safety features.

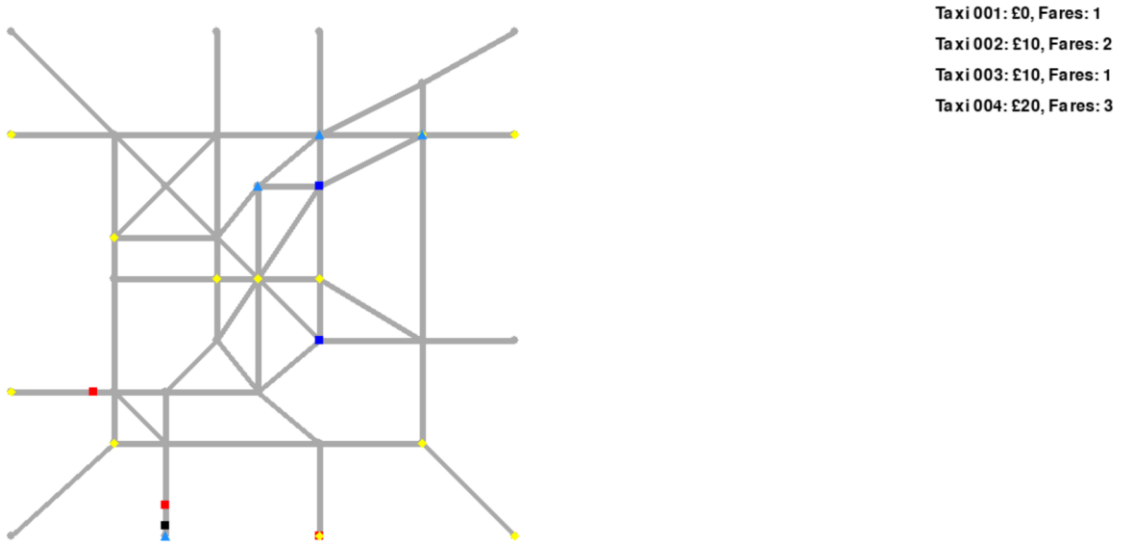
5.5 The Modules (Files) of the system

1. **Auto1.py:** Contains the main simulation loop for the taxi system. It manages the movement of taxis, fare handling, and the drawing of simulation elements (junctions, streets, taxis, and fares) on the screen. It handles the core simulation mechanics and display updates.
2. **Junc1.py:** This module defines and manages the junctions, and streets. It contains the logic for creating and managing junctions, including attributes like coordinates and the connections between junctions and streets.
3. **Tax.py:** This manages individual taxis in the simulation, including their movement, pathfinding (A*, BFS and DFS algorithms), fare handling and interactions with junctions and streets.
4. **NetWorld.py:** Contains high level simulation world logic, including placing fares, streets, and managing the interactions between different entities (taxis, streets, junctions, and obstacles/buildings).

5.6 The Constraints

Collision Avoidance: Taxis can not occupy the same grid position at the same time.
Obstacle Avoidance: Taxis must avoid collisions or spaces occupied by the obstacles (parked cars)
Traffic Regulations: Taxis must leave a specified gap between themselves.

5.7 The simulation



Visual representations (Elements):

- Junctions and streets are in the colour grey, and they are the permitted routes for the taxis to travel on.
- New fares are in the colour yellow and shaped like a polygon.
- Taxis are initially black but turn to red when they are carrying a fare. They randomly placed on either the streets or junctions at the start of the simulation.
- Dropped fares are indicated in the dodger blue colour, to indicate a fare has been dropped and at what junction it has been dropped at. After which the taxi turns back to black.
- At the right is the meter of every taxi, it keeps track of how many fares a taxi has picked up and the revenue it has accumulated so far.
- Obstacles are in blue, currently only two obstacles are in place, but a simple commenting out of the other obstacles can increase the number of obstacles to 6.

SECTION 6

6.1 Experiments

In this section we will outline the experiments designed to evaluate the performance, reliability, and safety of the taxi simulation. The experiments are constructed to test the system under various conditions, validate the integration of Bounded Model Checking (BMC), and assess the system's ability to handle dynamic constraints such as roadblocks/obstacles (parked cars) and traffic variations. All simulations are capped at 5 minutes dues to interest of time, and to also create a fair level of comparisons across all experiments.

6.1.1 Experiment 1: Performance of simulation with Collision avoidance (Avoiding collisions between taxis).

Objective

The objective of this experiment is to evaluate the effectiveness of the collision avoidance mechanism (collision avoidance constraint) in a simulated environment with varying number of taxis. The goal is to test whether the implemented collision avoidance strategy can prevent taxis from colliding with one another as they navigate the urban grid, particularly as the number of of taxis and fares increase. The experiment seeks to determine the robustness of the system's safety features under different conditions, ensuring that all taxis operate without collisions while completing as many fares as possible.

Procedure

Setup:

1. **Simulation Environment:** The simulation is set up with a grid environment where taxis operate, and fares are distributed randomly. The number of taxis is varied across three test cases (4, 6, and 8), and the number of fares is set to 20.
2. **Variable**: The number of active taxis in the simulation is changed in each test case
3. **N Steps:** The simulation runs for a varying number of steps (ranging from 3 to 15) to observe how the system performs over time.
4. **Collision Detection:** The collision avoidance system is monitored to ensure that no taxis occupy the same space at the same time.

Method:

1. **Run the Simulation:** For each configuration of the taxis, the simulation is executed, and data is collected on key metrics, including total revenue, peak memory usage, the number of completed fares, and whether any collisions are detected.
2. **Data Collection:** Record the following metrics for each test:
    - **Total Revenue:** The total earnings from completed fares.
    - **Peak Memory Usage:** The maximum amount of memory used during the simulation.
    - **Collisions Detected:** Whether any collisions were detected during simulation.
    - Number of Fares Completed: The number of fares successfully completed by the taxis.
    - **Pass/Fail Status:** Mark each test as a "Pass" if no collisions are detected and a "Fail" if any collisions are detected.
3. **Incremental Testing:** Increase the number of N Steps incrementally to observe the system's performance over long periods.

Evaluation and Analysis.

1. **Collision Avoidance Evaluation:** Analyse the effectiveness of the collision avoidance mechanism by observing the frequency of collisions detected as the number of taxis and steps increases. A higher frequency of collisions will indicate a weakness in the collision avoidance algorithm.

2. **Performance Metrics:** Evaluate how the number of taxis and the duration of the simulation (limited to 5 minutes) affect the total revenue and the number of completed fares. A drop in fare completion or revenue could indicate inefficiencies in the system as more taxis are introduced.

3. **Memory Usage:** Monitor the memory usage to ensure that the simulation remains within acceptable limits, even as the number of taxis increases.

4. **Scalability:** Assess the system's scalability by comparing the results across different taxi configurations and step intervals. Determine whether the system can maintain safety and efficiency as the complexity of the simulation increases.

Expected Outcome.

1. **Collision Avoidance:** The system is expected to avoid collisions, even as the number of taxis increases. A successful outcome would show that the collision avoidance mechanism is robust and reliable.

2. **Performance:** Total revenue and the number of completed fares should be maximised without compromising safety. Memory usage is expected to increase with more taxis, but it should remain within acceptable limits.

3. **Scalability:** The system should scale effectively, handling more taxis without a significant drop in performance or an increase in collisions.

Test 1
Number of Taxis = 4
Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Number of fares completed	Pass/Fails
3	£100	3.864011 MB	NO	10	PASS
5	£40	3.971079 MB	NO	4	PASS
7	£40	3.971079 MB	NO	4	PASS
9	£40	4.162874 MB	NO	4	PASS
12	£30	0.335861 MB	NO	3	PASS
15	£0	3.518244 MB	NO	0	PASS

Test 2
Number of Taxis = 6
Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Number of fares completed	Pass/Fails
3	£40	3.832446 MB	NO	0	PASS
5	£80	3.680702 MB	NO	0	PASS
7	£70	0.359624 MB	NO	0	PASS
9	£20	3.638506 MB	NO	0	PASS
11	£0	0.33382 MB	NO	0	PASS
13	£0	3.766459 MB	NO	0	PASS

Test 3
Number of Taxis = 8
Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Number of fares completed	Pass/Fails
3	£110	4.638866 MB	NO	11	PASS
5	£30	0.351379 MB	NO	3	PASS
7	£60	0.366941 MB	NO	6	PASS
9	£60	3.546691 MB	NO	6	PASS
11	£20	0.329195 MB	NO	2	PASS
13	£10	3.643116 MB	NO	1	PASS

Actual Outcome

1. **Test 1 (4 Taxis, 20 Fares):** The collision avoidance mechanism successfully prevents any collisions across all steps. The system maintained stable memory usage, with a peak of 3.97 MB, and completed a varying number of fares, reaching up to 10 fares in early steps. The test passed for all N Steps, indicating that the system handles 4 taxis effectively.

2. **Test 2 (6 Taxis, 20 Fares):** Similarly, no collisions were detected, but the number of completed fares was significantly lower, with no fares completed at some steps. Memory usage was consistent, peaking at 3.83 MB. The test passed, showing the system's ability to prevent colisions, though fare completion efficiency decreased.

3. **Test 3 (8 Taxis, 20 Fares):** The system continued to prevent collisions; however, the performance in terms of fare completion varied. The system completed up to 11 fares in the initial steps, but this number dropped as N Steps increased. Memory usage was slightly higher, peaking at 4.63 MB. The test passed which is evidence to prove that the system can prevent collisions even with more taxis, though the system's efficiency in completing fares decreased as complexity increased.

The experiment demonstrates that collision avoidance mechanism is effective across different taxi configurations, ensuring safety without collisions. However, as the number of taxis increases, the system shows a decline in fare completion efficiency, suggesting that future improvements could focus on optimizing both safety and performance for larger taxi fleets.

Experiment 2: Performance of simulation in Collision avoidance with varying number of Obstacles and N Steps.

Objective

The objective of this experiment is to evaluate the effectiveness of the obstacle avoidance mechanism within the simulation when faced with varying numbers of obstacles (parked cars) and different N Steps. The goal is to determine how well the taxis navigate the environment avoiding obstacles while maximizing the number of fares picked up and revenue generated. The experiment aims to assess the robustness of the system's ability to handle static obstacles, ensuring that the taxis operate safely and efficiently in the presence of these challenges.

Method/Procedure

Setup:

1. **Simulation Environment:** The simulation is configured with a grid environment, including 4 taxis, 20 fares, and 2 static obstacles (parked cars). The number of N Steps (3, 5, 7, 9, 11, 13) is varied to observe the impact over time.
2. **Variables:** The primary variables in this experiment are the number of obstacles and the number of steps in the simulation.
3. **Collision Detection:** The collision detection system is monitored to ensure that taxis do not collide with obstacles during the simulation.

Method:

1. **Run the Simulation:** For each configuration, the simulation is executed with the specified number of taxis, obstacles, and N Steps. Data is collected for total revenue, peak memory usage, collisions detected, and the total number of fares picked up.
2. **Data Collection:** Record the following metrics for each test:
    - **Total Revenue:** The total earnings from completed fares.
    - **Peak Memory Usage:** The maximum amount of memory used during the simulation.
    - **Collisions Detected:** Whether any collisions with obstacles were detected.
    - **Total Fares Picked:** The total number of fares successfully picked up by the taxis.
    - **Pass/Fail Status:** Mark each test as a "Pass" if no collisions are detected and a "Fail" if any collisions with obstacles occur.
3. **Incremental Testing:** Increase the number of N Steps incrementally to observe how the system's performance changes over time.

Evaluation and Analysis:

1. **Obstacle Avoidance evaluation:** Analyse the effectiveness of the obstacle avoidance mechanism by observing the number of collisions detected as the number N steps increases. A successful system should prevent all collisions with obstacles, even as the simulation progresses.

2. **Performance Metrics:** Evaluate how the presence of obstacles affects the total revenue, and the number of fares picked up. A significant drop in these metrics could indicate inefficiencies in the system's ability to navigate around obstacles.

3. **Memory Usage:** Monitor the memory usage like the experiment before. This is also important to ensure scalability and efficiency as complexity is added to the simulation in terms of obstacles (parked cars).

4. **Scalability:** Assess whether the system can handle the presence of obstacles without significant degradation in performance as the simulation runs for more steps.

Expected Outcome:

1. **Obstacle Avoidance:** The system is expected to successfully navigate around the obstacles without any collisions. A successful outcome would demonstrate the system's robustness in handling static obstacles.

2. **Performance:** The system should maintain high levels of revenue and fare pickups, indicating that the taxis can efficiently avoid obstacles while still fulfilling their primary function.

3. **Scalability:** The system should demonstrate the ability to scale, maintaining efficiency and safety as the number of steps increases.

TEST 1
Number of Taxis = 4
Number of Obstacles (Parked Cars) = 2
Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Total Fares Picked	Pass/Fails
3	£70	4.238531 MB	NO	11	PASS
5	£60	4.595662 MB	NO	9	PASS
7	£20	4.476988 MB	NO	5	PASS
9	£40	4.402963 MB	NO	7	PASS
11	£10	4.265637 MB	NO	4	PASS
13	£20	4.238531 MB	NO	6	PASS

TEST 1
Number of Taxis = 4
Number of Obstacles (Parked Cars) = 4
Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Total Fares Picked	Pass/Fails
3	£40	4.115442 MB		8	PASS
5	£70	0.355289 MB	NO	11	PASS
7	£50	3.697059 MB	NO	9	PASS
9	£0	3.751075 MB	YES	2	FAIL
11	£0	4.006541 MB	NO	4	PASS
13	£0	4.098517 MB	NO	4	PASS

TEST 1
Number of Taxis = 4
Number of Obstacles (Parked Cars) = 6
Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Total Fares Picked	Pass/Fails
3	£60	0.321143 MB	NO	10	PASS
5	£20	0.322778 MB	NO	6	PASS
7	£40	3.544892 MB	NO	8	PASS
9	£40	3.681626 MB	NO	8	PASS
11	£40	3.699859 MB	NO	8	PASS
13	£0	3.912158 MB	NO	4	PASS

Actual Outcome:

- **Test 1 (4 Taxis, 2 Obstacles, 20 Fares):** The taxis avoided collisions with the 2 parked cars across all tested steps. Total revenue and the number of fares picked remained generally stable, though some variation occurred as N Steps increased. Memory usage peaked at 4.55 MB. All scenarios passed, demonstrating effective obstacle avoidance with 2 obstacles

- **Test 2 (4 Taxis, 4 Obstacles, 20 Fares):** The simulation avoided collisions for most steps. However, a collision was detected at N Step 9, resulting in a decline in both total revenue and the number of fares picked. Memory usage reached a peak of 4.10 MB. The system performed well in most scenarios but revealed a potential limitation in managing increased obstacles, as indicated by the detected collision.

- **Test 3 (4 Taxis, 6 Obstacles, 20 Fares):** The simulation effectively avoided collisions in most scenarios; however, fare completion performance varied, especially as N Steps increased. Memory usage peaked at 4.63 MB. While the system successfully maintained obstacle avoidance, efficiency declined, with one collision detected at N Step 9.

The experiment shows that the obstacle avoidance mechanism is generally effective, successfully preventing collisions with parked cars even as the number of obstacles increases. However, fare completion efficiency declines as the environment becomes more complex, indicating the need for further optimization to maintain a balance between safety and performance in more challenging scenarios

6.1.3 Experiment 3: Performance of simulation in Collision avoidance and Obstacle avoidance with varying N Steps (BMC) and Taxis.

Objective

The objective of this experiment is to evaluate the performance of the simulation under varying number of taxis specifically testing the system's ability to handle memory usage, collision avoidance and fares within a fixed time limit. The aim is to observe how the number of taxis affects the number of completed fares, total revenue, memory usage, and overall system safety by detecting potential collisions. The experiment aims to determine the optimal balance between the number of taxis and system's performance while ensuring safety and efficiency.

Procedure

Setup

1. **Simulation Environment:** The simulation is set up with a fixed of number of parked cars (2) on the grid and a constant number of fares (10).
2. **Variable:** The number of active taxis is varied across three test cases, i.e., 4, 6, and 8.
3. **N Steps:** The simulation runs for multiple steps, with N Steps being incremented in each trial (ranging from 2 to 11).
4. **Fixed Time Limit:** The simulation runs for a fixed duration, ensuring consistency across all test cases.

Method

1. **Run the Simulation:** For each configuration of taxis (4, 6, and 8), the simulation is executed while collecting data on total revenue, peak memory usage, the number of completed fares, and whether any collisions are detected.
2. **Data Collection:** Record the key metrics at each step, which include:
    - Total Revenue
    - Peak Memory Usage
    - Number of Fares Completed
    - Collisions Detected
    - Pass/Fail Status for each step based on whether the simulation meets safety and performance criteria.
3. **Collision Detection**: Use the system's built-in collision detection mechanism to monitor for any violations, marking the trial as failure if collisions occur.

Evaluation and Analysis

1. **Performance Evaluation:** Analyse how the total revenue, memory usage, and number of completed fares vary with the increase in the number of taxis.
2. **Safety Assessment**: Evaluate the safety of the system by tracking the occurrence of any collisions. A "Pass" is marked if no collisions are detected within the simulation period, and a "Fail" if collisions are present.
3. **Memory Usage Analysis:** Asses the peak memory usage across different taxi configurations to determine if the system remains within acceptable limits as the number of taxis increases.
4. **Effectiveness of Fare Completion:** Compare the number of completed fares to evaluate how effectively the system manages multiple taxis.

Expected Outcome:

1. **Scalability:** The simulation should scale effectively as the number of taxis increases with a corresponding increase in total revenue and completed fares, without a significant rise in memory usage or collisions.
2. **Safety:** Ideally, no collisions should be detected across all tests, ensuring the system's collision avoidance mechanism is robust.
3. **Performance:** The simulation is expected to show optimal performance at a balanced number of taxis, beyond which efficiency might plateau or decline due to increased complexity.

TEST 1
No of Taxis = 4
No of Parked cars = 2
No of Fares = 10

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Number of fares completed	Pass/Fails
2	£70	2.301263 MB	No	7	PASS
3	£40	1.440784 MB	No	4	PASS
5	£50	0.305298 MB	No	5	PASS
7	£20	2.413872 MB	No	2	PASS
9	£0	2.413872 MB	No	0	PASS
11	£40	2.419554 MB	No	4	PASS

TEST 2

No of Taxis = 6
No of Parked cars = 2
No of Fares = 10

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Number of fares completed	Pass/Fails
2	£60	3.831364 MB	0	6	PASS
3	£50	3.696824 MB	0	5	PASS
5	£30	0.330452 MB	0	3	PASS
7	£0	2.714135 MB	0	0	PASS
9	£40	2.602658 MB	0	4	PASS
11	£0	2.49909 MB	0	0	PASS

TEST 3
No of Taxis = 8
No of Parked cars = 2
No of Fares = 10

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Number of fares completed	Pass/Fails
2	£100	3.86936 MB	NO	10	PASS
3	£70	0.329494 MB	NO	7	PASS
5	£40	0.33495 MB	NO	4	PASS
7	£0	0.276179 MB	YES	0	FAIL
9	£0	0.344559 MB	NO	0	PASS
11	£10	3.968328 MB	NO	1	PASS

Actual Outcome:

- **Test 1 (4 Taxis):** The system performed well across all steps with no collisions detected, maintaining stable memory usage, and completing a varying number of fares. The system passed all trials, indicating that the system handles 4 taxis efficiently with no significant issues.
- **Test 2 (6 Taxis):** Similarly, the system-maintained safety with no collisions detected. However, a minor reduction in fare completion was observed as N Steps increased. Memory usage didn't change much either remaining within acceptable limits. The system passed all trials showing good performance with 6 taxis.

- **Test 3 (8 taxis):** As the number of taxis increased to 8, the system portrayed strain signs. With the detection of collisions in the later steps, it is an indication that under higher taxi density, the collision detection mechanism is less effective. The system failed in trials with higher N steps with a slightly higher memory usage reflecting a decrease in safety and performance as the complexity increased.

The experiment demonstrates that the system performs optimally with 4 to 6 taxis, maintaining safety and efficiency. However, with 8 taxis, the simulation struggles to avoid collisions, suggesting a limit to the system's scalability under the current configuration. Future improvements could focus on enhancing the collision avoidance mechanism and optimising memory usage for higher taxi densities.

6.1.4 EXPERIMENT 4: Performance Comparison of BFS, DFS and A* Pathfinding Algorithms

Objective:

The objective of this experiment is to evaluate and compare the performance of three pathfinding algorithms: Breadth-First Search (BFS), Depth-First Search (DFS) and A* in a simulated environment. The effectiveness of each algorithm is tested in this experiment regarding handling fare pickups, avoiding collisions between taxis and avoiding obstacles, in addition to ensuring overall system efficiency. The aim is to identify the weaknesses and strengths of these algorithms under different N steps (3, 5, 10), with a consistent step up of 4 taxis, 20 fares, and 2 static obstacles.

Procedure

Setup:

1. **Simulation Environment:** 4 taxis navigate the grid to avoid collisions and obstacles and generate revenue. 2 static obstacles are included in each test case and 20 randomly distributed fares.

2. **Variable:** The algorithms used: BFS, DFS, A* with 3, 5, and 10 as the varying N steps.

3. **Collision Detection:** Ensure taxis avoid collision between themselves and taxis avoid obstacles (parked cars), while checking how well the different algorithms handle obstacle and collision avoidance.

4. **Fixed Time Limit:** The simulation runs for a fixed duration, ensuring consistency across all test cases.

Method:

1. **Run the Simulation:** The simulation is run for 3 (3, 5, 10) different N steps for each algorithm (BFS, DFS, A*) while collecting data on the total revenue, memory usage, number of fares picked, and collisions detected (YES/NO).

2. **Data Collection:** Just like the experiments before, we record the:

    - Total Revenue
    - Peak memory usage
    - Collisions Detected
    - Total Fares picked
    - Pass/Fail Status

3. **Algorithm Performance Comparison:** For each algorithm, compare how it performs based on the total revenue, memory usage, and fares collected.

Evaluation and Analysis

1. **Algorithm Effectiveness:** Compare the performance between the algorithms based on total revenue, fare completion, and memory usage. The number of completed fares is determined by total revenue divided by 10.

2. **Performance Metrics:** Observe how different N steps affect each algorithm's performance in terms of total revenue, memory usage and picked fares.

3. **Scalability:** Assess each algorithms ability to handle larger N steps (complexity), mostly regarding memory usage and efficiency.

Expected Outcome:

- BFS is expected to yield good results in terms of fare completion/ total revenue but may suffer higher memory usage and slower performance due to its exhaustive nature.

- DFS is expected to complete the least fares mostly as N steps increase due to its depth-first exploration, which can cause suboptimal paths.

- The best balance of computational efficiency and performance is expected to be provided by A*, given its heuristic nature. It should maintain reasonable memory usage while maximizing total revenue and memory usage.

TEST 1
Algorithm = BFS
Number of Taxis = 4
Number of Obstacles (Parked Cars) = 2
Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Total Fares Picked	Pass/Fails
3	£50	4.030503 MB	NO	9	PASS
5	£40	3.933458 MB	NO	8	PASS
10	£20	4.030503 MB	NO	6	PASS

TEST 2
Algorithm = DFS
Number of Taxis = 4
Number of Obstacles (Parked Cars) = 2
Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Total Fares Picked	Pass/Fails
3	£100	4.625344 MB	NO	14	PASS
5	£70	3.660469 MB	NO	11	PASS
10	£0	3.994983 MB	NO	2	PASS

TEST 3
Algorithm = A*
Number of Taxis = 4
Number of Obstacles (Parked Cars) = 2

Number of Fares = 20

N Steps	Total Revenue	Peak Memory Usage	Collisions Detected	Total Fares Picked	Pass/Fails
3	£80	5.008951 MB	NO	12	PASS
5	£20	5.190764 MB	NO	6	PASS
10	£10	5.303968 MB	NO	5	PASS

Actual Outcome:

- **Test 1(BFS):** It avoided collisions successfully across all N steps with a steady decline in the total revenue as N become larger. It also maintained consistent memory usage with 4.03 MB as the peak suggesting that BFS is effective at avoiding collisions but may not be the most efficient in terms of revenue generation and fare completion as the simulation progresses. All scenarios passed without collisions indicating that BFS is reliable for obstacle avoidance in smaller grid sizes and fewer steps. Note: efficiency in fare completion decreases as N becomes larger.

- **Test 2 (DFS):** Performed well in avoiding collisions as no collisions were recorded throughout the tests. It demonstrated strong performance in fares picked and total revenue when N is relatively small, but with a drastic drop in fares picked and total revenue as N grow larger. Its memory peaked at 4.63 MB but decreased as N grow bigger. This indicates that while DFS is effective in smaller, less complex scenarios, its performance diminishes as the grid complexity increases.

- **TEST 3 (A*):** Also effectively avoided collisions across all tested scenarios and performed well in fare collection (picked fares) consistently, mostly when N is smaller, but a significant decrease of picked fares is noticed as N grows bigger. A* experienced the highest memory usage among the 3 algorithms peaking at 5.03 MB, a reflection of its heuristic approach that inserts a balance between computational resources and pathfinding accuracy. It exempted strong obstacle and collision capabilities passing all tests. However, in similar fashion like other algorithms, it showed a decrease in fare picking as N got bigger.

This experiment highlights the strengths and weaknesses of all 3 algorithms in the context of fare completion and collision avoidance. BFS and A* were superior to DFS which showed a drastic decline in the number of fares picked as N become bigger. All algorithms showed limitations in fare completion efficiency mostly as the environment become more complex. A* stands out for its balance between safety and performance, achieving high fare completions in the early steps. As a drawback, as the complexity of the simulation grows, All algorithms experience a decline in efficiency, suggesting that further optimization is required to enhance performance in more challenging scenarios.

6.1.5 Experiment 5: Test the traffic regulation constraint and Scalability.

Objective

It introduces the final constraint for this project which is traffic regulations. Taxis must maintain a specified space between themselves. This experiment is different from the previous ones. Its focus is to purposely keep increasing complexity until no solution can be found. Therefore, the objective is to discover the scalability of the system.

Procedure

Setup: Choose any random variables and keep increasing the value whenever you get a "PASS" result meaning no collision whatsoever.

Method

Run the Simulation: Start small and grow bigger evry time the system passes the test, i.e., no collisions detected.

Data collection: State the N steps, number of taxis, number of obstacles and the gap between taxis.

Expected Outcome:

- The system is expected to fail the "PASS" test as system grows bigger (More complex). Exposing its level of scalability.

N steps	No of taxis	No of obstacles	Taxi space	PASS/FAIL
2	4	2	1	PASS
5	6	4	3	PASS
10	8	6	5	FAIL

Actual Outcome:

The system's limit was finally met. An indication for the level of its scalability.

This experiment was mainly focused on finding the systems limit and the chosen variables where able to achieve that. This doesn't prove weakness of the system but rather shade more light on the need for a wider service area, speed limit recommendations, and the use of more sophisticated mechanisms.

CONCLUSION

The integration of Satisfiability Modulo Theories (SMT) and Bounded Model Checking (BMC) into AI planning systems has portrayed significant potential in improving safety, reliability and efficiency of autonomous operations, particularly within the premises of a simulated urban taxi system. This project has effectively highlighted how formal verification techniques can be incorporated into complex real-time environments, where traffic regulations, static obstacles and scalability are critical factors. The project has also addressed the challenges of route optimisation, traffic regulation compliance and collision avoidance through the implementation of BMC and Z3 solver. As amplified by (www.roundtrip.ai, n.d.), these range from complexity, cost of implementation, dependency and unpredictable environments, and dynamic traffic among others.

Through experiments, the project has provided empirical evidence of the system's ability to adapt to varying environmental complexities and operational loads. The application of SMT ensured the handling of complex constraints, guaranteeing that the autonomous taxis managed to

stay within defined safety parameters. BMC was essential in verifying the accuracy of the system over bounded time frames, allowing for the prior detection of potential violations and the enactment of corrective measures. The results obtained from the multiple experiments underscore the effectiveness of the collision and obstacle avoidance mechanisms, mostly when the taxi fleet size remains within optimal range. However, the system's performance portrayed signs of strain as the system scaled up or when the complexity was intensified evidenced by a decline in fare collection and completion, and occurrence of collisions. This is a clear indication that further refinement of the formal verification processes and planning algorithms is required to ensure better handling of larger taxi fleets, and higher levels of complexity.

It is important to note that SMT is without its challenges as outlined by (Nieuwenhuis et al, 2007) and (Leonardo de Moura Microsoft Research, n.d.) who lists: annotation burden, machine arithmetic, and robustness among others as some of the SMT challenges. However, the findings from this project add valuable insights to the wide field of AI planning and urban transportation systems. They amplify the necessity of selecting appropriate verification techniques and planning techniques which align with environmental complexity and fleet size. The project demonstrates the practical application of formal verification in real-time and dynamic environments, creating a way for future development of more sophisticated and efficient autonomous systems.

In conclusion, while the project's implementation has proven to be fairly robust and effective within specified operational limits, future research and development should focus on enhancing the scalability and robustness of the system, mostly in handling larger taxi fleets and more complex (realistic) urban scenarios. This could include exploring more sophisticated SMT solvers, improving BMC's efficiency, and infusion of machine learning techniques to further optimize the planning and verification processes. The integration of these advanced methods shows significant potential for future AI-powered transportation and automation vehicle systems.

REFRENCE LIST

1. Barbosa, H., Barret, C., Brain, M., et al., (2022). cvc5: A Versatile and Industrial-Strength SMT Solver.Cvc5 https://www-cs.stanford.edu/~preiner/publications/2022/BarbosaBBKLMMMN-TACAS22.pdf

2. Barret, C. (2008). SMT Solvers: Theory and Practice. [online] Available at: https://resources.mpi-inf.mpg.de/departments/rg1/conferences/vtsa08/slides/barret2_smt.pdf

3. Barret, C. and Tinelli, C. (2018). Satisfiability Modulo Theories. [online] Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fb306c0dd0edb31233c7406564d5a916883d81bc

4. Barrett, C., Tinelli, C. (2018). Satisfiability Modulo Theories. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds) Handbook of Model Checking. Springer, Cham. https://doi.org/10.1007/978-3-319-10575-8_11

5. Kanig, J. (2010). *An Introduction to SMT Solvers*. [online] Available at: https://www.open-do.org/wp-content/uploads/2010/06/SMT_provers.pdf [Accessed 5 Aug. 2024]

6. klu.ai. (2023). *What is AI Planning (Automated Planning & Scheduling)? — Klu*. [online] Available at: https://klu.ai/glossary/automated-planning-and-scheduling.

7. www.javatpoint.com. (n.d.). *Forward Chaining and backward chaining in AI - Javatpoint*. [online] Available at: https://www.javatpoint.com/forward-chaining-and-backward-chaining-in-ai.

8. Singh, R. (2024). *Hierarchical Planning in AI*. [online] Available at: https://www.naukri.com/code360/library/hierarchical-planning-in-ai [Accessed 17 Aug. 2024].

9. GeeksforGeeks (2024). *Classical Planning in AI*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/classical-planning-in-ai/ [Accessed 5 Sep. 2024].

10. Wintersteiger, N. B., Leonardo de Moura, Lev Nachmanson, and Christop (n.d.). Programming Z3. [online] theory.stanford.edu. Available at: https://theory.stanford.edu/~nikolaj/programmingz3.html

11. Tau.ac.il. (2024). Z3Py Guide. [online] Available at: https://www.cs.tau.ac.il/~msagiv/courses/asv/z3py/guide-examples.htm#:~:text=Z3%20supports%20Boolean%20operators%3A%20And,simple%20set%20of%20Boolean%20constraints.&text=The%20Python%20Boolean%20constants%20True,to%20build%20Z3%20Boolean%20expressions. [Accessed 3 Sep. 2024]

12. Nareyek, A., Freuder, E.C., Fourer, R., Giunchiglia, E., Goldman, R.P., Kautz, H., Rintanem, J. and Tate, A. (2005). Constraints and AI Planning. IEEE Intelligent Systems, 20(2), pp.62-72. Available at:

https://www.aiai.ed.ac.uk/project/ix/documents/2005/2005-ieee-is-nareyek-constraints-print.pdf

13. Rintanen, J. (n.d.). *SAT in Artificial Intelligence Introduction SAT NP-completeness Phase transitions Resolution Unit Propagation Davis-Putnam CDCL Restarts SAT application: reachability MAXSAT Algorithms Applications Application: MPE Application: Structure Learning #SAT Algorithms Application: Probabilistic Inference SSAT Algorithms SSAT applications SMT Algorithms Application: Timed Systems Conclusion References 1 / 104 SAT in AI: high performance search methods with applications*. [online] Available at: https://users.aalto.fi/~rintanj1/jussi/aaai14tutorialSAT/tutohandout-4up.pdf

14. Clarke, E., Biere, A., Raimi, R. *et al.* (2001). Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design* **19**, 7–34. https://doi.org/10.1023/A:1011276507260

15. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R. (2013). The MathSAT5 SMT Solver. In: Piterman, N., Smolka, S.A. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2013. Lecture Notes in Computer Science, vol 7795. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-36742-7_7

16. www.roundtrip.ai. (n.d.). *AI Route Optimization: Balancing Efficiency and Challenges • Roundtrip*. [online] Available at: https://www.roundtrip.ai/articles/ai-route-optimization-balancing-efficiency-and-challenges.

17. Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A. (2007). Challenges in Satisfiability Modulo Theories. In: Baader, F. (eds) Term Rewriting and Applications. RTA 2007. Lecture Notes in Computer Science, vol 4533. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-73449-9_2

18. Beyer, D., Dangl, M. (2016). SMT-based Software Model Checking: An Experimental Comparison of Four Algorithms. In: Blazy, S., Chechik, M. (eds) Verified Software. Theories, Tools, and Experiments. VSTTE 2016. Lecture Notes in Computer Science(), vol 9971. Springer, Cham. https://doi.org/10.1007/978-3-319-48869-1_14

19. Dutertre, B. (2014). Yices 2.2. In: Biere, A., Bloem, R. (eds) Computer Aided Verification. CAV 2014. Lecture Notes in Computer Science, vol 8559. Springer, Cham. https://doi.org/10.1007/978-3-319-08867-9_49

20. Jiménez, S., Rosa, T.D.L., Fernández, S., Fernández, F. and Borrajo, D. (2012). A review of machine learning for automated planning. *The Knowledge Engineering Review*, [online] 27(4), pp.433–467. doi: https://doi.org/10.1017/S026988891200001X.

21. De Moura, L. and Bjørner, N. (2011). Satisfiability modulo theories. *Communications of the ACM*, 54(9), pp.69–77. doi: https://doi.org/10.1145/1995376.1995394.

22. Hořeňovský, M. (2018). *Modern SAT solvers: fast, neat and underused (part 1 of N)*. [online] The Coding Nest. Available at: https://codingnest.com/modern-sat-solvers-fast-neat-underused-part-1-of-n/ [Accessed 17 Aug. 2024].

23. www.thetaxicentre.com. (n.d.). *The Future of Autonomous Driving and Taxis | The Taxi Centre*. [online] Available at: https://www.thetaxicentre.com/news/the-future-of-autonomous-driving-and-taxis/.

24. Lutkevich, B. (2019). *What are Self-Driving Cars and How Do They Work?* [online] TechTarget. Available at: https://www.techtarget.com/searchenterpriseai/definition/driverless-car.

25. Partial-Order Planning. (n.d.). Available at: https://www.cs.utexas.edu/~mooney/cs343/slides/pop.pdf [Accessed 18 Sep. 2024].