Agent Interval Temporal Logic

Agent Interval Temporal Logic

by

Johannes Francois Oberholzer

A dissertation submitted in fulfilment of the requirements
for the degree

Masters in Philosophy

in the Department of Philosophy at the

UNIVERSITY OF PRETORIA

FACULTY OF HUMANITIES

SUPERVISOR: Prof Emma Ruttkamp-Bloem
CO-SUPERVISOR: Prof Stefan Gruner

January 2020

Acknowledgements

I would like to thank my supervisor Professor Emma Ruttkamp-Bloem of the Philosophy Department at the University of Pretoria. Without her endless patience and expert feedback, this work would not have been possible. Her endurance is legendary, and her caring attitude makes her a beloved mentor.

I would like to thank my co-supervisor Professor Stefan Gruner of the Computer Science Department at the University of Pretoria. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it. His unique insights were valuable.

I would like to thank Professor Valentin Goranko of the Philosophy Department at Stockholm University, who provided crucial advice at various stages of writing this dissertation. I am also grateful for his assistance in helping me attend the Nordic Logic Summer School 2017 in Stockholm, where I gained many valuable perspectives.

I would like to thank Dr. Nils Timm of the Computer Science Department at the University of Pretoria. His assistance with revising multiple key chapters is greatly appreciated.

I would like to thank Ken Halland of the School of Computing at the University of South Africa, as an additional reader of this dissertation. I am gratefully indebted to him for his very valuable comments on the entire dissertation in its final stages, comments without which this work might never have gotten done.

I would like to thank the CSIR and the Centre for Artificial Intelligence Research for funding this dissertation.

Finally, I would like to thank my parents, family and friends, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this dissertation. This accomplishment would not have been possible without them.

Thank you.
Francois

Abstract

Alternating-Time Temporal Logic (ATL), introduced by Alur, Henzinger and Kupferman, is a logic involving coalitions of agents performing actions which cause a state change in a turn-based time system. There have been game theoretic extensions on ATL, and they are very good at specifying systems of multiple agents cooperating or competing in a game-like situation. Unfortunately neither ATL nor its extensions are able to capture the idea of gradual change, or duration of actions or events. The concurrent game model of ATL operates like a turn based game, with sets of agents taking their turn, and then the environment changing based on their actions, before they take their next turn. The fact that some actions take longer than others, or that sometimes a state changes gradually, rather than immediately, is not representable in ATL. As an example, take a train entering a tunnel. Before the train enters the tunnel, it is outside the tunnel, after it has entered the tunnel, it is inside the tunnel, but for the few seconds it takes the train to enter the tunnel, it is neither inside nor outside the tunnel. ATL cannot represent this basic intuitive truth.

A family of logics called Interval Logic (IL) use finite state sequences called "intervals", which allow it to describe a more continuous model of time, rather than a discrete state based one such as ATL. This allows it to capture the idea of gradual change, of a train entering a tunnel, and the fact that actions and events have various durations. Most of the IL formulations do however not have any way of distinguishing multiple agents acting at the same time.

Both of these logics - ATL and IL - are useful for specific things, but combining them might produce new applications which are not possible when only using the one or the other. In this dissertation we present one such possible combination, called Agent Interval Temporal Logic (AITL). AITL combines the notion of agents, coalitions and strategies from ATL with the interval based model of time from IL, thus creating a new logic which might have some powerful applications in a wide range of areas in which gradual change and multiple agents acting at the same time can both be accommodated.

Contents

Chapter 1

Introduction

This dissertation aims to contribute to the field of formal logic and knowledge representation and reasoning by presenting a logic for reasoning about the strategic abilities of agents. Specifically we consider agents in a multi-agent system while incorporating aspects of interval time models. This new logic is an extension of *Alternating-Time Temporal Logic* (ATL) and will be called *Agent Interval Temporal Logic* (AITL).

Before any agent can act, it must decide on what action to take. Before it can make a decision on the best action to take, it must consider the possible outcomes of the action, and try to predict what effect the action will have. While in ordinary circumstances humans do not necessarily consider the process by which they make decisions, in the context of Artificial Intelligence for instance, it becomes necessary to have the ability to formally represent the decision making process for the purposes of programming. One of the functions of logic is to enable us to represent knowledge of a system and predict the outcomes of various events in that system. This will be our main focus in this dissertation: to develop a formal mechanism that allows correct prediction of some future state based on the current state and events which occur to change it.

McCarthy and Hayes (1981) state that: "A computer program capable of acting intelligently in the world must have a general representation of the world in terms of which its inputs are interpreted... More specifically, we want a computer program that decides what to do by inferring in a formal language that a certain strategy will achieve its assigned goal. This requires formalising concepts of causality, ability, and knowledge". While the field of AI has since moved from knowledge-based to data-driven platforms, the need for formal mechanisms to accurately represent the essence of these concepts remains.

A very important aspect of acting in the world is time. Shoham (1987) states that: "It is hard to think of a research area within Artificial Intelligence (AI) that does not involve reasoning about time in one way or another". Temporal

logic (the logic of time) has been very successful at allowing reasoning about time and has had "numerous important applications not only to philosophy, but also to computer science, artificial intelligence, and linguistics" (Goranko and Galton, 2015).

Alternating-Time Temporal Logic (ATL), introduced by Alur et al. (2002), is a very popular temporal logic for reasoning about the strategic abilities of agents Goranko et al. (2018), and will be fundamental to this dissertation. ATL is however restricted to a turn-based time model, where the agents and the system are in a back and forth game with each other. This discrete time model makes it difficult to represent various real time problems in ATL. Any change that takes place does so over a period of time, and reasoning about what effect a certain behaviour might have also means reasoning about the time during which that effect takes place. We will therefore extend ATL by modifying its time model to allow for overlapping intervals of time.

This dissertation is structured as follows: chapter 2 will give a brief overview of the main achievements in the historical development of logic from propositional to temporal logic, and will introduce various concepts and characteristics of temporal logic which will be used in later chapters. Chapter 3 will introduce three illustrating problems with reasoning about cases relating to time intervals and multiple actions: the *Banker Algorithm* (Dijkstra, 1982) and *Sleeping Barber* (Dijkstra, 1968), as well as a unique train problem created for this dissertation. These problems will be considered repeatedly throughout the dissertation, as different logics will be used to represent them. This will allow us to see the strengths and limitations of various logics. Chapter 4 will introduce *Alternating-Time Temporal Logic*, its syntax and semantics, related logics and extensions. Chapter 5 will consider various logics which use intervals. It will discuss some concepts which are important to intervals and compare different interval logics. There will be a strong focus on Allen and Ferguson's logic of actions and events, since it is one of the very few logics which use pure intervals (Allen and Ferguson, 1994). Chapter 6 will introduce the new *Agent Interval Temporal Logic* (AITL) which is the main contribution of this dissertation, followed by a conclusion in chapter 7.

Chapter 2

Background

All formal languages include two kinds of notation. Firstly there is a set of symbols, and rules for combining the symbols in various ways, this is called the *syntax*. Secondly, there is a system of some sort allowing us to interpret a combination of symbols (called a *formula*) and determine if this formula is true or false, this is called the *semantics*. The simplest example we can define would be as follows:

The syntax of a formula, represented by φ, is defined by:

$$\varphi ::= p \mid \varphi_1 \vee \varphi_2$$

where p is an atomic proposition.

Atomic propositions are the smallest building blocks of a logical language. They cannot be further broken up and are merely true or false. This syntax provides us with a recursive rule for building up a well formed formula. A formula is either an atomic proposition, or it is two formulas tied together by this symbol $\vee$. We might build a formula $p_1 \vee p_2$, which is well formed, since we used the first rule two times and the second rule once. We cannot build the formula $p_1 p_2$ since nowhere do we have a rule which allows us to place two atomic propositions next to each other. This is only syntax, we don't yet know what p_1 or p_2 or $\vee$ means, they are merely symbols. To give meaning, we must introduce the semantics. The semantics involves a mapping function or evaluation, which for any given well-formed formula can tell us if that formula is true or false. For this example we will just give an informal intuitive semantics [1].

[1] This is of course an over simplification for the sake of a clear explanation. The semantics involves an interpretation of the language at issue, which is a set (D, f) where D is a nonempty set called the domain of interpretation (or universe or domain of discourse) and f is a denotation

We know that an atomic proposition by itself can be either true or false, we now want to know if a larger well formed formula would be true. We will do this with one simple rule:

The formula $\varphi_1 \vee \varphi_2$ is true if either φ_1 is true or φ_2 is true.

Thus the formula $p_1 \vee p_2$ is true when either p_1 or p_2 is true. Atomic propositions represent some value in the world. We might say that p_1 means the cat is brown, and p_2 means it is raining. The formula $p_1 \vee p_2$ then means "The cat is brown or it is raining", which would be true if either of the two disjuncts are true. Later on we will see formal definitions of semantics for much more complicated models.

Building from here, this chapter will cover prerequisite concepts which will be required for later chapters in this dissertation. We start with a brief overview of the history of the development of logic from Aristotle to modern temporal logic, stopping along the way to consider various formulations and concepts. We then move on to discuss some properties of temporal logic, which can be used to distinguish various formulations. Finally we end this chapter with a brief look at some philosophical problems which haunt our efforts in logic to reliably predict outcomes.

2.1 Historical Development

2.1.1 Classical Logic

Formalised logic can be traced back to classical Greece. The Greeks were very interested in correct reasoning, and it was Aristotle who first formalised a way in which correct and incorrect inferences can be made. In so doing, he created a "systematic treatment of the principles of correct reasoning, the first logic" (Shields, 2016).

A very important contribution was his two principles, the *law of excluded middle*, that every statement is either true or false, and the *law of contradiction*, that no statement is both true and false (Kent, 2019).

A key part of Aristotle's logic is that of the *perfect deduction* or *syllogism* (Shields, 2016). A syllogism is a type of argument, in which a conclusion is derived from two assumed premises. The structure of a syllogism guarantees that if the premises are true, the conclusion will be true. The classic three line syllogism example is as follows:

function. f assigns to each element in the vocabulary of the language a member of D or a subset of D^n

All men are mortal.
Socrates is a man.
Therefore, Socrates is mortal.

It was the stoic philosophers who looked more closely at operators like *and*, *or* and *if...then....* They studied sentences and rules for deduction to build sentences, and though a lot of their work was lost, some of it survived. A famous stoic was Chrysippus, who created a list of rules for valid inferences (Kent, 2019).

- If the first, then the second; but the first; therefore the second.
- If the first, then the second; but not the second; therefore, not the first.
- Not both the first and the second; but the first; therefore, not the second.
- Either the first or the second [and not both]; but the first; therefore, not the second.
- Either the first or the second; but not the second; therefore the first.

During the middle ages many gradual improvements were made, but it was only in the 19th century with the work of DeMorgan and Boole in *symbolic logic* that propositional logic became fully formed. Boole provided an algebra to replace Aristotle's syllogistic form. Finally in 1879 Frege created the first modern axiomatic calculus for logic in his work *Begriffsschrift* (Kent, 2019). From here we get modern propositional and predicate (or first order) logic.

Propositional logic consists of a set of atomic propositions and a set of logical operators ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$). These operators are, in order, negation (not), conjunction (and), disjunction (or), implication (if...then) and equivalence (if and only if). Formulæ (syntax) are then built from these similar to how we saw it done at the start of this chapter. Propositional logic also contains rules for determining the truth value of a formula based on the truth values of the atomic propositions and the operators used(semantics), similar to the $\vee$ rule we saw in our earlier simple example.

First order logic has both terms and formulæ. Terms can be variables, often denoted by alphabet letters like x or y, or terms can also contain function symbols together with variables as arguments. Formulæ can be built recursively as follows:

- Predicate symbols. If P is an n-ary predicate symbol and t_1,...,t_n are terms in the argument of the predicate symbol, then $P(t_1, ..., t_n)$ is a formula.
- Operators. The logical operators $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$ can be used as before on formulæ to build additional formulæ.

- Equality. If the equality symbol is considered part of logic, and t_1 and t_2 are terms, then $t_1 = t_2$ is a formula.
- Quantifiers $\exists$ and $\forall$. If φ is a formula, and x is a variable, then $\exists x\varphi$ (there exists an x such that φ) and $\forall x\varphi$ (for all x, it is the case that φ) are formulæ.

Using these, first order formulæ can be built up. The truth of a formula focuses on a domain of variables, and a set of n-ary relationships on the domain (see footnote 1). For each relationship there is an n-ary predicate function that for each variable in the domain returns true if it is in that relationship. For example, $Brown(Cat)$ is true if the cat is indeed brown, while $Brown(Dog)$ is false if the dog is grey. [2] See Bellini et al. (2000).

Together these logics are called the classical logics, and are useful for representing and reasoning about static situations.

2.1.2 Modal Logic

Situations are not static, and things in the real world constantly change. Classical logic can represent the situation where it is raining and the cat is brown, but as soon as it stops raining or we paint the cat, those assertions need to be changed. If we want to predict future states, we must be able to model the changing values of states.

In the 1950's Saul Kripke introduced the idea that the truth of an assertion is dependent on which one of multiple possible worlds that assertion belonged to. As is already the case in classical logic, we have one actual world among many possible worlds, created by different combinations of truth values, and all assertions have their specific true and false values in that world. If we have a cat and a dog, and they can be either black or white, we can have four different worlds. One world in which both are black, one where both are white, and two where one is white and the other black. If we look outside and see the dog and cat playing, we can conclude we are currently in this or that world. Things can change though, and if we take out the paintbrushes we prepare to move to a different world. Truth is no longer a static thing in a system, a formula can evaluate to true in one world and false in the next. For example, the statement "The light is either on or off" is true in all worlds, since it is a tautology, while the statement "It is raining" is true only in those worlds where rain is falling. Kripke presented a formal syntax and semantics to more accurately capture the notion of possible worlds, where the operator $\Box$ denoted *necessarily* (the assertion is true for all worlds) and $\diamond$ denoted *possibly* (the assertion is true in at least one of the worlds) (Kripke, 1963). These operators

[2] In this example we imply that there is a variable, and that the formula is false when it is set to *Dog* and true when it is set to *Cat*.

function in much the same way as the universal and existential quantifiers from first order logic.

This is where modal logic was first formalised, starting with the logic called K (after Kripke). K has three symbols: $\neg$ for *not*, $\rightarrow$ for *if then*, and $\Box$ for *it is necessary that*. Additional operators ($\wedge$, $\vee$, $\leftrightarrow$, and $\Diamond$) can be derived from these. For more on K see Garson (2018).

K adds the following to the principles of propositional logic:

- The Necessitation Rule: If φ is a theorem of K, then so is $\Box\varphi$.
- The Distribution Axiom: $\Box(\varphi_1 \rightarrow \varphi_2) \rightarrow (\Box\varphi_1 \rightarrow \Box\varphi_2)$.

A theorem is a formula which can be proven using only other theorems and axioms, it is thus not dependent on any assumptions aside from axioms. An example of a theorem is a tautology, a tautology is provably true without relying on any assumptions aside from axioms. The necessitation rule thus states that if a formula is a theorem, such as a tautology, it is not only true, but necessarily true.

Another notion Kripke defined in the same paper was what is known as a Kripke structure. Kripke structures are used to define the semantics for various modal logics, since a formula is interpreted over a specific Kripke structure and is true or false locally in a possible world of that structure. Before discussing Kripke structures however, let us discuss the concept of a state. A state is simply a collection of assertions or atomic propositions which are true. A state is essentially a *possible world*, one line of a truth table in propositional logic for instance. The world where the cat is black and the dog is white is a state, and when we paint the dog black we *transition* to a new state (of the real system at issue) where now both are black. This concept of states, and transitions from one state to another, is called a *transition system*. A Kripke structure is a transition system, which is defined as follows:

Let Π be a set of atomic propositions, then a Kripke structure is a tuple $M = (S, I, R, L)$ where:

- S is a set of states.
- I is a set of initial states, $I \subseteq S$.
- R is a binary transitional relation between states.
- L is a labelling function $L : S \rightarrow \Pi$

There are states, transitions between states, and a labelling function mapping atomic propositions to states. There is also a set of initial states, which are the starting states of the system. We start with the initial state as the current state, but once the paintbrushes come out we transition into other states.

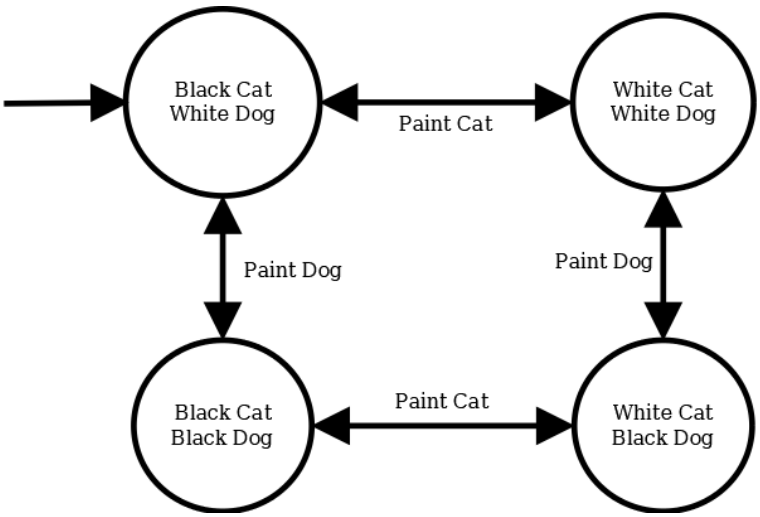


Figure 2.1: Example of a Kripke structure with 4 states

2.1.3 Temporal Logic

Bellini et al. (2000) states that: "Classical logic can express only atemporal (nontime-dependent) formulæ whose validity and satisfiability do not depend on the instant in which they are evaluated. In other words, time has no role in classical logic; when a proposition presents a value that changes over time, the time must be modelled as an explicit variable". One needs a logic that can take time into account if one wants to describe a system which is dynamic (Bulling et al., 2015), since the truth values of assertions will change as time goes on (Alur et al., 2002).

It is only a small jump from considering the truth values of assertions in different *worlds* to considering the truth values of assertions at different *times*. This gives us temporal logic, a sub type of modal logic where the different worlds are all instants on a timeline.

Reasoning about time can already be seen in Aristotle and other ancient thinkers, with examples found in Zeno's classical paradox about infinite time intervals of infinitesimal size, Aristotle claiming that statements about future contingent propositions could not be ascribed any truth values, and Diodorus Cronus distinguishing between possible and necessary future truths (Goranko and Galton, 2015). Modern temporal logic was pioneered by Arthur Prior, who called it *tense logic*. Prior suggested a temporal interpretation of $\Box$ as *always* rather than *necessarily* and $\diamond$ as *sometimes* rather than *possibly* (Prior, 1957). In addition to the classical operators, temporal logic adds four new unary operators (Prior, 1967):

- G - it will always be that (Future always).

- H - it always was that (Past always).
- F - it will be the case that (Future eventually).
- P - it was the case that (Past eventually).

Temporal logic also inherits the two rules from K, along with axioms to govern interaction between past and future (Garson, 2018):

- The Necessitation Rule: If φ is a theorem of the language, then so is $G\varphi$ and $H\varphi$.
- Distribution Axiom: $G(\varphi_1 \rightarrow \varphi_2) \rightarrow (G\varphi_1 \rightarrow G\varphi_2)$ and $H(\varphi_1 \rightarrow \varphi_2) \rightarrow (H\varphi_1 \rightarrow H\varphi_2)$.
- Interaction Axioms: $\varphi \rightarrow GP\varphi$ and $\varphi \rightarrow HF\varphi$.

Another operator quite common in temporal logic is the binary *until* operator, represented with U. A formula $\varphi_1 U \varphi_2$ would be interpreted as φ_1 is true from now until φ_2 becomes true in the future, and φ_2 will become true at some point in the future. Sometimes the operator *since* is also included, which is the opposite of *until*. A formal definition of *until* will be provided in the next section.

It is interesting to note that, as Bellini et al. (2000) points out, the basic four temporal operators can be defined in terms of *until* and *since* as follows:

- $F\varphi \equiv \top$ until φ
- $P\varphi \equiv \top$ since φ
- $G\varphi \equiv \neg F \neg \varphi$
- $H\varphi \equiv \neg P \neg \varphi$

Where $\top$ means *true*.

The last two common temporal operators are the unary operators *next* and *prev*, represented by $\bigcirc$ and $\bullet$ respectively. These operators indicate that the formula is true in the next or previous state of the model. The *Next* operator is also often written as X, which is how we will use it for the rest of this dissertation.

Operators such as *next* and *eventually* and *until* all assume some sort of sequence or path. We can define a *path* as an infinite sequence of states that can result from subsequent transitions in the model, represented as λ. A single state in this path can be referred to by writing $\lambda[3]$ to mean the third state on the path. A *successor state* in a path is any state q' which can be reached from the current state q in a single transition.

Probably the best known and most taught temporal logics are *Linear Time Temporal Logic* (LTL) and *Computational Tree Logic* (CTL). In the next section we will consider both, as well as their merger CTL*.

2.1.4 LTL, CTL, and CTL*

Linear Time Temporal Logic (LTL) was introduced by Pnueli (1977) and is the "most popular and widely used temporal logic in computer science" (Goranko and Galton, 2015). Formulas in LTL are interpreted over a path λ, which consists of an infinite sequence of states of a Kripke structure. It is interesting to note that in the original paper, Pnueli (1977) used a *dynamic discrete system* instead of a Kripke structure, which is essentially the same except without a labelling function.

A well formed formula in LTL is obtained as follows:

$$\varphi ::= p|\neg\varphi|\varphi_1 \vee \varphi_2|X\varphi|\varphi_1 U \varphi_2$$

where p is an atomic proposition $p \in \Pi$, X is the *next* operator and U is the *until* operator.

These formulæ are interpreted over an infinite path λ for a Kripke structure, where the satisfaction relation $\vDash$ indicates that a path satisfies a formula, in other words the formula is true for that path. The semantics for LTL is defined as follows:

- $\lambda \vDash p$, for all propositions $p \in \Pi$, iff $p \in L(\lambda[0])$.
- $\lambda \vDash \neg\varphi$, iff $\lambda \nvDash \varphi$.
- $\lambda \vDash \varphi_1 \vee \varphi_2$, iff $\lambda \vDash \varphi_1$ or $\lambda \vDash \varphi_2$.
- $\lambda \vDash X\varphi$, iff $\lambda[1] \vDash \varphi$, where $\lambda[1]$ is the successor state (next state in the path) to $\lambda[0]$.
- $\lambda \vDash \varphi_1 U \varphi_2$, iff there exists an i such that $\lambda[i] \vDash \varphi_2$ and for all k such that $0 \leq k < i$ we have $\lambda[k] \vDash \varphi_1$.

In order to write neat and concise formulæ some additional operators are needed. These are the operators $\wedge$, $\rightarrow$, $\leftrightarrow$, F and G, which can all be defined in terms of what has already been defined in the discussion of temporal logic in the above section. Specifically:

- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$
- $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
- $\varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$
- $F\varphi \equiv \top U \varphi$
- $G\varphi \equiv \neg F \neg\varphi$

As mentioned, only the essential parts are defined in the syntax and semantics of a logic, and other operators can then be derived and used from these. This will be seen in most of the logics in the rest of this dissertation, such as only negation and disjunction being included in the syntax, from which conjunction, implication and equivalence can be derived.

Not long after the introduction of LTL, a new logic was proposed by Emerson and Clarke (1982) called *Computational Tree Logic* (CTL). While LTL formulæ are interpreted over a path, a sequence of states, a CTL formula is interpreted over a *computational tree*, a set of branching paths representing various future possibilities. Computational trees are described by Goranko and Galton (2015) as "naturally obtained as tree unfoldings of discrete transition systems, hence they naturally represent the tree of all infinite computations arising in such systems". The discrete transition system used is specifically a Kripke structure.

CTL was introduced by Emerson and Clarke (1982), based on synchronisation skeletons of programs. Synchronisation skeletons are finite state abstractions of programs, represented as flow graphs where each node represents a piece of code to be executed. Relevant pieces of code are put together into a single node. For example, a program which performs some sequential steps on its own variables, then accesses a shared resource (critical section), and then performs some more steps on its own variables, can be divided into three code sections, each represented as a node on the synchronisation skeleton. The internal steps of each section is not relevant for synchronisation, only that the program is in a critical section or not, so that it can be guaranteed that two programs do not both access the same shared resource at the same time, if this shared resource can only be accessed by a single process at a time.

Alternating-Time Temporal Logic (ATL), which we will investigate in depth in a later chapter, is a generalisation of CTL (Alur et al., 2002), while ATL* is a multi-agent extension of CTL* (Goranko et al., 2018). CTL is also related to *unified branching time*.
[3]

The syntax for CTL was originally defined as follows (Emerson and Clarke, 1982):

Let p refer to an atomic proposition, and φ_1 and φ_2 refer to sub-formulæ.

- Each of p, $\varphi_1 \wedge \varphi_2$ and $\neg\varphi$ is a formula.
- $EX\varphi$ is a formula which intuitively means that there is an immediate successor state reachable by executing one step of process P in which formula φ holds. (Similar to a Next operator)

[3] *Unified branching time* is very similar, but does not have the U operator, for more refer to Ben-Ari et al. (1983).

- $A[\varphi_1 U \varphi_2]$ is a formula which intuitively means that for *every* computation path, there is some state along the path where φ_2 holds, and φ_1 holds at every state along the path until φ_2. (Similar to a Universal Until operator)
- $E[\varphi_1 U \varphi_2]$ is a formula which intuitively means that for *some* computation path, there is some state along the path where φ_2 holds, and φ_1 holds at every state along the path until ϕ_2. (Like the previous formula, but existential instead of universal)

where E refers to the existential quantifier $\exists$ and A refers to the universal quantifier $\forall$.

The semantics of CTL are defined in (Emerson and Clarke, 1982) over a structure $M = (S, A_1, ..., A_k, L)$ which consists of:

- S - a set of states.
- A_i - $\subseteq S \times S$, a binary relation on S giving the possible transitions by process i.
- L - a labelling function mapping each state to those atomic propositions true in that state.

The states here represent the nodes of the synchronisation skeleton. Let $A = A_1 \cup ... \cup A_k$. A *fullpath* (M) is defined as an infinite sequence of states $(s_0, s_1, s_2, ...)$ such that $\forall i (s_i, s_{i+1}) \in A$. The semantics are defined as follows:

- $s_0 \vDash p$ iff $p \in L(s_0)$.
- $s_0 \vDash \neg\varphi$ iff not $(s_0 \vDash \varphi)$.
- $s_0 \vDash \varphi_1 \wedge \varphi_2$ iff $s_0 \vDash \varphi_1$ and $s_0 \vDash \varphi_2$.
- $s_0 \vDash EX_j\varphi$ iff for some state $t, (s_0, t) \in A_j$ and $t \vDash \varphi$.
- $s_0 \vDash A[\varphi_1 U \varphi_2]$ iff for all M $\exists i[i \geq 0 \wedge s_i \vDash \varphi_2 \wedge \forall j(0 \leq j \wedge j < i \rightarrow s_j \vDash \varphi_1)]$.
- $s_0 \vDash E[\varphi_1 U \varphi_2]$ iff for some M $\exists i[i \geq 0 \wedge s_i \vDash \varphi_2 \wedge \forall j(0 \leq j \wedge j < i \rightarrow s_j \vDash \varphi_1)]$.

Other logical connectives can be derived as before.

CTL*, or full computational tree logic, was introduced by Emerson and Halpern (1985). It is an extension of CTL. Here the distinction between path formulæ and state formulæ is introduced. A state formula is some proposition which is true or false at some time, while a path formula shows the existence of such a state. For the notation, p is a proposition, and X, F and G are state quantifiers, which indicate when the proposition will be true. The universal and existential quantifiers (A and E in the original formulation) are path quantifiers. A formula in CTL is always a pairing of a path quantifier with a single state quantifier. So, if we interpret p to mean "the house is on fire", a formula in CTL might be EXp which means that there exists a path which will cause the house to be on fire during the next state. The fundamental change in CTL* is allowing more than one state quantifier to be paired with a path quantifier, so for example allowing the formula $E(\neg Xp \wedge Fp)$ which means that there exists a path where during the next state the house will not be on fire, but it is going to be on fire at some stage in the future. This is done by introducing state and path formulae into the syntax as Emerson and Halpern (1985) do in the following way:

- Each proposition p is a state formula.
- If φ_1, φ_2 are state formulae, then so are $(\varphi_1 \wedge \varphi_2)$ and $\neg\varphi_1$
- If φ is a state formula, then $F\varphi$ and $X\varphi$ are path formulae.
- If ψ is a path formula then $E\psi$ is a state formula.
- If ψ is a path formula then $A\psi$ is a state formula.
- If φ_1, φ_2 are state formulae then $(\varphi_1 U \varphi_2)$ is a path formula.
- If ψ_1, ψ_2 are path formulae then so are $\psi_1 \wedge \psi_2$ and $\neg\psi_1$.

Other logical connectives can be derived as before. The important change from CTL to CTL* is the distinction between path and state formulæ which allows us to pair more than one state quantifier with a single path quantifier. Or according to Goranko and Galton (2015) compared to CTL, CTL* has "no syntactic restrictions on the applications of temporal operators and path quantifiers, and [is] interpreted on the class of computation trees".

ATL/ATL* was introduced as a generalisation of CTL/CTL* to allow for open systems (Alur et al., 2002). One can see CTL/CTL* as ATL/ATL* but with a single agent. The existential path quantifier E is similar to $\ll i \gg$ where i is the single agent, while the universal path quantifier A is similar to $\ll \varnothing \gg$.

To get from CTL to ATL, we need to introduce the idea of agents and strategies, which will be done in the next section.

2.1.5 Agents and Strategies

The classical notion of a strategy is that of "a conditional plan that prescribes what action a given agent (or, a coalition of agents) should take in every possible situation" (Bulling et al., 2015). By this notion, every autonomous agent acting in a situation can have a strategy by which it makes decisions. A strategy is always aimed at some outcome, goal or objective that the agent would like to achieve. These ideas have been studied extensively in the field of game theory, where a situation is usually set up as a game with specific rules between rational agents, all acting to achieve some purpose, sometimes cooperating and sometimes competing. Ross describes game theory as "the study of the ways in which *interacting choices* of *economic agents* produce *outcomes* with respect to the *preferences* (or *utilities*) of those agents, where the outcomes in question might have been intended by none of the agents" (Ross, 2019).

The simplest strategic game is where all agents make a single decision, independently and simultaneously, and then an outcome is determined by the combination of their decisions, like a vote. The agents all make their decisions only once, and then the results play out, without the opportunity to change their decisions later or influence each other for a next round of decisions. The *prisoners' dilemma* is an example of this, where two prisoners are held separately, and offered a deal to testify against the other. If both testify, both get medium sentences, if both remain silent, both get short sentences, but if one testifies and the other not, the first goes free and the other receives a long sentence. The prisoners cannot discuss it, and even if they could and both agree to remain silent, it would be in their best interest to testify if they were able to convince the other to stay silent. What makes this hard is that they both have a single chance, and must decide to testify or not. They cannot later change their mind if they realise the other betrayed them.

A basic strategic game like this, with multiple agents making a single decision at the same time and then facing the consequences, can be formalised in a *strategic game form*. One example of a strategic game form is from Bulling et al. (2015):

A *strategic game form* (SGF) is a tuple $M = (Agt, \{Act_a | a \in Agt\}, Out, out)$ where:

- Agt is a nonempty finite set of agents.
- Act_a is a nonempty finite set of actions which a can perform where $a \in Agt$.
- Out is a nonempty finite set of outcomes.
- out is an outcome function mapping the list of actions (called an action profile) taken by the agents to a specific outcome, defined as $out : \Pi_{a \in Agt} Act_a \rightarrow$

Out.

To turn this into a strategic game, preference orders need to be added on the outcomes for the various agents. This can be done by adding a payoff function $u_a : Out \to \mathbb{R}$ where some real value is assigned to each outcome for each agent a. This might be the case in sports betting, where some monetary payout is assigned to each possible bet after the result has played out, taking the odds into consideration. Another more abstract way of showing preference is by ordering the results for each agent in the form $o \leq_a o'$ iff $u_a(o) \leq u_a(o')$ where $o, o' \in Out$.

Let us look at a simple example game played by a group of 10 people. Each person pays R5 into the pot to play, and chooses either action *Red* or *Blue*. If the majority of people choose *Red*, they lose, and those that chose *Blue* split the pot. Similarly, if the majority choose *Blue*, they lose, and those that chose *Red* split the pot. If all players choose the same colour, the pot is split between all, thus everyone receives back their initial buy in.

This game can be represented as a strategic game form
$M = (\{a_1, a_2, a_3, ..., a_{10}\}, \{x_{Red}, x_{Blue}\}, \{o_i | 1 \leq i \leq 1024\}, out)$
where:

- $\{a_1, a_2, a_3, ..., a_{10}\}$ are the 10 agents playing the game.
- $\{x_{Red}, x_{Blue}\}$ are the two options available to each agent, *Red* or *Blue*.
- $\{o_i | 1 \leq i \leq 1024\}$ are all possible outcomes to this game, a total of 2^{10} unique combinations.
- *out* is the outcome function, defined to map each combination of actions by the agents to a unique outcome according to the above rules. So if the first 3 players chose *Red*, and the rest chose *Blue*, the action profile would be $RRRBBBBBBB$, and this might map to outcome o_{896}.

To finally make this a game, a payoff function needs to be added, defined informally as:

$$u_a(o) = \begin{cases} 0 & \text{if choice is in majority} \\ 50/winners & \text{if choice is in minority} \end{cases}$$

Where $a \in Agt$, $o \in Out$ and winners is the number of agents who chose the minority option.

Now that we know what a strategic game looks like, we can consider the strategy from the perspective of an agent playing the game. A *strategy* is a conditional plan that specifies what to do in each possible situation, written as S_a to refer to

the strategy of agent a. A strategy can be for a single agent or for a coalition or set of agents.

There are two types of strategies, memory-less and memory-based. Memory-less, also known as positional, is represented by $S_a : St \rightarrow Act$ where $a \in Agt$. Memory-based, also known as perfect recall, is represented by $S_a : St^+ \rightarrow Act$ such that $S_a(\langle ..., q \rangle) \in act_a(q)$, and where St^+ is a set of histories, or finite sequences. So the memory-based strategy takes in not just the current state as in memory-less, but the sequence of states leading up to the current state. Memory-less strategies have the agents considering only the current state, and making a move based on that state, as would be appropriate in chess, where there mostly exists a best move (or set of better moves) for any given position, regardless of the order of events leading to that position. Memory-based strategies are used when the past is important to a choice of action, such as when an agent must attempt an action six times and then do something else. There also exist combined options, where agents have bounded memory, see Ågotnes and Walther (2009).

We can talk about single agents, or about a coalition of agents. A coalition is a set of agents working together, this can be defined as:

$$A = \{a_1, a_2, ..., a_r\} \text{ where } A \in Agt$$

A coalition of players may also have a joint strategy $S_A = \{S_{a_1}, S_{a_2}, ..., S_{a_k}\}$. The contribution of agent a to a joint strategy S_A is denoted by $S_A[a]$.

All of these ideas of paths, action profiles, coalitions and strategies will be revisited in the ATL chapter. For now, we move on to consider some characteristics of temporal logics which can be used to distinguish them.

2.2 Characteristics of Temporal Logics

Many different temporal logics exist, the following are some characteristics which distinguish different types.

2.2.1 Propositional or Predicate

Temporal logic can be based on either propositional or first order classical logic. Higher order classical logic is not often used. Propositional temporal logics are less expressive, but their decision procedures have a tractable complexity (Bellini et al., 2000). First order temporal logics are often more expressive but also more complex.

2.2.2 Points or Intervals

An important issue to consider in temporal logic is the question of which entity in the logic is primary, points or intervals. Time can be seen either as a collection of instants, happening one after the other, or as a collection of various intervals, which may overlap each other. Points or instants may also be used as a simplified abstraction for certain systems, an example of which may be chess. In chess, one player moves a piece, then the other moves a piece. The state of the board changes after each move, and each move follows a previous move. A chess game can be seen as a collection of moves. This is the case even though, in reality, one player might take a few seconds longer on a move that their opponent, and a piece might spend a second in the air while being moved from one location to another. These events and durations are irrelevant to someone who is interested only in the moves themselves, and the strategy behind them, for whom only a list of moves is sufficient.

According to Bellini et al. (2000) interval logics: "are more expressive, since they are capable of describing events in time intervals, and a single time instant is represented with a time interval of one. Usually, interval-based logics permit one to write formulæ with a greater level of abstraction, and so are more concise and easy to understand than point-based temporal logics".

Goranko and Galton (2015) state that: "Instant-based models are often not suitable for reasoning about real-world events with duration, which are better modelled if the underlying temporal ontology uses time intervals, rather than instants, as the primitive entities" and he adds that "Interval-based temporal models are ontologically richer than instant-based ones, as there are many more possible relationships between time intervals than between time instants".

We can approach this question from two different perspectives. Firstly, when doing prediction, which is the most useful? When talking about a chess game, a point based logic will obviously be simpler and sufficient, while in a problem involving trains moving at different speeds and arriving at different stations at various points in time, an interval logic might be needed. But there is a second perspective: when trying to model reality, which better captures the true nature of time? This is a deeper philosophical and scientific question, which doesn't yet have clear consensus. Is time made up of infinitesimally small instants culminating into our continuous experience of time? Or can time be more accurately seen as moments of various durations? A related idea is of absolute or relative time. Newton's classical mechanics uses an absolute model of time, independent from all other space or matter. Meanwhile Leibniz proposed a more relational view where time is dependent on events. Modern physics favours a more relative approach, ever since Einstein's theory of relativity (Goranko and Galton, 2015).

Goranko and Galton (2015) further state that: "the two types of temporal

ontologies are closely related and technically reducible to each other: on the one hand, time intervals can be determined by pairs of time instants (beginning–end); on the other hand, a time instant can be construed as a degenerate 'point interval', whose endpoints coincide" (Goranko and Galton, 2015).

Even when we decide on using intervals, there are two further sub types, pure and non-pure. Pure interval logics view intervals as primary objects and formulæ are evaluated with respect to intervals. Non-pure interval logics are really point or instant based logics, which have intervals only as secondary or auxiliary entities, defined with a starting point and ending point. Most interval temporal logics are really non-pure interval logics, and pure interval logics are rare. Goranko states "the single major challenge in the area of interval temporal logics is to identify expressive enough, yet decidable, fragments and/or logics which are genuinely interval-based, that is, not explicitly translated into point-based logics and not invoking locality or other semantic restrictions reducing the interval-based semantics to the point-based one" (Goranko et al., 2004).

2.2.3 Duration

Some temporal logics use a metric for time duration, where an interval might have an explicit duration of 3 seconds, or each state in a point based temporal logic might represent a second or millisecond. Bellini et al. (2000) states, "Temporal logics without a metric for time adopt a time model for which the events are those that describe system evolution... Each formula expresses what the system does at each event, events are referred to other events, and so on: this result in specifying relationships of precedence and cause-effect among events. Temporal logics with a metric for time allow the definition of quantitative temporal relationships - such as distance among events and durations of events in time units". We can see that chess is an example where we might be interested only in events (or moves), and not durations of events. Neither ATL nor any of the other logics we consider have a metric for duration, so we will not pursue this further.

2.2.4 Properties of Time

The order of the logic (propositional or first order), instants or intervals, and duration, are the largest properties which can differentiate temporal logics. Here we consider a few smaller properties which time structures may or may not have. These definitions, expressed in first order logic, are all taken from Goranko et al. (2004), with x, y and other letters representing points in time, and the $<$ and $\leq$ symbols representing the idea of one point occurring before another point in time.

An interval structure is *linear* if every two points are comparable, that is to say any two points can be found on a timeline and it can be seen which one occurs before the other, specifically:

$$\forall x \forall y(x < y \rightarrow \forall z_1 \forall z_2(x < z_1 < y \wedge x < z_2 < y \rightarrow z_1 < z_2 \vee z_1 = z_2 \vee z2 < z1))$$

An interval structure which is not linear is branching, where two points on different branches cannot always be compared to each other. Computational Tree Logic is an example of a branching logic.

An interval structure is *discrete* if every point with a successor/predecessor has an immediate successor/predecessor along every path starting/ending in it, specifically:

$\forall x \forall y(x < y \rightarrow \exists z(x < z \wedge z \leq y \wedge \forall w(x < w \wedge w \leq y \rightarrow z \leq w)))$ and
$\forall x \forall y(x \leq y \rightarrow \exists z(x \leq z \wedge z < y \wedge \forall w(x \leq w \wedge w < y \rightarrow w \leq z)))$

An interval structure which is not discrete is continuous.

An interval structure is *dense* if for every pair of different comparable points there exists another point in between, specifically:

$$\forall x \forall y(x < y \rightarrow \exists z(x < z \vee z < y))$$

An interval structure is *unbounded above* if every point has a successor, and *unbounded below* if every point has a predecessor.

Note that these properties all refer to points, and thus one cannot directly talk about these properties when using a pure interval logic which only talks about intervals and not points. But we can see that a logic which uses only intervals and not points is intuitively dense. The notion of linear can be understood in terms of intervals as well, in that any two intervals will have some relation to each other, and thus be comparable. Intervals may or may not be bounded above or below. It does not make sense to speak of discreteness when we do not have any points, rather we can describe a pure interval logic as being continuous.

Having gained a clearer understanding of temporal logic, and considered the various characteristics which differentiate them, we now turn to a potential problem for the entire logical task of prediction.

2.3 Problems with Prediction

Shoham (1987) points out three problems with prediction, known as the *qualification problem*, the *extended prediction problem* and the *persistence* or *frame problem*.

The qualification problem involves the vast amount of information required to make truly accurate predictions. If I throw the ball into the air, I might predict that it will rise a certain amount, then fall back down to the ground. However, what if there is no gravity where I am? What if the ball explodes when it reaches its highest point? What if the ball is filled with helium and continues to rise? These seem like the kind of off-the-wall concerns of a philosopher, but the distinction between such absurd concerns and more reasonable ones are not that clear. The vast amount of things which might influence an outcome is just too much for any real world prediction to be perfectly accurate. One can of course just not take many of those pieces of information into account, and make a more conservative prediction with only the most important information, though this has a higher chance of being incorrect since possibly relevant information is left out of the prediction. This relationship between the vast amount of information potentially relevant to a prediction, and the accuracy of that prediction if information is left out, is what Shoham calls the qualification problem.

There is also a problem involving the length of time over which predictions are made, called the extended prediction problem. While we can make a relatively safe prediction about what will happen in the next instant given the current state of the world and the events taking place to change it, it becomes a lot harder to make predictions about the world in longer periods of time from now. To make a prediction about the world a long time from now, you first make a prediction about the next instant, then the instant after that, and so on. We see this in chess, where you can safely predict that if you move this knight to that square this turn, it will be on that square and everything else will be in the same place at the start of the next turn. However, it becomes almost impossible to predict the state of the board 20 moves from now. Since the world is constantly changing, there are too many unknowns to take into account. Instead you might start making predictions over shorter and shorter time periods, and make an infinite number of predictions to get to your long prediction.

A related problem is what Shoham calls the persistence problem, and is similar to the frame problem of Situation calculus from McCarthy and Hayes (1981). [4]

[4]There have been various formulations of situation calculus. Originally it was formulated by McCarthy and Hayes (1981), based on the concept of a *situation*, which is a complete snapshot of the universe at some instant in time. It is thus a point based logic. Additional discussion of situation calculus and the frame problem can be found in Shoham (1987), Green (1969) and Schubert (1990).

This involves the assumption that something will stay the same over a lengthy period of time. If I paint a house red, it is now red. If I then rearrange the furniture, does the house stay red? If I add the rule that when the furniture moves, the colour of the house remains unchanged, this is incorrect, since someone might paint the house blue while I move the furniture. One would need an infinite number of these rules: when the furniture moves it does not change the colour of the sky, it does not change the value of gold, it does not change the shape of the house, and so on.

It is clear there are challenges with using logic to try to predict the future. We might run into these same problems when creating systems of artificial agents and trying to reason about their strategic abilities. Logic always uses an abstraction of the real world, a simplified model which can be reasoned about. These problems come in when the model proves inadequate to fully capture some important aspect of the real world. We then try to represent as best we can. But when designing a logic, we must try to have it as expressive as possible, so that the real world can be represented as accurately as possible, while also keeping it simple enough for algorithms to work with. This is the challenge of logic, and what the rest of this dissertation will focus on from the perspective of representing knowledge of temporal and multi-agent contexts.

2.4 Conclusion

Logic allows us to reason correctly, temporal logic allows us to specifically reason correctly about future and past events, and try to predict what the outcomes of some events or strategies might be. We have seen different types of temporal logics, and the problems one might face when designing such logics. In the next chapter we will introduce some problems which can be represented with various kinds of temporal logic, which we will use to compare various temporal logics.

Chapter 3

Problems Illustrating Practical Application of Temporal Logics

In order to compare the various logics examined in this dissertation, we will consider a few problems or puzzles. The first two are a well known and a lesser known puzzle by Edsger Dijkstra (1968) and (1982), while the third is a novel problem created for this dissertation. The goal will be to use the various logics discussed through the course of the dissertation to represent the aspects of the problem, then see how each logic enables us to reason about changes happening over time. While no single logic or formalism could properly represent any type of problem, we are interested in exactly how and where the various logics fall short, so that we may gain a better idea of how to combine them to try overcome some of these shortcomings.

All three problems contain undesirable or dangerous states, situations that should be avoided, as well as goal states that should be reached. The first problem, the *banker's algorithm*, requires various ways in which multiple agents must cooperate to achieve a goal. This will allow us to see the ways in which a logic represents multiple agents and their various actions. The second problem, the *sleeping barber*, involves overlappings of time, where a delay in the completion of one action causes unexpected results for other actions. This will allow us to see the ways in which a logic represents complex timing issues. The last problem involves trains moving gradually along a track, with agents cooperating to influence where the trains move to. This will allow us to see both multiple agents cooperating and a complex timing issue of gradual change.

In this chapter the problems will be introduced and formalised, then at the end of the following chapters these problems will be revisited in the context of the logic discussed in that chapter. We are especially interested in the strategies and decisions of multiple agents working together, as well as the structure of time which best suits each puzzle.

Note that the formal parts of the problems are very basic, and is purely for the sake of clarifying the problem to the reader. These problems will be represented in different ways by the different logics in later chapters.

3.1 The Banker's Algorithm

The classic banker's algorithm was introduced by Dijkstra (1982). It is based on the idea of a banker lending out money to a group of people who need different amounts to accomplish different things. The banker does not have enough to lend out to everyone according to their need, so instead he chooses certain individuals to lend to, until they complete their tasks and pay back the money, so that the banker can then lend to the others, and so on, until everyone has accomplished what they needed to and paid back to the banker. The puzzle is an analogy for resource allocation on computer systems. The dangerous part comes when the banker carelessly lends out only a bit to everyone, so that the banker runs out of money, while no one has enough to accomplish their task and pay back.

More formally, we have a non-empty set of N processes P, where each process p_i is engaged in some task for which it needs a number of units from a shared pool of resources to complete. All the processes share the same pool of resources, and all units of the resource are equivalent. A process may borrow one or more units from the pool and add it to their loan $loan(p_i)$, or a process may return one or more units from their loan to the pool. A process may not borrow more than it needs $need(p_i)$, nor return more than it has loaned. There is a limited number of units in the pool. The total number of units in the system is called cap, while the units currently in the pool is called $cash$, such that:

$$0 \leq cash \leq cap$$

Each process starts with some loan amount, which can be zero, and some $need$ amount, which must be less than or equal to the cap or else that process would never be able to finish. Thus:

$$0 \leq loan(p_i) \leq need(p_i) \leq cap$$

Once the process is able to loan up to its need, it completes its task and returns the loan. The units in the pool ($cash$) is also given by:

$$cash = cap - \sum_{i=0}^{N} loan(p_i)$$

An example of a potential dangerous state for two processes is as follows:

	need	loan
p_0	2	1
p_1	3	2

	cap	cash
banker	3	0

We see that none of the original 3 units are left in the pool. Process 0 needs 2 units, but has loaned only 1, while process 1 needs 3 units, but has loaned only 2. Since neither of the two processes can complete their tasks to return their loans, and since the banker is out of money, the system is stuck in this state. This is called *deadlock*, and is an undesirable state. If the banker had loaned 2 units to process 0, it would have been able to finish and return those 2. The banker could also have loaned all 3 units to process 1, which would have finished and returned those units.

Dijkstra (1982) defines a *safe pattern of loans* as when "a granting strategy exists such that it can be guaranteed that all (current and future) requests can be granted within a finite period of time".

The banker's algorithm involves the banker evaluating every possible loan and only granting it in a safe order. We first introduce the idea of a claim, which is the amount of units the process still needs before it can finish and return units:

$$claim(p_i) = need(p_i) - loan(p_i)$$

If we put all the processes in some order, called a permutation, from 0 to N, the key expression in the banker's algorithm is then:

$$\forall i : 0 \leq i < N : claim(p_i) \leq cash + \sum_{j=0}^{i-1} loan(p_j)$$

If this expression holds for a specific permutation, the pattern is safe. The expression claims that the need of any given process is less than or equal to the units in the pool and the sum of all the loans of all the processes before it. If the banker gives out loans in this order, starting with process p_0 and ending with process p_{N-1}, each process will be given enough to finish its task and return its loan.

The rest of the banker's algorithm deals with reordering the processes until the above expression holds (see Dijkstra, 1982). We will use this puzzle, but instead of the banker making decisions, we will recast it as the processes helping themselves to the pool, but making sure that they do not take so much that they cause a deadlock [1]. We thus have multiple agents taking decisions on when to take resources and how much to take. We have clear goal states, all agents accomplishing their tasks, and clear deadlock states, when no one is able to accomplish their task and the pool is empty. Time can be viewed as discrete, when in every state an agent can decide to take resources, and once an agent has enough, it will return its loan on the next state. Time can also be viewed as continuous, when an agent decides to take a loan after another agent has returned their loan, or before another agent takes a loan.

We now turn from a classic puzzle to a less known but also very interesting one.

3.2 The Sleeping Barber Problem

The sleeping barber problem is first found in (Dijkstra, 1968), where Dijkstra simply states:

"There is a barbershop with a separate waiting room... When the barber has finished a haircut, he opens the door to the waiting room and inspects it. If the waiting room is not empty, he invites the next customer, otherwise he goes to sleep in one of the chairs in the waiting room. The complementary behaviour of the customers is as follows: when they find zero or more customers in the waiting room, they just wait their turn, when they find, however, the Sleeping Barber... they wake him up."

The problem seems simple enough, the barber will continue cutting people's hair until there are no more customers, then go to sleep until the next customer arrives and wakes him. This is an analogy to certain systems in computer science, which shut down when there are no further tasks, to save resources, and only wakes again when the next task is received.

There exists some dangerous states. Take for example the situation where a single customer checks and sees the barber cutting hair, then goes to take a seat, but before he is able to take a seat the barber finishes and checks the seats, sees they are empty, and goes to sleep. The customer will not wake the barber since he is still watching the door waiting for the barber to finish and come call him, while the barber will be sleeping next to him. If no new customers come in, the

[1] Recasting the problem in this way, where the banker is no longer a decision making entity, makes it very similar to the Generalised Dining Philosophers problem, recently modelled and studied in De Masellis et al. (2019).

single customer will be waiting forever, never getting his hair cut. This state is called *starvation*, and similar to *deadlock*, should always be avoided. There are a few such situations which might arise in our barbershop if the customers and barber all follow their scripts perfectly and don't look around. All these situations are caused by events taking varying amounts of time, and some events finishing before they were expected to finish. The barber finishing the haircut and checking the chairs while the customer still walks to the chair is unexpected, but since we don't know the time any of these events take, we must design a system which will avoid Starvation regardless of how fast or slow different events are.

The following figure shows the various states which the barber and customer can be in. Note that the dangerous state occurs when the customer enters and sees the barber is busy, and while transitioning to the *Waiting* state but before reaching it, the barber finishes and sees there are no waiting customers. The barber then transitions to the *Sleeping* state, and remains there while the customer remains in the *Waiting* state.

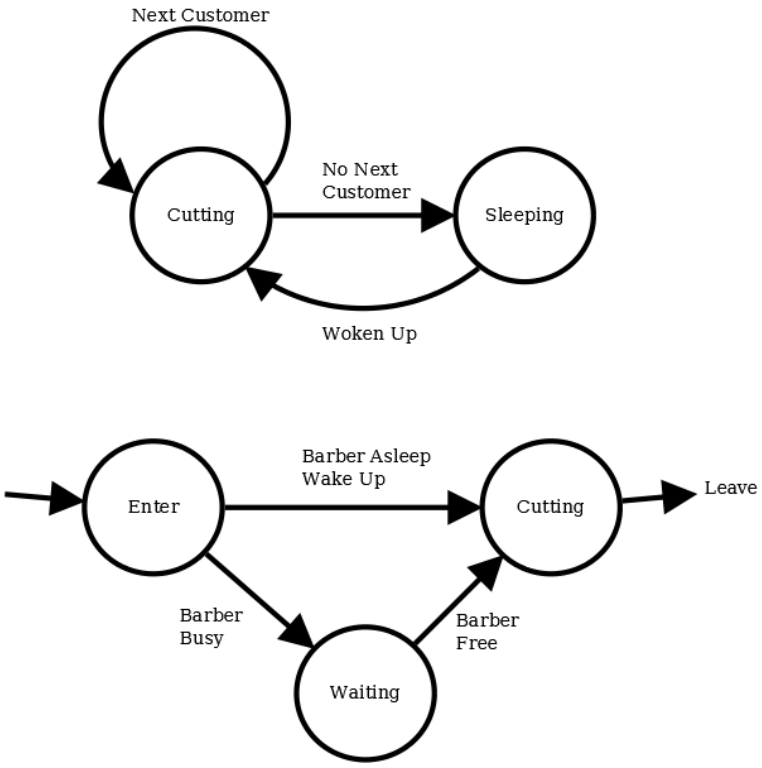


Figure 3.1: The Sleeping Barber Problem

There have been numerous solutions suggested to the problem, some of which may be found in Downey (2008). The most common solution is to restrict the barber and customers to a system where only one of them may change state at a

time. Thus while the customer goes from "Seeing the barber is cutting hair" to "sitting down on a chair", the barber is not allowed to finish the haircut or do anything else. Once the customer is seated, the barber may finish the haircut and check for more customers. This kind of solution often uses a mutex lock, a sort of mechanism that a process can acquire. There is only one lock, so only one process can have it at a time, and must release it when done before another process can acquire it. A process can only change state if they have the lock. So if the barber wants to finish cutting hair, he must first try to grab the lock, and if the lock is unavailable he must wait for it. Only once the lock becomes available can he grab it and then finish up the haircut and check for more customers.

This problem is interesting since it involves many agents making decisions as well as unknown periods of time, where the relations between the periods of time are very important. In a discrete system, each action may take multiple states. Thus it might take 4 states for the customer to see the barber is busy and take a seat, while the barber might finish the haircut in the next state and then take 2 states to check outside and see the chairs are empty.

Formally the problem for N number of waiting chairs can be represented by $N+1$ variables indicating the state of a chair, and one Boolean variable indicating the state of the barber. The N waiting chair variables, numbered 1 to N, are written in the form $Chair1 = Alice$ or $Chair2 = Empty$. The barber may also sleep in a chair, written as $Chair3 = Barber$. A special chair is the barber's chair, where the customer sits when getting a haircut, numbered as 0 and written $Chair0 = Bob$, thus the $N+1$ chairs in total. One last variable is a Boolean showing the state of the barber, called $BarberBusy$. If the barber is busy cutting hair it is true, otherwise it is false.

A new customer walking in would check $BarberBusy$ and if true, take the next seat from 1 to N which is empty. If $BarberBusy$ is false, the customer will search the chairs for the barber and wake him, then go get their hair cut. If the barber finishes a haircut, he will look at the first chair and if it is empty he will pick a random chair and go to sleep, if it is not empty he will take the next person for a cut and all customers shift one chair on.

The dangerous state then comes when the customer sees $BarberBusy = true$ and $chair1 = empty$, and goes to sit down on the first chair. Before the customer finishes this, while $chair1 = empty$, the barber finishes, sets $BarberBusy = false$ and checks the chairs, sees they are all empty, then goes to sleep in one of the other chairs. After this the customer sits down in the first chair, and never gets a haircut.

Having considered two classical Dijkstra problems, we now turn to a unique problem involving trains.

3.3 The Trains Problem

This scenario involves trains of different colours, running on the same track, each train heading for the station of its own colour, and not the stations of other colours.[2] A group of agents run around changing the settings on various crossroads to try to facilitate this. The trains have a common starting point, a tunnel, from which they emerge one after the other, with some time period in between, and then start moving along the track. The trains have no steering ability, and simply follow the track at a constant speed hoping to arrive at the destination. The tracks connect on crossroads, and each crossroad is in a specific setting. The setting on a crossroad will determine which way the train goes when it heads over the crossroad. The agents have to manually run to a crossroad and pull a switch to change the setting. This becomes hard when the red train needs to go left, the blue train right, then the green train left again, and all three are right behind one another. The conductor would need to make sure the crossroad is in a left setting, then pull the switch as soon as the red train has crossed and before the blue train does, and then pull the switch again once the blue train has passed and before the green train. If there are more crossroads than agents, it becomes necessary for the agents to plan ahead and work together so that there is always someone at the crossroad when there needs to be.

Formally we have a set T of n trains t_i, and a corresponding set S of n stations s_j, where each train must go to the station where $i = j$. We have a set R of rail pieces r_k and a set C of crossroads c_l. We also have a set A of agents. We have five mapping functions:

- $location : t \in T \rightarrow r \in R \cup c \in C \cup s \in S$ which maps each train to its current location on either a piece of rail, a crossroad or a station.
- $agentLocation : a \in A \rightarrow c \in C$ which maps each agent to the crossroad it is currently at.
- $connection : (r \in R) \rightarrow (r \in R) \cup (c \in C) \cup (s \in S)$ which maps each piece of rail to the next piece of rail, crossroad or station which it connects to.
- $crossConnection : c \in C \rightarrow \mathcal{P}(R)$ which maps each crossroad to all the rail pieces it can potentially connect to, where $\mathcal{P}$ denotes the power set.
- $setting : c \in C \rightarrow r \in R$ which maps each crossroad to its current setting for a single rail piece.

[2] This puzzle was inspired by the entertaining Train of Thought game by Lumosity on www.lumosity.com.

Note that this very basic formalisation would allow multiple trains on a single track, and would have no way of ordering them for that track. The different logics in later chapters have different ways of addressing this problem, so it is not addressed here.

The following figure shows one possible layout for a situation involving four trains and four stations, two agents (Alice and Bob), three crossroads and seven pieces of railroad. Note that the triangles pointing up are trains, while the triangles pointing down are agents. This convention will be kept to for other figures of the train problem in this dissertation.

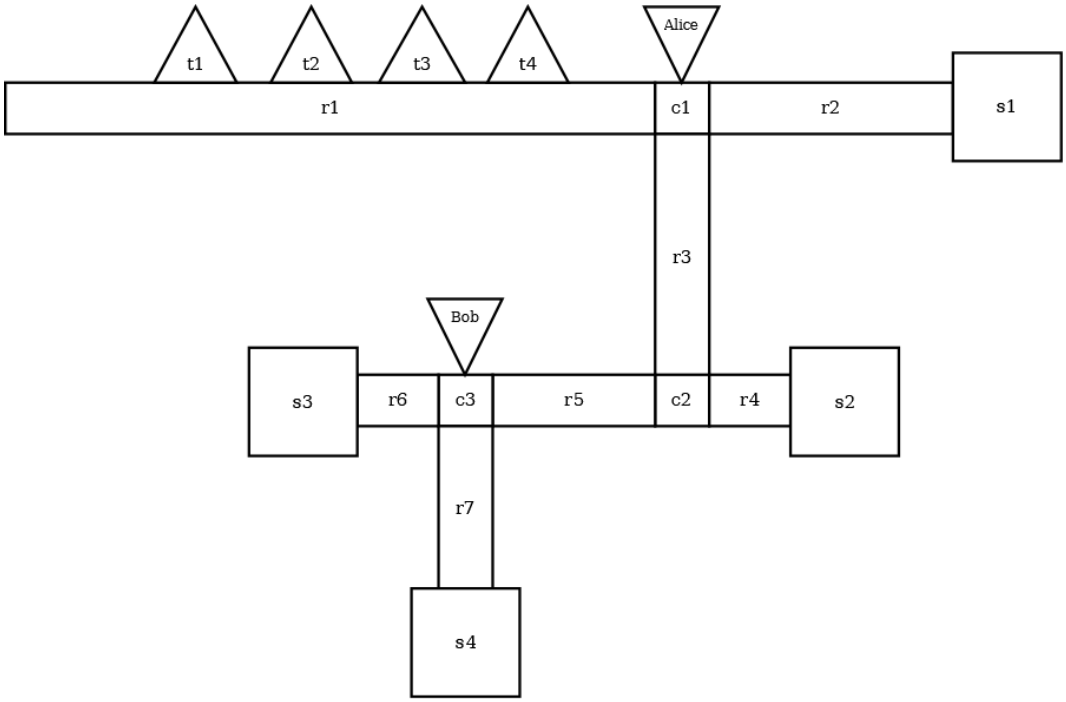


Figure 3.2: Example of a Train Problem

A train reaching the end of a rail will arrive on the next rail as determined by $connection(currentrail)$. Note that a rail only ever connects to a single other rail, crossroad or station, while a crossroad can potentially connect to every rail. A rail only runs in one direction, we do not have to be concerned with head on collisions. However the possibility exists of a crossroad connection to an earlier rail and forming a circular route. Agents are only ever at crossroads, and take a period of time to move from one crossroad to the next. An agent has to be at a crossroad to change the setting for that crossroad. When a train finally arrives at a station, it remains there until the end.

The timing of how long it takes a train to move to the next rail piece, and

how long it takes an agent to move from one crossroad to the next, is all for the respective logics to represent. This will also influence how the issue of multiple trains on a single track is addressed. The dangerous states we must avoid is when a train arrives at the wrong station, while the goal state is that every train arrives at its station.

A continuous time model seems most natural here, since we are reasoning about trains gradually travelling along variable length tracks. A discrete time model will involve breaking up the rail pieces into smaller pieces, so that in every state transition, a train may advance one piece, and an agent may take some set amount of state transitions to move from one crossroad to the next.

With these three puzzles, we will be able to compare the logics and see how they represent various situations, and how easy or hard it is to reason effectively about a situation in each logic.

Chapter 4

Alternating-time Temporal Logic

ATL was introduced by Alur et al. (2002) for temporal reasoning about open multi-agent systems. The semantics of ATL is defined over multi-agent transition systems, which are systems consisting of at least two agents, each with its own goals and action types, who must cooperate or compete, and whose actions advance the state of the system. The system has many states, and a transition from one state to another occurs when the agents carry out their chosen actions. The agents all take their actions at the same time.

In the original paper (Alur et al., 2002), ATL is introduced as an alternative to *linear-time temporal logic* which has universal quantification over all possible paths, and *branching-time temporal logic* which uses existential and universal quantifiers for the paths. The concurrent game model of ATL is presented as a game between the system (agents taking actions), and the environment (the states, effects of those actions), where the system and environment alternate turns. ATL and its extension ATL* have "gradually become the most popular logical formalism for reasoning about strategic abilities of agents in synchronous multi-agent systems" (Goranko et al., 2018).

This chapter will start by considering the syntax and semantics of ATL, followed by a section looking at the problems from the previous chapter from the perspective of ATL. Following this will be a chapter covering a related logic and a chapter covering the extensions of ATL.

4.1 Syntax and Semantics

4.1.1 Concurrent Game Models

While logics like LTL and CTL are interpreted over Kripke structures, ATL uses a different model called a *concurrent game model*. Recall the strategic game from

section 2.1.5, where each of the agents chose an action which then led to a payoff. One can decide to play this game multiple times, where each time the combined actions of the agents lead to the next state instead of ending the game and determining winners and losers. As before, every agent chooses an action, and the corresponding action profile determines the outcome. Now instead of stopping the game there, the outcome determines the state in which the next turn starts, where agents now repeat the process. Depending on the outcome of the previous turn, new actions might now be available to agents, and some old actions might no longer be valid. This game can then go through many turns before finally ending, or never end. An example of this is the classic board game *Diplomacy*, where all players take their turns simultaneously by writing their moves in secret on pieces of paper, which are then revealed at the same time and all moves made according to the papers. There is also a variant of chess, called *Simultaneous Chess*, which follows the same principle. *Concurrent game structures* (CGS) and *concurrent game models* (CGM) allow us to represent these kinds of situations. Bulling et al. (2015) provides a definition of both a CGS and a CGM, which we can see here:

A *concurrent game structure* (CGS) is defined as a tuple $M = (Agt, St, Act, act, out)$ where:

- Agt is a nonempty finite set of agents.
- St is a nonempty finite set of states, the system is in one state per turn.
- Act is a nonempty finite set of actions which can be performed.
- act is a function which assigns to each agent the list of actions available to that agent in a specific state, since not all actions are available to every agent in every state. It is defined as $act : Agt \times St \to \mathcal{P}(Act) \setminus \{\varnothing\}$
- out is an outcome function $out(q, x_1, x_2, ..., x_k)$ that assigns a new successor state in St to each combination of current state q and action profile $(x_1, x_2, ..., x_k)$, defined as

$$out : St \times \prod_{a \in Agt} x_a \to St$$

  where the Pi notation signifies the Cartesian product of the set of actions, one for each agent.

A *concurrent game model* (CGM) is defined in a similar way, but with the addition of a set of atomic propositions $Prop$ and a labelling function $V : St \to \mathcal{P}(Prop)$, which maps each state to the set of atomic propositions which are true

in that state. This is according to Bulling et al. (2015), but there are variations on this. Two variations to take note of will also be presented here.

In the original paper, Alur et al. (2002) define a *concurrent game structure* in a similar way to Bulling et al.'s *concurrent game model*. There are some differences though. The Alur et al. (2002) definition for a *concurrent game structure* is as follows:

$$M = (k, Q, \Pi, \pi, d, \delta)$$

where:

- k is the natural number of players or agents, numbered 1 to k. This is similar to Bulling et al.'s Agt, but is simply a number instead of a set of agents. This changes some of the formal definitions for the other functions, but works in a similar way.
- Q is the set of states similar to St.
- Π is set of propositions similar to $Prop$.
- π is the labelling function similar to V.
- d is called the *move function*, but is similar to Bulling et al.'s *act*.
- δ is the transition function similar to *out*.

One can note that the CGS from Alur et al. (2002) does not have the set of available actions Act which the CGM from Bulling et al. (2015) has.

Another important formulation of a CGM comes from Goranko et al. (2018) where such a model is defined as follows:

$$M = (Agt, St, \Pi, Act, d, o, v)$$

where:

- Agt, St, Act are the same as in Bulling et al.'s CGS.
- Π is the same as in Alur et al.'s CGS and similar to $Prop$ in Bulling et al.'s CGM.
- d is similar to *act* from Bulling et al.'s CGS.
- o is similar to *out* from Bulling et al. and δ from Alur et al.
- v is similar to V from Bulling et al. and π from Alur et al. Though it is defined $v : \Pi \to \mathcal{P}(St)$ which is the reverse of how it is defined by Bulling et al..

Thus there are many ways of defining a concurrent game structure and a concurrent game model. The principles are the same however. A list of agents and a list of states are given. In each state the agents all take actions simultaneously which affect what the next state will be. Going forward in this chapter, a CGM with the following formulation will be used:

$$M = (Agt, St, \Pi, \pi, Act, act, \delta)$$

where:

- Agt is a nonempty finite set of agents.
- St is a nonempty finite set of states.
- Π is a nonempty finite set of atomic propositions.
- π is a labelling function mapping each state to the set of atomic propositions which are true in that state, formally $\pi : St \to \mathcal{P}(\Pi)$.
- Act is a nonempty finite set of actions.
- act is a function which for a given agent and a given state returns the actions available to that agent in that state, formally $act : Agt \times St \to \mathcal{P}(Act) \setminus \{\varnothing\}$.
- δ is a transition function which for a state and set of actions $(x_1, x_2, ..., x_k)$ taken by the agents returns the next state.

  $$\delta : St \times \prod_{a \in Agt} x_a \to St$$

  Where the Pi notation signifies the product of the set of actions, one for each agent.

As a clarifying example of the usage of the CGM, consider two cats, Alice and Bob, sitting on either ends of a table, with an expensive pot standing between them. Alice is sitting on the left and Bob on the right of the table, where left and right is seen from the perspective of a viewer from the front and not relative to the cats. The cats have the option to either sit still, or swipe at the pot, which will move it. A cat can swipe to the left or to the right, moving the pot to the left or to the right, all still relative to the viewer from the front. The pot can be in one of four states, near Alice, near Bob, in the centre of the table, or on the floor, broken into many pieces. If the pot is near Alice, she can swipe it left to make it fall to the floor, or swipe right to move it to the centre, while Bob cannot reach it that turn, and vice versa. If the pot is in the middle, either cat can push it to the other cat if they swipe towards it, but have no effect if they swipe away from it.

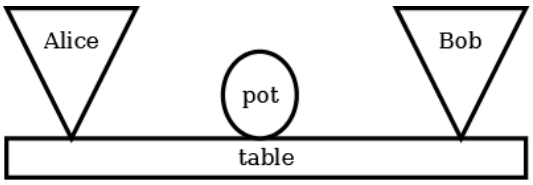


Figure 4.1: CGM Example with Two Cats

If both cats swipe at a central pot, it remains in position. Once the pot is on the floor, neither cat can do anything.

Formalised as a CGM, it would look like the following:
$M = (Agt, St, \Pi, \pi, Act, act, \delta)$

- $Agt = \{Alice, Bob\}$ the two cats.
- $St = \{Left, Centre, Right, Floor\}$ indicating the position of the pot.
- $\Pi = \{PotFloor, PotCentre, PotRight, PotLeft\}$ where one will be true and the other three will be false for any given state.
- $\pi(Left) = \{PotLeft, \neg PotRight, \neg PotCentre, \neg PotFloor\}$ is an example of the labelling for the $Left$ state, with similar definitions for the other three states.
- $Act = \{SwipeLeft, SwipeRight, DoNothing\}$ the actions which the cats can do.
- $act(Alice, Centre) = \{SwipeRight, DoNothing\}$ is an example of the actions which Alice can do if the pot is to the right of Alice (in the centre of the table). She can only swipe right to push it further, not left. Similar definitions for all other combinations of agents and states.
- $\delta(Left, \{SwipeLeft, DoNothing\}) = Floor$ is an example of the transition when the pot is in front of Alice and she swipes it left off the table and Bob does nothing. In this case, in the next state the pot will be in pieces on the floor. Note that the chosen action for Alice is first, followed by the chosen action for Bob. Similar definitions for all other combinations of states and action profiles.

Having considered the CGM over which a formula is interpreted, we now turn to the formulæ themselves.

4.1.2 Syntax

Let Π be a set of atomic propositions with $p \in \Pi$ and let Agt be a set of agents with $A \subseteq Agt$ being a coalition. Then the syntax of an ATL formula φ is given as:

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \ \ll A \gg X\varphi \mid \ \ll A \gg G\varphi \mid \ \ll A \gg \varphi_1 U \varphi_2$$

where $\ll A \gg$ is the path quantifier, and is read as "the coalition A has a strategy". The X symbol is the *next* operator, while G is the *always* operator. Note that Alur et al. (2002) represents the *next* operator with $\bigcirc$ and the *always* operator with $\Box$, but we will use X and G to be consistent with most other formulations. The U symbol is the *until* operator. The path quantifier ($\ll A \gg$) together with a temporal operator (X, G or U) indicate that the coalition has a strategy to make that formula true at that time.

Similarly to CTL, the *eventually* operator F (called $\Diamond$ by Alur et al. (2002)) can be derived as follows:

$$\ll A \gg F\varphi := \ll A \gg \top \ U \ \varphi$$

Where $\top$ means *true*.

4.1.3 Semantics

The evaluation of an ATL formula φ on a state q of M, where $M = (Agt, St, \Pi, \pi, Act, act, \delta)$ is a concurrent game model, written as $q \vDash \varphi$, is inductively defined as follows (this semantics is based on Alur et al. (2002), but symbols have been changed for consistency with the syntax above):

- $q \vDash p$, for all propositions $p \in \Pi$, iff $p \in \pi(q)$.
- $q \vDash \neg\varphi$, iff $q \nvDash \varphi$.
- $q \vDash \varphi_1 \vee \varphi_2$, iff $q \vDash \varphi_1$ or $q \vDash \varphi_2$.
- $q \vDash \ll A \gg X\varphi$, iff there exists a set of strategies S_A, one for each player in A, such that for all computations $\lambda \in \delta(q, S_A)$, we have $\lambda[1] \vDash \varphi$.
- $q \vDash \ll A \gg G\varphi$, iff there exists a set of strategies S_A, one for each player in A, such that for all computations $\lambda \in \delta(q, S_A)$ and all positions $i \geq 0$, we have $\lambda[i] \vDash \varphi$.
- $q \vDash \ll A \gg \varphi_1 U \varphi_2$, iff there exists a set of strategies S_A, one for each player in A, such that for all computations $\lambda \in \delta(q, S_A)$, there exists a position $i \geq 0$ such that $\lambda[i] \vDash \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \vDash \varphi_1$.

Recall from chapter 2 that λ is called a path, and refers to a sequence of states. Also recall that a strategy S_a is a function which for every state and for a specific agent a returns an action that agent will do in that state, while S_A is similar but for the coalition A.

4.1.4 ATL*

It is important to note that there also exists a version called ATL*, where a distinction is made between *state formulae* and *path formulae*. A state formula is evaluated on a specific state, while a path formula is evaluated on an entire path or play. The syntax is as follows:

$$\text{State Formula: } \varphi ::= p|\neg\varphi|\varphi \vee \varphi| \ll A \gg \varphi$$
$$\text{Path Formula: } \Phi ::= \varphi|\neg\Phi|\Phi \vee \Phi|X\Phi|\Phi U\Phi$$

ATL is a simpler fragment of ATL* which uses only state formulae.

To see how this syntax and semantics are used in practice, the following section represents our three problems in ATL.

4.2 Problems

4.2.1 The Banker's Algorithm

This problem will be approached from the perspective of the agents who are borrowing resources, rather than from the perspective of the banker. The agents must decide among themselves when and how much to borrow, and come up with a strategy for avoiding deadlock. Each turn, an agent can either take the *borrow* action, for a specific amount, or the *waiting* action. It would be helpful to represent numbers here, such as the *need* or *loan*, but in keeping with the CGM formulation, this must be represented as atomic Boolean propositions. We will have propositions for *loan(Alice)=5* and *loan(Alice)=4* and *loan(Alice)=3*, and so on for each possible value. These mean that Alice has taken a loan of 5, 4, and 3 respectively. Only one of these propositions should be true at a time. Three agents will be taking loans, Alice, Bob and Charlie. I am not aware of any other representations of the banker's algorithm in ATL.

The concurrent game model will be as follows:

$$M = (Agt, St, \Pi, \pi, Act, act, \delta)$$

where:

- $Agt = \{Alice,\ Bob,\ Charlie\}$ are the three agents.

- $St = \{q_0, q_1, q_2, ..., q_n\}$ to represent all possible states, specifically $St = \mathcal{P}(\Pi)$.
- Π = {*cash=6, cash=5, ..., loan(Alice)=6, loan(Alice)=5, ..., loan(Alice)=0, loan(Bob)=6, ..., loan(Charlie)=6, ..., need(Alice4), need(Bob6), need(Charlie3)*} are our propositions, where *cash* and *loan* are variables, which can change after every round, and thus need a proposition for every possible value. All the loan propositions start as false except for the *loan(agent)=0* ones, while all the cash propositions start as false except for *cash=6*, the starting amount in the bank. The three *need* propositions start out true, but will become false once an agent has borrowed enough to complete their task. For convenience we have included the amount each agent needs in the name, thus Alice needs 4 units, Bob needs 6 units and Charlie needs 3 units.
- π = the labelling function.
- *Act* = {*Borrow(1), Borrow(2), Borrow(3), ..., Borrow(6), Wait*} are our possible actions. For each turn, each agent may choose to borrow an amount or to wait.
- *act* = the function mapping for each state and each agent the actions available to that agent in that state. For this problem, no restrictions will be imposed here, as agents attempting to borrow more than what is in the pool will be dealt with by the transition function, where that attempt simply fails.
- δ = the transition function. If all agents wait, nothing changes. If agents attempt to borrow *cash* from the pool, if the sum of all attempts is less than the total in the pool, all agents succeed, and the system advances to a state where the *cash* proposition is now less by the appropriate amount, and each agent's *loan* is more by the appropriate amount. If the agents attempt to borrow more than what is in the pool, they all fail, and instead the system advances as if they had chosen *wait*, and nothing changes. If an agent's *loan* equals their *need*, in the next state that agent's *need* proposition becomes false, and they only take the *wait* action afterword, their loan becomes 0, and what was borrowed is added again to the *cash* proposition, thus the loan is returned.

Here are some examples of formulæ that can be written:

- $G\ \neg(loan(Alice) = 6\ \wedge\ loan(Bob) = 6)$ Both Alice and Bob will never be able to borrow 6 units each at the same time, as the total pool contains only 6 units.

- $(loan(Alice) = 4 \ \wedge \ need(Alice4)) \rightarrow (X(loan(Alice) = 0 \ \wedge \ \neg need(Alice4))$ If Alice has loaned the amount she needs, immediately thereafter she will return the loan and no longer need anything.
- $\ll A \gg (loan(Alice) = 2 \wedge loan(Bob) = 2 \wedge loan(Charlie) = 2)$ The coalition A consisting of Alice, Bob and Charlie has a strategy to ensure they each get a loan of 2 units. This is a simple strategy involving them each attempting to borrow 2 units, and thus all succeeding. It is a dangerous strategy though, which will result in deadlock.

Deadlock must be avoided, which is when the pool has been depleted and there are still agents which have needs and no agents able to finish their tasks. The pool might become depleted in one turn, and the agent only completes their task and returns the resources in the next turn. If there is a depleted pool, and in the next turn the pool is still depleted, then the system must be in deadlock, since no agents completed their tasks. Or in an ATL formula:

$$cash = 0 \ \wedge \ Xcash = 0$$

If the agents can avoid this formula from becoming true, they can avoid deadlock. Technically for the banker algorithm deadlock also includes the state where there are some resources left in the pool, but not enough for any agent to borrow and complete their task, but for simplicity a stricter definition will be used here which requires the pool to be empty for two consecutive turns. If there are still units left in the pool, the agents can still attempt to borrow, even though it won't be enough. The goal of the banker's algorithm is to allocated resources among the agent in such a way that they may all finish their work, which can be represented by the following formula:

$$\neg need(Alice) = 4 \ \wedge \ \neg need(Bob) = 6 \ \wedge \ \neg need(Charlie) = 3$$

The discrete time model of ATL is well suited to this problem, since the time it takes to make a loan or the time to pay back a loan is not relevant, instead events are instant and we only care about the order in which they happen. The multiple agents working together can also very naturally be represented in ATL.

4.2.2 The Sleeping Barber

This time Alice, Bob and Charlie will be getting a haircut. The main problem with ATL and the Sleeping Barber problem, is the time model. The Sleeping Barber problem depends on the fact that certain processes might be faster or slower than others, and finish at certain times. This cannot accurately be represented in ATL, since it was not made for such problems. A random feature will be added to the

transition function, where an action has a chance to fail and instead the agent only takes the *wait* action for that turn, and in the next turn attempts that action again. This will allow some agents to finish actions and move on before others. There will be a few positions each agent can occupy. There are 3 waiting chairs, called c_1, c_2 and c_3. There are the *enter* and *cut* positions, an agent entering the shop will be in the *enter* position from where they will check the status of the barber, while an agent getting a haircut and the barber giving a haircut will both be in the *cut* position. There are also the $goToCut$, $goToWait$ and $goToSleep$ positions, which involves going to the barber's chair, or going to wait or sleep in either of c_1, c_2 or c_3. It is in these $goTo$ positions where an agent might randomly stay for 2 or 3 turns before succeeding, and where various agent can overtake each other, thus leading to the Sleeping Barber problem.

The concurrent game model will be as follows:

$$M = (Agt, St, \Pi, \pi, Act, act, \delta)$$

Where:

- $Agt = \{Alice,\ Bob,\ Charlie,\ Barber\}$ are our four agents.
- $St = \{q_0, q_1, q_2, ..., q_n\}$ to represent all possible states, specifically $St = \mathcal{P}(\Pi)$.
- $\Pi = \{barberSleepC1,\ barberSleepC2,\ barberSleepC3,\ barberCut,\ barberGoToSleep,\ barberGoToCut,\ aliceEnter,\ aliceWaitC1,\ aliceWaitC2,\ aliceWaitC3,\ aliceCut,\ aliceGoToWait,\ aliceGoToCut,\ aliceOutside,\ bobEnter,\ ...\ charlieEnter,\ ...\}$ are our propositions. Only one of the barber statements, and one of each agent statement, can be true at a time. These all indicate the current position of that agent. All customers start outside and enter at random times.
- π = the labelling function.
- $Act = \{Cut, GoToCut, Wait, GoToWait, Sleep, GoToSleep, WakeBarber, GoToWakeBarber\}$ are our possible actions. For example, if the barber is done cutting hair, and there are no more customers waiting in the chairs, he takes the $GoToSleep$ action, which in the next state makes $barberGoToSleep$ true and $barberCut$ false. After this he continues to take the $GoToSleep$ action each turn until it succeeds, which might be a few turns, and then $barberGoToSleep$ becomes false and $barberSleepC2$ becomes true (an empty chair is randomly chosen once the $GoToSleep$ action succeeds).
- act the function mapping for each state and each agent the actions available to that agent in that state. Agent actions are predetermined according to the

rules laid out in the introduction for this problem, and there are no choices to be made by the agents. This function will thus return the one action available to that agent, and that agent will take it.

- δ the transition function. The key here is the randomness which can cause agents to be out of sync with each other. Before an agent can take an action, it must take the associated *GoTo* action to get into a position where it can take that action, and this might take a few tries. While in the *goTo* state, the agent is not looking around and changing strategy, but is continuing to attempt that action each turn. Some actions also require randomness when they succeed, such as which chair to wait or sleep on.

An example formula that will be true during a busy day at the barbershop might be:

$$barberCut \wedge aliceCut \wedge bobWaitC3 \wedge charlieWaitC1$$

Perhaps in the quiet afternoon, the barber goes to sleep, thus:

$$barberSleepC1 \wedge aliceOutside \wedge bobOutside \wedge charlieOutside$$

An example of a starvation situation will be if the barber is sleeping and a customer is waiting:

$$barberSleepC1 \wedge charlieWaitC3$$

Generally a customer will starve if they are waiting forever to be cut, or:

$$G\ charlieWaitC1\ \vee\ G\ charlieWaitC2\ \vee\ G\ charlieWaitC3$$

The goal then is to avoid starvation of any of the agents, or to avoid the following:

$$\begin{aligned}&(G(AliceWaitC1 \vee AliceWaitC2 \vee AliceWaitC3)\ \vee\\ &G(BobWaitC1 \vee BobWaitC2 \vee BobWaitC3)\ \vee\\ &G(CharlieWaitC1 \vee CharlieWaitC2 \vee CharlieWaitC3))\end{aligned}$$

or alternatively

$$F(AliceCut) \wedge F(BobCut) \wedge F(CharlieCut)$$

The idea of a coalition strategy does not really come up here, since there is only ever one action available to each agent, and thus they cannot have strategies. A difficulty here is also the randomness, which makes it much harder to predict certain outcomes. The randomness is needed however to ensure agents can overtake each other, something not possible in ATL otherwise. The formula $XBobCut$ does not follow from $BobGoToCut$, but $FBobCut$ does, since the action should eventually randomly succeed, but it is not known when. ATL does not seem to be designed for these types of problems.

4.2.3 The Trains Problem

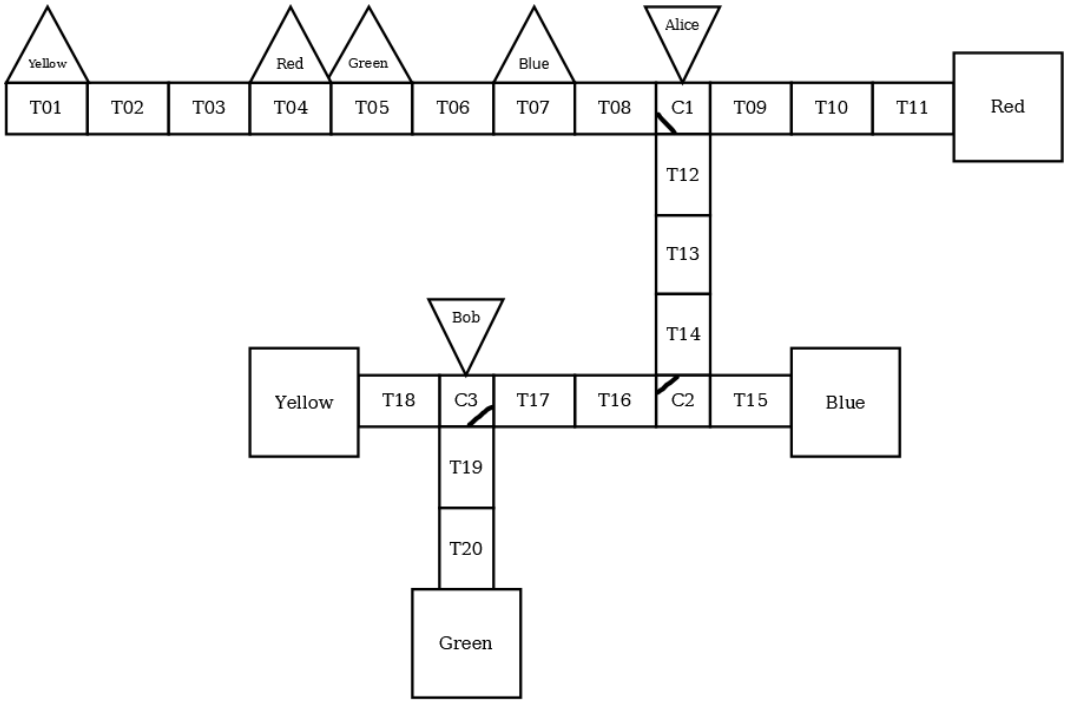


Figure 4.2: An Example of the Trains Problem

The figure shows a train track, divided into 20 sections, T01-T20. On the track there are 4 trains of different colours, each on its own piece of track, and 4 stations of matching colours. There are 3 crossroads, C1, C2 and C3 connecting pieces of track together. Lastly there are 2 agents, Alice and Bob, who are standing at different crossroads. A crossroad can be in of two settings, C2 for example can connect T14 to either T15 or T16. The initial settings for crossroads are shown by the filled section of the crossroad, T08 to T12, T14 to T16 and T17 to T19. The agents can only be at crossroads, and only at one crossroad at a time.

In this example, the trains will be moving, one piece of track every state change, or turn. If a train is on a piece of track just before a crossroad, in the next state it will be on the next piece of track that crossroad is connecting to. Trains can only move forward and cannot stop. Alice and Bob can pull levers to switch the setting for a crossroad, or they can move to another crossroad, both actions taking one state change. The idea is that each train should eventually reach the station of its colour, and not a station of a different colour. The agents Alice and Bob must work together to move between crossroads and pull levers to get the trains to the right places.

The concurrent game model will be as follows:

$$M = (Agt, St, \Pi, \pi, Act, act, \delta)$$

- $Agt = \{Alice, Bob\}$ are our two agents.
- $St = \{q_0, q_1, q_2, ..., q_n\}$ to represent all possible states, specifically $St = \mathcal{P}(\Pi)$.
- Π = {*yellowTrainT*01, *yellowTrainT*02, ..., *yellowTrainT*20, *blueTrainT*01, ..., *blueTrainT*20, *redTrain...*, *greenTrain...*, *aliceC*1, *aliceC*2, *aliceC*3, *bobC*1, *bobC*2, *bobC*3, *T*08*C*1*T*12, *T*08*C*1*T*09, *T*14*C*2*T*15, *T*14*C*2*T*16, *T*17*C*3*T*18, *T*17*C*3*T*19} are our propositions. The *train* propositions refer to their current positions on a piece of track, the *alice* and *bob* propositions refer to their current positions on one of the three crossroads, the *T..C..T..* propositions refer to the setting of the crossroads, so *T14C2T16* will mean that *T14* is connected to *T16* through crossroad *C2*. These are all the possible propositions in this example, a set will be made out of a set of these propositions which are true in that state.
- π = The labelling function.
- Act = {*FlipSwitch, TravelToC1, TravelToC2, TravelToC3*} are our possible actions.
- *act* the function mapping for each state and each agent the actions available to that agent in that state. *FlipSwitch* will always be available regardless of the state, but only 2 of the 3 *TravelTo* actions will be available depending on which crossroad an agent are currently at. For example, if Alice is at C1, it is represented as $act_{alice}(q) = \{FlipSwitch, TravelToC2, TravelToC3\}$ with $aliceC1 \in \pi(q)$.
- δ the transition function. In producing the next state, each train will be moved one track section forward. If a train was on *T05* it will be on *T06* in the next state. If a train is at the track just before a crossroad, it will be on a track just after the crossroad depending on the current setting of the crossroad. If a train is on the track just before a station, it will remain there forever, since it has arrived. If Alice or Bob attempt to *TravelTo* a crossroad, they will be at that crossroad during the next state. If Bob or Alice attempt to *FlipSwitch*, the crossroad will be in a different setting during the next state. Since every crossroad only has 2 settings, there is no need to specify to what setting the switch is flipped.

From here ATL formulæ can be written. If we take the coalition $\ll A \gg = \{Alice, Bob\}$, we can write:

$$\ll A \gg X\, blueTrainT08$$

meaning that the coalition has a strategy to move the blue train to track 8, which will be true in the initial state q_0 regardless of what Bob and Alice decide to do since the blue train will move into that space in the next turn.

$$\ll A \gg G\, bobC2$$

which might be true in q_1 if Bob decides first to travel to C2 during q_0, and then to stay there forever. The goal will be

$$\ll A \gg (F\, blueTrainT15 \wedge F\, greenTrainT20 \wedge F\, redTrainT11 \wedge F\, yellowTrainT18)$$

We see that ATL allows us to represent a dynamic system. The concurrent game model of ATL was originally introduced as a game between the system (agents) and the environment, alternating moves between each other (Alur et al., 2002). This is where the *Alternating-Time* part of the name comes from. On the one side we have a system of agents who can take actions to move themselves to a different location or move the switch on a crossroad. On the other side we have the environment moving the trains along each turn. This problem would of course be very easy if the trains could stop, in which case they would simply stop in front of a crossroad and wait until it can be changed correctly. The challenge of this game lies in the fact that the agents have a limited amount of time, and if they are unable to change a crossroad in time, a train would be on the wrong track. While we can represent the agents and their strategies, we cannot capture the gradual nature of smooth movement along a track. We had to divide each track into segments to accommodate the discrete nature of the ATL time model.

We will now consider some extensions of ATL. [1]

4.3 Extensions of ATL

While ATL is an expressive logic, it has some shortcomings. Over the years many extensions of ATL have been introduced to attempt to address these. Bulling et al. (2015) discusses three limitations and how they have been addressed. Firstly

[1] The interested reader might also want to have a look at *Coalition Logic*, introduced by Pauly (2002) around the same time as ATL. It has been shown by Goranko (2001) that the semantics of Pauly's Coalition Logic and ATL are equivalent, and that Coalition Logic can be embedded in ATL.

there is the issue of strategic commitment, where an agent may have a strategy to achieve some goal, yet not be bound by that strategy. Secondly, there is the issue of strategies not being explicit entities in the logic which relies so heavily on strategies. Thirdly, there is the issue of incomplete information, and agents not necessarily having full knowledge of the entire system. We would point out that an additional possible issue is the inability to reason about gradual changes in the system or about overlapping actions. Future chapters will explore the logic of intervals and how it might be combined with notions from ATL in order to address this last issue. The rest of this section will focus on the various extensions to ATL which seek to address the three issues pointed out by Bulling et al. (2015). By considering the various ways in which ATL has been extended in the past, we gain a better idea of how we might extend it in future.

4.3.1 Strategic Commitment

The semantics of ATL/ATL* does not commit an agent to its strategies. A formula may be evaluated and it is seen that in the current state the coalition A has a strategy to make p true at some stage, yet A may never take the actual actions which will end up making p true. This can become a problem in some situations, like the following example from Bulling et al. (2015):

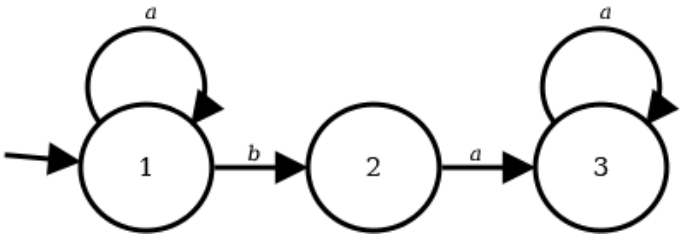


Figure 4.3: Example from Bulling et al. (2015)

Consider this single agent system, where 1, 2 and 3 are states and a and b are actions. Now while in state 1 the agent can take action a to remain in the same state, or action b to transition to state 2. If it is given that p is true only at state 2, consider the formula $\langle\langle A\rangle\rangle Xp$, and see that it is true at state 1, since if the agent A takes action b, p will become true in the next state. Consider also the formula $\langle\langle A\rangle\rangle G\ \langle\langle A\rangle\rangle Xp$, which states that A has a strategy to always ensure that it has a strategy to make p true in the next turn, notice that this is also true at 1, since $\langle\langle A\rangle\rangle Xp$ is true at 1 and the agent can choose to take action a each turn, thus remaining forever in 1. But this is a contradiction, since the agent can always remain in a state where p is possible the next turn, so long as the agent never actually takes the action to make p true. As soon as the agent takes action b,

and transitions to 2, the formula $\langle\langle A\rangle\rangle Xp$ is no longer true. Bulling et al. (2015) remarks: "However, this system does not have exactly the property we had in mind because by following that strategy, agent $[A]$ dooms itself to *never access the resource* [set p to true] - in other words, $[A]$ can ensure that it is forever *able* to access the resource, but only by never *actually* accessing it. Indeed, while $[A]$ can force the *possibility* of achieving p to be true forever, the actual achievement of p destroys that possibility".

There are two alternatives to deal with this issue:

1. **Irrevocable Strategies**, which forces commitment to a strategy by not allowing the agent to revoke a chosen strategy. The model is updated when agents choose a strategy, thus locking them into that strategy. Two extensions of ATL use this method, MATL and IATL, for both memory-based and memoryless strategies, see Ågotnes et al. (2007) and Ågotnes et al. (2008).

2. **Strategy Contexts**, proposed by Brihaye et al. (2009), do not change the model but instead keep track of all the strategies currently selected. Agents may select strategies, which are added to the context, but can change their strategies later, which then updates the context. Formulæ are evaluated with respect to the strategy contexts, thus in a nested formula, if the agent chooses one strategy in one part and a different strategy in a different part, the context is updated with only one of the two strategies for that formula. For more see Troquard and Walther (2012).

Strategy contexts are thus more flexible than irrevocable strategies, and do not require a change to the model, but add an additional element to keep track of. Irrevocable strategies require only a simple change to the model, but completely commit agents to a strategy.

We see that contradictions can arise when agents are not committed to their strategies. We saw two possible solutions, one modifies the Concurrent Game Model of ATL, while the other adds the notion of strategic contexts. Part of why problems arise from the agent's strategies is that in ATL the notion of strategy is not explicit. The next section explores this issue further.

4.3.2 Explicit Strategies

Strategies are not explicit entities in ATL, but instead are indirectly included with $\langle\langle A\rangle\rangle$. The concept of a strategy is fundamental to ATL though, as it is the path quantifier. In most situations which are represented in ATL we are interested in whether or not some property of the system can be satisfied, which is the core

purpose of model checking [2]. In ATL this is done through talking about the existence of a strategy of the system. Considering the importance of strategies in ATL, it therefore makes sense that one might extend ATL by adding strategies as explicit parts. Many of the extensions to ATL do exactly this. The four extensions considered by Bulling et al. (2015) will be presented here:

1. **Counterfactual ATL (CATL)**, proposed by van der Hoek et al. (2005), extends ATL by adding counterfactual operators of the form $C_i(\sigma, \varphi)$ where the agent i has a strategy σ which will make a formula φ true.

2. **ATL with Intentions (ATLI)**, proposed by Jamroga et al. (2005), is very similar to CATL, but uses operators of the form $(str_i\sigma)\varphi$ where the agent i has an intention σ which will make a formula φ true if acted upon. Intentions are similar to strategies, but without any commitment and which can be changed. Another richer variant of this is ATLP (ATL with Plausibility), proposed by Bulling et al. (2008).

3. **ATL with Explicit Strategies (ATLES)**, proposed by Walther et al. (2007), is a revised version of CATL which uses a partial function of the form $\rho = \{a_1 \mapsto \sigma_1, ..., a_k \mapsto \sigma_k\}$ where each agent a of k many agents, has a strategy σ which it must play. The focus here is on the coalition having a strategy to achieve a certain outcome if each agent in the coalition commits to its part of the strategy.

4. **ATEL with Actions (ATEL-A)**, proposed by Ågotnes (2006), is an extension of ATEL which is an epsitemic extension of ATL considered in the next section. ATEL-A allows reasoning about both agents' knowledge and their strategies.

As we can see, making a strategy explicit often also addresses the first issue of strategic commitment. We also saw that there are many different approaches to making strategies explicit in ATL. CATL and ATLI add operators, while ATLES adds a function. CATL and ATLES involve commitment to strategies while the intentions of ATLI are more flexible, allowing agents to change. ATEL-A not only makes strategies explicit but also reasons about the often incomplete knowledge that agents have. The next section will consider more extensions which also involve agent knowledge.

[2]Model checking is described by Emerson (2008) as follows: "The model checking problem is an instance of the verification problem. Model checking provides an automated method for verifying concurrent (nominally) finite state systems that uses an efficient and flexible graph search, to determine whether or not the ongoing behavior described by a temporal property holds of the system's state graph. The method is algorithmic and often efficient because the system is finite state, despite reasoning about infinite behavior."

4.3.3 Incomplete Information

An unstated assumption in ATL/ATL* is that agents have complete information on the entire structure of the game and the current state, and with memory-based strategies it is further assumed that agents have perfect recall of all the past states. One might want to model a system where this is not the case, where players lack certain information or have an imperfect recall of past states.

A simple extension to do this, is to add *indistinguishability* relations to ATL, which is written as $q \sim_a q'$ to show that agent a is unable to distinguish between state q and state q'. This allows us to have an agent be unable to distinguish between a whole set of different states, thus following the same strategy for any of those states, unaware that something has changed. For memory-based strategies, if the previous couple of states were all indistinguishable to the agent, the agent would still be attempting the same strategy it was a few turns ago, unaware of perhaps major changes taking place in the system. The agent's knowledge or awareness of certain propositions can also be indirectly modelled. It can be said that an agent knows a property p is true in the current state q if p is also true in all states indistinguishable from q for that agent. Likewise if a proposition p changes between true and false in a set of indistinguishable states, then the agent must not be aware of that property, else it would be able to use that proposition to distinguish between the states.

An extension of ATL which uses indistinguishability relations, is Alternating-time Temporal Epistemic Logic (ATEL), introduced by van der Hoek and Wooldridge (2003) as a combination of ATL and Epistemic Logic, in order to formalise the reasoning about interactions between agents' knowledge and agents' strategies.

ATEL is built on the basic relation $K_a\varphi$ which means that agent a knows that φ is true. Using indistinguishability relations, the semantics for this can be defined as follows:

$$M, q \vDash K_a\varphi \text{ iff } M, q' \vDash \varphi \text{ for all } q' \text{ such that } q \sim_a q'$$

In addition we can also have various relations for coalitions of agents. We have mutual knowledge $E_A\varphi$ meaning all agents in coalition A know that φ is true. We have common knowledge $C_A\varphi$ meaning all agents in coalition A know that φ is true and they all know that they all know it. We also have distributed knowledge $D_A\varphi$ meaning that if all the agents in coalition A were to share their knowledge they would be able to figure out that φ is true. For more on epistemic logic refer to Fagin et al. (1995).

ATEL combines epistemic logic with ATL by using a Concurrent Epistemic Game Model (CEGM) which is simply a Concurrent Game Model used in ATL with the indistinguishability relations added for each agent.

ATEL has some problems however, one of which is pointed out by Brihaye et al. (2009), which is that an agent having the ability to achieve some state should also imply that it has the knowledge to identify a strategy to achieve that state, yet this is not expressible in ATEL. Some solutions have been proposed relying on uniform strategies (strategies specifying the same actions in indistinguishable states). For more on ATEL refer to Jamroga (2003).

For more extensions of ATL using ideas of incomplete information, refer to van der Hoek and Wooldridge (2003), Jamroga and van der Hoek (2004), Jamroga and Ågotnes (2007), and Schobbens (2004). For a detailed comparison and analysis of various such logics, refer to Bulling and Jamroga (2014).

4.4 Conclusion

ATL is a powerful logic, and has proven very useful, spawning many extensions and a large literature surrounding it. The model-theoretic approach is elegant and ATL formulæ are easy to read. We can reason about multiple agents carrying out various different actions, and transitions to new states based on their actions. The only downside of ATL is that all agents must act at the same time, and their actions and the transitions resolve instantly. There is no notion of gradual change. There is no way for agents to interrupt each others' actions. There is no way for actions to overlap, or for two actions to have two separate effects when carried out separately, but a third synergistic effect when carried out at the same time. To do any of this, we must turn from instants to intervals.

Chapter 5

The Logic of Intervals

ATL uses the idea of states, where a formula is true or false at a specific state. States are essentially instants in time, a snapshot of a moment. The transitions from one state to another are also instantaneous. There are many situations where this is inadequate to represent the true nature of what is happening, like overlapping actions, or events taking various amounts of times to finish. With intervals, events happen inside time periods of various lengths, which might or might not overlap. An example might be a football game, where a player might take 3.2 seconds to get to the ball, 0.3 seconds to kick it, and then 1.45 seconds might pass before it reaches the goal. In this time other players are running around and changing direction at different times. Halpern and Shoham (1991) point to examples where "'the liquid level increased by three inches','the robot carried out the task' and 'I solved the problem while jogging to the ocean and back' may be true at certain intervals but at no time instant".

This chapter will focus on the idea of intervals as opposed to instants, and discuss various logics which use intervals. We will start by considering some preliminary concepts which are essential to interval logics, then move on to discussing and comparing various interval logic formulations. We will then discuss a specific interval logic, introduced by Allen and Ferguson (1994). We end this chapter by applying Allen and Ferguson's logic to the three example problems from earlier.

5.1 Preliminaries

In this section we consider two core concepts, intervals and relations. Intervals are the primary elements of an interval logic, replacing states. Relations exist between intervals, and indicate their positions relative to each other. Where states are simply in a sequence one after another, intervals can overlap in all kinds of way. Two intervals might start at the same time but one end before the other, or one

might start as soon as the other has ended, and so on.

5.1.1 Intervals

Allen and Ferguson (1994) (p.8) defines a time period (interval) intuitively as the "time associated with some event occurring or some property holding in the world". An interval thus has a start and an end, and therefore also a duration, which may or may not be explicitly expressed. Intervals can be defined in terms of their start and end points, as in Goranko et al. (2004), Moszkowski's ITL and most non-pure interval logics, or intervals can be defined in terms of other intervals, such as in Allen and Hayes (1985) and Allen and Ferguson (1994).

Propositions are true or false at different intervals, representing how things change over time. There are many ways to do this. For a propositional logic, such as that defined by Halpern and Shoham (1991), one might use a valuation function assigning to each set of intervals a set of atomic propositions which are true at that interval. For first-order logics, the simplest solution is to add an extra argument to each predicate, as is done in Allen and Ferguson (1994), which will be discussed later in this chapter.

There are two important principles to take note of regarding truth and intervals:

- Locality Principle - An atomic proposition is true at an interval if and only if it is true at the beginning point of that interval, even if it becomes false later during the interval.

- Homogeneity Principle - The truth of a formula at an interval implies the truth of that formula at every sub interval of it, thus it has to remain true throughout the interval.

An interval-based logic will usually follow either the locality or the homogeneity principle.

Intervals can be seen as events, since events have a duration during which something is changing. Events are things that seem to happen in the world when some change occurs. Allen and Ferguson (1994) point out that events are primarily linguistic or cognitive constructs, and do not really exist in the world. They claim that: "Rather, events are the way by which agents classify certain useful and relevant patterns of change" (Allen and Ferguson, 1994, p. 3). Something can happen in the world, leading to some change, and observers can look at the change and describe it in different ways. This is all one event, even if it is described differently from different perspectives and using different vocabulary. The key idea is that some change occurs in the world, and this change is the event.

Change is almost always gradual. What looks like an instant event to the observer is really just a change over a short time period. Events therefore have to

have duration. Some events, like building a cathedral, can take a hundred years, while other events, like a firework exploding, can happen in milliseconds. Events therefore are some change occurring in the world over some interval of time.

In logic, events only come up when representing time-based knowledge. A temporal logic will have some current state, usually based on a set of atomic propositions, and over time the state will change. Whatever is causing the state to change, is the event. A logic like ATL has the concept of events built into the transition function, which changes the state of the system at the end of each turn. Interval logics have to have a more explicit notion of events, since things do not automatically change at the end of each turn, as there are no turns. Instead events are defined in various ways which can allow atomic propositions to be changed over intervals. This way, a long time can pass where nothing happens, then a lot can happen all at once, then a long silence again, or different combinations of these. Explicitly defining events allows for those events to happen at any place, or not to happen. Events can happen at the same time, overlap in various ways or interact to have additional synergistic effects. We will now consider some ways in which intervals (event) can overlap and be related to each other.

5.1.2 Relations

Two points in time a and b can have one of three possible relations with each other: either a is before b, or b is before a, or they are the same point $a = b$. Intervals can have many more possible relations with each other, overlapping in various ways. Additionally, when defining an interval, a non-pure interval can simply be defined by its end-points, such as the interval "Seminar" being defined as beginning at point 13:00 and ending at point 14:30. Pure intervals (which do not consist of a set of points) are a bit trickier. Since they are not tied to any points or an external clock, pure intervals need to be defined in terms of each other, or their relations with each other. To do this, we need to consider the relations which might hold between various intervals.

Allen and Hayes (1985) distinguish 6 possible relations between any two different intervals, namely *Before*, *Meets*, *Overlaps*, *Starts*, *Finishes*, and *During*. Each of these relations also have an inverse relation, namely *After*, *MetBy*, *OverlappedBy*, *StartedBy*, *FinishedBy* and *Contains*. Lastly there is also the symmetrical *Equality* relation, bringing us to a total of 13 relations. Figure 5.1 gives a graphical representation of these relations.

A very important relation is the *Meets* relation, defined in Allen and Ferguson (1994) as:

Two periods i and j meet if and only if i precedes j, yet there is no time between i and j, and i and j do not overlap.

All of the other relations can also be defined in terms of the *Meets* relation (:)

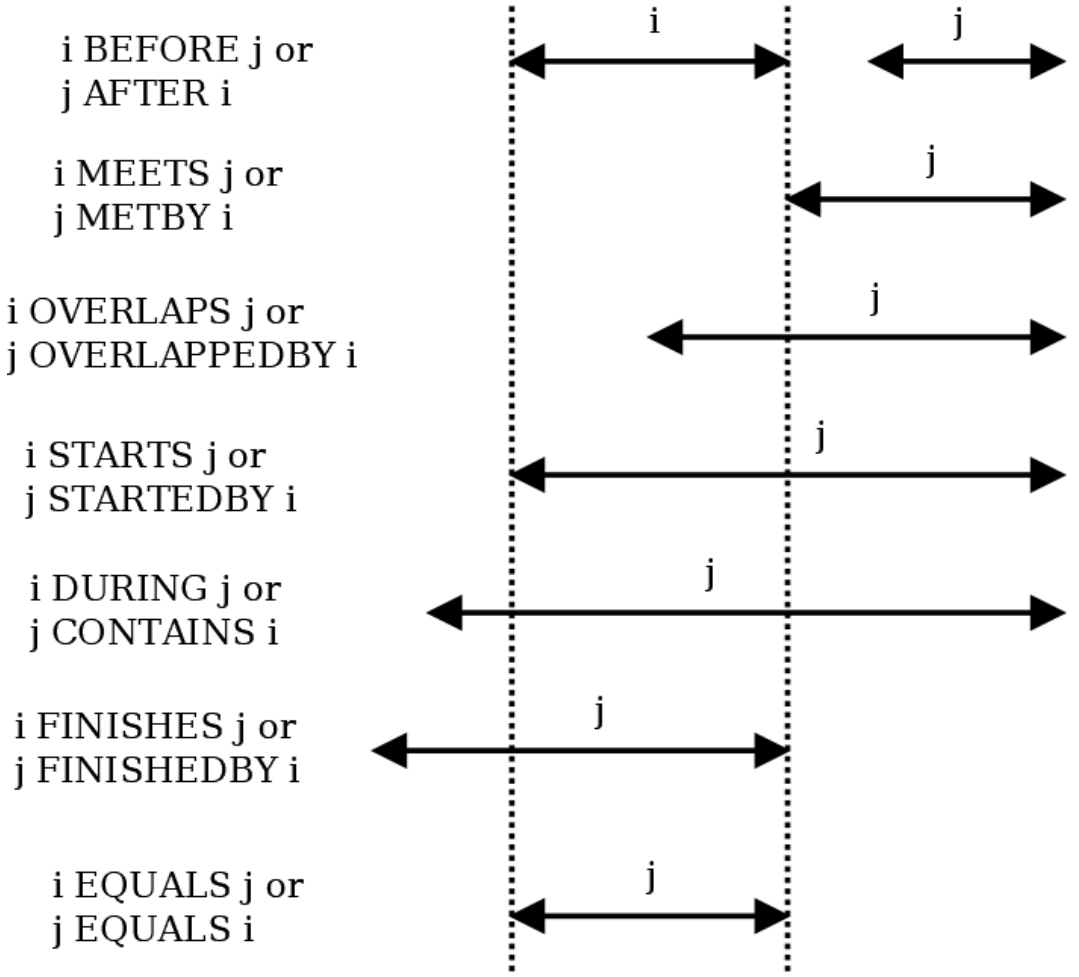


Figure 5.1: The 13 Interval Relations as Defined in Allen and Hayes (1985)

as follows:
Before(i,j): ∃ k. i:k:j
Overlaps(i,j): ∃ a,b,c,d,e. a:i:d:c ∧ a:b:j:c ∧ b:c:d
Starts(i,j): ∃ a,b,c. a:i:b:c ∧ a:j:c
During(i,j): ∃ a,b,c. a:b:i:c ∧ a:j:c
Finishes(i,j): ∃ a,b,c,d. a:b:i:c:d ∧ a:j:d

The inverses are similarly defined.

Having covered the notions of intervals and relations, let us look at the various ways they have been approached.

5.2 Interval Logics

This section will analyse and compare different formulations of interval logics.

5.2.1 Propositional Modal Logic of Time Intervals

Halpern and Shoham (1991) have introduced a logic of time intervals which extends point-based modal temporal logic with a simple syntax and semantics. Instead of a formula being satisfied by a state, it is here satisfied by an interval, where an interval is a partially ordered set of states. Following the convention from Goranko et al. (2004), I will refer to this logic as HS.

Intervals are not primitive objects in HS, but rather are viewed as being constructed out of points (specifically states), thus HS is a non-pure interval logic [1]. Recall that in a pure interval logic intervals are primary objects, while in a non-pure interval logic intervals are made up of other objects, like points. In HS, an interval is defined as an ordered pair $\langle s, t \rangle$ consisting of a start and end point s and t.

All the operators in HS are unary, and have the implicit notion of a current interval, similar to how many point based logics have a notion of a current state. A unary operator together with an interval shows that interval's relationship to the current interval. As mentioned before, two intervals can have any of 13 different relations with each other. Halpern and Shoham were able to express all these relations using only six modal operators: B (begin), E (end), A (after), $\overline{B}$ (begun by), $\overline{E}$ (ended by), and $\overline{A}$ (before). Venema (1990) later showed that all the relations can be expressed using only B, E, $\overline{B}$, and $\overline{E}$.

A notable feature of HS is that it remains uncommitted to specific temporal structures or connections between truth values and intervals. The logic only assumes that a set of points between any two points is totally ordered, thus allowing branching or linear, bounded or unbounded and dense or discrete time structures (refer to section 2.2.4). It also allows either homogeneity or locality for the relationship between truth values and intervals.

The following syntax of HS is from Halpern and Shoham (1991):

Given a set of atomic propositions Φ_0, a well formed formula φ is as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle A \rangle \varphi \mid \langle B \rangle \varphi \mid \langle E \rangle \varphi \mid \langle \overline{A} \rangle \varphi \mid \langle \overline{B} \rangle \varphi \mid \langle \overline{E} \rangle \varphi$$

For the semantics, Halpern and Shoham (1991) define an *interpretation* as a pair $\langle S, V \rangle$. S is a temporal structure $\langle T, \leq \rangle$ where T is a set of time points and

[1] I admit that there can be some debate here. I take the position that HS is a non-pure interval logic since the intervals consist of two points. But one might also take the position that the formulæ of HS are evaluated over interval objects, thus it should be a pure interval logic.

$\leq$ is a partial order on T. V is a function which maps the primitive propositions to the set of intervals where that proposition is true. Formulæ are interpreted over pairs $\langle t_1 t_2 \rangle$ such that $t_1, t_2 \in T$ and $t_1 \leq t_2$, thus formulæ are interpreted over intervals defined by their endpoints t_1 and t_2. Given an interpretation M and an interval $\langle t_1 t_2 \rangle$, a formula φ is either true or false in that interval, written as $\langle t_1 t_2 \rangle \vDash \varphi$ or $\langle t_1 t_2 \rangle \nvDash \varphi$.

The evaluation of a formula φ over an interval $\langle t_1 t_2 \rangle$, written as $\langle t_1 t_2 \rangle \vDash \varphi$, is defined as follows:

- $\langle t_1 t_2 \rangle \vDash p$, iff $\langle t_1 t_2 \rangle \in V(p)$.
- $\langle t_1 t_2 \rangle \vDash \neg\varphi$, iff $\langle t_1 t_2 \rangle \nvDash \varphi$.
- $\langle t_1 t_2 \rangle \vDash \varphi_1 \wedge \varphi_2$, iff $\langle t_1 t_2 \rangle \vDash \varphi_1$ and $\langle t_1 t_2 \rangle \vDash \varphi_2$.
- $\langle t_1 t_2 \rangle \vDash \langle A \rangle \varphi$, iff there exists t_3 such that $t_2 < t_3$ and $\langle t_2 t_3 \rangle \vDash \varphi$.
- $\langle t_1 t_2 \rangle \vDash \langle B \rangle \varphi$, iff there exists t_3 such that $t_1 \leq t_3$, $t_3 < t_2$ and $\langle t_1 t_3 \rangle \vDash \varphi$.
- $\langle t_1 t_2 \rangle \vDash \langle E \rangle \varphi$, iff there exists t_3 such that $t_1 \leq t_3$, $t_3 < t_2$ and $\langle t_3 t_2 \rangle \vDash \varphi$.
- $\langle t_1 t_2 \rangle \vDash \langle \overline{A} \rangle \varphi$, iff there exists t_3 such that $t_3 < t_1$ and $\langle t_3 t_1 \rangle \vDash \varphi$.
- $\langle t_1 t_2 \rangle \vDash \langle \overline{B} \rangle \varphi$, iff there exists t_3 such that $t_2 < t_3$ and $\langle t_1 t_3 \rangle \vDash \varphi$.
- $\langle t_1 t_2 \rangle \vDash \langle \overline{E} \rangle \varphi$, iff there exists t_3 such that $t_3 < t_1$ and $\langle t_3 t_2 \rangle \vDash \varphi$.

Halpern and Shoham (1991) showed that for all but the simplest classes of temporal structures, HS is undecidable. Venema (1990) showed that HS is more expressive than any temporal logic based on points when interpreted over linear orderings. Venema's paper also contains a geometrical interpretation of HS which results in a sound and complete axiomatic system. Goranko et al. (2004) claims that HS is the "most expressive propositional interval logic with unary modal operators".

The logic HS is very simple and natural, while being very expressive. It is able to capture all thirteen of Allen's interval relations while using only a few unary operators, six in the original formulation and four in Venema's version. It does however still define intervals as sets of two points, thus making points and not intervals the primary objects in the logic, making it a non-pure interval logic. The underlying structure of time in HS is still partially ordered sets of points, tied together by various intervals.

5.2.2 Interval Temporal Logic

While many temporal logics which deal with intervals are referred to as interval temporal logics, there is one logic which claims the name *Interval Temporal Logic* (ITL). This is why this chapter is called *The Logic of Intervals*, rather than Interval Temporal Logic, to avoid confusion. ITL was described by Moszkowski (1983a) in a PhD thesis and in a paper by Halpern et al. (1983). ITL is described in the thesis as a formalism which "augments standard predicate logic with time-dependent operators".

More than just another logic, ITL aims to also be a programming language. Moszkowski (1983b) showned how ITL can be used to represent various programming language control structures, such as variable assignment, iteration (loops), and sequential and parallel computations. Moszkowski also compared ITL with programming languages Lucid and Prolog in his paper. Moszkowski (1984) also makes the argument that it is redundant to use temporal logic such as ITL to specify and prove properties about a program while writing the actual program in a different programming language. Instead he suggests programming directly in ITL, and presents an interpreter for ITL called Tempura, thus expanding ITL to be a full programming language in addition to being a logic, able to both specify and prove properties about systems, while also implementing the systems at the same time. For more on Tempura, refer to Moszkowski (1996). Since the original formulation, Moszkowski and others have been adding to and improving ITL and the interpreter Tempura, making it a very well known and often used temporal logic.

The key notion in ITL is that of an interval, defined as a finite sequence of states, making it a non-pure interval logic. The length of an interval is equal to one less than the number of states in the interval. Intervals can be joined together to form larger intervals and decomposed into smaller intervals. Instead of having explicit state transitions, changes in variables [2] are represented with initial and final values for an interval. ITL is also a first order language, and as such uses an interpretation to map to a domain, rather than just having a set of propositions being true at some interval. The latest version of ITL can be found at http://www.antonio-cau.co.uk/ITL/.

In ITL there are two types of formulæ, one called *expressions* and the other called *formulæ*. The Syntax of ITL, from the original paper by Halpern et al. (1983), is given here:

[2] In state transition systems one can show a chagne in a variable through a change in state. For example, a door might be closed in one state and open in another, where the state transition implies opening the door. In an interval system variables are more complex, and can change at various points in an interval.

A well formed expression e is as follows:
$e ::= V \mid f(e_1, ..., e_k)$
where V is a variable, f is a function, and e_1 to e_k are expressions with $k \geq 0$.

A well formed formula f is as follows:
$f ::= e \mid p(e_1, ..., e_k) \mid e_1 = e_2 \mid \neg f \mid f_1 \wedge f_2 \mid \forall V.f \mid \bigcirc f \mid f_1; f_2$
where p is a predicate, e_1, e_2 and e_k are expressions, V is a variable, $\forall$ is the universal quantifier and $\bigcirc$ and ; are operators.

ITL also has a set of states $\Sigma = s, t, ...$ and a domain D together with an interpretation M mapping each variable V and interval $s_0...s_n$ to some value $M_{s_0...s_n}[[V]]$ in D.

For the semantics, also from Halpern et al. (1983), each function and predicate symbol is given some meaning. A function f has an interpretation $M[[f]]$ which maps k elements in D to a single value:

$$M[[f]] \in (D^k \rightarrow D)$$

Likewise predicates p have similar interpretations, but map to true or false:

$$M[[p]] \in (D^k \rightarrow \{true, false\})$$

The semantics for a formula is as follows:

- $M_{s_0...s_n}[[p(e_1, ..., e_k)]] = M[[p]](M_{s_0...s_n}[[e_1]], ..., M_{s_0...s_n}[[e_k]]$.
- $M_{s_0...s_n}[[e_1 = e_2]] =$ true, iff $M_{s_0...s_n}[[e_1]] = M_{s_0...s_n}[[e_2]]$.
- $M_{s_0...s_n}[[\neg f]] =$ true, iff $M_{s_0...s_n}[[f]] =$ false.
- $M_{s_0...s_n}[[f_1 \wedge f_2]] =$ true, iff $M_{s_0...s_n}[[f_1]] =$ true and $M_{s_0...s_n}[[f_2]] =$ true.
- $M_{s_0...s_n}[[\forall V.f]] =$ true, iff $M'_{s_0...s_n}[[f]] =$ true, for every interpretation M' that agrees with M on the assignments to all variables, function and predicate symbols except possibly the variable V.
- $M_{s_0...s_n}[[\bigcirc f]] =$ true, iff $n \geq 1$ and $M_{s_1...s_n}[[f]] =$ true.
- $M_{s_0...s_n}[[f_1; f_2]] =$ true, iff $M_{s_0...s_i}[[f_1]] =$ true and $M_{s_i...s_n}[[f_2]] =$ true for some $0 \leq i \leq n$.

A extension of ITL can be found in Duan (1996) which includes infinite models and infinite intervals, past operations such as 'previous' and 'past chop' and the

projection operator for dealing with concurrent computation, synchronous communication and framing in the context of temporal logic programming. Another variant of ITL is what Goranko refers to as ITL_D in Goranko et al. (2004), which is Dutertre's study of the fragment of ITL using only the *chop* operator (Dutertre, 1995), resulting in a sound and complete but undecidable logic.

Many years of work have gone into the ITL software Tempura. Looking at the syntax and semantics, it is clear that ITL is not a simple nor natural logic. One reason for this is that it is a first order logic, another is that it is meant to be a programming language in addition to a logic. In order to represent all of Allen's thirteen relations, the user would have to specify each relation as a predicate, in addition to all the syntax already present. But we do see here an instance of two types of formulæ, something which we will use for our own combined logic in a later chapter.

5.2.3 Neighbourhood Logic

The problem with a logic such as Moszkowski's ITL, is that it has what are called *contracting modalities*, which means one cannot express more abstract properties about intervals outside the current interval such as "eventually ϕ will hold". Expressing these kinds of properties are very important to proving liveness and fairness in the logic, since those properties involve multiple intervals outside the current interval. Chaochen and Hansen (1997) introduced a first order logic called Neighbourhood Logic (NL), which is based on the idea of *left* and *right* modalities, which refer to the *before* and *after* intervals. These modalities are represented by $\diamond_l$ and $\diamond_r$ respectively.

NL is an extension of ITL, adding only these two modalities to ITL. As such the syntax for a well formed formula is similar to ITL, with the two new modalities included:

A well formed formula f is as follows:

$$f ::= e | p(e_1, ..., e_k) \mid e_1 = e_2 \mid \neg f \mid f_1 \wedge f_2 \mid \forall V.f \mid \bigcirc f \mid f_1; f_2 \mid \diamond_l e \mid \diamond_r e$$

The semantics of NL is exactly the same as in ITL, except for the two new modalities, where the semantics are defined as follows:

- $M_{s_0 \ldots s_i}[[\diamond_r e]] =$ true, iff there exists s_n such that $s_i \leq s_n$ and $M_{s_i \ldots s_n}[[e]] =$ true.
- $M_{s_0 \ldots s_i}[[\diamond_l e]] =$ true, iff there exists s_n such that $s_n \leq s_i$ and $M_{s_n \ldots s_0}[[e]] =$ true.

The addition of these two modalities makes NL a more expressive language, able to express any of the thirteen Allen relations, and "can virtually express all interesting properties of the underlying linear orderings, such as discreteness, density, etc" Goranko et al. (2004).

For a completeness proof of NL, refer to Barua et al. (2000). For a discussion of NL as it relates to Allen's Interval Algebra, refer to Pujari (1997). For a propositional NL variant, refer to Goranko et al. (2003). Refer to Barua and Chaochen (1997) for an extension of NL which adds two more modalities for up and down, alongside the left and right of NL, allowing for a two dimensional NL.

Neighbourhood Logic extends ITL, and is able to express the thirteen interval relations. It is a non-pure interval logic, similar to ITL.

5.2.4 Duration Calculus

Another extension of ITL is Duration Calculus (DC), introduced by Chaochen et al. (1991). DC attempts to reason about time without explicitly mentioning absolute time, instead using the integrals of durations of states. A unique feature of DC is the *state expression*, which represents a state and has a duration for how long the system is in that state. A *state expression* is thus similar to an interval.

Duration Calculus will not be presented in any depth, since being a logic of intergrals of durations, it is far from the pure, non-duration intervals which we wish to investigate. It is mentioned here simply because of its close relation to the previous two logics.

The interested reader is referred to Guelev (1998) for a completeness proof of DC. See also Roy and Chaochen (1997), where DC has been combined with Neighbourhood Logic to create an undecidable logic called DC/NL. For even more refer to Chaochen et al. (1993), Chaochen (1999), and Hansen and Chaochen (1992).

We have considered a few non-pure logics, which keep the notion of states and add intervals as secondary elements over those states. But a particular focus of this dissertation is on pure interval logics, and unfortunately those are very rare. A well known pure interval logic which is different from all of these others so far considered, is Allen and Ferguson's logic, which will be discussed in the next section.

5.3 Allen's Logic of Actions and Events

Allen and Ferguson introduced a logic of actions and events in interval logic. The first line of the paper reads: "We present a representation of events and action based on interval temporal logic that is significantly more expressive and more

natural than most previous AI approaches" Allen and Ferguson (1994). Their use of the term *interval temporal logic* seems to be a general term referring to logics which use intervals. This can be deceiving, since their logic is not actually a temporal logic, nor even a modal logic, but rather a type of first order logic. Allen and Ferguson's logic is a domain specific first order logic with a specific interpretation of the *Meets* predicate, around which it is built. It is however, one of the very few interval logics to use intervals as primary elements, and not contain any states.

We mentioned in the background chapter that some languages can be defined with a set of axioms and rules for deducing true formulas from those axioms, such as propositional and first order logic, this is called the axiomatic approach, the proof-theoretic approach or a deductive system. The alternative we have seen so far is the model-theoretic approach of modal and temporal logic, where formulas are evaluated semantically based on the states of the model. While we saw the clean and elegant formulas of ATL in the previous chapter, we will now see how much more cumbersome the axiomatic route can be.

It starts with a single primitive object, the *Interval* (which is referred to as a *time period*), and one primitive relation, *Meets*. Allen's concept of an interval has already been discussed, as was the relation meets, and how the 13 possible relations between intervals can be defined in terms of the meets relation. The meets relation is represented by a predicate $Meets(j, i)$ where j and i are intervals such that j meets i.

Allen defines some axioms for the meets relation, the formal details of which can be found in Allen and Ferguson (1994), but can be summarised as follows:

- There is no beginning or ending of time and there are no semi-infinite or infinite intervals, thus every interval has an interval that meets it and another that it meets. Thus $\forall i.\exists j, k.Meets(j, i) \wedge Meets(i, k)$.

- Intervals can be concatenated into a larger interval, when two intervals meet, there is a third interval that contains them both precisely (which begins at the beginning of the first interval and ends at the end of the second interval). Thus $\forall i, j, k, l.Meets(i, j) \wedge Meets(j, k) \wedge Meets(k, l) \rightarrow \exists m.Meets(i, m) \wedge Meets(m, l)$.

- Intervals uniquely define an equivalence class of periods that meet them, in other words if an interval meets a pair of two different intervals, any other interval that meets one of those pair must also meet the other. Thus $\forall i, j, k, l.Meets(i, j) \wedge Meets(i, k) \wedge Meets(l, j) \rightarrow Meets(l, k)$.

- These equivalence classes also uniquely define the intervals, thus if two intervals both meet the same interval, and another interval meets both of them,

the two intervals are equivalent. Thus $\forall i, j, k, l.Meets(k, i) \wedge Meets(k, j) \wedge Meets(i, l) \wedge Meets(j, l) \rightarrow i = j$.

- Lastly, an ordering axiom: for any two pairs of intervals that meet each other, either the two pairs meet at the same place, or the one pair meets each other either before or after the other pair meets each other. Thus ($\oplus$ is the xor operator) $\forall i, j, k, l.Meets(i, j) \wedge Meets(k, l) \rightarrow Meets(i, l) \oplus (\exists m.Meets(k, m) \wedge Meets(m, j)) \oplus (\exists m.Meets(i, m) \vee Meets(m, l))$.

As a first order logic, Allen and Ferguson's logic uses predicates to build its formulæ such as the formula *Green(frog)* meaning that the frog is green. They extend this by adding a second argument, consisting of an interval, such as *Green(frog,t3)* which lets us know that the frog is only green during interval t3. The logic is homogeneous, that is, if a proposition is true for some interval, it is also true for all sub intervals of it. Allen distinguishes between strong negation and weak negation, where strong negation says that if a formula is false for some interval, it is also false in all sub intervals of that interval, while weak negation simply means that the proposition is not true for all sub intervals of a specific interval. We can see that a homogeneous truth structure allows for weak negation, thus if a formula is true for all sub intervals of interval t it is true for t, but if it is not true for all sub intervals of t it is false for t, which doesn't mean it is not true somewhere in t.

The notion of an event occurring is represented by a predicate, with details about the event in the arguments. As an example from Allen and Ferguson (1994), the event where Jack lifts the ball can be written as:
$LIFT(jack34, ball26, t1)$
where $t1$ is the interval during which the lifting takes place.

There may of course always be more details about the event one might wish to include, and continuing to add arguments to predicates is not an ideal solution, since each argument in a predicate has to be defined. Our lift example has three arguments, the one carrying out the event, the recipient of the event, and the time during which the event takes place. If we want to say that Jack lifts the ball onto the table, we might add as a fourth argument the position to which the second argument is lifted, but then we need to have an empty argument when we simply want to speak about him lifting it again. This becomes impractical when we want to describe an event in great detail, using perhaps 20 arguments.

Allen instead uses an idea from Davidson (1967) involving reifying events, where the event is an argument which can then have additional predicates if we wish to have additional information for a specific event. As another example from Allen and Ferguson (1994), we might have:

$$\exists e.LIFT(jack34, ball26, t1, e) \wedge dest(e) = table5 \wedge instrument(e) = tongs1$$

Thus the main event predicate lift has the same three arguments as before, with a fourth argument e representing the event. We can then specify that the destination *dest* of e is $table5$ and the tool *instrument* of e is $tongs1$. This tells us that Jack is lifting the ball onto the table using the tongs. This is a much more elegant way for representing additional information about an event.

We have three functions: pre, con, and eff, which represent the prerequisites, conditions and effects of an event respectively. Prerequisites need to hold before an event occurs, conditions hold during the event taking place, and effects hold after the event has taken place. Using these, we can define the conditions for an event to take place. As another example from Allen and Ferguson (1994), consider a robot placing box x on box y. Both boxes need to be clear beforehand (nothing on them), the robot will hold the box x while the event takes place, and the effect is that x is on y. This is represented by:

$$\forall x, y, t, e.STACK(x, y, t, e) \rightarrow$$
$$Clear(x, pre1(e)) \wedge Holding(x, con1(e)) \wedge Clear(x, eff1(e)) \wedge Clear(y, pre2(e)) \wedge$$
$$On(x, y, eff2(e))$$

Adding to event definitions, action definitions are also needed. Allen draws on Goldman (1970) and calls these *basic actions*. This is done with the $Try(\pi, t)$ predicate, where π is attempted during time t. A successful attempt (prerequisites considered) will cause an event to happen.

On top of the usual syntax of first order logic, Allen and Ferguson add the following three specific predicates to be used for actions and events:

Event definitions: $\forall e.E(e) \wedge \phi \rightarrow \psi$ where E is an event predicate and ϕ and ψ are conditional predicates.

Action definitions: $\forall t, ...Try(\pi, t) \wedge \phi \rightarrow \exists e.\exists E(e) \wedge t \circ time(e) \wedge \psi$ where ϕ is the prerequisites for the try to succeed, leading to the event E taking place together with the effect predicates ψ. The $\circ$ symbol represents a temporal relationship between the t interval in which the attempt takes place, and the $time(e)$ function which happens as a result of that attempt.

Event generation axioms: $\forall e.E(e) \wedge \phi \rightarrow \exists e'.E'(e') \wedge \psi$ where E and E' are event type predicates while ϕ and ψ represents constraints on the events.

There is one more important concept tying together Allen and Ferguson's approach which must be considered next, called *explanation closure axioms*.

5.3.1 Explanation Closure Axioms

In a state based logic like ATL, transition from one state to the next is quite simple. A single function maps the current state and the choices of the agents to a new state. Intervals complicate matters, it is not always clear which results must flow from which events, and which events from which actions. A logic like *situation calculus* McCarthy and Hayes (1981) originally used *frame axioms*, which involved explicitly stating which propositions did not change as a result of some event. The problem with this has already been discussed in the introduction chapter, namely the *frame problem*, where there are too many propositions to consider for every possible action.

Allen and Ferguson use a different approach to this, where for every proposition that can change, the events which may change it are defined. This reversal of frame axioms is called *explanation closure* and was introduced by Haas (1987) and Shoham (1987). Allen and Ferguson (1994) mention a few advantages of this approach: "the resulting axioms can be interpreted with the standard semantics of the first-order predicate calculus, meaning that there are well-defined notions of entailment and proof. We can show that our representation handles a particular class of examples by showing a proof of the desired consequences, without needing to appeal to model-theoretic arguments in a non-standard semantics. In addition, various forms of uncertainty are handled using the standard methods in logic with disjunction and existential quantification. Finally, the assumptions that are made appear explicitly as axioms in the system".

This approach rests on two basic assumptions:

- No propositions change unless explicitly changed by an event occurring.
- No events take place except those explicitly caused by a successful action.

These two assumptions are defined formally in Allen and Ferguson (1994) as follows (where : represents the *meets* relation):

EXCP (Strong Closure on Properties) Every property (proposition) change results from the occurrence of an instance of one of the event types defined in the axioms. These axioms are of the form:

$$\forall t, t'.P(t) \wedge \neg P(t') \wedge t : t' \rightarrow \exists e.(E_1(e) \wedge E_2(e) \wedge ... \wedge E_n(e)) \wedge time(e) : t'$$

where the E_i are the event-type predicates that (possibly) affect the truth value of P. These axioms are derived from the event definition axioms (EDEF).

EXCE (Strong Closure on Events) Every event that occurs does so as a result of some action being attempted, possibly indirectly via event generation.

These axioms are of the form:

$$\forall e.E(e) \rightarrow \Phi_1 \wedge \Phi_2 \wedge ... \wedge \Phi_n$$

where Φ_i is either of the form:

$$\exists t.Try(\pi, t) \wedge t \circ time(e) \wedge \psi$$

or

$$\exists e'.E'(e') \wedge time(e) \circ time(e') \wedge \psi$$

These axioms can be derived from the action and event generation axioms (ETRY and EGEN).

Having discussed the important aspects of Allen and Ferguson's logic, it will now be applied to the example problems to further clarify how it might be used.

5.4 Problems - The Banker's Algorithm

Presented here is a problem introduced in the introduction chapter, and how it may be represented and reasoned about in Allen and Ferguson's interval logic.

Since there is not a concept of agents in Allen and Ferguson's interval logic, this problem will be approached from the perspective of the system as a whole. There are two types of action, *borrow* and *return*, which are represented here as causing two types of events, also called *BORROW* and *RETURN*. There are three agents, Alice, Bob and Charlie, though they are not explicitly represented in the logic. Events are named in all *UPPERCASE*, actions in all *lowercase*, and predicates in *TitleCase*. Similar to the ATL Banker's Algorithm, we use Boolean predicates to represent values, such as $NeedAlice(4)$ and $LoanBob(2)$, showing Alice still needs 4 units to complete her task and that Bob currently has 2 units. There are predicates for all possible values, and only one is true at a time. To not make the axioms overly complicated and unnecessarily long, assume setting one predicate to true, automatically sets all similar predicates to false. For example, if $LoanCharlie(4)$ is true, and then $LoanCharlie(5)$ is set to true, this automatically sets $LoanCharlie(4)$ to false. This is trivial to include in each axiom. We also have predicate values for the available *cash*. Remember that:

$$0 \leq Loan(p_i) \leq Need(p_i) \leq Cap$$

and

$$\forall i : 0 \leq i < N : Claim(p_i) \leq Cash + \sum_{j=0}^{i-1} Loan(p_j)$$

There are two event definition axioms for each agent, for a total of six. They are defined as follows:

$$\begin{aligned}\forall e,l,w,x,y,z.BORROWALICE(e,l,w,x,y,z) \rightarrow \\ Cash(w,pre1(e)) \wedge SameEnd(pre1(e),time(e)) \wedge (w >= l) \wedge \\ LoanAlice(x,pre2(e)) \wedge SameEnd(pre1(e),time(e)) \wedge \\ Cash(y,eff1(e)) \wedge time(e) : eff1(e) \wedge (y = w - l) \wedge \\ LoanAlice(z,eff2(e)) \wedge time(e) : eff2(e) \wedge (z = x + l) \quad \textbf{(EDEF1)}\end{aligned}$$

In other words, Alice attempts to take a loan of l amount, and has x already, while there is w available. The only requirement is that the amount available is larger than or equal to the loan. If successful, the available amount is now y, which is $y = w - l$, while Alice now has a loan of z, which is $z = x + l$. The event axioms **EDEF2** and **EDEF3** are similarly defined for predicates $BORROWBOB$ and $BORROWCHARLIE$.

$$\begin{aligned}\forall e,w,x,y,z.RETURNALICE(e,w,x,y,z) \rightarrow \\ LoanAlice(w,pre1(e)) \wedge SameEnd(pre1(e),time(e)) \wedge \\ NeedAlice(y,pre2(e)) \wedge SameEnd(pre2(e),time(e)) \wedge (w >= y) \\ Cash(x,pre3(e)) \wedge SameEnd(pre3(e),time(e)) \wedge \\ LoanAlice(0,eff1(e)) \wedge time(e) : eff1(e) \wedge \\ NeedAlice(0,eff2(e)) \wedge time(e) : eff2(e) \wedge \\ Cash(z,eff3(e)) \wedge time(e) : eff3(e) \wedge (z = x + w) \quad \textbf{(EDEF4)}\end{aligned}$$

Alice can attempt to complete her task and return the resources. She only succeeds when her borrowed amount w is more than or equal to her needed amount y. If she succeeds, her need and loan becomes 0, and the available cash goes up by her loan amount. The event axioms **EDEF5** and **EDEF6** are similarly defined for $RETURNBOB$ and $RETURNCHARLIE$

For each event axiom, there is an action axiom, showing the attempt that will cause the event to take place:

$$\begin{aligned}\forall t,x,l.Try(borrowalice,t) \wedge Cash(x,t) \wedge (x >= l) \\ \rightarrow \exists e.BORROWALICE(t,e) \quad \textbf{(ETRY1)}\end{aligned}$$

Where x is the amount of cash available and l is the size of the attempted loan. Note that Allen and Ferguson are not clear on how to explicitly represent numbers like x and l in this context, and some liberty has been taken. The action axioms

ETRY2 and **ETRY3** are similarly defined for Bob and Charlie to take loans.

$$\forall t, x, y.Try(returnalice, t) \wedge LoanAlice(x, t) \wedge NeedAlice(y, t) \wedge (x >= y) \rightarrow \exists e.RETURNALICE(t, e) \quad (\textbf{ETRY4})$$

The action axioms **ETRY5** and **ETRY6** are similarly defined for Bob and Charlie to return loans.

The last thing that is needed is a couple of explanation closure axioms.

$$\forall t, t', x, y.LoanAlice(x, t) \wedge LoanAlice(y, t') \wedge t : t' \wedge x < y \rightarrow \exists e.BORROWALICE(e) \wedge time(e) : t' \quad (\textbf{EXCP1})$$

If the loan for an agent went up, it must be because that agent has taken a loan. Defined similarly for *LoanBob* and *LoanCharlie* as **EXCP2** and **EXCP3**.

$$\forall t, t', x.LoanAlice(x, t) \wedge LoanAlice(0, t') \wedge t : t' \rightarrow \exists e.RETURNALICE(e) \wedge time(e) : t' \quad (\textbf{EXCP4})$$

If an agent's loan went down to zero, it must be because the agent returned the loan. Defined similarly for *LoanBob* and *LoanCharlie* as **EXCP5** and **EXCP6**.

$$\forall t, t', x, y.NeedAlice(x, t) \wedge NeedAlice(y, t') \wedge t : t' \wedge x > y \rightarrow \exists e.BORROWALICE(e) \wedge time(e) : t' \quad (\textbf{EXCP7})$$

$$\forall t, t', x.NeedAlice(x, t) \wedge NeedAlice(0, t') \wedge t : t' \rightarrow \exists e.(BORROWALICE(e) \vee RETURNALICE(e)) \wedge time(e) : t' \quad (\textbf{EXCP10})$$

Similar for NeedBob and NeedCharlie as **EXCP11** and **EXCP12**. **EXCP7** to **EXCP12** are similar to **EXCP1** to **EXCP6** but concerns the *need* predicate rather than the *loan* predicate. Some axioms then are needed for the *cash* predicate.

$$\forall t, t', x, y.Cash(x, t) \wedge Cash(y, t') \wedge t : t' \wedge x > y \rightarrow \exists e.(BORROWALICE(e) \vee BORROWBOB(e) \vee BORROWCHARLIE(e)) \wedge time(e) : t' \quad (\textbf{EXCP13})$$

$$\forall t, t', x, y. Cash(x,t) \wedge Cash(y,t') \wedge t : t' \wedge x < y \rightarrow \exists e.(RETURNALICE(e) \vee RETURNBOB(e) \vee RETURNCHARLIE(e)) \wedge time(e) : t' \quad (\textbf{EXCP14})$$

If the amount of cash went down, then an agent must have borrowed. If the amount of cash went up, then an agent must have returned their loan.

In total these 14 axioms are the *closure on properties* axioms, where for each proposition which can change, the events are defined which can change it. Remember that a proposition cannot change unless changed in this way by one of these events. Next are the closure on events properties, where for each event is defined the actions which can cause it.

$$\forall e. BORROWALICE(e) \rightarrow \exists t. Try(borrowalice, t) \wedge t = time(e) \quad (\textbf{EXCE1})$$

The successful action *borrowalice* results in the event *BORROWALICE*. Similar definitions for Bob and Charlie as **EXCE2** and **EXCE3**.

$$\forall e. RETURNALICE(e) \rightarrow \exists t. Try(returnalice, t) \wedge t = time(e) \quad (\textbf{EXCE4})$$

The successful action *returnalice* results in the event *RETURNALICE*. Similar definitions for Bob and Charlie as **EXCE5** and **EXCE6**.

With that, all the necessary general axioms have been defined for the banker algorithm. To create a more specific case, more axioms are added to describe it. Here is one example (where the $\prec$ symbol is the *before* relation):

$$t0 \prec t1 \prec t2 \prec t3 \prec t4 \prec t5 \prec t6 \prec t7 \quad (\textbf{AX0})$$

$$Cash(10, t0) \quad (\textbf{AX1})$$

$$LoanAlice(0, t0) \quad (\textbf{AX2})$$

$$LoanBob(0, t0) \quad (\textbf{AX3})$$

$$LoanCharlie(0, t0) \quad (\textbf{AX4})$$

$$NeedAlice(7, t0) \quad \textbf{(AX5)}$$

$$NeedBob(10, t0) \quad \textbf{(AX6)}$$

$$NeedCharlie(4, t0) \quad \textbf{(AX7)}$$

$$Try(borrowcharlie(4), t1) \quad \textbf{(AX8)}$$

$$Try(borrowalice(7), t2) \quad \textbf{(AX9)}$$

$$Try(returncharlie, t3) \quad \textbf{(AX10)}$$

$$Try(borrowbob(10), t4) \quad \textbf{(AX11)}$$

$$Try(returnbob, t5) \quad \textbf{(AX12)}$$

$$Try(borrowalice(7), t6) \quad \textbf{(AX13)}$$

$$Try(returnalice, t7) \quad \textbf{(AX14)}$$

To see if this specific case works out, it must be shown that:

$$NeedAlice(0, t8) \wedge NeedBob(0, t8) \wedge NeedCharlie(0, t8) \wedge t7 \prec t8$$

To prove this formula is true, each part of this conjunction can be proven in turn. As a demonstration the first part, $NeedAlice(0, t8)$ will be shown.

$NeedAlice(7, t0)$ is given by **AX5**. The only events which can change this value are BORROWALICE or RETURNALICE, by **EXCP7** and **EXCP10**. **AX9** gives an action attempt, which does not succeed. It fails since it is given that $cash(10, t0)$ by and thus the action in **AX8** $Try(borrowcharlie(4), t1)$ succeeds.

From **AX1**, **AX8**, **ETRY3** and **EDEF3** follows **AX15** $Cash(6, t8)$ where $t1 : t8$. Thus Charlie successfully borrows 4 units. From **AX9**, **AX15** and **ETRY1** follows **AX16** $Cash(6, t9)$ where $t2 : t9$. Alice tries to borrow 7 units but fails to since there is not enough cash available, $NeedAlice(7)$ thus stays unchanged. Note that if $t1 \prec t2$ and $t1 : t8$ then $t8$ either starts, overlaps or or is before $t2$.

Any value that is true during $t8$ and remains unchanged would then also be true during $t2$.

From **AX10**, **AX16**, **ETRY6** and **EDEF6** follows **AX17** $Cash(10, t10)$ where $t3 : t10$. From **AX11**, **AX17**, **ETRY2** and **EDEF2** follows **AX18** $Cash(0, t11)$ where $t4 : t11$. From **AX12**, **AX18**, **ETRY5** and **EDEF5** follows **AX19** $Cash(10, t12)$ where $t5 : t12$.

Finally Alice gets a second chance to attempt to borrow: From **AX13**, **AX19**, **ETRY1** and **EDEF1** follows **AX20** $Cash(3, t13)$ where $t6 : t13$. From **AX13**, **AX19**, **ETRY1** and **EDEF1** follows **AX21** $NeedAlice(0, t14)$ where $t6 : t14$. Having successfully borrowed the amount needed, $NeedAlice(0)$ is now finally true. The last axiom, **AX14**, does not change this value.

Thus it is proven that $NeedAlice(0, t8)$.

The deadlock state which must be avoided can be represented by the following formula:

$$\forall t, w, x, y, z. NeedAlice(x, t) \wedge NeedBob(y, t) \wedge NeedCharlie(z, t) \\ \wedge Cash(w, t) \wedge (w < x) \wedge (w < y) \wedge (w < z)$$

It can be seen that this representation is able to represent all that needs to be represented about the scenario. Compared to the ATL representation, it requires a lot more cumbersome writing. This representation can however represent scenarios which ATL cannot, such as overlapping attempts at borrowing and returning. Consider the above case but where $t1$, $t2$, $t3$ and $t4$ are not one after the other, but overlapped. Such a case can be reasoned about with interval logic but not with ATL. This is because states in ATL cannot overlap, but are discrete and sequential.

It can be seen that this representation is cumbersome to write out, and difficult to read, requiring many axioms to represent simple situations. This banker problem is enough to establish that and get an idea of how this logic works. To save space and to spare the reader, the sleeping barber and trains problems have been placed in Appendix A and are not included here.

5.5 Conclusion

It has been shown how problems can be represented in Allen and Ferguson's logic. It can be seen that this requires more work to represent than using ATL, and some of the axioms are cumbersome compared to ATL. This is necessary because the goal is to use the axioms to derive expressions, and so the axioms need to be thoroughly defined. ATL simplifies the work by using a model-theoretic approach.

Allen and Ferguson (1994) point out two common criticisms of this explanation closure approach. Firstly, the process of writing down all of the axioms correctly is difficult and secondly the approach is "cheating" because the solution is already present in the axioms, nothing new has been added. To the first they respond that these axioms contain vital information that is part of the agent's knowledge about the world, and are thus necessary. They also add that these axioms may be generated by a computer, rather than manually created. To the second, they respond as follows: "As for the issue of cheating, it doesn't seem to us that explicitly encoding the assumptions that make the representation work is any more cheating than hiding the assumptions in a complex model theory. We must remember that these really are assumptions, and that they may be wrong. If we are ever to reason about cases where assumptions have been made and shown to be false, they need to be explicit in the representation." (Allen and Ferguson, 1994, p. 33).

It is clear that ATL is easier to work with, and encoding the assumptions in a model cleans up the formulas a lot. ATL also allows us to talk about multiple agents and their strategies, while ITL is restricted to the perspective of a single agent. Yet Allen and Ferguson's logic still offers us a treatment of pure intervals which cannot be found in any other logic, and which allows us to talk about situations which no other logic can describe, such as complex overlapping events. If intervals can somehow be incorporated into ATL, while keeping the model simple and the formulas elegant, we would have a powerful combination.

Chapter 6

Agent Interval Temporal Logic

This chapter will introduce the new logic called *Agent Interval Temporal Logic*. We saw in an earlier chapter that ATL was useful for talking about situations where a group of agents were working together to achieve some goal. However, it was not possible to represent situations where there was a complex relation between the timing of various actions. ATL had no notion of duration. Actions all happen at the same time, with no overlap of actions nor any interruptions. We then considered various interval logics, to see how we might combine the notion of an interval with the notions of agents and actions from ATL. This chapter will be an attempt to bring together these ideas into a single new logic.

This chapter will start by considering the domain of the new logic, with a discussion on all the concepts which one may or may not want to speak about. After this the model is defined, followed by the syntax and semantics. This is all brought together and illustrated by an example using an adapted dining philosophers problem, followed by an application of the new logic to the three problems which have been used throughout this dissertation.

6.1 Domain

Before we present the model, syntax and semantics, we need to discuss what actually is present in our language, what our logic is about. We want to talk about agents, and the things they do. We are interested in the actions taken by the agents, and the order those actions are in. All of this happens in the context of some scenario or situation or problem. The model of our logic will be called a *scenario model*, which will contain agents, actions and restrictions, and from which various *strategies* can be generated. In our syntax we will have two types of formulæ, one interpreted over a scenario model, the other interpreted over a strategy. These, along with related concepts, will now be informally discussed

before being formally defined in the next section.

A note about terminology: in different disciplines, terms like *model*, *domain*, *interpretation*, *system* and so on have various different and sometimes contradictory meanings. This dissertation is in the field of knowledge representation, but since a lot of the literature used is from the field of model checking, we have been and will continue to use their terminology. The *model* is the mathematical structure which contains the things of the world, while a formula is *interpreted over* a model. This means that a formula will be true or false for one or another model. If the formula talks about something in a model which is the case, then that formula is true for that model. We write this as $M \vDash f$ where M is the model and f is some formula. This is different from the knowledge representation terminology, which calls what is termed a model in the model checking context a *domain of discourse*, and a denotation function (and other functions - such as assignments - depending on whether it is a propositional or a predicate logic) together with a domain of discourse which makes a formula true is called a model.

6.1.1 Scenario

The primary notion, used for representing knowledge about the system at issue, is that of a *scenario* or *situation*. Recall the concurrent game model from ATL, and how ATL was used to represent our three problems. The concurrent game model used for the banker's algorithm and the one used for the sleeping barber problem were very different from each other. They contained different agents, different actions, different propositions, and different states. While it was the same formal concurrent game model, it was filled with different elements depending on the scenario it attempted to represent. The scenario consists of the setup, what there is, the specific agents and their specific actions. The scenario is the problem to be solved. A scenario in ATL might be represented by a concurrent game structure, or in CTL by a Kripke structure, or by any other formal structure for any other logic. The model of AITL is called a *scenario model*, and will be defined as a tuple containing all the other elements of the model.

6.1.2 States and Events

When we look at ATL, we see that it is fundamentally about *states*. A formula in ATL is either about some proposition that is true in the current state, or about a proposition which might become true in a later state. A state then is a way the world is at a moment in time, with various propositions describing the world at that moment.

It is then possible to move from one state to another, through a transition function, which uses some input to determine which state to go to next. This

transition will usually represent an *event*, since we think of events as things that happen which change something in the world. The transition from one state to the next, and thus a change in the state of the world, can then be thought of as an event.

Since this transition function is deterministic (at least for ATL's concurrent game model), we can predict what state we may or may not end up in later on. The logic of CTL was essentially made up of this: a collection of states, transitions between them, and a syntax which allowed us to talk about which states we can reach and when we can reach them. The key difference in ATL, which was an extension of CTL, was simply to complicate the transition function by adding in agents and actions. In AITL, we will not have a notion of state, but instead we will have the notion of an event represented by an action. This is explained more in the actions section.

6.1.3 Agents

Agents are the actors in the model, who cause changes to happen. Agents act, and their actions change the state of the environment. ATL extended CTL by adding the notion of *agents*. Where the transition in CTL was simply that a move from this state to that state was possible, in ATL the transition became an alternating turn "game" between the system and the environment. The system, is comprised of a coalition of agents, that all choose their actions, and the combination of their actions influence the environment. This allowed us to talk about the power an individual agent had to choose an action such that no matter what the other agents choose it could guarantee that some future state might be reached. Our scenario model will also contain a set of agents.

6.1.4 Actions and Tasks

Together with agents we always have *actions*. An action is carried out by an agent, and when carried out it changes the state of the world in ATL. Thus an action and an event both change the state of the world. An action causes an event, and an event changes the state of the world. Events and actions also have duration, and can thus be thought of as intervals. We have already mentioned that events will not be explicit elements in AITL. This is because everything about events that we care about is already captured in the idea of an action, namely the idea of causing some change in the world. Each action will have a unique interval during which it takes place. Rather than having both intervals and actions in the logic, with a one-to-one mapping between them, we will only have actions in AITL [1].

[1]Having both actions and intervals is redundant. We thus merge them and can call them either actions or intervals. Calling them intervals allows us to speak naturally about interval

Since actions are represented by intervals, we can have interval relations between actions. This is similar to how we naturally talk, for example: "I pick up the ball AFTER you drop it", or "He sneezes DURING my speech". Actions are treated as intervals when referring to the relations between them.

The only events which will take place will be actions, and every action needs an agent which carries it out. There are thus no natural events (like weather) in the logic which are not caused directly by some agent. If one wishes to represent a natural event, one may have *nature* as an agent which takes some action.

Each action entity will be a unique event, and cannot be repeated. If one wishes to represent a "repeated action" one needs a different action entity for each time that action is taken. Agents will take actions at different times, and can take two or more actions at the same time, or no actions for a time. If we want to represent an agent taking 2 or more actions at the same time, we can have those multiple actions with the *Equality* relation between them, meaning the intervals of those actions are all equal to each other.

An action may only be taken by a single agent however, two agents may not take the same action. If one wishes two agents to work together, such as carrying two sides of a coffin, one may create an action for the front carrier and an action for the back carrier and formulate a rule that those two actions must happen at the same time (such a rule is called a *restriction* and will be discussed later). What this rule means is that the interval of the one action and the interval of the other action must have the equality relation between them.

There is another closely related term which we must consider, that of a *task*. We might speak of an agent performing a task, but what is the difference between an action and a task? In the original ATL paper by Alur et al. (2002), the word task is used to refer to something which a coalition of agents wishes to carry out, almost like a goal. A task is therefore some larger activity which will achieve some purpose, and may consist of numerous smaller actions by different agents or a single agent. For example, the task of preparing dinner may consist of numerous actions like washing the beans, peeling the banana and cooking the spinach. Tasks are not explicit elements in the logic, only actions are, but in describing a problem one may use the term *task* to refer to a collection of actions.

6.1.5 Intervals

An *interval* is some duration of time, as opposed to an instant. While states can express the way the world is at some moment, intervals can express some event

relations among them, but we must speak unnaturally about agents doing intervals. Calling them actions allows us to speak naturally about agents doing actions, but we must speak unnaturally about interval relations among actions.

taking place over some time. Something can happen during an interval, some change can take place. We will represent intervals as actions in AITL.

Do we also need states? There are many ways to view the relationship between intervals and states. A state might be at the end or beginning of an interval, in which case the transition from one state to the next is an interval. Or an interval might be a set of states, such as in the logic HS from Halpern and Shoham (1991). For AITL, we are only interested in the action taken by the agents, and the timing of those actions. We are thus not interested in any states, or propositions which are true or false during states. As stated before, states will not be a part of AITL. Instead we will reason about events happening, represented as actions.

This approach does have the drawback of not being able to represent complex states at any one point in time. Instead if one wants to know the state of a specific aspect of the world, one could consider the actions that would change that aspect, and ask if those actions happened in a specific order which would change it. This way, one could get an idea of the state of the world after a specific chain of events (captured by an ordering of actions). But this reasoning happens at a higher level when using the logic to represent a situation, it is not captured in the model. This works well for actions that happen once, and change a certain variable, which holds from then on. This approach is less able to represent a situation where one wants to have an action which can be done repeatedly, changing a variable back and forth between two values. Ideally one knows how many times an action must take place, and then creates separate actions for each time.

Consider as an example the act of closing a door. With states, one might have an open and a closed state, and with a transition move from the open state to the closed state. There would be a *doorOpen* proposition, with a labelling function mapping it to true and false in the two states. To find out if the door is closed, we might ask what state we are in, and if in the current state the *doorOpen* proposition is true or false. With intervals however, we might have an action called *closeDoor*. This represents the action of closing the door, and the event of the door being closed. Now we simply ask, has *closeDoor* taken place already or not?

6.1.6 Relations

When adding intervals, relations must naturally be added as well, since relations always exist between intervals. Intervals can be placed in time relative to each other, as we have seen in the work of Allen and Hayes (1985) in chapter 5. Rather than tying intervals to an external clock or timeline, they can be arranged relative to each other through their relations to each other.

In the scenario we will only have a list of actions without any relations. Recall that the scenario is the raw situation, the list of tasks to be done, without any specific order to do them in. The actions in a scenario are thus unordered, and do

not yet have specific relations to each other. Relations are used when specifying restrictions on how they can be ordered (next section), or when creating a strategy by ordering all the actions in some specific way (the section thereafter).

As an example, consider baking a cake. We can have the following five actions which must be carried out: Add Eggs, Add Milk, Add Flour, Stir, Bake. This is the scenario, it contains only an unsorted set of actions, the things which must be completed in this situation. Later we will see how strategies are generated by arranging actions in a specific order using relations.

Recall from section 5.1.2 and figure 5.1 that between any two intervals there exists one of 13 possible relations. These will be represented by the following symbols, which have been adapted from Allen and Ferguson (1994):

- Equals is $=$
- Before is $\prec$ After is $\succ$
- Meets is $:$ MetBy is $\cdot\cdot$
- Overlaps is $\preceq$ OverlappedBy is $\succeq$
- Starts is $\rhd$ StartedBy is $\lhd$
- During is $\subset$ Contains is $\supset$
- Finishes is $\dagger$ FinishedBy is $\ddagger$

These are the 13 core relations. From these we will define five additional relations which are merely shorthand for some of the previous 13. These relations are useful to keep formulæ short. The first three of these relations were derived by Allen and Ferguson (1994).

The *disjoint* operator $\bowtie$ is defined as follows:

$$i \bowtie j \equiv i \prec j \text{ or } i \succ j \text{ or } i : j \text{ or } i \cdot\cdot j$$

The *beforemeets* operator $\prec\!:$ is defined as follows:

$$i \prec\!: j \equiv i \prec j \text{ or } i : j$$

The *equalduring* operator $\subseteq$ is defined as follows:

$$i \subseteq j \equiv i = j \text{ or } i \subset j$$

The *aftermetby* operator $\succ \cdot\cdot$ is defined as follows:

$$i \succ \cdot\cdot j \equiv i \succ j \text{ or } i \cdot\cdot j$$

The *equalcontains* operator $\trianglerighteq$ is defined as follows:

$$i \trianglerighteq j \equiv i = j \text{ or } i \rhd j$$

Note that the *disjoint* relation can also be derived using only *beforemeets* or *aftermetby*.

6.1.7 Restrictions

For a specific scenario, some restrictions might apply on the relations between actions. Perhaps one action must be completed before another can be attempted (recall that actions are seen as intervals). For example, I must open the bottle before I can pour out the liquid. In our scenario we must capture these restrictions, so that we may know in what way actions may be arranged and in what way they may not. From the cake example, we might have the restriction that the mixture must be stirred before baking.

There are two types of restrictions, *conditional* and *unconditional*. Unconditional restrictions hold for the entire scenario, it can never be broken. An example of an unconditional restriction is having to open the bottle before pouring out the liquid. Conditional restrictions only hold if some other relations hold, and thus only become relevant partway through generating a strategy. An example of a conditional restriction is, if I put a cupcake on my small plate, I must first eat it before I can put a muffin on the same plate. Suppose we have four actions, *putcupcake*, *putmuffin*, *eatcupcake*, *putmuffin*. We will have unconditional restrictions about putting the treats on the plate before eating them, but we will need a conditional restriction to say that when one treat has been put on the plate, it must be eaten before the other can be placed on the plate.

Unconditional restrictions are in the form iRj where i and j are actions, and R is a relation. This restriction is interpreted to mean that this relation must hold, it is *restricted to* this relation, not *restricted from* it. We may also write a negated restriction, in the form $\neg iRj$, which means that the relation may be anything but that relation. Sometimes, two or more restrictions could be written for the same two actions, such as $i \lessdot j$ and $i \gtrdot j$. This means that the relation can only be one of those two. When generating a strategy, for two actions which have no restrictions on their relation to each other, the relation can be any of the 13 relations. But as soon as there is at least one restriction, then the relation can only be one of those listed as a restriction. When a restriction is written as a negation, then the relation can be any of the 13 except that one. Note that while the additional five shorthand relations may be used while writing restrictions, when generating the strategy (next section) and deciding on the actual relation between two actions, only one of the 13 must be picked.

Conditional restrictions are written in a similar format, but with the additional introduction of the implication operator. If i, j, k, and l are actions, then a conditional restriction is written as $iRj \rightarrow kQl$, where R and Q are relations, including the additional shorthand relations. This restriction says that if i is in R relation to j, then k must be in Q relation to l. One may want to add some conjunction operator to the consequent, so that it can be said that if some relation holds, then another relation may be any of some group of possibilities. But this is unnecessary since one may simply add multiple restrictions with the same antecedent, which has the same meaning.

Using these restrictions, one may set up a complicated scenario with rules for how the scenario may unfold. In practical use, we suspect the most time and effort would be spent on formulating restrictions if one is to use AITL. However, as one adds more restrictions the computation for strategies later becomes computationally much easier to handle. As the number of actions increases, the possible arrangements of those actions increases exponentially. Each added restriction cuts down the number of possible arrangements.

6.1.8 Strategies and Goals

While ATL does not contain an explicit object representing the notion of a *goal*, it is a natural part of talking about agents and actions, and a property of the execution of a strategy in ATL. An agent carries out actions for some purpose after all, it must have a goal in mind. ATL is not so concerned with the goal of the agent, but rather with the power which the agent has to force the system into a specific state [2]. Though if the agent is forcing the system into a state, we can speak about the agent having a goal to get the system into that state. This also applies to avoiding a state. We saw examples of deadlock avoidance in ATL, here the goal of the agents can be said to be avoiding deadlock. While ATL does not have a notion of a goal, it does speak about the notion of a strategy.

An agent's *strategy* in ATL was defined by Alur et al. (2002) to be a function that for every element in a sequence of states determines a move for that agent. A strategy then, is something that an agent has, a sort of plan to reach a certain state. If the goal is to reach a state, the strategy is the plan on how to get there. Strategies are an important part of ATL, and are explicitly defined, but are not a

[2] If an agent or coalition is able to make certain choices which guarantee that the system will end up in a certain state, no matter what other parts of the system does, they are said to have the power to force the system into that state. Alur et al. (2002) states that: "besides universal (do all computations satisfy a property?) and existential(does some computation satisfy a property?) questions, a third question arises naturally: can the system resolve its internal choices so that the satisfaction of a property is guaranteed no matter how the environment resolves the external choices?".

part of the concurrent game model. We saw in the ATL chapter various extensions of ATL which made strategies an explicit part of the model.

From the AITL scenario model, one can generate many strategies. The scenario is the problem, the strategies are the possible solutions. For example, if the scenario contains the five actions of baking a cake, one can derive many strategies, such as adding the flour first and then the egg, or the egg first and then the flour. Specifically we must arrange all five actions with respect to each other, as well as assign each action an agent carrying it out, before we have a strategy. A strategy is only valid if it is within the restrictions of the scenario model. Some actions will be restricted with respect to the relations they may have to each other, and some actions may only be assigned to specific agents.

Having considered all the elements in AITL, we now move to formal definitions of those elements.

6.2 Model

The model is called a *scenario model*, and for each scenario model a range of *strategies* can be defined. We now define the scenario model and then we define a strategy.

6.2.1 Scenario Model

Definition 6.1. A *scenario model* SM will be represented by a tuple in the following form:

$$SM = (Agt, Act, tsk, Res)$$

where:

- Agt is a nonempty finite set of agents.
- Act is a nonempty finite set of actions.
- tsk is the *tasking* function, which assigns each action to a set of agents $tsk : Act \to \mathcal{P}(Agt)$
- Res is a nonempty finite set of restrictions

We have discussed the notions of agents, actions, and restrictions. Res contains both the conditional and unconditional restrictions.

The tsk function will indicate to us, for every action, which agents may carry out that action. It might be possible that a certain action can be carried out by any agent, or that only one agent may do so. The $\mathcal{P}$ symbol refers to the power set.

6.2.2 Strategies

A scenario model contains actions and agents, with restrictions on the relations between actions (Res) and restrictions on which agents may carry out which actions (tsk). A strategy arranges all the actions in keeping with the restrictions, as well as assigning those actions to agents, again in keeping with the restrictions of the tasking function. One scenario model can give us many strategies.

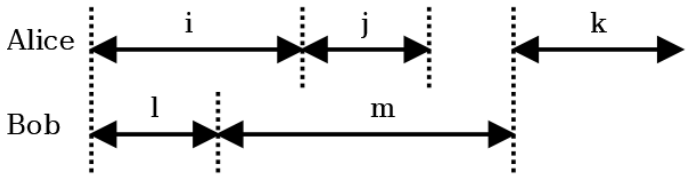
Definition 6.2. A *strategy* Str, defined for a specific scenario model SM, consists of a tuple in the following form:

$$Str = (tab, S_1, S_2, ..., S_k)$$

where:

- tab is a function which for every set of two actions gives the relation between them, $tab : Act \times Act \to R$ where R is one of the 13 relations.
- S_a is a *responsibility set* for each agent $a \in Agt$. This set contains those actions which have been assigned to agent a.

The tab function tells us how the actions have been arranged in this specific strategy, it can be visualised as a table showing every relation for each two actions. The responsibility sets tell us how the actions have been assigned to agents in this specific strategy. Note that in a scenario there are many options for the arrangements and assignment of actions, while a strategy is a single one of those arrangements and assignments. Consider the following example timeline and table:



Each timeline can be partially described by a table (which agents carry out which tasks is not represented).

	i	j	k	l	m
i		:	$\prec$	$\lhd$	$\leq$
j	..		$\prec$	$\succ$	$\subset$
k	$\succ$	$\succ$		$\succ$	..
l	$\rhd$	$\prec$	$\prec$		:
m	$\geq$	$\supset$	:	..	

To generate a strategy is to fill out such a table such that no contradictions exist, as well as assigning responsibilities to agents. For this example the responsibility sets would be as follows:

$$S_{Alice} = (i, j, k)$$

$$S_{Bob} = (l, m)$$

Having defined the scenario model and the strategy, let us consider the syntax for writing formulæ in AITL. Note that there are two different types of formulæ, one interpreted over a scenario model and one interpreted over a strategy.

6.3 Syntax

The two types of formulæ are called *strategic formulæ* (written as φ) and *scenario formulæ* (written as ψ).

Strategic formulæ are interpreted over a strategy. The syntax for a well formed *strategic formula* is as follows:

$$\varphi ::= iRj \mid \tau_a i \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$$

Where i and j are actions, a is an agent. The first formula, iRj will speak about the relation between two actions in this specific strategy. The second formula, $\tau_a i$ is read as "Agent a is responsible for action i", and will speak about the assignment of actions in this specific strategy. The last two formulæ are the common logical operators. Additional logical operators $\wedge$, $\rightarrow$ and $\leftrightarrow$ can be derived as before.

Scenario formulæ are interpreted over a scenario model. The syntax for a well formed *scenario formula* is as follows:

$$\psi ::= \exists\varphi \mid \forall\varphi \mid \neg\psi \mid \psi_1 \vee \psi_2$$

We see here the usage of the familiar symbols for the quantifiers from first order logic. These operators are however redefined, i.e., they and are used in a different way in AITL than the common usage in first order logic. This approach is similar to the usage of quantifiers in CTL, which were redefined to talk about the existence of paths which satisfied a formula. The φ here refers to a strategic formula. $\exists\varphi$ is read as "It is possible to construct a strategy which satisfies φ", while $\forall\varphi$ is read as "For every possible strategy which we can construct, φ is satisfied". The last two rules are the common logical operators. Additional logical operators $\wedge$, $\rightarrow$ and $\leftrightarrow$ can be derived as before.

Strategic formulæ speak about the actions and agents in a specific strategy, while scenario formulæ speak about which kinds of strategies can be constructed from them.

6.4 Semantics

The evaluation of a strategic formula φ on a strategy Str written as $Str \vDash \varphi$, is defined as follows:

- $Str \vDash iRj$, iff $tab(i, j) = R$, where R is a relation and tab the tasking function.
- $Str \vDash \tau_a i$, iff $i \in S_a$, where a is an agent and S_a is the reponsibility set of a.
- $Str \vDash \neg\varphi$, iff $Str \nvDash \varphi$.
- $Str \vDash \varphi_1 \vee \varphi_2$, iff $Str \vDash \varphi_1$ or $Str \vDash \varphi_2$.

The first two rules are primitives, one about the relations between two actions which is true if that relation is indeed given by the tab function, and the other about the assignment of actions to agents which is true if that assignment is indeed given by the responsibility sets. The last two rules are logical operators with their usual meanings, to allow us to build up more complex formulæ from the primitives.

The evaluation of a scenario formula ψ on a scenario SM, written as $SM \vDash \psi$, is defined as follows:

- $SM \vDash \exists\varphi$, iff a strategy Str can be generated such that $Str \vDash \varphi$.
- $SM \vDash \forall\varphi$, iff for every strategy Str that can be generated $Str \vDash \varphi$.
- $SM \vDash \neg\psi$, iff $SM \nvDash \psi$.
- $SM \vDash \psi_1 \vee \psi_2$, iff $SM \vDash \psi_1$ or $SM \vDash \psi_2$.

Here again the first two rules are primitives, and the second two are logical operators on those primitives. As mentioned in the Syntax, additional operators such as conjunction and implication can be derived from the operators here. Note that the quantifiers do not speak about the existence of formulæ, but about the existence of strategies in the model which can satisfy those formulæ. As we will see soon, strategies are generated for a scenario, and so instead of speaking in the formal semantics about the existence of strategies, we speak about the fact that a strategy can be generated, though it has similar meaning. Sometimes we may wish to speak about the existence of at least one strategy where some relation between two actions holds. In that case we can use the existential quantifier.

Other times, we may wish to speak about the fact that every possible strategy will have a specific relation between two actions. In that case we can use the universal quantifier. Any relation between two actions which was specified as a restriction will have that relation be true in every possible strategy, since only strategies generated in keeping with the restrictions are valid strategies.

6.5 Example

As a clarifying example, consider the following adaption of the classic dining philosophers problem (Hoare, 1978).

Three philosophers are eating pasta, seated around a table. Three forks are between them, one fork between each two philosophers. A philosopher will sit and think, until they decide to eat. They need a fork in each hand to be able to eat. Once they are done eating, they will put down the forks and go to sleep. They only need to eat once for this example, so once they sleep they continue to sleep. We want to avoid a deadlock situation where each philosopher has one fork in hand at the same time, since they cannot put down a fork before having eaten. We will try to find a strategy which ends up in all three philosophers being asleep, since this means they all successfully ate.

We start by creating the scenario model which will represent the situation.

$$SM = (Agt, Act, tsk, Res)$$

We will define these fields as follows:

- $Agt = \{P_1, P_2, P_3\}$
- Act is a set of actions consisting of the following elements:
  - $think_1$
  - $think_2$
  - $think_3$
  - $sleep_1$
  - $sleep_2$
  - $sleep_3$
  - $holdFork_1left$
  - $holdFork_2left$
  - $holdFork_3left$
  - $holdFork_1right$

  - $holdFork_2right$
  - $holdFork_3right$

- tsk is a function mapping each action to the agent which must carry out that action as follows (note that in this scenario each action can only be carried out by one agent, while in other scenarios for other problems an action might have multiple candidates):
  - $tsk(think_1) = P_1$
  - $tsk(think_2) = P_2$
  - $tsk(think_3) = P_3$
  - $tsk(sleep_1) = P_1$
  - $tsk(sleep_2) = P_2$
  - $tsk(sleep_3) = P_3$
  - $tsk(holdFork_1left) = P_1$
  - $tsk(holdFork_2left) = P_2$
  - $tsk(holdFork_3left) = P_3$
  - $tsk(holdFork_1right) = P_1$
  - $tsk(holdFork_2right) = P_2$
  - $tsk(holdFork_3right) = P_3$
- Res is a set of restrictions consisting of the following elements. Note that the inverses of all of these are implied and need not be included here, for example if i is after j, that implies that j is before i.
  - $think_1 : holdFork_1left$
  - $think_1 \prec holdFork_1left$
  - $think_1 : holdFork_1right$
  - $think_1 \prec holdFork_1right$ These four restrictions say that the agent will think before picking up a fork, and will stop thinking once they picked up the first fork
  - $holdFork_1left : sleep_1$
  - $holdFork_1right : sleep_1$ These two restrictions say that agent 1 sleeps immediately after holding forks in both hands]
  - The above six restrictions are repeated two more times for agents 2 and 3.

- $holdFork_1left \bowtie holdFork_3right$
- $holdFork_2left \bowtie holdFork_1right$
- $holdFork_3left \bowtie holdFork_2right$ These three restrictions say that there can be no overlap between the actions where two agents grab the same fork.

With the model defined, we can start to generate strategies and talk about those strategies. Recall that a strategy can be generated from a scenario. A strategy is defined as a tuple, and for this scenario with three agents it is written as:

$$Str = (tab, S_1, S_2, S_3)$$

where *tab* is the function giving the arrangement of actions and S_a is the *responsibility set* for agent a.

The *tab* function can be represented as a table. For a specific generated strategy, there will be only one relation between any two actions. We can however start to partially complete a table with the restrictions. We make a table with all the actions, and start filling in the restrictions and their inverses as follows:

	t_1	t_2	t_3	s_1	s_2	s_3	h_1l	h_2l	h_3l	h_1r	h_2r	h_3r
t_1							: ≺			: ≺		
t_2								: ≺			: ≺	
t_3									: ≺			: ≺
s_1							··			··		
s_2								··			··	
s_3									··			··
h_1l	·· ≻			:								: ·· ≺ ≻
h_2l		·· ≻			:					: ·· ≺ ≻		
h_3l			·· ≻			:					: ·· ≺ ≻	
h_1r	·· ≻			:				: ·· ≺ ≻				
h_2r		·· ≻			:				: ·· ≺ ≻			
h_3r			·· ≻			:	: ·· ≺ ≻					

We can derive additional restrictions from those already provided. One such restriction is that an agent's thinking action will be before its sleeping action (since thinking happens before taking forks, and sleeping happens after taking forks). Another is that an agent's two fork actions will either be equal or finish each other (since they both meet the sleeping action). We can fill these into the table.

	t_1	t_2	t_3	s_1	s_2	s_3	h_1l	h_2l	h_3l	h_1r	h_2r	h_3r
t_1				$\prec$			$: \prec$			$: \prec$		
t_2					$\prec$			$: \prec$			$: \prec$	
t_3						$\prec$			$: \prec$			$: \prec$
s_1	$\succ$						$\cdot\cdot$			$\cdot\cdot$		
s_2		$\succ$						$\cdot\cdot$			$\cdot\cdot$	
s_3			$\succ$						$\cdot\cdot$			$\cdot\cdot$
h_1l	$\cdot\cdot \succ$			$:$						$= \dagger \ddagger$		$: \cdot\cdot \prec \succ$
h_2l		$\cdot\cdot \succ$			$:$					$: \cdot\cdot \prec \succ$	$= \dagger \ddagger$	
h_3l			$\cdot\cdot \succ$			$:$					$: \cdot\cdot \prec \succ$	$= \dagger \ddagger$
h_1r	$\cdot\cdot \succ$			$:$			$= \dagger \ddagger$	$: \cdot\cdot \prec \succ$				
h_2r		$\cdot\cdot \succ$			$:$			$= \dagger \ddagger$	$: \cdot\cdot \prec \succ$			
h_3r			$\cdot\cdot \succ$			$:$	$: \cdot\cdot \prec \succ$		$= \dagger \ddagger$			

We can keep deriving more restrictions until there are no more restrictions to be derived. Using an automated algorithm, we could derive all possible restrictions. For the sake of this example, we stop here. Each cell in the table now is either blank, or has one or more relations on it. If there are relations in a cell, it means that when creating a strategy, that relation can only be one of those relations in the cell. If a cell is blank, it means that any one of the 13 relations are possible. The set of all strategies in the scenario can now be obtained by taking every possible combination of allowed relations in this table. Since the table is symmetrical (with inverses across the blue diagonal line of cells), only less than half of the cells need to be considered when generating the set of all possible strategies, since the other half will be the inverse. The more restrictions are originally derived in the scenario, the fewer possible strategies exist. When generating a strategy, for every cell which is given a relation, additional relations can be derived for other cells, imposing more restrictions. This process of filling out a table and generating possible strategies could be automated. Also note that the number of possible strategies will be very large, and exponentially more so for larger problems.

In order to derive more restrictions, we will use interval algebra from Allen (1990). The detail of this algebra and the automated algorithm which would use it is beyond the scope of this dissertation. The point is to keep deriving more restrictions which are already implied by existing ones. For example, if we must put on socks before putting on shoes, and if we must put on shoes before going running, we can derive that we must put on shoes before going running. It is important to derive as many restrictions as possible, since the more restrictions, the less possible strategies. Once all the restrictions which can be derived, have been derived, it is time to generate strategies.

To generate a strategy is to fill out the table in such a way that no contradictions exist, such as an action being before another action but also after it. This is done

by carefully following the interval algebra of Allen (1990) while filling out a table. One simple brute force algorithm is to generate every possible combination of relations between every two actions, and then cut out those combinations which contain contradictions according to Allen's interval algebra. A cell on the table which contains no restrictions (such as cell t_1 t_2 in the above table), can have one of a total of 13 possible values. A cell containing four restrictions (such as cell $h_3 r$ $h_1 l$ in the above table) can have one of those four values.

To summarise, we first insert the restrictions into the table, then we derive additional restrictions (already implied by the original restrictions) and insert those into the table, then we start generating a strategy by filling in the rest of the table without contradictions. All of this would of course be done by an automated algorithm, probably in a brute force fashion, and be of high complexity.

Having seen how strategies may be generated, let us now consider the specific strategy which may result in all the philosophers being asleep. This requires that there is some overlap between the sleep actions of the philosophers. We can now put this property into a well formed scenario formula F, using the shortened action names from the table for ease of reading:

$$F = \exists(\neg s_1 \bowtie s_2 \wedge \neg s_2 \bowtie s_3 \wedge \neg s_1 \bowtie s_3)$$

Let us look more closely at what the formula is saying. The very first symbol $\exists$ indicates that there exists a strategy which will make this formula true. The rest is a conjunction of three negated statements, claiming the sleep times of all three philosophers overlap each other, or specifically that they are not disjoint. If the sleep times of 1 and 2 overlap, and 2 and 3 overlap, and 3 and 1 overlap, then there must be a point at which all 3 overlap.

We must now show that this formula is well formed. Recall that the syntax of a well formed scenario formula is as follows:

$$\psi ::= \exists\varphi \mid \forall\varphi \mid \neg\psi \mid \psi_1 \vee \psi_2$$

Where φ is a strategic formula. Looking at our formula, we see that we have the first rule, $\exists\varphi$. We must then show that the rest of the formula, everything without $\exists$, is a strategic formula.

Recall that the syntax of a well formed strategic formula is as follows:

$$\varphi ::= iRj \mid \tau_a i \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$$

We will prove that our formula is well formed by constructing it from this syntax. We start with the first rule, iRj, where i and j are two actions, and R is one of the 13 relations, or R is the disjoint relation $\bowtie$ which was derived from the other relations. We generate three subformulæ using this rule:

- $\varphi_1 := s_1 \bowtie s_2$
- $\varphi_2 := s_2 \bowtie s_3$
- $\varphi_3 := s_1 \bowtie s_3$

Since for this task we are not interested in which agent carries out what action, and since each action can only be carried out by one agent in this task anyway, we will not be using the second rule. We generate the following three subformulæ using the third negation rule:

- $\varphi_4 := \neg\varphi_1 \equiv \neg s_1 \bowtie s_2$
- $\varphi_5 := \neg\varphi_2 \equiv \neg s_2 \bowtie s_3$
- $\varphi_6 := \neg\varphi_3 \equiv \neg s_1 \bowtie s_3$

Next we will generate a sub formula using the conjunction operator:

- $\varphi_7 := \varphi_4 \wedge \varphi_5 \wedge \varphi_6 \equiv \neg s_1 \bowtie s_2 \wedge \neg s_2 \bowtie s_3 \wedge \neg s_1 \bowtie s_3$

Thus this section is a well formed strategic formula φ, which together with the existential quantifier $\exists$ makes the scenario formula F, which states that a strategy exists which will make this strategic formula true. To show that this formula is true, we will generate a strategy where the three philosophers sleep at the same time, and prove that it satisfies the formula F. This strategy is called Str and is defined as follows:

$$Str = \{tab, S_1, S_2, S_3\}$$

where tab is a function relating every action to every other action, and S_1, S_2, and S_3 are responsibility sets for each agent.

Recall that the function tab can be written as a table, which we do here [3]:

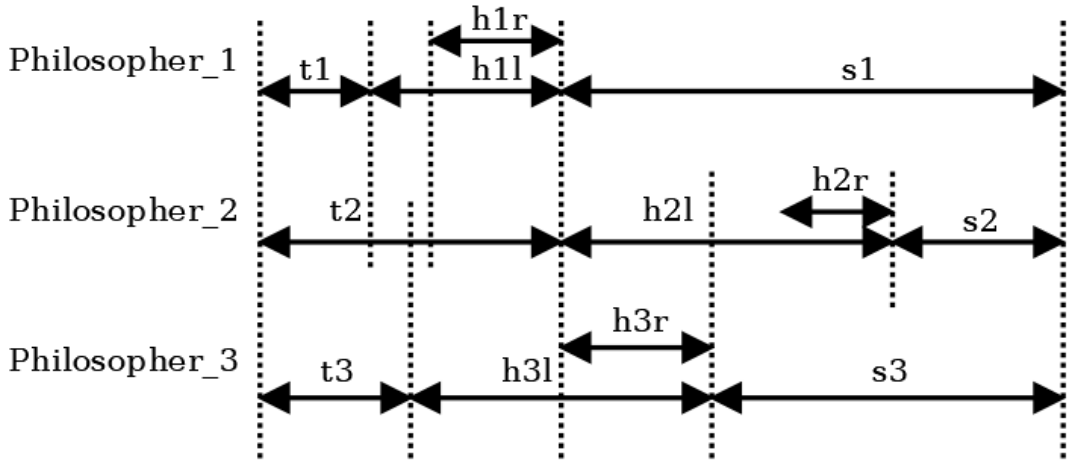
[3]Note that this is but one of many possible strategies. We know a strategy is complete when each cell contains only one value, since that specifies the relation between two actions. Many of these values in this table could have been different, for different strategies. The important thing is that any valid strategy has to be in line with the restrictions. The reader can compare this complete table to the in progress tables from earlier and see that it does not voilate any of the restrictions from earlier.

	t_1	t_2	t_3	s_1	s_2	s_3	h_1l	h_2l	h_3l	h_1r	h_2r	h_3r
t_1		$\rhd$	$\rhd$	$\prec$	$\prec$	$\prec$	:	$\prec$	$\prec$	$\prec$	$\prec$	$\prec$
t_2	$\lhd$		$\lhd$	:	$\prec$	$\prec$	$\ddagger$	:	$\preceq$	$\ddagger$	$\prec$	:
t_3	$\lhd$	$\rhd$		$\prec$	$\prec$	$\prec$	$\preceq$	$\prec$	:	$\prec$	$\prec$	$\prec$
s_1	$\succ$	$\cdot\cdot$	$\succ$		$\ddagger$	$\ddagger$	$\cdot\cdot$	$\lhd$	$\succeq$	$\cdot\cdot$	$\supset$	$\lhd$
s_2	$\succ$	$\succ$	$\succ$	†		†	$\succ$	$\cdot\cdot$	$\succ$	$\succ$	$\cdot\cdot$	$\succ$
s_3	$\succ$	$\succ$	$\succ$	†	$\ddagger$		$\succ$	$\succeq$	$\cdot\cdot$	$\succ$	$\supset$	$\cdot\cdot$
h_1l	$\cdot\cdot$	†	$\succeq$	:	$\prec$	$\prec$		:	$\preceq$	$\ddagger$	$\prec$	:
h_2l	$\succ$	$\cdot\cdot$	$\succ$	$\rhd$	:	$\preceq$	$\cdot\cdot$		$\succeq$	$\cdot\cdot$	$\ddagger$	$\lhd$
h_3l	$\succ$	$\succeq$	$\cdot\cdot$	$\preceq$	$\prec$	:	$\succeq$	$\preceq$		$\supset$	$\prec$	$\ddagger$
h_1r	$\succ$	†	$\succ$	:	$\prec$	$\prec$	†	:	$\subset$		$\prec$	:
h_2r	$\succ$	$\succ$	$\succ$	$\subset$	:	$\subset$	$\succ$	†	$\succ$	$\succ$		$\succ$
h_3r	$\succ$	$\cdot\cdot$	$\succ$	$\rhd$	$\prec$	:	$\cdot\cdot$	$\rhd$	†	$\cdot\cdot$	$\prec$	

For the responsibility sets, we know the four actions each agent will take, so for every strategy they will be the same, namely:

- $S_1 = \{t_1, h_1l, h_1r, s_1\}$
- $S_2 = \{t_2, h_2l, h_2r, s_2\}$
- $S_3 = \{t_3, h_3l, h_3r, s_3\}$

Putting these together, we can draw a timeline for this strategy.



Note that this timeline is just for clarification in this example, it does not form a part of the formal model of the logic. This timeline is implied by the *tab* function and the responsibility sets of the strategy.

We must now show that this strategy Str satisfies the strategic formula, or that $Str \vDash F_s$:

$F_s = \neg s_1 \bowtie s_2 \wedge \neg s_2 \bowtie s_3 \wedge \neg s_1 \bowtie s_3$

We start by considering the three subformulæ consisting only of primitives as follows:

- $\varphi_1 = s_1 \bowtie s_2$
- $\varphi_2 = s_2 \bowtie s_3$
- $\varphi_3 = s_1 \bowtie s_3$

Recall that $i \bowtie j$ is true if either $i \prec j$, $i \succ j$, $i : j$ or $i \cdot\cdot j$ is true. Looking at our table we see that $s_1 \ddagger s_2$, $s_2 \dagger s_3$ and $s_1 \ddagger s_3$, thus none of the above three subformulæ are true for Str, or:

- $Str \nvDash \varphi_1 = s_1 \bowtie s_2$
- $Str \nvDash \varphi_2 = s_2 \bowtie s_3$
- $Str \nvDash \varphi_3 = s_1 \bowtie s_3$

Next we consider the inverse of each subformula:

- $\varphi_4 = \neg\varphi_1 = \neg s_1 \bowtie s_2$
- $\varphi_5 = \neg\varphi_2 = \neg s_2 \bowtie s_3$
- $\varphi_6 = \neg\varphi_3 = \neg s_1 \bowtie s_3$

Since we saw that φ_1, φ_2 and φ_3 were not true in Str, their inverses will be true, or:

- $Str \vDash \varphi_4 = \neg\varphi_1 = \neg s_1 \bowtie s_2$
- $Str \vDash \varphi_5 = \neg\varphi_2 = \neg s_2 \bowtie s_3$
- $Str \vDash \varphi_6 = \neg\varphi_3 = \neg s_1 \bowtie s_3$

Next we consider the conjunction of φ_4, φ_5 and φ_6, which is true if each part is true, or:

$$Str \vDash F_s = \varphi_4 \wedge \varphi_5 \wedge \varphi_6$$

$$Str \vDash F_s = \neg s_1 \bowtie s_2 \wedge \neg s_2 \bowtie s_3 \wedge \neg s_1 \bowtie s_3$$

Thus the strategy we generated does satisfy the formula, or $Str \vDash F_s$. Note that this was a strategic formula satisfied by a strategy. We can also have a scenario formula of the form:

$$\begin{aligned} F_e &= \exists F_s \\ &= \exists(\neg s_1 \bowtie s_2 \wedge \neg s_2 \bowtie s_3 \wedge \neg s_1 \bowtie s_3) \end{aligned}$$

Which means that for scenario M, there exists at least one strategy which satisfies F_s. Since we have generated a strategy which does satisfy F_s, we have also proven that at least one such strategy does exist, and thus $M \vDash F_e$. Now consider a scenario formula of the form:

$$\begin{aligned} F_a &= \forall F_s \\ &= \forall(\neg s_1 \bowtie s_2 \wedge \neg s_2 \bowtie s_3 \wedge \neg s_1 \bowtie s_3) \end{aligned}$$

This states that for all possible strategies the philosophers sleep at the same time. We cannot prove this without generating every single strategy, but with some simple reasoning we can see that the philosophers will always end up sleeping: since the timeline ends at the same point for all agents, and since agents sleep until the end, the sleep actions will always either finish or be finished by each other, or in rare cases be equal. This means that for every possible strategy the sleep actions will be joint, thus making F_a true, or $M \vDash F_a$. Note that this does not state that deadlock is impossible in the dining philosophers problem as a whole, but rather that for our scenario model, including specifically the restrictions which we have specified, deadlock is impossible. If one wishes to avoid deadlock, or any other undesirable state, one must put that into the restrictions, and any strategy that can be generated would be deadlock (or other undesirable state) free.

Having seen an in depth example, we now turn to our three familiar problems from previous chapters.

6.6 Problems

6.6.1 The Banker's Algorithm

In order to show the amounts borrowed by each agent, we will have a unique action for each unit borrowed, with the names of those action indicating the total amount borrowed. Recall that an agent will borrow more and more over time until they have enough, then they will return everything. For example, we might have action *AliceBorrow1*, *AliceBorrow2*, *AliceBorrow3*, and so on, with restrictions ensuring

2 is after 1, 3 is after 1 and 2, and so on. The difficulty here is keeping track of the amount of units left in the bank, to ensure the agents do not borrow more than what is available. This will be done with a complex combination of restrictions ensuring there can be no amount of borrows at the same time which exceed the available cash. For example, if the available cash is 6, and there are only two agents, Alice and Bob, we must ensure that the following combinations are not possible:

- $AliceBorrow1 \wedge BobBorrow6$
- $AliceBorrow2 \wedge BobBorrow5$
- $AliceBorrow3 \wedge BobBorrow4$
- $AliceBorrow4 \wedge BobBorrow3$
- $AliceBorrow5 \wedge BobBorrow2$
- $AliceBorrow6 \wedge BobBorrow1$

Once an agent has borrowed enough, they can go into an action called *AliceDone* until the end. This is similar to the philosophers sleeping at the end. To see if a successful strategy exists, we see if the agents all end up being *done*.

For this specific example, let's say the available cash is 6. We have our common three agents. Alice needs to borrow 4, Bob needs to borrow 6, Charlie needs to borrow 2. The scenario model will thus be as follows:

$$M = (Agt, Act, tsk, Res)$$

where:

- $Agt = \{\text{Alice, Bob, Charlie}\}$
- Act is a set of actions consisting of the following elements:
  - $aliceBorrow1$
  - $aliceBorrow2$
  - $aliceBorrow3$
  - $aliceBorrow4$
  - $aliceDone$
  - $bobBorrow1$
  - $bobBorrow2$

  - $bobBorrow3$
  - $bobBorrow4$
  - $bobBorrow5$
  - $bobBorrow6$
  - $bobDone$
  - $charlieBorrow1$
  - $charlieBorrow2$
  - $charlieDone$

- tsk is a function mapping each action to the agent which must carry out that action. For this scenario each action can only be carried out by one single agent:
  - $tsk(aliceBorrow1) = Alice$
  - $tsk(aliceBorrow2) = Alice$
  - $tsk(aliceBorrow3) = Alice$
  - $tsk(aliceBorrow4) = Alice$
  - $tsk(aliceDone) = Alice$
  - $tsk(bobBorrow1) = Bob$
  - $tsk(bobBorrow2) = Bob$
  - $tsk(bobBorrow3) = Bob$
  - $tsk(bobBorrow4) = Bob$
  - $tsk(bobBorrow5) = Bob$
  - $tsk(bobBorrow6) = Bob$
  - $tsk(bobDone) = Bob$
  - $tsk(charlieBorrow1) = Charlie$
  - $tsk(charlieBorrow2) = Charlie$
  - $tsk(charlieDone) = Charlie$

- Res is a set of restrictions consisting of the following two types:
  - Restrictions to ensure the borrows happen in the correct order, borrow 1 first then 2 and so on. Also the *done* actions must be only after the last amount borrowed.

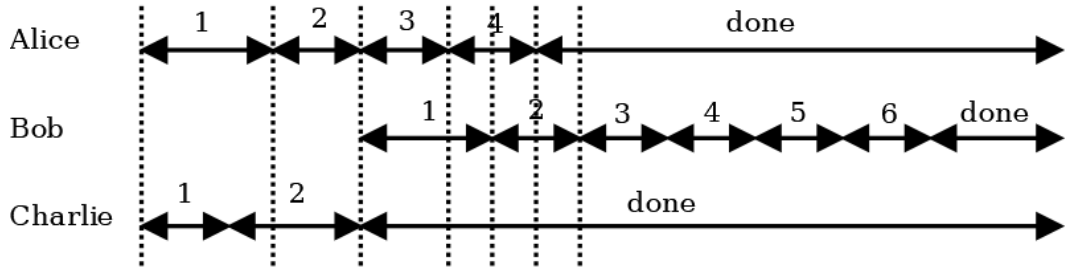


Figure 6.1: Example of a Successful Borrowing Strategy

– Restrictions to ensure the total amount borrowed does not exceed the amount available. This will be a very long list consisting of every possible combination which exceed the total.

From here strategies can be generated by arranging the actions in such a way as to not exceed the total, and thus ensuring all the borrows are successfully completed. We would try to generate a strategy where the *done* actions are all overlapping, which means the agents all completed their borrows.

6.6.2 The Sleeping Barber

The challenge here is that we do not know how many times the barber will sleep. The barber may cut the hair of one person after another with no sleep, or sleep in between each cut. To solve this, we find that for 3 people the barber may sleep a max of 3 times, since he will not wake himself. We can thus create 3 actions *firstSleep*, *secondSleep*, *thirdSleep*. If the barber only sleeps once in between cutting for example, we can put the second and third sleeps back to back at the end of the workday. We will not keep track of which chair the barber is sleeping in.

We have a barber, a cutting chair, three waiting chairs, and three customers Alice, Bob and Charlie. The scenario model will thus be as follows:

$$M = (Agt, Act, tsk, Res)$$

where:

- Agt = {Barber, Alice, Bob, Charlie}
- Act is a set of actions consisting of the following elements:
  - $enterAlice$
  - $enterBob$
  - $enterCharlie$

  - $waitAlice$
  - $waitBob$
  - $waitCharlie$
  - $sitAlice$
  - $sitBob$
  - $sitCharlie$
  - $cutAlice$
  - $cutBob$
  - $cutCharlie$
  - $firstSleep$
  - $secondSleep$
  - $thirdSleep$

  Each customer will take the enter action, if the barber is sleeping while the customer enters, the customer will wake up the barber, this is part of the enter action. If the barber is busy while the customer enters, the customer will take the wait action. Once the barber is done cutting hair, if there is a customer waiting that customer will finish waiting and take the sit action. While the customer sits, the barber takes the cut action. Once the barber is done cutting, he either goes to sleep or cuts the next depending on if a customer is waiting.

- tsk is a function mapping each action to the agent which must carry out that action. For this scenario each action can only be carried out by one single agent:

  - $tsk(enterAlice) = Alice$
  - $tsk(enterBob) = Bob$
  - $tsk(enterCharlie) = Charlie$
  - $tsk(waitAlice) = Alice$
  - $tsk(waitBob) = Bob$
  - $tsk(waitCharlie) = Charlie$
  - $tsk(sitAlice) = Alice$
  - $tsk(sitBob) = Bob$
  - $tsk(sitCharlie) = Charlie$

- $tsk(cutAlice)$ = $Barber$
- $tsk(cutBob)$ = $Barber$
- $tsk(cutCharlie)$ = $Barber$
- $tsk(firstSleep)$ = $Barber$
- $tsk(secondSleep)$ = $Barber$
- $tsk(thirdSleep)$ = $Barber$

- *Res* is a set of restrictions. These have to ensure the correct order of events as described before. Many conditional restrictions will be needed here for ensuring things like when the barber is done cutting, if a customer is waiting, cut the hair, or if none are waiting, go to sleep.

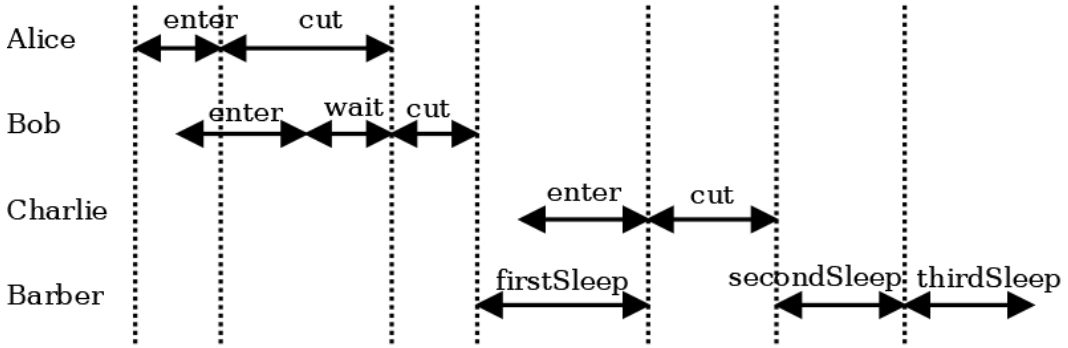


Figure 6.2: Example of a Barbershop Strategy Where the Barber Sleeps Once

A strategy can be generated in keeping with the restrictions. The danger with this problem, as we noted before, is when the barber finishes and notices no one waiting, and goes to sleep, however a customer enters while he is on his way, the customer sees he is not yet sleeping and goes to wait. This results in a forever sleeping barber and forever waiting customer. If a strategy is successfully generated with these restrictions, which contains all the actions in some order, and everyone ends up getting a cut, then we do not run into that problem.

6.6.3 The Trains Problem

This problem demonstrates a major shortcoming of the new logic AITL. While the banker problem did not have any repeated actions, the barbershop did. Since the barber could sleep multiple times, and we did not know how many, it presented a challenge. However, since we knew the maximum number of times the barber would sleep, we could create that number of actions, and put all unused actions at

the end. For the trains problem, this challenge is much greater. The trains have many different routes they can go, and the agents can run between crossroads an unlimited number of times. Furthermore we cannot store the setting of a crossroad, and will have to look at the most recent action which switched it to determine the current setting. This problem demonstrates the sorts of problems for which this logic is not suited.

One possible approach would be to divide the track into sections, as was done for ATL. We can have an action for each train traversing each section, and put unused actions at the end, as was done with unused naps for the barber. The agents may be given a maximum number of times they may run around. Restrictions will govern which traversal action is after which other based on the relation to the most recent crossroad switching action. While all this may eventually allow us to represent the problem, it will be very cumbersome and inelegant.

6.7 Conclusion

We have created a new logic to speak about the actions which agents take, and the order they take them in. We did this by thinking about actions as intervals, and ordering those actions relative to each other using relations. We created a scenario model, which contains the agents and actions, as well as an assignment of agents to actions, and a set of restrictions. The restrictions allow us to limit what is possible with the relations, and to add rules of which actions have to have specific relations to each other. From a scenario model we can derive a strategy, which is simply a specific arrangement of actions and an assignment of actions to specific agents. We have a syntax for formulæ to talk about properties of the strategies, and about the existence of certain strategies for scenarios. We then applied this new logic to various problems.

An interesting note about this new logic is that, with the correct restrictions, undesirable strategies cannot be derived. For example in a problem which might result in deadlock, with the correct restrictions, it becomes impossible to derive a strategy which results in deadlock. Since all actions have to be in the strategy, and have relations to all other actions, it is impossible to end up with a deadlock which excludes various actions. Compare this to a logic like ATL, where a certain state might be marked as a deadlock state, and we then look for a path or strategy which is able to avoid that deadlock state. In AITL, we create restrictions which make it impossible to derive a strategy which results in deadlock, and then if we derive a strategy that strategy must be deadlock free. It may very well be impossible to derive a strategy at all from some scenarios, if the scenario contains too many restrictions. The real work of AITL is then in creating proper restrictions based on what it is one wishes to prove, and then in deriving a strategy in keeping with

those restrictions.

A mayor weakness of this logic is that each action may only happen once, and must happen. For a real world action which may be repeatable, we must create in our logic an action for every time that real world action may occur. For this, we need to know beforehand exactly how many times an action will be taken. Consider the example of a game of golf, we can have the action of *playFirstHole* and *playSecondHole* and so on, with the action *visitClubhouse*. While a typical day on the course might start with the first hole, then progress from there in order, with a visit to the clubhouse after the ninth or eighteenth hole, we can play in any order. The only restriction is that we cannot play two holes at the same time. Various strategies can be generated for the order to play the holes in, and when to visit the clubhouse. But we assume each hole is only played once, and the clubhouse is only visited once. For any problems of this nature, where singular actions must be arranged, AITL works well. When we need to start reasoning about each individual stroke on a hole, we run into a problem, since we do not know beforehand how many strokes will be played. One solution might be to allow a maximum number of strokes, and have a *sinkHole* action happening after the hole is finished. If the player takes five strokes to sink the hole, we can arrange those strokes and afterwards have the *sinkHole* action. After this we may have all the unused strokes, but know that since they are after *sinkHole* they were not played. This is however an inelegant solution and such solutions do not always exist. We saw an example of this problem in terms of a repeatable action which had a maximum in the sleeping barber, as well as a repeatable action with no maximum in the trains problem.

This logic also lacks propositions, which both ATL and Allen and Ferguson's interval logic have. Propositions allow us to talk about specific states of the world, and properties of objects. With AITL, we lose this, but we gain a logic which is simple, elegant, and very expressive when it comes to talking about the exact order of events, of precisely what the relation is between all the actions which are carried out.

Chapter 7

Conclusion

7.1 Summary and Contribution

In this dissertation we introduced a new logic called Agent Interval Temporal Logic.

We started in chapter 2 by introducing preliminary concepts of logic, and looking broadly at the history of development from classical logic, to modal and then to temporal logic. We reviewed LTL and CTL, and covered concepts like agents, paths and strategies. Then we looked at the many types of temporal logics, their properties and how to distinguish them. We saw that we would have to choose between propositional or predicate logic and points or intervals when designing our logic. We also briefly reviewed the frame problem and similar problems with predication.

In chapter 3 we introduced three problems, which were used throughout the dissertation to illustrate the strengths and weaknesses of each logic. The first problem was the banker algorithm, which involved reasoning about the distribution of limited resources, and the careful allocation leading to a successful outcome. The second problem was the sleeping barber, which involved situations of overlapping events which might lead to infinite wait times. The last problem was a custom problem involving trains and conductors. This problem required precise timing by the agents to avoid trains going to incorrect destinations. All these problems involved multiple agents, and required complex timing.

In chapter 4 we looked at Alternating-Time Temporal Logic (ATL). We considered the syntax and semantics, as well as the variant ATL^*. We represented the three problems in ATL, and saw that while it could neatly represent all the agents and their actions, it lacked the notion of overlapping actions or interruptions. Next we looked at the related coalition logic, and some extensions of ATL including explicit strategies, strategic commitment, incomplete information and

epistemic extensions.

In chapter 5 we considered intervals. First we covered some preliminaries around intervals and relations, before introducing a few formalisations of interval logics. We looked at Halpern and Shoham's logic, at Moskowsky's ITL, at neighbourhood logic, and duration calculus. Finally, we looked at Allen and Ferguson's logic of actions and events, a first order logic involving pure intervals. We covered it in a lot of depth, and then applied it to the three problems. We saw that this logic was rather cumbersome, and involved a lot of axioms to represent rather simple situations. It does however provide a logic of pure intervals, and allows for some very complex timing to be represented.

In chapter 6 we finally introduced Agent Interval Temporal Logic, the main contribution of this dissertation. This is a new logic, taking concepts of intervals and concepts of agents and actions, and combining them together. The resulting logic could speak about actions, and the ordering of actions over time, by representing actions as intervals. We could represent various scenarios which contained actions to be done, and then we could derive various strategies of how to do those actions. There were also some limitations to our logic, which will be expanded upon in the next section.

This dissertation also contains an overview of the development of temporal logic, as well as covering various temporal logics and their relations to each other, which we believe can be very useful to a reader who is new to the field.

7.2 Applications

AITL is primarily a logic of arranging intervals while keeping to various restrictions on the arrangement of those intervals. It would be useful to any activity which involved the arrangement of actions. There are numerous fields in philosophy where this could be applied.

Most planning activities involve the arrangement of tasks and the assigning of responsibilities. Everything from self driving cars to social robots to automated factories, from project management to timetable generation, involves arranging intervals. As we have seen, AITL is not quite suited for situations where actions are optional, but rather where all the actions to be carried out are known beforehand. AITL would work well for a planning system, where a project is broken down into tasks (which must all be carried out) which are then assigned to teams and arranged on a timeline. Various applications in Artificial Intelligence involve similar planning activities.

Prediction and explanation is also a large part of the field of Artificial Intelligence. The arrangement of a set of past events can be used to determine cause and effect, while arranging future events can help predict how the world may look

if one thing is done before the other. Here again, AITL can be a useful formalism for reasoning about the arrangement of such events.

In the field of moral reasoning, one of the concerns involves reasoning as a group about moral questions. Richardson (2018) states that "such joint reasoning is best pursued as a matter of working out together, as independent moral agents, what they ought to do with regard to an issue on which they have some need to cooperate." If AITL were to be combined with some concepts from epistemic logic, group discussions could be represented. Different agents involved in the discussion could be represented, alongside their contributions to the conversation. The arrangement of arguments, such as this argument being before that counter-argument, could also be represented.

The field of mereology involves reasoning about "parthood relations: of the relations of part to whole and the relations of part to part within a whole" (Varzi, 2019). We have seen how intervals can be broken up into smaller intervals, and how smaller intervals can have relations to larger intervals and to other smaller intervals. While mereology considers many different types of parts, such as attached or detached, extended or unextended, spatial or temporal, AITL might be useful for reasoning about temporal parts. Having a representation for various intervals, and various parts of intervals, and relations between them, could contribute to the work done in mereology.

Many potential philosophical applications for AITL exist, and many more so when combined with elements from other logics. A lot more work can be done.

7.3 Future Work

While AITL could represent simple scenarios, and we could manually derive strategies from it, with larger and more complex scenarios this would become too much. There is still much potential for extending AITL to allow it to represent more complex scenarios in simpler ways. Perhaps not requiring every interval to be in the strategy, or having repeating intervals, or similar ideas, are all ideas still to be explored.

An automated algorithm based on Allen's interval algebra, which can be used to derive restrictions and strategies, would be very useful.

While this dissertation focused on pure interval logics, where intervals are primary objects in the logic, there is much potential for non-pure interval logics. A combination of ATL and HS, or ATL and ITL, are both interesting projects which may have great potential. Since ATL is based on states, adding intervals over those states and adding operators to be used on those intervals, could make ATL a much richer and more expressive language. There is also much more literature on non-pure interval logics, and it is computationally much simpler.

This dissertation merely scratched the surface of an unexplored topic, much more potential awaits in combining intervals with agents.

Appendix A

In this appendix we represent the Sleeping Barber and Trains problems using Allen and Ferguson's logic of actions and events.

The Sleeping Barber

In this problem, the issue of overlapping timelines becomes very important. As before, agents will not be directly represented, but their actions will be considered as part of the actions of a single system. There are four agents, the customers Alice, Bob and Charlie, and the Barber, and five chairs, numbered 0 to 4, as well as a sixth special chair for cutting. There are four actions leading to four events. They are *cut*, *enter*, *next*, *nap* and *wake*, leading to *CUT*, *ENTER*, *NEXT*, *NAP* and *WAIT*. As before, events are named in all *UPPERCASE*, actions in all *lowercase*, and predicates in *TitleCase*. *Cut* is where the barber cuts the hair of a customer. *Enter* is where a customer enters the shop and checks the chairs, and consists of three actions, one for each customer. *Next* is when the barber finishes a cut and calls the next customer. *Nap* is when the the barber goes to take a nap. *Wake* is when a customer tries to wake the barber, and three actions exist for each of the three customers. The predicates in the introduction description of this problem will be used, namely $Chair_i(a)$ for each of the five chairs $i = 0to4$, $BarberChair(a)$ for the customer receiving a haircut and $BarberStand(a)$ for the barber in position to cut.

The event definitions are as follows:

$$\begin{aligned}\forall e.CUT(e) \rightarrow \\ ((BarberChair(Alice, pre1(e)) \vee (BarberChair(Bob, pre1(e)) \\ \vee (BarberChair(Charlie, pre1(e))) \wedge SameEnd(pre1(e), time(e)) \\ \wedge BarberStand(Barber, pre2(e)) \wedge SameEnd(pre2(e), time(e)) \\ \wedge BarberChair(Empty, eff1(e) \wedge time(e) : eff1(e) \quad (\textbf{EDEF1})\end{aligned}$$

There needs to be a customer in the *BarberChair* and the barber needs to be in position behind the chair for a successful cutting to take place. Once completed, the customer walks out and disappears from the domain of discourse. The barber remains in position and should soon attempt the *next* action.

$$\begin{aligned}\forall e,i,j.ENTERALICE(e) \rightarrow \\ (Chair_0(Empty, pre1(e)) \wedge SameEnd(pre1(e), time(e)) \\ \wedge BarberStand(Barber, pre2(e)) \wedge SameEnd(pre2(e), time(e)) \\ \wedge Chair_0(Alice, eff1(e)) \wedge time(e) : eff1(e)) \\ \vee (Chair_i(Empty, pre1(e)) \wedge \neg Chair_j(Empty, pre1(e)) \\ \wedge SameEnd(pre1(e), time(e)) \wedge i = j + 1 \\ \wedge BarberStand(Barber, pre2(e)) \wedge SameEnd(pre2(e), time(e)) \\ \wedge Chair_i(Alice, eff1(e)) \wedge time(e) : eff1(e)) \quad \textbf{(EDEF2)}\end{aligned}$$

Either the first chair is empty, in which case it is taken, or another chair is empty which is after one that is not empty, in which case that one is taken. In both cases the barber must be at the cutting chair, if not the action fails and the customer tries the *wake* action instead. *EDEF3* and *EDEF4* are similarly defined for Bob and Charlie.

$$\begin{aligned}\forall e,a.NEXT(e) \rightarrow \\ (BarberChair(Empty, pre1(e)) \wedge SameEnd(pre1(e), time(e)) \\ \wedge BarberStand(Barber, pre2(e)) \wedge SameEnd(pre2(e), time(e)) \\ \wedge Chair_0(a, pre3(e)) \wedge a! = Empty \wedge SameEnd(pre3(e), time(e)) \\ \wedge BarberChair(a, eff1(e) \wedge time(e) : eff1(e) \quad \textbf{(EDEF5)}\end{aligned}$$

When the barber calls "Next!", the cutting chair must be empty, the barber must be at the cutting chair, and there must be someone in the first chair. If this succeeds, the customer who was in the first chair will walk over and end up in the cutting chair.

$$\begin{aligned}\forall e,i.NAP(e) \rightarrow BarberStand(Barber, pre1(e)) \wedge \\ SameEnd(pre1(e), time(e)) \wedge Chair_i(Empty, pre2(e)) \wedge \\ SameEnd(pre2(e)) \wedge BarberStand(Empty, eff1(e)) \wedge \\ time(e) : eff1(e) \wedge Chair_i(Barber, eff2(e)) \wedge time(e) : eff2(e) \quad \textbf{(EDEF6)}\end{aligned}$$

If the *next* action fails, probably because the first chair is empty, the barber then attempts to sleep. The barber must be at the cutting chair when attempting, and end up in one of the empty chairs when done.

$$\forall e, i.WAKEALICE(e) \rightarrow Chair_i(Barber, pre1(e)) \wedge SameEnd(pre1(e)) \wedge BarberStand(Barber, eff1(e)) \wedge time(e) : eff1(e) \wedge BarberChair(Alice, eff2(e)) \wedge time(e) : eff2(e) \quad (\textbf{EDEF7})$$

Alice wakes up the barber, then she takes a seat in the cutting chair and the barber stands at the cutting chair. *EDEF8* and *EDEF9* are similarly defined for Bob and Charlie.

The action axioms leading to the above events are now defined:

$$\forall t.Try(cut, t) \wedge \neg BarberChair(Empty, t) \wedge BarberStand(Barber, t) \rightarrow \exists e.CUT(t, e) \quad (\textbf{ETRY1})$$

$$\forall t.Try(enteralice, t) \wedge BarberStand(Barber, t) \wedge \exists i.Chair_i(Empty, t) \rightarrow \exists e.ENTERALICE(t, e) \quad (\textbf{ETRY2})$$

$$\forall t.Try(enterbob, t) \wedge BarberStand(Barber, t) \wedge \exists i.Chair_i(Empty, t) \rightarrow \exists e.ENTERBOB(t, e) \quad (\textbf{ETRY3})$$

$$\forall t.Try(entercharlie, t) \wedge BarberStand(Barber, t) \wedge \exists i.Chair_i(Empty, t) \rightarrow \exists e.ENTERCHARLIE(t, e) \quad (\textbf{ETRY4})$$

$$\forall t.Try(next, t) \wedge BarberChair(Empty, t) \wedge BarberStand(Barber, t) \wedge \neg Chair_0(Empty, t) \rightarrow \exists e.NEXT(t, e) \quad (\textbf{ETRY5})$$

$$\forall t.Try(nap, t) \wedge BarberStand(Barber, t) \wedge \exists i.Chair_i(Empty, t) \rightarrow \exists e.NAP(t, e) \quad (\textbf{ETRY6})$$

$$\forall t.Try(wakealice,t) \wedge \exists i.Chair_i(Barber,t) \rightarrow \exists e.WAKEALICE(t,e)$$

(**ETRY7**)

$$\forall t.Try(wakebob,t) \wedge \exists i.Chair_i(Barber,t) \rightarrow \exists e.WAKEBOB(t,e) \quad (\textbf{ETRY8})$$

$$\forall t.Try(wakecharlie,t) \wedge \exists i.Chair_i(Barber,t) \rightarrow \exists e.WAKECHARLIE(t,e)$$

(**ETRY9**)

Next the explanation closure axioms are defined.

$$\forall t,t',i.Chair_i(Empty,t) \wedge \neg Chair_i(Empty,t') \wedge t:t' \rightarrow \exists e.(ENTERALICE(e) \vee ENTERBOB(e) \vee ENTERCHARLIE(e) \vee NAP(e)) \wedge time(e):t' \quad (\textbf{EXCP1})$$

$$\forall t,t',i.\neg Chair_i(Empty,t) \wedge Chair_i(Empty,t') \wedge t:t' \rightarrow \exists e.(NEXT(e) \vee WAKEALICE(e) \vee WAKEBOB(e) \vee WAKECHARLIE(e)) \wedge time(e):t' \quad (\textbf{EXCP2})$$

$$\forall t,t'.BarberChair(Empty,t) \wedge \neg BarberChair(Empty,t') \wedge t:t' \rightarrow \exists e.NEXT(e) \wedge time(e):t' \quad (\textbf{EXCP3})$$

$$\forall t,t'.\neg BarberChair(Empty,t) \wedge BarberChair(Empty,t') \wedge t:t' \rightarrow \exists e.CUT(e) \wedge time(e):t' \quad (\textbf{EXCP4})$$

$$\forall t,t'.BarberStand(Empty,t) \wedge BarberStand(Barber,t') \wedge t:t' \rightarrow \exists e.(WAKEALICE(e) \vee WAKEBOB(e) \vee WAKECHARLIE(e)) \wedge time(e):t' \quad (\textbf{EXCP5})$$

$$\forall t, t'.BarberStand(Barber, t) \wedge BarberStand(Empty, t') \wedge t : t' \rightarrow \exists e.NAP(e) \wedge time(e) : t' \quad \textbf{(EXCP6)}$$

$$\forall e.CUT(e) \rightarrow \exists t.Try(cut, t) \wedge t = time(e) \quad \textbf{(EXCE1)}$$

$$\forall e.ENTERALICE(e) \rightarrow \exists t.Try(enteralice, t) \wedge t = time(e) \quad \textbf{(EXCE2)}$$

$$\forall e.ENTERBOB(e) \rightarrow \exists t.Try(enterbob, t) \wedge t = time(e) \quad \textbf{(EXCE3)}$$

$$\forall e.ENTERCHARLIE(e) \rightarrow \exists t.Try(entercharlie, t) \wedge t = time(e) \quad \textbf{(EXCE4)}$$

$$\forall e.NEXT(e) \rightarrow \exists t.Try(next, t) \wedge t = time(e) \quad \textbf{(EXCE5)}$$

$$\forall e.NAP(e) \rightarrow \exists t.Try(nap, t) \wedge t = time(e) \quad \textbf{(EXCE6)}$$

$$\forall e.WAKEALICE(e) \rightarrow \exists t.Try(wakealice, t) \wedge t = time(e) \quad \textbf{(EXCE7)}$$

$$\forall e.WAKEBOB(e) \rightarrow \exists t.Try(wakebob, t) \wedge t = time(e) \quad \textbf{(EXCE8)}$$

$$\forall e.WAKECHARLIE(e) \rightarrow \exists t.Try(wakecharlie, t) \wedge t = time(e) \quad \textbf{(EXCE9)}$$

With that, all the necessary general axioms have been defined for the sleeping barber. Now a specific case is presented:

$$t0 < t1 < t2 < t3 < t4 < t5 < t6 < t7 < t8 < t9 < t10 \quad \textbf{(AX0)}$$

$$Chair_0(Alice, t0) \quad \textbf{(AX1)}$$

$$Chair_1(Empty, t0) \quad \textbf{(AX2)}$$

$$Chair_2(Empty, t0) \quad \textbf{(AX3)}$$

$$Chair_3(Empty, t0) \quad \textbf{(AX4)}$$

$$Chair_4(Empty, t0) \quad \textbf{(AX5)}$$

$$BarberChair(Empty, t0) \quad \textbf{(AX6)}$$

$$BarberStand(Barber, t0) \quad \textbf{(AX7)}$$

$$Try(next, t1) \quad \textbf{(AX8)}$$

$$Try(enterbob, t2) \quad \textbf{(AX9)}$$

$$Try(cut, t3) \quad \textbf{(AX10)}$$

$$Try(next, t4) \quad \textbf{(AX11)}$$

$$Try(cut, t5) \quad \textbf{(AX12)}$$

$$Try(next, t6) \quad \textbf{(AX13)}$$

$$Try(nap, t7) \quad \textbf{(AX14)}$$

$$Try(entercharlie, t8) \quad \textbf{(AX15)}$$

$$Try(wakecharlie, t9) \quad \textbf{(AX16)}$$

$$Try(cut, t10) \quad \textbf{(AX17)}$$

It is clear that this case will work out fine, with all actions succeeding except for $Try(next, t6)$ at $AX13$, which fails since there is no one next in the queue, leading to the barber taking a nap during $t7$. The problem comes in when the times for *nap* and *enter* actions overlap. As an example, here is a subsection of axioms:

$$Try(next, t1) \quad \textbf{(AX1)}$$

Assume this attempt fails, as there are no customers waiting.

$$Try(nap, t2) \quad \textbf{(AX2)}$$

$$Try(entercharlie, t3) \quad \textbf{(AX3)}$$

If $t2$ starts before $t3$, but the two times overlap, this will lead to a problem. Since *nap* requires that the barber is at their stand and that there is an open chair, it will succeed. At the start of *entercharlie*, the barber is still at their stand, and $chair_0$ is open, so Charlie takes a seat at $chair_0$, while the barber goes to nap on $chair_2$.

The sleeping barber problem has been represented in Allen and Ferguson's logic, and now the last problem, the trains problem, will be presented.

The Trains Problem

A simple example will be used, consisting of 3 trains (t_0, t_1, t_2), 3 stations (s_0, s_1, s_2), 5 pieces of rail track $(r_0, r_1, r_2, r_3, r_4)$, 2 crossroads (c_0, c_1) and a single agent (*agent*). The trains will arrive onto r_0, which connects to crossroad c_0, which splits to r_1 and r_2. The rail r_1 terminates in station s_1, while $r2$ connects to crossroad c_1, which splits to r_3 and r_4. The rail r_3 terminates in s_2 while the rail r_4 terminates in s_3. A useful function will be $next(r)$ which returns the track, crossroad or station which connects to the current track or crossroad r, taking into account the current setting of a crossroad. For example, $next(r_2) = c_1$.

To represent the state of the system, the following predicates will be used: $Crossroad_i(a)$ will indicate if an agent is at a crossroad, where a is either *Agent* or *Empty*, $CrossConnection_i(r_j)$ will indicate the current setting of a crossroad, where r_j is a piece of railway, $On_i(t_j)$ will indicate the presence of a train on a track i, which may be either rail tracks or a crossroad. Note that the agent may only be at one of the two crossroads at a time, and must run between them, and a crossroad may only be in one setting at a time, while a track may contain many trains. For example, if $Crossroad_0(Agent)$ then $Crossroad_1(Empty)$, and if $CrossConnection_1(r_3)$ then $\neg CrossConnection_1(r_4)$. However, it may be the case that $On_r2(t_0)$ and $On_r2(t_1)$ at the same time.

The agent can take the actions *switch* to change a crossroad and *run* to get to the other crossroad. When a train reaches the end of the current section of track

or crossroad the system will take the $advance_i$ action, which will put the train on the next track. There will be one $advance_i$ action for each i train.

The event axioms are as follows:

$$\forall e, r, r`.SWITCH(e) \rightarrow \exists i.Crossroad_i(Agent, pre1(e)) \\ \wedge SameEnd(pre1(e), time(e)) \wedge CrossConnection_i(r, pre2(e)) \\ \wedge SameEnd(pre2(e), time(e)) \wedge CrossConnection_i(r', eff1(e)) \\ \wedge time(e) : eff1(e) \quad \textbf{(EDEF1)}$$

$$\forall e, i, j.RUN(e) \rightarrow Crossroad_i(Agent, pre1(e)) \wedge SameEnd(pre1(e), time(e)) \\ \wedge Crossroad_j(Empty, pre2(e)) \wedge SameEnd(pre2(e), time(e)) \\ \wedge Crossroad_i(Empty, eff1(e)) \wedge time(e) : eff1(e) \\ \wedge Crossroad_j(Agent, eff2(e)) \wedge time(e) : eff2(e) \\ \wedge i! = j \quad \textbf{(EDEF2)}$$

The agent can flip a switch at a crossroad or run across to the other crossroad.

$$\forall e, c, i, j.ADVANCE_c(e) \rightarrow On_i(t_c, pre1(e)) \wedge SameEnd(pre1(e), time(e)) \wedge \\ On_j(t_c, eff1(e)) \wedge time(e) : eff1(e) \wedge i! = j \wedge next(i) = j \quad \textbf{(EDEF3)}$$

The train car c advances from i to j, keep in mind that i and j can be rail track pieces, crossroads or stations.

The action axioms are as follows:

$$\forall t.Try(switch, t) \wedge \exists i.Crossroad_i(Agent, t) \rightarrow \exists e.SWITCH(t, e) \quad \textbf{(ETRY1)}$$

$$\forall t, i, j.Try(run, t) \wedge Crossroad_i(Agent, t) \\ \wedge Crossroad_j(Empty, t) \rightarrow \exists e.RUN(t, e) \quad \textbf{(ETRY2)}$$

$$\forall t, i, c.Try(advance_c, t) \wedge On_i(c, t) \rightarrow \exists e.ADVANCE_i(t, e) \quad \textbf{(ETRY3)}$$

The explanation closure axioms are as follows:

$$\forall t, t', i. Crossroad_i(Agent, t) \wedge Crossroad_i(Empty, t') \wedge t : t' \rightarrow \exists e. RUN(e) \wedge time(e) : t' \quad \textbf{(EXCP1)}$$

$$\forall t, t', i. Crossroad_i(Empty, t) \wedge Crossroad_i(Agent, t') \wedge t : t' \rightarrow \exists e. RUN(e) \wedge time(e) : t' \quad \textbf{(EXCP2)}$$

$$\forall t, t', i, r, r'. CrossConnection_i(r, t) \wedge CrossConnection_i(r', t') \wedge t : t' \wedge r! = r' \rightarrow \exists e. SWITCH(e) \wedge time(e) : t' \quad \textbf{(EXCP3)}$$

$$\forall t, t', i, c. On_i(c, t) \wedge \neg On_i(c, t') \wedge t : t' \rightarrow \exists e. ADVANCE_c(e) \wedge time(e) : t' \quad \textbf{(EXCP4)}$$

$$\forall t, t', i, c. \neg On_i(c, t) \wedge On_i(c, t') \wedge t : t' \rightarrow \exists e. ADVANCE_c(e) \wedge time(e) : t' \quad \textbf{(EXCP5)}$$

$$\forall e. RUN(e) \rightarrow \exists t. Try(run, t) \wedge t = time(e) \quad \textbf{(EXCE1)}$$

$$\forall e. SWITCH(e) \rightarrow \exists t. Try(switch, t) \wedge t = time(e) \quad \textbf{(EXCE2)}$$

$$\forall e. ADVANCE(e) \rightarrow \exists t. Try(advance, t) \wedge t = time(e) \quad \textbf{(EXCE3)}$$

The failure condition, that a train ends up in the wrong station, can be represented by the following:
$(On_{s_i} = t_j) \wedge i! = j$
While the win condition is:
$\forall i. (On_{s_i} = t_i)$

Bibliography

Ågotnes, T. (2006). Action and knowledge in alternating-time temporal logic, *Synthese* **149**(2): 375–407.

Ågotnes, T., Goranko, V. and Jamroga, W. (2007). Alternating-time temporal logics with irrevocable strategies, *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge*, pp. 15–24.

Ågotnes, T., Goranko, V. and Jamroga, W. (2008). Strategic commitment and release in logics for multi-agent systems, *Technical report*, Clausthal University of Technology.

Ågotnes, T. and Walther, D. (2009). A logic of strategic ability under bounded memory, *Journal of Logic, Language and Information* **18**(1): 55–77.

Allen, J. F. (1990). Maintaining knowledge about temporal intervals, *in* M. Kaufmann (ed.), *Readings in Qualitative Reasoning about Physical Systems*, Elsevier, pp. 361–372.

Allen, J. F. and Ferguson, G. (1994). Actions and events in interval temporal logic, *Journal of Logic and Computation* **4**(5): 531–579.

Allen, J. F. and Hayes, P. J. (1985). A common-sense theory of time, **1**: 528–531.

Alur, R., Henzinger, T. A. and Kupferman, O. (2002). Alternating-time temporal logic, *Journal of the ACM (JACM)* **49**(5): 672–713.

Barua, R. and Chaochen, Z. (1997). Neighbourhood logics: Nl and nl2, *Technical Report 120*, United Nations University / International Institute for Software Technology.

Barua, R., Roy, S. and Chaochen, Z. (2000). Completeness of neighbourhood logic, *Journal of Logic and Computation* **10**(2): 271–295.

Bellini, P., Mattolini, R. and Nesi, P. (2000). Temporal logics for real-time system specification, *ACM Computing Surveys (CSUR)* **32**(1): 12–42.

Ben-Ari, M., Pnueli, A. and Manna, Z. (1983). The temporal logic of branching time, *Acta Informatica* **20**(3): 207–226.

Brihaye, T., Costa, A. D., Laroussinie, F. and Markey, N. (2009). Atl with strategy contexts and bounded memory, *International Symposium on Logical Foundations of Computer Science*, pp. 92–106.

Bulling, N., Goranko, V. and Jamroga, W. (2015). Logics for reasoning about strategic abilities in multi-player games, *in* J. van Benthem (ed.), *Models of Strategic Reasoning*, Springer, Berlin, Heidelberg, pp. 93–136.

Bulling, N. and Jamroga, W. (2014). Comparing variants of strategic ability: How uncertainty and memory influence general properties of games, *Autonomous Agents and Multi-Agent Systems* **28**(3): 474–518.

Bulling, N., Jamroga, W. and Dix, J. (2008). Reasoning about temporal properties of rational play, *Annals of Mathematics and Artificial Intelligence* **53**(1-4): 51–114.

Chaochen, Z. (1999). Duration calculus, a logical approach to real-time systems, *International Conference on Algebraic Methodology and Software Technology*.

Chaochen, Z. and Hansen, M. (1997). An adequate first order interval logic, *in* W. de Roever, H. Langmaack and A. Pnueli (eds), *Compositionality: The Significant Difference*, pp. 584–608.

Chaochen, Z., Hansen, M. and Sestoft, P. (1993). Decidability and undecidability results for duration calculus, *Annual Symposium on Theoretical Aspects of Computer Science*, pp. 58–68.

Chaochen, Z., Hoare, C. and Ravn, A. (1991). A calculus of durations, *Information Processing Letters* **40**(5): 269–276.

Davidson, D. (1967). The logical form of action sentences, *in* N. Rescher (ed.), *The Logic of Decision and Action*, University of Pittsburgh Press.

De Masellis, R., Goranko, V., Gruner, S. and Timm, N. (2019). Generalising the dining philosophers problem: Competitive dynamic resource allocation in multi-agent systems, *European Conference on Multi-Agent Systems*, pp. 30–47.

Dijkstra, E. W. (1968). Cooperating sequential processes, *in* P. B. Hansen (ed.), *The Origin of Concurrent Programming*, Springer, New York, pp. 65–138.

Dijkstra, E. W. (1982). The mathematics behind the banker's algorithm, pp. 308–312.

Downey, A. B. (2008). *The Little Book of Semaphores*, Green Tea Press.

Duan, Z. (1996). *An Extended Interval Temporal Logic and A Framing Technique for Temporal Logic Programming*, PhD thesis, University of Newcastle.

Dutertre, B. (1995). Complete proof systems for first order interval temporal logic, *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pp. 36–43.

Emerson, E. A. (2008). The beginning of model checking: A personal perspective, *in* O. Grumberg and H. Veith (eds), *25 Years of Model Checking*, Springer, Berlin, Heidelberg, pp. 27–45.

Emerson, E. A. and Clarke, E. M. (1982). Using branching time temporal logic to synthesise synchronisation skeletons, *Science of Computer Programming* **2**(3): 241–266.

Emerson, E. A. and Halpern, J. Y. (1985). Decision procedures and expressiveness in the temporal logic of branching time, *Journal of Computer and Systems Science* **30**(1): 1–24.

Fagin, R., Halpern, J., Moses, Y. and Vardi, M. (1995). *Reasoning About Knowledge*, MIT Press, London.

Garson, J. (2018). Modal logic, *in* E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, fall 2018 edn.
URL: *https://plato.stanford.edu/archives/fall2018/entries/logic-modal/*

Goldman, A. I. (1970). *A Theory of Human Action*, Princeton University Press.

Goranko, V. (2001). Coalition games and alternating temporal logics, *Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge*, pp. 259–272.

Goranko, V. and Galton, A. (2015). Temporal logic, *in* E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, winter 2015 edn.
URL: *https://plato.stanford.edu/archives/win2015/entries/logic-temporal/*

Goranko, V., Kuusisto, A. and Rönnholm, R. (2018). Game-theoretic semantics for alternating-time temporal logic, *ACM Transactions on Computational Logic (TOCL)* **19**(3): 17.

Goranko, V., Montanari, A. and Sciavicco, G. (2003). Propositional interval neighborhood temporal logics, *Journal of Universal Computer Science* **9**(9): 1137–1167.

Goranko, V., Montanari, A. and Sciavicco, G. (2004). A road map of interval temporal logics and duration calculi, *Journal of Applied Non-Classical Logics* **14**(1-2): 9–54.

Green, C. (1969). An application of theorem proving to problem solving, *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pp. 219–239.

Guelev, D. P. (1998). A calculus of durations on abstract domains: Completeness and extensions, *Technical Report 139*, United Nations University / International Institute for Software Technology.

Haas, A. R. (1987). The case for domain-specific frame axioms, *in* F. M. Brown (ed.), *The Frame Problem in Artificial Intelligence*, Morgan Kaufmann - Elsevier, pp. 343–348.

Halpern, J., Manna, Z. and Moszkowski, B. (1983). A hardware semantics based on temporal intervals, *in* J. Diaz (ed.), *Proceedings of the 10th International Colloquium on Automata, Languages and Programming*, pp. 278–291.

Halpern, J. Y. and Shoham, Y. (1991). A propositional modal logic of time intervals, *Journal of the ACM (JACM)* **38**(4): 935–962.

Hansen, M. and Chaochen, Z. (1992). Semantics and completeness of duration calculus, *in* J. W. de Bakker, C. Huizing, W. P. de Roever and G. Rozenberg (eds), *Real-Time: Theory in Practice*, pp. 209–225.

Hoare, C. A. R. (1978). Communicating sequential processes, *Communications of the ACM* **21**(8): 666–677.

Jamroga, W. (2003). Some remarks on alternating temporal epistemic logic, *in* B. Dunin-Keplicz and L. C. Verbrugge (eds), *Proceedings of the 1st International Workshop on Formal Approaches to Multi-Agent Systems*, pp. 133–140.

Jamroga, W. and Ågotnes, T. (2007). Constructive knowledge: What agents can achieve under incomplete information, *Journal of Applied Non-Classical Logics* **17**(4): 423–475.

Jamroga, W. and van der Hoek, W. (2004). Agents that know how to play, *Fundamenta Informaticae* **63**(2-3): 185–219.

Jamroga, W., Wooldridge, M. and van der Hoek, W. (2005). Intentions and strategies in game-like scenarios, *in* C. Bento, A. Cardoso and G. Dias (eds), *Proceedings of the 12th Portuguese Conference on Progress in Artificial Intelligence*, pp. 512–523.

Kent, K. C. (2019). Propositional logic, *The Internet Encyclopedia of Philosophy*. **URL:** *https://www.iep.utm.edu/prop-log/*

Kripke, S. (1963). Semantical considerations of the modal logic, *Acta Philosophica Fennica* **16**: 83—-94.

McCarthy, J. and Hayes, P. (1981). Some philosophical problems from the standpoint of artificial intelligence, *in* B. L. Webber and N. J. Nilsson (eds), *Readings in Artificial Intelligence*, Morgan Kaufmann - Elsevier, pp. 431–450.

Moszkowski, B. (1983a). *Reasoning about Digital Circuits*, PhD thesis, Stanford University.

Moszkowski, B. (1983b). Reasoning in interval temporal logic, *in* E. Clarke and D. Kozen (eds), *Workshop on Logic of Programs*, pp. 371–382.

Moszkowski, B. (1984). Executing temporal logic programs, *Technical Report 55*, University of Cambridge.

Moszkowski, B. (1996). The programming language tempura, *Journal of Symbolic Computation* **22**(5/6): 730–733.

Pauly, M. (2002). A modal logic for coalitional power in games, *Journal of Logic and Computation* **12**(1): 149–166.

Pnueli, A. (1977). The temporal logic of programs, *Proceedings of the 18th IEEE Annual Symposium on Foundations of Computer Science*, pp. 46—-57.

Prior, A. (1967). *Past, Present and Future*, Oxford: Oxford University Press.

Prior, A. N. (1957). *Time and Modality*, Oxford: Oxford University Press.

Pujari, A. K. (1997). Neighbourhood logic & interval algebra, *Technical Report 116*, United Nations University / International Institute for Software Technology.

Richardson, H. S. (2018). Moral reasoning, *in* E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, fall 2018 edn.
URL: *https://plato.stanford.edu/archives/fall2018/entries/reasoning-moral/*

Ross, D. (2019). Game theory, *in* E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, spring 2019 edn.
URL: *https://plato.stanford.edu/archives/spr2019/entries/game-theory/*

Roy, S. and Chaochen, Z. (1997). Notes on neighbourhood logic, *Technical Report 97*, United Nations University / International Institute for Software Technology.

Schobbens, P. Y. (2004). Alternating-time logic with imperfect recall, *Electronic Notes in Theoretical Computer Science* **85**(2): 82–93.

Schubert, L. (1990). Monotonic solution of the frame problem in the situation calculus, *in* H. E. K. Jr., R. P. Loui and G. N. Carlson (eds), *Knowledge Representation and Defeasible Reasoning*, Dordrecht: Springer, pp. 23–68.

Shields, C. (2016). Aristotle, *in* E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, winter 2016 edn.
URL: *https://plato.stanford.edu/archives/win2016/entries/aristotle/*

Shoham, Y. (1987). *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence*, PhD thesis, Yale University.

Troquard, N. and Walther, D. (2012). On satisfiability in atl with strategy contexts, *13th European Workshop on Logics in Artificial Intelligence*, Vol. 7519, pp. 398–410.

van der Hoek, W., Jamroga, W. and Wooldridge, M. (2005). A logic for strategic reasoning, *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 157–164.

van der Hoek, W. and Wooldridge, M. (2003). Cooperation, knowledge and time: Alternating-time temporal epistemic logic and its applications, *Studia Logica* **75**(1): 125–157.

Varzi, A. (2019). Mereology, *in* E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, spring 2019 edn.
URL: *https://plato.stanford.edu/archives/spr2019/entries/mereology/*

Venema, Y. (1990). Expressiveness and completeness of an interval tense logic, *Notre Dame Journal of Formal Logic* **31**(4): 529–547.

Walther, D., van der Hoek, W. and Wooldridge, M. (2007). Alternating-time temporal logic with explicit strategies, *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge*, pp. 269–278.