Symbolic Model Checking for Natural Strategies

Rabih Chamas

June 2022

Abstract

Symbolic model checking is an automatic verification tool to check for specifications in a state transition diagram, by representing the transition system implicitly using Binary Decision Diagrams (BDDs). The BDD data structure is a tree diagram that represents a boolean function. MCMAS is an OBDD-based symbolic model checker for multi-agent systems. It supports the semantics of ATL (Alternating-time temporal logic), which is a logical formalism for reasoning about strategies. The purpose of my thesis is to extend MCMAS software to check for natural strategies of certain complexities. A natural strategy is a set of couples of actions guarded by propositional formulas. A group of agents follows a natural strategy, which means they apply an action in the state where the condition guarding the action holds. The complexity of a propositional formula is proportional to the number of symbols and logical operations appearing in the formula, and the complexity of a natural strategy is the summation of complexities of the propositional formulas that exist in it. Accordingly, to execute the extension, we had to find an algorithm that gets the complexity of a propositional formula, then use it in the algorithm that checks for natural strategies. In this thesis, we follow a theoretical approach to explain the main ideas of symbolic model checking, and we introduce two algorithms that check for different complexities of propositional formulas represented by BDDs, under some restrictions. Finally, we propose an algorithm to check for natural strategies. The algorithms are reinforced by the theory, yet only one algorithm, for formula complexity, was implemented and verified on numerous examples.

Chapter 1

Introduction

MCMAS is an OBDD-based symbolic model checker for multi-agent systems, that considers combinatorial strategies and does not address natural strategies. In this work, our goal is to provide the ground for extending MCMAS to model-check the strategic abilities of groups of agents (coalitions) whose behavior is described by natural strategies with bounded complexity. For doing so, we propose a new algorithm for calculating the complexity of propositional formulas, which can be further extended to capture the complexity of natural strategies.

In the introduction we clarify the model checking problem on a state transition diagram, discuss the semantics of CTL (Computation tree logic) and ATL logic, and give the definitions of natural strategies with different complexity interpretations. In the second chapter, we give the necessary definitions, theorems, and proofs of boolean algebra that serve as prerequisites for our results, and we explain the theory of symbolic model checking and MCMAS algorithms. In chapter 3 we present the algorithms which are the main results of the thesis.

Therac-25 overdosing error (1985-87) was the cause of the death of three cancer patients when the radiation machine for cancer treatment Therac-25 gave patients massive overdoses of radiation. In the period 1985-1987, at least six overdose cases were reported with an intensity of about 100 times the regular one. It came out after investigating the whole system, that the source of the error was a bug in the software of the machine. This accident highlighted how incorrectness in a safety-critical system may lead to catastrophic results. There are many techniques to verify rigorously that a control system has no mistakes, one of the approved techniques is model checking. Moreover, with the growth of distributed and decentralized AI, a functional automated verification tool for the multi-agent system is becoming more and more crucial for verifying the correctness, safety, or fairness of designs.

The idea of model checking is to model our control system with directed graphs (State transition diagram), and then use logical reasoning on the graphs and check for specifications that contain for example existence of strategies under some constraints or avoidance of states that represent system crash. For that let us introduce the state transition diagram.

1.1 State transition diagram and temporal logic

Let L be a set of states and R be a universal relation in L. A state transition diagram $M = (L, R, Atm, V, I)$ is the directed graph which is generated starting from an initial set $I \subseteq L$, and following the relation $R \subseteq L \times L$, in which sRs' means that there is a directed edge from s to s', and an evaluation function $V : Atm \to 2^L$ that associate each atomic proposition in Atm to a subset of L. We call the nodes of our graph the reachable states and we denote them by *reach*.
In the diagram, transition (i.e. an edge) may represent the successor state, giving a temporal semantics to M, so this model offers a meaning for a temporal logic statement to be true in a state. Given a model $M = (L, R, Atm, V, I)$ the language that is generated from atomic propositions in Atm, by propositional and temporal operators, is called temporal logic. Formally a formula ϕ in branching temporal logic is recursively defined as follows:

$$\phi \ ::= \ p \mid \neg\phi \mid \phi \wedge \phi \mid EX\phi \mid \phi \ EU \ \phi \mid EG \ \phi.$$

Where $p \in Atm$ is an atomic proposition, E stands for there exists, and the classical temporal operators that appear here are "X"(in the next state), "U"(Until), and "G"(from now on). Next, we give the formal definitions of these temporal operators, we use the notation $M, s \models \phi$ to mean that a formula ϕ is true in state s.

$M, s \models EX\phi$ if and only if there exist $s' \in L$ such that $(s, s') \in R$ and $M, s' \models \phi$.

$M, s \models \psi EU\phi$ if and only if there exists a path starting from s, where ϕ will eventually be true and ψ is true along all the path until ϕ is true.

$M, s \models EG\phi$ if and only if there exists an infinite path starting from s, where ϕ is true over the path (Here path is a sequence of nodes $(s_1, ..., s_k)$, such that there exists an edge directed from s_i to s_{i+1} for all $1 \leq i \leq k$). Note that loops can bring out infinite paths and since R is a universal relation then any node could be a source of an infinite path.

The propositional operations (logical and, and negation) $\wedge$ and $\neg$ on the atomic propositions are defined by their truth-values in states as follows:
Let ϕ,$\psi \in Atm$ and V be the evaluation function given with the model,

$M, s \models (\phi_1 \wedge \psi_2)$ if and only if $s \in V(\phi_1) \cap V(\phi_2)$.

$M, s \models (\neg \phi)$ if and only if $s \in V(\phi)^c$.

From now on we will use the same notation for logical formulas and the set of states where the formula is true, so when we mention a logical formula ϕ we refer to the set of states where ϕ is true. We say that a formulas ϕ is true in the model M if

ϕ is true in all reachable states i.e., $\phi \cap reach = reach$ ($reach$ is the set of reachable states). And we say ϕ is true in $S \subseteq L$ if $S \cap reach \subseteq \phi \cap reach$.

1.2 Concurrent game structure and ATL Logic

A concurrent game structure $\{(L_i, Act_i, Pr_i, Ev_i, Atm, V, In)\}_{1 \leq i \leq n}$ is a particular case of state transition diagrams that model a multi-agent system (mas) with n number of agents. So it defines a diagram where nodes are the reachable global states and edges are identified with joint actions which we will define soon.
local and global states For each agent i, L_i stands for the set of local state of the agent, each local state is characterised uniquely by finite values of finite variables that only agent i can observe and change. For example if agent i has two boolean variables that only she can observe, then she has four local states $(1,1), (1,0), (0,1), (0,0)$. The set of global states $L = \bigtimes L_i$ is the product set of all local states of all agents.
Actions and joint Actions Act_i is the finite set of actions that agent i can apply. The set of joint actions $Act = \bigtimes Act_i$ is the product set of all set of actions over all the agents, so each joint action $(a_1, ..., a_n)$ represents the actions took by all agents at once.
The environment agent The environment is a special agent which have some of it's variables observable by all agents and some observable to some specific agents. Now, for an agent i let L^i_E represent the state of variables associated to the environment that agent i can observe, we call $L_i \times L^i_E$ the set of extended local states of agent i.
The protocol function The protocol function $Pr_i : L_i \times L^i_E \to 2^{Act_i}$ is a rule that associate to each extended local state a set of actions that the agent i is allowed to play in that local state.
The Evolution function Let $B_i \subseteq (L_i \times L^i_E \times A_i)$ be the subset of state action pairs which are compatible with the protocol, formally $(s, a) \in B_i$ if and only if $a \in Pr_i(s)$. The evolution function $Ev_i : L_i \times L^i_E \times Act \to L_i$ is the function that represent the transition relation between local states. So the change of local state of each agent depend on the global action and the extended local state of the agent.
Global protocol and evolution function To define the transition relation on global states, we need a definition for global evolution function. Let $proj_{L_i} : L \to L_i$ be the projection map, we define the global protocol function $Pr : L \to 2^{Act}$ as follows: $Pr(s) = \cap Pr_i(proj_{L_i \times L^i_E}(s))$. And the global evolution function $Ev : B \subseteq L \times Act \to L$ is defined by: $Ev(s) = s'$ iff $Ev_i(proj_{L_i \times L^i_E}(s)) = proj_{L_i}(s')$, where $(s, a) \in B$ iff $a \in Pr(s)$.
So in this model the reachable states which forms the nodes of the graph are subset of the set of global states L, and the transitive relation R is defined on L by, sRs' iff there exist a global action a such that $Ev(s, a) = s'$.

The kind of specifications we want to verify in a multi-agent system involves checking whether a group of agents can reach a certain state regardless of the actions taken by other agents. This brings out the concept of strategy.

The concurrent game structure gives a nice framework to define strategies and reasons for them. The Alternating-time temporal logic (ATL) is an extension of the temporal logic in which statements about strategies are expressed. The language of ATL is defined as follows:

$$\phi \ ::= \ p \mid \neg\phi \mid \phi \wedge \phi \mid EX\phi \mid \phi \ EU \ \phi \mid EG \ \phi \mid \langle\langle \mathrm{T} \rangle\rangle X\phi \mid \langle\langle \mathrm{T} \rangle\rangle \phi U \phi \mid \langle\langle \mathrm{T} \rangle\rangle G\phi.$$

$\langle\langle \mathrm{T} \rangle\rangle X\phi$ reads, group T have a strategy to reach ϕ in the next state. Formally, given a model M $= \{(L_i, Act_i, Pr_i, Ev_i, Atm, V, In)\}_{1\leq i\leq n}$, let L be the set of global states and $s \in L$, T be a group of agents, and let Act_{T} and $Act_{out\mathrm{T}}$ denote the set of joint actions of agents inside and outside T respectively. The semantics of ATL is defined as follows:

$M, s \models \langle\langle \mathrm{T} \rangle\rangle X\phi$ if and only if there exist $a_{in} \in Act_{\mathrm{T}}$, such that for every $a_{out} \in Act_{out\mathrm{T}}$ if $a = (a_{in}, a_{out}) \in Pr(s)$, then $M \ , s' \models \phi$, where $s' = Ev(s, a)$.

$M, s \models \langle\langle \mathrm{T} \rangle\rangle G\phi$ if and only if $M, s \models \phi$ and there exist a path $(s, s_1, s_2, ...)$ starting from s, such that ϕ is true along the path, and for every state s_i of the path, the group T has a strategy to reach s_{i+1}
.

$M, s \models \langle\langle \mathrm{T} \rangle\rangle \psi U\phi$ if and only if there exists a path starting from s, where ψ is true in s and ϕ is true at some state on the path, and T can apply a joint action to maintain ψ along the path until they reach ϕ.

1.3 Natural strategies

A strategy for an agent i on a concurrent game structure could be defined as a set of couples, each couple is formed of a sequence of states or a path, and an action that is compatible with the protocol of the agent in the last state of the sequence. Intuitively, the sequence of states represents the history of the game, in this context the last state of the sequence is the current state where the action has to be applied. The semantics of ATL defined in the last section is restricted to memoryless strategies, in which the action only depends on one state, which is the last state in history. Thus henceforth, a strategy will be considered as a set of couples, of state actions.

Given a concurrent game structure, since a strategy is a relation that corresponds a state to a unique action, a memoryless strategy of a group of agents T could be seen as a map from a subset of global states to the set of joint actions of T.

Now consider for example a multi-agent system model that is applied for medical protocol supervision, (see $[Als + 00]$). In this domain, one deals with strategies, that are practical for human interference, which are sequences of instructions like "If exposure to confined covid-19 case OR travel to an area with confirmed covid-19 cases, then apply isolation". So forth, we want to express strategies in a more "natural" way than just a mapping or a relation. One representation of strategies is a sequence of actions guarded by propositional formulas, made by atomic propositions combined with logical operators. Strategies that are represented in this way are called **Natural strategies**.

Moreover, we can define the complexity of natural strategies. There are several ways to define the complexity of a natural strategy. Intuitively, the more complex the strategy is, the harder it becomes to understand or memorize the strategy. So the definition given should meet this criterion. We will be dealing with two types of complexities introduced in [JMM19], the first is the number of atomic propositions that exist in the formulas, and the other is the number of atomic propositions added to the number of propositional operators (not, and, or) used in each of the formulas. In practice, some strategies do not have natural representations, and some have natural representations of different complexities. In a multi-agent system, we are concerned with checking for natural strategies of certain complexity for a group of agents, to meet some specific properties. The notation $\langle\langle \mathrm{T} \rangle\rangle^{\leq k}\psi$ denotes that there exists a natural strategy of complexity less than k for the group T, to enforce the property ψ.

Chapter 2

Boolean Algebra

2.1 Boolean Algebra

The breakthrough made by symbolic model checking is that it provides a method of state-space search that does not explicitly represent the states of the model, instead, it provides implicit representation using boolean functions, which is stimulated by a binary decision diagram (BDD)[Min96]. The BDD data structure is a tree that represents a boolean function, each BDD has a compressed unique canonical form called reduced binary decision diagram (ROBDD), we can get the information that we need from the model by applying arithmetic operations directly on the canonical reduced form[LMc93].
Boolean functions are elements of a boolean algebra. Using the theory of boolean algebra, in this chapter, we will try to verify the identification between states, sets of states, and relations from one side and boolean function from the other side, as elements of a boolean algebra. Moreover, the theorems represented in this chapter will also provide an algorithm to compute the complexity of a propositional logical formula in its reduced form.

Definition 2.1.1. *A set A together with two binary operations $+$ and $*$ form a boolean algebra if $(A, +, *, 1, 0)$ have the structure of commutative unitary semiring and for every $a \in A$ we have:*
• *$a + a = a$ and $a * a = a$.*
• *there exist a unique element denoted by $\bar{a}$, such that $a + \bar{a} = 1$ and $a * \bar{a} = 0$.*

We call the image of the unitary operation $\neg : a \to \bar{a}$ the negation of a. It comes directly from the definition that $\bar{\bar{a}} = a$.

Example 2.1.1. *let S be any set, and 2^S be it's power set. The system $(2^S, \cup, \cap, c, S, \{\})$ is a boolean algebra, where c is the complement operator on 2^S.*

Theorem 2.1.1. *De Morgan's Theorem*
Let A be a boolean algebra and a,b $\in$ A then, $a \bar{+} b = \bar{a} * \bar{b}$.

Moreover, from the fact that $\bar{\bar{a}} = a$ we can deduce that $a \bar{*} b = \bar{a} + \bar{b}$.

Example 2.1.2. *Let A be a boolean algebra and X be any element of A. Consider the equivalence relation $a \sim_X b$ if and only if $X * a = X * b$. We denote the quotient class of a in $A/\sim_X$ by Xa.*

$A/\sim_X$ has the structure of boolean algebra by defining the summation and multiplication operations by $Xa * Xb = X(a*b)$ and $Xa + Xb = X(a+b)$, $X1$ the unitary element and $X0$ is the null element, and $\bar{Xa} = X\bar{a}$.

Definition 2.1.2. *A subset B of A, which have the structure of boolean algebra, under the same operations and trivial elements of A, is called a sub boolean algebra of A. Then B is asub boolean algebra of A i.e., $B \subseteq A$, $0 \in B$, $1 \in B$ and B is closed under addition, multiplication and negation.*

Example 2.1.3. *let A be a boolean algebra and $B \subseteq A$ be any subset of A, we can generate a sub boolean algebra of A by taking the extension B with all possible combinations of addition, multiplication and negations of elements of B together with 0 and 1.*

Definition 2.1.3. Homomorphism of Boolean Algebra
Let A and B be two boolean algebra. A boolean algebra homomorphism from A to B is a map $f : A \to B$ satisfies the following:
- $f(x+y) = f(x) + f(y)$,
- $f(x*y) = f(x) * f(y)$,
- $f(1) = 1$,
- $f(0) = 0$,

for all $x, y \in A$.

An Endomorphism of Boolean Algebra A is an homomorphism from A to A, and an isomorphism of boolean algebra is a bijective homomorphism.

Proposition 1. *Let A and B be two boolean algebras and f be a boolean algebra homomorphism from A to B, then*
$f(\bar{x}) = \bar{f(x)}$, for all $x \in A$, and
$Img(f)$ is a sub boolean algebra of B.

Proof. $f(x) + f(\bar{x}) = f(x + \bar{x}) = f(1) = 1$ and $f(x) * f(\bar{x}) = f(x * \bar{x}) = f(0) = 0$. □

2.2 The Lattice structure on boolean algebra and The Fixed Point Theorem

Definition 2.2.1. *Let A be a boolean algebra. Define the relation $\subseteq$ on A by, $x \subseteq y$ if and only if $x + y = y$. We can check that $\subseteq$ is an order relation.*

Proposition 2. *for any x, y in A.*

(i) *$x \subseteq 1$ and $0 \subseteq x$.*

(ii) *$x * y \subseteq x$ and $x * y \subseteq y$ and if there is z such that $z \subseteq x$ and $z \subseteq y$ then $z \subseteq x * y$.*

(iii) *$x \subseteq (x + y)$ and $y \subseteq (x + y)$ and if there is z such that $x \subseteq z$ and $y \subseteq z$ then $(x + y) \subseteq z$.*

This proposition shows that, every two elements x and y in A have a unique supremum or least upper bound which is $x + y$, and a unique infimum or greatest lower bound which is $x * y$. Moreover, for any subset B of A we have $\Sigma_{x \in B} x$ a least upper bound and $\Pi_{x \in B} x$ a greatest lower bound. Therefore, (S,$\subseteq$) forms a complete lattice.

Theorem 2.2.1. *Tarski's fixed point theorem*
Let $L = (A, \subseteq)$ be a complete lattice, and f be an increasing function from A to A (if $x \subseteq y$, then $f(x) \subseteq f(y)$), then
The set P of fixed points of f is non-empty and the system $(P, \subseteq)$ is a complete lattice.

For the proof and more details see [Tar55].

Theorem 2.2.2. *Let $L = (A, \subseteq)$ be a complete lattice, and f be an increasing function from A to A. if A is finite then the least fixed point is equal to $f^k(0)$ for some $k \in \mathbb{N}$; $f^k = f \circ ... \circ f$ k times.*

Proof. see [Lar07], Theorem 2. □

The fixed point theorem play a significant role in model checking, particularly in temporal logic. In fact, given a state transition diagram, if we identify formulas with set of states where it is true, then temporal operators could be defined as a fixed points of certain monotonic operators. Consider a state transition diagram $M = (L, R, Atm, V, I)$, let $\phi \in Atm$, we associate ϕ to $V(\phi)$ and every conjunction of atomic proposition to conjunction of sets, and negation to complement. In this way we associated every propositional formula to the sets where it is true. Logical formulas with temporal operators are also associated to states where they are true, by representing the temporal operators, defined in the previous chapter, as fixed point of monotonic operators on 2^L.
First for every subset $Y \in 2^L$ we define the operator $pre_E(Y) = \{s \in L$ such that $(s, s') \in R$ for some $s' \in Y\}$

• M ,s$\models$ $EX(\phi)$ if and only if $s \in pre_E(\phi)$.

• M ,s$\models$ $EG(\phi)$ if and only if s belong to the fixed point of $\tau : 2^L \to 2^L$, defined by $\tau(x) = \phi \cap x \cap pre_E(x)$ starting the iteration from L.

• M ,s$\models$ $\phi U \psi$ if and only if s belong to the fixed point of the operator, defined by $\tau(x) = x \cup (\phi \cap pre_E(x))$ starting iterating from ψ.

The fixed point theorem ensure the existence of the fixed points.

2.3 Minterms

Definition 2.3.1. minterms
Let A be a boolean algebra. $m \in A$ is a minterm iff for every y in A, if $y \neq 0$ and $y \subseteq m$ then $y = m$.

Example 2.3.1. *Let S be a set, in the boolean algebra 2^S all singletons are minterms.*

Proposition 3. *Let A be a boolean algebra*

(i) *If m_1 and m_2 are two different minterms then $m_1 * m_2 = 0$.*

(ii) *If A is finite, then any element of A can be uniquely written as a summation of minterms.*

Proof. (i) We have $m_1 * m_2 \subseteq m_1$ and $m_1 * m_2 \subseteq m_2$ which gives $m_1 = m_2 = m_1 * m_2$ or $m_1 * m_2 = 0$.

(ii) Let a be any element of A. If a is not a minterm, then there exist a non-zero element in $y \neq a$ such that $a = y + a = y + x\bar{y}$. Repeating the argument on y and $x\bar{y}$ and since A is finite we get a finite summation of minterms, and the uniqueness is a consequence of (i) .

□

Theorem 2.3.1. *Any finite boolean Algebra is of cardinality 2^k for some $k \in \mathbb{N}$.*

Proof. In fact if A is finite boolean algebra then we have finite number of minterms say $\{m_1, ..., m_k\}$, and we can make 2^k possible summations of these elements, each corresponds to one and only one element of A by previous proposition. □

By the same reasoning every finite boolean algebra of cardinality 2^k has k minterms.

Theorem 2.3.2. *Any two finite boolean algebras of the same cardinality are isomorphic.*

Proof. let A and B be two finite boolean both having cardinality 2^k and let $\{m_1, ..., m_k\}$ and $\{s1, ..., s_k\}$ be the set of the minterms of A and B respectively, we can construct an isomorphism of boolean algebra
$\psi : A \to B$, defined by $\psi(m_i) = s_i$ for all $i \in \{1, ..., k\}$.

□

Lemma 2.3.3. *If A is a finite boolean algebra and $m_1, ..., m_k$ are minterms in A, then*
$m_1 + ... + m_k = 1$ if and only if $m_1, ..., m_k$ are all the minterms of A.

Proof. Since A is finite then 1 could be written uniquely as finite number of minterms $1 = m_1 + ... + m_k$, now suppose that m is a minterm not in $\{m_1, ..., m_k\}$ then $m = m * 1 = m(m_1 + ... + m_k) = 0$ by proposition 3, which is a contradiction. □

Theorem 2.3.4. *Let A be a finite boolean algebra, and let $x = m_1 + ... + m_k$ be an element of A, written in the unique minterms expansion form, then in the boolean algebra $A/\sim_x$ the classes $xm_1, ..., xm_k$ are all the minterms.*

Proof. Let $xm_j \in \{xm_1, ..., xm_k\}$ lets prove that xm_j is a minterm in $A/\sim_x$, let $xy \in A/\sim_x$ such that $xy \subseteq xm_j$ then $xm_j = xy + xm_j$, so $x * m_j = x * (y + m_j)$ in A, but $x * m_j = m_j$ then $m_j = x * y + m_j$, since m_j is a minterm in A then $x * y = m_j$ implies $x * y = x * m_j$ and so $xy = xm_j$ or $x * y = 0$ then $xy = x0$. To prove that $xm_1, ..., xm_k$ are all the minterms, by previous lemma it is sufficient to show that $xm_1 + ... + xm_k = x1$ which is straight forward. □

2.4 The free boolean Algebra generated by finite set of variables

For implementation in c++, we are able to stimulate a free boolean algebra generated by a finite set $\{x_1, ..., x_n\}$ using the cudd library $[Som18]$, by constructing the ordered binary decision diagrams BDD of n variables. A BDD is a diagram that represent a boolean function, and a boolean function, of n variables, is nothing but one combination of n boolean variables.

Definition 2.4.1. *let $N = \{x_1, ..., x_n\}$ be a set of finite elements and $\bar{N} = \{\bar{x}_1, ..., \bar{x}_n\}$ be an associated set. The free boolean algebra AN generated by N is the free unitary commutative semiring generated by $N \cup \bar{N} \cup \{0, 1\}$ under the following restrictions:*
• *$x * x = x$ and $x + x = x$ for every x in $N \cup \bar{N} \cup \{0, 1\}$.*
• *$x_i + \bar{x}_i = 1$ and $x_i * \bar{x}_i = 0$ for every i in $\{1, ..., n\}$.*

Using the fact that $x_i \bar{+} x_j = \bar{x_i} * \bar{x_j}$, we can show that any element a of AN has a unique negation $\bar{a}$, and hence AN is a boolean algebra.

Proposition 4. *Let $N = \{x_1, ..., x_n\}$ be a set and AN be the free boolean algebra generated by N. set $M = \{\}$*

Definition 2.4.2. *Let A be a finite boolean algebra. A set $\{p_1, ..., p_k\}$ of elements of A is a partition of A, if and only if*
• *$p_i * p_j = 0$ for all $i \neq j$.*
• *$p_1 + ... + p_k = 1$.*

Proposition 5. *If $\{p_1, ..., p_k\}$ is a partition of a boolean algebra A, then for every $p_j \in \{p_1, ..., p_k\}$, we have $\bar{p_j} = p_1 + .. + p_{j-1} + p_{j+1} + ..p_k$.*

Proof. In fact it comes directly from the definition that $p_j * (p_1 + .. + p_{j-1} + p_{j+1} + ..p_k) = 0$ and $p_j + p_1 + .. + p_{j-1} + p_{j+1} + ..p_k = 1$. □

Proposition 6. *Let A be a boolean algebra and $P = \{p_1, ..., p_k\}$ be a partition of A. Consider $A(P) = \bigcup_{I \subseteq \{1,...,k\}} \{\Sigma_{i \in I} p_i;\} \cup \{0\}$ the set of all possible summations of $\{p_1, ..., p_k\}$ together with the zero element.*
$A(P)$ is a sub boolean algebra of A, and P is the set of all minterms of $A(P)$.

Proof. Under the same operations of A and by proposition 4, $A(P)$ form a boolean algebra. And since any element in AP is written as a summation of $\{p_1, ..., p_k\}$, it becomes easy to show that P form the set of all minterms of S.

□

Proposition 7. *AN is a free boolean algebra generated by the finite set of variables $N = \{x_1, ..., x_n\}$ let $M = \{x_{i1}, ..., x_{ik}\}$, $k \leq n$ be any subset of N. Call PM the set formed of products of variables in M, where each variable appear only once in it's normal or negation form. The set PM partition AN. Moreover, the set defined in the same way for N forms the set of all minterms of AN.*

Note that the cardinality of M is 2^k, and so by proposition 6, $A(PM)$ has 2^k minterms.

Example 2.4.1. *let $N = \{x_1, x_2, x_3\}$ and let AN be the free boolean algebra generated by N. Let $M = \{x_1, x_2\}$, then $PM = \{x_1x_2, \bar{x_1}x_2, x_1\bar{x_2}, \bar{x_1}\bar{x_2}\}$ is a partition of AN. And $A(PM) \sim AM$ is a sub boolean algebra of AN, with 4 minterms.*

Given AN a free boolean algebra generated by $N = \{x_1, ..., x_n\}$ we will give an algorithm that returns a vector of all the minterms of AN.

Algorithm 1 An algorithm that returns a vector of BDDs all the minterm of a given free boolean algebra

```
  procedure $Minterms$(BDD vector X)
Require: A BDD vector X that have each variables as BDD
Ensure: A BDD vector that have all the minterms the free boolean algebra generated
  by X        ▷ If the input vector is of size k then the output vector will be of size $2^k$
    BDD vector myvector = {1};     ▷ myvector initialy contain only the one BDD
    for BDD $x$ in X do
        for BDD m in myvector do
            myvector.pushBack($m * x$);
            myvector.pushBack($m * \bar{x}$);
        end for
    end for
    return myvector;
```

In the algorithm above, if we choose the input to be a vector that contains only some variables, but not all. Then, by the previous proposition we will get a partition of AN.

2.5 Boolean algebra on state transition diagram

Presenting relations by boolean formulas

Now, we see how can we represent a relation on state as an element of a boolean algebra.

Let $S = \{s_1, ..., s_m\}$ be a finite set of m elements, and let R be a relation defined on S. Let $N = \{x_1, ..., x_n\}$ be a set of variables, such that $n = \lceil \log_2(m) \rceil$. The free boolean algebra AN have $k = 2^n$ minterms. Let $L = \{r_1, ..., r_k\}$ be the set of minterms of AN.
Choose a sequence of m minterms of AN and associate each element of S to one minterm in the selected sequence. This define an injection from S to minterms AN $\gamma : S \to AN$. By proposition 6 AN is a sub boolean algebra of $A(N \cup N')$. So consider the mapping

$$\theta : S \xrightarrow{\gamma} AN \overset{i}{\hookrightarrow} A(N \cup N')$$

where i is the inclusion map.
To encode the relations, we need another set $N' = \{x'_1, ..., x'_n\}$, with same cardinality of N. Likewise N, we construct the surjection γ' associated with N'. And in the same way we define

$$\theta' : S \xrightarrow{\gamma} AN' \overset{i}{\hookrightarrow} A(N \cup N')$$

By proposition 7, the image of θ and θ', $\theta(S)$ and $\theta'(S)$ each define a partition of $A(N \cup N')$.
Now on $A(N \cup N')$ define the element $f_R = \Sigma\theta(s) * \theta(s')$ for all $(s, s') \in R$.

Let $s, s' \in S$. Since $\theta(S)$ and $\theta'(S)$ both partition $A(N \cup N')$, then

$$sRs' \text{ iff } \theta(s) * \theta(s') * f_R \neq 0$$

Before giving an example we need to define two important boolean algebra endomorphismes, that will be used much in model checking algorithms, the existential abstraction and the swap variables.

Cofactors and The Existential abstraction EAb()

Definition 2.5.1. *Let $N = \{x_1, ..., x_n\}$ be a set of variables, and AN be the free boolean algebra generated by N. Given any $x_i \in N$, we can define the boolean algebra endomorphism $f_{\bar{x_i}}$, by defining it on the variables as follows*

$f_{\bar{x_i}}(x_i) = 0$

and

$f_{\bar{x_i}}(x_j) = x_j$ *for all* $j \neq i$,

and the boolean algebra endomorphism f_{x_i} as,

$f_{x_i}(x_i) = 1$

and

$f_{x_i}(x_j) = x_j$ *for all* $j \neq i$.

Let a be in A; $f_{x_i}(a)$ and $f_{\bar{x_i}}(a)$ are called the cofactors of A with respect to x_i.

Definition 2.5.2. *Let AN as in the last definition, $x_i \in N$, and $\phi \in AN$. Consider the map $ExistAbstract_{x_i}(\phi) = f_{x_i}(\phi) + f_{\bar{x_i}}(\phi)$. This map is called the existential abstraction of x_i from ϕ.*

The map $Exist_v(\phi)$, where v is a vector of variables, apply $ExistAbstract_x(\phi)$, for every variable x in v.

This map will remove the variable x_i from ϕ, but it return 1 if $x_i \subseteq \phi$.

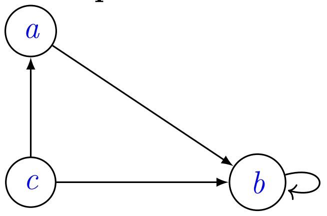
Definition 2.5.3. The SwapVariables() method

Let $N = \{x_1, ..., x_n\}$ be a set of variables and AN the free boolean algebra generated by N, $x_i, x_j \in N$. The endomorphism of boolean algebra on AN defined by it's application on the variables by

$$SV_{x_i,x_j}(x_i) = SV_{x_i,x_j}(x_j);$$
$$SV_{x_i,x_j}(x_j) = SV_{x_i,x_j}(x_i);$$
$$SV_{x_i,x_j}(x_k) = SV_{x_i,x_j}(x_k) \text{ if } k \neq i, j.$$

is called the swap variable method. If $v = (x_1, ..., x_k)$ and $v' = (x'_1, ..., x'_k)$, are two vectors of variables we get $SwapVariables_{v,v'}(\phi)$ by applying $SV_{xi,xi'}(\phi)$ for all $1 \leq i \leq k$

Example 2.5.1. *Consider the state transition diagram:*



We have three states, so to encode the set of states, we need 2 boolean variables $N = \{x_1, x_2\}$. In the free boolean algebra AN we have 4 minterms $\{x_1 * x_2\ ,\ \bar{x_1} * x_2\ ,\ x_1 * \bar{x_2}\ ,\ \bar{x_1} * \bar{x_2}\}$ this set defines L the set of all states. To encode the relation we also need $N' = \{x'_1, x'_2\}$, whose minterms will be identified to L. Then the relation R is an element of the boolean algebra $AN \otimes AN' = A(N \cup N')$.
Specifically, in our case, if we associate a to $x_1 * x_2$, b to $\bar{x_1} * x_2$, and c to $x_1 * \bar{x_2}$, and making a similar association for the states to the minterms of AN'. Then the relation R will be defined by $x_1 * \bar{x_2} * x'_1 * x'_2\ +\ x_1 * \bar{x_2} * \bar{x'_1} * x'_2\ +\ x_1 * x_2 * \bar{x'_1} * x'_2\ +\ \bar{x_1} * x_2 * \bar{x'_1} * x'_2$.
Taking $\{c\}$ to be the set of initial states, containing only the sate c, and encoded by $I = x_1 * \bar{x_2}$ we claim that the system (L, R, I) represent our digraph above, where the nodes of the diagram are the reachable states.
Reachable states in the last example is the BDD $x_1 * x_2\ +\ \bar{x_1} * x_2\ +\ x_1 * \bar{x_2}$.

The atomic propositions are letters associated to the nodes of the digraph by the evaluation map V. As mentioned before, in model checking all formulas, generated by propositional and temporal operations on atomic propositions, are represented by sets of the states, that appear in the diagram. In this example, all formulas are represented by elements of the boolean algebra $2^{\{a,b,c\}}$, the power set of $\{a, b, c\}$. By Theorem 1.3.4 $2^{\{a,b,c\}} \sim AN/_{reach}$, by associating each minterm in $2^{\{a,b,c\}}$ to one and only one minterm in $AN/_{reach}$.

Reachable States

Now we show how we get the reachable states as disjunction of minterms of AN. using arithmetic operations on $AN \otimes AN'$.

In the last example, BDDs have $\{x_1, x_2, x_1', x_2'\}$ as variables. When we use the data structure BDD, we are referring to an element in $AN \otimes AN' := A(N \cup N')$. The reachable states are the fixed point of the operator defined by $\tau(x) = x + next(x)$ by starting the iteration from I, where $next(x) = \{s \in L$, such that there exists $s' \in x$, where $(s', s) \in R\}$. Clearly τ is monotonic.

Algorithm 2 Given a state transition diagram, this algorithm returns the element reachable states as a BDD

```
  procedure reach(BDD $I$, BDD $f_R$, BDD vector $N$, BDD vector $N'$)
Require: The BDD $I$ and $f_R$ represents the initial states and the relation. The BDD
  vectors $N$ and $N'$ have the variables as BDDs.
Ensure: A BDD in $AN$ that represent the reachable states
    BDD $reach = zeroBDD$
    BDD $q_1 = I$
    BDD $tmp = zeroBDD$
    while $q_1$ not equal $reach$ do
        $reach = q_1$
        $tmp = q_1$
        $tmp = tmp * f_R$
        $tmp = Exist_N(tmp)$                                ▷ Remove the variables of $N$
        $tmp = SwapVariables_{N,N'}(tmp)$
        $q_1 = q_1 + tmp$
    end while
    return $reach$
```

2.6 Model checking multi-agent system

Consider a concurrent game structure $\{(L_i, Act_i, Pr_i, Ev_i, Atm, V, In)\}_{1 \leq i \leq n}$ of n agents. The encode, all possible local and global states, the actions, the protocol and the evolution functions, as BDDs.

Let n_i be the number of variables we need to encode the local state of agent i, then we need $n = \Sigma_i n_i$ to encode all global states. In multi-agent system we also need to encode actions. If m_i is the number of variables to encode the actions of agent i, then we need $m = \Sigma_i m_i$ variables to encode joint actions.

Using the Cudd library, we initialise a BDD manager, with $N = 2n + m$ number of variables. $2n$ for states and next states, and m for actions.

We encode the protocol and the evolution function as follows:

The Protocol formula Pr

If L_i is the set of extended local states and A_i is the set of actions of the agent i, then the protocol is defined by the map $P_i : L_i \to 2^{A_i}$. For instance the fact that the state s_i is mapped to the set $\{a_1, ..., a_k\}$ is encoded by $s_i * (a_1 \oplus, ... \oplus a_2)$, where $a_1 \oplus a_2 = \bar{a}_1 * a_2 + a_{1*\bar{a}_2}$, (the $\oplus$ here indicates that only one of the action could be applied at once). The disjunction of such elements over the set L_i, is the element that identify the protocol function of the agent Ag_i, we will denote it by $Pr_i = \Sigma_{s_i \in L_i}(s_i \bigoplus_{a \in P_i(s_i)} a)$. The global protocol formula Pr that represent the protocol map of each agent at once is the conjunction of the protocol formulas over all agents, $Pr = \Pi_{i=1}^{i=m} Pr_i$, m is the number of agents.

The Evolution formula Ev

Let A_i be the set of actions, and L_i be the set of extended local states of agent i. Let $B \subseteq (L_i \times A_i)$ be the subset of pairs of states actions, which are compatible with protocol of the agent. $EV_i(s,a) = s'$ is encoded by $s*a*s'$. If we take the disjunction over all states, we get the formula for $Ev_i : B \to L_i$. The global evolution formula E_v which encode the evolution of all agents is the conjunction of evolution formulas over the agents.

$\mathbf{SX}_{\mathrm{T}}$

In a state s, we say that, a group of agents T have a strategy to reach ϕ in the next state, if there exist a joint action, compatible with the protocol of agents in T in state s, which will guaranty that a state in ϕ will be reached in the next state, regardless of the action taken by agents outside the group.

Let's start with one simple example;

Example 2.6.1. *In this example we have 2 agents, agent A with actions $\{a, \bar{a}\}$, and agent B with actions $\{b, \bar{b}\}$. The goal is to find the $SX_{\mathrm{T}}(s_4)$.*

The idea is to get all the states with their compatible joint actions of A that could reach a sates other than s_4, and then remove them from the protocol formula and finally remove the actions and get the result.

The following figures shows the procedure,

$$\begin{tikzcd}
s_2 \arrow[r, "a*b"] & s_4 \arrow[dr, "a*b"] & \\
s_1 \arrow[u, "\bar{a}*b"] \arrow[r, "a*b"'] & s_3 \arrow[u, "a*\bar{b}"'] \arrow[r, "a*b"'] & s_5 \arrow[loop right, "a*b"]
\end{tikzcd}$$

$$\begin{array}{c}
Ev \\ = \\ s_1 * \bar{a} * b * s'_2 \\ + \\ s_1 * a * b * s'_3 \\ + \\ s_2 * a * b * s'_4 \\ + \\ s_3 * a * \bar{b} * s'_4 \\ + \\ s_3 * a * b * s'_5 \\ + \\ s_4 * a * b * s'_5 \\ + \\ s_5 * a * b * s'_5
\end{array}$$

$$\begin{array}{ccccc}
Pr & & & & \\
= & & & & \\
s_1 * \bar{a} \oplus a * b & & s_1 * \bar{a} \oplus a & & \\
+ & & + & & \\
s_2 * a * b & & s_3 * a & & \\
+ & - & + & = & s_2 * a * b \\
s_3 * a * \bar{b} \oplus b & & s_4 * a & & \\
+ & & + & & \\
s_4 * a * b & & s_5 * a & & \\
+ & & & & \\
s_5 * a * b & & & &
\end{array}$$

After we remove the actions from the result above, we get s_2 which is our final result. Notice that $s_3 * a$ linked to s_4, but it is also linked to s_5, that's why s_3 is not included in the result.

Algorithm 3 An algorithm that returns the states where there is a stratrgy for the group T to reach ϕ in the next state

```
procedure $SX_{\mathrm{T}}(\phi)$(BDD $\phi$)
    partial = $\bar{\phi} * reach$                        ▷ get the sets outside $\phi$
    partial = partial.SwapVariables(pv,v)          ▷ change states to next state.
    partial = partial * Ev          ▷ get sates and actions linked to states outside $\phi$
    partial = Exists(pv,partial)$*reach$           ▷ remove variables of next state.
    partial = partial * Pr
    partial = partial.ExistAbstract(actagntout) ▷ remove actions of agents outside T.
    partial = !partial$*reach$
    partial = Pr$*$ partial          ▷ subtract the result from the protocol formula.
    partial = partial.ExistAbstract(actagntout)   ▷ remove actions of agents out T
    return partial.ExistAbstract(actagntin)         ▷ remove actions of agents in T.
```

SF$_{\mathrm{T}}$

The goal is to fined the states where there exists a strategy for group T to reach ϕ in the future. In the last example, we can see that agent A has a strategy to reach s_2 in the next state from s_1. So starting from s_1, if agent A play $\bar{a}$ to reach s_2, and then play a from s_2 she will reach s_4. Then $\mathbf{SF}_{\mathrm{T}}(s_4) = s_1 + s_2$. $SF_{\mathrm{T}}(\phi)$ could be defined as, the fixed point of the operator $\tau(p) = SX_{\mathrm{T}}(\phi) + p + SX_{\mathrm{T}}(p)$

Algorithm 4 An algorithm that returns the states where there is a stratrgy for the group T to reach ϕ eventually in the future

```
procedure $SF_{\mathrm{T}}(\phi)$(BDD $\phi$)
    BDD $p = \phi$
    BDD $q =$ one BDD
    while p $\neq q$ do        $q = p$
     while p do = p + $SX_{\mathrm{T}}$(p)
     end while
     return p
```

Chapter 3

Complexity of natural strategies

3.1 Complexity of BDDs

Definition 3.1.1. *Let A be a boolean algebra, and let $P = (p_1, ..., p_k)$ be a vector of elements of A. If we apply the method $Minterms()$, represented in $Algorithm$ 1, we will get a partition of A. Let us call the elements of the partition, minimal cubes generated by P.*

Example 3.1.1. *Let A be a boolean algebra, and let $p_1, p_2 \in A$. The set $\{p_1p_2, \bar{p_1}p_2, p_1\bar{p_2}, \bar{p_1}\bar{p_2}\}$ is the set of minimal cubes generated by (p_1, p_2).*

Definition 3.1.2. *Let A be a boolean algebra. We will call, $P = \{p_1, ..., p_k\}$ a finite subset of A, a set of free variables of A, iff non of the minimal implicants generated by P vanishes. One can show that, this is equivalent to, no element in P could be written completely as combination of the others.*

Definition 3.1.3. *Let A be a boolean algebra, and $P = \{p_1, ..., p_k\}$ be a set of free variables of A. Take a conjunction of elements of P in their normal or negation form and denote it by c, we call it a cube, we say c is an implicant of a with respect to P, iff $c \subseteq a$ i.e., $a = c + x$ for some $x \in A$. c is called a prime implicant of a, if c is not $\subseteq$ of any other implicant of a. Moreover, we define the size of c to be the size of the product i.e., the number of variables that appear in the product.*

Algorithm 5 An algorithm that calculate the size of a cube

```
  procedure $CubeSize$(BDD cube, BDD vector $V$)
Require: A cube $c$ of variables in a vector $V$ and the vector $V$
Ensure: An integer that represent the size of the cube with respect to the variables
  in $V$
    int counter = 0
    for var in V do
        if $\bar{var} * cube == 0$ then
            counter++
        end if
    end for
    return counter
```

Definition 3.1.4. *Let A be a boolean algebra, P finite set of free variables, and $a \in A$. Suppose that a could be written in terms of elements of P. The sum of all prime implicants of a with respect to P, which is equal to A, is called the minimal covering sum of a with respect to P. If the sum of prime implicants does not equal a, then a could not be written as combination of elements of P.*

Definition 3.1.5. *First type complexity Let V be a vector of BDD, and ψ a BDD which is a combination of elements of V. We define the 1^{st} kind complexity of ψ with respect to V, denoted $compl_{\Sigma}^{V}(\psi)$ to be the total size of the minimal covering sum of ψ with respect to V i.e., the total number of variables + number of operations used including the negation.*

Example 3.1.2. *Let $P = \{p_1, p_2, p_3, p_4\}$ and $\psi = p_1 * p_2 + \bar{p_3}$, then $compl_{\Sigma}^{P}(\psi) = 6$.*

We implemented in $c++$ programming an algorithm that count complexity of a BDD, the algorithm was verified by many examples. The algorithm take a BDD and calculate the complexity by iterating through the prime implicants, so the main task was to get the prime implicants of the BDD as a BDD vector, after that the remaining becomes simple.

Now we introduce an algorithm of a method, that take a vector of free variables P and a BDD ψ, and returns the vector of prime implicants of ψ with respect to P. The idea of the algorithm is to generate cubes starting by cubes with only one variables, and increase the size of the cube (the size of the product) by iterations, while checking if the cube is an implicant of ψ, until the summation of the implicants selected is equal to to ψ. By starting with cubes of small size, and checking if any new implicant of greater size is $\subseteq$ of an already selected one, we guarantee that all selected implicants are prime implicants.

Algorithm 6 An algorithm that returns the prime implicants of a formula, as a vector of BDDs

```
procedure PrimeImplicants(BDD $\psi$,BDD vector $P$)
    BDD vector $myvector$ ←{}
    BDD vector $Bvector$ ←{one BDD}
    BDD vector $Cvector$
    BDD $sum$ ← zero BDD
    int e
    if $\psi == 0$ then return {zero BDD}
    end if    ▷ In the trivial case, it return the vector that contains the zero BDD
    for i=1, to $A.size()$ do
        for cube in $Bvector$ do
            for $\phi$ in $A$ do
                if $cube * \phi \neq 0$ then ▷ We only generate cubes that intersect with $\psi$
                    $Cvector.pushBack(cube * \phi)$
                end if
                if $cube * \phi \neq 0$ then
                    $Cvector.pushBack(cube * \bar{\phi})$
                end if
            end for
        end for
        clear Bvector
        for $cube$ in $Cvector$ do
            if $cube * \bar{\psi} == 0$ then                    ▷ check if cube ⊆ $\psi$
                $e \leftarrow 0$
                for $cube2$ in $myvector$ do
                    if $cube * \bar{cube}2 == 0$ then
                        $e \leftarrow 1$
                        break
                    end if
                end for
                if e == 0 then
                    $myvector.pushBack(cube)$
                    $sum = sum + cube$ ▷ sum contains the sum of cubes in myvector
                    if $sum == \psi$ then return myvector
                    end if                  ▷ if sum = $\psi$ we return myvector
                end if
            else
                if $cube * \psi \neq 0$ then
                    $Bvector.pushBack(cube)$
                end if
            end if
        end for
        clear $Cvector$
    end for
    return empty vector
```

This method will be used to calculate the complexity of a formula.

3.2 Complexity using existential abstraction

Let A be a free boolean algebra generated by $N = \{x_1, ..., x_n\}$

A BDD with variables $N = \{x_1, ..., x_n\}$ could be seen as an element of the free boolean algebra AN generated by N. In the cudd library there is an inbuilt method $ExistAbstract(BDD\phi, BDDx_i)$ that apply existential abstraction of x_i from ϕ, as explained in chapter 2.

For example if $\phi = x_1 * x_2 + x_3$, then $ExistAbstract(\phi, x_1) = x_2 + x_3$; if $\phi = x_1 + x_3$, then $ExistAbstract(\phi, x_1) = 1$, intuitively, we can imagine it as dividing x_1 from ϕ.

One type of complexity for propositional formulas is the number of variables appear in the reduced form of the formula. We can use the $ExistAbstract()$ method to check for complexity with respect to the variables in $N = \{x_1, ..., x_n\}$. Since a variable x_i appear in the reduced form of a formula ϕ, iff $ExistAbstract(\phi, x_i) \neq \phi$. But if we wish to find the complexity of a BDD in AN in terms of any free set of variables $P = \{p_1, ..., p_k\}$ this is not possible using the inbuilt $ExistAbstract()$ method.

Here we propose an algorithm $ExistAbsProp(BDD\phi, p_i, BDDvectorP)$ that apply extensional abstraction of p_i from ϕ assuming that ϕ could be written as a combination of elements of P.

Algorithm 7 An algorithm that apply extensional abstraction of p_i from ϕ

```
procedure ExistAbsProp(BDD $\phi$, $p_i$, BDD vector $P$)
    BDD result = zero BDD
    BDD vector $V$ = $P.remove(p_i)$                ▷ remove $p_i$ from $P$
    BDD vector MV = $Minterms(V)$      ▷ Generate the partition formed by $V$
    for cube in MV do
        if $\phi * cube \neq 0$ then
            result += cube
        end if
    end for
    return result
```

Now using this algorithm we can find to find the second type complexity with respect to P.

Moreover, this algorithm could also be used to get the prime implicants of a formula in a way different than the way represented in last section, certainly by forming the minterms of P "or partition of P" that intersect with ϕ, then in an inverse way to the algorithm represented before, we apply exist abstract to each cube through iterations till we reach prime implicants.

3.3 Model checking Natural strategy

Here we try for an algorithm to check for natural strategies with certain complexity. In the introduction we defined natural strategies as set of actions each guarded by a propositional formula.

In a concurrent game structure $\{(L_i, Act_i, Pr_i, Ev_i, Atm, V, In)\}_{1\leq i\leq n}$, let $N = \{x_1, ..., x_l\}$ be the set of variables to encode global states, $N' = \{x'_1, ..., x'_l\}$ for next states and $A = \{a_1, ..., a_r\}$ for joint actions, and the BDD *reach* encode the set of reachable states.

To check if a group of agents T have a natural strategy to meet ϕ in the next state. First we need to get states where there is a strategy to reach ϕ in the next state together with the joint actions of T, that guarantees the condition, then separate the states linked to the same actions from each other in a vector. Now we have a vector contains a subsets of the reachable states, we claim that, if each of the subsets represented as an element in $AN/_{reach}$ could be written as a combination of the free set $P = \{V(\phi_1) * reach, ..., V(\phi_n) * reach\} \subseteq AN/_{reach}$, then there exist a natural strategy of complexity equals to the sum of the complexities of each subset with respect to the free set.

$SXNat_{\mathrm{T}}$ is exactly the same method SX_{T} represented in chapter 1, except that we do not remove the actions from the formula. The next algorithm apply $SXNat_{\mathrm{T}}(\phi)$ on a BDD ϕ. And after generating all the joint actions in a vector, we multiply each joint action with $SXNat_{\mathrm{T}}(\phi)$, if the multiplication is not zero, we remove the variables of the actions from the result and add it to a vector, which will be the final result after iterating through all joint actions.

$MintermsBDDvectorX$

Algorithm 8 An algorithm that returns a vector of BDDs each represent the states where same action should be applied, in order for the group T to reach ϕ in the next state

```
procedure $NatStratX_{\mathrm{T}}$(BDD $\phi$)
    BDD vector JntAct = $Minterms(A)$    ▷ $A$ is a BDD vector that contains all the variables of the actions of agents inside T
    for act in JntAct do
        if $SXNat_{\mathrm{T}}(\phi) *$ act $\neq 0$ then
            partial = Exist$_A$(SXNat$_{\mathrm{T}}(\phi) *$ act) $* reach$    ▷ remove the actions and make sure we are in the reachable states
            myvector.push-back(partial)
        end if
    end for
    return myvector
```

If each BDD in $NatStratX_{\mathrm{T}}(\phi)$ can be written in terms of elements from $P = \{V(\phi_1) * reach, ..., V(\phi_n) * reach\}$ we say there is a natural strategy.

Algorithm 9 An algorithm that check for natural strategies for T to reach ϕ in the next state. It returns the complexity of the natural strategy, with respect to set of free variables P, if it exists, and -1 if no natural strategy exists.

```
procedure INT $CmplxNatStratX_{\mathrm{T}}$(BDD $\phi$,BDD vector $P$)
    for BDD form in $NatStratX_{\mathrm{T}}(\phi)$ do
        BDD $tmp = PrimeImplicants(form, P)$
        if $tmp$ is empty then
            return $-1$
        end if
        for BDD cube in $tmp$ do
            sum $+=$ CubeSize(cube,$P$)
        end for
    end for
    return sum
```

Conclusion

Symbolic model checking highlights the importance of representing data structure to overcome obstacles and provide a good framework for elegant algorithms. Implicit representations only cover the necessary information, yet the necessity of the information depends on the application. For instance, the BDD representation of logical statements limitate our definition of the complexity of logical formulas to classes of the same truth values, as a result of which, the definition of natural strategies and their complexity was also limited. Further work could be done to check for natural strategies considering complexities defined in the syntax of a statement. In this work, we set up the road map for symbolic model checking of natural strategies, by introducing algorithms that calculate the complexity of a logical formula, as a BDD, and by outlining an algorithm to check for natural strategy according to our definition. The next step is to introduce an algorithm to model check formulas using a strategic operator with bounded complexity.

Bibliography

[Tar55] A. Tarski. *A lattice-theoretical fixpoint theorem and its application.* Pacific Journal of Mathematics, 5:285–309, 1955.

[LMc93] Kenneth L.McMillan. *Symbolic Model Checking.* Carnegie Mellon University, 1993.

[Min96] Shin-ichi Minato. *Binary Decision Diagrams and Applications for Vlsi Cad.* Kluwer Academic Publishers, 1996.

[Als+00] T. Alsinet et al. *A Multi-agent system architecture for monitoring medical protocols.* Department of Computer Science Universitat de Lleida, 2000.

[Rai06] Franco Raimondi. *Model Checking Multi-Agent Systems.* Degree of Doctor of Philosophy at the University of London, 2006.

[Lar07] Kim S. Larsen. *A Note on Lattices and Fixed Points.* Department of Mathematics and Computer Science University of Southern Denmark, 2007.

[SR15] Elhadi Shakshukia and Malcolm Reidb. *Multi-Agent System Applications in Healthcare: Current Technology and Future Roadmap.* The 6th International Conference on Ambient Systems, Networks and Technologie, Elsevier, 2015.

[Som18] Fabio Somenzi. *CUDD: CU Decision Diagram Package Release 2.7.0.* Department of Electrical, Computer, and Energy Engineering University of Colorado at Boulder, 2018.

[JMM19] Wojciech Jamroga, Vadim Malvone, and Aniello Murano. *Natural Strategic Ability.* Elsevier, 2019.