A General Multi-Agent Modal Logic K Framework for Game Tree Search

Abdallah Saffidine* and Tristan Cazenave**

LAMSADE, Université Paris-Dauphine, 75775 Paris Cedex 16, France

Abstract. We present an application of Multi-Agent Modal Logic K (MMLK) to model dynamic strategy game properties. We also provide several search algorithms to decide the model checking problem in MMLK. In this framework, we distinguish between the solution concept of interest which is represented by a class of formulas in MMLK and the search algorithm proper. The solution concept defines the shape of the game tree to be explored and the search algorithm determines how the game tree is explored. As a result, several formulas class and several of search algorithms can represent more than a dozen classical game tree search algorithms for single agent search, two-player games, and multi-player games. Among others, we can express the following algorithms in this work: depth-first search, Minimax, Monte Carlo Tree Search, Proof Number Search, Lambda Search, Paranoid Search, Best Reply Search.

1 Introduction

1.1 Motivation

Deterministic perfect information strategy games constitute a broad class of games ranging from western classic CHESS and eastern GO to modern abstract games such as HEX or multiplayer CHINESE CHECKERS [22]. Single-agent search problems and perfect information planning problems can also naturally be seen as one-player strategy games. A question in this setting is whether some agent, can achieve a specified goal from a given position. The other agents can either be assumed to be cooperative, or adversarial.

For example, an instance of such a question in CHESS is: "Can *White* force a capture of the Black Queen in exactly 5 moves?" In CHINESE CHECKERS, we could ask whether one player can force a win within ten moves. Ladder detection in GO and *helpmate* solving in CHESS also belong to this framework. The latter is an example of a cooperative two player game.

1.2 Intuition

The main idea of this article is that we should see the structure of a game and the behaviour of the players as two distinct parts of a game problem.

Thus, a game problem can be seen as the combination of a Game Automaton (the structure of the game) and a solution concept represented by a modal logic formula (the behaviour of the players).

* abdallah.saffidine@dauphine.fr
** cazenave@lamsade.dauphine.fr

1.3 Contributions and Outline

Our contributions in this work are:

- We establish a relation between strategy games and the Multi-Agent Modal Logic K (MMLK). Then, we show that many abstract properties of games such as those mentioned in the introduction can be formally expressed as model checking problems in MMLK with an appropriate formula (Section 2).
- We describe three possible algorithms to solve the model checking problem in MMLK. These algorithms are inspired by depth-first search, effort numbers, and Monte Carlo playouts (Section 3).
- We show that numerous previous game tree search algorithms can be directly expressed as combinations of model checking problems and model checking algorithms (Section 4).
- We demonstrate that the MMLK allows new solution concepts to be rigorously defined and conveniently expressed. Moreover, many new algorithms can be derived through new combinations of the proposed search algorithms and existing or new solution concepts (formulas). Finally, it is a convenient formal model to prove some kind of properties about game algorithms (Section 5).

We believe that these contributions can be of interest to a broad class of researchers. Indeed, the games that fall under our formalism constitute a significant fragment of the games encountered in General Game Playing [11]. We also express a generalization of the Monte Carlo Tree Search algorithm [10] that can be used even when not looking for a winning strategy. Finally, the unifying framework we provide makes understanding a wide class of game tree search algorithms relatively easy, and the implementation is straightforward.

2 Strategy Games and Modal Logic K

2.1 Game model

We now define the model we use to represent games, namely the Game Automaton (GA). We focus on a subset of the strategy games that are studied in Game Theory. The games we are interested in are turn-based games with perfect and complete information. Despite these restrictions, the class of games considered is quite large, including classics such as CHESS and GO, but also multiplayer games such as CHINESE CHECKERS, or single player games such as SOKOBAN.

Informally, the states of the game automaton correspond to possible positions over the board, and a transition from a state to another state naturally refers to a move from a position to the next.

Although the game is turn-based, we do not assume that positions are tied to a player on turn. This is natural for some games such as GO or HEX. If the turn player is tightly linked to the position, we can simply consider that the other players have no legal moves, or we can add a *pass* move for the other players that will not change the position.

We do not mark final states explicitly, neither do we embed the concept of game outcome and reward explicitly in the following definition. We rather rely on a labelling of the states through atomic propositions. It is then possible to generate an atomic proposition for each possible game outcome and label each final state with exactly one such proposition.

Definition 1. *A* Game Automaton *is a 5-tuple* $G = (\Pi, \Sigma, Q, \pi, \delta)$ *with the following components :*

- Σ *is a non-empty finite set of agents (or* players*)*
- Π *is a non-empty set of* atomic propositions
- Q *is a set of* game states
- $\pi : Q \to 2^{\Pi}$ *maps each state* q *to its labels, the set of atomic propositions that are true in* q
- $\delta : Q \times \Sigma \to 2^{Q}$ *is a transition function that maps a state and an agent to a set of next states.*

We write $q \xrightarrow{a} q'$ when $q' \in \delta(q, a)$. We understand δ as: in a state q, agent a is free to choose as the next state any q' such that $q \xrightarrow{a} q'$.

Note that we lift the restriction that the turn order is fixed and that in a given position, only one player can move. That is, we assume that any player can move from a given position if asked to. This generalisation is straightforward for many games. For the other games where moves for non-turn players cannot be conceived easily, we either add a single `pass` move or simply accept that there are no legal moves for non-turn players.

We will assume for the remainder of the paper that one distinguished player is denoted by A and the other players (if any) are denoted by B (or B^1, …, B^k). Assume two distinct atomic propositions w and l, such that w is understood as a label of terminal positions won by A, while l is understood as a label of terminal positions not won by A.[1]

2.2 Multi-Agent Modal Logic K

Modal logic [5] is often used to reason about the knowledge of agents in a multi-agent environment. In such environments, the states in the GA are interpreted as possible worlds and additional constraints are put on the transition relation which is interpreted through the concepts of knowledge or belief. In this work, though, the transition relation is interpreted as a *legal move* function, and we do not need to put additional constraints on it. Since we do not want to reason about the epistemic capacities of our players, we use the simplest fragment of Multi-Agent Modal Logic K (MMLK) [5].

Syntax Let Π be a finite set of state labels and Σ be finite set of agents. We define the *Multi-Agent Modal Logic K (MMLK)* over Π and Σ, noted T, as follows:

Definition 2. *The* MMLK T *is defined inductively.*

$$\forall p \in \Pi, p \in T$$
$$\forall \phi_1, \phi_2 \in T, \neg\phi_1 \in T, (\phi_1 \wedge \phi_2) \in T$$
$$\forall a \in \Sigma, \forall \phi \in T, \Box_a \phi \in T$$

[1] Note that the atom l is not formally needed, as it can be defined using w and δ.

That is a formula (or *threat*) is either an atomic proposition, the negation of a formula, the conjunction of two formulas, or the modal operator $\Box_a$ for a player a applied to a formula. We read $\Box_a \phi$ as *all moves for agent a lead to states where ϕ holds*.

We define the following syntactic shortcuts.

- $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$
- $\Diamond_a \phi \equiv \neg \Box_a \neg\phi$

We read $\Diamond_a \phi$ as *there exists a move for agent a leading to a state where ϕ holds*. The precedence of $\Diamond_a$ and $\Box_a$, for any agent a, is higher than $\vee$ and $\wedge$, that is, $\Diamond_a \phi_1 \vee \phi_2 = (\Diamond_a \phi_1) \vee \phi_2$.

Semantics For a GA $G = (\Pi, \Sigma, Q, \pi, \delta)$, a state q in Q, and a formula ϕ, we write $G, q \models \phi$ when state q *satisfies* ϕ in game G. We omit the game G when obvious from context. The formal definition of satisfaction is as follows.

- $q \models p$ with $p \in \Pi$ if p is a label of q: $p \in \pi(q)$
- $q \models \neg\phi$ if $q \not\models \phi$
- $q \models \phi_1 \wedge \phi_2$ if $q \models \phi_1$ and $q \models \phi_2$
- $q \models \Box_a \phi$ if for all q' such that $q \xrightarrow{a} q'$, we have $q' \models \phi$.

It can be shown that the semantics for the syntactic shortcuts defined previously behave as expected.

Proposition 1.

- *$q \models \phi_1 \vee \phi_2$ if and only if $q \models \phi_1$ or $q \models \phi_2$*
- *$q \models \Diamond_a \phi$ if there exists an actions of agent a in q, such that the next state satisfies ϕ: $\exists q \xrightarrow{a} q'$, $q' \models \phi$.*

2.3 Formalization of some game concepts

We now proceed to define several classes of formulas to express interesting properties about games.

Reachability A natural question that arises in one-player games is *reachability*. In this setting, we are not interested in reaching a specific state, but rather in reaching any state satisfying a given property.

Definition 3. *We say that a player A can* reach *a state satisfying ϕ from a state q in exactly n steps if $q \models \underbrace{\Diamond_A \ldots \Diamond_A}_{n \text{ times}} \phi$.*

Winning strategy We now proceed to express the concept of having a winning strategy in a finite number of moves in an alternating two-player game.

Definition 4. *Player A has a* winning strategy *of depth less or equal to n in state q if $q \models \mathrm{WS}_{\alpha_n}$, where WS_{α_n} is defined as*

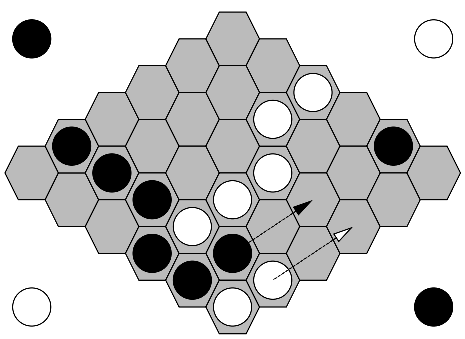
- $\mathrm{WS}_{\alpha_0} = \mathrm{WS}_{\beta_0} = w$
- $\mathrm{WS}_{\alpha_n} = w \vee (\neg l \wedge \Diamond_A \mathrm{WS}_{\beta_{n-1}})$
- $\mathrm{WS}_{\beta_n} = w \vee (\neg l \wedge \Box_B \mathrm{WS}_{\alpha_{n-1}})$

Ladders The concept of *ladder* occurs in several games, particularly GO [16] and HEX. A threatening move for player A is a move such that, if it was possible for A to play a second time in a row, then A could win. A ladder is a sequence of threatening moves by A followed by defending moves by B, ending with A fulfilling their objective.

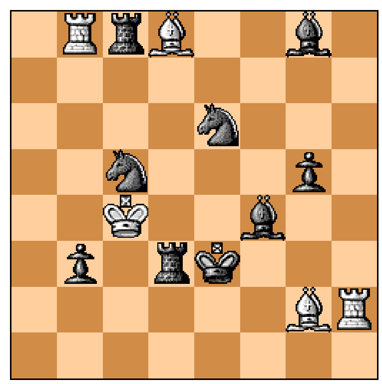
Definition 5. *Player A has a* ladder *of depth less or equal to n in state s if $q \models \mathrm{L}_{\alpha_n}$, where L_{α_n} is defined as*

- $\mathrm{L}_{\alpha_0} = \mathrm{L}_{\beta_0} = w$
- $\mathrm{L}_{\alpha_n} = w \vee (\neg l \wedge \Diamond_A (w \vee (\Diamond_A\, w \wedge \mathrm{L}_{\beta_{n-1}})))$
- $\mathrm{L}_{\beta_n} = w \vee (\neg l \wedge \Box_B\, \mathrm{L}_{\alpha_{n-1}})$

For instance, Figure 1a presents a position of the game HEX where the goal for each player is to connect their border by putting stones of their color. In this position, *Black* can play a successful ladder thereby connecting the left group to the bottom right border.



(a) HEX position featuring a *ladder* for *Black*.



(b) CHESS position featuring a *helpmate* for *Black* in four moves.

Fig. 1: Game positions illustrating the concepts of *ladder* and *helpmate*.

Helpmates In a CHESS *helpmate*, the situation seems vastly favourable to player *Black*, but the problemist must find a way to have the Black king checkmated. Both players move towards this end, so it can be seen as a cooperative game. *Black* usually starts in helpmate studies. See Figure 1b for an example. A helpmate in at most $2n$ plies can be represented through the formula H_n where $\mathrm{H}_0 = w$ and $\mathrm{H}_n = w \vee \Diamond_B \Diamond_A \mathrm{H}_{n-1}$.

Selfmates A *selfmate*, on the other hand, is a situation where *Black* forces *White* to checkmate the Black King, while *White* must do their best to avoid this. *Black* starts moving in a selfmate and a position with a selfmate satifies S_n for some n, where $\mathrm{S}_0 = w$ and $\mathrm{S}_n = w \vee \Diamond_B \Box_A \mathrm{S}_{n-1}$.

3 Search paradigms

We now define several model checking algorithms. That is, we present algorithms that allow to decide whether a state q satisfies a formula ϕ ($q \models \phi$).

3.1 Depth First Threat Search

Checking whether a formula is satisfied on a state can be decided by a depth-first search on the game tree as dictated by the semantics given in Section 2.2. Pseudo-code for the resulting algorithm, called Depth First Threat Search (DFTS) is presented in Algorithm 1.

```
dfts(state q, formula φ)
switch on the shape of φ do
    case p ∈ Π
        return p ∈ π(q)
    case φ1 ∧ φ2
        return dfts(q, φ1) ∧ dfts(q, φ2)
    case ¬φ1
        return ¬ dfts(q, φ1)
    case □a φ1
        let l = {q', q -a-> q'};
        foreach q' in l do
            if not dfts(q', φ1) then
                return false
        return true
```

Algorithm 1: Pseudo-code for the DFTS algorithm.

3.2 Best-first Search Algorithms

We can propose several alternatives to the DFTS algorithm to check a given formula in a given state. We present a generic framework to express best first search model checking algorithms. Best-first search algorithms must maintain a partial tree in memory, the shape of which is determined by the formula to be checked.

Nodes are mapped to a (state q, formula ϕ) label. A leaf is terminal if its label is an atomic proposition $p \in \Pi$ otherwise it is non-terminal. Each node is associated to a unique position, but a position may be associated to multiple nodes. [2]

The following static observations can be made about partial trees:

- an internal node labelled $(q, \neg\phi)$ has exactly one child and it is labelled (q, ϕ);
- an internal node labelled $(q, \phi_1 \wedge \phi_2)$ has exactly two children which are labelled (q, ϕ_1) and (q, ϕ_2);
- an internal node labelled $(q, \Box_a \phi)$ has as many children as there are legal transition for a in q. Each child is labelled (q', ϕ) where q' is the corresponding state.

```
bfs(state q, formula ϕ)
let r = new node with label (q, ϕ);
r.info ← init-leaf(r);
let n = r;
while r is not solved do
    while n is not a leaf do
        n ← select-child(n);
    extend(n);
    n ← backpropagate(n);
return r

extend(node n)
switch on the label of n do
    case (q, p)
        n.info ← info-term(n);
    case (q, ϕ1 ∧ ϕ2)
        let n1 = new node with label (q, ϕ1);
        let n2 = new node with label (q, ϕ2);
        n1.info ← init-leaf(n1);
        n2.info ← init-leaf(n2);
        Add n1 and n2 as children of n;
    case (q, ¬ϕ1)
        let n′ = new node with label (q, ϕ1);
        n′.info ← init-leaf(n′);
        Add n′ as a child of n;
    case (q, □a ϕ1)
        let l = {q′, q -a-> q′};
        foreach q′ in l do
            let n′ = new node with label (q′, ϕ1);
            n′.info ← init-leaf(n′);
            Add n′ as child of n;

backpropagate(node n)
let new_info = update(n);
if new_info = n.info ∨ n = r then
    return n
else
    n.info ← new_info;
    return backpropagate(n.parent)
```

Algorithm 2: Pseudo-code for a best-first search algorithm.

The generic framework is described in Algorithm 2. An instance must provide a data type for node specific information which we call *node value* and the following procedures. The `info-term` defines the value of terminal leaves. The `init-leaf` procedure is called when initialising a new leaf. The `update` procedure determines how the value of an internal node evolves as a function of its label and the value of the children. The `select` procedure decides which child is best to be explored next depending on the node's value and label and the value of each child. We present possible instances in Sections 3.3 and 3.4.

3.3 Proof Number Threat Search (PNTS)

We present a first instance of the generic best-first search algorithm described in Section 3.2 under the name PNTS. This algorithm uses the concept of *effort numbers* and is inspired from Proof Number Search (PNS) [2, 28].

The node specific information needed for PNTS is a pair of numbers which can be positive, equal to zero, or infinite. We call them *proof number* (PN) and *disproof number* (DN). Basically, if a subformula ϕ is to be proved in a state s and n is the corresponding node in the constructed partial tree, then the PN (resp. DN) in a node n is a lower bound on the number of nodes to be added to the tree to be able to exhibit a proof that $s \models \phi$ (resp. $s \not\models \phi$). When the PN reached 0 (and the DN reaches ∞), the fact has been proved and when the PN reached ∞ (and the DN reaches 0) the fact has been disproved.

The `info-term` and `init-leaf` procedures are described in Table 1, while Table 2 and 3 describe the `update` and `select-child` procedures, respectively.

Table 1: Initial values for leaf nodes in PNTS.

	Node label	PN	DN
`info-term`	(q, p) when $p \in \pi(q)$	0	∞
	(q, p) when $p \notin \pi(q)$	∞	0
`init-leaf`	(q, ϕ)	1	1

Table 2: Determination of values for internal nodes in PNTS.

Node label	Children	PN	DN
$(q, \neg\phi)$	$\{c\}$	$\mathrm{DN}(c)$	$\mathrm{PN}(c)$
$(q, \phi_1 \wedge \phi_2)$	C	$\sum_C \mathrm{PN}$	$\min_C \mathrm{DN}$
$(q, \Box_a \phi)$	C	$\sum_C \mathrm{PN}$	$\min_C \mathrm{DN}$

[2] While it is possible to store the state q associated to a node n in memory, it usually is more efficient to store move information on edges and reconstruct q from the root position and the path to n.

Table 3: Selection policy for PNTS.

Node label	Children	Chosen child
$(q, \neg\phi)$	$\{c\}$	c
$(q, \phi_1 \wedge \phi_2)$	C	$\arg\min_C \mathrm{DN}$
$(q, \Box_a \phi)$	C	$\arg\min_C \mathrm{DN}$

3.4 Monte Carlo Proof Search (MCPS)

Monte Carlo Tree Search (MCTS) [9, 8] is a recent game tree search technique based on multi-armed bandit problems [4]. MCTS has enabled a huge leap forward in the playing level of artificial GO players. MCTS has been extended to prove wins and losses under the name MCTS Solver [31] and it can be seen as the origin of the algorithm presented in this section which we call MCPS.

The basic idea in MCPS is to evaluate whether a state s satisfies a formula via probes in the tree below s. A probe, or Monte Carlo playout, is a random subtree of the tree below s whose structure is given by the formula to be checked in s. In the original MCTS algorithm, the structure of playouts is always a path. We lift this constraint here as we want to model check elaborate formulas about states. A probe is said to be *successful* if the formulas at the leaves are satisfied in the corresponding states. Determining whether a new probe generated on the fly is successful can be done as demonstrated in Algorithm 3.

```
probe(state q, formula φ)
switch on the shape of φ do
    case p ∈ Π
        return p ∈ π(q)
    case φ1 ∧ φ2
        return probe(q, φ1) ∧ probe(q, φ2)
    case ¬φ1
        return ¬ probe(q, φ1)
    case □a φ1
        let q′ be a random state such that q –a→ q′;
        return probe(q′, φ1)
```

Algorithm 3: Pseudo-code for a Monte-Carlo Probe.

Like MCTS, MCPS explores the GA in a best first way by using aggregates of information given by the playouts. For each node n, we need to know the total number of probes rooted below n (denoted by N) and the number of successful probes among them (denoted by R). We are then faced with an exploration-exploitation dilemma between running probes in nodes which have not been explored much (N is small) and running probes in nodes which seem successful (high $\frac{\mathrm{R}}{\mathrm{N}}$ ratio). This concern is addressed using the UCB formula [4].

Similarly to MCTS Solver, we will add another label to the value of nodes called P. P represents the proof status and allows to avoid solved subtrees. P can take three values: $\top$, $\bot$, or $?$. These values respectively mean that the corresponding subformula was proved, disproved, or neither proved nor disproved for this node.

We describe the `info-term`, `init-leaf`, `update`, and `select-child` procedures in Table 4, Table 5, and Table 6.

Table 4: Initialisation for leaf values in MCPS for a node n.

	Node label	P	R	N
`info-term`	(q, p) where $p \in \pi(q)$	$\top$	1	1
	(q, p) where $p \notin \pi(n)$	$\bot$	0	1
`init-leaf`	(q, ϕ)	$?$	`probe`(q, ϕ)	1

Table 5: Determination of values for internal nodes in MCPS.

Node label	Children	P	R	N
$(q, \neg\phi)$	$\{c\}$	$\neg\,\mathrm{P}(c)$	$\mathrm{N}(c) - \mathrm{R}(c)$	$\mathrm{N}(c)$
$(q, \phi_1 \wedge \phi_2)$	C	$\bigwedge_C \mathrm{P}$	$\sum_C \mathrm{R}$	$\sum_C \mathrm{N}$
$(q, \Box_a\, \phi)$	C	$\bigwedge_C \mathrm{P}$	$\sum_C \mathrm{R}$	$\sum_C \mathrm{N}$

Table 6: Selection policy for MCPS in a node n.

Node label	Children	Chosen child
$(q, \neg\phi)$	$\{c\}$	c
$(q, \phi_1 \wedge \phi_2)$	C	$\arg\max_{C, \mathrm{P}(c)=?} \frac{\mathrm{N}-\mathrm{R}}{\mathrm{N}} + \sqrt{\frac{2\log \mathrm{N}(n)}{\mathrm{N}}}$
$(q, \Box_a\, \phi)$	C	$\arg\max_{C, \mathrm{P}(c)=?} \frac{\mathrm{N}-\mathrm{R}}{\mathrm{N}} + \sqrt{\frac{2\log \mathrm{N}(n)}{\mathrm{N}}}$

4 Simulation of existing game tree algorithms

By defining appropriate formulas classes, we can simulate many existing algorithms by solving model checking problems in MMLK with specific search algorithms.

Definition 6. *Let ϕ be a formula, S be a model checking algorithm and A be a specific game algorithm. We say that (ϕ, S)* simulates *A if for every game, for every state q where*

A can be applied, we have the following: solving $q \models \phi$ with S will explore exactly the same states in the same order and return the same result as algorithm A applied to initial state q.

Table 7 presents how combining the formulas defined later in this section with the model checking algorithms defined in Section 3 allows to simulate many important algorithms. For instance, using the DFTS algorithm to model-check an APS_n formula on a HEX position represented as a state of a GA is exactly the same as running the Abstract Proof Search algorithm on that position.

Table 7: Different algorithms expressed as a combination of a formula class and a search paradigm.

Formula	Search Paradigm		
	DFTS	PNTS	MCPS
π_n	Depth-first search		Single-player MCTS [21]
WS_{α_n}	$\alpha\beta$ [14]	PNS [2]	MCTS Solver [31]
PA_n	Paranoid [26]	Paranoid PNS [19]	Multi-player MCTS [17]
$\mathrm{LS}_{d,n}$	Lambda-search [27]	Lambda-PNS [33][1]	
BRS_n	Best Reply Search [20]		
APS_n	Abstract proof search [6]		

[1] We actually need to change the update rule for the PN in internal $\phi_1 \wedge \phi_2$ nodes in PNTS from $\sum_C \mathrm{PN}$ to $\max_C \mathrm{PN}$.

4.1 One-player games

Many one-player games, the so-called puzzles, involve finding a path to a terminal state. Ideally this path should be the shortest possible. Examples of such puzzles include the 15-PUZZLE and RUBIK'S CUBE.

Recall that we defined a class of formulas for reachability in exactly n steps in Definition 3. Similarly we define now a class of formulas representing the existence of a path to a winning terminal state within n moves.

Definition 7. *We say that agent A has a* winning path *from a state q if q satisfies π_n where π_n is defined as $\pi_0 = w$ and $\pi_n = w \vee \Diamond_A \pi_{n-1}$ if $n > 0$.*

4.2 Two-player games

We already defined the *winning strategy* formulas WS_{α_n} and WS_{β_n} in Definition 4. We will now express a few other interesting formulas that can be satisfied in game states in two player games.

λ-Trees λ-trees have been introduced [27] as a generalisation of ladders as seen in Section 2.3. We will refrain from describing the intuition behind λ-trees here and will be satisfied with giving the formal corresponding property as they only constitute an example of the applicability of our framework.

Definition 8. *A state q has an* λ-tree *of order d and maximal depth n for player A if $q \models \mathrm{LS}_{\alpha_{d,n}}$, where $\mathrm{LS}_{\alpha_{d,n}}$ is defined as follows.*

- $\mathrm{LS}_{\alpha_{0,n}} = \mathrm{LS}_{\alpha_{d,0}} = \mathrm{LS}_{\beta_{0,n}} = \mathrm{LS}_{\beta_{d,0}} = w$
- $\mathrm{LS}_{\alpha_{d,n}} = w \vee \Diamond_A(\neg l \wedge \mathrm{LS}_{\alpha_{d-1,n-1}} \wedge \mathrm{LS}_{\beta_d,n-1})$
- $\mathrm{LS}_{\beta_{d,n}} = w \vee \Box_B(\neg l \wedge \mathrm{LS}_{\alpha_{d,n-1}})$

λ-trees are a generalisation of ladders as defined in Definition 5 since a ladder is a λ-tree of order $d = 1$.

Abstract proof trees Abstract proof trees were introduced to address some perceived practical limitations of $\alpha - \beta$ when facing a huge number of moves. They have been used to solve games such as PHUTBALL or ATARI-GO. We limit ourselves here to describing how we can specify in MMLK that a state is root to an an abstract proof tree. Again, we refer the reader to the literature for the intuition about abstract proof trees and their original definition [6].

Definition 9. *A state q has an* abstract proof tree *of order n for player A if $q \models \mathrm{APS}_{\alpha_n}$, where APS_{α_n} is defined as follows.*

- $\mathrm{APS}_{\alpha_0} = \mathrm{APS}_{\beta_0} = w$
- $\mathrm{APS}_{\alpha_n} = w \vee \Diamond_A(\neg l \wedge \mathrm{APS}_{\alpha_{n-1}} \wedge \mathrm{APS}_{\beta_{n-1}})$
- $\mathrm{APS}_{\beta_n} = w \vee \Box_B(\neg l \wedge \mathrm{APS}_{\alpha_{n-1}})$

Other concepts Many other interesting concepts can be similarly implemented via a class of appropriate formulas. Notably minimax search with iterative deepening, the Null-move assumption, and Dual Lambda-search [25] can be related to model checking some MMLK formulas with DFTS.

4.3 Multiplayer games

Paranoid Algorithm The Paranoid Hypothesis was developed to allow for more $\alpha - \beta$ style safe pruning in multi-player games [26]. It transforms the original $k + 1$-player into a two-player game A versus B. In the new game, the player B takes the place of $B^1, \ldots, B^k$ and B is trying to prevent player A from reaching a won position. Assuming the original turn order is fixed and is $A, B^1, \ldots, B^k, A, B^1, \ldots$, we can reproduce a similar idea in MMLK.

Definition 10. *Player A has a* paranoid win *of depth n in a state q if $q \models \mathrm{PA}_{\alpha_n}$, where PA_{α_n} is defined as follows.*

- $\mathrm{PA}_{\alpha_0} = \mathrm{PA}_{\beta_0^i} = w$
- $\mathrm{PA}_{\alpha_n} = w \vee \Diamond_A(\neg l \wedge \mathrm{PA}_{\beta^1_{n-1}})$
- $\mathrm{PA}_{\beta^k_n} = w \vee \Box_{B^k}(\neg l \wedge \mathrm{PA}_{\alpha_{n-1}})$
- $\mathrm{PA}_{\beta^i_n} = w \vee \Box_{B^i}(\neg l \wedge \mathrm{PA}_{\beta^{i+1}_{n-1}})$ *for* $1 \leq i < k$

Best Reply Search Best Reply Search (BRS) [20] is a new search algorithm for multi-player games. It consists of performing a minimax search where only one opponent is allowed to play after A. For instance a principal variation in a BRS search with $k = 3$ opponents could involve the following turn order $A, B_2, A, B_1, A, B_1, A, B_3, A, \ldots$ instead of the regular $A, B_1, B_2, B_3, A, B_1, B_2, B_3, \ldots$.

The rationale behind BRS is that the number of moves studied for the player in turn in any variation should only depend on the depth of the search and not on the number of opponents. This leads to an artificial player selecting moves exhibiting longer term planning. This performs well in games where skipping a move does not influence the global position too much, such as CHINESE CHECKERS.

Definition 11. *Player A has a* best-reply search win *of depth n in a state q if $q \models \mathrm{BRS}_{\alpha_n}$, where BRS_{α_n} is defined as follows.*

- $\mathrm{BRS}_{\alpha_0} = \mathrm{BRS}_{\beta_0} = w$
- $\mathrm{BRS}_{\alpha_n} = w \vee \Diamond_A(\neg l \wedge \mathrm{BRS}_{\beta_{n-1}})$
- $\mathrm{BRS}_{\beta_n} = w \vee \wedge_{i=1}^{k} \Box_{B^i}(\neg l \wedge \mathrm{BRS}_{\alpha_{n-1}})$

5 Creation of new game tree algorithms

We now turn to show how MMLK Model Checking framework can be used to develop new research in game tree search. As such, the goal of this section is not to put forward a single well performing algorithm, nor to prove strong theorems with elaborate proofs, but rather to demonstrate that the MMLK Model Checking is an appropriate tool for designing and reasoning about new game tree search algorithms.

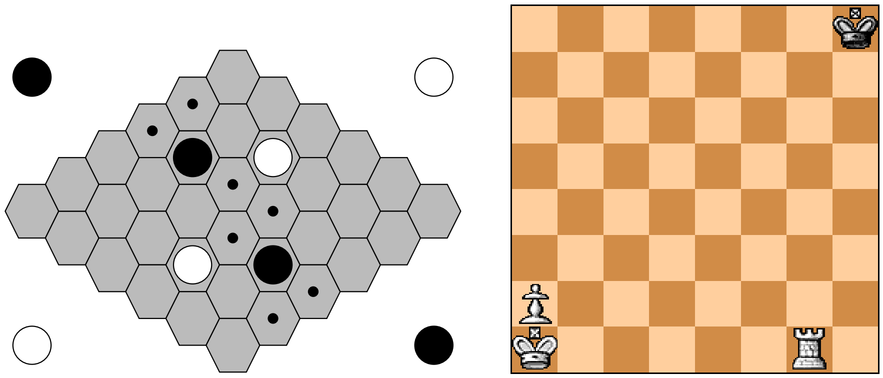
Progress Tree Search It occurs in many two-player games that at some point near the end of the game, one player has a winning sequence of n moves that is relatively independent of the opponent's moves. For instance Figure 2 presents a HEX position won for *Black* and a CHESS position won for *White*. In both cases, the opponent's moves cannot even delay the end of the game.

To capture this intuition, we define a solution concept we name *progress tree*. The idea giving its name to the concept of progress trees is that we want the player to focus on those moves that brings them closer to a winning state, and discard the moves that are out of the winning path.

Definition 12. *Player A has a* progress tree *of depth $2n+1$ in a state q if $q \models \mathrm{PT}_{\alpha_{2n+1}}$, where $\mathrm{PT}_{\alpha_{2n+1}}$ is defined as follows.*

- $\mathrm{PT}_{\beta_0} = w$
- $\mathrm{PT}_{\alpha_{2n+1}} = w \vee \Diamond_A(\neg l \wedge \pi_n \wedge \mathrm{PT}_{\beta_{2n}})$
- $\mathrm{PT}_{\beta_{2n}} = w \vee (\neg l \wedge \Box_B \, \mathrm{PT}_{\alpha_{2n-1}})$

We can check states for progress trees using any of the model checking algorithms presented in Section 3, effectively giving rise to three new specialised algorithms. Note that if a player has a progress tree of depth $2n+1$ in some state, then they also have a winning strategy of depth $2n+1$ from that state (see Proposition 2). Therefore, if we



(a) HEX position featuring a *progress tree* of depth 7 for *Black*.

(b) CHESS endgame featuring a *progress tree* of depth 11 for *White*.

Fig. 2: Positions illustrating the concepts of *progress tree*.

prove that a player has a progress tree in some position, then we can deduce that have a winning strategy.

We tested a naive implementation of the DFTS model checking algorithms on the position in Figure 2 to check for progress trees and winning strategies. The principal variations consists for *White* in moving the pawn up to the last row and move the resulting queen to the bottom-right hand corner to deliver checkmate. To study how the solving difficulty increases with respect to the size of the formula to be checked, we model checked every position on a principal variation and present the results in Table 8.

We can see that proving that a progress tree exists becomes significantly faster than proving an arbitrary winning strategy as the size of the problem increases. We can also notice that the overhead of checking for a path at each α node of the search is more than compensated by the early pruning of moves not contributing to the winning strategy.

Examining new combinations We have seen in Section 3 that we could obtain previously known algorithms by combining model checking algorithms with solution concepts. On the one hand, some solution concepts such a winning strategy and paranoid win, were combined with the three possible search paradigms in previous work. On the other hand, other solution concepts such as best-reply search win were only investigated within the depth-first paradigm.

It is perfectly possible to model check a best-reply search win using the MCPS algorithm, for instance, leading to a new Monte Carlo Best Reply Search algorithm. Similarly model checking abstract proof trees with PNTS would lead to a new Proof Number based Abstract Proof Search (PNAPS) algorithm. Preliminary experiments in HEX without any specific domain knowledge added seem to indicate that PNAPS does not seem to perform as well as Abstract Proof Search, though.

Finally, most of the empty cells in Table 7 can be considered as new algorithms waiting for an optimised implementation and a careful evaluation.

Table 8: Search statistics for a DFTS on positions along a principal variation of the CHESS problem in Figure 2b.

MC problem	Time (s)	Number of queries		
		`atomic`	`listmoves`	`play`
PT_{α_5}	0.1	6040	328	5897
WS_{α_5}	0.2	11172	624	5587
PT_{α_7}	1.4	99269	5312	98696
WS_{α_7}	3.5	194429	10621	97217
PT_{α_9}	23.6	1674454	88047	1668752
WS_{α_9}	63.8	3382102	181442	1691055
$\mathrm{PT}_{\alpha_{11}}$	260.4	25183612	1297975	25106324
$\mathrm{WS}_{\alpha_{11}}$	953.6	52209939	2759895	26104986

Expressing properties of the algorithms We now demonstrate that using the MMLK model checking framework for game tree search makes some formal reasoning straightforward. Again, the goal of this section is not to demonstrate strong theorems with elaborate proofs but rather show that the framework is convenient for expressing certain properties and helps reasoning on them.

It is easy to prove by induction on the depth that lambda trees, abstract proof trees, and progress trees are all refinements of winning strategies.

Proposition 2. *For all order d and depth n, we have* $\mathrm{LS}_{\alpha_{d,n}} \Rightarrow \mathrm{WS}_{\alpha_n}$, $\mathrm{APS}_{\alpha_n} \Rightarrow \mathrm{WS}_{\alpha_n}$, *and* $\mathrm{PT}_{\alpha_n} \Rightarrow \mathrm{WS}_{\alpha_n}$.

Therefore, whenever we succeed in proving that a position features, say, a lambda tree, then we know it also has a winning strategy for the same player: $\forall q, q \models \mathrm{LS}_{\alpha_{d,n}} \rightarrow q \models \mathrm{WS}_{\alpha_n}$.

On the other hand, in many games, it is possible to have a position featuring a winning strategy but no lambda tree, abstract proof tree, or even progress tree. Before studying the other direction further, we need to rule out games featuring *zugzwangs*, that is, positions in which a player would rather pass and let an opponent make the next move.

Definition 13. *A* ϕ-zugzwang *for player A against players $B^1, \ldots, B^k$ is a state q such that $q \models \neg\phi \wedge (\Box_{B^1}\,\phi \vee \cdots \vee \Box_{B^k}\,\phi)$. A game is* zugzwang-free *for a set of formulas Φ and player A against players $B^1, \ldots, B^k$ if for every state q, and every formula $\phi \in \Phi$, q is not a ϕ-zugzwang for A against $B^1, \ldots, B^k$.*

The usual understanding of zugzwang is in two player games with ϕ a winning strategy formula or a formula representing forcing some material gain in CHESS.

We can now use this definition to show that in games zugzwang-free for winning strategies, such as HEX or CONNECT-6, an abstract proof tree and a progress tree are equivalent to a winning strategy of the same depth.

Proposition 3. *Consider a two-player game zugzwang-free for winning strategies. For any depth n and any state q, $q \models \mathrm{APS}_{\alpha_n} \leftrightarrow q \models \mathrm{PT}_{\alpha_n} \leftrightarrow q \models \mathrm{WS}_{\alpha_n}$.*

6 Conclusion and discussion

We have defined a general way to express the shape of a search tree using MMLK. We have shown it is possible to use different search strategies to search the tree shape. This combination of a tree shape and of a search strategy yields a variety of search algorithms that can be modelled in the same framework. This makes it easy to combine strategies and shapes to test known algorithms as well to define new ones.

We have shown that the Multi-Agent Modal Logic K was a convenient tool to express various kind of threats in a game independant way. Victor Allis provided one of the earliest study of the concept of threats in his *Threat space search* algorithm used to solve GOMOKU [1].

Previous work by Schaeffer et al. was also concerned with providing a unifying view of heuristic search and the optimizations tricks that appeared in both single-agent search and two-player game search [23].

Another trend of related previous work is connecting modal logic and game theory [29, 32, 15]. In this area, the focus is on the concept of Nash equilibria, extensive form games, and coalition formation. As a result, more powerful logic than the restricted MMLK are used [3, 30, 12]. Studying how the model checking algorithms presented in this article can be extended for these settings is an interesting path for future work.

The model used in this article differs from the one used in General Game Playing (GGP) called Multi-Agent Environment (MAE) [24]. In an MAE, a transition correspond to a joint-action. That is, each player decide a move simultaneously and the combinaison of these moves determines the next state. In a GA, as used in this article, the moves are always sequential. It is possible to simulate sequential moves in an MAE by using *pass* moves for the non acting agents, however this ties the turn player into the game representation. As a result, testing for solution concepts where the player to move in a given position is variable is not possible with an MAE. For instance, it is not possible to formally test for the existence of a ladder in a GGP representation of the game of GO because we need to compute the successors of a given position after a white move and alternatively after a black move.

Effective handling of transpositions is another interesting topic for future work. It is already nontrivial in PNS [13] and MCTS [18], but it is an even richer subject in this model checking setting as we might want to prove different facts about a given position in the same search.

Table 7 reveals many interesting previously untested possible combinations of formula classes and search algorithms. Implementing and optimising one specific new combination for a particular game could lead to insightful practical results. For instance, it is quite possible that a Monte Carlo version of Best Reply Search would be successful in MULTIPLAYER GO [7].

References

[1] Louis Victor Allis, H. Jaap van den Herik, and M. P. H. Huntjens. Go-Moku solved by new search techniques. *Computational Intelligence*, 12:7–23, 1996.

[2] Louis Victor Allis, M. van der Meulen, and H. Jaap van den Herik. Proof-Number Search. *Artificial Intelligence*, 66(1):91–124, 1994.

[3] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[4] Peter Auer, Nicoló Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[5] Patrick Blackburn, Maarten De Rijke, and Yde Venema. *Modal Logic*, volume 53. Cambridge University Press, 2001.

[6] Tristan Cazenave. Abstract Proof Search. In T. Anthony Marsland and Ian Frank, editors, *Computers and Games 2000*, volume 2063 of *Lecture Notes in Computer Science*, pages 39–54. Springer, Berlin / Heidelberg, 2002.

[7] Tristan Cazenave. Multi-player Go. In H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H.M. Winands, editors, *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, pages 50–59. Springer, Berlin / Heidelberg, 2008.

[8] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game AI. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 216–217. AAAI Press, 2008.

[9] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. van den Herik, Paolo Ciancarini, and H. Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer, Berlin / Heidelberg, 2007.

[10] Sylvain Gelly and David Silver. Achieving master level play in 9×9 computer Go. In *Proceedings of the 23rd national conference on Artifical Intelligence (AAAI'08)*, pages 1537–1540, 2008.

[11] Michael Genesereth and Nathaniel Love. General game playing: Overview of the AAAI competition. *AI Magazine*, 26:62–72, 2005.

[12] Valentin Goranko and Govert van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theoretical Computer Science*, 353(1):93–117, 2006.

[13] Akihiro Kishimoto and Martin Müller. A solution to the GHI problem for depth-first proof-number search. *Information Sciences*, 175(4):296–314, 2005.

[14] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[15] Lena Kurzen. *Complexity in Interaction*. PhD thesis, Universiteit van Amsterdam, 2011.

[16] Martin Müller. Computer go. *Artificial Intelligence*, 134(1-2):145–179, 2002.

[17] J.A.M. Nijssen and Mark H.M. Winands. Enhancements for multi-player Monte-Carlo Tree Search. In H. van den Herik, Hiroyuki Iida, and Aske Plaat, editors, *Computers and Games*, volume 6515 of *Lecture Notes in Computer Science*, pages 238–249. Springer, Berlin / Heidelberg, 2011.

[18] Abdallah Saffidine, Tristan Cazenave, and Jean Méhat. UCD: Upper Confidence bound for rooted Directed acyclic graphs. *Knowledge-Based Systems*, December 2011.

[19] Jahn-Takeshi Saito and Mark H.M. Winands. Paranoid Proof-Number Search. In Georgios N. Yannakakis and Julian Togelius, editors, *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG-2010)*, pages 203–210, 2010.

[20] Maarten P.D. Schadd and Mark H.M. Winands. Best reply search for multiplayer games. *IEEE Transactions Computational Intelligence and AI in Games*, 3(1):57–66, 2011.

[21] Maarten P.D. Schadd, Mark H.M. Winands, H. Jaap van den Herik, Guillaume M. J.-B. Chaslot, and Jos W.H.M. Uiterwijk. Single-player monte-carlo tree search. In H. van den Herik, Xinhe Xu, Zongmin Ma, and Mark Winands, editors, *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, pages 1–12. Springer, Berlin / Heidelberg, 2008.

[22] Jonathan Schaeffer. A gamut of games. *AI Magazine*, 22(3):29–46, 2001.

[23] Jonathan Schaeffer, Aske Plaat, and Andreas Junghanns. Unifying single-agent and two-player search. *Information Sciences*, 135(3-4):151–175, July 2001.

[24] Stephan Schiffel and Michael Thielscher. A multiagent semantics for the game description language. In Joaquim Filipe, Ana Fred, and Bernadette Sharp, editors, *Agents and Artificial Intelligence*, volume 67 of *Communications in Computer and Information Science*, pages 44–55. Springer, Berlin / Heidelberg, 2010.

[25] Shunsuke Soeda, Tomoyuki Kaneko, and Tetsuro Tanaka. Dual lambda search and shogi endgames. In H. van den Herik, Shun-Chin Hsu, Tsan sheng Hsu, and H. Donkers, editors, *Advances in Computer Games*, volume 4250 of *Lecture Notes in Computer Science*, pages 126–139. Springer, Berlin / Heidelberg, 2006.

[26] Nathan R. Sturtevant and Richard E. Korf. On pruning techniques for multi-player games. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, AAAI/IAAI 2000, pages 201–207, 2000.

[27] Thomas Thomsen. Lambda-search in game trees - with application to Go. *ICGA Journal*, 23(4):203–217, 2000.

[28] H. Jaap van den Herik and Mark Winands. Proof-Number Search and its variants. *Oppositional Concepts in Computational Intelligence*, pages 91–118, 2008.

[29] Wiebe van der Hoek and Marc Pauly. Modal logic for games and information. *Handbook of modal logic*, 3:1077–1148, 2006.

[30] Wiebe van der Hoek and Michael Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

[31] Mark H.M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte-Carlo tree search solver. In H. Japp van den Herik, Xinhe Xu, Zongmin Ma, and Mark H.M. Winands, editors, *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, pages 25–36. Springer, Berlin / Heidelberg, 2008.

[32] Michael Wooldridge, Thomas Agotnes, Paul E. Dunne, and Wiebe van der Hoek. Logic for automated mechanism design — a progress report. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-07)*, volume 22, page 9. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[33] Kazuki Yoshizoe, Akihiro Kishimoto, and Martin Müller. Lambda Depth-First Proof Number Search and its application to Go. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2404–2409, 2007.