# Stochastic Boolean Satisfiability for Two-Player Sequential Games under Uncertainty

Chi-Kit Ng*†, Chih-Hsiang Chuang*†, Jie-Hong R. Jiang†‡
†Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
‡Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
Email: {r14943168, r14943082, jhjiang}@ntu.edu.tw


***Abstract*—Stochastic Boolean satisfiability (SSAT) provides a compact framework for PSPACE-complete decision-making under uncertainty. Despite the theoretical inclusion of universal quantifiers (∀), existing solvers typically restrict inputs to random (R) and existential (∃) quantifiers, limiting applications to single-player games against nature. In this work, we generalize SSAT to include universal quantification, enabling the modeling of competitive two-player games under uncertainty. Theoretically, we establish a reduction from finite two-player zero-sum stochastic games with perfect information to general SSAT. Unlike QBF-based solvers that yield binary deterministic outcomes, our approach computes rational-valued Nash equilibrium payoffs as satisfying probabilities. Practically, we extend SharpSSAT to create the first solver capable of handling all three quantifier types. It supports equilibrium value computation and generates strategies for both players (Skolem for ∃ and Herbrand for ∀). By utilizing an implicit representation, our solver offers a more scalable alternative to traditional explicit-state game theory algorithms. Experimental results demonstrate that our solver effectively reasons about complex stochastic adversarial interactions that were previously difficult to encode.**

## I. Introduction

*Stochastic Boolean satisfiability* (SSAT), first presented as games against nature [1], is a powerful logical formalism for decision-making under uncertainty. SSAT generalizes the well-known Quantified Boolean Satisfiability (QSAT) problem by incorporating random quantifiers alongside existential and, in some formulations, universal quantifiers. QSAT can be reduced in polynomial time to SSAT with only existential and random quantifiers, and the decision problem for solving such formulas is PSPACE-complete, the same complexity as QSAT. This PSPACE-completeness persists even when universal quantification is permitted [1].

SSAT solvers are under active development, much motivated by emerging applications in optimization and probabilistic reasoning, as well as the progress in SAT solving and model counting [2]. Several SSAT solvers have been developed to date, such as `DC-SSAT` [3], `SSAT-Prime` [4], `ClauSSat` [5], `ElimSSAT` [6], and `SharpSSAT` [7]. However, these solvers are designed under the assumption that SSAT formulas include only existential and random quantifiers. Although this assumption works for a variety of applications, e.g., in artificial intelligence and electronic design automation, it limits expressiveness in domains that naturally involve adversarial interactions. To the best of our knowledge, no existing method is known to reduce a prenex-CNF SSAT formula containing both universal and existential quantifiers to one with only universal or existential quantifiers, except for an indirect Turing machine conversion. Consequently, existing solvers cannot be applied to these adversarial problems without significant modification.

This limitation motivates the extension of SSAT solvers to support general SSAT formulas that incorporate existential, universal, and random quantifiers. Although the addition of universal quantification does not increase the computational complexity beyond PSPACE, it introduces significant theoretical and practical challenges. First, from a game-theoretic perspective, the natural interpretation is a three-agent setting: the existential and universal players, with competing objectives, both act under uncertainty generated from random variables representing nature, a disinterested agent. This interpretation is formally defined in Section III. Second, from a practical standpoint, in addition to solving such formulas for the equilibrium value, it is also desirable to give strategy profiles of the equilibrium, expressed as Skolem and Herbrand functions, for the existential and universal players, respectively.

The utility of this extension can be demonstrated by finite (in the horizon and decision space) two-player zero-sum stochastic games with perfect information, where one player aims to prevent the other from reaching a goal. Such games can be naturally encoded as SSAT formulas, where the existential player intends to satisfy the formula and the universal player intends to falsify it. We establish in Section IV a polynomial-time reduction (Theorem 3) from finite two-player zero-sum stochastic games with perfect information to SSAT problems, demonstrating that the satisfying probability of the SSAT formula corresponds to the Nash equilibrium payoff for the existential player. This connection underscores the potential for applying SSAT as a tool for computing the equilibrium of two-player zero-sum games under uncertainty.

The computational complexity of finding Nash equilibria has been extensively studied. For instance, both the computation and approximation of Nash equilibria in two-player games have been shown to be PPAD-complete [8]. However, these results primarily focus on explicit game representations, i.e., matrix games or normal-form games, where the payoff values for all possible strategies are stored explicitly in tables.

*These authors contributed equally to this work.

https://doi.org/10.34727/2026/isbn.978-3-85448-093-8_64

In contrast, finite-state machine games defined in Section IV represent payoffs implicitly and thus scale to settings where strategy–payoff enumeration is infeasible.

A closely related line of research involves *Boolean games*, as introduced in [9], [10]. Boolean games are two-player, zero-sum, simultaneous-move games represented by a single propositional formula and a variable partition. Deciding the existence of a pure Nash equilibrium for Boolean games is $\Sigma_2^p$-complete [11], while computing the equilibrium payoffs is EXP-complete [12], [13]. Despite their similarities, Boolean games and finite-state machine games differ in that the former is deterministic and simultaneous (thus having imperfect information), while the latter is probabilistic and sequential with perfect information.

In the realm of sequential perfect-information games, a recent work [14] applied QBF solvers to compute winning strategies for general two-player zero-sum games described in the Game Description Language (GDL), a logic language based on normal logic program syntax. These games are specified using stable model semantics rather than classical propositional logic, necessitating translations to answer set programming. While effective for deterministic environments with binary win-loss outcomes, such QBF-based approaches do not naturally accommodate stochastic elements. Our work aims to bridge this gap by applying SSAT to general game solving, thereby enabling reasoning under uncertainty and rational-valued outcomes.

In this paper, we present extensions of existing SSAT solving techniques and incorporate them into the SSAT solver `SharpSSAT`, which is built upon the CNF model counter `sharpSAT` [15] and supports efficient Skolem function generation. In Section V, we extend `SharpSSAT` to not only compute the satisfying probability of general SSAT formulas, but also to extract corresponding strategy functions for both players. Experimental results are shown in Section VI, which demonstrate the practical utility of the solver in efficiently computing Nash equilibrium strategies for finite two-player zero-sum stochastic games with perfect information, particularly showcasing its scalability advantage over existing exact algorithms.

## II. Preliminaries

In this work, symbols "$\top$" and "$\bot$" denote Boolean values *true* and *false*, respectively. Let $\mathbb{B} = \{\top, \bot\}$. Symbols "$\neg$", "$\vee$", "$\wedge$", "$\leftrightarrow$" denote the usual Boolean connectives negation, disjunction, conjunction, and equivalence respectively. A *literal* is either a Boolean variable or the negation of a Boolean variable. For a literal $l$, $\mathrm{var}(l)$ denotes the underlying variable of $l$. A literal $l$ with $\mathrm{var}(l) = x$ is called *positive* if $l = x$ and *negative* if $l = \neg x$. A *clause* is a disjunction of literals with unique underlying variables. A *conjunctive normal form* (*CNF*) formula is a conjunction of a set of clauses. $\mathrm{Vars}(\phi)$ for a formula $\phi$ denotes the set of all variables on which $\phi$ depends. An *assignment* $\tau : \mathcal{A} \to \mathbb{B}$ is a mapping from a subset of variables $\mathcal{A} \subseteq \mathrm{Vars}(\phi)$ to Boolean values. For convenience, an assignment can also be specified by a conjunction of literals (also called a *cube*), representing the assignment to the underlying variables such that each literal in the set evaluates to $\top$. An assignment is called *complete* if $\mathcal{A} = \mathrm{Vars}(\phi)$. The formula $\phi$ under assignment $\tau$, denoted $\phi[\tau]$ or $\phi_\tau$ and also called the *cofactor* of $\phi$ with respect to $\tau$, is obtained by the substitution $\phi[\tau(x)/x]$ for each variable $x \in \mathcal{A}$ with its assigned value $\tau(x)$.

A *stochastic Boolean satifiability* (SSAT) formula $\Phi$ over a set of variables $X = \{x_1, \ldots, x_n\}$ can be expressed in the form $\Phi = Q_1 x_1 \ldots Q_n x_n.\phi$, where $Q_i \in \{\exists, \forall, \mathrm{R}^p\}$ is either an existential quantifier $\exists$, a universal quantifier $\forall$, or a random quantifier $\mathrm{R}^p$ associated with a probability $p$, and $\phi$ is a propositional formula with $\mathrm{Vars}(\phi) \subseteq X$. We refer to $\mathcal{Q} = Q_1 x_1, \ldots, Q_n x_n$ and $\phi$ as the prefix and matrix of $\Phi$, respectively. Also we refer to $X_E = \{x_i \mid Q_i = \exists\}$, $X_A = \{x_i \mid Q_i = \forall\}$, and $X_R = \{x_i \mid Q_i = \mathrm{R}^p\}$ as the set of existential, universal, and random variables, respectively. The *dependency set* $D_i$ of variable $x_i$ is $\{x_j \in X_A \cup X_R \mid j < i\}$ if $x_i \in X_E$ and $\{x_j \in X_E \cup X_R \mid j < i\}$ if $x_i \in X_A$.

Given an SSAT formula $\Phi$, the *satisfying probability* of $\Phi$ is computed recursively by the rules: $\Pr[\top] = 1$, $\Pr[\bot] = 0$, $\Pr[\exists x \Phi] = \max_{b \in \mathbb{B}} \Pr[\Phi[b/x]]$, $\Pr[\forall x \Phi] = \min_{b \in \mathbb{B}} \Pr[\Phi[b/x]]$, and $\Pr[\mathrm{R}^p\, x \Phi] = p \Pr[\Phi[\top/x]] + (1 - p) \Pr[\Phi[\bot/x]]$. The *function problem* of SSAT asks for the value of $\Pr[\Phi]$, while the *decision problem* of SSAT asks to decide whether $\Pr[\Phi] \geq \theta$ for some threshold $\theta$.

Building on the concepts of Skolem-function models and Herbrand-function countermodels for quantified Boolean formulas [16], we define a *Skolem strategy* $\mathbf{s}_\exists \in S_\exists$ as a collection of Skolem functions $s_{\exists,i}$ for each existential variable $x_i \in X_E$. Each function $s_{\exists,i}$ is a propositional formula with $\mathrm{Vars}(s_{\exists,i}) \subseteq D_i$. Analogously, a *Herbrand strategy* $\mathbf{s}_\forall \in S_\forall$ consists of Herbrand functions $s_{\forall,i}$ for each universal variable $x_i \in X_A$, where each $s_{\forall,i}$ is a propositional formula with $\mathrm{Vars}(s_{\forall,i}) \subseteq D_i$.

The satisfying probability of an SSAT formula $\Phi = \mathcal{Q}.\phi$ under a Skolem (resp. Herbrand) strategy $\mathbf{s}_\exists$ (resp. $\mathbf{s}_\forall$) is defined as $\Pr[\Phi[\mathbf{s}_\exists]]$ (resp. $\Pr[\Phi[\mathbf{s}_\forall]]$), where $\Phi[\mathbf{s}_\exists]$ (resp. $\Phi[\mathbf{s}_\forall]$) denotes the formula obtained by substituting each occurrence of $x_i \in X_E$ (resp. $x_i \in X_A$) in the matrix $\phi$ with $s_{\exists,i}$ (resp. $s_{\forall,i}$). Note that the prefix $\mathcal{Q}$ after the substitution can be reduced to only containing universal (resp. existential) and random quantifiers when a Skolem (resp. Herbrand) strategy is applied to $\Phi$. Alternatively, we may also substitute both Skolem and Herbrand strategies in $\Phi$, denoted by $\Phi[\mathbf{s}_\exists, \mathbf{s}_\forall]$, and in this case, the substitution starts from the innermost non-random quantified variable, and the prefix $\mathcal{Q}$ can be reduced to only containing random quantifiers. A Skolem strategy $\mathbf{s}_\exists^*$ is called *optimal* if $\Pr[\Phi[\mathbf{s}_\exists^*]] \geq \Pr[\Phi[\mathbf{s}_\exists]]$ for all $\mathbf{s}_\exists \in S_\exists$. Similarly, a Herbrand strategy $\mathbf{s}_\forall^*$ is *optimal* if $\Pr[\Phi[\mathbf{s}_\forall^*]] \leq \Pr[\Phi[\mathbf{s}_\forall]]$ for all $\mathbf{s}_\forall \in S_\forall$.

We also extend the notion of the application of an assignment $\tau$ to strategies by defining $\mathbf{s}_\exists[\tau]$ to be $s_{\exists,i}[\tau]$ for every $i$ such that $x_i \in X_E$.

## III. SSAT as Two-player Games

We adopt standard game-theoretic notation, based on the framework established by [17], to formalize the interpretation of SSAT as an alternating-move game.

**Definition 1.** *A* finite two-player zero-sum stochastic game with perfect information *is a seven-tuple* $(T, \mathcal{P}, u, \pi, \{A_v\}_{v\in V\setminus Z}, \{\delta_v\}_{v\in V\setminus Z}, d)$*, where* $T = (V, E)$ *is a finite directed rooted tree, with an initial game state (root node)* $v_0 \in V$ *and terminal game states (leaf nodes)* $Z \subseteq V$*,* $\mathcal{P} = (P_1, P_2)$ *consists of the maximizing player* $P_1$ *and minimizing player* $P_2$*,* $u : Z \to \mathbb{R}$ *is the payoff function for the maximizing player,* $\pi : V \setminus Z \to \{P_1, P_2, N\}$ *is a label on the internal nodes, with* $\pi(v) = P_i$ *indicating that it is the move of* $P_i$ *in the game state* $v$*, and* $\pi(v) = N$ *indicating that it is the random move of the nature agent,* $A_v$ *is a finite set of possible actions of* $\pi(v)$ *for each* $v \in V \setminus Z$*,* $\delta_v : A_v \to \{w \in V \mid E(v, w)\}$ *is a bijection between actions and the next states for each* $v \in V\setminus Z$*, and* $d(v)$ *is a probability distribution of* $A_v$ *when* $\pi(v) = N$ *[17].*

This game model assumes perfect information in the sense that at every possible game state, each player knows the complete history of prior actions, including random moves, and can anticipate the resulting state for any sequence of actions and random moves. Since the game tree is finite, such games have both a finite horizon (i.e., tree height) and finite decision choices (i.e., actions at each state).

While general multi-player games may assign distinct, potentially uncorrelated payoff functions to each player, two-player zero-sum games are strictly competitive. Specifically, one player's gain is precisely the other's loss. Thus, specifying a payoff function for the maximizing player $P_1$ suffices, as $P_2$'s payoff is implicitly defined as its negation.

A *(pure) strategy* for player $P_i$, for $i \in \{1, 2\}$, is an element of the strategy space as a Cartesian product $S_i = \prod_{v:\pi(v)=P_i} A_v$, which is the collection of all the actions available for the player in each state. A strategy profile is a tuple containing one strategy for each player. Given a strategy profile $(\mathbf{s}_1, \mathbf{s}_2) \in S_1 \times S_2$, the probability distribution of terminal game states $Z$ is determined by $d$. The expected payoff of the game corresponding to the strategy profile $(\mathbf{s}_1, \mathbf{s}_2)$ is denoted $\mathbb{E}[u|\mathbf{s}_1, \mathbf{s}_2]$.

A strategy profile $(\mathbf{s}_1^*, \mathbf{s}_2^*) \in S_1 \times S_2$ is a (pure-strategy) *Nash equilibrium* if $\mathbb{E}[u|\mathbf{s}_1^*, \mathbf{s}_2^*] \geq \mathbb{E}[u|\mathbf{s}_1, \mathbf{s}_2^*]$ for all $\mathbf{s}_1 \in S_1$, and $\mathbb{E}[u|\mathbf{s}_1^*, \mathbf{s}_2^*] \leq \mathbb{E}[u|\mathbf{s}_1^*, \mathbf{s}_2]$ for all $\mathbf{s}_2 \in S_2$, i.e., no player can improve their expected payoff by changing their own strategy.

We interpret an SSAT formula $\Phi = \mathcal{Q}.\phi$ with variables $x_1, \ldots, x_n$ as a two-player zero-sum game with perfect information. Define the set of game states as $V = \mathbb{B}^{\leq n} = \bigcup_{i=0}^{n} \mathbb{B}^i$, i.e., the set of bit strings of length up to $n$. Note that $v = \mathbb{B}^0$, i.e., the empty string, corresponds to the root node. Also, define the edge relation as $(v, w) \in E$ iff $w = vb$, i.e., the concatenation of bit string $v$ with a bit $b \in \mathbb{B}$. Note that the terminal states correspond to the complete assignments $z \in \mathbb{B}^n$. The players $P_1$ and $P_2$ correspond to the existential and universal players, respectively. The payoff function is defined as $u(z) = 1$ if $\phi[z_1/x_1, \ldots, z_n/x_n] = \top$, and $u(z) = 0$ if $\phi[z_1/x_1, \ldots, z_n/x_n] = \bot$. For a node $v \in \mathbb{B}^i$, let $\pi(v) = P_1$ if $x_{i+1}$ is existentially quantified, $\pi(v) = P_2$ if universally quantified, and $\pi(v) = N$ if randomly quantified. The action set at each non-terminal state is $A_v = \mathbb{B}$, and the transition function is $\delta_v(b) = vb$. The probability distribution $d(v)$ on actions is defined so that $d(v, \{\top\}) = p$ when $x_{i+1}$ is quantified by $\mathfrak{A}^p$. We refer to this interpretation as the *SSAT game* induced by $\Phi$.

We remark that for a node $v \in \mathbb{B}^i$, the SSAT game of $\Phi_v = \Phi[v]$ can be viewed as a subgame of $\Phi$ induced by $v$, with game states $V_v = \bigcup_{k=i}^{n} \mathbb{B}^k$ and strategy space $S_{vj} = \prod_{w:\pi(vw)=P_j} A_w$, $j \in \{1, 2\}$. Skolem and Herbrand strategies correspond precisely to pure strategies in $S_\exists = S_1$ and $S_\forall = S_2$, respectively, modulo choices made at "don't care" game states when a player fixes a strategy. This correspondence is reflected in the fact that Skolem strategies do not depend on existential variables, and dually, Herbrand strategies do not depend on universal variables. Moreover, for any pair of Skolem and Herbrand strategies $\mathbf{s}_1$ and $\mathbf{s}_2$, the satisfying probability $\Pr[\Phi[\mathbf{s}_1, \mathbf{s}_2]]$ coincides with the expected payoff $\mathbb{E}[u|\mathbf{s}_1, \mathbf{s}_2]$ of the strategy profile.

In general, a game may lack a pure Nash equilibrium, as illustrated by the well-known rock-paper-scissors game, which involves simultaneous moves and thus falls outside the class of games with perfect information [18]. However, for the special case of games with perfect information, there always exists a pure strategy Nash equilibrium, which can be computed via backward induction [19]. This strategy is *subgame-perfect*, which means that every subgame achieves Nash equilibrium under this strategy profile. In our context, the value $\Pr[\Phi]$ corresponds definitionally to the expected equilibrium value of the pure strategy equilibrium derived by backward induction on the SSAT game.

**Lemma 1** (Equilibrium existence [19])**.** *A finite two-player stochastic game with perfect information always has a Nash equilibrium in pure strategies.*

Another issue is the uniqueness of payoffs in Nash equilibria. In general, a game may admit multiple equilibria with different expected payoffs, e.g., the Battle of the Sexes game [20]. However, for two-player zero-sum games, a classical result states that the expected payoff for any equilibrium is the same.

**Theorem 1** (Minimax theorem [21])**.** *For any finite two-player zero-sum stochastic game with perfect information, where* $P_1$ *is the maximizing player and* $P_2$ *is the minimizing player, we have*

$$\min_{s_2\in S_2} \max_{s_1\in S_1} \mathbb{E}[u|s_1, s_2] = \max_{s_1\in S_1} \min_{s_2\in S_2} \mathbb{E}[u|s_1, s_2].$$

The minimax theorem supports the faithfulness of our game-theoretic interpretation of SSAT formulas. That is, if both

players select their optimal strategies, the existential player can always expect a reward of $\Pr[\Phi]$.

Furthermore, the following lemma asserts that the value obtained by following a strategy is bounded by the equilibrium value of the game if the opponent acts optimally.

**Lemma 2.** *For every possible Skolem strategy $\mathbf{s}_\exists$ of an SSAT formula $\Phi$, we have $\Pr[\Phi] \geq \Pr[\Phi[\mathbf{s}_\exists]]$. Similarly, for every possible Herbrand strategy $\mathbf{s}_\forall$ of an SSAT formula $\Phi$, we have $\Pr[\Phi] \leq \Pr[\Phi[\mathbf{s}_\forall]]$.*

The detailed proof, by induction on the structure of $\Phi$, can be found in the Appendix at [22].

**Theorem 2.** *Given an SSAT formula $\Phi$, we have*

$$\min_{\mathbf{s}_\forall \in S_\forall} \Pr[\Phi[\mathbf{s}_\forall]] = \max_{\mathbf{s}_\exists \in S_\exists} \Pr[\Phi[\mathbf{s}_\exists]] = \Pr[\Phi],$$

*Proof.* By Lemma 1, there is a pure strategy Nash equilibrium $(\tilde{s}_\exists, \tilde{s}_\forall)$, obtained by backward induction, such that $\Pr[\Phi[\tilde{s}_\exists, \tilde{s}_\forall]] = \Pr[\Phi]$. Therefore,

$$\Pr[\Phi[\tilde{\mathbf{s}}_\forall]] = \max_{\mathbf{s}_\exists \in S_\exists} \Pr[\Phi[\mathbf{s}_\exists, \tilde{\mathbf{s}}_\forall]] = \Pr[\Phi[\tilde{\mathbf{s}}_\exists, \tilde{\mathbf{s}}_\forall]] = \Pr[\Phi],$$

where the first equality holds by induction, as in the case of SSAT without universal quantifications. By Lemma 2, it follows that $\min_{\mathbf{s}_\forall \in S_\forall} \Pr[\Phi[\mathbf{s}_\forall]] = \Pr[\Phi]$. Similarly, $\max_{\mathbf{s}_\exists \in S_\exists} \Pr[\Phi[\mathbf{s}_\exists]] = \Pr[\Phi]$. □

## IV. Encoding Two-player Games with SSAT

In this section, we highlight the practical expressiveness of the generalized SSAT in addressing two-player games, which serves as motivation for further research in SSAT solving. We begin by providing a concrete representation for the games defined in Definition 1. Next, we demonstrate a polynomial reduction from these games to SSAT formulas. This transformation allows us to convert a wide range of two-player games into SSAT problems, thus enabling the application of SSAT solvers with strategy generation capabilities for two-player games.

**Definition 2.** *A* finite-state machine game *is a tuple*

$$\Gamma = (\mathbf{y}, c, \mathbf{a_1}, \mathbf{a_2}, \mathbf{r}, (p_1, \ldots, p_n), \delta, \mathbf{u}, h)$$

*where $\mathbf{y} = (y_1, \ldots, y_q)$ are the state variables, $c \in \mathbb{B}^q$ is the initial state, $\mathbf{a_1} = (a_{1,1}, \ldots, a_{1,m_1})$ and $\mathbf{a_2} = (a_{2,1}, \ldots, a_{2,m_2})$ are the input variables from players $P_1$ and $P_2$, respectively, $\mathbf{r} = (r_1, \ldots, r_n)$ are the Bernoulli random variable inputs, with $(p_1, \ldots, p_n)$ being the probabilities for each variable to be $\top$, $\delta : \mathbb{B}^q \times \mathbb{B}^{m_1} \times \mathbb{B}^{m_2} \times \mathbb{B}^n \to \mathbb{B}^q$ is the state-transition function, $\mathbf{u} = (u_0, \ldots, u_{m_u-1})$, for $\mathbf{u} : \mathbb{B}^q \to \mathbb{B}^{m_u}$, is the integer payoff function represented in binary of length $l$, and $h \in \mathbb{N}$ is the horizon. In each game state $\mathbf{y}$, the game evolves according to the following steps in sequence: 1) player $P_1$ selects an action $\mathbf{a_1} \in \mathbb{B}^{m_1}$, 2) player $P_2$ observes $\mathbf{a_1}$ and selects an action $\mathbf{a_2} \in \mathbb{B}^{m_2}$, 3) a list of independent Boolean values $\mathbf{r}$ is selected where each value $r_i$ is $\top$ with probability $p_i$, and 4) the game state transitions to $\delta(\mathbf{y}, \mathbf{a_1}, \mathbf{a_2}, \mathbf{r})$. The payoff for $P_1$ is $\sum_{0 \leq i < m_u, u_i(\mathbf{y}) = \top} 2^i$ after $h$ state transitions, while the payoff for $P_2$ is its negation.*

We note that although this representation allows for two moves of different players and a random move (to select $\mathbf{r}$) before the state transitions, it preserves the perfect information property required by Definition 1, as no decisions are made simultaneously without observation.

Moreover, the state-transition function $\delta$ is flexible enough to enforce the turn order defined by $\pi$ as in Definition 1. We can introduce an additional input 'no-op' variable for both players, which indicates that the player does not perform any action when 'no-op' input is true while all other input variables are false. If an inactive player performs any action other than a mandatory 'no-op', the game state will transition to a losing state. Similarly, we distinguish between deterministic and random game states by controlling the influence of the random inputs $\mathbf{r}$. For deterministic game states $v$ with $\pi(v) \neq N$, we define $\delta$ to be independent of $\mathbf{r}$. For random game states $v$ with $\pi(v) = N$, we define $\delta$ to depend solely on $\mathbf{r}$. Consequently, the finite-state machine games can model any abstract game characterized by Definition 1, limited only by the approximation error in discretizing a real-valued payoff $u$ into a fixed-width binary integer $\mathbf{u}$.

Given a game $\Gamma$, we want to solve for its equilibrium value with an SSAT problem. Therefore, we need a formula to convert binary representations to satisfying probabilities in order to encode the payoff function as an SSAT formula. We propose an encoding by dividing the binary representation and weighting the probabilities adequately. Let $\chi_k$ be defined recursively by $\chi_0 = t_0$ and $\chi_{k+1} = (w_k \wedge \chi_k[t_{2^k}/t_0, \ldots, t_{2^{k+1}-1}/t_{2^k-1}]) \vee (\neg w_k \wedge \chi_k)$, with fresh indexed variables $t_i$ and $w_i$. Then, the variables used in $\chi_k$ are $\mathrm{Vars}(\chi_k) = \{t_0, t_1, \ldots, t_{2^k-1}, w_0, \ldots, w_{k-1}\}$.

**Lemma 3.** *For any assignment $\tau : \{t_0, t_1, \ldots, t_{2^k-1}\} \to \mathbb{B}$, we have*

$$\Pr[\bar{\mathrm{R}}^{2^{2^{k-1}}/(2^{2^{k-1}}+1)}\, w_{k-1} \ldots \bar{\mathrm{R}}^{2/3}\, w_0 . \chi_k[\tau]] = \sum_{0 \leq i < 2^k, \tau(t_i) = \top} 2^i/(2^{2^k} - 1).$$

This is exactly the integer represented by $\tau$, divided by $(2^{2^k} - 1)$. The detailed proof, by induction on $k$, can be found in the Appendix at [22].

We note that the size of the propositional formula $\chi_k$ is linear in $2^k$ as each $t_i$ appears exactly once in the formula, with each occurrence of $w_i$ combining two sub-formulas with disjoint variables. This compact conversion of binary representations to satisfying probabilities allows for the encoding of two-player games with a more diverse payoff than a binary-outcome game.

Given $\Gamma = (\mathbf{y}, c, \mathbf{a_1}, \mathbf{a_2}, \mathbf{r}, (p_1, \ldots, p_n), \delta, \mathbf{u}, h)$, we define its encoding as an SSAT formula:

$$
\begin{aligned}
\Phi_\Gamma = & \exists \mathbf{y}^{(0)} \exists \mathbf{a_1}^{(0)} \forall \mathbf{a_2}^{(0)} \, \mathbb{R}^{p_1} \, r_1^{(0)} \ldots \mathbb{R}^{p_n} \, r_n^{(0)} \\
& \exists \mathbf{y}^{(1)} \exists \mathbf{a_1}^{(1)} \ldots \mathbb{R}^{p_n} \, r_n^{(h-1)} \exists \mathbf{y}^{(h)} \\
& \exists t_0 \ldots \exists t_{2^k-1} \, \mathbb{R}^{2^{2^{k-1}}/(2^{2^{k-1}}+1)} \, w_{k-1} \ldots \mathbb{R}^{2/3} \, w_0. \\
& (\mathbf{y}^{(\mathbf{0})} \leftrightarrow c) \wedge \bigwedge_{i=0}^{h-1} (\mathbf{y}^{(i+1)} \leftrightarrow \delta(\mathbf{y}^{(i)}, \mathbf{a_1}^{(i)}, \mathbf{a_2}^{(i)}, \mathbf{r}^{(i)})) \\
& \wedge \bigwedge_{i=0}^{m_u-1} (t_i \leftrightarrow u_i(\mathbf{y}^{(h)})) \wedge \left( \bigwedge_{i=l}^{2^k-1} \neg t_i \right) \wedge \chi_k,
\end{aligned}
$$

where $k = \lceil \log_2 m_u \rceil$, and the notation $x^{(i)}$ stands for the value of $x$ at timestep $i$. A quantification of a list of variables serves as a shorthand for quantifying its individual components, whereas the equivalence between two lists of variables is defined as the conjunction of the equivalences of their corresponding components.

**Theorem 3.** *For any finite-state machine game $\Gamma$, its expected Nash equilibrium value is equal to $(2^{2^k} - 1) \cdot \Pr[\Phi_\Gamma]$.*

The detailed proof, by induction on the horizon $h$, can be found in the Appendix at [22].

With Theorem 3, finding the value of a finite-state machine game $\Gamma$ can be converted to the problem of solving an SSAT formula $\Phi_\Gamma$, with the number of variables $\in O(h(q + m_1 + m_2 + n) + m_u)$, and the matrix formula size $\in O(h(\|\delta\| + q) + \|u\| + m_u))$, where $\|\phi\|$ is the size of a propositional formula $\phi$. Therefore, for a horizon $h$ specified in unary, this reduction is polynomial.

## V. Computing Nash Equilibrium and Strategies of SSAT Games

### A. DPLL-based Solving

`SharpSSAT` [7] is a DPLL-based SSAT solver that incorporates multiple techniques from model counting and SAT solving, including component analysis with caching, clause learning, pure literal elimination, and early return at existential quantifiers.

We extend the techniques used in `SharpSSAT` to support efficient solving of general SSAT formulas with universal quantification, as shown in Algorithm 1, where the modifications are highlighted, including Boolean constraint propagation in Line 1, component analysis in Line 2, pure literal detection in Line 3, and the implementation of universal quantification in Lines 12 and 13. In addition, we perform universal clause detection as a prescreening check before the recursive procedure of Algorithm 1, and extend the prior strategy generation algorithm [7] substantially.

*1) Universal Clause Detection:* To enable the solver's capability in detecting trivial instances before applying Algorithm 1, we detect *universal clauses*, that is, clauses composed entirely of universally quantified literals. Such clauses can be trivially falsified by the universal player, regardless of the existential player's choices. If a universal clause exists, we terminate early and skip Algorithm 1. This observation is leveraged during strategy generation: when a universal clause is identified, we construct a Herbrand strategy that falsifies it.

**Algorithm 1** solveSSAT

**Input**: An SSAT formula $\Phi$, Component Cache $\mathcal{C}$
**Output**: $p_{ret} = \Pr[\Phi]$

1: *BCP*($\Phi$)
2: *componentAnalysis*($\Phi$)
3: *pureElimination*($\Phi$)
4: $p_{ret} \leftarrow 1$
5: **for** $\Phi_i \in$ *Components*($\Phi$) **do**
6: **if** $(p_i, \Phi_i) \in \mathcal{C}$ **then**
7: $p_{ret} \leftarrow p_i$
8: continue
9: $x \leftarrow$ *chooseBranchVar*($\Phi_i$)
10: $p_0 \leftarrow$ *solveSSAT*($\Phi[0/x], \mathcal{C}$)
11: $p_1 \leftarrow$ *solveSSAT*($\Phi[1/x], \mathcal{C}$)
12: **if** $x \in X_A$ **then**
13: $p \leftarrow \min(p_0, p_1)$
14: **else if** $x \in X_E$ **then**
15: $p \leftarrow \max(p_0, p_1)$
16: **else**
17: $p \leftarrow (1 - \Pr[x]) \cdot p_0 + \Pr[x] \cdot p_1$
18: **if** $p = 0$ **then**
19: *removePollutedCache*($\mathcal{C}, \Phi_i$)
20: *addLearnedClause*()
21: $p_{ret} \leftarrow 0$
22: break
23: $p_{ret} \leftarrow p_{ret} \cdot p$
24: $\mathcal{C} \leftarrow \mathcal{C} \cup (\Phi, p_{ret})$
25: **return** $p_{ret}$

*2) Boolean Constraint Propagation:* During *Boolean constraint propagation* (BCP) [23], a value of a variable at the outermost quantifier level is determined, and unit clause constraints are propagated. If a unit clause consisting of a universally quantified literal is encountered, we immediately return a satisfying probability of 0 (because the universal player can falsify this clause), allowing the solver to prune the current search branch early. This modification is necessary to prevent unsound implications, in contrast to the cases of unit existential or random variables, in which the literal is implied true and propagated to other constraints.

*3) Component Analysis: Component analysis* is a key technique in model counting used to avoid redundant computation by exploiting formula decomposability [24], [25]. Component analysis is also sound for SSAT formulas with existential and random quantifiers [4]. This result can be naturally extended to SSAT formulas that include universal quantifiers. Consequently, minimal modification to the original `SharpSSAT` data structures is required to support instances involving universal variables.

**Theorem 4** (Component Analysis)**.** *Given an SSAT formula $\Phi = \mathcal{Q}.\phi_1 \wedge \phi_2$, where $\mathcal{Q}$ may contain existential, universal, and random quantifiers, and $\mathrm{Vars}(\phi_1) \cap \mathrm{Vars}(\phi_2) = \emptyset$, we have*

$$\Pr[\Phi] = \Pr[\mathcal{Q}.\phi_1] \Pr[\mathcal{Q}.\phi_2].$$

This can be proven by induction on the number of variables,

as detailed in the Appendix at [22].

*4) Pure Literal Detection:* A literal $l$ is *pure* in a CNF formula $\phi$ if $\neg l$ does not appear in $\phi$. The appearance of a pure literal $l$ in $\phi$ indicates that the underlying Boolean function is unate in $\mathrm{var}(l)$, i.e., the implication $\phi[\neg l] \rightarrow \phi[l]$ is valid. In this case, we can safely assert that $l$ should be set to $\top$ (resp. $\bot$) if $\mathrm{var}(l) \in X_E$ (resp. $X_A$), even if $\mathrm{var}(l)$ is not in the outermost quantifier level [26].

We enhance the detection of pure existential literals during component analysis by considering the effect of pure literal assignments to infer more pure literals in the process. This enhancement allows making fewer decisions during solving, with little overhead compared to ignoring the effects of clauses satisfied by pure literals. We note that the detection of pure universal literals is not currently implemented in `SharpSSAT` due to its complications in the BCP and conflict learning process, and we leave it for future work.

**Remark.** *The original `SharpSSAT` performs an early return and skips computing the values of the remaining sub-components at a branch $x = \top$, for existential $x$, if its satisfying probability is guaranteed to be lower than the other branch $x = \bot$ [7]. Performing early return on existential variables is sound for SSAT formulas that contain existential, universal, and random quantifiers. However, performing early returns on random and universal variables is unsound. A counterexample is provided in the Appendix at [22].*

**Theorem 5.** *Given a general SSAT formula $\Phi$ that involves existential, universal, and random quantifiers, the extended `SharpSSAT` algorithm computes* $\Pr[\Phi]$.

The detailed proof can be found in the Appendix at [22].

*5) Trace Representation:* The SSAT solving steps, called the *trace* [7], are recorded as a directed acyclic graph of decision nodes with a non-decision root node representing the start of solving.

**Definition 3.** *Let $\Phi = \mathcal{Q}.\phi$ be an SSAT formula . A trace $T$ is a directed acyclic graph $T = (V, E)$ with a root node $v_0 \in V$ representing the initial formula $\Phi$. Each internal decision node $v \in V \setminus \{v_0\}$ represents a component encountered during the solving process, labeled with its respective branching variable $v.x$. A directed edge $e = (v, w) \in E$ is labeled with a Boolean assignment $b \in \{\top, \bot\}$ to $v.x$, connecting to a child node $w$ that represents one of the independent sub-component formulas derived via component analysis. Edge labels along $e$ store the set of implied existential or universal literals $L$ discovered via BCP or pure literal elimination for the specific component's formula and decision variable assignment.*

The independent components decomposed by component analysis produced from a decision are all attached to the same decision branch of its parent node. For nodes with existential or universal decision variables, the decision that leads to the optimal result for the player is recorded as winBranch, which is the result that leads to the maximum (resp. minimum) probability for the existential (resp. universal) player. At each decision, all the unit clauses with existential or universal variables will be recorded separately during BCP, since the existential (resp. universal) player must make the choice to satisfy all the unit clauses (resp. falsify one of the unit clauses) in order to maximize (resp. minimize) the satisfying probability of this component. Similarly, when a pure literal with an existential variable is discovered, the corresponding choice to maximize the satisfying probability is also chosen by the existential player. We say that these choices made by the existential (resp. universal) player are the existential (resp. universal) literals implied by the branch decision, which are recorded for each decision branch. The implied literals obtained before the recursive SSAT solving phase, such as those from universal clause detection, will be attached to the root node, in which the precondition is set as $\top$.

**Algorithm 2** extractHerbrandStrategyFromTrace

**Input**: A trace $T$
**Output**: $\mathbf{s}_\forall = \{s_x \mid x \in X_A\}$

**for** each decision node $v \in T$ **do**
  $v.\tau \leftarrow \emptyset$  // **Preconditions for** $v$
**for** $x \in X_A$ **do**
  $s_x \leftarrow \bot$
$L \leftarrow$ universal literals implied by preprocessing
$\mathbf{s}_\forall \leftarrow$ *updateHerbrandFunc*$(\mathbf{s}_\forall, L, \top)$
**for** $c \in T.root$.children **do**
  $c.\tau \leftarrow \{\top\}$
**for** each decision node $v \in T$ in topological order **do**
  $\lambda \leftarrow \bigvee_{\lambda' \in v.\tau} \lambda'$
  **if** $v.x \in X_A$ **then**
    **if** $v$.winBranch $= \top$ **then**
      **for** $c \in v$.posBranch **do**
        $c.\tau \leftarrow c.\tau \cup \lambda$
      $l_{v.x} \leftarrow v.x$
    **else**
      **for** $c \in v$.negBranch **do**
        $c.\tau \leftarrow c.\tau \cup \lambda$
      $l_{v.x} \leftarrow \neg v.x$
    $L \leftarrow$ universal literals implied by $l_{v.x}$
    $\mathbf{s}_\forall \leftarrow$ *updateHerbrandFunc*$(\mathbf{s}_\forall, L \cup \{l_{v.x}\}, \lambda)$
  **else**
    $\lambda' \leftarrow \lambda \wedge v.x$
    $L \leftarrow$ universal literals implied by $v.x$
    $\mathbf{s}_\forall \leftarrow$ *updateHerbrandFunc*$(\mathbf{s}_\forall, L, \lambda')$
    **for** $c \in v$.posBranch **do**
      $c.\tau \leftarrow c.\tau \cup \lambda'$
    $\lambda' \leftarrow \lambda \wedge \neg v.x$
    $L \leftarrow$ universal literals implied by $\neg v.x$
    $\mathbf{s}_\forall \leftarrow$ *updateHerbrandFunc*$(\mathbf{s}_\forall, L, \lambda')$
    **for** $c \in v$.negBranch **do**
      $c.\tau \leftarrow c.\tau \cup \lambda'$
**return** $\mathbf{s}_\forall = \{s_x \mid x \in X_A\}$

**Algorithm 3** updateHerbrandFunc

**Input**: Herbrand strategy $\mathbf{s}_\forall$, literal set $L$, precondition $\lambda$
**Output**: Updated Herbrand strategy $\mathbf{s}_\forall$

**for** $l \in L$ **do**
  $x \leftarrow \mathrm{var}(l)$  // **assert** $s_x \in \mathbf{s}_\forall$
  **if** $l$ is positive **then**
    $s_x \leftarrow s_x \vee \lambda$
**return** $\mathbf{s}_\forall$

*6) Skolem and Herbrand Strategy Generation:* When the solving phase is completed, we extract the Skolem and Herbrand strategy profile from the trace with respect to their respective variables' decisions in each node. For the original `SharpSSAT` Skolem function generation [7], one only needs to generate a set of Skolem functions for existential variables with respect to the random variable in its dependency set, traversing the trace exactly once. When the formula contains universal variables, we must generate both Skolem functions $\mathbf{s}_\exists$ and Herbrand functions $\mathbf{s}_\forall$ with respect to their own dependency sets, each traverses the trace once separately. Algorithms 2 and 3 describe the process for Herbrand strategy generation.

Algorithm 2 traverses the trace in a topological order (Line 9). When each node is reached, the algorithm first calculates in Line 10 its precondition, i.e., the disjunction of all the path condition formulas that reach this node. The formula set $v.\tau$ is used for each node $v$ to record the preconditions, which, together with the branching decision, forms the precondition $\lambda$ in which the strategy should make a decision of $\{\bot, \top\}$ on a certain variable $x$, such as the minBranch decision on universal variables in Line 12, and the implied universal literals induced from BCP and pure literal detection in Lines 13, 18, 24, and 29. Since the corresponding strategy function $s_x$ is initialized as $\bot$, the function will only be updated if the decision made for $x$ should be $\top$ under the precondition $\lambda$ (Algorithm 3).

The Skolem strategy generation algorithm is largely similar to Algorithm 2, except that the universal variable set $X_A$ in Lines 3 and 11 of Algorithm 2 is replaced with the existential variable set $X_E$ and $L$ should be assigned with existential literals instead of universal literals in Lines 5, 20, 24, and 29 to generate the Skolem strategy for each existential variable.

**Theorem 6.** *Given a general SSAT formula $\Phi$ that involves existential, universal, and random quantifiers, the strategy profile generated from Algorithm 2 and its Skolem strategy counterpart is a Nash equilibrium on the SSAT game of $\Phi$.*

*Proof.* We first prove that $\Pr[\Phi[\mathbf{s}_\exists]] = \Pr[\Phi[\mathbf{s}_\forall]] = \Pr[\Phi]$ by induction on the structure of the trace. With this fact, we further show that the strategy profile $(\mathbf{s}_\exists, \mathbf{s}_\forall)$ is a Nash equilibrium. The detailed proof can be found in the Appendix at [22]. □

### B. BDD-based Solving

Because there are no other SSAT solvers that support universal quantification, to serve as a baseline for comparison with our DPLL-based solving, we present a BDD-based alternative. Specifically, we represent the matrix $\phi$ of the SSAT formula $\mathcal{Q}.\phi$ with a Reduced Ordered Binary Decision Diagram (ROBDD) [27], where the BDD variables are ordered according to the prefix $\mathcal{Q}$, such that the variables appearing earlier in the prefix are closer to the root of the BDD.

To support efficient BDD construction, we utilize dynamic reordering. Variables of an SSAT formula are called of the same *quantification level* if they are adjacent in the prefix and have the same quantifier type. Variables of the same level can be permuted in the prefix without affecting the satisfying probability. This freedom allows us to apply *group sifting* [28] to optimize the variable order while maintaining the levelized structure required by different quantification blocks.

**Algorithm 4** computeSatisfyingProbabilityWithBDD

**Input**: An SSAT formula $\Phi$
**Output**: $\Pr[\Phi]$

```
1: BDD.groups ← Φ.quantification levels
2: g ← BDD.One
3: for each clause c ∈ Φ.matrix do
4:     g ← g ∧ c
5: BDD.One.pr ← 1
6: BDD.Zero.pr ← 0
7: for each non-leaf node v traversing g bottom-up do
8:     if v.x ∈ X_E then
9:         v.pr ← max(v.then.pr, v.else.pr)
10:    else if v.x ∈ X_A then
11:        v.pr ← min(v.then.pr, v.else.pr)
12:    else
13:        p ← probability associated with v.x
14:        v.pr ← p · v.then.pr + (1 − p) · v.else.pr
15: return g.pr
```

Once the BDD is constructed, the satisfying probability $\Pr[\Phi]$ is computed with a bottom-up traversal as outlined in Algorithm 4. For each node $v$ representing a Boolean function $\psi_v$, we compute $\Pr[\mathcal{Q}.\psi_v]$ based on the quantifier of the variable associated with $v$.

To perform strategy generation for BDD-based solving, we can simply record the optimal branch decision made at each node with an existential or universal variable in Algorithm 4, and treat the entire BDD structure as a trace to perform strategy generation with the same process as `SharpSSAT` (Section V-A6), but with empty implied literals for each decision branch.

## VI. Experiments

### A. Benchmarks

Our total benchmark suite consists of 696 SSAT instances across 25 benchmark families. The statistics of all benchmark families can be found in the Appendix at [22]. We adapted 20 of these families from the benchmark set used in `ClauSSat` [5] by randomly replacing existential and random quantifiers with universal quantifiers for each instance. Detailed descriptions of these benchmarks can be found in [5]. Although these families do not represent the actual encoding of two-player games, they are included to provide diversity for testing. In addition, to evaluate the performance of `SharpSSAT` in two-player zero-sum games, we created five new benchmark families based on well-known two-player games: `breakthrough-rand`, `candy-nim-rand`, `dots-and-boxes-rand`, `nim-rand`, and `pig`. As defined in Definition 2, all encoded games have payoffs represented by fixed-width binary integers and enforce a finite horizon for termination. The transition and utility functions of the games are initially specified in Verilog HDL, which is then further synthesized into and-inverter graphs (AIG) using the

synthesis tool Yosys [29]. After that, we convert the games to SSAT formulas and apply Tseitin transformation [30] to rewrite the matrix in CNF.

The benchmark `breakthrough-rand` is a randomized variant of Breakthrough, a two-player board game invented by Dan Troyka in 2000 [31]. It is played on a rectangular board with each player's pawns filling their respective two home rows. Players take turns to move one of their pawns one square straight or diagonally forward. Diagonal moves into an opponent's occupied square result in a guaranteed capture, while straight forward captures succeed only 50% of the time. If the forward capture fails, both pawns remain in their original positions. A player wins by either moving a pawn to the opposite edge of the board or capturing all of their opponent's pawns. The payoff for $P_1$ is 2 if $P_1$ wins, 0 if $P_2$ wins, and 1 in any state where neither player has yet won.

The `nim-rand` benchmark encodes a randomized variant of the classic two-player game Nim. There are $n$ piles of stones with $c$ stones each. On each turn, a 6-sided die is rolled to decide the number of stones they must remove from a single selected pile. If the pile contains fewer stones than the die roll, the player takes all the stones from it. A player wins by taking the final stone from the board. The payoff for $P_1$ is 2 if $P_1$ wins, 0 if $P_2$ wins, and 1 in any state where neither player has yet won.

The `candy-nim-rand` benchmark modifies only the payoff structure of `nim-rand` similar to Candy Nim [32], [33]. The players aim to maximize their accumulated stones, but the primary objective is to capture the final stone, which awards a bonus payoff equivalent to $l$ stones. Thus, the payoff for $P_1$ is the number of stones taken by $P_1$, plus the bonus $l$ if they take the final stone from the board.

The benchmark `dots-and-boxes-rand` encodes a two-player game in which, unlike the previous games, the moving player cannot be inferred directly from the depth of the game state. In `dots-and-boxes-rand`, two players compete on a rectangular grid of dots. Each player takes turn to draw a horizontal or vertical line between two neighboring dots. When the line drawn by a player forms a $1 \times 1$ box, the player gains one point and gets to draw an additional line with 50% probability. The payoff for $P_1$ is the points gained by $P_1$.

The benchmark `pig` encodes the two-player dice game Pig [34]. On each turn, the active player rolls a die and chooses to either *hold* or *roll*. If the player chooses to *hold*, they score the sum of their rolls for that round and pass the turn to the opponent; choosing to *roll* attempts to accumulate additional points. The turn ends immediately if the active player rolls a 1, in which case they score zero points for that round. The first player to reach a target score wins. The payoff for $P_1$ is 2 if $P_1$ wins, 0 if $P_2$ wins, and 1 in any state where neither player has yet won.

### B. Evaluation

We evaluated the performance of our extensions to `SharpSSAT`, with optional pure literal detection. Because there are no SSAT solvers that support universal quantifiers, we implemented a BDD-based baseline solver in C++ using ROBDD compilation via the CUDD package [35] for comparison (Section V-B). All experiments were conducted on a single CPU core of an Intel Xeon w7-2495X with 512 GB of RAM running on Ubuntu 22.04. Each instance was run once, given a time limit of 1000 seconds, and the component cache of `SharpSSAT` was restricted to 64 GB.

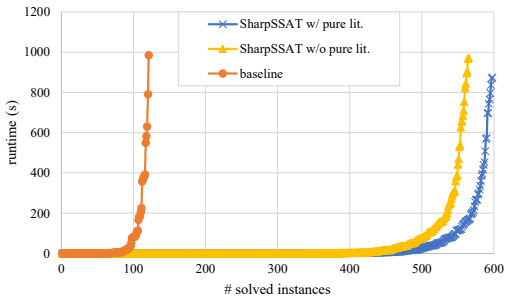



Fig. 1. Number of solved instances vs. runtime.

Table I compares `SharpSSAT` with and without pure literal detection against the BDD baseline, without generating strategies, where the number of instances per benchmark family (#I), the number of solved instances (#S), Penalized Average Runtime (PAR2) in seconds, and average memory usage for solved instances (MEM) in MB are reported. Overall, `SharpSSAT` solved more instances within the runtime limit compared to the baseline solver for all benchmark families, demonstrating the effectiveness of DPLL search with component caching over a variety of SSAT formulas. We note that for the instances generated by our proposed reduction, enabling pure literal detection does not significantly affect the solver's runtime. This is because the Tseitin variables for the state-transition functions are combined with random variables in $\chi_k$ to form the final matrix. As a result, most variables appear in both phases in the matrix, and pure literal elimination does not take effect.

Note that because `SharpSSAT` was able to solve more large instances within the runtime limit, this contributed to `SharpSSAT`'s larger average memory usage compared to the BDD-based solver.

Figure 1 depicts the sorted runtime of solving the complete set of instances under three solver settings. Among the compared settings, `SharpSSAT` with pure literal elimination outperforms the others. The result demonstrates the effectiveness of pure literal elimination.

To evaluate the overhead of strategy generation, we conducted experiments to compare `SharpSSAT` with and without enabling strategy generation. `SharpSSAT` successfully generated the strategies for all instances that were solved without enabling strategy generation. As shown in Fig. 2a, the runtime overhead of strategy generation is minor, averaging a 16% increase across all instances. This outcome highlights the efficiency of our witness extraction algorithm. Although strategy generation requires maintaining the trace throughout

TABLE I
EXPERIMENTAL RESULTS COMPARING SHARPSSAT AND THE BASELINE SOLVER.

| | | SharpSSAT | | | | | | baseline | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | w/o pure lit. | | | w/ pure lit. | | | | | |
| Family | #I | #S | PAR2 | MEM | #S | PAR2 | MEM | #S | PAR2 | MEM |
| tlc | 13 | **13** | 0.00295 | 6 | **13** | **0.00294** | 6 | 0 | 2,000.00 | N/A |
| gttt_3x3 | 9 | **9** | **0.0754** | 15 | **9** | 0.0769 | 13 | 0 | 2,000.00 | N/A |
| ev-pr-4x4 | 7 | **7** | **0.00406** | 6 | **7** | 0.00446 | 5 | 0 | 2,000.00 | N/A |
| arbiter | 10 | 5 | 1,100.09 | 1,091 | **7** | **605.40** | 24 | 0 | 2,000.00 | N/A |
| Tree | 14 | **14** | 0.00396 | 10 | **14** | **0.00387** | 10 | 12 | 288.48 | 38 |
| RobotsD2 | 10 | **4** | 1,231.47 | 5,364 | **4** | **1,202.49** | 347 | 0 | 2,000.00 | N/A |
| depots | 9 | **3** | **1,333.34** | 5 | **3** | **1,333.34** | 5 | 0 | 2,000.00 | N/A |
| pipesnotankage | 9 | **1** | **1,777.78** | 5 | **1** | **1,777.78** | 5 | 0 | 2,000.00 | N/A |
| Counter | 8 | **6** | 538.40 | 4,127 | **6** | **537.29** | 20 | 4 | 1,024.91 | 295 |
| Connect2 | 16 | **16** | 5.17 | 71 | **16** | **0.0347** | 68 | 1 | 1,880.13 | 1,080 |
| Adder | 6 | **5** | **333.33** | 4 | **5** | **333.33** | 4 | 2 | 1,333.39 | 28 |
| k_branch_n | 10 | **3** | **1,400.00** | 9 | **3** | **1,400.00** | 9 | 0 | 2,000.00 | N/A |
| conformant | 24 | **15** | **750.01** | 7 | **15** | **750.01** | 7 | 0 | 2,000.00 | N/A |
| tiger | 5 | **5** | 0.00143 | 4 | **5** | **0.000442** | 4 | 2 | 1,216.79 | 86 |
| ToiletA | 77 | **77** | **9.41** | 197 | **77** | 9.53 | 131 | 18 | 1,543.65 | 96 |
| sand-castle | 25 | **24** | 144.06 | 2,733 | **24** | **141.41** | 1,497 | 15 | 809.56 | 68 |
| MaxCount | 25 | **10** | **1,201.25** | 146 | **10** | 1,201.31 | 112 | 2 | 1,879.39 | 143 |
| MPEC | 8 | **8** | **1.99** | 53 | **8** | 2.17 | 7 | 0 | 2,000.00 | N/A |
| PEC | 8 | **5** | **750.02** | 9 | **5** | **750.02** | 7 | 0 | 2,000.00 | N/A |
| stracomp | 60 | 15 | 1,522.81 | 5,254 | **48** | **416.40** | 221 | 6 | 1,819.76 | 80 |
| breakthrough-rand | 60 | **43** | **648.05** | 1,090 | **43** | 652.72 | 1,089 | 0 | 2,000.00 | N/A |
| candy-nim-rand | 79 | **79** | **25.39** | 604 | **79** | 27.79 | 604 | 18 | 1,551.80 | 58 |
| dots-and-boxes-rand | 45 | **42** | **190.41** | 96 | 41 | 215.16 | 95 | 5 | 1,811.84 | 68 |
| nim-rand | 79 | **73** | **166.16** | 78 | **73** | 167.30 | 78 | 20 | 1,507.91 | 57 |
| pig | 80 | **80** | **85.23** | 1,983 | 79 | 103.87 | 1,712 | 15 | 1,633.53 | 80 |
| total | 696 | 562 | 417.89 | 850 | **595** | **319.32** | 505 | 120 | 1,666.10 | 85 |

the solving process, resulting in increased memory usage, the peak memory consumption remains roughly twice that of solving without strategy generation in most cases, which can be seen in Fig. 2b.

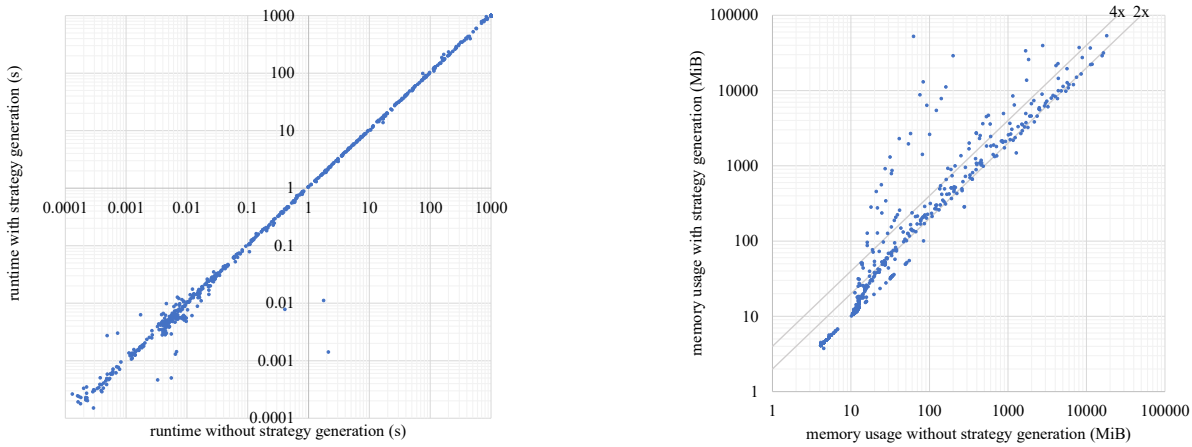
Furthermore, we studied the effects of pure literal detection on the quality of generated strategies. We compared the size of the synthesized circuit (in AIG nodes) for each strategy with and without pure literal elimination in Fig. 3. Strategy circuits were synthesized using `ABC` [36] and optimized with the `dc2` command. We excluded trivial strategies with zero AIG nodes from the comparison. In Fig. 3, each data point plots the circuit size generated with pure literal detection (vertical axis) against the size generated without it (horizontal axis). Most of the strategy functions are roughly the same size with and without pure literal elimination. Almost 80% of the cases have strategy sizes within a 5% deviation, while some extreme instances may see up to 99% reduction when allowing pure literal elimination. This improvement is expected, as identifying pure literals enables pruning of the search space and eliminates unnecessary dependencies for variables at inner quantifier levels. The variation can be explained by the way the SSAT instances are generated. As mentioned in the analysis of Table I, the specific formula structures of the game-based instances limit the utility of pure literal detection. Consequently, their generated strategies will only have limited size improvements, since the strategy size of an instance is linear in the size of the trace created from the solving process. In contrast, some benchmarks lack this structural property, enabling significant reductions of strategy sizes.

Our generated strategy profiles $(\mathbf{s}_\exists, \mathbf{s}_\forall)$ for SSAT formulas $\Phi$ were verified by first forming $\Phi[\mathbf{s}_\exists]$ and $\Phi[\mathbf{s}_\forall]$, defined in Section II, with a procedure similar to the AIG method described in Section VI-A. Then `SharpSSAT` was applied again to confirm that $\Pr[\Phi[\mathbf{s}_\exists]] = \Pr[\Phi[\mathbf{s}_\forall]] = \Pr[\Phi]$, which shows that $(\mathbf{s}_\exists, \mathbf{s}_\forall)$ is an equilibrium by Theorem 6.

To evaluate the efficiency of our encoding of finite two-player zero-sum games with perfect information, we compared `SharpSSAT` against `OpenSpiel` [37], an established framework for game theory research, across five stochastic game families. We implemented the benchmark families in C++ to leverage `OpenSpiel`'s exact Nash equilibrium algorithm `expectiminimax` for stochastic games [38]. The timeout limit is set at 10,000 seconds for both solvers.

As summarized in Table II, our approach outperforms `OpenSpiel` overall, solving more instances faster and reducing the PAR2 score by over two-thirds. However, `OpenSpiel` manages to solve more instances in `breakthrough-rand` and `dots-and-boxes-rand` than `SharpSSAT`. We attribute this to the relatively limited role of stochastic elements for these games.

The performance of `SharpSSAT` and `OpenSpiel` as exact Nash equilibrium solvers is depicted in Fig. 4. For smaller game instances, `OpenSpiel` generally evaluates faster than `SharpSSAT`. In the initial scaling phases across the plots,




(a) Runtime comparison.

(b) Memory usage comparison.

Fig. 2. Runtime and memory usage comparison of `SharpSSAT` with and without strategy generation.

TABLE II
EXPERIMENTAL RESULTS OF EXACT NASH EQUILIBRIUM COMPUTATION COMPARING `SharpSSAT` AND `OpenSpiel`.

| | | `SharpSSAT` | | | `OpenSpiel` | | |
|---|---|---|---|---|---|---|---|
| Family | #I | #S | PAR2 | MEM | #S | PAR2 | MEM |
| breakthrough-rand | 60 | 49 | 4075.71 | 2,222 | **60** | **365.68** | 9 |
| candy-nim-rand | 79 | **79** | **27.79** | 604 | 69 | 2,753.82 | 9 |
| dots-and-boxes-rand | 45 | 43 | 973.51 | 99 | **44** | **700.70** | 9 |
| nim-rand | 79 | **77** | **740.32** | 128 | 68 | 2,887.42 | 9 |
| pig | 80 | **80** | **92.00** | 1,983 | 52 | 7,340.18 | 9 |
| total | 343 | **328** | **1039.04** | 1,004 | 293 | 3,167.19 | 9 |

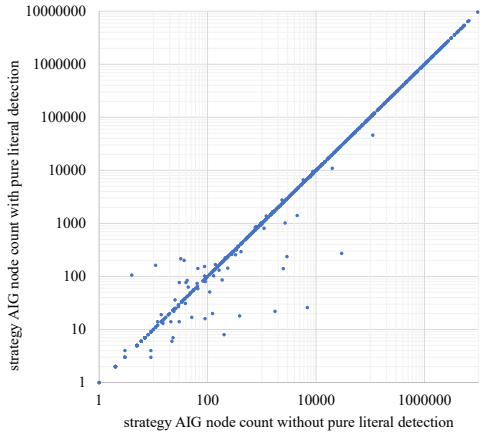



Fig. 3. Strategy size comparison with and without pure literal elimination.

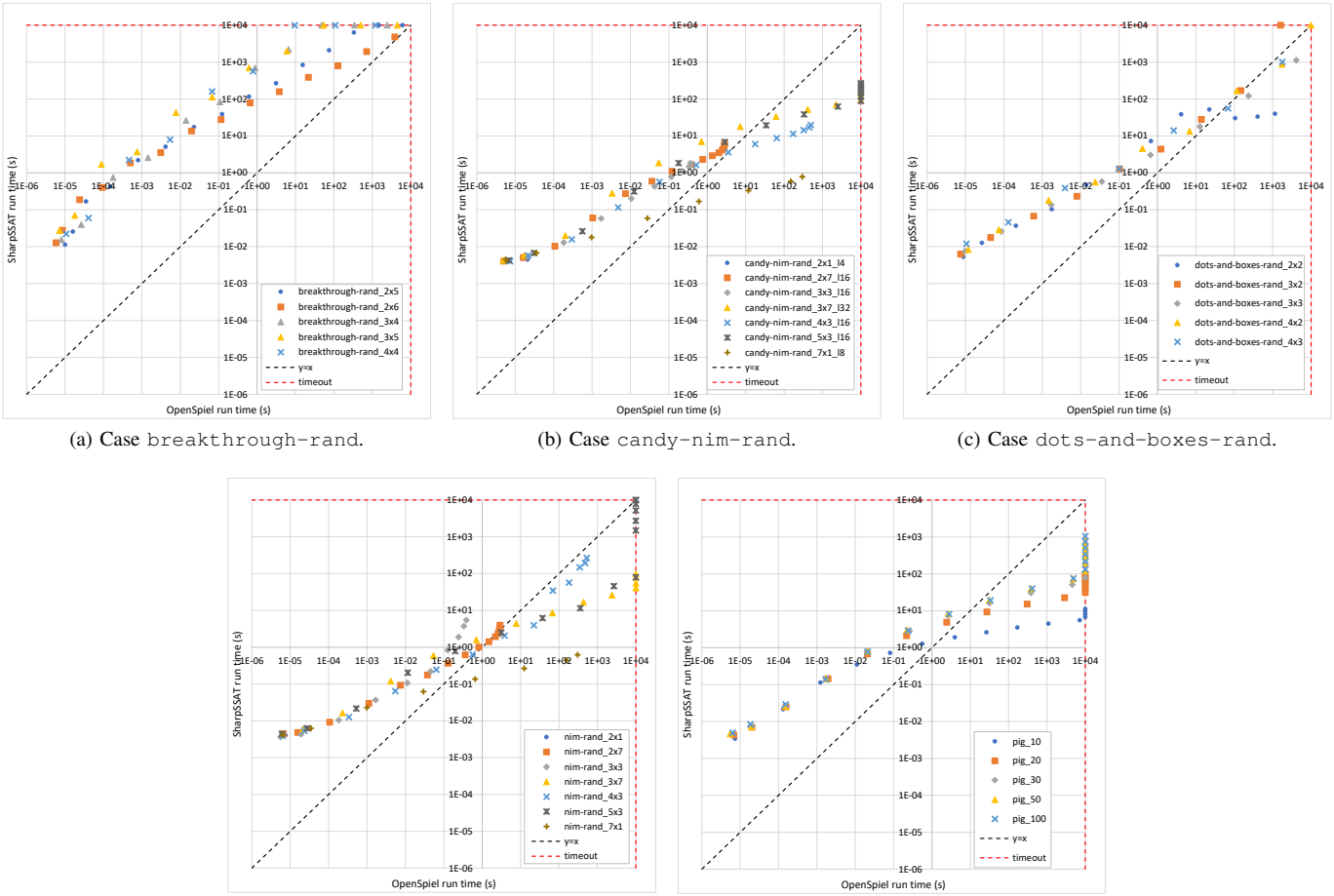
data points frequently appear above the $y = x$ reference line. This is expected, since `OpenSpiel` computes state transitions and payoffs directly without the upfront processing overhead associated with CNF encoding.

As the problem size and the horizon of the game increase, `SharpSSAT` demonstrates a substantial scalability advantage in `candy-nim-rand`, `nim-rand`, and `pig`. In these three families, there are clear crossover points in the plots where `SharpSSAT` overtakes `OpenSpiel`, demonstrated by the main clustering of data points shifting below the $y = x$ line as execution times grow. Most notably, `OpenSpiel` hits the 10,000 seconds timeout limit on numerous hard instances in these families, while `SharpSSAT` successfully solved these identical game instances, frequently in under 100 seconds.

The `breakthrough-rand` game is a notable exception to this trend. In this benchmark family, `OpenSpiel` maintains a performance lead across nearly all tested instances. `SharpSSAT` experiences timeouts on several instances that `OpenSpiel` solves within the time limit. These results reinforce our earlier observation that explicit state search algorithms like `expectiminimax` remains competitive for games with less stochastic influence.

## VII. CONCLUSIONS

We extended `SharpSSAT` as the first SSAT solver with strategy generation that handles formulas involving existential, universal, and random quantifiers. We also proposed a polynomial reduction from an implicitly specified finite two-player zero-sum stochastic game with perfect information to SSAT. This generic transformation enables the computation of Nash equilibrium and the corresponding strategy profile for two-player games with the extended SSAT solver. Experimental results demonstrate the practicality of `SharpSSAT` in solving




(a) Case `breakthrough-rand`. (b) Case `candy-nim-rand`. (c) Case `dots-and-boxes-rand`.

(d) Case `nim-rand`. (e) Case `pig`.

Fig. 4. Runtime comparison between `SharpSSAT` and `OpenSpiel` across five stochastic game families.

general SSAT formulas and Nash equilibrium computation for games with strong stochastic influence.

For future work, we plan to integrate more enhancement techniques, such as pure universal literal elimination, universal reduction [39], and dependency schemes [40] into SSAT solvers and strategy generation. In addition, we plan to create more game benchmarks and provide correctness proofs by extending `CPOG` [41], [42]. For real-world applications beyond game equilibrium calculation, we plan to investigate SSAT encoding for other multi-agent systems such as the design of secure circuits resilient to adversarial attack.

## Acknowledgments

This work was supported in part by the National Science and Technology Council of Taiwan under grant 114-2221-E-002-183-MY3, and the NTU Center of Data Intelligence: Technologies, Applications, and Systems.

## Data Availability

The artifact(s) associated with this paper is/are available at Zenodo under the following DOI: 10.5281/zenodo.20134207.

## References

[1] C. H. Papadimitriou, "Games against nature," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 288–301, 1985.

[2] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, vol. 336.

[3] S. M. Majercik and B. Boots, "DC-SSAT: a divide-and-conquer approach to solving stochastic satisfiability problems efficiently," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 20. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 416.

[4] R. Salmon and P. Poupart, "On the relationship between satisfiability and Markov decision processes," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 1105–1115.

[5] P.-W. Chen, Y.-C. Huang, and J.-H. R. Jiang, "A sharp leap from quantified Boolean formula to stochastic Boolean satisfiability solving," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 3697–3706.

[6] H.-R. Wang, K.-H. Tu, J.-H. R. Jiang, and C. Scholl, "Quantifier elimination in stochastic Boolean satisfiability," in *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022, pp. 23–1.

[7] Y.-W. Fan and J.-H. R. Jiang, "SharpSSAT: A witness-generating stochastic Boolean satisfiability solver," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 3949–3958.

[8] X. Chen and X. Deng, "Settling the complexity of two-player Nash equilibrium." in *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006*, vol. 6, 2006, pp. 261–272.

[9] P. Harrenstein, W. van der Hoek, J.-J. Meyer, and C. Witteveen, "Boolean games," in *Proceedings of the 8th Conference on Theoretical Aspects of Rationality and Knowledge*, 2001, pp. 287–298.

[10] P. E. Dunne and W. van der Hoek, "Representation and complexity in Boolean games," in *European Workshop on Logics in Artificial Intelligence*. Springer, 2004, pp. 347–359.

[11] E. Bonzon, M.-C. Lagasquie-Schiex, J. Lang, and B. Zanuttini, "Boolean games revisited," in *European Conference on Artificial Intelligence*, vol. 141, 2006, pp. 265–269.

[12] J. Feigenbaum, D. Koller, and P. Shor, "A game-theoretic classification of interactive complexity classes," in *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*. IEEE, 1995, pp. 227–237.

[13] L. Fortnow, R. Impagliazzo, V. Kabanets, and C. Umans, "On the complexity of succinct zero-sum games," *Computational Complexity*, vol. 17, no. 3, pp. 353–376, 2008.

[14] Y. He, A. Saffidine, and M. Thielscher, "Solving two-player games with QBF solvers in general game playing," in *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 2024, pp. 807–815.

[15] M. Thurley, "sharpSAT–counting models with advanced component caching and implicit BCP," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2006, pp. 424–429.

[16] V. Balabanov and J.-H. R. Jiang, "Unified QBF certification and its applications," *Formal Methods in System Design*, vol. 41, no. 1, pp. 45–65, 2012.

[17] D. Fudenberg and J. Tirole, *Game theory*. MIT press, 1991.

[18] P. Duersch, J. Oechssler, and B. C. Schipper, "Pure strategy equilibria in symmetric two-player zero-sum games," *International Journal of Game Theory*, vol. 41, no. 3, pp. 553–564, 2012.

[19] H. W. Kuhn, "Extensive games and the problem of information," *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 193–216, 1953.

[20] R. D. Luce and H. Raiffa, *Games and decisions: Introduction and critical survey*. Courier Corporation, 2012.

[21] J. v. Neumann, "Zur theorie der gesellschaftsspiele," *Mathematische annalen*, vol. 100, no. 1, pp. 295–320, 1928.

[22] C.-K. Ng, C.-H. Chuang, and J.-H. R. Jiang, "Stochastic Boolean satisfiability for two-player sequential games under uncertainty," 2026. [Online]. Available: https://doi.org/10.5281/zenodo.20134207

[23] J. M. Silva and K. A. Sakallah, "GRASP-a new search algorithm for satisfiability," in *Proceedings of International Conference on Computer Aided Design*. IEEE, 1996, pp. 220–227.

[24] F. Bacchus, S. Dalmao, and T. Pitassi, "DPLL with caching: A new algorithm for #SAT and Bayesian inference," in *44th Annual Symposium on Foundations of Computer Science*. IEEE, 2003.

[25] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi, "Combining component caching and clause learning for effective model counting," in *International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[26] M. L. Littman, S. M. Majercik, and T. Pitassi, "Stochastic Boolean satisfiability," *Journal of Automated Reasoning*, vol. 27, no. 3, pp. 251–296, 2001.

[27] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[28] S. Panda and F. Somenzi, "Who are the variables in your neighbourhood," in *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*. IEEE, 1995, pp. 74–77.

[29] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free Verilog synthesis suite," in *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, vol. 97, 2013, pp. 1–6.

[30] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*. Springer, 1983, pp. 466–483.

[31] K. Handscomb, "8×8 game design competition: The winning game: Breakthrough... and two other favorites," *Abstract Games Magazine*, vol. 7, pp. 8–9, 2001.

[32] M. Albert, "Candy nim," *The CMS Summer Meeting*, 2004. [Online]. Available: https://www.cs.otago.ac.nz/research/theory/Talks/CandyNim.pdf

[33] M. Albert, R. Nowakowski, and D. Wolfe, *Lessons in play: an introduction to combinatorial game theory*. AK Peters/CRC Press, 2019.

[34] T. W. Neller and C. G. Presser, "Optimal play of the dice game Pig," *The UMAP Journal*, vol. 25, no. 1, pp. 25–47, 2004.

[35] F. Somenzi, "CUDD: CU decision diagram package, 3.0," University of Colorado at Boulder, Tech. Rep., 2015.

[36] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.

[37] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. D. Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka, and J. Ryan-Davis, "OpenSpiel: A framework for reinforcement learning in games," *CoRR*, vol. abs/1908.09453, 2019. [Online]. Available: http://arxiv.org/abs/1908.09453

[38] D. Michie, "Game-playing and game-learning automata," in *Advances in Programming and Non-Numerical Computation*. Elsevier, 1966, pp. 183–200.

[39] H. K. Buning, M. Karpinski, and A. Flogel, "Resolution for quantified Boolean formulas," *Information and Computation*, vol. 117, no. 1, pp. 12–18, 1995.

[40] M. Samer and S. Szeider, "Backdoor sets of quantified Boolean formulas," *Journal of Automated Reasoning*, vol. 42, no. 1, pp. 77–97, 2009.

[41] R. E. Bryant, W. Nawrocki, J. Avigad, and M. J. Heule, "Certified knowledge compilation with application to verified model counting," in *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2023, pp. 6–1.

[42] C. Cheng, Y.-R. Luo, and J.-H. R. Jiang, "Knowledge compilation for incremental and checkable stochastic Boolean satisfiability." in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 2024, pp. 1862–1872.