Computing Game Strategies

Darko Stefanovic[1,2] and Milan N. Stojanovic[3,4]

[1] Department of Computer Science, University of New Mexico
[2] Center for Biomedical Engineering, University of New Mexico
Albuquerque, NM 87131, United States
[3] Division of Experimental Therapeutics, Department of Medicine, Columbia University
[4] Department of Biomedical Engineering, Columbia University
New York, NY 10032, United States

Abstract. We revisit the problem of constructing strategies for simple position games. We derive a general, executable formalism for describing game rules and desired strategy properties. We present the outcomes for several variants of the familiar game of tic-tac-toe.

Keywords: DNA computing, games of strategy, constraint satisfaction

1 Introduction

Games of strategy, in particular simple board games, have often been used to demonstrate the capabilities of new computing devices. A famous example was Donald Michie's tic-tac-toe (noughts and crosses) with matchboxes [1]. To show off the versatility of solution-phase biochemical computation using deoxyribozyme logic gates beyond demonstrations of binary arithmetic [2, 3], we built two versions of automata that played tic-tac-toe [4, 5]. Each played the game perfectly following its rules, and implemented a strategy that never made mistakes, i.e., it won whenever possible. The strategy was chosen by us and specified as a mapping from the opponent's moves to the automaton's responses. It was then translated by hand into a set of Boolean formulae, and each formula was implemented by a set of deoxyribozyme logic gates. Each such set of gates became a circuit that operated in one of nine test wells, accepting the opponent's moves coded as DNA strands, and signalling the automaton's responses via fluorescence.

As the power of molecular computing increases, larger circuits become practical. For instance, we showed three-input gates with arbitrary assignment of polarity [3]. We have also made progress on cascading gates, allowing greater effective fan-in. Groups working on DNA computing using other basic chemistry, such as strand-displacement reactions, have also built circuits capable of interrogating a large number of inputs [6]. This prompted us to revisit the problem of designing games and games strategies that can be rendered in biochemistry. The desiderata for a demonstration of a biochemical game-playing automaton could be:

– the game is non-trivial, perhaps even interesting, but is simple to describe (has compact rules);

- (mandatory) the game possesses a favorable strategy for one of the players, say, a strategy that never loses, and the logic design faithfully implements it;[5]
- (mandatory) the logic design is consistent with an implementation using biochemical logic gates;
- the complexity of (some of) the logic gates used is higher than before, to serve as a showcase for new capabilities, but not excessively high;
- the number of the gates needed is manageable, to contain cost;
- the amount of laboratory work to exhaustively test the automaton is manageable (which means the strategy covers a reasonably small number of game plays).

Of the above, using biochemical logic gates is the most serious constraint, for we consider gates that *cannot be turned off once activated*, as we shall explain shortly. Intuition suggests that this makes querying sequences of inputs difficult, and when, through trial and error, we had found a working strategy for tic-tac-toe [4] we were rather surprised. Here we tackle the two mandatory criteria above by developing a *systematic, constructive procedure* to determine whether, given a game, there exists a favorable strategy computable in biochemical logic. We consider only *position games*—games played on a board with a finite number of fields, in which the two players alternate claiming the fields, these claims are permanent, and the winner is decided depending on whether certain board configurations are reached.

In the following, we carefully define the notion of a game of strategy, its translation to logic, and the idiosyncratic realization of the logic in the wet lab (Section 2); first steps in this direction were taken by Andrews [7]. We now elaborate these definitions (Section 2), and then derive a computational procedure for generating strategies (Section 3). Finally we present the results of the procedure for several games in the tic-tac-toe family (Section 4). Perhaps surprisingly, the difficulty of a game, measured by the size or other complexity metrics applied to the results found, is not easy to predict from its apparent properties. Furthermore, the procedure may alternatively prove that no suitable strategy exists, and we exhibit such examples. The very existence of a solution also appears to be a subtle property and hard to predict. Our approach to computing strategies is effective on small examples; however, it does not assume or exploit any structure in the games and therefore does not scale to games with many fields.

2 Games and their biochemical representation

Deoxyribozyme logic. We are interested in circuits made out of deoxyribozyme logic gates. Deoxyribozymes are catalytically active single-stranded DNA; the catalytic function we exploit is that of cleaving other single-stranded DNA [8]. By adding a stem-loop control module to a basic deoxyribozyme, we are able to switch the complex from an inactive state to the catalytically active state by providing a control input, itself a single-stranded DNA [9]. We have built gates containing two such control modules, as well as a control module with the opposite sense. Interpreting the function of the complex as a

[5] Whether particular games possess a favorable strategy, and for which player, is a matter of keen study by combinatorial game theorists, and many familiar games, such as Connect Four, have only been solved in the last two decades.

digital logic gate, we were able to implement the Boolean functions id, $\neg$, $\wedge$, $\lambda xy.x \wedge \neg y$, and $\lambda xyz.x \wedge y \wedge \neg z$, i.e., some elementary conjunctions.

The rest of this paper can be understood without reference to biochemistry, except for the following points. A catalytic gate in its active state continuously cleaves substrate molecules into product molecules. It is the buildup of the product that we are able to observe, usually by fluorescence techniques. Thus, viewed as a dynamic system, an active gate is an integrator. In laboratory experiments, gates are run sufficiently long so that enough product builds up for reliable detection, thus the dynamic aspects need not concern us. However, once a product molecules has been created, it never goes away; therefore, even if the gate molecule itself is made inactive again (possible but nontrivial), the fluorescence of the products remains. In effect, the gate once on cannot be switched off, and this means the gate is usable for one-shot computation only. While this may be a disadvantage in some contexts, it is a good match to position games in which moves are permanent.

It is very easy to make a circuit with multiple conjunctions in a disjunction, simply by arranging several gates in the same solution to bind the same substrate and thus yield the same product—the activation of any gate suffices to build up detectable fluorescence. By the same token, *subsequent activation of another gate has no visible effect*, and this we can exploit to hide moves.

From strategy to Boolean formulae. We restrict attention to deterministic games of perfect information in which two players alternate, in each move claiming a field of a finite-size board, and these claims are permanent. Many widely played games fall into this category, including tic-tac-toe, Hex, and Connect Four. A strategy for one of the players (first or second) is a self-consistent map from board configurations to that player's moves. The map is only defined for board configurations in which it is that player's turn to move; furthermore by "self-consistent" we mean that the map needs to be defined exactly for those board configurations that are reachable by following the strategy itself.

A Boolean implementation of a strategy is a set of formulae, one for each field, to turn on or off the automaton's response in that field, as a function of the current board configuration. We now place further restrictions on the Boolean implementation in light of its subsequent realization in biochemical logic. Each field is represented as a separate well (test tube) which implements one formula in disjunctive normal form as a circuit consisting of several logic gates working in parallel. Opponent's moves are the inputs to the formulae and are presented as molecules keyed to each field but added to all the wells; this is the only means by which information is shared between the wells. Thus, the circuit in one well can query all the opponent's inputs, but not the outputs of the circuits in the remaining wells: the Boolean implementation of a strategy must be a function of the opponent's inputs alone, not the complete board configuration. We can now explain the idea of move hiding: we can tolerate the spurious activation of a gate upon the setting of an input (i.e., addition of a molecule), so long as, on that game path, another gate in the same well has already been activated by a previously set input (at an earlier move of the opponent).

A further distinction is possible depending on how we associate opponent's moves with Boolean inputs. A rich encoding [5] maps each ⟨field, turn number⟩ pair to a

distinct input, so the formulae can query the order of the opponent's moves. In a compact encoding, each field is keyed to just one input [4], so the formulae can only query the current set of the opponent's moves but not their order. In the following we assume a compact encoding.

The notion of isomorphism. In our first tic-tac-toe automaton, we restricted our opponent's first move; the automaton went first and claimed the center field  , upon which the two allowed moves for the opponent were the top left corner  or the left side  . We justified this restriction by an appeal to symmetry—all corners are alike, and all sides are alike. But we could have gone further. Indeed, in treating a game on a square board it is natural to consider board configurations as distinct only up to rotations and perhaps reflections. In general, this notion of board isomorphism is arbitrary, and represents a chosen interpretation of the game. For the 3×3 tic-tac-toe board, the equivalence classes under the natural isomorphism (rotation and reflection) contain at most 8 board configurations each (some, sparsely occupied, have just 4, 2, or 1).

To play a game under a notion of isomorphism, only one representative of each equivalence class under isomorphism is used. Given a board configuration, the appropriate player chooses an unoccupied field and claims it. The resulting board's equivalence class is looked up, and the representative of that class becomes the next board configuration. The new board configuration is a reshuffled version of the old one, which humans can grasp easily for geometrically motivated isomorphisms, but find confusing for arbitrary ones. However, it is not clear how to effect reshuffling in our laboratory protocol. Rotating a well plate is possible but it would be an external non-molecular computation; making a mirror image of a plate would be rather tricky. Reshuffling using molecular computation would require wells to turn off after they were on, and we must avoid that. But what if, somehow, reshuffling didn't actually have to reshuffle at all?

That is the approach we take: if the representative of each equivalence class is chosen deliberately, it may be possible to arrange for the new board configuration always to agree with the old one—for all possible moves from all possible board configuration, as prescribed by a strategy. Whether this "deliberate choice" of representatives actually exists for a given game turns out to be a complex problem in its own right. If it does, then it will *appear* to the human opponent as if the game was played naturally, without reshuffling—the only difference being that from a given board configuration, when it is the human's turn, not all the usual fields are permitted, but rather only at most one for each equivalence class under isomorphism. It is as if the human were supplied with a *rule book* that listed all board configurations and how one may move among them. This rule book is fair, as it simply projects the adopted notion of isomorphism onto the automaton's strategy; it is a generalization of our old restriction "on 1st move, go only to top left or left side".

3 From games to strategies: problem graph and constraints

To analyze a game we first construct a problem graph that captures all possible game plays; its vertices are board configurations (up to isomorphism), and its edges are moves.

We then choose a subgraph that represents a feasible strategy; to find one, we translate the problem graph into a constraint satisfaction problem and solve with an external tool.

We formalize the *rules of a game* parametrically, using here the names of these parameters in our Haskell implementation: A numFields value gives the board size. An initialBoard, commonly empty, gives the root for the problem graph. A mkSucc function maps a board to the set of legal successors, claiming one of the unoccupied fields. A hasMotif function computes if the board is final (commonly because one player's claimed fields form a motif, such as a three-in-a-row). A victoryMode value distinguishes normal game play (achieving a motif) from misère play (avoiding a motif). A fieldPerms list enumerates all permutations of the sequence $1 \cdots$ numFields that are to underlie the notion of board isomorphism.

Further parameters describe the *strategy* sought. A playerMode value selects whether the strategy is for the first or the second player. An outcomeMode governs which final outcomes the strategy must guarantee, commonly a win, or either a win or a draw.

To construct the problem graph, we compute all board configurations reachable from initialBoard using mkSucc, collapsing isomorphic ones. Obviously, the need to construct the graph explicitly limits our approach to fairly small games. After labelling final configurations as win, draw, or loss, we can solve the game, i.e., determine whether either player has a winning strategy, by a simple bottom up propagation.

We have not been able to express feasibility of strategies by any such simple computation, so we resort to a constraint-based approach. By inspection of the problem graph, we calculate all the constraints that must be met. Here we present one key example, clauses that express the feasibility of a strategy (deferring to a longer report the full description of all types of clauses used). This constraint captures the idea of avoiding conflicts in the strategy while exploiting move hiding, as first suggested in Ref. [4] and further elucidated in Andrews' thesis [7]:

> Consider vertices at a level of the graph where the automaton moves, i.e., where a choice needs to be made which single outgoing edge to keep for the strategy. For all pairs of these vertices, we compute the potential opponent move sets they have in common. Then for each such opponent move set, we formulate a clause stating that *if both vertices are kept, and if at both vertices the equivalence class representative under isomorphism is chosen such as to agree with the opponent move set, then either the outgoing moves from both vertices are labelled with the same field, or the outgoing edges from both fields are directed to the same vertex*.

In our current implementation, we express constraints directly in the Boolean domain and invoke an external SAT solver, Minisat [10]. If there is a solution, we interpret the satisfying assignment to recover the chosen vertices and edges to be kept, and the chosen equivalence class representatives. This forms the chosen strategy. We then read out the automaton response formulae and subject them to a step of Boolean minimization. For games with a notion of isomorphism we also read out the rule book for the opponent. We have written an independent validation procedure that follows all paths in the strategy and checks that the formulae evaluate correctly; the generated trace also represents the high-level protocol to be followed by the laboratory technician validating a biochemical realization of the strategy.

4 Results

The procedure described is effective and fast (completes in less than a minute of CPU time) at analyzing games on modest-size boards, such as tic-tac-toe. Table 1 is a summary of our initial findings. We examined eight game variants based on tic-tac-toe. All use the standard 3×3 board and three-in-a-row victory motif, and strategies aim for a win or a draw (since in each case, in game-theoretic terms, the game is a draw). As shown in the leftmost pane of the table, we vary the victory criterion (either achieving or avoiding the motif), the player for whom the strategy is sought (1st or 2nd), and the notion of board isomorphism (we consider only two options, the natural eight-way symmetry (rotations and reflection), or no isomorphism). The next pane gives the size of the problem graph from which the constraints are constructed. The third pane gives the size of the SAT instance generated; we include the ratio of the number of clauses to the number of variables as a tentative indicator of instance hardness. If there is no solution, the remaining columns have a dash. Otherwise, there is a solution for the strategy, and the fourth pane describes the circuit we obtained for it, after Boolean minimization, with the number of logic gates used and the maximum fan-in (number of input literals) needed. Finally, the rightmost column gives the number of play paths the chosen strategy entails.

Table 1: Analysis for eight problem variants based on tic-tac-toe.

victory	player	isomorphism	vertices	edges	variables	clauses	ratio	gates	fan-in	paths
normal	1st	natural	539	1212	45878	279798	6.1	23	3	18
normal	1st	none	3878	9331	338345	1623395	4.8	65	4	155
normal	2nd	natural	290	530	18642	119045	6.4	—	—	—
normal	2nd	none	2094	4085	126773	625558	4.9	—	—	—
misère	1st	natural	45	60	1749	11875	6.8	—	—	—
misère	1st	none	294	433	13951	65139	4.7	9	1	216
misère	2nd	natural	546	1395	43239	283159	6.6	39	5	121
misère	2nd	none	3950	10849	305599	1629398	5.3	120	8	503

Of the eight variants, five have solutions. In Figures 1 and 2 we focus on the first two rows of the table, first player normal play, and we show the problem graphs, derived solution strategies, and the Boolean formulae that implement them. The graphs are scaled down, and too small to be read in print, so we intend them primarily to convey a gestalt impression. However, they can be zoomed into to consult the vertex numbers, which match between the problem and the solution. The edges are labelled with the field being played into. The Boolean formulae map inputs i_0–i_8 to outputs o_0–o_8; the fields are numbered according to the schema $\begin{smallmatrix}0&1&2\\3&4&5\\6&7&8\end{smallmatrix}$. As expected, without isomorphism (Figure 2) the problem graph is nearly eight times bigger; the chosen strategy requires three times as many gates and nine times as many paths for validation. This comparison shows the utility of a notion of isomorphism as a disciplined way of reducing the materials and labor cost of the implementation.

On the other hand, it is instructive to compare the two variants for first player misère play (fifth and sixth row of Table 1). Both have small problem graphs, and the variant with no isomorphism has an exceptionally simple solution (which, after some thought, should be anticipated: one constant 1 in the center, and nine YES-gates surrounding it). Yet, the version with isomorphism has no solution. In this case, the additional constraints that result from merging distinct board configurations into single equivalence class representatives are responsible for the lack of a solution.

5 Discussion

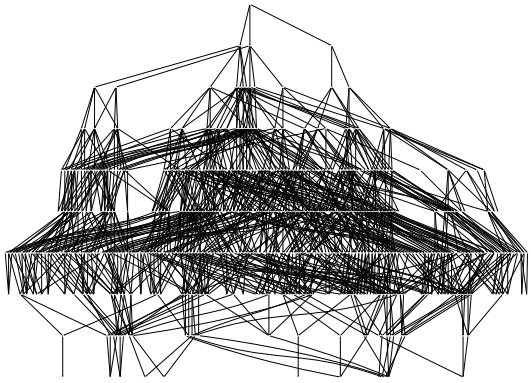
The space of strategies for position games can be enormous. As shown by Andrews [7], for tic-tac-toe (normal play, first player, no isomorphism) it contains $1.24 \cdot 10^{124}$ strategies, of which $2.6 \cdot 10^{103}$ are favorable. This is in contrast to just a quarter million possible plays. In a custom Monte Carlo approach, Andrews constructed strategies for this game that were feasible and favorable by intelligent design (effectively, by enumerating the local constraint space on the fly as the strategy was constructed level by level) and checked them for implementability within particular technologies (available elementary AND gates). Sampling 50 million strategies, he quickly found five-literal implementations, but could not find a four-literal one; our speculation at the time was that gates with five inputs of arbitrary polarity were needed. We were surprised that the very first strategy generated by our new method was implementable with just up to four-input gates. A possible topic for future work is using an all-solutions constraint solver to systematically sample the solution space.

Our new method improves on the previous approach—it is general and applicable to arbitrary games; it is modular, separating play rules from victory modes from strategy requirements; and it is purely declarative, the constraint-solving procedural details having been offloaded to an external solver. The principal conceptual advance, however, is the principled and integrated treatment of game symmetries via the notion of board isomorphism.

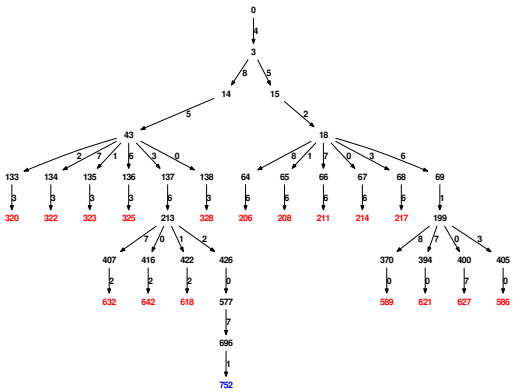
We are currently investigating how well our approach will scale to larger games. The problem graphs grow quickly with the number of fields, and thus also the SAT instances we generate, surpassing the capacity of off-the-shelf solvers at about 15 fields in the board. It is not clear whether these instances also become intrinsically harder, and this seems to be subtly dependent on the game rules.

Related work. The literature on combinatorial games is vast and scattered, though the periodical *Games of No Chance* (Mathematical Sciences Research Institute) is a focal point. We have not been able to find prior work on the question of computability of a strategy in (limited) logic. There is a growing body of work in molecular computing; our game analysis approach can be applied to various logic-gate implementations, adapting the constraints to match the particulars of those implementations. Most demonstrations in molecular computing so far have been small to medium-scale. We hope that the development of design tools will facilitate the construction of convincing large circuits.

Acknowledgments. We are grateful to Ben Andrews for many inspirational conversations. We sincerely thank the conference reviewers for their incisive comments, which



(a) Problem graph

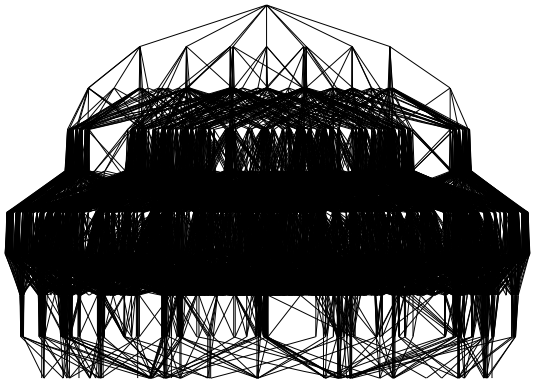


(b) Selected strategy (red leaves: win; blue leaves: draw)

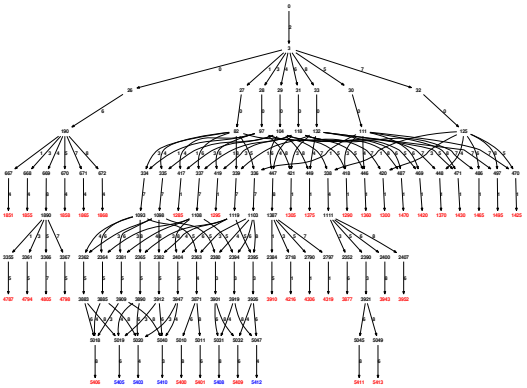
$$
\begin{aligned}
o_0 &= (i_3 \wedge i_6) \vee (i_6 \wedge i_7) \vee (i_2 \wedge i_3) \vee (i_5 \wedge i_6 \wedge i_8) \\
o_1 &= (i_6 \wedge \neg i_8) \vee (i_2 \wedge i_7) \\
o_2 &= i_5 \vee (i_0 \wedge i_3) \vee (i_1 \wedge i_3) \vee (\neg i_2 \wedge i_3 \wedge i_7) \\
o_3 &= (\neg i_5 \wedge i_6) \vee (i_2 \wedge \neg i_3) \vee (\neg i_3 \wedge i_7 \wedge i_8) \vee (i_0 \wedge \neg i_3 \wedge i_8) \vee (i_1 \wedge \neg i_3 \wedge i_8) \\
o_4 &= 1 \\
o_5 &= (\neg i_5 \wedge i_8) \\
o_6 &= (i_3 \wedge \neg i_6) \vee (i_1 \wedge \neg i_8) \vee (i_5 \wedge \neg i_6 \wedge i_8) \vee (\neg i_6 \wedge i_7 \wedge \neg i_8) \vee (i_0 \wedge \neg i_6 \wedge \neg i_8) \\
o_7 &= (i_0 \wedge i_6) \\
o_8 &= 0
\end{aligned}
$$

(c) Boolean formulae implementing the strategy

Fig. 1: Tic-tac-toe: Normal play, first player, isomorphism is rotation and reflection.



(a) Problem graph



(b) Selected strategy (red leaves: win; blue leaves: draw)

$$
\begin{aligned}
o_0 &= i_6 \vee (\neg i_0 \wedge i_1) \vee (\neg i_0 \wedge i_4) \vee (\neg i_0 \wedge i_3) \vee (\neg i_0 \wedge i_5) \vee (\neg i_0 \wedge i_8) \vee (\neg i_0 \wedge i_7) \\
o_1 &= (\neg i_1 \wedge i_4 \wedge i_8) \vee (\neg i_1 \wedge i_3 \wedge i_5) \vee (\neg i_1 \wedge i_3 \wedge i_8) \vee (\neg i_1 \wedge i_5 \wedge i_8) \vee (\neg i_1 \wedge i_3 \wedge i_6) \\
&\quad \vee (\neg i_1 \wedge i_5 \wedge i_6) \vee (\neg i_1 \wedge i_6 \wedge i_8) \vee (\neg i_1 \wedge i_3 \wedge i_7) \vee (\neg i_1 \wedge i_5 \wedge i_7) \vee (\neg i_1 \wedge i_7 \wedge i_8) \\
&\quad \vee (\neg i_1 \wedge i_6 \wedge i_7) \vee (\neg i_0 \wedge i_4 \wedge i_7) \vee (\neg i_0 \wedge \neg i_1 \wedge i_3 \wedge i_4) \vee (\neg i_0 \wedge \neg i_1 \wedge i_4 \wedge i_5) \\
o_2 &= 1 \\
o_3 &= (i_1 \wedge i_4 \wedge i_5) \vee (i_4 \wedge i_6 \wedge i_8) \vee (i_1 \wedge i_5 \wedge i_7) \vee (i_1 \wedge \neg i_3 \wedge i_5 \wedge i_8) \vee (i_1 \wedge \neg i_3 \wedge i_5 \wedge i_6) \\
o_4 &= (i_1 \wedge i_7) \vee (i_0 \wedge i_8) \vee (i_1 \wedge i_3 \wedge i_5) \vee (i_3 \wedge i_6 \wedge i_8) \vee (i_5 \wedge i_6 \wedge i_8) \vee (i_0 \wedge i_1 \wedge \neg i_4) \\
&\quad \vee (i_0 \wedge i_3 \wedge \neg i_4) \vee (i_0 \wedge \neg i_4 \wedge i_5) \vee (i_0 \wedge \neg i_4 \wedge i_7) \\
o_5 &= (i_1 \wedge i_3 \wedge i_4) \vee (i_0 \wedge i_1 \wedge i_4) \vee (i_0 \wedge i_3 \wedge i_4) \vee (i_0 \wedge i_4 \wedge i_7) \vee (i_1 \wedge i_4 \wedge \neg i_5 \wedge i_8) \\
&\quad \vee (i_1 \wedge i_4 \wedge \neg i_5 \wedge i_6) \vee (i_1 \wedge i_3 \wedge \neg i_5 \wedge i_8) \vee (i_1 \wedge i_3 \wedge \neg i_5 \wedge i_6) \vee (i_1 \wedge \neg i_5 \wedge i_6 \wedge i_8) \\
o_6 &= i_0 \vee (i_3 \wedge i_4 \wedge i_8) \vee (i_4 \wedge i_5 \wedge i_8) \vee (i_3 \wedge i_5 \wedge i_8) \vee (i_1 \wedge i_3 \wedge i_7) \vee (i_1 \wedge i_7 \wedge i_8) \\
o_7 &= (i_0 \wedge i_4 \wedge i_5) \vee (i_1 \wedge i_3 \wedge \neg i_7) \vee (i_1 \wedge i_5 \wedge \neg i_7) \vee (i_1 \wedge \neg i_7 \wedge i_8) \vee (\neg i_0 \wedge i_1 \wedge i_4 \wedge \neg i_6) \\
&\quad \vee (i_1 \wedge \neg i_4 \wedge i_6 \wedge \neg i_7) \\
o_8 &= (i_0 \wedge i_4) \vee (i_3 \wedge i_4 \wedge i_6) \vee (i_4 \wedge i_5 \wedge i_6) \vee (i_3 \wedge i_5 \wedge i_6) \vee (i_1 \wedge i_6 \wedge i_7) \vee (\neg i_1 \wedge i_4 \wedge i_6)
\end{aligned}
$$

(c) Boolean formulae implementing the strategy

Fig. 2: Tic-tac-toe: Normal play, first player, no isomorphism.

helped us to clarify the presentation, we hope. This material is based upon work supported by the National Science Foundation under grant 1028238.

References

1. Michie, D.: Trial and error. In: Science Survey, part 2. Penguin, Harmondsworth (1961) 129–145
2. Stojanovic, M.N., Stefanovic, D.: Deoxyribozyme-based half adder. Journal of the American Chemical Society **125**(22) (2003) 6673–6676
3. Lederman, H., Macdonald, J., Stefanovic, D., Stojanovic, M.N.: Deoxyribozyme-based three-input logic gates and construction of a molecular full adder. Biochemistry **45**(4) (2006) 1194–1199
4. Stojanovic, M.N., Stefanovic, D.: A deoxyribozyme-based molecular automaton. Nature Biotechnology **21**(9) (September 2003) 1069–1074
5. Macdonald, J., Li, Y., Sutovic, M., Lederman, H., Pendri, K., Lu, W., Andrews, B.L., Stefanovic, D., Stojanovic, M.N.: Medium scale integration of molecular logic gates in an automaton. Nano Letters **6**(11) (2006) 2598–2603
6. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science **332** (2011) 1196–1201
7. Andrews, B.: Games, strategies, and boolean formula manipulation. Master's thesis, University of New Mexico (December 2005)
8. Breaker, R.R., Joyce, G.F.: A DNA enzyme that cleaves RNA. Chemistry and Biology **1** (December 1994) 223–229
9. Stojanovic, M.N., Mitchell, T.E., Stefanovic, D.: Deoxyribozyme-based logic gates. Journal of the American Chemical Society **124**(14) (April 2002) 3555–3561
10. Eén, N., Sorensson, N.: An extensible SAT-solver. In: SAT. (2003)