Homework 3 — ML core language — assigned Tuesday 22 February — due Tuesday 8 March

In all exercises, you should use higher-order functions, especially library functions from the `List` structure, in preference to pointwise recursive function definitions, wherever possible. You may use all features of ML other than mutable storage. It is permissible to use all code developed in previous homework exercises.

3.1 Using lists for arithmetic: writing recursive functions over lists (30pts)

This is a continuation of exercise 2.1. Numerals can be represented as lists of integers. For instance, decimal numerals can be expressed as lists of integers from 0 to 9. In this representation, the integer 12345678901234567890 would be represented as the ML list
`[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]: int list`.

Write the function `mulLongInts: int -> (int list * int list) -> int list`, such that `mulLongInts` r (a,b) computes the product of the nonnegative integers given by the lists a and b; all lists use the same radix r. Lists a and b can represent arbitrarily large positive integers. Be careful to state any preconditions that must be met by r.

It is not permissible to implement the multiplication of x and y as $\sum_1^x y$.

3.2 Drawing (30pts)

This is a continuation of exercise 2.2. Write an ML function

```
plotFunctions:
  {
   functions: (real -> real) list,
   interval: {lower: real, upper: real},
   numPoints: int
  }
  -> string
```

The argument to the function contains a list of ML `real -> real` functions that compute mathematical real functions; a real interval given by its lower and upper boundary; and the number of interior points to be used for function evaluation, spaced uniformly *between* the lower and the upper boundary. The result of the function is a string consisting of valid PostScript code.

The ML function `plotFunctions` should divide the given interval into equal-length sections and thus determine a set of points, including both the interior points and the boundaries of the interval. It should evaluate each of the functions given at each of these points. Finally, it should emit PostScript commands that draw a line graph of each of the functions given. Each line graph should be in a different color, but none should be black.

You may assume that there will be no singularities.

The function `plotFunctions` should determine the region of the plane spanned by the given functions over the given interval. If the *x*-axis passes through this region, it should be drawn in black. If the *y*-axis passes through this region, it should be drawn in black.

Optional: label the line graphs, and draw labelled tick marks on the axes.

Hint: we are attempting to write a crude version of the gnuplot utility; you may follow gnuplot's choice of line graph colors. Alternatively, you may make the colors as different from each other as possible, in a suitable color space.

3.3 Processing text (40pts)

This is a continuation of exercise 2.4.

3.3.1 Simple word counting (20pts)

The Unix command `wc` reports the number of lines, words, and bytes in a file. For instance:

```
# wc homework2.pdf
     426    1492   25002 homework2.pdf
```

Write an ML function `wc: string -> string` to mimic the Unix `wc`. E.g., `wc "homework2.pdf"` ⇝ `"     426    1492   25002 homework2.pdf\n"`.

It is not permissible to invoke the Unix `wc`.

Note that the definition of what a word is and what a line is must be exactly as in the Unix `wc`, which is not the same as in our exercise 2.3. In particular, the POSIX standard treats control characters and we did not. Consult the `wc` and `iswspace` manual pages.

3.3.2 Word counting and tabulating (20pts)

Write an ML function `tabulate: string -> (string * int) list`, such that `tabulate` *f* reads a file named *f*, splits it into words in the same way as for `wc`, and as the result produces a list containing all the words paired with the number of times each occurs in the text, sorted in decreasing order of that number.

3.4 Games and strategies (40pts extra credit)

The accompanying document (see Appendix) describes a systematic (but manual) procedure for deriving a logic circuit array that operates as an automaton for the (restricted) game of tic-tac-toe. The array consists of nine logic circuits, each of which implements the automaton's response move in one field of the tic-tac-toe board. The inputs to the circuits are the eight permitted opponent's moves. Each circuit is defined by a Boolean formula in disjunctive normal form. Refer to the document for full details.

In this assignment we explore an alternative representation. Again, the goal is to implement an automaton that plays tic-tac-toe and does not lose. The automaton again goes first and moves into the center field.

However, the strategy should not use symmetry pruning, i.e., the opponent must be free to make any legal move.

We use a richer encoding of the opponent's moves. Namely, instead of 8 input signals, we use 32. Each input signal encodes both the field moved to and the number of the move. The opponent can make at most four moves. On each it moves to one of eight fields. We name the inputs $i_{m,f}$, where $m \in \{1,2,3,4\}$ and $f \in \{1,2,3,4,6,7,8,9\}$. The purpose of this encoding is to not have *conflicts* as described in the Appendix.

Again, we have nine circuits, eight of which are interesting. (The circuit for field 5 is always on.) Each interesting circuit should be expressed as a disjunctive normal form. Each disjunct must be a single signal, or a single signal negated, or a conjunction of two signals (each of which may be negated).

Devise a strategy (a "game tree"), analogous to Figure 2, for the richer encoding, such that the strategy can be implemented using logic circuits as just described. There must be no conflicts.

You need not worry about the "technology mapping" stage.

Report the set of logic circuits. Test them by emulating the play of all games permissible within the strategy.

Strategies belong to a space of possible game trees, which are (essentially) different labellings of a common tree structure; it is in this space that you must search for a solution.

Use ML to write your search program.

How to turn in

Turn in your code by running

~jackmp/cs451TA/handin your-file

on a regular UNM CS machine. You should use whatever filename is appropriate in place of your-file.

Include the following statement with your submission, signed and dated:
I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.

A From the Supplementary Material to *Deoxyribozyme-Based Molecular Automaton, Nature Biotechnology*, 21, 1069–1074 (2003).

Milan N. Stojanovic* & Darko Stefanovic†

**Department of Medicine, Columbia University, New York, New York 10023, USA,*
†*Department of Computer Science, University of New Mexico, Albuquerque, New Mexico 87131, USA*

Analysis of the game of tic-tac-toe, derivation of the logic gate implementation, and mapping to deoxyribozyme logic gate technology

The purpose of this addendum is to describe a systematic procedure for deriving a logic gate design of an automaton for playing a simple game described by a decision tree, and to describe how the logic gate design is adapted so it can be implemented using currently available deoxyribozyme logic gates.

Symmetry-pruned game

In building the first molecular-scale automaton for the game of tic-tac-toe, we wished to reduce the number and complexity of needed molecular species for the chemical implementation. We make the following simplifiying assumptions. The automaton moves first. Furthermore, its first move is into the center (square 5, Figure 1). Without loss of generality, we exploit the symmetries available in the game and we restrict the first move of the human, which must be either a side move or a corner move, to be either square 1 (corner) or square 4 (side). From this point in the game on, we make no restrictions. Of course, we assume that each move of the human is legal, just as we ensure that each move of the automaton is legal (and clever). We assume that the game ends immediately if a straight line has been achieved, i.e., a player has won.

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

Figure 1: The tic-tac-toe game board.

Game tree

The game tree in Figure 2 represents the chosen strategy for the automaton playing the pruned game. For example, if the human opponent moves into square 1 following the automaton's opening move into square 5, the automaton responds by moving into square 4 (as indicated on edge 21). If the human then moves into square 6, the automaton responds by moving into square 3 (edge 22). If the human then moves into square 7, the automaton responds by moving into square 2 (edge 23). Finally, if the human then moves into square 8, the automaton responds by moving into square 9, and the game ends in a draw.

Towards Boolean formulæ for square outputs

From the game tree we read off the Boolean formulæ to compute automaton outputs. For instance, in state $\cdot 1 \cdot 6 \cdot$ on input 2, output 7 should become active (edge 6). In this state inputs 1 and 6 are already

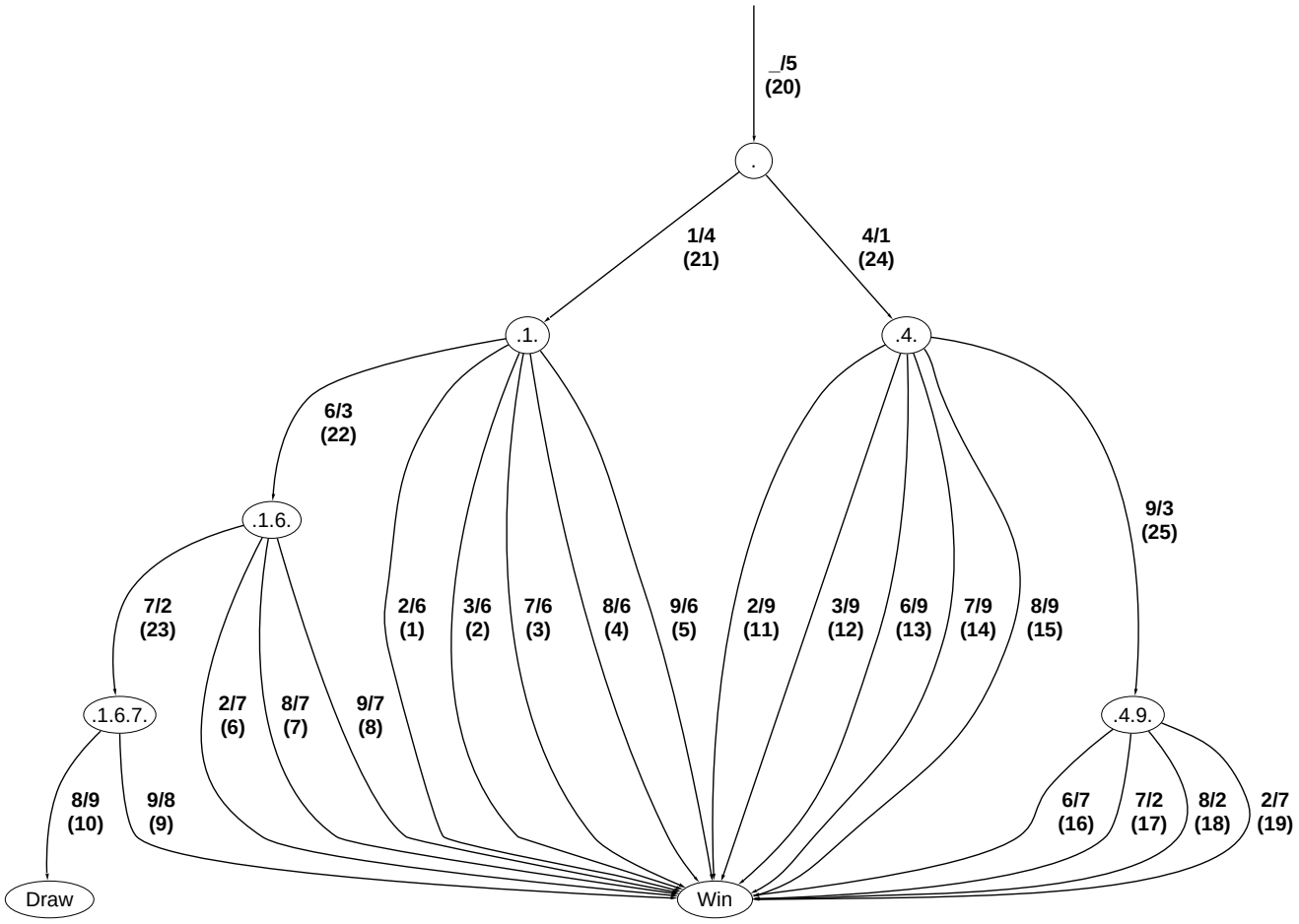

Figure 2: Game tree for the symmetry-pruned game of tic-tac-toe, drawn as the diagram of a Mealy automaton. Each state is labelled according to the inputs seen on the path to it. Each edge is labelled $a/b(n)$, where b is the output that is activated on input a, and n is the edge identifier.

present in solution, thus the formula for o_7 should include a conjunctive term $i_1 \wedge i_6 \wedge i_2$.

Proceeding similarly for all edges, we arrive at the following tentative formulæ, in which edges are identified underneath each conjunction:

$$o_1 = \underbrace{i_4}_{24}$$

$$o_2 = \underbrace{(i_1 \wedge i_6 \wedge i_7)}_{23} \vee \underbrace{(i_4 \wedge i_9 \wedge i_7)}_{17} \vee \underbrace{(i_4 \wedge i_9 \wedge i_8)}_{18}$$

$$o_3 = \underbrace{(i_1 \wedge i_6)}_{22} \vee \underbrace{(i_4 \wedge i_9)}_{25}$$

$$o_4 = \underbrace{i_1}_{21}$$

$$o_5 = \underbrace{1}_{20}$$

$$o_6 = \underbrace{(i_1 \wedge i_2)}_{1} \vee \underbrace{(i_1 \wedge i_3)}_{2} \vee \underbrace{(i_1 \wedge i_7)}_{3} \vee \underbrace{(i_1 \wedge i_8)}_{4} \vee \underbrace{(i_1 \wedge i_9)}_{5}$$

$$o_7 = \underbrace{(i_1 \wedge i_6 \wedge i_2)}_{6} \vee \underbrace{(i_1 \wedge i_6 \wedge i_8)}_{7} \vee \underbrace{(i_1 \wedge i_6 \wedge i_9)}_{8} \vee \underbrace{(i_4 \wedge i_9 \wedge i_6)}_{16} \vee \underbrace{(i_4 \wedge i_9 \wedge i_2)}_{19}$$

$$o_8 = \underbrace{i_1 \wedge i_6 \wedge i_7 \wedge i_9}_{9}$$

$$o_9 = \underbrace{(i_1 \wedge i_6 \wedge i_7 \wedge i_8)}_{10} \vee \underbrace{(i_4 \wedge i_2)}_{11} \vee \underbrace{(i_4 \wedge i_3)}_{12} \vee \underbrace{(i_4 \wedge i_6)}_{13} \vee \underbrace{(i_4 \wedge i_7)}_{14} \vee \underbrace{(i_4 \wedge i_8)}_{15}$$

Each output formula is intended to specify a set of gates for a single well representing a board square. Each formula is in disjunctive normal form, and each conjunction maps to a deoxyribozyme logic gate, and the several gates act disjunctively by cleaving the same substrate.

Unfortunately, the formulæ as written do not produce correct automaton behaviour; instead, in many cases more than one square is claimed by the automaton in response to the human move. The reason is in conflicts in the paths (sequences of inputs seen) within the game tree. A conflict arises if on two paths the same input must produce different outputs (unless the spurious output is already present). More formally: there is a conflict between two paths $p = p_1 \cdots p_{l_p}$ and $q = q_1 \cdots q_{l_q}$, with lengths l_p and l_q, where $l_p < l_q$, if $p_{l_p} = q_{l_q}$, and $\{p_1, \ldots, p_{l_p}\} \subset \{q_1, \ldots q_{l_q}\}$, and $p \mapsto o_p$, and $(\forall k \in \{1, \ldots, l_q\}) o_p \neq o$, where $q_1 \cdots q_k \mapsto o$.

Conflicts in the game tree arise because an admixture of chemical inputs, in addition to participating in the reactions requiring *all* the inputs, also participates in reactions enabled by all *subsets* thereof. We are working on new mechanisms that permit sequencing the inputs, analogous to synchronous (clocked) electronic logic circuits; for the simple game of tic-tac-toe, this was not necessary.

In the game tree at hand, there are the following conflicts:[1]

[1]The path is a string of inputs, but for convenience we also indicate the outputs, wedged as superscripts between inputs, the

$$^5 1^4 6^3 2 \mapsto 7 \text{ conflicts with } ^5 1^4 2 \mapsto 6$$
$$^5 1^4 6^3 8 \mapsto 7 \text{ conflicts with } ^5 1^4 8 \mapsto 6$$
$$^5 1^4 6^3 9 \mapsto 7 \text{ conflicts with } ^5 1^4 9 \mapsto 6$$
$$^5 1^4 6^3 7 \mapsto 2 \text{ conflicts with } ^5 1^4 7 \mapsto 6$$
$$^5 1^4 6^3 7^2 8 \mapsto 9 \text{ conflicts with } ^5 1^4 8 \mapsto 6$$
$$^5 1^4 6^3 7^2 8 \mapsto 9 \text{ conflicts with } ^5 1^4 6^3 8 \mapsto 7$$
$$^5 1^4 6^3 7^2 9 \mapsto 8 \text{ conflicts with } ^5 1^4 9 \mapsto 6$$
$$^5 1^4 6^3 7^2 9 \mapsto 8 \text{ conflicts with } ^5 1^4 6^3 9 \mapsto 7$$
$$^5 4^1 9^3 2 \mapsto 7 \text{ conflicts with } ^5 4^1 2 \mapsto 9$$
$$^5 4^1 9^3 6 \mapsto 7 \text{ conflicts with } ^5 4^1 6 \mapsto 9$$
$$^5 4^1 9^3 7 \mapsto 2 \text{ conflicts with } ^5 4^1 7 \mapsto 9$$
$$^5 4^1 9^3 8 \mapsto 2 \text{ conflicts with } ^5 4^1 8 \mapsto 9$$

Resolving conflicts

Since the positive presence of sets of inputs is not sufficient to disambiguate states, our remedy is to disambiguate subtrees of the game tree explicitly, by introducing negative literals. We do this by inspection of the game tree.

Paths $^5 1^4 6^3 2$, $^5 1^4 6^3 8$, $^5 1^4 6^3 9$ and $^5 1^4 6^3 7$ (with final edges 1, 3, 4 and 5) are uniquely distinguished from the paths they conflict with, $^5 1^4 2$, $^5 1^4 8$, $^5 1^4 9$ and $^5 1^4 7$, by the absence of input 6 in the latter. Therefore a factor $\wedge\neg i_6$ will be added to the formulæ for the latter to resolve the conflicts.

Similarly, paths $^5 4^1 9^3 2$, $^5 4^1 9^3 6$, $^5 4^1 9^3 7$ and $^5 4^1 9^3 8$ (with final edges 19, 16, 17 and 18) are uniquely distinguished from the paths they conflict with, $^5 4^1 2$, $^5 4^1 6$, $^5 4^1 7$ and $^5 4^1 8$, by the absence of input 9 in the latter. Therefore a factor $\wedge\neg i_9$ will be added to the formulæ for the latter to resolve the conflicts.

Paths $^5 1^4 6^3 7^2 8$ and $^5 1^4 6^3 7^2 9$ now no longer conflict with paths $^5 1^4 8$ and $^5 1^4 9$, since the latter have had $\wedge\neg i_6$ added. They still conflict with $^5 1^4 6^3 8$ and $^5 1^4 6^3 9$, but can be uniquely distinguished from the latter by the absence of input 7 in it. Therefore a factor $\wedge\neg i_7$ will be added to the formulæ for the latter to resolve the conflicts.

In consequence we obtain the following formulæ for the logic gates in all 9 squares of the game board:

---

outputs being fully determined by the inputs.

$$o_1 = \underbrace{i_4}_{24}$$

$$o_2 = \underbrace{(i_1 \wedge i_6 \wedge i_7)}_{23} \vee \underbrace{(i_4 \wedge i_9 \wedge i_7)}_{17} \vee \underbrace{(i_4 \wedge i_9 \wedge i_8)}_{18}$$

$$o_3 = \underbrace{(i_1 \wedge i_6)}_{22} \vee \underbrace{(i_4 \wedge i_9)}_{25}$$

$$o_4 = \underbrace{i_1}_{21}$$

$$o_5 = \underbrace{1}_{20}$$

$$o_6 = \underbrace{(i_1 \wedge i_2 \wedge \neg i_6)}_{1} \vee \underbrace{(i_1 \wedge i_3)}_{2} \vee \underbrace{(i_1 \wedge i_7 \wedge \neg i_6)}_{3} \vee \underbrace{(i_1 \wedge i_8 \wedge \neg i_6)}_{4} \vee \underbrace{(i_1 \wedge i_9 \wedge \neg i_6)}_{5}$$

$$o_7 = \underbrace{(i_1 \wedge i_6 \wedge i_2)}_{6} \vee \underbrace{(i_1 \wedge i_6 \wedge i_8 \wedge \neg i_7)}_{7} \vee \underbrace{(i_1 \wedge i_6 \wedge i_9 \wedge \neg i_7)}_{8} \vee \underbrace{(i_4 \wedge i_9 \wedge i_6)}_{16} \vee \underbrace{(i_4 \wedge i_9 \wedge i_2)}_{19}$$

$$o_8 = \underbrace{i_1 \wedge i_6 \wedge i_7 \wedge i_9}_{9}$$

$$o_9 = \underbrace{(i_1 \wedge i_6 \wedge i_7 \wedge i_8)}_{10} \vee \underbrace{(i_4 \wedge i_2 \wedge \neg i_9)}_{11} \vee \underbrace{(i_4 \wedge i_3)}_{12} \vee \underbrace{(i_4 \wedge i_6 \wedge \neg i_9)}_{13} \vee \underbrace{(i_4 \wedge i_7 \wedge \neg i_9)}_{14} \vee \underbrace{(i_4 \wedge i_8 \wedge \neg i_9)}_{15}$$

In general problems of this nature, it may not always be possible to resolve conflicts. In particular, if two permutations of inputs (same number of inputs but in different order) must trigger distinct outputs, that conflict cannot be resolved.

Technology mapping

The formulæ above use 25 different conjunctions, summarized in Table 1.

type of formula	number of formulæ	used for edges
0-input (constant) 1	1	20
1-input affirmative a	2	21, 24
2-input $a \wedge b$	4	2, 12, 22, 25
3-input $a \wedge b \wedge c$	6	6, 16, 17, 18, 19, 23
3-input $a \wedge b \wedge \neg c$	8	1, 3, 4, 5, 11, 13, 14, 15
4-input $a \wedge b \wedge c \wedge d$	2	9, 10
4-input $a \wedge b \wedge c \wedge \neg d$	2	7, 8

Table 1: Boolean formulæ resulting from the game tree, before technology mapping.

We wish to implement these formulæ using single gates of deoxyribozyme logic. At the time the experiment was performed, we had not yet perfected gates for all of the types of Boolean formulæ proposed; in particular, only 0-, 1-, and 2-input gates could be constructed arbitrarily, whereas 3-input gates had to have one inhibitory input (one negative literal), and no 4-input gates were available. Therefore we must

replace all 10 conjunctions of types $a \wedge b \wedge c$, $a \wedge b \wedge c \wedge d$ and $a \wedge b \wedge c \wedge \neg d$ with equivalent ones using only available types. In general this may require using multiple gates to build an equivalent replacement network. However, for the game tree at hand there exists a replacement that maintains the single-layer disjunctive normal form of the formula for each output but each conjunct is permitted by technology.

In our game tree, the law $i_1 \Leftrightarrow \neg i_4$ holds. Therefore we have the following simple replacements for the 3-input gates:

for edge 6: $i_1 \wedge i_6 \wedge i_2 = i_2 \wedge i_6 \wedge \neg i_4$
for edge 16: $i_4 \wedge i_9 \wedge i_6 = i_6 \wedge i_9 \wedge \neg i_1$
for edge 17: $i_4 \wedge i_9 \wedge i_7 = i_7 \wedge i_9 \wedge \neg i_1$
for edge 18: $i_4 \wedge i_9 \wedge i_8 = i_8 \wedge i_9 \wedge \neg i_1$
for edge 19: $i_4 \wedge i_9 \wedge i_2 = i_9 \wedge i_2 \wedge \neg i_1$
for edge 23: $i_1 \wedge i_6 \wedge i_7 = i_6 \wedge i_7 \wedge \neg i_4$

Now consider the 4-input gates. For edge 9, $i_1 \wedge i_6 \wedge i_7 \wedge i_9$, we observe that the law $i_1 \wedge i_9 \wedge i_7 \Rightarrow i_6$ holds in the game tree, the only path with all three inputs 1, 9 and 7 being $\cdot 1 \cdot 6 \cdot 7 \cdot 9 \cdot$. Therefore $i_1 \wedge i_6 \wedge i_7 \wedge i_9 = i_1 \wedge i_7 \wedge i_9 = i_9 \wedge i_7 \wedge \neg i_4$.

Similarly for edge 10, $i_1 \wedge i_6 \wedge i_7 \wedge i_8$, we observe that the law $i_7 \wedge i_8 \Rightarrow i_6$ holds in the game tree, the only path with both inputs 8 and 7 being $\cdot 1 \cdot 6 \cdot 7 \cdot 8 \cdot$. Therefore $i_1 \wedge i_6 \wedge i_7 \wedge i_8 = i_1 \wedge i_7 \wedge i_8 = i_7 \wedge i_8 \wedge \neg i_4$.

For edge 7, $i_1 \wedge i_6 \wedge i_8 \wedge \neg i_7$, we observe that the law $i_6 \wedge i_8 \Rightarrow i_1$ holds in the game tree, the only two paths with both inputs 6 and 8 being $\cdot 1 \cdot 6 \cdot 7 \cdot 8 \cdot$ and $\cdot 1 \cdot 6 \cdot 8 \cdot$. Therefore $i_1 \wedge i_6 \wedge i_8 \wedge \neg i_7 = i_6 \wedge i_8 \wedge \neg i_7$.

Finally, we must deal with edge 8, $i_1 \wedge i_6 \wedge i_9 \wedge \neg i_7$. We cannot simplify it in isolation; instead we will merge it with another term, for edge 16, that contributes to the same output, o_7, and then simplify. In our game tree, the law $i_6 \wedge i_9 \Rightarrow \neg i_7 \vee i_1$ holds, by inspection of all the paths with both inputs 6 and 9, viz. $\cdot 1 \cdot 6 \cdot 7 \cdot 9 \cdot$, $\cdot 1 \cdot 6 \cdot 9 \cdot$ and $\cdot 4 \cdot 9 \cdot 6 \cdot$. We can rewrite the law as $i_6 \wedge i_9 \Rightarrow (\neg i_7 \Leftrightarrow (i_1 \wedge \neg i_7) \vee \neg i_1)$.

Therefore the disjunction of the formulæ for edges 8 and 16 is: $(i_1 \wedge i_6 \wedge i_9 \wedge \neg i_7) \vee (i_4 \wedge i_9 \wedge i_6) = (i_6 \wedge i_9 \wedge i_1 \wedge \neg i_7) \vee (i_6 \wedge i_9 \wedge \neg i_1) = i_6 \wedge i_9 \wedge ((i_1 \wedge \neg i_7) \vee \neg i_1) = i_6 \wedge i_9 \wedge \neg i_7$ by the law above.

Finally, the result of technology mapping is the following set of formulæ:

$$o_1 = \underbrace{i_4}_{24}$$

$$o_2 = \underbrace{(i_6 \wedge i_7 \wedge \neg i_4)}_{23} \vee \underbrace{(i_7 \wedge i_9 \wedge \neg i_1)}_{17} \vee \underbrace{(i_8 \wedge i_9 \wedge \neg i_1)}_{18}$$

$$o_3 = \underbrace{(i_1 \wedge i_6)}_{22} \vee \underbrace{(i_4 \wedge i_9)}_{25}$$

$$o_4 = \underbrace{i_1}_{21}$$

$$o_5 = \underbrace{1}_{20}$$

$$o_6 = \underbrace{(i_1 \wedge i_2 \wedge \neg i_6)}_{1} \vee \underbrace{(i_1 \wedge i_3)}_{2} \vee \underbrace{(i_1 \wedge i_7 \wedge \neg i_6)}_{3} \vee \underbrace{(i_1 \wedge i_8 \wedge \neg i_6)}_{4} \vee \underbrace{(i_1 \wedge i_9 \wedge \neg i_6)}_{5}$$

$$o_7 = \underbrace{(i_2 \wedge i_6 \wedge \neg i_4)}_{6} \vee \underbrace{(i_6 \wedge i_8 \wedge \neg i_7)}_{7} \vee \underbrace{(i_6 \wedge i_9 \wedge \neg i_7)}_{8 \text{ and } 16} \vee \underbrace{(i_9 \wedge i_2 \wedge \neg i_1)}_{19}$$

$$o_8 = \underbrace{i_9 \wedge i_7 \wedge \neg i_4}_{9}$$

$$o_9 = \underbrace{(i_7 \wedge i_8 \wedge \neg i_4)}_{10} \vee \underbrace{(i_4 \wedge i_2 \wedge \neg i_9)}_{11} \vee \underbrace{(i_4 \wedge i_3)}_{12} \vee \underbrace{(i_4 \wedge i_6 \wedge \neg i_9)}_{13} \vee \underbrace{(i_4 \wedge i_7 \wedge \neg i_9)}_{14} \vee \underbrace{(i_4 \wedge i_8 \wedge \neg i_9)}_{15}$$

In solving general problems of this nature, automated tools for Boolean minimization can be used, with some adaptation.

Technology mapping in the laboratory

For edge 2, the actual automaton built in the laboratory has $i_1 \wedge i_3 \wedge \neg i_6$ instead of $i_1 \wedge i_3$. This is correct because the law $i_1 \wedge i_3 \Rightarrow \neg i_6$ holds in our game tree.

For edge 6, the actual automaton built in the laboratory has $i_2 \wedge i_6 \wedge \neg i_7$ instead of $i_2 \wedge i_6 \wedge \neg i_4$. This is correct because the law $i_2 \wedge i_6 \Rightarrow (i_4 \Leftrightarrow i_7)$ holds in our game tree.

For edge 12, the actual automaton built in the laboratory has $i_4 \wedge i_3 \wedge \neg i_9$ instead of $i_4 \wedge i_3$. This is correct because the law $i_4 \wedge i_3 \Rightarrow \neg i_9$ holds in our game tree.

For edge 23, the actual automaton built in the laboratory has $i_6 \wedge i_7 \wedge \neg i_2$ instead of $i_6 \wedge i_7 \wedge \neg i_4$. This is correct because the law $i_6 \wedge i_7 \Rightarrow (i_4 \Leftrightarrow i_2)$ holds in our game tree.

Initial experimentation with parts of the automaton in the laboratory preceded the treatment above. In particular, gates activated by the gross error move (Figure 3B in the main text) were constructed to be deactivated by the correct move blocking the automaton's three in a row, thus ensuring minimum background cleavage. This led to the redundant not-loops in oligonucleoitdes for edges 2, 6 and 12. For edge 23 the change was empirically introduced, because the initial construct (with $\neg i_4$) led to high background cleavage in the presence of input 2. Therefore, input $\neg i_2$ was introduced instead, and background cleavage was eliminated.

The actual automaton built in the laboratory was thus:

$$o_1 = \underbrace{i_4}_{24}$$

$$o_2 = \underbrace{(i_6 \wedge i_7 \wedge \neg i_2)}_{23} \vee \underbrace{(i_7 \wedge i_9 \wedge \neg i_1)}_{17} \vee \underbrace{(i_8 \wedge i_9 \wedge \neg i_1)}_{18}$$

$$o_3 = \underbrace{(i_1 \wedge i_6)}_{22} \vee \underbrace{(i_4 \wedge i_9)}_{25}$$

$$o_4 = \underbrace{i_1}_{21}$$

$$o_5 = \underbrace{1}_{20}$$

$$o_6 = \underbrace{(i_1 \wedge i_2 \wedge \neg i_6)}_{1} \vee \underbrace{(i_1 \wedge i_3 \wedge \neg i_6)}_{2} \vee \underbrace{(i_1 \wedge i_7 \wedge \neg i_6)}_{3} \vee \underbrace{(i_1 \wedge i_8 \wedge \neg i_6)}_{4} \vee \underbrace{(i_1 \wedge i_9 \wedge \neg i_6)}_{5}$$

$$o_7 = \underbrace{(i_2 \wedge i_6 \wedge \neg i_7)}_{6} \vee \underbrace{(i_6 \wedge i_8 \wedge \neg i_7)}_{7} \vee \underbrace{(i_6 \wedge i_9 \wedge \neg i_7)}_{8 \text{ and } 16} \vee \underbrace{(i_9 \wedge i_2 \wedge \neg i_1)}_{19}$$

$$o_8 = \underbrace{i_9 \wedge i_7 \wedge \neg i_4}_{9}$$

$$o_9 = \underbrace{(i_7 \wedge i_8 \wedge \neg i_4)}_{10} \vee \underbrace{(i_4 \wedge i_2 \wedge \neg i_9)}_{11} \vee \underbrace{(i_4 \wedge i_3 \wedge \neg i_9)}_{12} \vee \underbrace{(i_4 \wedge i_6 \wedge \neg i_9)}_{13} \vee \underbrace{(i_4 \wedge i_7 \wedge \neg i_9)}_{14} \vee \underbrace{(i_4 \wedge i_8 \wedge \neg i_9)}_{15}$$