Boolean Recombinase-Based Devices

Guillaume Pérution-Kihli, Sarah Guiziou, Federico Ulliana, Michel Leclère, Jérôme Bonnet

Boolean Recombinase-based Devices

G. Pérution-Kihli[1][https://orcid.org/0000-0002-8502-2465], S. Guiziou[2,3][https://orcid.org/0000-0002-1185-5421], F. Ulliana[1][https://orcid.org/0000-0002-9192-9573], M. Leclère[1][https://orcid.org/0000-0003-0484-3964], and J. Bonnet[2][https://orcid.org/0000-0002-8420-9359]

[1] LIRMM, CNRS UMR 5506, University of Montpellier, France
[2] CBS, INSERM U154, CNRS UMR 5048, University of Montpellier, France.
[3] Department of Biology, University of Washington, Seattle, Washington 98195, USA

Abstract. This paper relates to a central problem in synthetic biology, which is that of designing *Recombinase-based biological devices* by matching a functional specification expressed as a Boolean function. This task is challenging as exploring the space of possibilities is typically unfeasible, and therefore many non-trivial design alternatives remain unexplored. Also, the issue has been so far regarded mainly from a practical perspective and is still lacking of formal foundations on which the definition of algorithms for assisting the biologists in their design tasks can be based. In this work, we present the first formal study of the problem, and give a formal semantics for a family of Recombinase-based biological devices. We then exhibit a set of semantic properties leading to the definition of *representative devices*, a notion that allows one to express infinitely large classes of design possibilities in a *finite* way. Building on this, we then provide a terminating algorithm for generating representative devices for n-input Boolean functions. An open online database of 18M design solutions for 4-inputs devices generated with our method has been released at `http://recombinator.lirmm.fr`.

1 Introduction: the Design Issue

One of the main targets of the field of synthetic biology is that of designing new biological systems (also called *devices*) by matching a functional specification with state of the art techniques in biological engineering. As a concrete example, a novel type of bacteria can be assembled for clinical diagnosis purposes to react to the presence of a combination of molecules by exhibiting a fluorescence recognizable via microscopy [9]. This is a challenging task as, when designing a new biological device, the space of possibilities rapidly becomes too large to be explored and the biological reliability of each of them impossible to be experimentally tested. This implies that biological engineering solutions often fall back to best-practices and established architecture patterns, and many non-trivial design alternatives remain unexplored. Synthetic biology has nowadays key applications in health, environment, and manufacturing, and there is therefore a huge need for tools assisting biologists in best engineering their biological devices [6].

The specification of a new biological device includes 1) its intended behaviour and 2) the biotechnologies to be used to implement it. The biologist formalizes the behaviour of a device as a function of its inputs. Intuitively, the purpose of this is to precisely describe how the device must react to the substances in the environment where it will be deployed. Many approaches are today used for synthetic biology. Among them, we mention transcriptional regulators [10], RNA molecules [13], proteins [5], and Recombinases [3]. In this work, we focus on the problem of assisting the design of biological systems whose behaviour is 1) specified as *Boolean functions* and 2) based on (single-layer single-cell) *Recombinases*.

Boolean functions affirmed as an intuitive means for biologists to precisely define the device behaviour [3]. Inputs serve to acquire the environment stimuli. A biological system with a binary output means that it emits a signal (or not) depending on the inputs. In biological terms, such a "signal" corresponds to the outcome of a well understood biological process called *gene expression*, which is the natural process by which living cells produce proteins. In other words, gene expression is the natural process exploited by biotechnologists to artificially assemble the DNA of living organisms implementing logic functions [3].

Our interest in *Recombinase-based* devices stems from their importance in biological engineering. It is worth mentioning that Recombinase-based devices are named after *Recombinases* [11], which are the enzymes used to control gene expression. This type of devices offer several design advantages. First, they can be adapted to various species of living organisms with minimal modifications [12, 2]. Then, and most importantly for the study conducted here, they are modular and compact. *Modular* means among other that the device inputs can be easily interchanged. From a biological engineering point of view, this implies that the same device architecture can be *reused* to implement different logic functions (by simply interchanging the inputs). This is reminiscent of the notion of P-class introduced for analogous problems in circuit-logic design [1]. *Compact* means that this type of devices allow the biologists to minimize the number of living cells implementing together a certain logic function, hence increasing the device reliability. It is known that any Boolean function can be implemented over multicellular [8] or multi-layer devices [4] (i.e., where distinct "parts" of the Boolean function are implemented separately). Here, we will focus on the study of the capacity of *single-layer single-cell* devices whose expressivity limits, that is, the Boolean functions they can implement in a "monolithic" way (i.e., without distributing the Boolean function in several parts), are still unknown. From a computational point of view, recombinase-based devices enable two types of operations, named *inversion* and *excision*, on the elements controlling the process of gene expression. Hence, abstracting away, biological devices can be seen as programs, written in a specific formal language allowing for inversion and excision like operations on word expressions, that implement a logic function.

Despite the importance of recombinase-based devices in synthetic biology, the development of most design solutions mainly follows best-practices and fixed architecture patterns which limit the possibilities of such type of devices. Indeed, so far mostly 2-input functions only have been considered [3] with the exception

of [12] that considered 3-input functions implemented with excisions only (that is, all designs using inversions have not been considered). Also, at the best of our knowledge, no foundational study of the properties of devices has been done.

The first goal of this paper is to present a formal study of recombinase-based devices (presented in Section 2) which can serve as foundations of the development of new biological engineering solutions for this type of devices. Precisely, we present the first formal semantics allowing one to compare them from a computational point of view (Section 3). From this, a set of minimality properties naturally emerge, which lead us to the notion of *canonical and representative* devices, by which infinitely large classes of design solutions can be expressed in a *finite* way (Section 4). The second goal of this paper is to actually compute devices that can help biologists defining their engineering solutions. We then present a terminating algorithm which allows one to generate all representative canonical devices for n-input systems (Section 5). Then, since each device has an associated Boolean function, the output of this algorithm can be used by biologist as a starting point to design new devices starting from a Boolean logic specification. An online platform for search and detailed comparison of 18M canonical and representative design solutions of up to 4-input devices generated with our method has been released (`http://recombinator.lirmm.fr`). Finally, our results also indicate some interesting expressivity limits for *single-layer single-cell* devices (Section 6). Indeed, the generation process shows that 8% among all 4-input Boolean functions cannot be implemented. We also show that they cannot implement all n-input Boolean functions, for every $n \geq 7$.

2 Recombinase-based Devices

This section presents the main biological aspects of Recombinase-based devices [11]. The notions presented here are intended to introduce the reader to the formal semantics presented next. In biological terms, a device is a DNA construct in a living cell capable of gene expression. The DNA is artificially assembled thereby controlling gene expression according to the stimuli of the environment where the cell is. Roughly speaking, the two macro steps of gene expression are called *transcription* and *translation*. First, the cell DNA is copied to RNA (*transcription*). Then, the RNA is used to manufacture proteins (*translation*). By this process, a bacteria can for instance express a protein triggering a fluorescence.

To control gene expression, the main goal is to *control the transcription phase* of the device, i.e., which segment of the DNA is transcribed in RNA. Towards this goal, the DNA sequence of a cell is assembled by concatenating DNA segments called *biological parts*. As DNA, every biological part has a forward (F) or reverse (R) orientation, and therefore the transcription of a gene (and hence gene expression) *can succeed in both directions*. As a convention, we assume the forward orientation to run from left to right, opposed to the reverse orientation.

The two basic types of biological parts are *promoters* and *terminators*, which are responsible to start and stop the transcription phase, respectively. Then, the

genes encode the information transcribed for the manufacturing of a protein at the end of the translation phase (and hence of gene expression).

P (*F-promoter*)	T (*F-terminator*)	G (*F-gene*)
d (*R-promoter*)	L (*R-terminator*)	Ɠ (*R-gene*)

Note at this point that whether a gene is transcribed from a device with only promoters, terminators, and genes, does not depend from the environment stimuli. Gene transcription directionally occurs if and only if a promoter starts a transcription (of course, without this being stopped by a terminator) that transcribes a gene. For instance, gene expression occurs for the device PLG but not for PTG. In principle, many distinct genes can be expressed by the same device. Here, we consider the case where a single gene can be expressed.

Recombinase-based devices use a third type of parts, namely the attachment *sites*, which enable the rearrangement of the DNA segments. We assume that there is a unique pair of sites for every input. Every rearrangement occurs between a pair of sites, and is triggered by an input i, so every site is labeled by its corresponding input. These rearrangements perform either the *inversion* which reverses the segment between two sites (e.g., PLG becomes ƓTd) and the *excision* which suppresses the segment between two sites. The modified DNA is likely to behave differently after the DNA rearranging. In practice, what happens is that a transcription process can be started or stopped. Also, a gene can be suppressed or transcribed thereby changing the outcome of the process.

3 Syntax and Transcriptional Semantics

This section presents a formalization of Recombinase-based devices. A Device is defined starting from an architecture and a naming of the inputs. A *biological architecture* A is an expression obtained from 1) biological parts, 2) the concatenation of expressions and finally 3) the nesting of expressions between excision and inversion marks. Two types of delimiters are marking excision and inversion operations. The excision marks are denoted by a pair of square brackets [] while the inversion marks are denoted by a pair of round brackets (). The set of architectures we consider is thus defined by the following grammar.

$$A ::= \epsilon \mid P \mid AA \mid [\ A\ ] \mid (\ A\ ) \qquad P ::= \mathtt{P} \mid \mathtt{T} \mid \mathtt{G} \mid \mathtt{d} \mid \mathtt{L} \mid \mathtt{Ɠ}$$

Here, ϵ is the empty architecture, P the set of parts, and AA the concatenation of two architectures. Note that the grammar generates only balanced parenthesized expressions, whose backbone are Dyck words with two types of delimiters. An architecture with n excisions or inversions is called a n-input architecture; 0-input architectures are special expressions also called *elementary sequences* and will be denoted by E. An architecture obtained by suppressing (at least) one biological part of an architecture A is called a *part-erasure* of A. An architecture obtained by suppressing (at least) one pair of marks denoting an excision or inversion of an architecture A is called an *input-erasure* of A. Note that architectures are closed for erasure, that is, any erasure of an architecture is still an architecture. A *device* is a couple $D = (A, \rho)$ where A is a n-input architecture

and ρ is a bijective labeling function which associates every pair of companion delimiters of A to its corresponding input, i.e. an integer in $\{1, \ldots, n\}$. For example the architecture $A = \texttt{[P]}\texttt{(G)}$ can be used to build the devices $D = \texttt{[P]}_1\texttt{(G)}_2$ and $D' = \texttt{[P]}_2\texttt{(G)}_1$. Hence, there exists $n!$ devices for a given architecture. We denote by $D_{[i]}$ the sub-device of D delimited by the pair of marks corresponding to input i. For instance, $D_{[1]} = \texttt{[P]}_1$ and $D_{[2]} = \texttt{(G)}_2$. The notions of part and input erasures are naturally extended to devices.

3.1 Device Activation and Architecture Rearrangement

The underlying architecture of a device is rearranged depending on the excisions and inversions activated by the environment stimuli. Let us denote by $\mathbf{B} = \{0, 1\}$ the Boolean domain. We define a possible configuration of the environment where a n-input device D is deployed as a tuple $\mathbf{c} \in \mathbf{B}^n$. Then, the activation function, we denote by $\mathsf{activ}(D, \mathbf{c})$ yields an elementary sequence by rearranging D as the result of successively performing all excisions and inversions according to $\mathbf{c}$. For every elementary sequence $\mathsf{activ}(E, \mathbf{c}) = E$, no rearrangement happens. Otherwise, let $\mathbf{c}[i]$ be the i-th element of the tuple $\mathbf{c}$ and D the sequence on which the activation of the i-th input is performed. If $D_{[i]}$ is an excision, i.e. $D_{[i]} = [D']_i$, the rearrangement of input i consists at substituting $D_{[i]}$ with D' if $\mathbf{c}[i] = 0$ and with ϵ if $\mathbf{c}[i] = 1$. Moreover, if $D_{[i]}$ is an inversion, i.e. $D_{[i]} = (D')_i$, the rearrangement of input i consists at substituting $D_{[i]}$ with D' if $\mathbf{c}[i] = 0$ and with $\mathsf{inverse}(D')$ if $\mathbf{c}[i] = 1$, where the $\mathsf{inverse}()$ function is defined as follows.

$$\begin{array}{cccc}
\mathsf{inverse}(\epsilon) = \epsilon & & \mathsf{inverse}(A\,A') = \mathsf{inverse}(A')\,\mathsf{inverse}(A) & \\
\mathsf{inverse}(\,[A]_i\,) = [\,\mathsf{inverse}(A)\,]_i & & \mathsf{inverse}(\,(A)_i\,) = (\,\mathsf{inverse}(A)\,)_i & \\
\mathsf{inverse}(\texttt{P}) = \texttt{d} & \mathsf{inverse}(\texttt{d}) = \texttt{P} & \mathsf{inverse}(\texttt{T}) = \texttt{L} & \mathsf{inverse}(\texttt{L}) = \texttt{T} \\
& \mathsf{inverse}(\texttt{G}) = \texttt{Ɔ} & \mathsf{inverse}(\texttt{Ɔ}) = \texttt{G} &
\end{array}$$

The result of $\mathsf{activ}(D, \mathbf{c})$ consists at performing all architecture rearrangements, in any order, for all inputs $i \in \{1, \ldots, n\}$. Note that the result of the activation is an elementary sequence where all marks have been removed. The result of the rearrangement function is unique, and does not depend on the order on which elements are activated. Finally, note that for any two devices with the same architecture produce the same set of elementary sequences.

To illustrate, $\texttt{[P(L)]G}$ is a 2-input architecture yielding two devices $D_1 = \texttt{[P(L)}_2\texttt{]}_1\texttt{G}$ and $D_2 = \texttt{[P(L)}_1\texttt{]}_2\texttt{G}$. So, $\mathsf{inverse}(D_1) = \texttt{Ɔ[(T)}_2\texttt{d]}_1$ and $\mathsf{inverse}(D_2) = \texttt{Ɔ[(T)}_1\texttt{d]}_2$. Consider now the possible input configurations. If no input is activated, then both devices transcribe the gene $\mathsf{activ}(D_1, 00) = \texttt{PLG} = \mathsf{activ}(D_2, 00)$. In all other cases no gene is transcribed as $\mathsf{activ}(D_1, 01) = \texttt{PTG} = \mathsf{activ}(D_2, 10)$, and $\mathsf{activ}(D_1, 10) = \mathsf{activ}(D_1, 11) = \texttt{G} = \mathsf{activ}(D_2, 01) = \mathsf{activ}(D_2, 11)$.

3.2 The Transcriptional Semantics of a Device

We formalize in this section the transcriptional semantics of devices, a notion which defines the computation that they perform. We model this as a transcrip-

$\mathbb{P}$	
$s_\mathbb{0}$	$s_\mathbb{P}$
$s_\mathbb{P}$	$s_\mathbb{P}$
$s_\mathbb{1}$	$s_\mathbb{1}$

$\mathbb{T}$	
$s_\mathbb{0}$	$s_\mathbb{0}$
$s_\mathbb{P}$	$s_\mathbb{0}$
$s_\mathbb{1}$	$s_\mathbb{1}$

$\mathbb{G}$	
$s_\mathbb{0}$	$s_\mathbb{0}$
$s_\mathbb{P}$	$s_\mathbb{1}$
$s_\mathbb{1}$	$s_\mathbb{1}$

$\mathbb{0}$	
$s_\mathbb{0}$	$s_\mathbb{0}$
$s_\mathbb{P}$	$s_\mathbb{P}$
$s_\mathbb{1}$	$s_\mathbb{1}$

$\mathbb{1}$	
$s_\mathbb{0}$	$s_\mathbb{1}$
$s_\mathbb{P}$	$s_\mathbb{1}$
$s_\mathbb{1}$	$s_\mathbb{1}$

$\mathbb{S}$	
$s_\mathbb{0}$	$s_\mathbb{P}$
$s_\mathbb{P}$	$s_\mathbb{1}$
$s_\mathbb{1}$	$s_\mathbb{1}$

$\circ$	$\mathbb{0}$	$\mathbb{P}$	$\mathbb{T}$	$\mathbb{G}$	$\mathbb{S}$	$\mathbb{1}$
$\mathbb{0}$	$\mathbb{0}$	$\mathbb{P}$	$\mathbb{T}$	$\mathbb{G}$	$\mathbb{S}$	$\mathbb{1}$
$\mathbb{P}$	$\mathbb{P}$	$\mathbb{P}$	$\mathbb{P}$	$\mathbb{S}$	$\mathbb{S}$	$\mathbb{1}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{G}$	$\mathbb{G}$	$\mathbb{1}$

$\circ$	$\mathbb{0}$	$\mathbb{P}$	$\mathbb{T}$	$\mathbb{G}$	$\mathbb{S}$	$\mathbb{1}$
$\mathbb{G}$	$\mathbb{G}$	$\mathbb{1}$	$\mathbb{T}$	$\mathbb{G}$	$\mathbb{1}$	$\mathbb{1}$
$\mathbb{S}$	$\mathbb{S}$	$\mathbb{1}$	$\mathbb{P}$	$\mathbb{S}$	$\mathbb{1}$	$\mathbb{1}$
$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$	$\mathbb{1}$

Fig. 1. Transcriptional function of parts (left). Neutral and composed functions (center). Composition of characteristic functions (right).

tion state function running over a finite number of states: *neutral* ($s_\mathbb{0}$), *transcription* ($s_\mathbb{P}$), and *gene transcribed* ($s_\mathbb{1}$). We denote this set of states $\{s_\mathbb{0}, s_\mathbb{P}, s_\mathbb{1}\}$ by S. In the neutral state no transcription is started and no gene is transcribed. In the transcription state no gene is transcribed but a transcription is started. The final state is reached when the gene is transcribed.

We will present first the transcriptional semantics of elementary sequences and then turn to that of general devices. We define the transcriptional semantics $\tau()$ for a sequence of parts without a specific orientation (again, we write in the forward orientation merely for readability).

$$\tau(\epsilon) = \mathbb{0} \qquad \tau(EE') = \tau(E') \circ \tau(E) \qquad \tau(\mathtt{P}) = \mathbb{P} \quad \tau(\mathtt{T}) = \mathbb{T} \quad \tau(\mathtt{G}) = \mathbb{G}$$

For each part `P`, `T`, and `G` we define a characteristic transition function $\mathbb{P}$, $\mathbb{T}$, and $\mathbb{G}$, respectively, reported in Figure 1. We now briefly comment on these. A system which is in the neutral state $s_\mathbb{0}$ moves to the transcription state $s_\mathbb{P}$ when a promoter part starts the transcription. All other parts do not modify the $s_\mathbb{0}$ state. When a system is in the transcription state $s_\mathbb{P}$ the gene part lifts it to the gene expressed state $s_\mathbb{1}$. However, a terminator part gets it back to state $s_\mathbb{0}$. Finally, the state $s_\mathbb{1}$ is the final state of the computation as no part can further impact it. Figure 1 also presents a neutral transition function $\mathbb{0}$ which behaves as the identity and models the empty architecture ϵ.

More interestingly, the concatenation of any two biological parts (in the same orientation) and thus the composition of any of the previously mentioned functions, yields only two new composite functions $\mathbb{S}$ and $\mathbb{1}$, reported in Figure 1. The function $\mathbb{1}$, denoting gene expression, results from a promoter (function $\mathbb{P}$) followed by a gene (function $\mathbb{G}$). The function $\mathbb{S}$ describes the effect of a sequence containing an expressible gene (function $\mathbb{G}$) followed by a promoter (function $\mathbb{P}$). We denote by $\mathcal{C}$ the set of *characteristic functions* $\mathbb{0}, \mathbb{P}, \mathbb{T}, \mathbb{G}, \mathbb{S}, \mathbb{1}$. To illustrate, $\tau(\mathtt{PG}) = \mathbb{1}$, $\tau(\mathtt{TGPT}) = \mathbb{T}$ and $\tau(\mathtt{GPPTGP}) = \mathbb{S}$.

Proposition 1. *Characteristic functions are closed under composition.*

An elementary sequence E with parts on both orientations can be seen as a pair of distinct elementary sequences (E_F, E_R), containing only the parts in each orientation in the order of the original sequence. For example `dTG` can be seen as the pair (`TG`,`d`). The transcriptional semantics can then be described by a function $T : P^* \to \mathcal{C} \times \mathcal{C}$ yielding a characteristic functions for each orientation. Formally, we define $T(E) = \langle\, \tau(E_F), \tau(\mathsf{inverse}(E_R)) \,\rangle$. We are ready to define the semantics of general devices, which depends on the environment configuration.

Definition 2 (Transcriptional Semantics). *Let D be a n-input device. The transcriptional semantics of D is a function $\delta_D : \mathbf{B}^n \to \mathcal{C} \times \mathcal{C}$ such that, for any configuration $\mathbf{c} \in \mathbf{B}^n$, $\delta_D(\mathbf{c}) = T(E)$ where $E = \mathsf{activ}(D, \mathbf{c})$.*

To illustrate, consider again $D_1 = [\mathtt{P}(\mathtt{L})_2]_1\mathtt{G}$. The configuration 00 gives the elementary sequence $E = \mathtt{PLG}$, thus $E_F = \mathtt{PG}$ and $E_R = \mathtt{L}$. Then $\delta_{D_1}(00) = (\mathbb{1}, \mathbb{T})$, $\delta_{D_1}(01) = (\mathbb{0}, \mathbb{0})$, and $\delta_{D_1}(10) = \delta_{D_1}(11) = (\mathbb{G}, \mathbb{0})$.

Note that with six characteristic functions, for an elementary sequence we have 36 possible bidirectional transcriptional semantics. From this it follows that the number of possible logical semantics of a n-input device is $2^n \cdot 36$.

4 The Logical Semantics of Gene Transcription

The purpose of this section is to establish a connection between devices and the Boolean functions modeling gene transcription they implement. Boolean functions are important as they constitute a starting point for the biologist willing to design a new device that implements a certain logic.

A Boolean function is a function $\varphi : \mathbf{B}^n \to \mathbf{B}$, where n denotes its arity. We associate to each device a Boolean function. The goal of a Boolean function is to model the fact that gene transcription occurs after a certain activation of the inputs. Importantly, the function should not make any distinction regarding the orientation on which gene transcription succeeds. This leads to the following definition.

Definition 3. *Let D be a n-input device. The Boolean function of gene transcription $\varphi_D : \mathbf{B}^n \to \mathbf{B}$ associated with D is such that, for any configuration $\mathbf{c} \in \mathbf{B}^n$, $\varphi_D(\mathbf{c}) = 1$ if $\delta_D(\mathbf{c}) = \langle \mathbb{1}, \mathbb{C} \rangle$ or $\delta_D(\mathbf{c}) = \langle \mathbb{C}, \mathbb{1} \rangle$ with $\mathbb{C} \in \mathcal{C}$.*

One first important observation is that two devices can be different from a transcriptional semantics point of view, yet implement the same Boolean function. To see that consider $E = \mathtt{PG}$ and $E' = \mathtt{\eth dP}$. In this case $\delta_E = (\mathbb{1}, \mathbb{0})$ and $\delta_{E'} = (\mathbb{P}, \mathbb{1})$ but $\varphi_E = 1 = \varphi_{E'}$. The same holds for the devices $D = [E]_1$ and $D' = [E']_1$ with different transcriptions but such that $\varphi_D = \varphi_{\neg i_1} = \varphi_{D'}$.[1]

We wish now to have a mean to compare devices on the basis of the Boolean functions they implement, so as to propose to the biologists a set of viable design alternatives. As already explained, by means of input permutations one has access to the implementation of different Boolean functions with the same architecture. Also, we should take into account the fact that architectures with a different number of inputs may turn out to implement the same logics. The rest of the section is dedicated to the formalization of such notions of equivalence.

The P and M classes of Boolean functions. A classical notion for Boolean functions, introduced in the context of circuit design, is that of P-class [1].

Definition 4. *Two n-input Boolean functions φ and φ' belong to the same* permutation class *(or simply P-class) if $\varphi' = \varphi \circ \rho$, where ρ is an input permutation.*

[1] φ also denotes the Boolean function associated with the logic formula built with the inputs. For instance, $\varphi_{(\neg i_1 \wedge i_2)} = \{00 \mapsto 0, 01 \mapsto 1, 10 \mapsto 0, 11 \mapsto 0\}$.

For example, the functions $\varphi_{(i_1 \wedge \neg i_2)} \neq \varphi_{(\neg i_1 \wedge i_2)}$ belong to the same P-class by the permutation that interchanges the two inputs. Therefore, the permutation class of a function $\varphi_{(A,\rho)}$ contains all Boolean functions modeling gene transcription that can be implemented by using a given architecture A and a permutation of its inputs. This is the case for instance for $D_1 = \mathtt{P[T]_1(G)_2}$ and $D_2 = \mathtt{P(T(\eth)_2)_1}$ simply because $\varphi_{D_1} = \varphi_{(i_1 \wedge \neg i_2)} = \varphi_{D_2}$. However, also $D_3 = \mathtt{[P(\eth)_2]_1}$ yields functions in the same permutation class because $\varphi_{D_3} = \varphi_{(\neg i_1 \wedge i_2)}$. Finally $D_4 = \mathtt{P\eth[TI]_1(Gd)_2}$ is also such that $\varphi_{D_4} = \varphi_{(i_1 \wedge \neg i_2)}$ but it does not uses a minimal number of part, as the gene is transcribed in both directions here, and can hence be simplified to again obtain D_1.

The notion of P-class does not allow however to compare functions that have a different number of inputs. To do so, we now define the notion of M-class, which allows one to group Boolean functions with a different number on inputs, but that are equal once they have reached their minimal form, i.e., where all redundant inputs have been removed. We formalize this as follows. We say that a n-input function φ is the jth-input minimization of a $(n+1)$-input function φ', when given the mapping removing the jth input of φ, namely $\pi_j : \mathbf{B}^{n+1} \to \mathbf{B}^n$, which is defined as $\pi_j(i_1, \ldots, i_{n+1}) = (i_1, \ldots, i_{j-1}, i_{j+1}, \ldots, i_{n+1})$ we have that $\varphi \circ \pi_j = \varphi'$. We say that φ is a *minimization* of φ' if φ is obtained as the result of a sequence of input minimizations from φ'. A function that cannot be minimized is said to be in *minimal form*. The minimal form of a Boolean function φ is unique, and we denote it by $min(\varphi)$.

Definition 5. *Two Boolean functions φ and φ' belong to the same* minimization class *(or simply M-class) if $min(\varphi) = min(\varphi')$.*

To illustrate, the function $\varphi_{(i_2 \wedge \neg i_3 \wedge (i_1 \vee \neg i_1))}$ (where the input i_1 is redundant) belongs to the same M-class as $\varphi_{(i_2 \wedge \neg i_3)}$, which is its minimal form, but not to the same P-class. The function $\varphi_{(i_2 \wedge \neg i_3 \wedge (i_1 \vee \neg i_1))}$ is implemented for instance by the 3-input device $D_5 = \mathtt{P([T]_2[G]_3)_1d}$. It turns out that D_1, D_2, D_3, D_4, D_5 are all viable design alternatives (albeit D_4, D_5 are not minimal in the number of parts and inputs and shall be avoided). We can now define equivalent devices.

Definition 6. *Two devices D and D' are equivalent for gene transcription, denoted $D \equiv_{\mathsf{GenTran}} D'$, if φ_D and $\varphi_{D'}$ belong both to the same P-class or M-class.*

Note that $\equiv_{\mathsf{GenTran}}$ is an equivalence relation that partitions every set of devices with at most n inputs in a finite number of classes. Still, every class has an infinite number of elements, which makes the space of design configurations impossible to explore for the biologist willing to compare all n-input architectures implementing the same logic. For example, there is a class which contain all sequences `PG`, `PPG`, `PPPG`, etc. Furthermore, it may be interesting to consider only devices that satisfy some minimality criteria concerning their inputs and number of parts.

The remainder of the section is dedicated to answering the following questions.

1. Is it possible to characterize the devices which are *representative of all elements* of an equivalence class defined by $\equiv_{\mathsf{GenTran}}$?

2. Are such representative device finite in number ?
3. Are all n-input Boolean functions implementable with this type of devices ?

Representative Devices We say that a device is representative if it is minimal in regard of the number of parts and inputs it uses. It is said *irreducible* if none of its parts can be erased without affecting gene expression. It is said *irredundant* when reducing its input set does affect gene expression.

Definition 7. *A device D is* representative *when the following holds.*

1. *if for any of its part-erasures D' holds $D' \not\equiv_{\mathsf{GenTran}} D$* (Irreducibility)
2. *if for any of its input-erasures D' holds $D' \not\equiv_{\mathsf{GenTran}} D$* (Irredundancy)

Note that if a device is not irreducible then there exists another equivalent one in the same P-class that uses less parts. Similarly, if a device is not irredundant then there exists another equivalent device in the same M-class that uses less inputs. It is not difficult to see that for every device D there exists an equivalent device D' which is irredundant and irreducible.

Theorem 8. *The number of representative devices in an equivalence class defined by $\equiv_{\mathsf{GenTran}}$ is finite.*

Take the minimum number of inputs n for a device in a class defined by $\equiv_{\mathsf{GenTran}}$. All devices with more than n inputs are redundant, and hence not representative. Then, a n-input device architecture is a well-parenthesized expression with n pairs of marks. The parenthesis themselves (we call architecture skeleton) constitute Dyck words with two types of delimiters of size $2 \times n$. There are $2^n \times C(n)$, where $C(n)$ is the n-th Catalan number. This means that an architecture interleaves such a word with $2 \times n + 1$ elementary sequences. One can check that there are only 53 irreducible (representative) elementary sequences, which implies that the number of representative devices is bounded by $m!$ where $m = 2^n \times C(n) \times 53^{2n+1}$. We also derive the following on the device expressivity.

Proposition 9. *There is a n-input Boolean function which is not implementable for every $n \geq 7$.*

We know that $2^n \times C(n) \times 53^{2n+1}$ is an upper bound for the number of n-input architectures yielding representative devices. Since the number of n-input Boolean functions is 2^{2^n}, and the number maximum of n-input Boolean functions belonging to a n-input P-class is bounded by $n!$ (corresponding to the number of permutations of its n inputs), then $(2^{2^n}/n!)$ is a lower bound for the number of n-input P-classes. It follows that from 7 inputs the upper bound of n-input architectures yielding representative devices is strictly lower than the number lower bound for the number of n-input P-classes.

5 Generating Canonical Representative Architectures

Theorem 8 directly allows us to define a terminating algorithm for generating all n-input representative devices. Importantly, the generation method rather works on *generating the architectures* leading to the representative devices.

1. Generate all possible architecture skeletons with n inputs
2. Fill the architecture skeletons with $2n+1$ representative elementary sequences
3. Filter the architectures that don't yield representative devices

Once an architecture is generated, the devices are obtained by input permutation. Note that the third step of the algorithm is necessary because by assembling irreducible elementary sequences we are not guaranteed to build an architecture yielding a representative device. For example, `P`, `G`, and `PG` taken alone are irreducible elementary sequences but the device $\texttt{P[PG]}_1\texttt{G}$ is not representative as equivalent to `PG`. In this respect, it is important to see that to test if all devices obtained from one architecture are representative it suffices to test only one device with that architecture. Indeed, it holds that any device D from A is representative if and only all devices obtained from A are representative.

Removing Mirror Structures An important optimization of the method we just presented accounts for the bidirectionality of architectures. Since the orientation of the architecture is merely conventional, there is no difference between an architecture and its inverse. We call a *mirror architecture* an architecture A such that $\mathsf{inverse}(A) = A$. Hence, we can optimize step (1) by discarding one among a pair of distinct mirror skeletons that have been generated. For example we keep one between the skeletons `[]()` and `()[]`. This almost drops by half the number of architectures generated. For the special case of skeletons that are mirror of themselves, like `[]`, we can then optimize step (2) by carefully avoiding to generate pair of equivalent mirror architectures like `[P]G` and `Ɒ[d]`. The correctness of this optimization is stated as follows.

Proposition 10. *Let A be an architecture and D a device obtained from A. Let D' be a device obtained from the architecture* $\mathsf{inverse}(A)$*. Then, $D \equiv_{\mathsf{GenTran}} D'$.*

Using Canonical Elementary Sequences We know that with respect to gene transcription all elementary sequences whose transcriptional semantics contains $\mathbb{1}$ in one of the two orientations are equivalent.

Definition 11. *Two elementary sequences E and E' are gene transcription equivalent, denoted $E \equiv_{\mathbb{1}} E'$, if either (i) $\varphi_E = 1 = \varphi_{E'}$ or (ii) $\delta_E = \delta_{E'}$.*

Therefore, for each equivalence class defined by $\equiv_{\mathbb{1}}$ we can consider only one irreducible elementary sequence, we call *canonical*. This optimization is key because in practice biologists prefer to work with a single canonical elementary sequence per semantics. This allows us to further limit the number of irreducible elementary sequence to consider in step (2), which drops from 53 to 26 (as all elementary sequences E such that $\varphi_E = 1$ have a unique canonical). The correctness of the optimization is now stated.

Proposition 12. *Let D be a representative device and E an irreducible elementary sequence. For any irreducible elementary sequence E' such that $E \equiv_{\mathbb{1}} E'$ and D' obtained by substituting some occurrence of E with E' in D, we have that $D \equiv_{\mathsf{GenTran}} D'$. Moreover, D' is also a representative device.*

Table 1. Generation results up to 4-inputs.

# inputs	generation time	# canonical architectures	# implemented functions vs. # possible functions
0	-	2	2 / 2 (100%)
1	3 seconds	10	2 / 2 (100%)
2	3 seconds	724	10 / 10 (100%)
3	5 minutes	96,981	218 / 218 (100%)
4	87 minutes	18,065,512	59,590 / 64,594 (92.25%)

6 Implementation and Results

We implemented our method for generating all n-input architectures without mirror structures in C++(version 17), which allows us to have an optimized implementation. Our tool has been run on a High Performance Computer (HPC) cluster to generate all representative architectures from 1 to 4-input. The HPC processor is an Intel Xeon E5-2680 v4, 2.40GHz, 14 physical cores, with 128Go RAM DDR4 2400Mhz. Our implementation features a parallelization of the generation method which exploits multi-threading to fully use the 28 logical cores of the machine.

Table 1 reports the results of the generation. The generation time (column 2) follows the number of architectures leading to canonical representative devices (column 3) which, as already outlined, have an exponential growth in the number of inputs. Recall that each architecture allows to produce $n!$ different canonical devices. The last column indicates the number of Boolean functions searchable by biologists which are implemented by *at least* one architecture, over the total number of Boolean functions without redundant inputs (those that are not in minimal form are excluded). The results for 4-inputs functions complement Proposition 9, as they show that already for $n = 4$ some Boolean functions cannot be implemented with the devices considered here, while for every $0 \leq n \leq 3$ all functions are implemented.

The Recombinator Web Platform To help biologists in their design tasks, a database containing the results of the generation have been made openly available on the *Recombinator* Web platform `http://recombinator.lirmm.fr`. The application offers an interface allowing the biologist to identify, for a given Boolean function, the set of canonical devices which implement it. The Web interface also offers a number of filters with relevant criteria for biologists that can be applied for a detailed search. For instance, it is possible to control the size of the device, as well as the number of promoters, terminators, genes, excisions, and inversions, in the sequence, and finally their relative positioning. The database has also been used to carry a statistical analysis of the relationships between devices and Boolean functions [7].

7 Conclusions and Perspectives

In this paper, we carried a formal study of a type of Recombinase-based devices an important family of biotechnologies used in synthetic biology. We presented a formal semantics for such type of devices and outlined a generation method for listing all representative and canonical n-input devices. We believe that the notions presented here can be used for further developments of design methods in synthetic biology. Our method has been implemented and the result of the generation published in a platform for detailed search of devices implementing a certain Boolean functions. Our results also show some limits in the expressivity of devices, in terms of the Boolean functions they implement. Future work will focus on the precise characterization of the (non-)implementable Boolean functions. The framework can also be extended with a probabilistic time-dependent activation of the inputs to express more complex logics within living organisms.

References

1. Astola, J., Stankovic, R.: Fundamentals of Switching Theory and Logic Design A Hands on Approach. Springer (2006)
2. Bischof, J., Maeda, R.K., Hediger, M., Karch, F., Basler, K.: An optimized transgenesis system for drosophila using germ-line-specific φc31 integrases. Proceedings of the National Academy of Sciences **104**(9), 3312–3317 (2007)
3. Bonnet, J., Yin, P., Ortiz, M.E., Subsoontorn, P., Endy, D.: Amplifying genetic logic gates. Science **340**(6132) (2013)
4. Chiu, T.Y., Jiang, J.H.R.: Logic synthesis of recombinase-based genetic circuits. Scientific reports **7**(1), 12873 (2017)
5. Dueber, J.E., Yeh, B.J., Chak, K., Lim, W.A.: Reprogramming control of an allosteric signaling switch through modular recombination. Science **301**(5641), 1904–1908 (2003)
6. Endy, D.: Foundations for engineering biology. Nature **438**(7067), 449 (2005)
7. Guiziou, S., Pérution-Kihli, G., Ulliana, F., Leclère, M., Bonnet, J.: Exploring the design space of recombinase logic circuits. bioRxiv (2019). https://doi.org/10.1101/711374
8. Guiziou, S., Ulliana, F., Moreau, V., Leclère, M., Bonnet, J.: An automated design framework for multicellular recombinase logic. ACS synthetic biology **7**(5), 1406–1412 (2018)
9. Kumari, A., Pasini, P., Daunert, S.: Detection of bacterial quorum sensing n-acyl homoserine lactones in clinical samples. Analytical and bioanalytical chemistry **391**(5), 1619–1627 (2008)
10. Nielsen, A.A., Der, B.S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E.A., Ross, D., Densmore, D., Voigt, C.A.: Genetic circuit design automation. Science **352**(6281), aac7341 (2016)
11. Wang, Y., Yau, Y.Y., Perkins-Balding, D., Thomson, J.G.: Recombinase technology: applications and possibilities. Plant cell reports **30**(3), 267–285 (2011)
12. Weinberg, B.H., Pham, N.H., Caraballo, L.D., Lozanoski, T., Engel, A., Bhatia, S., Wong, W.W.: Large-scale design of robust genetic circuits with multiple inputs and outputs for mammalian cells. Nature biotechnology **35**(5), 453 (2017)
13. Win, M.N., Smolke, C.D.: Higher-order cellular information processing with synthetic rna devices. Science **322**(5900), 456–460 (2008)