GUBS, a Behavior-based Language for Open System Dedicated to Synthetic Biology

Adrien Basso-Blandin
IBISC lab.
Evry University
abasso@ibisc.univ-evry.fr

Franck Delaplace
IBISC lab.
Evry University
franck.delaplace@ibisc.univ-evry.fr

In this article, we propose a domain specific language, GUBS (Genomic Unified Behavior Specification), dedicated to the behavioural specification of synthetic biological devices, viewed as discrete open dynamical systems. GUBS is a rule-based declarative language. By contrast to a closed system, a program is always a partial description of the behaviour of the system. The semantics of the language accounts the existence of some hidden non-specified actions that possibly alter the behaviour of the programmed devices. The compilation framework follows a scheme similar to automatic theorem proving, aiming at improving synthetic biological design safety.

1 Introduction

Synthetic biology is an emerging scientific field combining the investigative nature of biology with the constructive nature of engineering [22] to design synthetic biological systems. The issue is to devise new functionality/behaviour that does not exist in nature. Then, the field of synthetic biology is looking forward principles and tools to make the biological devices inter-operable and programmable [19]. Synthetic biology projects were first focusing on the design and the improvement of small genetic devices comparable to logical gates for electronic circuits [23, 11]. Recently, projects have attempted to develop large bio-systems integrating different devices with as a long-term goal, the design of *de-novo* synthetic genome [16]. In this endeavour, the computer-aided-design (CAD) environments play a central role by providing the required features to engineer systems: specification, analysis, and tuning [4, 20, 25, 12]. Pioneer applications show the valuable potential of such environments in IGEM competition.

Currently, the design specifies the structural assembly of DNA sequences (biobrick) as in GENOCAD [7]. Although this description is indispensable to provide a finalized specification of devices, the abstraction level seems inappropriate for tackling large bio-systems. The required size of programs for sequence description likely makes the task error-prone and un-come-at-able. In the same way as large softwares cannot be programmed in binary, large biological systems cannot be described as aDNA sequence assembly. Then, scaling up the complexity of the synthetic biological systems needs to complete the structural description by an additional abstract programming layout based on a high-level programming language and harness the automatic conversion of the design specification into a DNA sequence, like compilers. High level programming language for synthetic biology is announced as a key milestone for the second wave of synthetic biology to overcome the complexity of large synthetic system design [22]. Nonetheless, in this domain, language technology is still in its infancy and transforming this vision into a concrete reality remains a daunting challenge.

Such high-level language should describe the devices in term of functionalities, offering the ability to program the behaviour directly instead of the structure supporting this behaviour. Indeed, behaviour specification contributes to accurately document the device by adding its behavioural description, to

Gabriel Ciobanu (Ed.): Membrane Computing and
Biologically Inspired Process Calculi (MeCBIC 2012)
EPTCS 100, 2012, pp. 29–47, doi:10.4204/EPTCS.100.3

assess its functionality automatically and formally, notably by generating test-benches from this specification, and to get a relative independence to technology because different biological structures can carry out the same functionality. In this framework, the components are selected and organized automatically or semi-automatically to generate a structural description of the device at compile phase whose behaviour complies with the specified function. A such approach has been already achieved in hardware by using languages as VHDL [1] or VERILOG [24] to overcome the growing complexity of electronic circuits. However, the major difference in synthetic biology relates to the openness of biological system. Thereby, the issue is to propose a behavioural language for open systems. More precisely, GUBS is a rule-based declarative language dedicated to the behavioural specification of *discrete open dynamical systems* for synthetic biology interacting with its environment. GUBS symbolically defines the behaviours to provide a relative independence from structures by postponing the biological component selection at compile phase. Within this framework, the compiler translates the behavioural specification to a structural description of a device whose behaviour carries the functional features defined by a program. The proposed compilation method is inspired by automated theorem proving.

After introducing the main features of GUBS language (Section 2), we define the semantics of GUBS based on hybrid logic. Then, we detail the proof-based principles governing the compilation (Section 3) illustrated with a complete example (Section 4). After a survey of the related works, Section 5, we conclude (Section 6).

2 GUBS language

In this section, we describe the main features of GUBS.

Constant and variables. In GUBS, two kinds of objects are distinguished: the *constants* and the *variables*. The constants designate the pre-defined objects in a corpus of knowledge. In biology, the constants may refer to proteins or genes of interest. For example, the agent *LacZ* refers to LacZ protein or gene. By convention, their name starts with a capital letter. The variables refer to an abstraction of these predefined objects and can be potentially replaced (substituted) by any constant. By convention, the variable names start with a minuscule letter.

Agents, attributes and states. The *agents* represent the biological objects. Their different observable *states* characterize their different *behaviours*. The behaviours actually define the different capacities for actions on the state of the other agents. They are characterized symbolically by a set of *attributes* categorizing these different capacities. The real significance of the attributes is a matter of convention depending on the targeted realization (*e.g.*, protein pathways, gene network) and will be addressed through examples. For instance, the regulatory activity of a gene is observationally related to thresholds of RNA transcripts concentration. At a given threshold, a gene regulates a given set of genes whereas at another one the regulation applies to another set of genes (See Figure 1). The different thresholds define the levels of gene activities leading to different regulatory activities. For a gene G, if we identify three different kinds of regulatory activities, the state of this gene will be defined by three different attributes $\{Low, Mid, High\}$ that characterize symbolically three possible behaviours. For example, $G(Low)$ expresses the fact that agent G is in state Low and then ready for the action corresponding to this attribute. In some cases, a single state is sufficient to qualify the capacity for the action of the agent. Hence, the agent is identified to its capacity. Then, G means that agent G is available.

By contrast, $G(\overline{Low})$ signifies that the state of the agent differs from *Low* ($\overline{G}$ when an agent has a single capacity). It is worth to point out that, not being in a state defined by an attribute, does not necessarily means that the agent state is in another attribute. Indeed, for open systems the state of the agents could be of any sort that does not necessarily belong to the pre-defined attributes.

Two kinds of relations on attributes are defined: an order, $\prec$, meaning "less capacity than" and an inequality, $\not\approx$, meaning "different capacity than". Then $Low \prec Mid$ implies that the capacity for the action of *Mid* includes the capacity related to *Low*. Usually, in gene regulatory model [14], the set of genes regulated at a given level will also be regulated at a higher level. By contrast, in signalling pathways, the phosphorylation of a protein induces a conformational change of the structure leading to a specific signalling potentiality not occurring in the unphosphosrylated conformation. Assuming that *Phos* and *UnPhos* respectively represents the phosphorylated and the unphosphorylated conformations of protein P, we have $Phos \not\approx UnPhos$. Then, $P(Phos)$ implies $P(\overline{UnPhos})$ implicitly. The attributes and the relation between attributes will be declared as follows: $G :: \{Low \prec Mid, Mid \prec High\}, P :: \{Phos \not\approx UnPhos\}$. A simple set of attributes replaces the relations if unknown and no specific relation is set between attributes.

Finally, the description of the agent state is extended to a collection of agent states as follows: $g_1 + \ldots + g_n$, meaning that all the agent states, g_i, are observed concomitantly.

Trace, event, and history. A GUBS program describes a behaviour, its interpretation is based on the observations of designed systems. Then, the issue is to formalize the notion of behaviour observation. To this end, we focus on the notion of *trace* that symbolically represents the evolution of some quantities related to the agents of interest by the evolution of these agent states. A trace can be obtained from experiments by establishing a correspondence between measurements of some quantities (*e.g.*, RNA transcript concentration) and attributes of agents. Formally, a trace, $(T_t)_{1 \le t \le m}$, is a finite sequence of agent state sets where each set contains the agent states at a given instant. For instance, the evolution of a concentration evolving from *Low* to *High* for G may be described by the following trace of 6 instants: $(\underset{1}{\{G(Low)\}}, \underset{2}{\{G(Low)\}}, \underset{3}{\{G(Mid)\}}, \underset{4}{\{G(Mid)\}}, \underset{5}{\{G(Mid)\}}, \underset{6}{\{G(High)\}}), \underset{7}{}$. However, all the events in a trace are not necessarily relevant with regards to the behaviour description. For example, if we focus on the evolution from *Low* to *High* for G , only three events are relevant for the behaviour description: $G(Low), G(Mid), G(High)$; without accounting the intermediary evolution stages occurring between. Then, the behaviour recognition always emphasizes the key events in a trace entailing its contraction to show their succession. Such a contracted series is called a *consistent history* of the expected behaviour. Generally speaking, an history is related to a *chronological division* of a trace into periods where the events of a period represent all the agent states occurring at each instant. Then, an history is a sequence of these event sets. Given a trace $(T_t)_{0 \le t \le m}$, and a chronological division, $(d_i)_{1 \le i \le n}$, corresponding to a sequence of the starting dates for each period, the history is a sequence of agent states occurring in each period, $(H_i)_{1 \le i < n}$, such that each $H_i = \bigcup_{d_i \le t < d_{i+1}} T_t$. Hence, a consistent history is purposely made to point the characteristic event steps of a behaviour description out.

In the previous example, a chronological division[1] of the trace leading to an history consistent with the expected evolution from *Low* to *High* for G is $(1,3,6,7)$ which corresponds to following discrete time-intervals $([1,2],[3,5],[6,6])$. The resulting history is: $(\{G(Low)\},\{G(Mid)\},\{G(High)\})$. Notice that $(1,2,4,7)$ also fits. However, the chronological division $(1,3,7)$ leads to an inconsistent history because the level *Mid* is not seen as an intermediary event in the history (See also Figure 1 depicting the trace and consistent history of the dependences). The formal definition of the consistency in the scope of the semantics will be given in Section 2.1.

[1] Step 7 is inserted as an extra step to comply with the definition of the chronological division.

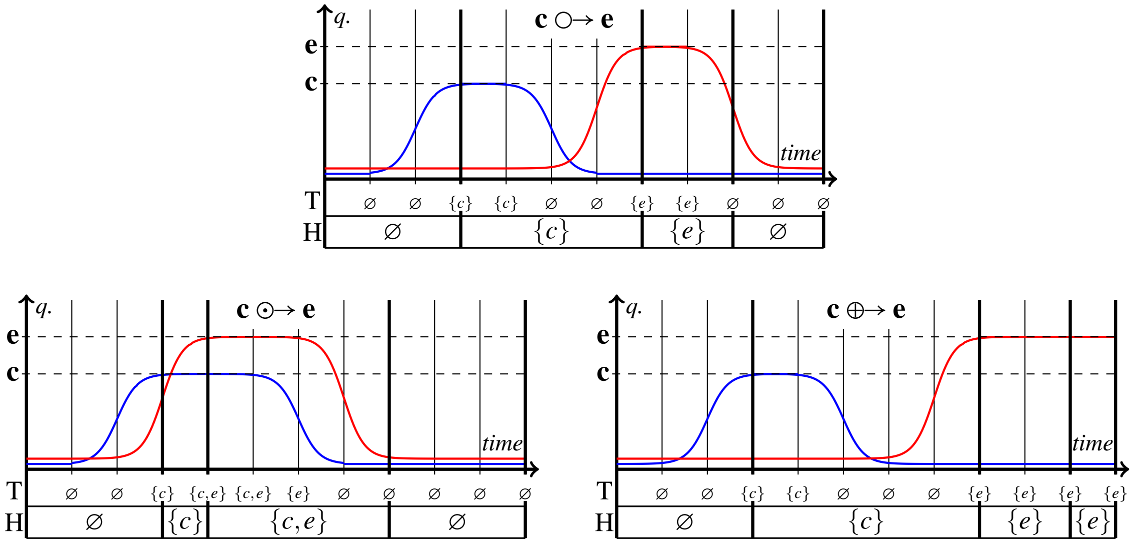

Figure 1: The curves represent the typical behaviours of the causal dependences based on the time evolution of a quantity (q) related to agents c and e (*e.g.*, RNA transcript for gene regulation). The symbolic agent states c and e are here both associated to the maximal threshold of the quantity. The symbolic trace (T) is issued from a periodic sampling of the evolution by identifying whether c or e occur. A consistent history (H) complying to a causal dependence definition is represented below the trace. The first graphic illustrates the normal causality: $c \bigcirc\!\!\!\rightarrow e$, the second the persistent: $c \odot\!\!\rightarrow e$ and the third the remanent one: $c \oplus\!\!\rightarrow e$.

Behavioral dependence and observation spot. A behavioural dependence identifies a relation between behaviours as a causal relation on events. Basically, the dependences should define the control of agents on another. However, the definition of the causality also needs to tackle the openness of a system by adapting it to this context. A seminal definition of the causality, proposed by Hume [17], is formulated in terms of regularity on events: "[we may define] a cause to be an object, followed by another, and where all the objects similar to the first are followed by objects similar to the second". Although this definition appropriately characterizes the notion of control, the openness of the system implies to account the environment actions that possibly alter the causal dependence chain. For example, a programmed activation $G_1 \xrightarrow{+} G_2$ may be contradicted by an existing inhibition $G_3 \xrightarrow{-} G_2$ addressing the same target gene G_2. Hence, while G_1 is active, it may appear that G_2 will not be active because the regulatory strength of G_3 is greater than the regulatory strength of G_1, contradicting the expected activation by a hidden inhibition. Hence, pushed to the limit, this consideration prevents the ability to describe any behaviour causally because any programmed action can be unexpectedly preempted by an external one.

However, by assuming that the design always describes a new functionality which is not observed naturally, the effect becomes the event indicating the effectiveness of a causal relation. As no cause external to the description can trigger the effect, the over-determination by unknown causes is prevented, then insuring that the program is the sole device entailing the expected effect in the biological system. Hence, the definition of the causal dependence will be governed by the effect leading to the following definition of the dependence: *"if effect e would occur then c occurs"*. Moreover, the scope of future (resp. past) is narrowed to a *closest future (resp. past) period*, representing the fact that a response is always expected in a given delay. Notice that, the proposed definition circumvents the afore mentioned problem illustrated by the hidden inhibition because if the effect does not occur the question of the existence of a cause is meaningless. This definition is somehow equivalent to the causal claims proposed by Lewis [18] in terms of counter-factual conditionals, *i.e.*, “If c had not occurred, e would not have occurred”.

Three behavioural dependences are defined in GUBS: the *normal* denoted by $\bigcirc\!\!\rightarrow$, *persistent* by $\odot\!\!\rightarrow$, and *remanent* by $\oplus\!\!\rightarrow$. Informally, for normal dependence the cause precedes the effect providing the effect is observed; for persistent dependence the cause still precedes the effect but it is maintained while the effect is observed; and for remanent dependence, the effect is maintained despite the cause has disappeared. These dependences symbolize common biological interactions. For instance, in genetic engineering, the recombination enables the emergence of a regulated gene or an hereditary trait permanently. A such mechanism typifies the remanent dependence in biology. The relations between gene expression at steady state are symbolized by persistent dependence. The behavioural dependences are defined as follows (see Section 2.1 for their formalization):

- $c \bigcirc\!\!\rightarrow e$: if e occurs then c occurs in the closest past.
- $c \odot\!\!\rightarrow e$: if e occurs then c occurs in the closest past and also currently.
- $c \oplus\!\!\rightarrow e$: if e occurs then, either e occurs in the closest past or the dependence complies to the property of the normal dependence.

Figure 1 exemplifies the correspondence between experimental traces, symbolic traces and the history for the causal dependences. All the dependences are extended to a set of causes and a set of consequences, *i.e.*, $c_1 + \ldots + c_n \bigcirc\!\!\rightarrow e_1 + \ldots + e_m$. For example, let us define the activation and the inhibition as follows: $g_1 \xrightarrow{+} g_2 \equiv g_1 \odot\!\!\rightarrow g_2, \overline{g}_1 \bigcirc\!\!\rightarrow \overline{g}_2$ and $g_1 \xrightarrow{-} g_2 \equiv \overline{g}_1 \odot\!\!\rightarrow g_2, g_1 \bigcirc\!\!\rightarrow \overline{g}_2$, the program depicting a negative regulatory circuit with two genes, *i.e.*, $g_1 \xrightarrow{+} g_2, g_2 \xrightarrow{-} g_1$, is: $\{g_1 \odot\!\!\rightarrow g_2, \overline{g}_1 \bigcirc\!\!\rightarrow \overline{g}_2, \overline{g}_2 \odot\!\!\rightarrow g_1, g_2 \bigcirc\!\!\rightarrow \overline{g}_1\}$.

The *observation spots* describe the set of observations expected in a trace. For instance, observing that gene G is at level high is written $Obs\mathbf{::}G(High)$. As the activation of a dependence lies on the observation of the effect, the observation spot is used to determine which effects must be necessarily observed. For example, in the negative regulatory circuit, the characteristic observation spots are: $obs_1\mathbf{::}g_1 + \overline{g}_2, obs_2\mathbf{::}\overline{g}_1 + g_2$.

Compartment & Context. A compartment encloses a set of dependences making them local to the compartment. For instance, $C\{g_1 \bigcirc\!\!\rightarrow g_2\}$ describes a normal dependence occurring in compartment C. The compartments are hierarchically organized and all the compartments are included in another except for the outermost one. Although the compartments directly refer to the compartmentalized cellular organization (*e.g.*, nucleus, mitochondria), they are also used to emphasize the isolation of some interactions by syntactically enclosing the dependences into a compartment. $C.s$ refers to an agent state in compartment C.

A context refers to a stimulus acting on the system, as environmental conditions or external signalling. The application of a context c to a set of dependences b is written $[c]b$ where c is either a variable or a constant. This means that dependences of b are triggered when the context c is present. For instance, recently Ye et al. [26] explore the opto-genetics signalling to control the expression of target transgenes. The blue-light induces the expression of transgene (tg) via a signalling cascade leading to the binding of NFAT transcription factor to a specific promoter (PNFAT). The following program using a context summarizes the process: $[\text{BlueLight}]\{\text{NFAT} \odot\!\!\rightarrow tg\}$. A context can be decomposed to several contexts, $[k_1, \ldots, k_n]b$, meaning that all the conditions must be met to trigger the dependences of b. The interpretation is equivalent to a context cascading, $[k_1][k_2]\ldots[k_n]b$. Moreover, the observation spots and the attribute definition are context insensitive.

2.1 Semantics of GUBS

The interpretation of GUBS is a formula such that the set of all the models validating it defines all the possible histories complying to the programmed behaviour. The interpretation is based on multi-modal hybrid logic with the "Always" operator, $\mathcal{H}(\mathbf{A}, @)$.

Hybrid logic. In what follows, we recall the formal syntax and semantics of hybrid logic. The hybdrid logic [5, 6] offers the possibility to denominate worlds by new symbols called *nominals*. They will be used in satisfaction modal operators $@_a$; the formula $@_a\phi$ asserts that ϕ is satisfied at the unique point named by the nominal a identifying a particular truth values of a formula at this point. Given a set of propositional symbol, PROP, a set of relational symbol REL, and a set of nominal NOM disjoint to PROP, a set of well formed formula in the signature of $\langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL} \rangle$ is defined as follows:

$$\phi ::= \top \mid p \mid a \mid \neg\phi \mid \phi \wedge \phi \mid @_a\phi \mid \langle k \rangle \phi \mid \langle k \rangle^{-} \phi \mid \mathbf{A}\phi.$$

with $p \in \mathsf{PROP}, a \in \mathsf{NOM}$ and $k \in \mathsf{REL}$. Moreover, the syntax is extended to other logical operators classically [2]: $\bot, \vee, \rightarrow, [k], \mathbf{E}$.

The interpretation is carried out using the Kripke model satisfaction definition (Table 2.1). $\mathcal{M}, w \Vdash \phi$ is interpreted as the satisfaction of a formula ϕ by a model $\mathcal{M}$ at world w where $\Vdash$ stands for the realizability relation (*i.e.*, "is a model of"). A model *validates* a formula, denoted by $\mathcal{M} \Vdash \phi$, if and only if it is satisfied for all the worlds of the model (*i.e.*, $\forall w \in \mathrm{Dom}\,\mathcal{M} : \mathcal{M}, w \Vdash \phi$).

Definition 1 (Kripke model)**.** *A Kripke model is a structure $\mathcal{M} = \langle W, (R_k)_{k \in \tau}, V \rangle$ where $W = Dom\,\mathcal{M}$ is a non-empty set of* worlds, *$\tau \subseteq REL$ a subset of relational symbols denoting the* modalities, *$R_k \subseteq W \times W, k \in \tau$ a relation of accessibility, $V : (PROP \cup NOM) \rightarrow \mathbf{2}^W$ an interpretation attributing to each nominal and propositional variable a set of worlds such that any nominal addresses one world at most (i.e., $\forall a \in NOM : |V(a)| \leq 1$).*

By convention, R stands for the union of the accessibility relation, $R = (\bigcup_{k \in \tau} R_k)$.

A *modal theory* of a model $\mathcal{M}$ regarding to a set of formulas F, $\mathrm{TH}_F(\mathcal{M})$, is the set of formulas of F validated by $\mathcal{M}$, *i.e.*, $\mathrm{TH}_F(\mathcal{M}) = \{\phi \in F \mid \mathcal{M} \Vdash \phi\}$. $\mathsf{KS}(\phi)$ denotes the set of all models validating ϕ, *i.e.*, $\mathsf{KS}(\phi) = \{\mathcal{M} \mid \mathcal{M} \Vdash \phi\}$.

$\mathcal{M}, w \Vdash \top$	iff *true*	$\mathcal{M}, w \Vdash @_a\phi$	iff $\exists w' \in W : \mathcal{M}, w' \Vdash \phi$ and $\{w'\} = V(a)$
$\mathcal{M}, w \Vdash a$	iff $w \in V(a)$, $a \in \mathsf{NOM} \cup \mathsf{PROP}$	$\mathcal{M}, w \Vdash \langle k \rangle \phi$	iff $\exists w' \in W : \mathcal{M}, w' \Vdash \phi$ and $w R_k w'$
$\mathcal{M}, w \Vdash \neg\phi$	iff $\mathcal{M}, w \nVdash \phi$	$\mathcal{M}, w \Vdash \langle k \rangle^{-} \phi$	iff $\exists w' \in W : \mathcal{M}, w' \Vdash \phi$ and $w' R_k w$
$\mathcal{M}, w \Vdash \phi_1 \wedge \phi_2$	iff $\mathcal{M}, w \Vdash \phi_1$ and $\mathcal{M}, w \Vdash \phi_2$	$\mathcal{M}, w \Vdash \mathbf{A}\phi$	iff $\forall w' \in W : \mathcal{M}, w' \Vdash \phi$

Table 1: Hybrid logic interpretation.

Semantics. A GUBS program is interpreted by a hybrid logic formula where the modal operators characterize here the temporal observations on an history: $[\ ]$ means *"observed in all the closest futures"* and $\langle\ \rangle$ means *"observed in a possible closest future at least"* (resp. $\langle\ \rangle^{-}, [\ ]^{-}$ for the closest past). Moreover, we assume that the accessibility relations, $(R_k)_{k \in \tau}$, are indexed by the non empty

[2] $\bot = \neg\top, \psi \vee \phi = \neg(\neg\psi \wedge \neg\phi), \psi \rightarrow \phi = \neg(\psi \wedge \neg\phi), [k]\phi = \neg\langle k \rangle \neg\phi, \mathbf{E}\phi = \neg\mathbf{A}\neg\phi.$

parts of the set of all the contexts of a program P, denoted by K_P (*i.e.*, $\tau = \mathbf{2}^{K_P} \setminus \{\varnothing\}$). Then, a non-empty set of contexts ,$\varnothing \subset K \subseteq K_P$, is a modality, *i.e.*, $\langle K\rangle, [K]$ with $\langle\ \rangle = \langle\varnothing\rangle$ by convention. Let $\langle \mathsf{W}, \bullet, \Lambda\rangle$ be the set of words W with the concatenation operation and the neutral element, the empty word Λ and $\mathsf{F}_{\mathcal{H}}$ the set of well-formed formulas of $\mathcal{H}(\mathbf{A}, @)$, the semantics is defined by four functions: $\llbracket . \rrbracket : \mathsf{P} \to \mathsf{F}_{\mathcal{H}}, \llbracket . \rrbracket_P : \mathsf{P} \to \mathsf{W} \to \mathbf{2}^{\mathsf{W}} \to \mathsf{F}_{\mathcal{H}}, \llbracket . \rrbracket_B : \mathsf{B} \to \mathsf{W} \to \mathsf{F}_{\mathcal{H}}, \llbracket . \rrbracket_R : \mathsf{R} \to \mathsf{W} \to \mathsf{F}_{\mathcal{H}}$, where $\mathsf{P}, \mathsf{B}, \mathsf{R}$ respectively stand for the set of GUBS programs, the set of agent state set and the set of relations on attributes. $\llbracket . \rrbracket$ initiates the interpretation. Table 2.1 defines these functions. For instance, the program of the negative regulatory network, $\{g_1 \odot\!\!\rightarrow g_2, \overline{g}_1 \circ\!\!\rightarrow \overline{g}_2, g_2 \odot\!\!\rightarrow \overline{g}_1, \overline{g}_2 \circ\!\!\rightarrow g_1, obs_1 :: g_1 + \overline{g}_2, obs_2 :: \overline{g}_1 + g_2\}$, is translated into the following formula:

$$\begin{array}{ll}
\llbracket \{b\} \rrbracket & = \mathbf{A}\big(\llbracket b \rrbracket_P(\Lambda)(\varnothing)\big) \\
 & \\
\llbracket \varepsilon \rrbracket_P(C)(K) & = \top \\
\llbracket b_1, b_2 \rrbracket_P(C)(K) & = \llbracket b_1 \rrbracket_P(C)(K) \wedge \llbracket b_2 \rrbracket_P(C)(K) \\
\llbracket s_1 \circ\!\!\rightarrow s_2 \rrbracket_P(C)(K) & = \llbracket s_2 \rrbracket_B(C) \to \langle K\rangle^{-}\big(\llbracket s_1 \rrbracket_B(C)\big) \\
\llbracket s_1 \odot\!\!\rightarrow s_2 \rrbracket_P(C)(K) & = \llbracket s_2 \rrbracket_B(C) \to \big(\llbracket s_1 \rrbracket_B(C) \wedge \langle K\rangle^{-}\big(\llbracket s_1 \rrbracket_B(C)\big)\big) \\
\llbracket s_1 \oplus\!\!\rightarrow s_2 \rrbracket_P(C)(K) & = \llbracket s_2 \rrbracket_B(C) \to \big((\langle\ \rangle^{-} \llbracket s_2 \rrbracket_B(C)) \vee (\langle K\rangle^{-} \llbracket s_1 \rrbracket_B(C))\big) \\
\llbracket g_1, \cdots, g_n : \{r_1, \cdots, r_m\} \rrbracket_P(C)(K) & = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} \llbracket r_j \rrbracket_R(C.g_i) \\
\llbracket l\mathbf{::}s \rrbracket_P(C)(K) & = @_l \llbracket s \rrbracket_B(C) \\
\llbracket C'\{b\} \rrbracket_P(C)(K) & = \llbracket b \rrbracket_P(C.C')(K) \\
\llbracket [K]\{b\} \rrbracket_P(C)(K') & = \llbracket b \rrbracket_P(C)(K \cup K') \\
 & \\
\llbracket s_1 + \ldots + s_n \rrbracket_B(C) & = \bigwedge_{i=1}^{n} \llbracket s_i \rrbracket_B(C) \\
\llbracket C'.s \rrbracket_B(C) & = \llbracket s \rrbracket_B(C.C') \\
\llbracket g(a) \rrbracket_B(C) & = C.g_a \\
\llbracket g(\overline{a}) \rrbracket_B(C) & = \neg C.g_a \\
\llbracket g \rrbracket_B(C) & = C.g \\
\llbracket \overline{g} \rrbracket_B(C) & = \neg C.g \\
 & \\
\llbracket a_1 \prec a_2 \rrbracket_R(g) & = g_{a_2} \to g_{a_1} \\
\llbracket a_1 \not\approx a_2 \rrbracket_R(g) & = g_{a_1} \to \neg g_{a_2} \wedge g_{a_2} \to \neg g_{a_1} \\
\llbracket a \rrbracket_R(g) & = \top
\end{array}$$

Table 2: Semantics of GUBS. In the definition, a represents an attribute, b a behaviour, g an agent, s a set of agent states or an agent state, r a relation on attributes, C a compartment, K a set of contexts and b a set of behaviours (*i.e.*, contexts, compartments, dependences, attributes, observation spots).

$$\begin{array}{ll}
\mathbf{A}(& g_2 \to ((\langle\ \rangle^{-} g_1) \wedge g_1) \wedge \neg g_2 \to (\langle\ \rangle^{-} \neg g_1) \wedge g_1 \to ((\langle\ \rangle^{-} \neg g_2) \wedge \neg g_2) \wedge \neg g_1 \to (\langle\ \rangle^{-} g_2) \wedge \\
 & @_{obs_1}(g_1 \wedge \neg g_2) \wedge @_{obs_2}(\neg g_1 \wedge g_2)
\end{array}$$

Consistent history. Now, we formally define the consistency of the history with regards to models. An history is assimilated to a path in a model ending by a world labelled with an observation spot label. The set of Kripke-models validating the interpretation of a program P, $\mathsf{KS}(\llbracket P \rrbracket)$, not only contains all the consistent histories, but also the possible histories corresponding to behavioural alterations due to external perturbations. Thus, the compilation generates a device such that all the models validating its interpretation integrate all the observations related to the program, including the consistent and the inconsistent ones.

More precisely, the consistency lies on the identification of the largest number of "relevant" events characterizing a complete causal chain described in a program. As an history is also a model, a consistent history should validate the interpretation of the complete causal chain. The dependence formula set F_P of a program P corresponds to a set of formulas where each formula is the interpretation of a dependence taken separately with the attributes related to the involved agents. By definition of the semantics, any model validating the interpretation of a program also validates each formula of this set. The consistency of an history is then based on the validated formulas of this set by this history. *An history $\mathcal{M}_H$ is consistent for P if and only if no other modal theory of histories based on F_P (i.e., $TH_{F_P}(\mathcal{M})$ with $\mathcal{M}$ as an history), ending with the same labelled world includes the modal theory of this history (i.e., $TH_{F_P}(\mathcal{M}_H) \nsubseteq TH_{F_P}(\mathcal{M})$).*

3 Compilation

At compile phase, a program is transformed to a structure (*e.g.*, a DNA sequence) while inserted in a vector cell, should behave according to the programmed specification. The structure will result to an assembly of several devices stored in a library of components (*e.g.*, parts registry). As the design relates here to a behavioural/functional description, we need to bridge the gap between structural and functional description. This stage is called the*functional synthesis*. The issue is to select a set of components whose assembly preserves the behaviour of the program. To achieve this goal, a GUBS program is associated to each component to describe its behaviour. Thereby, the component assembly corresponds to a program assembly preserving the behaviour of the compiled program. Preserving a behaviour is laid on a property called the *behavioural inclusion* formalizing the fact that the characteristic observational traits of the specified function must be recognized in traces related to the device experiments. In other words, we can exhibit histories consistent with the programmed behaviour from histories consistent with the device behaviour description. The behavioural inclusion is defined from the interpretation of the programs, as a logical consequence (Definition 2).

Definition 2 (Behavioral inclusion)**.** *A program Q* behaviourally includes *another program P, if and only if the interpretation of the latter is a logical consequence of the interpretation of the former:*

$$P \sqsubseteq Q \triangleq \forall \mathcal{M} : \mathcal{M} \Vdash [\![Q]\!] \implies \mathcal{M} \Vdash [\![P]\!] .$$

The behavioural inclusion is a pre-order[3] such that the empty program, denoted by ε, is a minimum, meaning that a program with no behaviour can be observed in all traces. And a program whose interpretation equals $\bot$, is a maximum. Figure 2 illustrates the behavioural inclusion on a particular model.

Observability. It may arise that no history will be consistent with a programmed behaviour. For example, the program $\{Obs :: g, \overline{g} \odot\!\!\rightarrow g\}$ is not observable in a trace. Indeed, its interpretation yields to the following formula: $\mathbf{A}((@_{Obs} g) \wedge (g \rightarrow ((\langle\ \rangle^{-} \neg g) \wedge \neg g)))$, false in all models because world *Obs* must both satisfies g and $\neg g$ by definition of the persistent dependence. A GUBS program is said *observable* if and only if the formula resulting from its interpretation is validated by one model at least. Hence, the interpretation of an unobservable program is an antilogy. An unobservable program can be assimilated to a programming error. The detection of such errors can be carried out at compile-phase by using tableaux method [9] that automatically determines whether a formula is satisfiable in a model. Indeed GUBS uses fragment of *HL(@)* logic which is decidable. Notice that an observable program always behaviourally includes an observable program (Proposition 1).

[3] A reflexive and transitive relation.

P	**Q**
$\{g_1 \circ\!\!\rightarrow g_3,$ $[k_3]\{g_3 \circ\!\!\rightarrow g_4\},$ $[k_4]\{g_3 \circ\!\!\rightarrow g_5\},$ $[k_6]\{g_8 \circ\!\!\rightarrow g_9\},$ $[k_7]\{g_8 \circ\!\!\rightarrow g_{10}\},$ $g_9 \circ\!\!\rightarrow g_{11},$ $g_{10} \circ\!\!\rightarrow g_{11},$ $a :: g_4\}$	$\{[k_1]\{g_0 \circ\!\!\rightarrow g_1\},$ $[k_2]\{g_0 \circ\!\!\rightarrow g_2\},$ $g_1 \circ\!\!\rightarrow g_3,$ $g_2 \circ\!\!\rightarrow g_5,$ $[k_3]\{g_3 \circ\!\!\rightarrow g_4\},$ $[k_4]\{g_3 \circ\!\!\rightarrow g_5\},$ $g_6 \circ\!\!\rightarrow g_8,$ $[k_5]\{g_6 \circ\!\!\rightarrow g_7\},$ $[k_6]\{g_8 \circ\!\!\rightarrow g_9\},$ $[k_7]\{g_8 \circ\!\!\rightarrow g_{10}\},$ $g_9 \circ\!\!\rightarrow g_{11},$ $g_{10} \circ\!\!\rightarrow g_{11},$ $a :: g_4, b :: g_5, c : g_{11}\}$

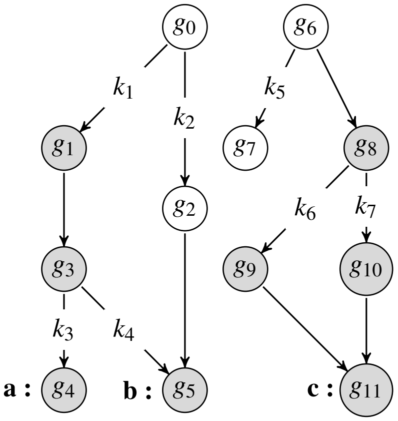

Figure 2: Behavioral inclusion example. Consistent histories of P necessary contains worlds coloured in gray.

Proposition 1. *A program behaviourally included in an observable program is observable:* $\forall P, Q \in P$: $\boldsymbol{obs}\, Q \wedge P \sqsubseteq Q \implies \boldsymbol{obs}\, P$.

3.1 Functional synthesis

The functional synthesis is the operation whereby biological components of a library are selected and assembled to generate a device behaviourally including the designed function. The behaviour of each component is described by a GUBS program. At its simplest, the functional synthesis could be considered as a proper substitution of variables by constants. For example, in the following activation $\{G_1 \xrightarrow{+} g_2\}$, g_2 will be substituted by gene G_2, providing that component Q describes the activation $\{G_1 \xrightarrow{+} G_2\}$. However, more complex situations may arise during component selection. For example, if the activation $G_1 \xrightarrow{+} G_2$ occurs with another regulation only *i.e.*, $Q = \{G_1 \xrightarrow{+} G_2, G_3 \xrightarrow{+} G_4\}$ then the selection of Q adds a supplementary regulation.

Formally, a finite substitution is a set of mappings, $\sigma = \{v_i/b_i\}_i$, on variables and constants such that a variable can be substituted by a variable or a constant, and a constant can only substituted by itself[4]. For instance, we have: $\{Obs{::}G(l) + b_2, b_1 \circ\!\!\rightarrow G(l)\}[\{b_1 \mapsto B_1, b_2 \mapsto B_2, l \mapsto Low\}] = \{Obs{::}G(Low) + B_2, B_1 \circ\!\!\rightarrow G(Low)\}$.

Functional synthesis rules. The functional synthesis is defined by rules (Table 3) governing the component assembly. Only the dependences and the attributes will be functionally synthesize. The observation spots are considered as annotations used for the compilation process. To insure the correctness, each transform must preserved the seminal behaviour. Hence, each program resulting from the application of a rule must behaviourally includes the previous one. Formally, the functional synthesis is modelled by a relation on programs denoted by $\mapsfrom$, *i.e.*, $Q \mapsfrom_\sigma P$ where P is the initial program and Q the transformed one, such that each rule insures that: *$Q \mapsfrom_\sigma P$ is correct with regards to a substitution σ, that is $P[\sigma] \sqsubseteq Q[\sigma]$ and $Q[\sigma]$ is observable.* Also notice that the behavioural inclusion is preserved by substitution (Proposition 2).

Proposition 2. *For all substitutions σ, we have: $P \sqsubseteq Q \implies P[\sigma] \sqsubseteq Q[\sigma]$.*

[4] $P\sigma$ or $P[\sigma]$ represents its application on program P and identity substitutions are omitted.

Table 3 describes the functional synthesis rules[5]. Γ is a set of components representing the library. $P \subseteq_{\text{Asm}} Q$ denotes the fact that program Q corresponds to an assembly including P *i.e.*, $Q = (Q_1, P, Q_2)$ where Q_1 or Q_2 may be an empty program. Rule (Inst.) describes the fact that an observable instance of a part of a component in the library is functionally synthesized. Rule (Com.) expresses the commutativity of the assembly. Rule (Cont.) contracts the redundant formulation of programs. Finally, Rule (Asm.) details the conditions for an assembly of two components, each representing a functional synthesis of a part of the designed function. A detailed example of their use on a real case is given in Section 4.

- INSTANTIATION -

$$\frac{Q[\sigma] \subseteq_{\text{Asm}} P[\sigma] \qquad \mathbf{obs}(Q[\sigma]) \qquad Q \in \Gamma}{Q \hookleftarrow_{\sigma} P} \text{ (Inst.)}$$

- COMMUTATIVITY, CONTRACTION -

$$\frac{Q \hookleftarrow_{\sigma} P, P'}{Q \hookleftarrow_{\sigma} P', P} \text{ (Com.)} \qquad\qquad \frac{Q \hookleftarrow_{\sigma} P}{Q \hookleftarrow_{\sigma} P, P} \text{ (Cont.)}$$

- ASSEMBLY -

$$\frac{Q \hookleftarrow_{\sigma} P \qquad Q' \hookleftarrow_{\sigma'} P' \qquad \sigma|_{\text{VA}(P)\cap\text{VA}(P')} = \sigma'|_{\text{VA}(P)\cap\text{VA}(P')} \qquad \mathbf{obs}(Q[\sigma], Q'[\sigma'])}{Q, Q' \hookleftarrow_{\sigma\cup\sigma'} P, P'} \text{ (Asm.)}$$

Table 3: Functional synthesis rules

Theorem 1. *The functional synthesis rules (Table 3) are correct.*

- DEPENDENCES -

$$\frac{Q \hookleftarrow_{\sigma} S_1 \odot\!\!\rightarrow S_2, S_2 \odot\!\!\rightarrow S_3, \Delta}{Q \hookleftarrow_{\sigma} S_1 \odot\!\!\rightarrow S_3, \Delta} \text{ (Trans.)} \qquad \frac{Q \hookleftarrow_{\sigma} S_1 \odot\!\!\rightarrow S_2, \Delta}{Q \hookleftarrow_{\sigma} S_1 \circ\!\!\rightarrow S_2, \Delta} \text{ (N2P.)} \qquad \frac{Q \hookleftarrow_{\sigma} S_1 \circ\!\!\rightarrow S_2, \Delta}{Q \hookleftarrow_{\sigma} S_1 \oplus\!\!\rightarrow S_2, \Delta} \text{ (R2N.)}$$

- AGENT STATES -

$$\frac{S_1 + S_2}{S_2 + S_1} \text{ (SCom.)} \qquad \frac{S + s}{S + s + s} \text{ (SCont.)} \qquad \frac{S + s}{S} \text{ (Incl.)}$$

Table 4: Rules for the dependences and the agent states. S_i stands for a collection, $s_1 + \ldots + s_n$, of agent states, including negation, and Δ stands for the rest of the program.

Another set of rules, more specifically devoted to dependences (Table 4), defines the alternate possibilities to express similar behaviours. The table also includes the rules for agent sets. Rule (Trans.) expands the chain of the persistent dependences by adding intermediary dependence to refine a pathway. Rule (N2P.) transforms a normal dependence to a persistent one since the latter is a normal dependence with an additional property. And Rule (R2N.) transforms a remanent dependence to a normal dependence, since normal dependence is also remanent dependence with a repetition of the effect restricted to one step. According to these rules, all the dependence chains can be implemented with persistent dependences.

A possible algorithm for the assembly could be based on a combinatorial application of the rules. However, such algorithm may reveal inefficient in practice. The conditions for an efficient algorithm of compilation should be based on an internal representation of a program, as a set of contextualized dependences with attributes, $\{\{A, [K] S_1 \circledast\!\!\rightarrow S_2\}\}$, such that A, K, S_1, S_2 are respectively: a set of attributes

[5] Rules are of the form: $\dfrac{\text{hypothesis}}{\text{conclusion}}$.

specification related to the agent involved in the dependency, a set of contexts and sets of agent states. Any program can be encoded under this representation from a normal form of the program (not detailed here). Accordingly, the problem solved by the compilation algorithm can be defined as follows (Definition 3):

Definition 3 (Functional Synthesis Problem)**.** *Let* $\Gamma = \{Q_i\}_{1\leq i\leq n}$ *be set where each* Q_i *is a set of contextualized dependences with attributes and* P *a set of contextualized dependences with attribute, can we find the smallest observable subset of components* $C \subseteq \Gamma$*, such that there exists a substitution* σ *so that its application on the components of* C *form a cover of* $P[\sigma]$*,i.e.,* $\exists\sigma : P[\sigma] \subseteq \bigcup_{Q_j\in C} Q_j[\sigma] \wedge \boldsymbol{obs}\, C$.

As the set cover problem is reducible to this problem, the problem is NP-complete. Then, the resolution is oriented towards a heuristic algorithm.

4 Example

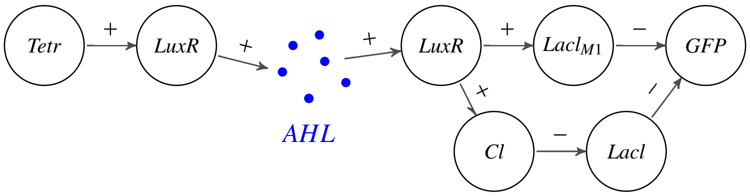

Figure 3: The band detector regulatory circuit.

The compilation process is here exemplified in a real case by the design of the Band Detector proposed in [2]. This example explains how from a simple abstract definition of the functionality a complex design can be synthesized. Accordingly, GUBS may be used to describe a behaviour with a high-level of programming well as a low-level, detailing the components involved in the process. Although, the functional synthesis is not yet performed automatically, it is worth to point out that the different transforms of the high-level program to obtain the final design complies to rules of Tables 3, 4, insuring its correctness and so, its functional safety in the context of open system.

The design aims at forming patterns of different colours in a population of bacteria exploiting the quorum sensing phenomenon by staining with fluorescent protein (GFP). The amount of molecules of interest that receives a cell depends on its relative position to the cell diffusing the molecule of interest controlled by an external event: more the cell is far from the source, the fewer is the amount of molecules received. The activation or inhibition of the fluorescent protein due to the concentration will distinguish the bands surrounding the source. In the original design, the protein does not fluoresce in an intermediary band.

From a computing standpoint, we can assimilate the design to a message transmission coupled to a sensor/actuator responsible for fluorescence, then leading to a concise GUBS program presented below: the diffusive molecule is *AHL* which production is controlled by a context and the observation is applied on *GFP*. Two categories of cells are defined: the *Sender* and the *Receiver*. Therefore, two GUBS programs identify the two cell types.

$$\begin{aligned}
&\mathit{Sender} \;\;=\{\ \mathsf{AHL}{:}\{\mathit{low} \not\approx \mathit{mid} \not\approx \mathit{high}\},[\mathit{Light}]\{\mathit{detect} \mathrel{\circ\!\!\rightarrow} \mathsf{AHL}(\mathit{low}),\mathit{detect} \mathrel{\circ\!\!\rightarrow} \mathsf{AHL}(\mathit{mid}),\mathit{detect} \mathrel{\circ\!\!\rightarrow} \mathsf{AHL}(\mathit{high})\}\}\\
&\mathit{Receiver}{=}\{\ \mathsf{AHL}(\mathit{low}) \mathrel{\circ\!\!\rightarrow} \overline{\mathsf{GFP}},\mathsf{AHL}(\mathit{mid}) \mathrel{\circ\!\!\rightarrow} \mathsf{GFP},\mathsf{AHL}(\mathit{high}) \mathrel{\circ\!\!\rightarrow} \overline{\mathsf{GFP}},\mathit{obs}_1{::}\mathsf{GFP},\mathit{obs}_2{::}\overline{\mathsf{GFP}}\}
\end{aligned}$$

Figure 3 describes the original genetic circuit used in the article. The diffusible molecule is the constant *AHL*. The gene *LuxR* has three activation thresholds: at Level 2, it activates both *LaclM1* and *Cl*, at level 1, the amount of *AHL* only allows activation of *Cl*, and finally, at level 0, none are activated. We show that from the sender-receiver program, we obtain the original design by applying the aforementioned rules with an appropriate selection of components. The regulations of Figure 3 are described in GUBS program (Table 5) translating in term of dependences and relations on their attributes their regulatory action. We focus here on some illustrative steps of the sender program compilation. The complete functional synthesis is given in Appendix. The compilation consists in finding the appropriate components whose assembly behaviourally includes the sender-receiver program, with the particularity that the diffusive molecule must be the same in both programs. To ease compilation follow-up, we label each dependency of the sender-receiver program (Table 6). Let us consider P_{11} whose compilation is closed to P_{12} and P_{13}. Notice that P_{11} cannot be directly instantiated with any component because, in the one hand, the component Q_1 contains a context like P_{11} but applied on gene *Tetr* instead of *AHL*, and on the other hand Q_3 has the *AHL* molecule but no context is defined. So, to fit P_{11} with the components Q_1, Q_2 and Q_3, first, the normal dependence is converted to persistent one (Rule (N2P.)).

$$
\begin{aligned}
&Q_1 = \{[\text{Light}]\{detect \circ\!\!\rightarrow \mathsf{Tetr}\}\}\\
&Q_2 = \{\mathsf{Tetr} \xrightarrow{+} \mathsf{LuxI}\}\\
&Q_3 = \{\mathsf{AHL}{:}\{low \not\approx mid \not\approx high\}, \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(low), \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(mid), \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(high)\}\\
&Q_4 = \{\mathsf{AHL}{:}\{low \not\approx mid \not\approx high\}, \mathsf{LuxR}{:}\{low \not\approx \{mid \prec high\}\}, \mathsf{AHL}(mid) \circ\!\!\rightarrow \mathsf{LuxR}(mid), \mathsf{AHL}(high) \circ\!\!\rightarrow \mathsf{LuxR}(high)\}\\
&Q_5 = \{\mathsf{LuxR}{:}\{low \not\approx \{mid \prec high\}\}, \mathsf{LuxR}(mid) \xrightarrow{+} \mathsf{CI}, \mathsf{LuxR}(high) \xrightarrow{+} \mathsf{CI} + \mathsf{LacIM1}\}\\
&Q_6 = \{\mathsf{CI} \xrightarrow{-} \mathsf{LacI}\}\\
&Q_7 = \{\mathsf{LacIM1} \xrightarrow{-} \mathsf{GFP}\}\\
&Q_8 = \{\mathsf{LacI} \xrightarrow{-} \mathsf{GFP}\}
\end{aligned}
$$

Table 5: Part of the database dedicated to the Band Detector.

$$
\frac{Q_1, Q_2, Q_3 \hookleftarrow_\sigma \{[light]\{detect \odot\!\!\rightarrow AHL(low)\}\}}{Q_1, Q_2, Q_3 \hookleftarrow_\sigma P_{11}}\ \text{(N2P.)}
$$

Thereby, the resulting dependence can be separated to match the assembly Q_1, Q_2, Q_3 by applying (Trans.) rule twice. v_1 and v_2 are fresh variables.

$$
\frac{\dfrac{Q_1, Q_2, Q_3 \hookleftarrow_\sigma P'_{11} = \{[light]\{detect \odot\!\!\rightarrow v_2, v_2 \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow AHL(low)\}}{Q_1, Q_2, Q_3 \hookleftarrow_\sigma [light]\{detect \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow AHL(low)\}}\ \text{(Trans.)}}{Q_1, Q_2, Q_3 \hookleftarrow_\sigma [light]\{detect \odot\!\!\rightarrow AHL(low)\}}\ \text{(Trans.)}
$$

Finally, we obtain a new program program P'_{11} compatible with Q_1, Q_2, Q_3, and each variable is substituted by a constant (biological element) with the application of Rule (Inst.). For P'_{11} we have:

$$
\frac{Q_1, Q_2, Q_3[\sigma = \{light/Light, v_1/Tetr, v_2/LuxI\}] \subseteq_{Asm} P'_{11}[\sigma] \qquad obs(Q_1, Q_2, Q_3[\sigma])}{Q_1, Q_2, Q_3 \hookleftarrow_\sigma [light]\{detect \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow v_2, v_2 \odot\!\!\rightarrow AHL(low)\}}\ \text{(Inst.)}
$$

By following this scheme for P_{12} and P_{13}, we respectively obtain P'_{12} and P'_{13}. The final assembly corresponds to the functional synthesis of *Sender* program.

$$
\frac{\begin{matrix} Q_1, Q_2, Q_3 \hookleftarrow_\sigma P'_{11} \\ \vdots \\ Q_1, Q_2, Q_3 \hookleftarrow_\sigma P_{11} \end{matrix} \qquad \begin{matrix} Q_1, Q_2, Q_3 \hookleftarrow_\sigma P'_{12} \\ \vdots \\ Q_1, Q_2, Q_3 \hookleftarrow_{\sigma'} P_{12} \end{matrix} \qquad \begin{matrix} Q_1, Q_2, Q_3 \hookleftarrow_\sigma P'_{13} \\ \vdots \\ Q_1, Q_2, Q_3 \hookleftarrow_{\sigma''} P_{13} \end{matrix}}{Q_1, Q_2, Q_3 \hookleftarrow_{\sigma \cup \sigma' \cup \sigma''} P_{11}, P_{12}, P_{13}}\ \text{(Asm.)}
$$

Sender	*Receiver*
$P_{11} = \{[Light]\{detect \circ\!\!\!\rightarrow \mathsf{AHL}(low)\}\}$	$P_{21} = \{\mathsf{AHL}(low) \circ\!\!\!\rightarrow \overline{\mathsf{GFP}}\}$
$P_{12} = \{[Light]\{detect \circ\!\!\!\rightarrow \mathsf{AHL}(mid)\}\}$	$P_{22} = \{\mathsf{AHL}(mid) \circ\!\!\!\rightarrow \mathsf{GFP}\}$
$P_{13} = \{[Light]\{detect \circ\!\!\!\rightarrow \mathsf{AHL}(high)\}\}$	$P_{23} = \{\mathsf{AHL}(high) \circ\!\!\!\rightarrow \overline{\mathsf{GFP}}\}$

with $\{\mathsf{AHL}{:}\{low \not\approx mid \not\approx high\}\}$ as attributes of *AHL*.

Table 6: Separation of the dependences.

In conclusion, the functional synthesis generates the original genetic circuit (Figure 3) from the sender program. A similar approach can be also applied to obtain the receiver program (see the complete proof in Appendix 6).

$$\begin{aligned} Sender \quad = \quad & \{\mathsf{AHL}{:}\{low \not\approx mid \not\approx high\}, [\mathrm{Light}]\{detect \circ\!\!\!\rightarrow \mathsf{Tetr}\}, \\ & \mathsf{Tetr} \xrightarrow{+} \mathsf{LuxI}, \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(low), \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(mid), \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(high)\} \end{aligned}$$

5 Related works

Several domain specific languages have been developped to model and simulate biological systems. Based on process-calculus, seminally used to model process concurrency, several rule-based languages model protein interactions [21, 13, 10]. Another approach is based on logic, such as BIOCHAM [8] that formalizes the temporal properties of a biological system. As these languages are dedicated to simulation, the objective is to close the systems because the simulations need to integrate all the characteristics of the analysed systems. By comparison, the purpose of GUBS is different since the issue is to represent the behaviour of a synthetic device in an organism, leading to translate the notion of the openness of biological systems by the semantics of the language.

In synthetic biology, the structural description languages [12, 20, 4] allow to specify well-formed genome sequences by grammars modularly and hierarchically. Although the sequence description is necessary, the programmer must previously anticipate the behaviour of the device to conceive. Besides, the behavioural design is not included in the program while it initially motivates it. In GUBS, the design is driven by a behaviour description and sequence selection is postponed at compile phase. Moreover, the size of the structural description is also subject to a combinatorial explosion when the complexity of programmed systems increases.

Amorphous programming language has been also investigated to specify the biological devices at the scale of cell colony, here considered as a possible computing medium for amorphous program. J. Beal [3] demonstrates the proof of concept of this approach in PROTO, showing the feasibility of an automatic compile chain. In GUBS, the compile chain is based on rewriting rules whose correctness have been formally proved with regards to a semantics describing the constraints of an open system.

Developing a language for biological systems actually involves to consider several unknown due to their openness: lack of knowledge on all the interactions in biological circuits and imprecise definition of initial conditions. We only know the result of a chain of effects. Then, the major constraint for programming open system seems to be: how to provide an expressive language to describe the dynamics of such systems, but simple enough to capture the essence of the biological questions in a small program in order to allow programming of large biological systems with a program humanly achievable.

In the future, the design in synthetic biology will certainly require different programming layouts based on different paradigms addressing the integration levels of biological systems. In a tower of languages, starting from a language with collective operations on cell colony, using an amorphous programming language as Proto [3] or a language for dynamical systems with dynamical structures as MGS [15], and ending by a structural description programmed in a grammar based language, GUBS language occupies the intermediary level dedicated to cell entity behavioural programming.

6 Conclusion

In GUBS language, we propose to characterize a programming paradigm abstracting the molecular interactions in the context of open system, that differs to an approach dedicated to biological system modeling. Accordingly, the interactions are symbolized by causal dependences whose interpretation is driven by effect. We have demonstrated the proof-of-concept of the compilation based on rewriting rules, and illustrated it on a realistic example. The perspective of this work is to find an efficient compilation algorithm. Identifying the biological parameters guiding the component selection should be a key issue in this undertaking.

Acknowledgements. The funding for most of this work is granted by the ANR SYNBIOTIC (ANR BLAN 0307 01) and we would like to thank the colleagues of this project for their fruitful discussions.

References

[1] PJ Ashenden (2008): *The Designer's Guide to VHDL.* Morgan Kaufmann Publishers.

[2] S. Basu, Y. Gerchman, C. H Collins, F. H Arnold & R. Weiss (2005): *A Synthetic Multicellular System for Programmed Pattern Formation.* Nature 434(7037), pp. 1130–4, doi:10.1038/nature03461.

[3] J. Beal, T. Lu & R. Weiss (2011): *Automatic Compilation from High-Level Biologically-Oriented Programming Language to Genetic Regulatory Networks.* PLoS ONE 6(8), p. e22490, doi:10.1371/journal.pone.0022490.

[4] L. Bilitchenko, A. Liu, S. Cheung, E. Weeding, B. Xia, M. Leguia, J C. Anderson & D. Densmore (2011): *Eugene–A Domain Specific Language for Specifying and Constraining Synthetic Biological Parts, Devices, and Systems.* PloS one 6(4), p. e18882, doi:10.1371/journal.pone.0018882.

[5] P. Blackburn, J. F. A. K. van Benthem & F. Wolter (2006): *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning).* Elsevier Science Inc., doi:10.1016/S1570-2464(07)80017-6.

[6] T. Braüner (2010): *Hybrid Logic and Its Proof-Theory.* Springer, doi:10.1007/978-94-007-0002-4.

[7] Y. Cai, M. W Lux, L. Adam & J. Peccoud (2009): *Modeling Structure-function Relationships in Synthetic DNA Sequences Using Attribute Grammars.* PLoS Computational Biology 5(10), doi:10.1371/journal.pcbi.1000529.

[8] L. Calzone, F. Fages & S. Soliman (2006): *BIOCHAM: An Environment for Modeling Biological Systems and Formalizing Experimental Knowledge.* Bioinformatics (Oxford, England) 22(14), pp. 1805–7, doi:10.1093/bioinformatics/btl172.

[9] S. Cerrito & M. C. Mayer (2011): *A Tableaux Based Decision Procedure for a Broad Class of Hybrid Formulae with Binders.* In: Proceedings of the 20th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX'11, Springer-Verlag, pp. 104–118, doi:10.1007/978-3-642-22119-4_10.

[10] F. Ciocchetta & J. Hillston (2009): *Bio-PEPA: A Framework for the Modelling and Analysis of Biological Systems.* Theoretical Computer Science 410(33-34), pp. 3065–3084, doi:10.1016/j.tcs.2009.02.037.

[11] K. Clancy & C. A Voigt (2010): *Programming Cells: Towards an Automated Genetic Compiler.* Current Opinion in Biotechnology 21(4), pp. 581–572, doi:10.1016/j.copbio.2010.07.005.

[12] M. J Czar, Y. Cai & J. Peccoud (2009): *Writing DNA with GenoCAD. Nucleic Acids Research* 37(Web Server issue), pp. W40–7, doi:10.1093/nar/gkp361.

[13] V. Danos, J. Feret, W. Fontana, R. Harmer & J. Krivine (2007): *Rule-Based Modelling of Cellular Signalling*. In: *CONCUR*, pp. 17–41, doi:10.1007/978-3-540-74407-8_3.

[14] F. Delaplace, H. Klaudel & A. Cartier-Michaud (2010): *Discrete Causal ModelView of Biological Networks*. In: *Proceedings of the 8th International Conference on Computational Methods in Systems Biology - CMSB '10*, ACM Press, New York, New York, USA, pp. 4–13, doi:10.1145/1839764.1839767.

[15] J.L. Giavitto, O. Michel, J. Cohen & A. Spicher (2005): *Computations in Space and Space in Computations*. In: *Unconventional Programming Paradigms*, *Lecture Notes in Computer Science* 3566, Springer Berlin / Heidelberg, pp. 97–97, doi:10.1007/11527800_11.

[16] D.G. Gibson, J.I. Glass, C. Lartigue, V.N. Noskov, R.Y. Chuang, M.A. Algire, G.A. Benders, M.G. Montague, L. Ma, M.M. Moodie & Others (2010): *Creation of a Bacterial Cell Controlled by a Chemically Synthesized Genome*. *Science* 329(5987), p. 52, doi:10.1126/science.1190719.

[17] D. Hume (1739): *A Treatise of Human Nature, Being an Attempt to Introduce the Experimental Method of Reasoning into Moral Subjects*. unknow, doi:10.1037/12868-000.

[18] D. Lewis (2000): *Causation as Influence*. *The Journal of Philosophy* 97(4), pp. 182–197, doi:10.2307/2678389.

[19] T. K Lu, A. S Khalil & J. J Collins (2009): *Next-generation Synthetic Gene Networks*. *Nature Biotechnology* 27(12), pp. 1139—-1150, doi:10.1038/nbt.1591.

[20] M. P. Pedersen (2009): *Towards Programming Languages for Genetic Engineering of Living Cells*. *Journal of the Royal Society, Interface* 6 Suppl 4, pp. S437–450, doi:10.1098/rsif.2008.0516.focus.

[21] C. Priami, A. Regev, E. Shapiro & W. Silverman (2001): *Application of a Stochastic Name-passing Calculus to Representation and Simulation of Molecular Processes*. *Information Processing Letters* 80(1), pp. 25–31, doi:10.1016/S0020-0190(01)00214-9.

[22] P. E M Purnick & R. Weiss (2009): *The Second Wave of Synthetic Biology: From Modules to Systems. Nature Reviews. Molecular Cell Biology* 10(6), pp. 410–22, doi:10.1038/nrm2698.

[23] S. Regot, J. Macia, N. Conde, K. Furukawa, J. Kjellén, T. Peeters, S. Hohmann, E. de Nadal, F. Posas & R. Solé (2010): *Distributed Biological Computation with Multicellular Engineered Networks*. *Nature*, pp. 2–6, doi:10.1038/nature09679.

[24] D. E. Thomas & P. Moorby (1998): *The Verilog Hardware Description Language*. Kluwer Academic Publishers, doi:10.1007/978-1-4615-3992-6.

[25] P. Umesh, F. Naveen, C.U.M. Rao & S.A. Nair (2010): *Programming Languages for Synthetic Biology*. *Systems and Synthetic Biology* 4(4), pp. 265–269, doi:10.1007/s11693-011-9070-y.

[26] H. Ye, M. Daoud-El Baba, R-W. Peng & M. Fussenegger (2011): *A Synthetic Optogenetic Transcription Device Enhances Blood-glucose Homeostasis in Mice. Science (New York, N.Y.)* 332(6037), pp. 1565–8, doi:10.1126/science.1203535.

program	::= {behaviour}
behaviour	::= behaviour, behaviour | behaviour
behaviour	::= compartment | dependence | context | observation | defattributes
compartment	::= varconstant {behaviour}
observation	::= varconstant**::**worlds
context	::= [varconstants] {behaviour}
dependence	::= worlds ○→ worlds | worlds ⊙→ worlds | worlds ⊕→ worlds
world	::= attribute | varconstant(attribute) | varconstant.world
worlds	::= worlds + world | world
attribute	::= varconstant | $\overline{\text{varconstant}}$
defattribute	::= varconstants : attspec
attspec	::= defspec{varconstants} | {attrels}
defspec	::= exclusion | inclusion
attrels	::= attrels, attrel | attrel
attrel	::= varconstant ≺ varconstant | varconstant ≉ varconstant | varconstant
varconstant	::= *word* | *Word*
varconstants	::= varconstants, varconstant | varconstant

Table 7: Syntax of GUBS program

Appendix

Proofs

Proposition 1. By contradiction, assume that P is unobservable, then there does not exist a model satisfying the formula. As Q is observable, we deduce that there exists models satisfying Q, but no restricted model must satisfy P, that contradicts the definition of the behavioural consequence. □

Proposition 3. *Let $\psi \in F_{\mathcal{H}}$ be a formula, let $\sigma : (\mathit{NOM} \cup \mathit{PROP} \cup \mathit{REL}) \to (\mathit{NOM} \cup \mathit{PROP} \cup \mathit{REL})$ be a substitution on nominals, variables and relational symbols, let $\mathcal{M} = \langle W, (R_k)_{k \in \tau}, V \rangle$ be a model, we define the model $\tilde{\mathcal{M}} = \langle W, (\tilde{R}_k)_{k \in \tilde{\tau}}, \tilde{V} \rangle$ from $\mathcal{M}$ as follows:*

1. $\forall a \in \mathit{NOM} \cup \mathit{PROP}, \forall w \in W : w \in V(a\sigma) \iff w \in \tilde{V}(a)$
2. $\forall k \in \tilde{\tau} : wR_{k\sigma}w' \iff w\tilde{R}_k w'$;

we have: $\mathcal{M}, w \Vdash \psi\sigma \iff \tilde{\mathcal{M}}, w \Vdash \psi$.

Proof. The proof is defined by induction on the formula:

without loss of generality, we assume that ψ is in Negation Normal Form where negation occurs only immediately before variables only. Recall that every formula can be set in Negation Normal Form.

- $\mathcal{M}, w \Vdash a \iff \tilde{\mathcal{M}}, w \Vdash a, a \in \mathsf{PROP} \cup \mathsf{NOM}$. By (1), we have $w \in V(a\sigma) \iff w \in \tilde{V}(a)$ leading to the equivalence.
- $\mathcal{M}, w \Vdash \neg a \iff \tilde{\mathcal{M}}, w \Vdash \neg a$. By definition of the realizability relation, this is equivalent to: $\tilde{\mathcal{M}}, w \nVdash a \iff \tilde{\mathcal{M}}, w \nVdash a$. By (1), this equivalence holds.
- $\mathcal{M}, w \Vdash (\psi_1 \wedge \psi_2)\sigma \iff \tilde{\mathcal{M}}, w \Vdash (\psi_1 \wedge \psi_2)$. By definition of the substitution, we have to prove: $\mathcal{M}, w \Vdash (\psi_1\sigma) \wedge (\psi_2\sigma) \iff \tilde{\mathcal{M}}, w \Vdash (\psi_1 \wedge \psi_2)$. By definition of the realizability relation we can formulate the property equivalently as follows:

$$\mathcal{M}, w \Vdash (\psi_1\sigma) \wedge \tilde{\mathcal{M}}, w \Vdash (\psi_2\sigma) \iff \tilde{\mathcal{M}}, w \Vdash \psi_1 \wedge \tilde{\mathcal{M}}, w \Vdash \psi_2.$$

By induction hypothesis, we have: $\tilde{\mathcal{M}},w \Vdash (\psi_1\sigma) \iff \tilde{\mathcal{M}},w \Vdash \psi_1$ and $\tilde{\mathcal{M}},w \Vdash (\psi_2\sigma) \iff \tilde{\mathcal{M}},w \Vdash \psi_2$, implying the previous condition.

- $\mathcal{M},w \Vdash (\psi_1 \vee \psi_2)\sigma \iff \tilde{\mathcal{M}},w \Vdash (\psi_1 \vee \psi_2)$. The proof is similar to the proof of the previous item ($\wedge$).
- $\mathcal{M},w \Vdash (@_a\psi)\sigma \iff \tilde{\mathcal{M}},w \Vdash @_a\psi$. By definition of the substitution we have to prove that: $\mathcal{M},w \Vdash (@_{a\sigma}\psi\sigma) \iff \tilde{\mathcal{M}},w \Vdash @_a\psi$ By definition of the realizability relation, this is equivalent to:
$$\exists w' \in W : w \in V(a\sigma) \wedge \mathcal{M},w' \Vdash \psi\sigma \iff \exists w'' \in W : w'' \in \tilde{V}(a)\sigma \wedge \tilde{\mathcal{M}},w'' \Vdash \psi.$$
By setting $w' = w''$, from (1) we have: $w' \in V(a\sigma) \iff w' \in V(a)$. By induction hypothesis, we have: $\mathcal{M},w' \Vdash \psi\sigma \iff \tilde{\mathcal{M}},w' \Vdash \psi$. The both last properties imply that:
$$\exists w' \in W : w \in V(a\sigma) \wedge \mathcal{M},w' \Vdash \psi\sigma \iff \exists w' \in W : w' \in \tilde{V}(a)\sigma \wedge \tilde{\mathcal{M}},w'' \Vdash \psi,$$
which implies the initial property.
- $\mathcal{M},w \Vdash (\langle k\rangle\psi)\sigma \iff \tilde{\mathcal{M}},w \Vdash \langle k\rangle\psi$. By definition of the substitution we prove that: $\mathcal{M},w \Vdash \langle k\sigma\rangle\psi\sigma \iff \tilde{\mathcal{M}},w \Vdash \langle k\rangle\psi$.
By definition of the realizability relation the condition is equivalent to:
$$\exists w' \in W : \mathcal{M},w' \Vdash \psi\sigma \wedge wR_{k\sigma}w' \iff \exists w'' \in W : \tilde{\mathcal{M}},w'' \Vdash \psi \wedge w\tilde{R}_k w''.$$
By setting $w' = w''$, the following equivalence holds from (2): $wR_{k\sigma}w' \iff w\tilde{R}_k w'$. By induction hypothesis, we have: $\mathcal{M},w' \Vdash \psi\sigma \iff \tilde{\mathcal{M}},w' \Vdash \psi$. The both last properties imply that:
$$\exists w' \in W : \mathcal{M},w' \Vdash \psi\sigma \wedge wR_{k\sigma}w' \iff \tilde{\mathcal{M}},w' \Vdash \psi \wedge w\tilde{R}_k w'$$
which implies the initial property.
- $\mathcal{M},w \Vdash ([k]\psi)\sigma \iff \tilde{\mathcal{M}},w \Vdash [k]\psi$. The proof is similar to the previous item.
- $\mathcal{M} \Vdash (\mathbf{E}\psi)\sigma \iff \tilde{\mathcal{M}} \Vdash \mathbf{E}\psi$. By definition of the substitution we prove that: $\mathcal{M},w \Vdash \mathbf{E}(\psi\sigma) \iff \tilde{\mathcal{M}},w \Vdash \mathbf{E}\psi$.
By definition of the realizability relation, we have:
$$\exists w \in W : \mathcal{M},w \Vdash (\psi\sigma) \iff \tilde{\mathcal{M}},w \Vdash \psi,$$
which is directly verified by induction hypothesis.
- $\mathcal{M} \Vdash (\mathbf{A}\psi)\sigma \iff \tilde{\mathcal{M}} \Vdash \mathbf{A}\psi$. The proof is similar to the previous item.

□

Proposition 2. First, let us remark that when $P \not\sqsubseteq Q$, the property is trivially verified. Besides, under the assumption $P \sqsubseteq Q$, if $Q[\sigma]$ is not observable the property is also verified because an unobservable program includes all programs behaviourally (Definition 2).

In the rest of the proof, we assume that P is behaviourally included in Q and $Q[\sigma]$ is observable (*i.e.*, $P \sqsubseteq Q$ and $\mathbf{obs}\,Q[\sigma]$). Hence, by definition of the observability there exists a model $\mathcal{M}$ such that $\mathcal{M} \Vdash [\![Q[\sigma]]\!]$. By proposition 3, we deduce that there exists a model $\tilde{\mathcal{M}}$ such that: $\tilde{\mathcal{M}} \Vdash [\![Q]\!]$. Moreover, as $P \sqsubseteq Q$ by hypothesis, there exists $\tilde{S} \subseteq \mathrm{Dom}\,\tilde{\mathcal{M}}$ such that: $\tilde{\mathcal{M}}_{\tilde{S}} \Vdash [\![P]\!]$. By construction of $\tilde{\mathcal{M}}$ we deduce that there exists a sub model of $\mathcal{M}$, denoted by $\mathcal{M}'$, complying to the properties, (1) and (2) of Proposition 3 which corresponds to $\tilde{\mathcal{M}}_{\tilde{S}}$. Moreover, we have $\mathcal{M}' \Vdash P[\sigma]$ by Proposition 3. Then we conclude that: $P[\sigma] \sqsubseteq Q[\sigma]$. □

Theorem 1. First, let us remark that $P \sqsubseteq Q$ is true whenever $\mathcal{M} \nVdash Q$ by definition of the behavioural inclusion (Definition 2). Hence, the proof doesn't consider the trivial verified case but rather the case where $\mathcal{M} \Vdash Q$.

Inst. Directly from the definition of the behavioural inclusion (Definition 2).

Com. By definition of the semantics $[\![P,P']\!] = \mathbf{A}(\phi \wedge \phi') = \mathbf{A}(\phi' \wedge \phi) = [\![P',P]\!]$ with $[\![P]\!]_P = \phi$ and $[\![P']\!]_P = \phi'$. Thus, for all $\mathcal{M}$ we have: $\mathcal{M} \Vdash [\![P,P']\!] \iff \mathcal{M} \Vdash [\![P',P]\!]$. Hence, if $Q \sqsubseteq P,P'$ we conclude that: $Q \sqsubseteq P',P$.

Cont. Similar to the proof of (Com.).

Asm. First let us remark that $\sigma|_{\mathrm{VA}(P)\cap\mathrm{VA}(P')} = \sigma'|_{\mathrm{VA}(P)\cap\mathrm{VA}(P')}$ means that the substitution of the common variables are the same for σ and σ', leading to, $Q[\sigma \cup \sigma'] = Q[\sigma]$ and $Q'[\sigma \cup \sigma'] = Q'[\sigma']$. Let $\sigma'' = \sigma \cup \sigma'$. Then, we have the following property by definition of the semantics (Table 2.1) and σ''.

$$\forall \mathcal{M} \in \mathsf{KS}([\![(Q,Q')[\sigma'']]\!]) : \mathcal{M} \Vdash [\![Q[\sigma]]\!] \wedge \mathcal{M} \Vdash [\![Q'[\sigma']]\!].$$

Notice that the set of models, $\mathsf{KS}([\![(Q,Q')[\sigma'']]\!])$, is not empty since, by hypothesis, $\mathbf{obs}(Q[\sigma],Q'[\sigma'])$ holds. As $Q \hookleftarrow_{\sigma} P$ and $Q' \hookleftarrow_{\sigma'} P'$, any model validating Q (resp. Q') also validates P, (resp. P') by definition of the functional synthesis. Then, we deduce that:

$$\forall \mathcal{M} \in \mathsf{KS}([\![(Q,Q')[\sigma'']]\!]) : \mathcal{M} \Vdash [\![P[\sigma]]\!] \wedge \mathcal{M} \Vdash [\![P'[\sigma']]\!].$$

Then, we conclude that:

$$\forall \mathcal{M} \in \mathsf{KS}([\![(Q,Q')[\sigma'']]\!]) : \mathcal{M} \Vdash [\![(P,P')[\sigma'']]\!].$$

□

Complete compilation of the Band Detector

- Sender -

$$\dfrac{Q_1,Q_2,Q_3[\sigma = \{detect/Detect, light/Light, v_1/Tetr, v_2/LuxI\}] \subseteq_{Asm} P'_{11}[\sigma] \qquad obs(Q_1,Q_2,Q_3[\sigma])}{Q_1,Q_2,Q_3 \leftharpoonup_{\sigma} P'_{11}\}}\ (\text{Inst.})$$

$$\dfrac{Q_1,Q_2,Q_3[\sigma' = \{detect/Detect, light/Light, v_3/Tetr, v_4/LuxI\}] \subseteq_{Asm} P'_{12}[\sigma'] \qquad obs(Q_1,Q_2,Q_3[\sigma'])}{Q_1,Q_2,Q_3 \leftharpoonup_{\sigma'} P'_{12}}\ (\text{Inst.})$$

$$\dfrac{Q_1,Q_2,Q_3[\sigma'' = \{detect/Detect, light/Light, v_5/Tetr, v_6/LuxI\}] \subseteq_{Asm} P'_{13}[\sigma''] \qquad obs(Q_1,Q_2,Q_3[\sigma''])}{Q_1,Q_2,Q_3 \leftharpoonup_{\sigma''} P'_{13}}\ (\text{Inst.})$$

$$\dfrac{\dfrac{\dfrac{P'_{11} = [light]\{detect \odot\!\!\to v_1, v_1 \odot\!\!\to v_2, v_2 \odot\!\!\to AHL(low)\}}{[light]\{detect \odot\!\!\to v_1, v_1 \odot\!\!\to AHL(low)\}}\ (\text{Trans.})}{[light]\{detect \odot\!\!\to AHL(low)\}}\ (\text{Trans.})}{P_{11}}\ (\text{N2P.}) \quad \dfrac{\dfrac{\dfrac{P'_{12} = [light]\{detect \odot\!\!\to v_3, v_3 \odot\!\!\to v_4, v_4 \odot\!\!\to AHL(mid)\}}{[light]\{detect \odot\!\!\to v_3, v_3 \odot\!\!\to AHL(mid)\}}\ (\text{Trans.})}{[light]\{detect \odot\!\!\to AHL(mid)\}}\ (\text{Trans.})}{P_{12}}\ (\text{N2P.}) \quad \dfrac{\dfrac{\dfrac{P'_{13} = [light]\{detect \odot\!\!\to v_5, v_5 \odot\!\!\to v_6, v_6 \odot\!\!\to AHL(high)\}}{[light]\{detect \odot\!\!\to v_5, v_5 \odot\!\!\to AHL(high)\}}\ (\text{Trans.})}{[light]\{detect \odot\!\!\to AHL(high)\}}\ (\text{Trans.})}{P_{13}}\ (\text{N2P.})$$

$$\dfrac{Q_1,Q_2,Q_3 \leftharpoonup_{\sigma} P_{11} \qquad Q_1,Q_2,Q_3 \leftharpoonup_{\sigma'} P_{12} \qquad Q_1,Q_2,Q_3 \leftharpoonup_{\sigma''} P_{13}}{Q_1,Q_2,Q_3 \leftharpoonup_{\sigma\cup\sigma'\cup\sigma''} P_{11},P_{12},P_{13}}\ (\text{Asm.})$$

- Receiver -

$$\dfrac{Q_4,Q_5,Q_6,Q_8[\sigma = \{v_1/LuxR, v_2/Cl, v_3/LacI\}] \subseteq_{Asm} P'_{21}[\sigma] \qquad obs(Q_4,Q_5,Q_6,Q_8[\sigma])}{Q_4,Q_5,Q_6,Q_8 \leftharpoonup_{\sigma} P'_{21}}\ (\text{Inst.})$$

$$\dfrac{Q_4,Q_5,Q_6,Q_8[\sigma' = \{v_4/LuxR, v_5/Cl, v_6/LacI\}] \subseteq_{Asm} P'_{22}[\sigma'] \qquad obs(Q_4,Q_5,Q_6,Q_8[\sigma'])}{Q_4,Q_5,Q_6,Q_8 \leftharpoonup'_{\sigma} P'_{22}}\ (\text{Inst.})$$

$$\dfrac{Q_4,Q_5,Q_7[\sigma'' = \{v_7/LuxR, v_8/LacM1\}] \subseteq_{Asm} P'_{23}[\sigma''] \qquad obs(Q_4,Q_5,Q_7[\sigma''])}{Q_4,Q_5,Q_7 \leftharpoonup''_{\sigma} P'_{23}}\ (\text{Inst.})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{P'_{21} = AHL(low) \odot\!\!\to v_1, v_1 \odot\!\!\to v_2, v_2 \odot\!\!\to v_3, v_3 \odot\!\!\to \overline{GFP}}{AHL(low) \odot\!\!\to v_1, v_1 \odot\!\!\to v_2, v_2 \odot\!\!\to \overline{GFP}}\ (\text{Trans.})}{AHL(low) \odot\!\!\to v_1, v_1 \odot\!\!\to \overline{GFP}}\ (\text{Trans.})}{AHL(low) \odot\!\!\to \overline{GFP}}\ (\text{Trans.})}{P_{21}}\ (\text{N2P.}) \quad \dfrac{\dfrac{\dfrac{\dfrac{P'_{22} = AHL(mid) \odot\!\!\to v_4, v_4 \odot\!\!\to v_5, v_5 \odot\!\!\to v_6, v_6 \odot\!\!\to GFP}{AHL(mid) \odot\!\!\to v_4, v_4 \odot\!\!\to v_5, v_5 \odot\!\!\to GFP}\ (\text{Trans.})}{AHL(mid) \odot\!\!\to v_4, v_4 \odot\!\!\to GFP}\ (\text{Trans.})}{AHL(mid) \odot\!\!\to GFP}\ (\text{Trans.})}{P_{22}}\ (\text{N2P.}) \quad \dfrac{\dfrac{\dfrac{P'_{23} = AHL(high) \odot\!\!\to v_7, v_7 \odot\!\!\to v_8, v_8 \odot\!\!\to \overline{GFP}}{AHL(high) \odot\!\!\to v_7, v_7 \odot\!\!\to \overline{GFP}}\ (\text{Trans.})}{AHL(high) \odot\!\!\to \overline{GFP}}\ (\text{Trans.})}{P_{23}}\ (\text{N2P.})$$

$$\dfrac{Q_4,Q_5,Q_6,Q_8 \leftharpoonup_{\sigma} P_{21} \qquad Q_4,Q_5,Q_6,Q_8 \leftharpoonup_{\sigma'} P_{22} \qquad Q_4,Q_5,Q_7 \leftharpoonup_{\sigma''} P_{23}}{Q_4,Q_5,Q_6,Q_7,Q_8 \leftharpoonup_{\sigma\cup\sigma'\cup\sigma''} P_{21},P_{22},P_{23}}\ (\text{Asm.})$$

- Final design -

Sender		*Receiver*		
$\{\mathsf{AHL}{:}\{low \not\approx mid \not\approx high\}$,	$[\mathsf{Light}]\{detect \circ\!\!\to \mathsf{Tetr}\}$,	$\{\mathsf{AHL}{:}\{low \not\approx mid \not\approx high\}$,		$\mathsf{LuxR}{:}\{low \not\approx \{mid \prec high\}\}$,
$\mathsf{Tetr} \xrightarrow{+} \mathsf{LuxL}$,	$\mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(low)$,	$\mathsf{AHL}(mid) \circ\!\!\to \mathsf{LuxR}(mid)$,		$\mathsf{AHL}(high) \circ\!\!\to \mathsf{LuxR}(high)$,
$\mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(mid)$,	$\mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(high)\}$	$\mathsf{LuxR}(mid) \xrightarrow{+} \mathsf{CI}$,		$\mathsf{LuxR}(high) \xrightarrow{+} \mathsf{LacIM1}$,
		$\mathsf{CI} \xrightarrow{-} \mathsf{LacI}$,	$\mathsf{LacIM1} \xrightarrow{-} \mathsf{GFP}$,	$\mathsf{LacI} \xrightarrow{-} \mathsf{GFP}\}$

Table 8: Complete band detector compilation.