Automated design, build, test, learn workflows to engineer synthetic genetic networks

Synthetic Biology

Gonzalo Andrés Vidal Peña

School of Computing
Newcastle University

This dissertation is submitted for the degree of
Doctor of Philosophy

ICOS March 2025

I would like to dedicate this thesis to my family, friends and heroes ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments. This dissertation contains fewer than 80,000 words inclusive of notes, but excluding references and appendices.

Gonzalo Andrés Vidal Peña
March 2025

Acknowledgements

Words cannot express my gratitude to Renata, my wife and the love of my life, and to Aurora, my daughter and my light in my sky. Thanks for helping me through the whole process of the PhD and for sharing with me so many insightful conversations and experiences. I am also grateful to my family, they always believe in me and guide my path towards being a better person. I would like to recognise my friend Carolus Vitalis for all his help and deep discussions about synthetic biology, Fernán Federici for facilitating the Loop kit, Chris Myers for supporting my work and sharing a joint vision of the field, Jake Beal for supervising and supporting my work for the sake of synthetic biology, Natalio Krasnogor for being my co-supervisor and Tim Rudge for being my supervisor and for sharing a deep drive and motivation to leverage the engineering method in synthetic biology. Thank you for introducing me to the field of synthetic biology with your perspective.

Lastly, I would like to acknowledge everyone involved in my education. I see education as the set up of a software that runs societies. For some reason after the obligatory education students are dropped into the game of life without knowing the rules. Education is failing in some critical areas like forming citizens, not teaching how to learn and how the current economic system works. This could be why politicians still do not understand politics and are playing around the power not knowing how to use it. Because at the end of the day politicians are part of the same system that they have created so they are not prepared for that role. After the pandemic, the world tasted a very technological way of working. Remote work and remote teaching is possible, and as a person who took online classes, they were pretty good. Access to the Internet and technologies like artificial intelligence, the metaverse and synthetic biology will revolutionise education for the better. In the hands of the future generations is to update the software that runs societies.

...

Abstract

Synthetic biology is an interdisciplinary field that pursues the engineering of biological systems. The design, build, test, learn (DBTL) cycle is at the core of engineering disciplines and is iterated until a desired goal is achieved. Synthetic biology is still defining abstractions, standards and developing a software ecosystem to iterate the DBTL cycle.

The aim is to work in a similar way as other engineering disciplines, making designs with a computational aided design (CAD) tool that can simulate the expected behaviour of the designed biological system, and that can communicate to build tools to create a physical implementation of the biological system. After the biological system is built, it is tested by taking measurements of its behaviour. The test has to be automated, calibrated and standardised to get high quantity and quality data that can inform the learn stage properly. Given the diversity of synthetic biology and its applications the DBTL cycle could have different needs when the researcher needs to engineer a genetic network, a metabolic pathways, a strain or a protein, among others. The focus of this work is in creating DBTL cycle workflows for engineering synthetic genetic network dynamics, because it allows to control the logic of a system and how that logic state is reached and maintained over time with direct applications in biochemical production, drug dosage, and the study of pattern formation and developmental biology. Existing tools for engineering genetic network dynamics do not cover the whole DBTL cycle and lack connections, leaving several gaps. Most tools do not use standardised inputs and outputs hindering the connectivity between tools and slowing the research process.

To iterate faster through the DBTL cycle it has to be closed and automated by leveraging software tools and liquid handling robots. Software tools have to be compatible with standards to make them useful and accessible for the community, promoting the use of best practices. The workflow has to be flexible to accommodate different needs and resources, to be used for researchers without a wetlab, with non-automated wetlab and with lab automation. Here I have created a set of software tools tackling different DBTL cycle stages that are modular and leverage standards to connect and automate the DBTL cycle for genetic network engineering. The workflows developed in this work provides novel teaching and research tools available for different needs.

Table of contents

List of figures **xv**

List of tables **xxvii**

Nomenclature **xxix**

1 Introduction **1**

1.1 Synthetic Biology . . . 2
1.2 Molecular genetics . . . 3
1.2.1 Gene . . . 3
1.2.2 Deoxyribonucleic Acid . . . 3
1.2.3 Replication . . . 4
1.2.4 Transcription . . . 5
1.2.5 Translation . . . 6
1.3 Foundational technologies . . . 8
1.3.1 Synthesis . . . 8
1.3.2 Amplification . . . 10
1.3.3 Sequencing . . . 10
1.3.4 Assembly . . . 11
1.3.5 Transformation . . . 13
1.3.6 Reporters . . . 13
1.3.7 Cloning . . . 13
1.4 Genetic network engineering . . . 14
1.4.1 Genetic networks . . . 14
1.4.2 Abstraction . . . 15
1.4.3 Standardisation . . . 15
1.4.4 Decoupling . . . 16
1.5 Synthetic biology open language . . . 16

1.6 Engineering method . . . 17
1.6.1 Design . . . 17
1.6.2 Build . . . 18
1.6.3 Test . . . 18
1.6.4 Learn . . . 19
1.7 Software . . . 20
1.7.1 Computation . . . 20
1.7.2 Biocomputation . . . 22
1.7.3 Computational biology . . . 22
1.8 Laboratory automation . . . 22
1.9 iGEM . . . 23
1.10 Aim . . . 23
1.11 Objectives . . . 24
1.11.1 Objective A. Create a design tool for synthetic genetic networks. . . . 24
1.11.2 Objective B. Automate the build stage with software and robotics. . . 25
1.11.3 Objective C. Develop and implement a method for characterising genetic networks to connect Learn and Design stages. . . . 25
1.11.4 Objective D. Demonstrate DBTL workflows. . . . 25
1.12 Structure of the thesis . . . 25
1.13 Contribution of the thesis . . . 26

2 Design abstraction for genetic devices 31
2.1 Introduction . . . 32
2.2 Results . . . 32
2.2.1 Design Abstraction for genetic devices . . . 32
2.2.2 Part architecture and transcriptional unit design . . . 35
2.2.3 Assembly strategy to compose genetic devices . . . 36
2.2.4 Functional synthetic biology . . . 37
2.3 Discussion and conclusions . . . 38

3 Programmatic genetic network design automation 41
3.1 Introduction . . . 42
3.2 Results . . . 43
3.2.1 Design Abstraction for Genetic Networks . . . 43
3.2.2 Model generation and simulation . . . 45
3.2.3 Software design and architecture . . . 49
3.2.4 Encoding designs in SBOL3 . . . 50

3.2.5 Genetic network design and simulation examples . . . 53
3.2.6 Operator model parameterization . . . 69
3.3 Methods . . . 70
3.3.1 Software development . . . 70
3.3.2 Noise in Kinetic Gene Expression and Biomass Simulations . . . 70
3.4 Discussion and conclusions . . . 72

4 Standardizing and automating build planning 75
4.1 Introduction . . . 76
4.2 Results . . . 76
4.2.1 Encoding building plans in SBOL3 . . . 76
4.2.2 Implementing building plans using Python . . . 77
4.2.3 Creating build metadata in SBOL3 . . . 81
4.2.4 Automating build using Opentrons OT-2 liquid handling robot . . . 85
4.3 Methods . . . 95
4.3.1 Software development . . . 95
4.4 Discussion and conclusions . . . 95

5 Characterization of growth and gene expression rates 99
5.1 Introduction . . . 100
5.2 Results . . . 100
5.2.1 Gene Expression and Growth Dynamics . . . 100
5.2.2 Dynamics Estimation as an Inverse Problem . . . 102
5.2.3 Accurate Estimation of Growth Rate Dynamics . . . 104
5.2.4 Accurate Estimation of Gene Expression Rate Dynamics . . . 106
5.2.5 Characterizing Growth and Gene Expression Rate Dynamics in *Escherichia coli* . . . 108
5.2.6 Characterization of Gene Expression Rate Dynamics Relative to *in vivo* Reference . . . 110
5.3 Methods . . . 112
5.3.1 Software development . . . 112
5.3.2 Kinetic Gene Expression and Growth Assays . . . 113
5.3.3 Computational Methods . . . 113
5.3.4 Web-based Software Implementation . . . 113
5.3.5 Random Profile Generation . . . 114
5.3.6 Simulation of Kinetic Gene Expression and Biomass Data . . . 114
5.3.7 Choice of hyperparameters . . . 115

5.4 Discussion and conclusions 115

6 Automated workflows to engineer synthetic genetic networks 119
6.1 Introduction 120
6.2 Results 120
6.2.1 Genetic design automation exploring the design space with simulated DBTL cycles 120
6.2.2 Automating the DBTL cycle with manual build to characterise constitutive gene expression 123
6.2.3 Connecting software tools and liquid handling robots to automate the DBTL cycle 128
6.3 Methods 136
6.3.1 Software Development 136
6.3.2 dsDNA design 136
6.3.3 Supplemented M9 media preparation 136
6.3.4 Transformation solution 1 136
6.3.5 Transformation solution 2 137
6.3.6 Competent cell preparation 137
6.3.7 DNA extraction 138
6.3.8 DNA assembly 138
6.3.9 Transformation 138
6.3.10 Automated DNA assembly 138
6.3.11 Automated Transformation 139
6.3.12 Automated test setup 139
6.3.13 Calibration 139
6.3.14 Kinetic gene expression and growth assays 139
6.4 Discussion and conclusions 140

7 Conclusion 143

8 Discussion 147

References 153

Appendix A Automated design build test learn workflows to engineer synthetic genetic networks 169

List of figures

1.1 Watson and Crick proposed the double helix model for DNA. (A) The sugar-phosphate backbones are on the outside of the double helix and purines and pyrimidines form the "rungs" of the DNA helix ladder. (B) The two DNA strands are antiparallel to each other. (C) The direction of each strand is identified by numbering the carbons (1 through 5) in each sugar molecule. The 5' end is the one where carbon 5 is not bound to another nucleotide; the 3' end is the one where carbon 3 is not bound to another nucleotide and has a free hydroxyl group. Extracted from penStax-CNX. 4

1.2 Bacterial transcription. (A) Bacterial RNA polymerase (RNAP) is a multi-subunit enzyme. The core RNAP is composed of five subunits: $\alpha 2$ (cyan bubbles), β and β' (grey) and ω (black). The core RNAP contains the active site that catalyses the formation of the phosphodiester bond of nascent RNA. The RNAP α subunits interact with the upstream promoter (UP) element, which consists of two distinct subsites located upstream of the 35 element [53]. The RNAP core enzyme interacts in a sequence-specific manner with the template-strand positions –4 to +2, which constitute the core recognition element (CRE) (navy blue) [200]. To form the holoenzyme (Eσ), RNAP is associated with a σ factor. The σ70-related factors contain up to four functional domains (σ1–4; the spirals represent linkers between the domains). The $\sigma 2$ domain recognizes and interacts with the 10 element, and the $\sigma 4$ domain interacts with the 35 element. The extended 10 element interacts with $\sigma 3$; this interaction is crucial in promoters whose 35 and 10 elements show a poor match to consensus sequences [85]. Some promoters have a discriminator (DISC), which is recognized and interacts with the $\sigma 2$ domain [73, 83]. The σ70 factor bound to the non-template strand captures the 10 region in an open complex and allows the single-strand template DNA to enter the active site. (B) The RNAP holoenzyme (R) and promoter DNA (P) interact to form the closed complex (RPc). The duplex DNA around the transcription start site is separated (represented by a 'bubble' in the DNA that is bound by the Eσ) in order to form the open complex (RPo). The initiating complex (RPinit) is formed and begins synthesis of the RNA chain (shown as a small red line), directed by the DNA template strand. In initiation of the synthesis of RNA, a phosphodiester bond is formed between the initiating and adjacent phosphodiester nucleoside triphosphates (NTPs). The final stage of transcription initiation is elongation. In the elongation phase, the RNA chain length increases, shown as a solid red line. Part a adapted from ref. [146], CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/). Part b adapted from ref. [22], Springer Nature Limited. Illustration adapted from ref. [114]. 7

1.3 Translation complex. A cognate aminoacyl-tRNA bound to EF-Tu binds to its cognate codon of the mRNA in the ribosome at the acceptor site (A site), adjacent to the tRNA in the peptidyl site (P site), which is bound to the nascent polypeptide (1). Decoding occurs on the 30S subunit of the ribosome, GTP is hydrolysed by EF-Tu, which leads to release of aminoacyl-tRNA into the A site if there is correct codon–anticodon pairing. Subsequent peptide bond formation between the nascent polypeptide in the P site and the amino acid attached to the tRNA in the A site results in a peptidyl-tRNA in the A site that is one amino acid longer (2). After translocation of the ribosome, the tRNA attached to the nascent polypeptide is moved to the P site and a new aminoacyl-tRNA is delivered to the empty A site (3). Extracted from [18] 9

1.5 Flapjack's data model. The basic unit of data is the measurement, which is the readout of a reporter or biomass signal in time. A measurement refers to a specific sample, which can be a well in a plate-reader experiment or a colony in a Petri dish time-lapse. Samples are grouped in a specific assay, which corresponds to a single kinetic procedure. Assays belong to a particular study that aims to answer a scientific question or project. Along with signal data, each measurement is described in terms of its context through experimental metadata, mainly composed of the media substrate, the host strain, the DNA which the host is transformed with, and inducer chemicals to study stress or other regulation behaviours of the system. It is important to mention that some metadata could not be present, such as strain in Cell-free assays or DNA in control samples, to name a few. PK is the primary key, FK is foreign key, and MM is a many-to-many relation. From Flapjack publication [196]. 21

1.6 Graphical abstract. 24

2.1 Genetic device design abstraction. SBOL is used to represent a genetic device that senses and input and produces an output. 34

2.2 Transcriptional unit design. (A) Example of a trivial transcriptional unit using SBOL Visual. (B) Representation of the transcriptional unit design with collapsed parts using SBOL Visual. (C) Representation of the transcriptional unit design parts using SBOL Visual. (D) Assembly strategy using the Phytobricks syntax for degradation tag combinatorial design. (E) Assembly strategy using the Phytobricks syntax for trivial transcriptional units. . . . 35

2.3 Functional synthetic biology. Diagram of the flow of structural and functional information under the functional synthetic biology paradigm. The functional loop, related to the Design stage, is on the left side. In the example we have the CDS glyph representing the function "Green Fluorescent Protein coding sequence". When an interface is specified the function becomes a device. Then, devices can be composed using that interface to form systems. If the system has a higher level function (i.e. if x is sensed, produce green fluorescence) it can circle back and interfaces can be added to form a new device. The structural loop, related to the Build stage, is on the right side. In the example we have fragment of the coding sequence for GFPmut3. When an interface is specified the sequence becomes a part, such as adding MoClo fusion sites C and D flanking the sequence. Then, parts can be compose using the interfaces to form composites. Because composite are just larger sequence they can circle back and become a part if interfaces are added. There is no direct relationship between a function and a part, as in the structural paradigm. The functions can be fulfilled by different parts, allowing for more adaptability on the build stage and working as a base for device versioning, an analogy to software engineering. 38

3.1 LOICA classes. Encapsulation relation of classes in LOICA design abstraction. 44

3.2 Relation between the mathematical and computational model. (A) In the equation $\frac{d\vec{p}}{dt}$ is the dynamics of the $\vec{p}$ vector of proteins over time, $\Psi(\vec{r})$ is the protein synthesis rate or expression rate linked to the `Operator` and may depend in regulator proteins from the vector of regulators $\vec{r}$. $\eta(t)$ is a function of time that adjust gene expression for time and context creating a profile or time-series. $\Gamma\vec{p}$ is the vector of degradation rates of $\vec{p}$ linked to `GeneProducts`. $\mu(t)\vec{p}$ is the growth rate effect on the dilution of proteins from the vector of proteins $\vec{p}$ linked to `Metabolism`. (B) Diagrammatic representation of the computational model used in LOICA. TUs can be splitted and represented by an `Operator` and a `GeneProduct`, both are encapsulated by `GeneticNetwork` and `Metabolism` sets the context where the genes are expressed. Both are encapsulated by `Sample` to run simulations. 46

3.3 SBOL visual diagram of a TU containing the promoter J23101. The SBOL components contained in the `Operator` and `GeneProduct` are in the green and brown boxes respectively. 47

3.4 SBOL visual diagram of a TU containing the HSL regulated promoter pLux. The SBOL components contained in the `Operator`, `GeneProduct` and `Supplement` are in the green, brown and yellow boxes respectively. . . 47
3.5 SBOL visual diagram of a TU containing the repressible promoter pLac. The SBOL components contained in the `Operator` and `GeneProduct` are in the green and brown boxes respectively. 48
3.6 SBOL visual diagram of a TU containing the tandem promoter P_{Tac} and P_{PhiF}. The SBOL components contained in the `Operator` and `GeneProduct` are in the green and brown boxes respectively. 49
3.7 Objects relationship. Diagram of an `Assay` encapsulating a `Sample` which in turn encapsulates `Metabolism`, `Supplement`, and `GeneticNetwork`. `GeneticNetwork` encapsulates the `Operator` and `Regulator` which generate a model through their interactions. On the right side the different interactions with the Flapjack and SBOL models are shown. 51
3.7 Class Diagram. Purple boxes represent Abstract Class entities and blue boxes represent Class entities. Diamond end arrows represents composition relationships, and triangular end arrows represents inheritance relationships. 52
3.8 Source genetic network. (A) Full genetic network diagram. (B) Contracted genetic network diagram. As `Operators` and `Reporters` are shown in the both, the visualzation is the same. 53
3.9 Source simulations. (A) Source GFP and biomass profiles simulated with LOICA using Gompertz metabolism and ODEs. (B) Source GFP and biomass profiles simulated with LOICA using Gompertz metabolism and ODE NSR 10^{-3}. (C) Source GFP profile simulated with LOICA using growth rate 0, biomass 1 and the Gillespie algorithm. 55
3.10 Receiver genetic network.(A) Full genetic network diagram. (B) Contracted genetic network diagram. As `Operators` and `Reporters` are shown in the contracted network the only difference is the `Supplement`. 56
3.11 Receiver expression profiles. Receiver GFP profiles simulated with LOICA using Gompertz metabolism and ODEs over different concentrations of inducer. 57

3.12 Receiver transfer curves. Average MEFL over profile expression at simulated samples with different concentration of inducer. (A) Parameters used in the `Receiver`: alpha = [0,100], K = 1, n = 2. (B) Parameters used in the `Receiver`: alpha = [20,200], K = 1, n = 2. (C) Parameters used in the `Receiver`: alpha = [0,100], K = 10, n = 2. (D) Parameters used in the `Receiver`: alpha = [0,100], K = 1, n = 4. 59
3.13 Inverter genetic network. (A) Full genetic network diagram. (B) Contracted genetic network diagram. 60
3.14 Inverter expression profiles. (A). Receiver GFP profiles simulated with LOICA using Gompertz metabolism and ODEs over different concentrations of inducer. (B) Inverter RFP profiles simulated with LOICA using Gompertz metabolism and ODEs over different concentrations of inducer. 61
3.15 Inverter transfer curves. Average green and red fluorescence profile expression at simulated samples with different concentrations of inducer. In this simulation a `Receiver` represents a TU that senses a biochemical signal and produces a the `Regulator` LacI and the `Reporter` GFP. The `Hill1` represents a TU that senses the `Regulator` LacI and in turn express the `Reporter` RFP. The `Receiver` parameters are fixed alpha = [0,100], K = 1, n = 2. (A) Parameters used in the `Hill1`: alpha = [0,100], K = 1, n = 2. (B) Parameters used in the `Hill1`: alpha = [20,200], K = 1, n = 2. (C) Parameters used in the `Hill1`: alpha = [0,100], K = 10, n = 2. (D) Parameters used in the `Hill1`: alpha = [0,100], K = 1, n = 4. 62
3.16 Oscillator genetic network. (A) Full genetic network diagram. (B) Contracted genetic network diagram. 63
3.17 Bistable genetic network simulations. (A) Bistable genetic network, simulated with LacI set to 5 as initial concentration. (B) Bistable genetic network, simulated with TetR set to 5 as initial concentration. 65
3.18 Oscillator genetic network. (A) Full genetic network diagram. (B) Contracted genetic network diagram. 66
3.19 Genetic oscillator simulations. (A) Oscillator GFP and biomass profiles simulated with LOICA using Gompertz metabolism and ODEs. (B) Oscillator GFP and biomass profiles simulated with LOICA using Gompertz metabolism and ODE NSR 10^{-3}. (C) Oscillator GFP profile simulated with LOICA using growth rate 0, biomass 1 and the Gillespie algorithm. 68

3.20 Characterization from simulated data workflow. (A) Accessory genetic network designs in LOICA. (B) LOICA simulates experimental data. (C) Simulated data stored in Flapjack. (D) `Operators` model characterization. . 71

4.1 Metadata representation flow diagram. SBOL Components (Blue) and representation of their physical implementation (Purple) used to represent the metadata produced at Build stage. The abstract elements `DNA` and `Strain` are encapsulated by `GMO`, `Chemical` is encapsulated by `Supplement`. `Media`, `GMO` and `Supplement` are encapsulated by `Sample` which in turn is encapsulated into `Assay`. The metadata from the physical implementations of these elements to prepare a test, such as sample design which includes `Chemical`, `Strain`, `Media` and `GMO`. In this context DNA implementations can be found in three forms, `Extracted DNA` represents DNA that has been extracted from a `GMO` through miniprep for example. `Diluted DNA` represents a DNA with a known concentration that can be product of diluting an `Extracted DNA` or synthesis product, it can be in fmol/μL and used for assemblies. `Assembled DNA` represents assembly products. All three can be used with a `Strain` to create new `GMO` and optionally have quality and sequencing information. 84

4.2 PUDU workflow diagram. Synthetic biology workflow automated using PUDU. The DNA Assembly class automates domestication and the assembly using a design in SBOL or a list of parts using Loop. This process can be automated using a a DNA Assembly script. Then, the Transformation class automates the transformation of bacteria with the assembled DNA. Finally, the Test Setup class automates the setup of a 96 well plate with the transformed bacteria under different conditions. Furthermore, The Calibration class automates the preparation of a 96 well plate to obtain calibrated data. 86

4.3 Standard build plan representation of a Loop domestication. The design process starts with a new part core, in this example a new green CDS. Then its position on the assembly is decided and flanking regions are added to the design creating a part insert. The part insert then can be synthesised as a part in the linear backbone with the restriction enzyme recognition sites for part extraction. The backbone represents the universal acceptor. The part in the linear backbone and the backbone are digested using a restriction enzyme producing a part extract and an open backbone. These two digestion products then are ligated using a ligase to create a circular part in backbone that represents a level 0 part for Loop assembly. 87

4.4 Standard build plan representation of a level 1 Loop assembly. On the top left two parts in backbone are represented, one containing a promoter and other containing a green CDS. On the bottom left a backbone is represented containing a red CDS. These three plasmids are digested using a restriction enzyme, producing part extracts from the parts in backbone and an open backbone from the backbone. The three digestion products are then ligated using a ligase, producing a part in backbone where the part is a composite part of the two parts extract. 88
4.5 OT-2 deck setup for running `DNA Assembly`. The temperature module with a 24 well aluminum block was placed in the slot 1. The p20 tip rack was placed in the slot 9. The thermocycler module was placed in the slots 7,8,10 and 11. The place of the reagents in the aluminum block and of products in the thermocycler is generated during the simulation and informed as a dictionary in the terminal. 89
4.6 OT-2 deck setup for running `Chemical Transformation`. The temperature module with a 24 well aluminum block was placed in the slot 1.The p200 tip rack was placed in the slot 6. The p20 tip rack was placed in the slot 9. The thermocycler module was placed in the slots 7,8,10 and 11. The place of the reagents in the aluminum block and of products in the thermocycler is generated during the simulation and informed as a dictionary in the terminal. 91
4.7 OT-2 deck setup for running `Plate Samples`. The 24 well aluminum block was placed in the slot 4. The 96 well plate for test was placed in slot 7. The p200 tip rack was places in slot 9. The place of the reagents in the aluminum block and of samples in the 96 well plate is generated during the simulation and informed as a dictionary in the terminal. 93
4.8 OT-2 deck setup for running `iGEM RGB OD`. The 24 well aluminum block was placed in the slot 1. The 96 well plate for test was placed in slot 7. The p200 tip rack was places in slot 9. The place of reagents in the aluminum block is informed in a protocol. 94
4.9 Class Diagram. Purple boxes represent Abstract Class entities and blue boxes represent Class entities. Triangular end arrows represent inheritance relationships. 97

5.1 Algorithm overview. **A, B** Growth and gene expression rates approximation using Gaussian basis. The light green curves are individual Gaussian curves that compose the basis. The turquoise line represents the sum of the Gaussian basis. The black dashed line represents the growth rate profile (A) or gene expression rate profile (B) to be fitted. **C** Inverse problem algorithm diagram. Once an experiment is performed, biomass data ($\bar{B}$) and fluorescence data ($\bar{y}$) are collected. These data are input to the models, where the biomass model requires only biomass as input and the fluorescence model requires both biomass and fluorescence. Finally, Growth rate ($\mu(t)$) and Expression rate ($\phi(t)$) profiles are generated from the biomass and fluorescence models, respectively. The gray arrows indicate inputs and black arrows indicate outputs. 103

5.2 Method comparison on growth rate simulations. **A** The errors corresponding to the inverse, direct and indirect methods are presented, the error of the inverse method were almost 30-fold lower than the direct method and two-fold lower than the indirect method, error bars represent 95% confidence interval. **B** Error of the initial growth rate over all noise levels. Growth rate, due to its nature, has a peak, which is not easily identified with the direct method. Since the initial growth rate should be low, the direct method overestimates and the indirect method underestimates the initial growth rate, showing that the inverse method is more accurate at low biomass. Error calculated for growth rate values at time 0 compared to the true profile. **C** Examples of accurate growth rate reconstructions from simulated data. Gompertz profiles are characterized using the inverse, direct and indirect methods (blue, red, and yellow lines respectively) compared to the true profile (black dashed line). **D** Examples of inaccurate growth rate reconstructions from simulated data. Gompertz profiles are characterized using the inverse, direct and indirect methods (blue, red, and yellow lines respectively) compared to the true profile (black dashed line). (n=100) 105

5.3 Method comparison on gene expression rate simulations. **A** The errors corresponding to the inverse, the direct and the indirect method are presented; the inverse method has been shown to be four-fold better than the direct method and close to three-fold better than the indirect method, error bars represent 95% confidence interval. **B** Error at initial time. Since the initial gene expression rate simulations were not restricted to be low at the beginning the three methods show similar errors. Error calculated for expression rate values at time 0 compared to the true profile. **C** Representative examples of accurate gene expression reconstructions from simulated data. Random profiles characterised using the inverse, direct and indirect methods (blue, red, and yellow lines respectively) compared to the true profile (black dashed line). **D** representative examples of inaccurate gene expression rate reconstructions from simulated data. Random profiles characterised using the inverse, direct and indirect methods (blue, red, and yellow lines respectively) compared to the true profile (black dashed line). (n=100) 107

5.4 Gene expression dynamics shape reconstructed from experimental data is different for transcription units with the same promoter. **A** Plasmid pAAA SBOL visual diagram. **B** Raw experimental data of OD and fluorescence. **C** Growth and gene expression rates reconstructed with the inverse, direct and indirect methods (blue, red and yellow lines respectively) on strain MG1655z1 in media M9-glucose. (n=30) 109

5.5 The gene expression profile shape reconstructed from experimental data is similar for promoters under constitutive expression. TUs from pAAA were used as reference (black dashed line) for TUs within the same conditions but with different promoters (solid lines). **A** RFP reference TU compared to RFP TU with J23106 in different downstream TUs compositional context. **B** YFP reference TU compared to YFP TU with J23107 in different upstream TUs compositional contexts. **C** RFP reference TU compared to RFP TU with R0040 basal expression in different downstream TUs compositional contexts. **D** Reference YFP TU compared to YFP TU with R0011 with different upstream TUs compositional context. All profiles are shown normalised by mean and standard deviation.(n=30) 111

6.1 DBTL workflows with increasing levels of connection and automation. (A) A virtual DBTL workflow with simulated data. (B) A DBTL workflow with manual build stage automated with software. (C) A DBTL workflow using lab automation and software. 120

6.2 Simulated DBTL workflow diagram. LOICA is used at the design stage, then the output from LOICA is formatted using sbol-utilities which works as input for PUDU. PUDU simulates the build and generates metadata. Simulated data from LOICA is uploaded to Flapjack where it can be processed and analysed. 121
6.3 Repressilator generator simulations. (A) Functional oscillatory `GeneticNetwork` diagram (B) Functional oscillatory `GeneticNetwork` simulation. (C) Non functional oscillatory `GeneticNetwork` diagram (D) Non functional oscillatory `GeneticNetwork` simulation. 122
6.4 Analysis on the count of number of peaks and frequency on simulated repressilators. (A) Count of peaks over samples and signals. (B) Count of frquencies over samples and signals. 123
6.5 Workflow execution with simulated build. LOICA was used to generate simulations from parameters from the literature. Simulated data from LOICA was uploaded to Flapjack. Data was analysed, looking for peaks to identify functional repressilators and calculate frequency. 123
6.6 Manual build workflow diagram. LOICA is used at the design stage, then the output from LOICA is formatted using sbol-utilities which works as input for PUDU. PUDU simulates the build and generates metadata. The build stage, which includes the final definition of the sequence to construct, the construction of DNA and the transformation of an organism with that DNA is done manually. Experimental data is uploaded to Flapjack where it can be processed and analysed. 124
6.7 Degradation tags characterization data and analysis on manual build workflow. (A) Experimental data points of green fluorescence and OD 600. (B) Gene expression rate and growth rate from experimental data. 128
6.8 Mean expression and biomass over different `Reporters on manual build workflow from experimental data`, bars represent SD, N=3. (A) Mean expression rate in MEFL, calibrated units. (B) Mean biomass in particles, calibrated units. 129
6.9 Workflow with manual build execution. LOICA were used to generate simulations from parameters from the literature. Build were simulated using PUDU and performed manually. Experimental data were uploaded to Flapjack. Data were visualised and analysed in Flapjack. 129

6.10 Automated build workflow diagram. LOICA is used at the design stage, then the output from LOICA is formatted using sbol-utilities which works as input for PUDU. PUDU simulates the build and generates metadata. Build is done using PUDU to control lab automation. Experimental data is uploaded to Flapjack where it can be processed and analyzed. 130
6.11 Repressible promoter characterization data and analysis. (A) Fluorescence of GFP and OD raw experimental data. (B) Mean gene expression rate and growth rate of experimental data. 134
6.12 Automated build workflow mean expression and biomass over different `Operators` in the automated build workflow experimental data, bars represent SD, N=3. (A) Mean expression rate in MEFL, calibrated units. (B) Mean biomass in particles, calibrated units. 135
6.13 Workflow execution with automated build. LOICA were used to generate simulations from parameters from the literature. Build were simulated using PUDU and performed with the help of lab automation tools. Experimental data were uploaded to Flapjack. Data were visualised and analysed in Flapjack. 135
6.14 Extended framework diagram. The framework developed in this work can be easily extended with tools connected to standards. 141

7.1 Graphical abstract with software tools. 145

List of tables

1.1 SBOL visual glyphs used in this work. 29

4.1 Part related terminology. 78
4.2 Backbone related terminology 79
4.3 Assembly terminology. 80

6.1 Parts for `Operators` for manual build workflow. 125
6.2 `Operators` for manual build workflow. 125
6.3 Parts for `Reporters` on manual build workflow. 126
6.4 `Reporters` for manual build workflow. 127
6.5 Parts for `Operators` for automated build workflow. 131
6.6 Parts for `Reporters` for automated build workflow. 132
6.7 Transformation solution 1 recipe. 136
6.8 Transformation solution 2 recipe. 137

A.1 Receiver receptor promoters 1. 170
A.2 Receiver receptor promoters 2. 171
A.3 Receiver receptor CDS 1. 172
A.4 Receiver receptor CDS 2. 173
A.5 Receiver sender CDS. 174
A.6 Repressible promoters 1. 175
A.7 Repressible promoters 2. 176
A.8 Repressor CDS 1. 177
A.9 Repressor CDS 2. 178
A.10 Repressor CDS 3. 179
A.11 Repressor CDS 4. 180
A.12 Repressor CDS 5. 181
A.13 Repressor CDS 6. 182
A.14 Reporters 1. 183

A.15 Reporters 2. 184

Nomenclature

MEFL Molecules of Equivalent Fluorescein

Acronyms / Abbreviations

A	Adenine
AI	Artificial Intelligence
API	Application Programming Interface
ATP	adenosine triphosphate
bp	base pairs
C	Cytosine
CDS	Coding Sequence
COVID-19	Coronavirus Disease 2019
DBTL	Design-Build-Test-Learn
DL	Deep learning
DNA	Deoxyribonucleic Acid
dsDNA	double stranded DNA
GDA	Genetic Design Automation
GFP	Green Fluorescent Protein
G	Guanine
GMO	Genetically Modified organism

GTP guanosine triphosphate

GUI Graphical User Interface

iGEM international Genetically Engineered Machine

ISO International Organization for Standardization

LLM Large Language Model

ML Machine learning

mRNA messenger RNA

NGS Next Generation Sequencing

NN Neural Network

ODE Ordinary Differential Equation

PCR Polymerase Chain Reaction

PDE Partial Differential Equation

PEG Polyethylene Glycol

PoPS Polymerases Per Second

PyPI Python Package Index

PYPL PopularitY of Programming Language

RBS Ribosome Binding Sequence

RNAP holoenzyme $E\sigma$

RNAP RNA Polymerase

RNA Ribonucleic Acid

rRNA ribosomal RNA

SBML Systems Biology Markup Language

SBML Systems Biology Markup Language

SBOL Synthetic Biology Open Language

SMRT Single Molecule, Real-Time

SO Operating System

SPICE Stochastic Rate Parameter Inference Using the Cross-Entropy Method

SSA Stochastic Simulation Algorithms

SSB single-strand binding

ssDNA single stranded DNA

Tm melting Temperature

tRNA transfer RNA

TSS Transcription Start Site

T Thymine

TTS Transcription Termination Site

TU Transcriptional Unit

UCF User Constraint File

UI User Interface

UNS Unique Nucleotide Sequence

USD United States Dollar

U Uracil

XDC eXperimental Data Connector

Chapter 1

Introduction

1.1 Synthetic Biology

Synthetic Biology also known as Engineering Biology is an interdisciplinary field that aims to apply the engineering method in biological systems. To this aim foundations for engineering biology like modelling, standardisation and abstraction have to be developed [51], as well as a set of tools that professionals of this area will use. The start of the field can be tracked to two foundational papers in 2000, the "repressilator" [50] and the toggle-switch [58]. The "repressilator" paper describes a synthetic oscillatory network in which three transcriptional regulators repress each other in a ring fashion [50]. The toggle switch paper describes a synthetic bi-stable network in which two transcriptional regulators mutually repress each other [58]. These papers addressed key concepts of computing and electronics using biology, oscillators can keep track of time and toggle switches can work as memory units.

Genetic regulatory mechanisms seen as logic elements were explored by Francois Jacob and Jacques Monod [126, 79], and later on by Ron Weiss and Frances Arnold in 2002 [199], among others. In 2010 Alvin Tamsir, Jeffrey Tabor and Christopher Voigt developed NOR gates encoded in tandem repressible promoters [168]. This was a major breakthrough given that NOR and NAND gates are unique because they are functionally complete, therefore any logic can be created by a combination of NOR gates. The same group in 2016 used combinations of NOR and NOT gates to create a genetic circuit design automation tool, where a Verilog input and a constraint file were used to create circuit diagrams, assign gates, balance constraints to build genetic code, and simulate its performance [123]. These advances motivated more researchers to perform computing in a biological substrate using similar biological logic gates to create genetic circuits. Amazing genetic circuits have been created, for example: perfect adaptation [87], band detector [81], half adder [194], full adder[106], digital display [159] among others.

The main abstraction analogy used in synthetic biology has been the electrical engineering abstraction. One problem that this way of thinking has, is that gene sequence and function are coupled. Synthetic biology can also be thought-out with a software engineering abstraction analogy where gene sequence and function are decoupled. The international genetically engineered machine (iGEM) engineering committee is pushing forward this way of thinking about synthetic biology that could allow the construction of larger genetic networks and provide a framework for a more collaborative work.

Synthetic biology has departed from being a pure academic endeavour and big enterprises have been created around this technology, some examples are Ginkgo, Asimov, IDT, and Twist. Pharmaceuticals like Pfizer and Moderna have used this technology to create vaccines to tackle the coronavirus disease 2019 (COVID-19), a major pandemic that halted societies around the world and killed millions of people caused by the virus SARS-CoV-2. The

synthetic biology global market size was estimated around 10 billions of united states dollar (USD) and is estimated to grow one order of magnitude in 10 years.

1.2 Molecular genetics

To understand synthetic biology, a better understanding of the molecular mechanisms underlying gene expression is needed. In this section, a definition and description of fundamental biological entities and processes is provided.

1.2.1 Gene

The gene is considered the basic unit of inheritance. Genes are made up of deoxyribonucleic acid (DNA) passed down from parents to offspring and contain the information needed to specify physical and biological traits. Most genes encode specific proteins, segments of proteins, or non coding ribonucleic acid (RNA) with a certain function or interaction. An important distinction between genotype and phenotype, the genotype is the collection of inherited DNA molecules, while the phenotype is the expression of a genotype given a biochemical context or environment [140]. Other ways of inheritance such as setting the initial biochemical context [139, 57] are out of the scope of this work.

1.2.2 Deoxyribonucleic Acid

The DNA is formed by 4 nitrogenous bases of two types: adenine (A) and guanine (G) are purines (two ring heterocyclic compound), and cytosine (C) and thymine (T) are pyrimidines (one ring heterocyclic compound). Two nitrogenous bases can form a phosphodiester bond between the 5' carbon of the sugar from one nitrogenous base and the 3' carbon on the other base. In 1948 Erwin Chargaff found that the ratio of purines and pyrimidines is 1:1 in double stranded DNA molecules [182]. This discovery contributed to determining the molecular structure of DNA by James Watson and Francis Crick in 1953 [189]. They reported that DNA is a right-handed double strand helix, with the two strands connected in an antiparallel way by hydrogen bonds (Figure 1.3 A). The A bases always pair with Ts, and Cs always pair with Gs, forming base pairs (bp) consistent with Chargaff's rules [31, 105] (Figure 1.3 B,C). Our knowledge about DNA has been growing over time, and although most DNA found in nature corresponds to the Watson and Crick structure other structures and alternative base pairing have been described, increasing our understanding about DNA structure and function [174].

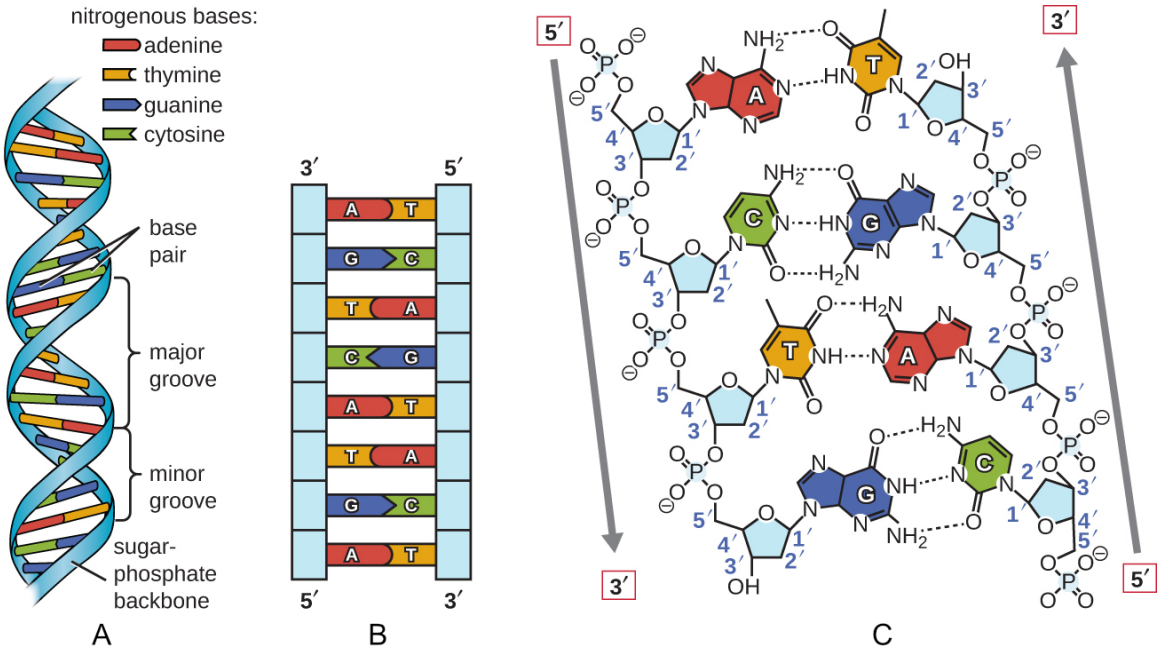


Fig. 1.1 Watson and Crick proposed the double helix model for DNA. (A) The sugar-phosphate backbones are on the outside of the double helix and purines and pyrimidines form the "rungs" of the DNA helix ladder. (B) The two DNA strands are antiparallel to each other. (C) The direction of each strand is identified by numbering the carbons (1 through 5) in each sugar molecule. The 5' end is the one where carbon 5 is not bound to another nucleotide; the 3' end is the one where carbon 3 is not bound to another nucleotide and has a free hydroxyl group. Extracted from penStax-CNX.

1.2.3 Replication

New DNA molecules can be produced in cells by a process called replication which is different for prokaryotic [108] and eukaryotic [13] cells. Replication in prokaryotic cells can be divided into 3 stages: initiation, elongation, and arrest. Initiation in *E. coli* chromosomal DNA replication starts bidirectionally at a specific sequence oriC. The minimal oriC is 245 bp and is highly conserved among enterobacteriaceae with two main features. It contains four copies of the same sequence in the form of two inverted repeats denoted R1-R4 that binds DnaA and directly upstream a series of three 13mers with a high A-T content for localised denaturation, forming an initial complex. In the presence of adenosine triphosphate (ATP), DnaA forms an open complex leaving the A-T rich zone sensitive to cleavage by the P1 endonuclease. Then, DnaA guides the DnaB-DnaC complex in solution to its place between the strands of the denaturation bubble, forming the prepriming complex. If provided with single-strand binding protein (SSB) and DNA gyrase the DnaB will act to unwind the DNA template. In addition, DnaG and DNA Pol II holoenzyme are added, and the

complete replication fork is formed. Elongation occurs at the replication fork where the DNA polymerase is engaged to the leading strand and its counterpart is engaged in the lagging-strand. The enzyme responsible for the unwinding is the DNA helicase and for priming the primase, which attract each other, forming the primosome complex. One RNA primer is needed for the leading strand as the DNA polymerase has a 5'-3' activity. On the other strand RNA primers, called Okazaki-fragments, have to be constantly synthesised. Once the replication fork enters the terminus region it is arrested and destabilised. This zone is located at approximately 180 degrees from the oriC and has six terminus region sites symmetrically positioned, although the requirements for the *in vivo* termination mechanism are not clear, cells without the terminus region are viable but produce many nonviable cells and exhibit filamentation. In plasmids the origin of replication is a DNA sequence that controls its replication and therefore plasmid copy number.

1.2.4 Transcription

Transcription is the process where DNA is used as a template to synthesise RNA. RNA is similar to the DNA as both ribonucleases differ just in an oxygen group. RNA is generally formed by 4 nitrogenous bases of two types: A and G are purines, and C and uracil (U) are pyrimidines. In *E. coli*, RNA is synthesised by the RNA polymerase (RNAP), RNA can have different functions, the messenger RNA (mRNA) is a transcript that encodes instructions for protein synthesis.

The first step of transcription in prokaryotes is the initiation, characterised by the formation of the RNAP holoenzyme (Eσ), a molecular complex composed of the core RNAP plus a σ factor, which is capable of initiating gene transcription at specific DNA positions[22]. The σ factor interacts with different promoter elements to position the Eσ close to the transcription start site (TSS) to unwind the DNA. In general bacteria rely on different σ factors to guide the Eσ to different groups of promoters in response to changes in the environment. [103] The σ factors are classified in two families, σ^{70} and σ^{54}. The σ^{54} family has one member and the σ^{70} has several. The promoter is the contiguous DNA sequence essential for the specific initiation of transcription at a defined location in a DNA molecule, although this location might not be one single base. It is recognized by a specific Eσ, and this recognition is not necessarily autonomous.

The second step is elongation, a process where the RNAP in the transcription bubble moves from 5' to 3' synthesising RNA with a copy of the information in DNA [98].

The last step is termination, in bacteria termination can be Rho dependent or Rho independent (intrinsic) [2]. Rho dependent termination relies on the binding of a transcription termination factor like Rho, Tau or nusA for example. Rho independent termination relies on

the RNA structure to destabilise the RNAP. The terminator is a contiguous DNA sequence essential for transcription termination.

The transcriptional unit is the contiguous DNA sequence that ranges from the promoter to the terminator and has the capability to transcribe a defined RNA. The TSS and transcription termination site (TTS) vary in the same transcriptional unit producing a population of similar genetic products. When the transcription rate is just affected by the properties of the promoter itself and not by components outside of the normal transcription machinery we call it a constitutive promoter and the produced RNA/protein is under constitutive expression When the transcription rate is affected by external factors it is referred to as regulated expression. The regulation of gene expression in bacteria occurs predominantly at the level of transcription. The σ factors acts as transcriptional regulators by changing the affinity of the Eσ to the promoter. In addition to σ factors, different families of regulatory proteins are involved in the regulation of gene expression at the transcriptional level. Negative regulators (repressors) usually bind to promoters and block the access of the RNAP to initiate transcription. On the other hand, positive regulators (activators) usually bind in the upstream regions of promoters and allow efficient transcriptional activation upon association with RNAP [96].

1.2.5 Translation

Translation is the process where RNA is used as a template to synthesise a protein. Proteins are polymers composed of aminoacids, although there are more than 500 of them, essentially 20 are usually incorporated into proteins [34]. Proteins are synthesised by the ribosome, a macromolecule formed by ribosomal RNA(rRNA) and proteins. The translation mechanisms vary from eukaryotic to prokaryotic organisms [44, 142]. In prokaryotes like *E. coli*, translation has 3 steps, initiation, elongation and termination [142].

The first step in translation is initiation, where the ribosome 50S and 30S subunits are separated by IF3 which binds to the 30s subunit increasing its affinity for IF1, initiator tRNA (transfer RNA) with IF2, and for RNA [62]. The activated ribosome 30s subunit can bind to the RNA directly in a site specific manner in the start codons close to a Shine-Dalgarno sequence and then recruit the IF1 and the initiator tRNA or recruit IF1 and the initiator tRNA and then bind to the RNA in the same sequence. Once this complex is bound to the RNA it performs a conformational change that binds the ribosome 50S subunit removing IF1 and IF3. Finally, IF2 catalysed the guanosine triphosphate (GTP) and is released leaving the ribosome assembled with the initiator tRNA.

The second step is elongation. The ribosome has three tRNA binding sites, the aminoacyl site (site A) and the peptidyl site (site P) and the exit site (site E), and one factor binding

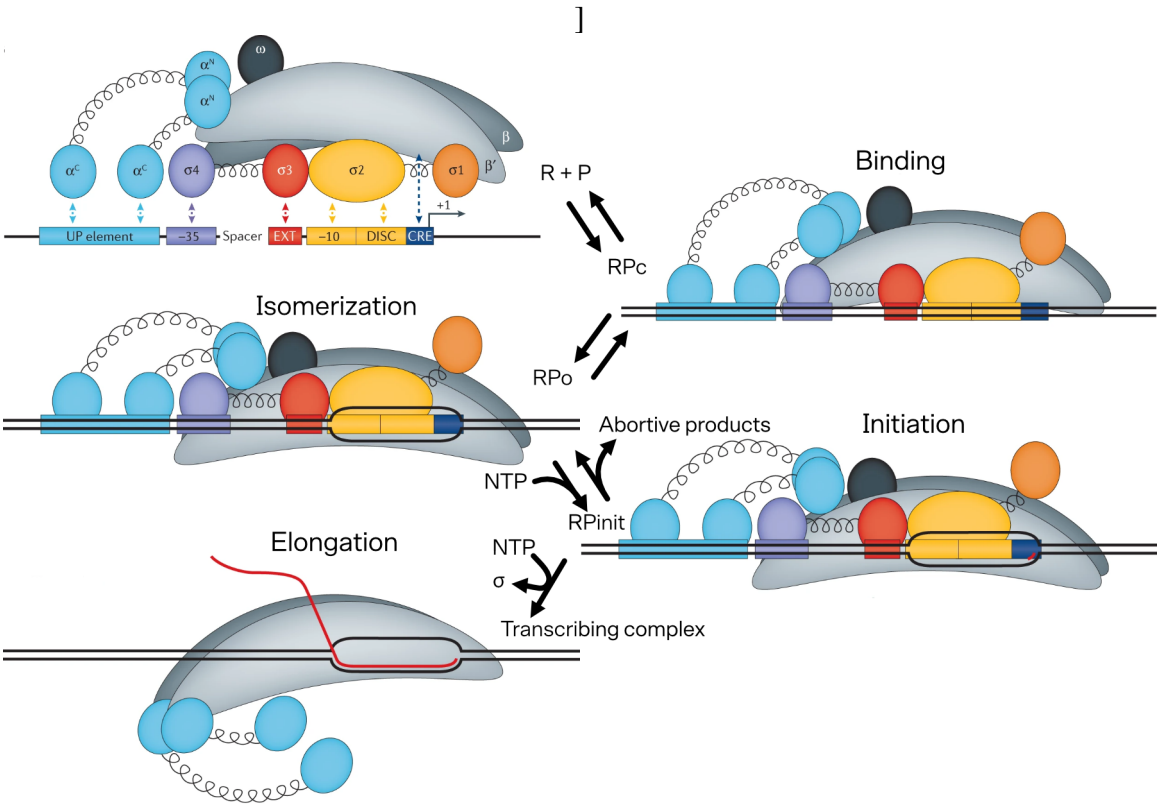


Fig. 1.2 Bacterial transcription. (A) Bacterial RNA polymerase (RNAP) is a multisubunit enzyme. The core RNAP is composed of five subunits: $\alpha 2$ (cyan bubbles), β and β' (grey) and ω (black). The core RNAP contains the active site that catalyses the formation of the phosphodiester bond of nascent RNA. The RNAP α subunits interact with the upstream promoter (UP) element, which consists of two distinct subsites located upstream of the 35 element [53]. The RNAP core enzyme interacts in a sequence-specific manner with the template-strand positions –4 to +2, which constitute the core recognition element (CRE) (navy blue) [200]. To form the holoenzyme (Eσ), RNAP is associated with a σ factor. The σ70-related factors contain up to four functional domains (σ1–4; the spirals represent linkers between the domains). The σ2 domain recognizes and interacts with the 10 element, and the σ4 domain interacts with the 35 element. The extended 10 element interacts with σ3; this interaction is crucial in promoters whose 35 and 10 elements show a poor match to consensus sequences [85]. Some promoters have a discriminator (DISC), which is recognized and interacts with the σ2 domain [73, 83]. The σ70 factor bound to the non-template strand captures the 10 region in an open complex and allows the single-strand template DNA to enter the active site. (B) The RNAP holoenzyme (R) and promoter DNA (P) interact to form the closed complex (RPc). The duplex DNA around the transcription start site is separated (represented by a 'bubble' in the DNA that is bound by the Eσ) in order to form the open complex (RPo). The initiating complex (RPinit) is formed and begins synthesis of the RNA chain (shown as a small red line), directed by the DNA template strand. In initiation of the synthesis of RNA, a phosphodiester bond is formed between the initiating and adjacent phosphodiester nucleoside triphosphates (NTPs). The final stage of transcription initiation is elongation. In the elongation phase, the RNA chain length increases, shown as a solid red line. Part a adapted from ref. [146], CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/). Part b adapted from ref. [22], Springer Nature Limited. Illustration adapted from ref. [114].

site (site F) [138]. From the RNA perspective from 5' to 3' the sites are located in the order site E,P,A,F. Starting in the position where the ribosome has a tRNA bound to the nascent peptide in the site P, a new amino acid can be bound in the site A. The EF1A recruits a tRNA and binds to the site F, then catalyses GTP undergoing a conformational change that places the tRNA in the site A and unbinds from the site F to recruit a new tRNA. The peptidyl transfer is the process where the new amino acid from the tRNA in the site A attacks the nascent polypeptide C-terminal stabilised by the ribosome large subunit adenine which provides the hydrogen to create a hydroxyl end in the tRNA on site P leaving the nascent polypeptide in bound to the tRNA on site A. Then, EF2 binds to the site F and undergoes a conformational change that pushes the tRNAs translocating the ribosome, leaving the empty tRNA in the site E and the tRNA with the nascent polypeptide on site P. Then EF2 catalysed GTP undergoing a conformational change that releases the tRNA on site E and unbinds from the site F. Now the ribosome is in the same condition as it started and this process repeats during the elongation.

The last step is termination. Although the scientific community agrees that during the termination the ribosome unbinds from the RNA there are two proposed mechanisms [119]. In the first, when a ribosome reaches a stop codon, it binds RF1, then recruits RF3, which catalyses GTP and the ribosome unbinds being separated the subunits by EF-G and RF4. In the second, the only difference is that the ribosome unbinds as a whole instead as separated subunits. After termination the protein is released and matures to obtain its 3D conformation. Proteins are the main functional and structural macromolecules in cells.

1.3 Foundational technologies

As the name of the field suggest the name synthetic biology is related to the capacity to create synthetic DNA. In this section I describe the major technologies that allows to write, read, copy and paste DNA, as well as how to insert it into a host and measure its function.

1.3.1 Synthesis

Synthesis is the ability to write DNA, this means to produce arbitrary DNA fragments.

Chemical DNA synthesis using phosphoramidite was developed by Marvin Caruthers in the 1970s where protected 2'-deoxynucleoside phosphoramidites with different bases are concatenated to a 3' to 5' nascent DNA strand in sequence [28]. Because these bases are chemically protected only one nucleotide can be added each time, then the last added base needs to be deprotected or activated to continue with the reaction [29]. This technique is the

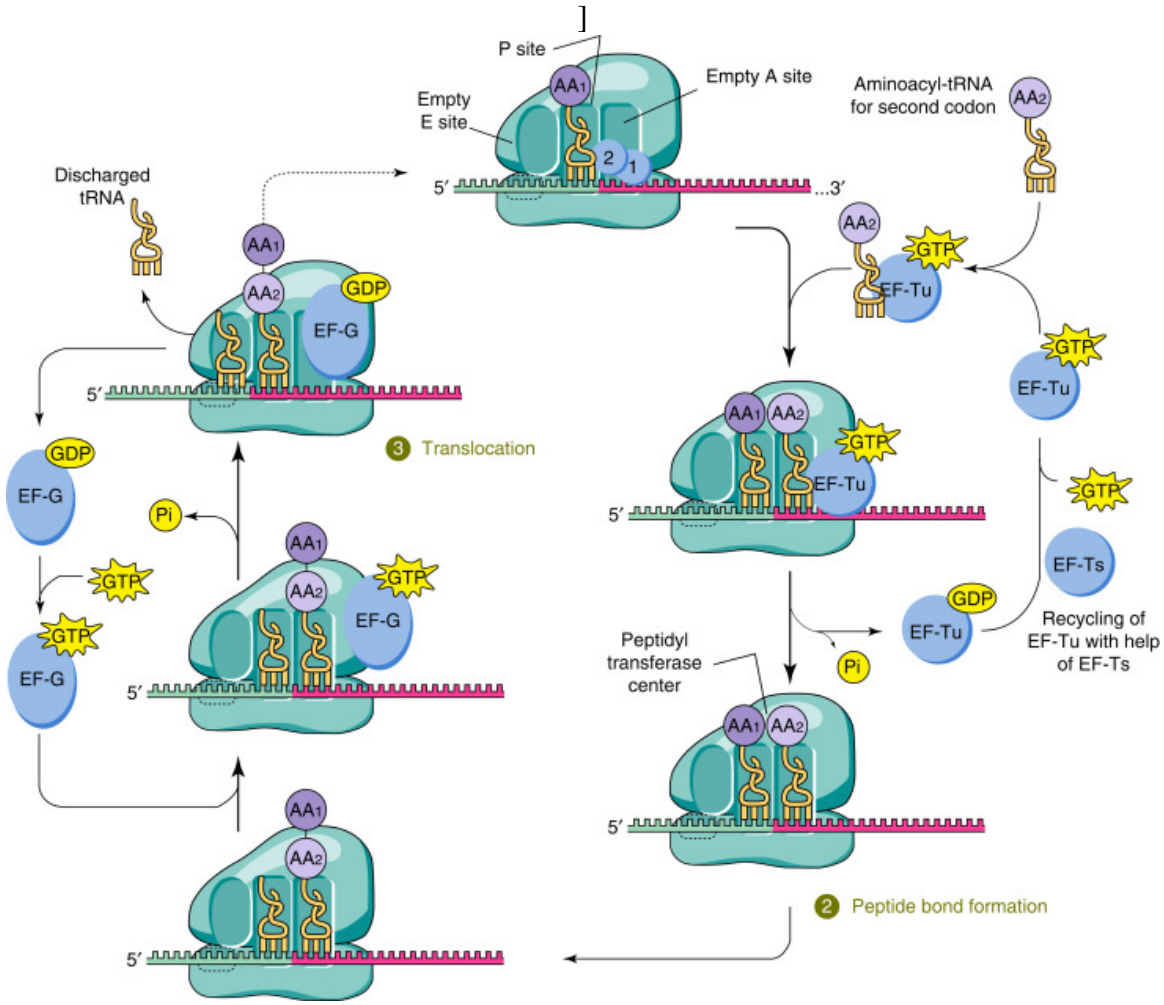


Fig. 1.3 Translation complex. A cognate aminoacyl-tRNA bound to EF-Tu binds to its cognate codon of the mRNA in the ribosome at the acceptor site (A site), adjacent to the tRNA in the peptidyl site (P site), which is bound to the nascent polypeptide (1). Decoding occurs on the 30S subunit of the ribosome, GTP is hydrolysed by EF-Tu, which leads to release of aminoacyl-tRNA into the A site if there is correct codon–anticodon pairing. Subsequent peptide bond formation between the nascent polypeptide in the P site and the amino acid attached to the tRNA in the A site results in a peptidyl-tRNA in the A site that is one amino acid longer (2). After translocation of the ribosome, the tRNA attached to the nascent polypeptide is moved to the P site and a new aminoacyl-tRNA is delivered to the empty A site (3). Extracted from [18]

most used by DNA synthesis companies like IDT Technologies and Twist Bioscience. In this work we use this technique through IDT Technologies to create double stranded DNA fragments or gBlocks.

1.3.2 Amplification

Amplification is the ability to copy DNA, this means to produce millions to billions copies of a determined segment of DNA.

In 1955, Arthur Kornberg and associates isolated DNA polymerase, an enzyme with many functions including DNA replication and repair, and was awarded with the Nobel prize in 1959 [97, 100]. Later on, in the early 1960's Gobind Khorana explored the use of synthetic oligonucleotides as primers for the DNA polymerase, earning the Nobel prize in 1968 [88]. A few years later in 1971 Kjell Kleppe, part of Khorana's group, reported the repair replication of a specific sequence of DNA using a system with two primers, and described primer properties needed for this process. [90]. Using these principles Kary Mullis invented in 1983 an *in vitro* technique to replicate and amplify DNA sequences and called it polymerase chain reaction (PCR) [118]. The PCR reaction consists of 3 basic steps that can be looped. First in the denaturation step, the templates of double-stranded DNA (dsDNA) are separated by heat into two single-stranded DNAs (ssDNA) around 95 Celsius degrees. Then in the annealing step, temperature is reduced approximately 5 Celsius degrees below the melting temperature (Tm), to allow primers to bind to the ssDNA. The Tm is the temperature at which half of the DNA fragments are dissociated and depends on the GC content on the sequence. Finally in the extension step, temperature is raised again to approximately 72 Celsius degrees which is optimum for the DNA polymerase activity. The DNA polymerase binds to primer-template complexes extending the template and creating two dsDNA. This process can be looped creating an exponential increase in the amount of DNA copies. Although at the beginning on each cycle more DNA polymerase was added on each step as it was denatured with the DNA, this hindrance was overcomed by using a thermo stable DNA polymerases coming from extremophiles like *Thermus aquaticus* that resisted this process [35].

1.3.3 Sequencing

Sequencing is the ability to read DNA, this means to know the nitrogenous bases sequence that composes a DNA fragment.

In 1976, Allan Maxam and Walter Gilbert developed a chemical method for sequencing up to 100 bp based on radioactive labels, selective cleavage, and electrophoresis of DNA fragments [109]. One year later in 1977, Frederick Sanger developed a chain termination

method for sequencing approximately 600 bp, based on the random incorporation of fluorescent labelled dideoxynucleotides that inhibit elongation during DNA replication and posterior electrophoresis of these DNA fragments [151]. Sanger sequencing is still used and was used in this work for sequencing inserts, showing its relevance in the field. In the mid 1990s new technologies for sequencing were developed, for example Roche developed pyrosequencing and Illumina reversible dye terminator [48, 14]. All these new technologies had in common that they are massive parallel sequencing, and were called next generation sequencing (NGS) or second generation sequencing [64, 102]. They did not increase the read length, but they parallelized and automated the process allowing sequencing on the scale of 1000 Gbp per run.

Finally, since 2008-2009 a new set of technologies are being developed, for example Pacific Biosciences developed Single Molecule, Real-Time (SMRT) [141] sequencing and Oxford developed nanopore [186]. They focused on increasing the read length, and were called long-read sequencing or third generation sequencing. Oxford Nanopore measures changes in the electrical field surrounding a pore as DNA passes through it, enabling it to produce reads in the order of 10 Gbp and sequencing about 14 Tbp per run [80]. Oxford Nanopore was used in this work for whole plasmid sequencing.

1.3.4 Assembly

Assembly is the ability to paste DNA, this means to concatenate DNA sequences in a determined order.

Restriction enzyme assembly is a method based on type II restriction enzymes where the enzymes cut on its recognition site, therefore two DNAs digested by the same restriction enzyme can be joined. To simplify this assembly BioBricks was proposed [93]. In BioBricks the DNA parts, such as promoter and CDS are flanked by specific restriction enzymes. Although this method is still used, it is not widely adopted as it needs the use of too many restriction enzymes. Gibson assembly is a method based in PCR where blunt DNA fragments share a sequence on their flanking regions allowing a 5' exonuclease to digests both 5' ends leaving complementary base pairs or sticky ends [59]. These sticky ends anneal and work as primers for PCR completing the missing bases and then a ligase merges the DNA fragments. To simplify Gibson assembly a set of unique nucleotide sequences (UNSs) were developed to flank and join DNA parts [173]. Although this method does not produce assembly scars, PCR based methods can produce mutations. These mutations can be useful for directed evolution but bad for normal rational design of DNA sequences. Golden gate is a method based on type IIS restriction enzymes where these enzymes cut a few bases downstream from their recognition site, allowing for the engineering of the fusion sites [52]. Fragments with base

complementarity anneal and a ligase close the nicks. Several assembly standard syntaxes, conventions on flanking fusion sites, have been created exploiting the fact that the fusion site can be an arbitrary sequence. Examples of assembly standard syntaxes are: Modular Cloning (MoClo) [78], PhytoBricks [26] and Loop [135]. In this work I use a combination of these three standards.

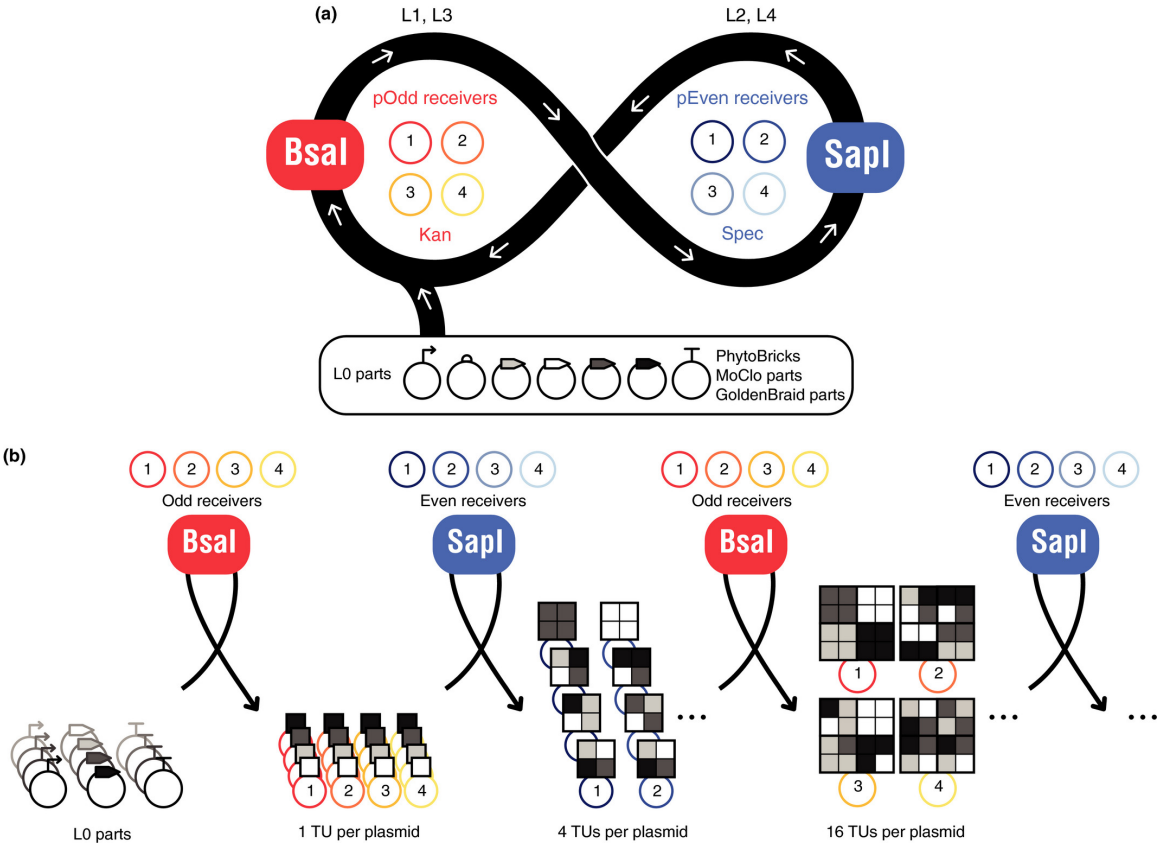


Fig. 1.4 Overview of Loop assembly. (a) Loop assembly workflow. L0 parts are assembled to L1 transcription units (TUs) into one pOdd receiver by BsaI-mediated Type IIS assembly. L1 TUs are assembled to L2 multi-TUs into one pEven receiver by SapI-mediated Type IIS assembly. This workflow is then repeated for higher level assemblies. Only four odd level and four even level receiver plasmids are required for Loop assembly. (b) Combinatorial and exponential assembly. L0 parts can be assembled to L1 TUs into any of the four positions of odd receivers. Genetic modules can easily be swapped in each TU arrangement and receiver position. L1 TUs can then be assembled into L2 multi-TUs with variable combinations of the L1 TUs, also into any of the four positions of the even receivers. Each round of assembly generates four assembled plasmids, and consequent rounds of assembly increase the number of TUs by a factor of four, leading to an exponential increase in TU number. Extracted from ref. [134].

1.3.5 Transformation

Transformation is the ability to install DNA in a chassis, this means to introduce DNA into a host organism and set it up as a stable genetic component. Transformation was first reported by Fred Griffith in 1928 [66] and is one of the multiple ways in which a bacteria can incorporate foreign DNA, where transformation is the incorporation of DNA from the environment, conjugation is the incorporation of DNA from another bacteria and transduction is the incorporation of DNA from a virus. An organism is considered competent if it is capable of incorporating and maintaining external DNA. In 1972, Akira Taketo reported that the competence of calcium treated *Escerichia coli* (*E. coli*) was increased[167]. The physicochemical process of chemical transformation of *E. coli* using divalent cations and heat shock was developed by Roy Curtis III in his research to construct safer bacterial host strains for recombinant DNA research [38]. Over time this method has been refined and became a standard in molecular biology and was used in this work. Another chemical transformation method using polyethylene glycol (PEG) was developed by Robert Klebe in 1983 for bacteria and yeast [89]. Then in 1988 William J. Dower developed electroporation for *E. coli* [45]. This allowed the transformation of organisms where chemical transformation does not work.

1.3.6 Reporters

Reporters of gene expression are a source of information for analysis, as they produce a measurable signal. One of the classic reporters is the use of the *E. coli* LacZ gene which produces a blue product by the β-galactosidase enzyme cleaving X-gal [24]. Fluorescent proteins are proteins or genetic products that emit fluorescence when excited with light at a certain wavelength [157, 37]. Fluorescent components were first noticed in *Aequorea victoria* jellyfish by Davenport and Nicol in 1955 [41]. Later on, Osamu Shimomura realised that the fluorophore was indeed a protein and isolated it in 1962. The complete sequence of 238 amino acids of the *A. victoria* green fluorescent protein (GFP) was reported by Prasher in 1992 [137]. This led to the recombinant expression of GFP in *E. coli* by Inouye and Tsuji in 1994, and in plants by Haseloff in 1995 [77, 71]. Chromoproteins on the other hand express a colour that can be seen and detected by the naked eye on white light [101]. Other than proteins, aptamers are RNAs that produce fluorescence [163, 110].

1.3.7 Cloning

The term cloning describes a number of different processes that can be used to produce genetically identical copies of a biological entity [148]. The copied material, which has the

same genetic material as the original, is referred to as a clone. Researchers have cloned a wide range of biological materials, including genes, cells, tissues and even entire organisms, such as a sheep [84]. In nature, some plants and single cell organisms, such as bacteria, produce genetically identical offspring through asexual reproduction [43]. In asexual reproduction, a new individual is generated from a copy of a single or group of cells from the parent organism. Researchers routinely use cloning techniques to make copies of genes that they wish to study. The procedure consists of inserting a gene from one organism, often referred to as "foreign DNA," into the genetic material of a carrier called a vector [115]. Examples of vectors include bacteria, yeast cells, viruses or plasmids, which are small DNA circles carried by bacteria [195]. After the gene is inserted, the vector is placed in laboratory conditions that prompt it to multiply, resulting in the gene being copied many times over. In synthetic biology cloning techniques are used to assemble DNA with a new synthesised part then transform it into a bacterial vector and select colonies with the right assembly by production of a reporter or DNA sequencing [15]. When the DNA sequence on the colony is confirmed it can be grown and stored as stock. These stocks are then used to grow the bacteria containing the designed DNA and extract it for its use in another assembly.

1.4 Genetic network engineering

1.4.1 Genetic networks

Genetic networks are composed of genetic elements like genes or transcriptional units (TU) as nodes. TUs are DNA fragments that can transcribe a segment of their sequence into RNA which in turn can be translated into proteins. Gene expression is the number of RNA molecules produced by a DNA fragment, in a determined context and time. Depending on the organism context, RNA is translated into proteins which can be enzymes or fluorescent proteins. Enzymes can produce chemicals that can be measured using colorimetric assays and fluorescent proteins emits light that can be measured using fluorescence assays. Both assays can be used as a proxy of gene expression as the amount of signal depends on the number of proteins and the number of proteins depends on the number of mRNAs. RNA and proteins produced by TUs can be transcriptional regulators and regulate their own or another TU creating edges between TU nodes.

These networks control the production rate of molecules over time and together with the degradation machinery constitute the basis for the control of the dynamics of these molecules. Controlling the concentration of biochemical products over time is important in many areas of biotechnology and bioindustry. This approach has been used for metabolic

engineering, which looks for the improvement and design of cells, focused on the use of new substrates, production of novel compounds, increased productivity and increased cellular robustness [124]. Control theory has been used to engineer genetic networks to create, for example, controllers for robust perfect adaptation and dynamic performance [54].

The study of genetic networks dynamics is relevant to understanding spatio-temporal pattern formation. It has been demonstrated that temporal oscillations in single cells can lead to the formation of ring patterns in bacterial colonies [136, 197]. Genetic network dynamics, for example oscillators and feedback loops, can produce perturbations, such as symmetry-breaking, driving morphogenetic processes that determine key steps in embryogenesis [184]. Applications of synthetic genetic networks have been used for drug delivery, and a better understanding of their dynamics could lead to better dosage systems, and avoid errors in its function, such as logic "glitches" [32, 55].

1.4.2 Abstraction

Abstraction is a good tool to deal with complexity and what is more complex than a biological system. In synthetic biology the first abstraction emerged from the registry of standard biological parts, which linked DNA sequences to functions, that can be inside a TU, i.e. promoter, ribosome binding site (RBS), coding sequence (CDS) and terminator or parts composed of one or more TUs, i.e. composite and inverter [51]. To represent one or more TUs synthetic biologists have adopted an abstraction based on electrical engineering. Using this design abstraction, a repressible TU can be abstracted as a NOT gate where the repressor concentration is the input and the output is the protein production. A composition of one or more of these parts based on TUs creates a genetic circuit.

1.4.3 Standardisation

In engineering, standards are developed by experts and published at the international organization for standardization (ISO) and can be thought of as a formula that describes the best way of doing something. They could be about making a product, managing a process, delivering a service or supplying materials, covering a huge range of activities. Some ISO examples are: ISO/IEC 27000 family for information security management, ISO 8601 for date and time format, ISO 4015:2022 for hexagon head bolts. Standards are the distilled wisdom of people with expertise in their subject matter and who know the needs of the organisations they represent such as manufacturers, sellers, buyers, customers, trade associations, users or regulators. Standards in synthetic biology are the agreement of the community in a design abstraction, assembly method or parts basic features. Sequence features were adopted from

the registry of standard biological parts [51]. After standardising the parts, or sequence features, synthetic biologists had to standardise the way to join these parts. Assembly methods like Biobrick and Golden Gate were used in segments of DNA that can be joined together in a specific way and order dictated by annealing of DNA strands created on digestion. As with type IIS restriction enzymes a 4 bp overhang is created independent of the recognition site, these can be chosen arbitrarily or engineered and standardised to contain certain DNA sequence features. One of those standards is Phytobricks that created an assembly standard for plants and bacteria parts [26] which is compatible with MoClo [190] both used at iGEM and therefore with lots of described parts. Loop assembly is compatible with Phytobricks and enables the hierarchical assembly of parts into TUs and then composing TUs iteratively.

1.4.4 Decoupling

Abstraction and standardisation enables the decoupling between different levels of work. Decoupling allows researchers to work and specialise at one of these levels with basic understanding of the others. The use of sequence features to describe DNA parts and the capability of composing them through assembly or synthesis enabled the decoupling of tasks. This decoupling would be for example between; scientists doing research about the DNA function and structure with the expertise to create DNA parts, and the scientists composing those parts to control gene expression creating TUs and composing TUs to create genetic circuits. The first proposed abstraction hierarchy has 4 levels [51]. The DNA level is in charge of creating DNA, i.e. by synthesis or assembly. The Part level is in charge of designing parts or sequences of DNA, i.e. creating a repressible promoter or a CDS that expresses a protein that binds to a specific site. The Device level is in charge of composing parts design TUs where with polymerases per second (PoPS) as input and output, i.e. a NOT encoded in a repressible TU or a source encoded in a constitutive TU [5]. The Systems level is in charge of composing devices to create systems like a repressilator, a set of 3 NOT gates repressing each other in a ring fashion. A set of standardised interfaces would allow the decoupling of tasks, advancing synthetic biology towards a mature engineering discipline.

1.5 Synthetic biology open language

The abstraction developed by the registry of standard biological parts [51] evolved creating the synthetic biology open language (SBOL) to describe biological designs [112, 23]. The synthetic biology community created SBOL to work as a standard among synthetic biologists and related areas to represent genetic designs. SBOL is a standardised format for the

electronic exchange of information on the structural and functional aspects of biological designs. SBOL has two main components, SBOL visual and SBOL data model. These two standards are entangled, SBOL visual focuses in represent sequences as glyphs to be used for informal DNA design in a whiteboard or for its communication in a scientific journal [6]; SBOL data standard focuses on metadata capture in a machine readable format. Although SBOL visual has been adopted by most synthetic biologists (Table 1.1), the data standard is not widely adopted by the community making it difficult to collaborate, reproduce results and to build knowledge collectively. A possible cause is the steep learning curve needed to represent systems properly. Although the first tools of the SBOL community were focused on libraries for developers and tools that facilitate the use of SBOL, now the community is consolidating developers tools and directing the efforts to produce software tools that uses SBOL under the hood.

1.6 Engineering method

The design-build-test-learn (DBTL) cycle is central to engineering disciplines and each phase requires appropriate tools, standards, and workflows, which are still in development.

Software tools that cover different stages of the DBTL cycle have been developed automating and formalizing it [94, 120, 150]. Tool boxes with software covering multiple stages of the DBTL cycle, have been developed as well [95, 17]. The lack of standardization in these tools renders them incompatible with other tools and difficult to connect.

SBOL is an open standard for the representation of *in-silico* biological designs that covers the DBTL cycle and has attracted a community of developers that have produced an ecosystem of software tools tackling different parts of the DBTL cycle [112, 170, 72, 188].

There is still a need for software tools covering dynamic systems, model parametrization from raw data, data management, programmatic design and protocol automation [74].

1.6.1 Design

The design stage is the process throughout which a researcher encodes a feature or function in a sequence of DNA. Depending on the abstraction level, designing could involve the selection of parts to be used or the general genetic network architecture. Modelling is key to the DBTL cycle and is essential to the design and learning stages, given that a model states a well-defined hypothesis about the system operation. This hypothesis or engineering goal shapes the rest of the cycle stages build, test and learn to improve future designs. Abstraction enables the construction and analysis of models based on components, devices, and systems

that can be used to compose genetic networks and derive their DNA sequences. It is the basis for genetic design automation (GDA), which can accelerate and automate the genetic network design process by compiling models into DNA sequences. In general there are two main modelling methods, deterministic and stochastic methods. Ordinary differential equations (ODE) are used to model dynamical systems in a deterministic fashion. These equations depend on a single variable and the unknowns could be one or more functions and involve the derivative of those functions. Stochastic simulation algorithms (SSA) are used to model dynamical systems in a stochastic fashion. The Gillespie algorithm is an SSA to simulate chemical reactions [60, 61]. In order for GDA to proceed in a rational way, the abstract elements of genetic networks must be accessible to characterization, allowing parameterization of models of their operation and interactions. Data driven modelling, which is based on artificial intelligence (AI) and machine learning (ML) methods, requires characterization data from genetic networks to predict the characterization data of novel genetic networks [162]. Some examples of data driven approaches are: neural networks (NNs), deep learning (DL), fuzzy rule-based systems and genetic algorithms.

1.6.2 Build

The build stage is the process through which a researcher takes a design encoded in a sequence and creates a physical implementation of that DNA. In this stage DNA synthesis and assembly are key, synthesis is usually provided as a service and assembly can be done manually at a lab with Containment Level 1 (Biosafety Level 1), and molecular biology equipment and reagents. Once a DNA is in a plasmid it can be inserted into a host organism through transformation creating a genetically engineered organism (GMO). Liquid handling robots can be used to automate protocols such as assembly and transformation to standardise it as a series of robot liquid movement steps.

1.6.3 Test

The test stage is the process through which a researcher tests the designed function or feature. This usually involves the creation of samples containing the GMO and obtaining measurements from it. The four most used test devices in synthetic biology are the plate reader, flow cytometer, microscope and DNA sequencing. Plate readers as can be inferred from its name obtain measurements from a plate, the plate can differ on material and number of wells, i.e. flat glass bottom 96 well plate. The measurements obtained may vary but it usually includes absorbance and fluorescence. It can obtain kinetic measurements of a population of cells, as its readings are non destructive and have to be performed on the whole

sample with low effects on its constitution. Flow cytometers obtain data taking samples with cells that can be in a tube or in a well plate and passing them inside channels which at some point arrange them to pass one by one though lasers to take individual cell measurements, giving information about the distribution of the behaviour across the population. Microscopes obtain images providing spatial resolution, microscopy can be set with different setups capturing for example fluorescence or taking images of cells in microfluidic devices over time. DNA sequencing can also be used on cDNA to test RNA levels giving information about how much genes are transcribed. Calibration is essential in engineering disciplines, improving the reproducibility of results and collaboration. The common practice in science of sharing data in arbitrary units is a hindrance to the engineering method. Fluorescence from green fluorescent proteins produced by bacteria can be calibrated with fluorescein fluorescent dye to obtain molecules of equivalent fluorescein (MEFL)[10]. This approach was expanded to red and blue using sulforhodamine 101 and cascade blue respectively [11]. Recently, another alternative for calibration using fluorescent protein absolute quantification [36] has been proposed and is being developed by the iGEM engineering committee. Absorbance has been calibrated using silica microspheres, to obtain measurements in particle units [9]. This increases reproducibility and use of published data in further analysis.

1.6.4 Learn

The learn stage is the process through which a researcher uses the data produced at the test stage to gain insights about the designed biological system. During learning the experimental data and the meta data are combined to perform analysis. These analyses can involve gene expression rate calculation or model parameterization for example.

In the prototypical case, data from the test stage is in Excel format, or in an instrument related format. The data is copy pasted or imported into a software tool where the analysis is performed. Software tools have mainly two User Interfaces (UI) a graphical user interface (GUI) or a programmatic interface. Using a GUI the user can copy and paste, drag and drop and edit input fields for example, this approach is optimised to be used by a human. Using a programmatic user interface allows the automation of tasks through code, this approach is optimised to be used by machines and enables the management of large volumes of data.

Programming languages offer automation, processing and visualisation tools that are useful for synthetic biologists. Python is a high-level, general-purpose and dynamic programming language that emphasizes code readability. It is open-source, is the most popular programming language in the popularity of programming languages (PYPL) index and it is usually taught to engineering students due to its simplicity. A lot of the apps that people

frequently use depend on Python, such as Netflix, Google, Facebook, Amazon, PayPal and Uber, even NASA uses it.

Given that it is open-source, communities built packages targeted to their needs. Python has a comprehensive set of packages to handle data science tasks in synthetic biology. Numerical operations are improved using NumPy [70], work with tabular data is aided with Pandas [111] and scientific and technical computing can be performed with SciPy [181]. Visualization tools such as Matplotlib [75] and Seaborn[187] provide a lot of ways to inspect data. NetworkX library is used for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks [68]. ML algorithms can be used from scikit-learn [130] and tensor manipulation for DL can be performed with TensorFlow[1], PyTorch[128] and Keras[33] for example. These tools with others from the Python ecosystem are used for data analysis at research labs.

Flapjack is a data management and analysis tool that connects the Test and Learn stages [196]. It works as a database accessible through an application programming interface (API) in the back-end, that connects to a front-end with a GUI. The back-end can also be accessed and operated using the pyFlapjack Python package. This tool combines the simplicity of using a GUI with the automation provided by accessing data with programming languages. Experimental data in Flapjack is stored in a tidy format which allows it to be easily manipulated, modelled and visualised in a programmatic fashion [192].

In this work, I used and contributed to the development of Flapjack [183]. Its data model stores relevant metadata for plate reader experiments and can be connected with SynBioHub, a synthetic biology design repository, through URIs.

1.7 Software

1.7.1 Computation

Computation had a huge impact on humanity, its development started the Information Age. The information age marked a shift from traditional industries to economies centered on information technologies. To perform computation a combination of two components is needed. The hardware is the physical instance or device that performs the computation meanwhile the software is the set of instructions that controls what and how the hardware behaves, and computation is performed. The software can be divided into Operating System (OS) the software that communicates with the hardware and Application Software which interfaces the user and the OS. Programming languages are systems of notation for writing computer programs. They can be low level, closer to the hardware interface, for example assembly

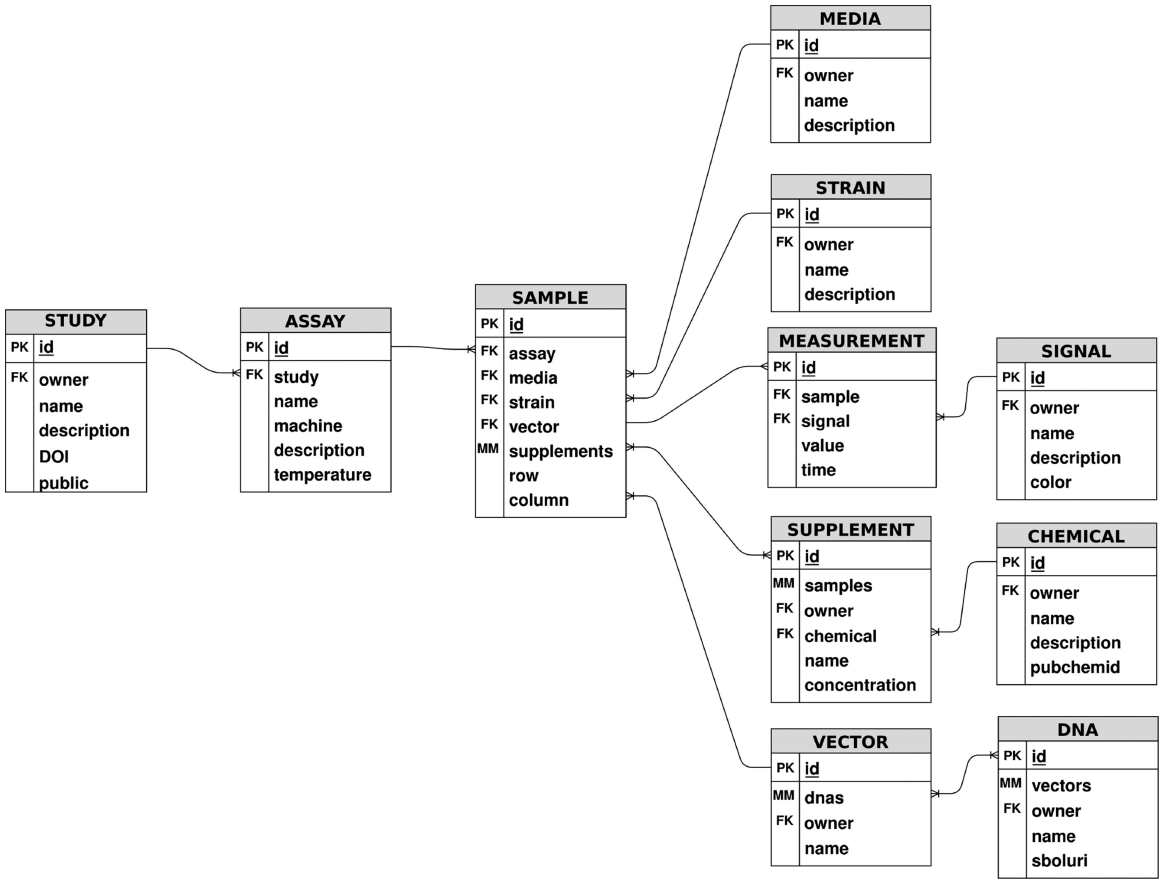


Fig. 1.5 Flapjack's data model. The basic unit of data is the measurement, which is the readout of a reporter or biomass signal in time. A measurement refers to a specific sample, which can be a well in a plate-reader experiment or a colony in a Petri dish time-lapse. Samples are grouped in a specific assay, which corresponds to a single kinetic procedure. Assays belong to a particular study that aims to answer a scientific question or project. Along with signal data, each measurement is described in terms of its context through experimental metadata, mainly composed of the media substrate, the host strain, the DNA which the host is transformed with, and inducer chemicals to study stress or other regulation behaviours of the system. It is important to mention that some metadata could not be present, such as strain in Cell-free assays or DNA in control samples, to name a few. PK is the primary key, FK is foreign key, and MM is a many-to-many relation. From Flapjack publication [196].

language has a strong correspondence to the computer machine language instructions and is translated into it using an assembler. High level programming languages are closer to the user interface, being closer to natural languages than machine languages, for example Python language has a strong correspondence to English natural language, emphasising code readability. Software tools have been developed and used to deliver a man on the moon,

control cars, automate processes, create intelligent systems [175], perform mathematical calculations and to analyse data, being a valuable tool for researchers.

1.7.2 Biocomputation

Computation can be also done in biological devices, biocomputation uses bioengineering to build systems that perform computation. In biology the hardware can be mapped to the host cell and its biochemical composition, and the software can be mapped to the DNA and its sequence. Logic gates are encoded inside chips and can be encoded in cells as well [123, 168, 65]. The genetic circuit idea has implicit the idea of performing computation in a biological substrate, but also constrains its behaviour to logic functions and uses the electrical engineering abstraction [67]. The genetic network idea includes the genetic circuit but also any other behaviours that can be encoded in a graph and uses a software engineering abstraction. Graph theory can be used to analyse genetic networks, where nodes are DNA fragments and edges are interactions between them, to operate the system mathematically and use tools from dynamical systems like linear stability analysis.

1.7.3 Computational biology

Computational biology is the use of computation to solve problems from biological sciences. It mainly involves the use of software to model and analyze biological systems. Software is integrated in biological research, storing and managing experimental results, performing analysis and running measurement hardware. Although software tools are well integrated in punctual stages, it is difficult to connect different tools to close the DBTL cycle, and useful metadata is often lost. In this work I used software to run simulations and analyse data. More over, I developed software to control liquid handling robots affecting the physical experiment. I process and analyse simulated data and experimental data using the same methods, so I will make a clear distinction between them.

1.8 Laboratory automation

Laboratories can be divided into wet labs and dry labs. Wet labs are where physical experiments are performed. These laboratories have facilities for liquid handling, i.e. pipettes, micro-pipettes, tubes, burette and micro-fluidic devices. Dry labs are laboratories without facilities for liquid handling, therefore focused in the mathematical or computational aspect of research, i.e. computers and high performance computers (HPC). The regular work in wet labs is highly manual, samples lack traceability and protocols can be interpreted differently

by different researchers. The lack of standards for sequence representation and metadata capture, among other practices, have contributed to a reproducibility crisis in synthetic biology hindering the velocity of scientific discoveries [129]. Software and robots have been used to improve the traceability and reproducibility in protocol execution. The biological industry and clinical laboratories have successfully implemented high levels of automation. Lab automation tools are usually expensive and require a high use to be cost effective which can be difficult to achieve for small wetlabs. Liquid handling robots are one of the main tools for lab automation. Opentrons have developed an Open-Source and relatively inexpensive liquid handling robot, making more accessible lab automation for wet labs. The Opentrons OT-2 is a liquid handling robot used in this work as a lab automation tool. The scientific community has developed software to run protocols in OT-2 robots for DNA assembly, PyLabRobot [193] and DNA-BOT [166] for example.

1.9 iGEM

The international Genetic Engineered Machine (iGEM) is a synthetic biology competition initially aimed for undergraduate students organised at the Massachusetts Institute of Technology (MIT) since 2003. This competition is at the core of synthetic biology and rapidly evolved in an independent organisation, the iGEM foundation that holds this competition, the "iGEM Jamboree". The iGEM Engineering Committee was formed with the aim to set standards and best practices to advance in the "engineerability" of biological systems. Furthermore, iGEM has one of the biggest repositories of biological parts, and characterization data available at the moment making it one of the key resources for the advance of synthetic biology [180, 161].

1.10 Aim

As an engineering discipline, the DBTL cycle is at the core of synthetic biology research and development. The field of synthetic biology is in a maturation process developing standards and software tools to automate and connect the different DBTL cycle stages. The aim of this thesis is to develop workflows that automate and connect the DBTL cycle for engineering genetic networks.

This would accelerate scientific discovery, improve reproducibility and the collective construction of knowledge in the field. To achieve this, it is imperative to develop standards, a design abstraction, DNA parts, DNA assembly strategies, protocols and software tools that

tackle different DBTL stages, or connections between them, that work in a modular fashion to be part of the framework.

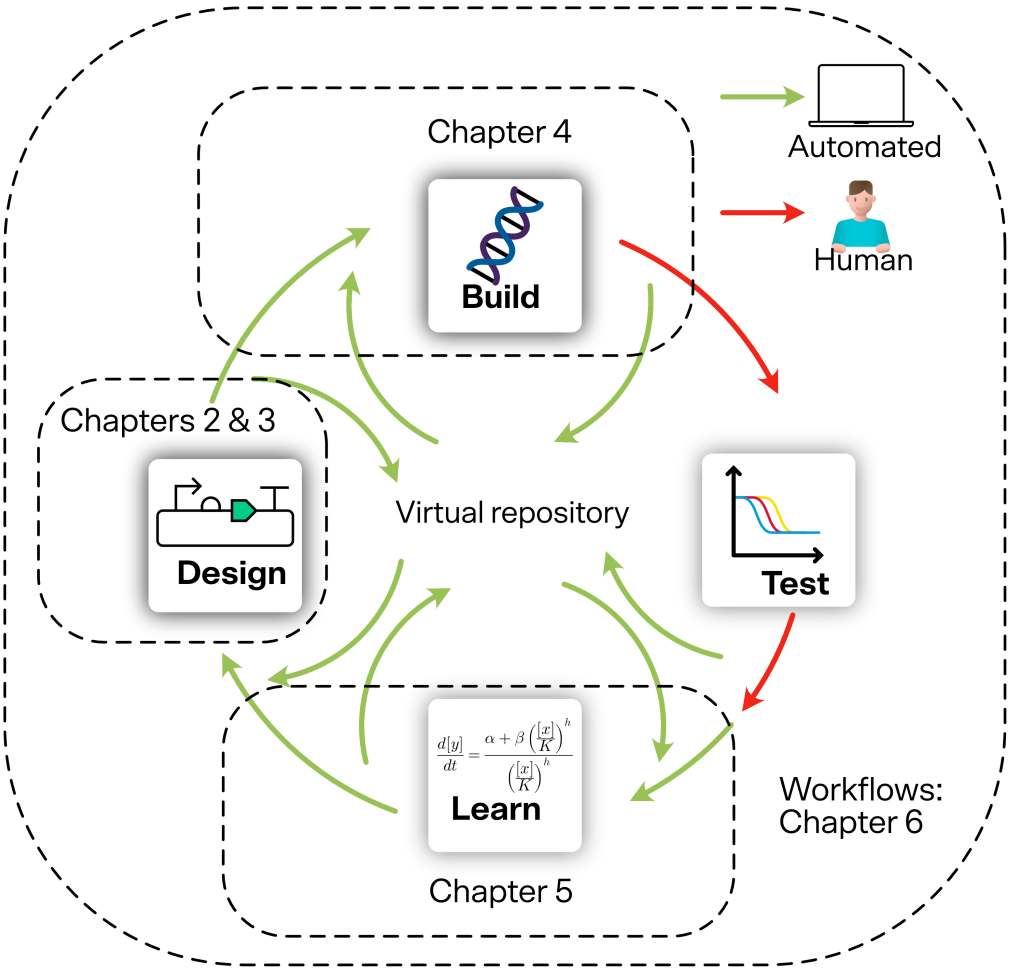


Fig. 1.6 Graphical abstract.

1.11 Objectives

1.11.1 Objective A. Create a design tool for synthetic genetic networks.

In order to engineer synthetic genetic networks, it is necessary to abstract them to use them as parts iteratively to create larger and larger networks and an appropriate software tool to aid in the design of such networks. The objective is to create a design abstraction compatible with modular assembly and use it to develop a synthetic genetic network design tool. This

software tool will produce a standardised representation of the design and use kinetic plate reader data to characterise their gene expression.

1.11.2 Objective B. Automate the build stage with software and robotics.

To connect the design and build stages a software capable of reading standardised design instruction is needed. The objective is to contribute to the build plan standardisation and its codification in SBOL3 to use it as input in a software tool for build automation. The build automation tool provides the protocol information to the user and produces instructions for the Opentrons OT-2 liquid handling robot.

1.11.3 Objective C. Develop and implement a method for characterising genetic networks to connect Learn and Design stages.

To improve the parameter estimation and the information provided by characterizations new methods where required as existing ones are difficult to replicate, to interpret and to use. The objective is to develop an algorithm for kinetic data characterization to obtain gene expression rate and growth rate from populations of bacteria and implement it as a method in my GDA software tool. The algorithms are compared to state of the art methods and used to characterise gene expression from a set of transcriptional units in a plasmid with an *in vivo* reference.

1.11.4 Objective D. Demonstrate DBTL workflows.

To create automated DBTL workflows. The objective is to use the tools developed throughout this work to create different DBTL workflows, including a simulated DBTL workflow, a DBTL workflow with manual DNA assembly and a DBTL workflow with automated DNA assembly.

1.12 Structure of the thesis

The structure of the thesis is as follows. In Chapter 2, I create a design abstraction for transcriptional units compatible with modular assembly and functional synthetic biology, controlling key parts for modelling. In Chapter 3, I build upon the previous design abstraction to develop LOICA, a programmatic tool for genetic network design, simulation and characterization. LOICA uses an object oriented design abstraction to design genetic networks. These networks can be simulated using deterministic and stochastic approaches. The designs can be

exported using the SBOL standard. In Chapter 4, I describe a standardised representation of build plans and software to create them and simulate the assembly using Python. SBOL outputs from LOICA can be formatted with the build plan representation which works as build instructions for PUDU. PUDU is a software tool that can use standard build plans to simulate and create instructions for a liquid handling robot and a human to perform the assembly, connecting Design and Build, similar protocols were implemented for transformation, plate setup and calibration. In Chapter 5, I describe and benchmark a characterization method for growth rate and gene expression rate from kinetic measurements obtained from a plate reader. This method was implemented in Flapjack and used to parameterize LOICA gene expression models. In Chapter 6, I combined the tools developed in this work to create automated DBTL workflows for different engineering needs: dry lab, with wet lab and with automated wet lab. In Chapter 7, I provide my conclusions. In Chapter 8, I discuss my work, and provide my perspective about its implications and future steps.

1.13 Contribution of the thesis

In this work I have created a framework for engineering synthetic genetic networks that leverages standards to close and automate the DBTL cycle. The framework includes standards, design abstractions, DNA parts, DNA assembly strategy, protocols and software tools that tackle different DBTL stages, or connections between them, and work in a modular fashion.

- Paper 1: Yáñez Feliú, G., Vidal, G., Muñoz Silva, M., Rudge, T. J. (2020). Novel tunable spatio-temporal patterns from a simple genetic oscillator circuit. Frontiers in Bioengineering and Biotechnology, 8, 893.
- Paper 2: Yanez Feliu, G., Earle Gomez, B., Codoceo Berrocal, V., Munoz Silva, M., Nuñez, I. N., Matute, T. F., ... Rudge, T. J. (2020). Flapjack: Data management and analysis for genetic circuit characterization. ACS Synthetic Biology, 10(1), 183-191.
- Paper 3: Vidal, G., Vidal-Céspedes, C., Muñoz Silva, M., Castillo-Passi, C., Yáñez Feliú, G., Federici, F., Rudge, T. J. (2022). Accurate characterization of dynamic microbial gene expression and growth rate profiles. Synthetic Biology, 7(1), ysac020.
- Paper 4: Vidal, G., Vitalis, C., Rudge, T. J. (2022). LOICA: Integrating models with data for genetic network design automation. ACS Synthetic Biology, 11(5), 1984-1990.
- Paper 5: Rudge, T. J., Vidal, G. (2022). Phase-based genetic logic circuits. bioRxiv, 2022-12. (Pre-print)

- Paper 6: Aldulijan, I., Beal, J., Billerbeck, S., Bouffard, J., Chambonnier, G., Ntelkis, N., ... Vignoni, A. (2023). Functional synthetic biology. Synthetic Biology, 8(1), ysad006.
- Paper 7: Samineni, S. P., Vidal, G., Vitalis, C., Feliú, G. Y., Rudge, T. J., Myers, C. J., Mante, J. (2023). Experimental data connector (XDC): integrating the capture of experimental data and metadata using standard formats and digital repositories. ACS Synthetic Biology, 12(4), 1364-1370.
- Paper 8: Buecherl, L., Mitchell, T., Scott-Brown, J., Vaidyanathan, P., Vidal, G., Baig, H., ... Myers, C. (2023). Synthetic biology open language (SBOL) version 3.1. 0. Journal of Integrative Bioinformatics, 20(1), 20220058.
- Paper 9: Beal, J., Selvarajah, V., Chambonnier, G., Haddock, T., Vignoni, A., Vidal, G., Roehner, N. (2023). Standardized Representation of Parts and Assembly for Build Planning. ACS Synthetic Biology, 12(12), 3646-3655.
- Paper 10: Macuada, J., Molina-Riquelme, I., Vidal, G., Pérez-Bravo, N., Vásquez-Trincado, C., Aedo, G., ... Eisner, V. (2024). OPA1 and disease-causing mutants perturb mitochondrial nucleoid distribution. Cell Death Disease, 15(11), 870.
- Software 1: LOICA, Programmatic genetic design software tool. https://github.com/RudgeLab/LOICA
- Software 2: PUDU, Liquid handling robot control on synthetic biology workflows. https://github.com/RudgeLab/PUDU
- Software 3: Excel2flapjack, Generalized excel interface to upload data to Flapjack. https://github.com/SynBioDex/Excel-to-Flapjack
- Software 4: XDC, Generalized excel interface to upload data to SynBioHub and Flapjack connecting metadata to experimental data. https://github.com/SynBioDex/Xperimental-Data-Connector
- Software contribution 1: Inverse method for bacterial growth and expression rate characterization was contributed to Flapjack. https://github.com/flapjacksynbio/pyFlapjack
- Software contribution 2: Standardized representation of build plans and experimental components was contributed to SBOL utilities. https://github.com/SynBioDex/SBOL-utilities

- Book chapter 1: Vitalis, C. et al. (2024). Flapjack: Data Management and Analysis for Genetic Circuit Characterization. In: Braman, J.C. (eds) Synthetic Biology. Methods in Molecular Biology, vol 2760. Humana, New York, NY. https://doi.org/10.1007/978-1-0716-3658-9_23
- Book chapter 2: Vidal, G., Vitalis, C., Matúte, T., Núñez, I., Federici, F., Rudge, T.J. (2024). Genetic Network Design Automation with LOICA. In: Braman, J.C. (eds) Synthetic Biology. Methods in Molecular Biology, vol 2760. Humana, New York, NY. https://doi.org/10.1007/978-1-0716-3658-9_22
- Book chapter 3: Vidal, G., Vitalis, C., Guillén, J. (2024). Standardized Golden Gate Assembly Metadata Representation Using SBOL. In Golden Gate Cloning: Methods and Protocols (pp. 89-104). New York, NY: Springer US. https://doi.org/10.1007/978-1-0716-4220-7_6

Software tools created in this work were used in combination with existing tools in the field to develop three workflows that target different research lab needs, from dry labs to wet labs with or without lab automation.

These workflows are highly automated reducing the time that researchers dedicate to experimental tasks and accelerating scientific discoveries. They support the generation of high quality data in a standardised format so it can be reused and combined with more data generated in the same way to build knowledge in a truly incremental and collective fashion. Although one of the goals is to promote the adoption of the SBOL standard on the field, the tools can be used without them but offer more functions when it is used to help in this transition. The workflows facilitates the production of calibrated data and the use of liquid handling robots to reduce human error, improving reproducibility.

Glyph	Description
	Promoter
	RBS
	Ribonuclease site
	Coding sequence
	Protein stability element
	Terminator
	Assembly scar
	Circular plasmid
	Omitted detail
	Engineered region
	Simple chemical
	Protein

Table 1.1 SBOL visual glyphs used in this work.

Chapter 2

Design abstraction for genetic devices

2.1 Introduction

Synthetic biology poses a new mindset where scientists are not constrained to existing natural DNA and now have the freedom to design and synthesise almost any DNA sequence. Although in future DNA synthesis will be trivial and inexpensive, for now DNA assembly is the most convenient way for the physical implementation of DNA. Also, physicochemical constraints in DNA synthesis do not allow for the synthesis of complex DNA sequences, so assembly techniques are complementary to DNA synthesis allowing for the concatenation of synthesized sequences. Interesting and useful genetic parts and networks have been constructed, but most of the scientific endeavour is not reusable by other practitioners [107]. Assembly standard syntaxes 1.3.4 allow the reuse of DNA parts across different laboratories. Parts implemented with one assembly syntax are compatible within it, but they might not be compatible with parts in another assembly syntax. This is why the selection of the assembly strategy to use is key for part acquisition and collaboration. Although final plasmids are shared on repositories like Addgene, the parts to build them are not shared and therefore it is difficult to modify them to adjust for other practitioner needs. Having repositories of DNA parts and characterised TUs will provide new resources to analyse gene expression mechanisms in a collective way driving new discoveries in molecular biology and applications such as biosensors [127] or fuel production [191].

In this chapter, I describe a design abstraction to construct genetic devices, composing parts into TUs and composing TUs into genetic networks. The assembly strategy incorporates parts to control production and degradation of proteins, and is compatible with assembly strategies to build trivial TUs using the Phytobrics assembly syntax. Furthermore, TUs can be readily used to compose genetic networks when inserted in Loop Odd receivers. The DNA parts were updated with new insulation technologies to reduce compositional context of parts in TUs and of TUs in genetic networks. Finally, functional synthetic biology, a proposal of a functional paradigm compatible with the design abstraction and assembly strategy, is described as a theoretical framework.

2.2 Results

2.2.1 Design Abstraction for genetic devices

The design choice is to use SBOL to represent genetic devices, the rational is as follows. The gene is a sequence of DNA that produces an RNA that may be translated into a protein. In this work genes correspond to TUs, entirely engineered and composed of parts. Parts are DNA sequences with the appropriate interface for it to be integrated into a composite in the

context of a build plan. For example, inside an assembly plan by designing the part flanking fusion sites its position and orientation is decided. Devices are biological functions with the appropriate interface for it to be integrated into a system in a biological context. Let's define a device as a unit of gene expression, a black box that with certain inputs retrieves an output (Figure 2.1). The output can be a characteristic gene expression profile in a determined biological context, being the simplest a TU, up to an entire repressilator and more complex genetic networks. To be more specific I defined four biological contexts: (1) The short-range sequence context, relates to the interaction of parts inside a TU, i.e. How changes in the promoter or CDS sequence affects the gene expression? It is an analog to cis-regulation but the short-range sequence context only refer to effects inside a TU not the whole DNA molecule, for example the effect of the origin of replication on a TU in the same plasmid would be cis-regulation but not a short-range sequence context effect. How the expression rate or the gene product of a gene changes when the sequence of that gene changes. (2) Long-range sequence context, relates to the interactions between TUs and other DNA in the host, including plasmids, the genome and their genetic products with direct effect on expression, i.e. How does gene products from other DNA regions affect my designed TU? How does the origin of replication of the plasmid where the TU is inserted affect its gene expression? How does the expression rate or gene product of a gene change when other genes or genetic material sequences change? Similar to trans-regulation, but long-range sequence context includes regulation of TUs in the same plasmid. For example a TU expressing LacI that represses a promoter in the same plasmid is considered long-range sequence context but not trans-regulation. (3) Organism context, relates to the intrinsic biochemical and biophysical state of an organism which is host to the designed DNA. Includes all interactions with genetic products and chemicals with indirect effect on expression, i.e. pH, codon usage, ribosome architecture, RNAP. (4) Environmental context, relates to the extrinsic biochemical and physical state of the environment that surrounds the organism hosting the designed TU, i.e. is the media carbon source glycerol or glucose? Is the viscous drag high or low?

SBOL can capture the sequence and function of a TU. Following the SBOL standard DNA sequences can be associated with a sequence feature or function. One or more sequence features can be a part inside a TU that has a function that affects the expression rate or the genetic product. A clear distinction between a sequence feature and a part has to be made, the former relates to the function or feature of a DNA sequence in the context of a biochemical environment and the latter relates to a position and direction of a DNA sequence in the context of a build plan. The trivial TU is composed of promoter, RBS, CDS and terminator; these sequence features that can correspond to parts need to be in a determined order (Figure 2.2 A). The sequence compilation of the parts inside a TU does not follow the commutative

law, therefore the order of the parts affects the product. An ordered composite of parts can be seen as a vector given its directionality which is imposed by how the RNAP transcribes the DNA, from 5' to 3'. The vector of parts is an abstract design that is useful for simulations, but needs more information to be constructed, like defining a backbone for the composite. Once the sequence is fully defined, its physical implementation can be built through DNA synthesis or DNA assembly.

The design abstraction to build genetic devices follows the SBOL standard and is composed of a vector or parts similar to the trivial TU design. The first goal is that parts inside a TU can be swapped with other parts following a compatible assembly syntax, although the interaction between sequence features in those parts might not be linear, general trends can be extrapolated. For example the promoter sequence feature can be synthesised to be a part that goes at the beginning of a TU. By swapping the promoter, in the context of the same TU expressing a reporter, its "strength" can be measured and characterised to be i.e. low, medium, or high. When using this promoter in another TU the raw "strength" will change but their relative "strength" will be conserved if there are no complex interactions with the short-range context or organism context. The second goal is that TUs inside a genetic network can be swapped or their order in the composition changed, maintaining their expression profiles or that it changes in a predictable manner. For this TUs needs to have appropriate isolation and interfaces to be assembled and rearranged in the insertion site. For example two contiguous TUs in a design should maintain their profile expression if there are no complex interactions with the long-range sequence context or organism context interactions.

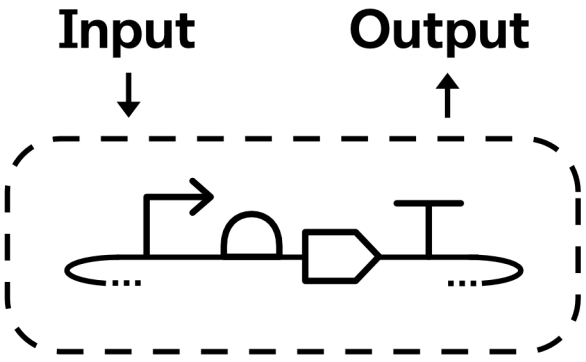


Fig. 2.1 Genetic device design abstraction. SBOL is used to represent a genetic device that senses and input and produces an output.

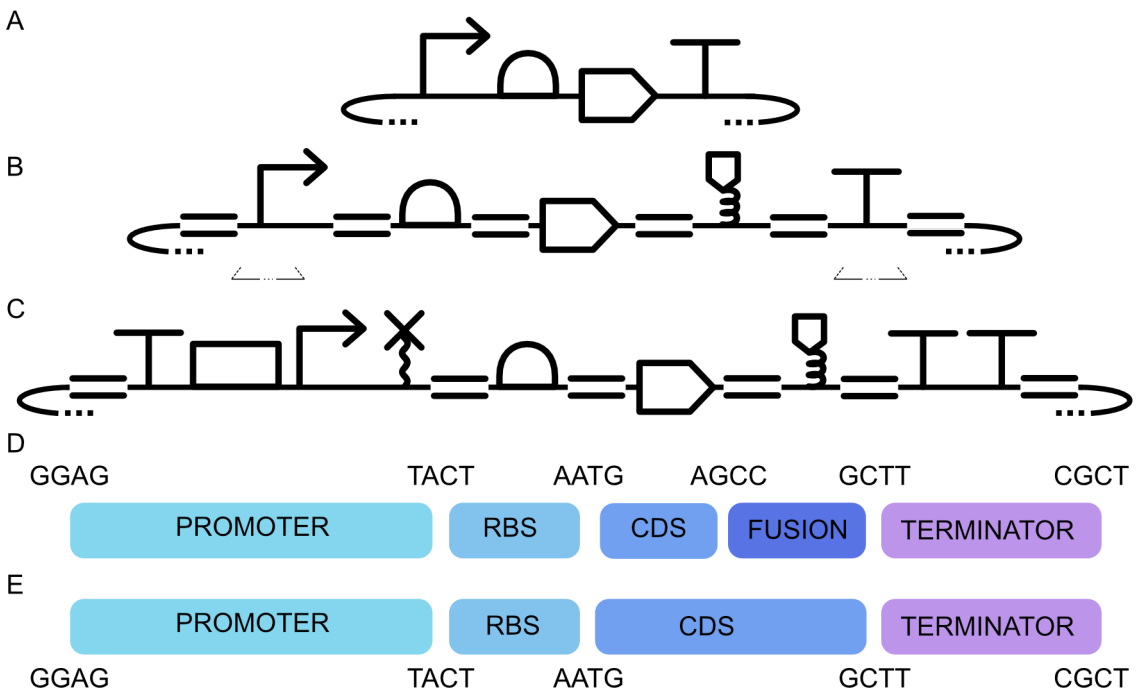


Fig. 2.2 Transcriptional unit design. (A) Example of a trivial transcriptional unit using SBOL Visual. (B) Representation of the transcriptional unit design with collapsed parts using SBOL Visual. (C) Representation of the transcriptional unit design parts using SBOL Visual. (D) Assembly strategy using the Phytobricks syntax for degradation tag combinatorial design. (E) Assembly strategy using the Phytobricks syntax for trivial transcriptional units.

2.2.2 Part architecture and transcriptional unit design

The idea of composing a TU using parts is now common practice in synthetic biology. Usually parts in an assembly have a one to one correspondence with sequence features. For example, the trivial TU (Figure 2.2 A) is usually assembled in synthetic biology practicals and competitions. Although it is a very useful set of parts, first it does not allow for the tuning of degradation and second new technologies on part isolation have been developed. To allow the control of degradation a fusion part was added to the trivial TU to create a design compatible with the trivial TU (Figure 2.2 B). To implement this the CDS were modified removing the stop codon and adding two bp so a 4 bp scar reconstitutes the reading frame just adding 2 amino acids. In this case this scar GGAGCC were engineered to code for two of the smallest amino acids glycine and alanine in that order.

The promoters were engineered by implementing new technologies in isolation (Figure 2.2 C). The promoter part starts in 5' with a terminator, reducing the probabilities of trailing RNAPs that can influence the transcription initiation of the promoter, increasing expression

and noise. For example if I concatenate 2 simple TUs composed of a Pro, RBS, CDS and Ter, the expression of the upstream TU will produce a flow of RNAPs that can reach the promoter of the downstream TU, increasing its transcription rate and producing artifacts in its gene expression rate. In between this terminator and the promoter sequence features an upstream element were added. It has been demonstrated that the 36 bp upstream of the promoter changes its expression [27]. Between the terminator with 51 bp and the upstream element with 15 bp a total of 66 bp upstream of the promoter are maintained constant. For example, when using a promoter sequence feature as the first part in an assembly, changes in the upstream sequence of the promoter are produced by changing the plasmid where the TU is inserted. After the promoter a ribozyme was added to cleave the RNA just before the RBS [104]. This removes the sequence of the promoter or other upstream sequence that may affect the translation process. For example if I express the same GFP protein with the promoter J23101 or J23100 the RNAs will be different because of the different promoter sequences that are transcribed after the TSS. The ribozyme cleaves sequences leaving the same RNA product when using different promoters and therefore the translation process is more comparable. Finally, to reduce the production of trailing RNAPs a terminator part composed of two terminator sequence features was used.

2.2.3 Assembly strategy to compose genetic devices

In order to compose genetic devices incrementally I took the design choice of using two compatible assemblies Phytobricks and Loop for my assembly strategy. Phytobricks was chosen because it is used by iGEM, it is compatible with MoClo and has a syntax for fusion proteins. Because, its syntax requires a linker and a fusion protein, but I just needed to add one part I used the prefix of the linker and the suffix of the fusion protein to make it compatible with other designs [26]. Furthermore, the strategy for assembling TUs with controlled degradation (Figure 2.2 D) is compatible with the strategy to assemble the trivial TU (Figure 2.2 E). Then, Loop was chosen because it is compatible with Phytobricks and just involves the use of Loop receiver vectors [135]. The assembly strategy to compose TUs into genetic networks follows the Loop syntax for bacteria. Loop is not widely used, but to adopt it the user just needs a minimum of 8 receiver plasmids, 4 Odd receivers and 4 Even receivers. The complete kit has Odd and Even receivers with multiple origins of replication allowing for plasmid copy number control.

The workflow for this assembly strategy is as follows. The first step is domestication or level 0 assembly, in this step a proper interface is added to a DNA sequence to work as a part. Domestication was performed from synthetic dsDNA fragments inserted into the universal acceptor backbone. The second step is level 1 (Odd) assembly, in this step TUs are assembled

and a proper interface is added to them. Level 1 assembly was performed composing domesticated parts using the Phytobrics strategy in an Odd receiver which provides the interface for that TU to be used in an Even assembly. The third step is level 2 (Even) assembly, in this step genetic networks of up to 4 TUs are assembled and a proper interface is added to it. Level 2 assembly were performed composing TUs in Odd receivers following the Loop strategy in an Even receiver which provides the interface for the genetic network to be used in an Odd assembly. If the design uses less parts than 4, spacers can be used in any position. More complex designs can be done in level 3 (Odd) assembly composing 16 TUs and level 4 (Even) assembly composing a maximum of 64 TUs.

2.2.4 Functional synthetic biology

Note: *The work presented in this section was carried out in collaboration with Ibrahim Aldulijan, Jacob Beal, Sonja Billerbeck, Jeff Bouffard, Gaël Chambonnier, Nikolaos Ntelkis, Isaac Guerreiro, Martin Holub, Paul Ross, Vinoo Selvarajah, Noah Sprent and Alejandro Vignoni, and some of it was published in reference [3]. This is my perspective on this work and may not represent the vision of all authors.*

Combining the abstraction of genetic devices with its assembly strategy it creates a good basis to implement functional synthetic biology. The idea of functional synthetic biology is to change the actual sequence-based paradigm to a function-based paradigm. In the sequence-based paradigm the function of a part is linked to its sequence, and the function is omitted or captured as description. This paradigm obstructs the decoupling between the knowledge of how to create a TU device to how to compose devices to create a function. In the function-based paradigm the function of a part is not necessarily linked to its sequence, and the function is informed to the user in a standardised and machine readable way (Figure 2.3). Therefore, compatible parts can be swapped or updated without modifying the overall function of a device.

Following the assembly strategy, sequence feature parts can be composed into TUs using the Phytobricks syntax in a Loop Odd receiver. That TU can encode a device and because it is inserted in an Odd receptor it has the interface to be a part in further assemblies. A set of four TU parts can be joined in a Loop Even receptor to create a system of four TU devices or a genetic network. That genetic network is composed of up to four TUs because it is inserted in a Even receptor it has the interface to be a part. In a subsequent step up to four genetic network devices can be composed to build a system or genetic network of up to 16 TUs, and up to 64 TUs in the next level. The paradigm works in a recursive way allowing the implementation of bigger and bigger systems.

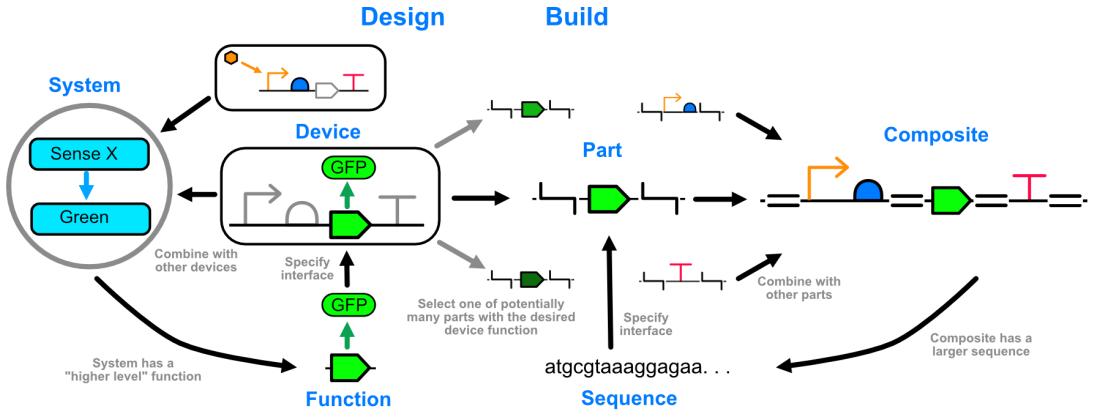


Fig. 2.3 Functional synthetic biology. Diagram of the flow of structural and functional information under the functional synthetic biology paradigm. The functional loop, related to the Design stage, is on the left side. In the example we have the CDS glyph representing the function "Green Fluorescent Protein coding sequence". When an interface is specified the function becomes a device. Then, devices can be composed using that interface to form systems. If the system has a higher level function (i.e. if x is sensed, produce green fluorescence) it can circle back and interfaces can be added to form a new device. The structural loop, related to the Build stage, is on the right side. In the example we have fragment of the coding sequence for GFPmut3. When an interface is specified the sequence becomes a part, such as adding MoClo fusion sites C and D flanking the sequence. Then, parts can be compose using the interfaces to form composites. Because composite are just larger sequence they can circle back and become a part if interfaces are added. There is no direct relationship between a function and a part, as in the structural paradigm. The functions can be fulfilled by different parts, allowing for more adaptability on the build stage and working as a base for device versioning, an analogy to software engineering.

2.3 Discussion and conclusions

In this chapter I developed a design abstraction for genetic devices compatible with modular assembly and with a functional synthetic biology paradigm.

The functional synthetic biology paradigm is compatible with a software engineering design abstraction. Genetic devices are similar to functions imported from packages with libraries of genetic devices that can be swapped and updated, implementing for example new technologies in insulation. Then users could run tests to see how this change can affect the design and choose the version of the device to use. Although there are still many challenges to implement it, the iGEM engineering committee is working to make functional synthetic biology a reality, developing package systems for biological designs in SBOL.

The assembly strategy developed in this chapter is compatible with the functional synthetic biology paradigm where new parts just have to follow the proposed Phytobrics syntax

and then follow the same Loop assembly steps to update a genetic device. It also has the capability to include more TUs in a recursive way that exceeds the number of TUs used in common genetic networks. Furthermore, having too many TUs in one cell might impose a huge burden and the metabolic limit might be way below 64 TUs posing challenges for the construction of big genetic networks. This hindrance has been acknowledged in the field and systems distributed across many strains in a consortium are one of the possible solutions and the assembly strategy developed from this chapter is compatible with it [20].

Chapter 3

Programmatic genetic network design automation

3.1 Introduction

Genetic design is the process of designing genes or genetic elements with certain structure or function. Traditional engineering disciplines have developed, decades ago, software tools to aid the design process, for example mechanical engineering and electrical engineering have autoCAD and autoCAD electrical respectively from Autodesk. Synthetic biology is at an early stage of development of such tools.

Modelling is at the core of synthetic biology. The two foundational papers the repressilator [50] and the toggle switch [58] stand out for their mathematical modelling of the biochemical processes. Design tools need these models to define relevant parameters to consider and to increase the chances to get a successful product.

iBioSim was one of the firsts tools for genetic network design. It uses SBOL as the abstraction level, this provides a detailed customization but a very steep learning curve. Characterization then needs information at the part level, although there are tools to calculate RBS [147] and promoter strength [99, 185] they are in different formats, not compatible or not experimentally accessible. Structural and functional information has to be specified separately so users also need to learn how to represent models in iBioSim and connect them to the structural information. Simulations are performed at the cell level so can only be comparable to single cell data which is difficult to acquire. This approach has enabled the design of relatively small genetic networks [50, 58, 40], however for large-scale genetic network design higher-level abstractions are required.

Cellular Logic (Cello) [123] is a CAD tool that accepts an input from the user in Verilog and user constraints files (UCF) to compile the DNA sequence to build a genetic circuit implementing that logic. Designs are created in an analogous way to electronic circuits, based on the required discrete logical truth table, however this specification requires knowledge of the domain-specific programming language *Verilog* [171]. It uses logic gates as the abstraction level, this provides a high-level understanding of the circuit built, but little information of the genetic parts. This issue has been partially solved in a newer version, Cello 2.0 [82] that implements the SBOL export of the files. Despite its academic relevance and apparent utility, Cello did not have an impact in how synthetic biologists work. Some reasons for this could be that, as an academic software it lacks support and reliability, and one of its inputs is Verilog, a low-level language very rare among the life sciences, increasing the entry barrier. Despite the discrete logical design formalism, these genetic networks are dynamical systems and can have autonomous, continuous non-steady-state dynamics, displaying complex and rich behaviours from bi-stability to oscillations and even chaos [50, 58, 188]. Furthermore, typical operating conditions for engineered networks like colonies, bioreactors or microbiomes are time varying, which can lead to complex

behaviours from even simple genetic networks [131]. To design genetic networks, we therefore require kinetic gene expression data generated at the test phase. This data must be integrated with models to enable characterization of abstracted parts, devices, and systems, as well as metadata, including the DNA part composition and sequence, to enable automated design. Thus, there is a need for software design tools that integrate abstract network designs, dynamical models, kinetic gene expression data, DNA part composition, and sequence via common exchange standards in a user-friendly and accessible fashion.

Here I present LOICA, a Python package for the programmatic design, modelling, parameterization and sequence compilation of synthetic genetic networks. LOICA was implemented in Python given its readability and because it is among the most used programming languages, usually taught to engineering students. LOICA provides a high-level design abstraction that simplifies the design process by representing networks as combinations of components accessible to parameterization. This parameterization of genetic network models is done using plate reader measurements and enabled by direct connection to experimental data via Flapjack [183], which also provides a platform for publishing and sharing simulation results. Furthermore, while LOICA abstracts genetic networks at a higher level, designs can be represented using the latest SBOL3 standard for biological design representation. While perhaps not as accessible as a graphical user interface, this approach is more flexible, extensible, and amenable to automation.

3.2 Results

Note: *The work presented in this chapter was carried out in collaboration with Carolus Vitalis (CV), and some of it was published in reference [178]. All simulations and analysis presented below were performed by GV, except where explicitly noted as the work of CV.*

3.2.1 Design Abstraction for Genetic Networks

Genetic networks can be abstracted as a network composed of a set of TUs as nodes, and their interaction as edges. TUs can be represented as a composition of SBOL `Components` containing features that describe individual parts and their DNA sequences. These parts can vary a lot within TUs, being the most simple the ordered combination of a Pro, RBS, CDS and Ter but can be more complex by adding insulators like ribozymes and protein stability elements. Although TUs can be abstracted in their standard SBOL parts, this captures more complexity than is necessary for decoupling sequence engineering from genetic network engineering.

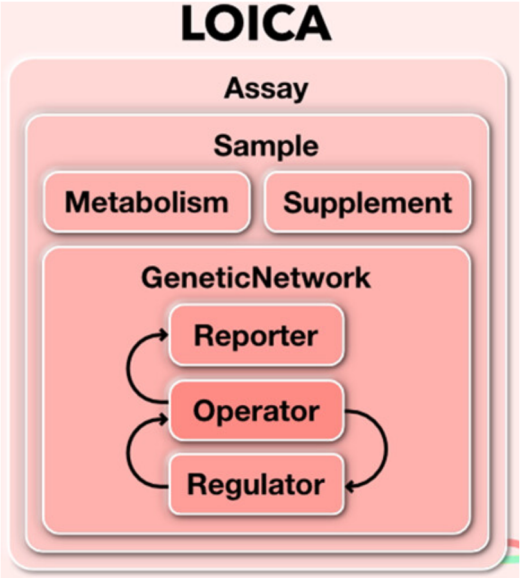


Fig. 3.1 LOICA classes. Encapsulation relation of classes in LOICA design abstraction.

In LOICA, transcriptional units or genes were abstracted in two classes, `Operator` and `GeneProduct`. The `Operator` represents a DNA fragment that regulates gene expression and the `GeneProduct` represents the molecular species that is expressed. `Operators` change their expression rate which modulates `GeneProducts` production. `GeneProducts` have a concentration and a degradation rate that modulates `GeneProduct` degradation or destruction. The `Regulator` is a `GeneProduct` that represents a molecular species that can regulate gene expression. The `Reporter` is a `GeneProduct` that represents a molecular species that can be measured or emits a signal. The `GeneticNetwork` class encapsulates `Operators` and `GeneProducts`. `GeneticNetworks` represents a graph where the nodes are `Operators` and `GeneProducts`, and the edges are the relations of production and control between them. The `Supplement` class represents an externally controllable molecule that is the input of a `Receiver`.

The simulation setup is done by adding context information related to the metabolism of the bacteria running the genetic network and sample composition. The `Metabolism` class represents chassis or host conditions through the biomass and growth rate information. The `Sample` class encapsulates `GeneticNetwork`, `Metabolism` and `Supplements` and defines the experimental unit. The `Assay` class encapsulates `Samples`. `Assay` represents a set of samples in an experimental design and define simulation conditions for them such as number of measurements and interval between them. The `Colony` class represents a bacterial colony growing expressing a genetic circuit. Similar to an `Assay` with one `Sample` that is the colony.

The basic classes and their relation is shown in figure 3.1 and the full class diagram is in figure 3.7.

3.2.2 Model generation and simulation

The interactions between the `Operators` and the `Regulators` encode models for genetic network temporal dynamics, which can be simulated with ODE or SSA (Figure 3.2 A). `Operators` input or profile parameter defines its expression rate, inputs can be `Regulators` or `Supplements`. The profile is a time-series that is used to represent gene expression and growth rate over time. The output parameter, what is being produced or expressed, outputs must be `GeneProducts`. The system of ODEs is thus:

$$\frac{d\vec{p}}{dt} = \Psi(\vec{r})\eta(t) - \Gamma\vec{p} - \mu(t)\vec{p}, \tag{3.1}$$

$$\Psi(\vec{r}) = \sum_k \Phi_k(\vec{r},t), \tag{3.2}$$

where $\vec{p} = (p_0, p_1, ...p_{N-1})^T$ is the vector of `GeneProducts`, which includes different `Regulators` ($\vec{r} = (r_0, r_1, ...r_{M-1})^T$) and `Reporters` ($\vec{s} = (s_0, s_1, ...s_{N-M-1})^T$). The non-linear operator Ψ maps `Regulator` concentrations to `GeneProduct` synthesis rates. η is a function of time that modifies Ψ to create a profile or time-series. Γ is a diagonal matrix of `GeneProduct` degradation rates γ_i, and $\mu(t)$ is the instantaneous growth rate of the cells. Equation 3.1 shows the overall system where Ψ encodes the whole network and consists of a sum of individual LOICA `Operators` Φ_k (Equation 3.2).

In the stochastic simulation approach, these `Operators` encode the `GeneProduct` production reactions ($* \rightarrow p_i$), with propensities a_i given by the sum of `Operator` synthesis rates,

$$a_i = \sum_j \Phi_j(\vec{r}), \tag{3.3}$$

where the sum is over all `Operators` that synthesise `GeneProduct` i. The degradation rate γ_i and growth rate $\mu(t)$ determine the propensities b_i of the `GeneProduct` extinction reactions ($p_i \rightarrow *$),

$$b_i = \gamma_i + \mu(t). \tag{3.4}$$

The `Operator` is the set of genetic parts that regulate gene expression (Figure 3.2 A). At its core is a promoter containing repressor or activator binding sites, such that the input `Regulator` either increases or decreases transcription and thus the gene expression rate (Figure 3.2 B).

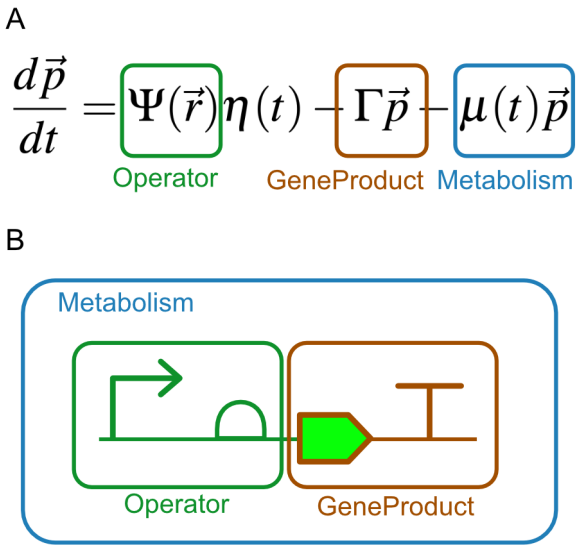


Fig. 3.2 Relation between the mathematical and computational model. (A) In the equation $\frac{d\vec{p}}{dt}$ is the dynamics of the $\vec{p}$ vector of proteins over time, $\Psi(\vec{r})$ is the protein synthesis rate or expression rate linked to the `Operator` and may depend in regulator proteins from the vector of regulators $\vec{r}$. $\eta(t)$ is a function of time that adjust gene expression for time and context creating a profile or time-series. $\Gamma\vec{p}$ is the vector of degradation rates of $\vec{p}$ linked to `GeneProducts`. $\mu(t)\vec{p}$ is the growth rate effect on the dilution of proteins from the vector of proteins $\vec{p}$ linked to `Metabolism`. (B) Diagrammatic representation of the computational model used in LOICA. TUs can be splitted and represented by an `Operator` and a `GeneProduct`, both are encapsulated by `GeneticNetwork` and `Metabolism` sets the context where the genes are expressed. Both are encapsulated by `Sample` to run simulations.

Logical `Operators` can thus be instantiated as genetic devices that are regulated by input `Regulators` and output `GeneProduct` synthesis rates or gene expression rates. As well as the 1-input and 2-input `Operators` described below,

Source Operator

The `Source` is an `Operator` that represents constitutive expression. `Source` synthesis rate follows an input profile of gene expression rate over time. The input profile can be synthetic or characterised from experimental data.

$$\Phi_{(t)} \tag{3.5}$$

Where Φ is the gene expression rate, and t is time.

An implementation example is the constitutive promoter J23101 from the Anderson collection in the iGEM distribution (Figure 3.3).

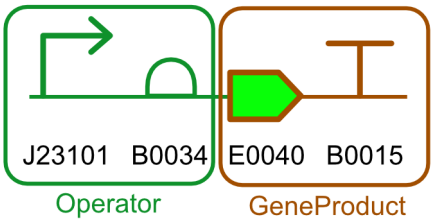


Fig. 3.3 SBOL visual diagram of a TU containing the promoter J23101. The SBOL components contained in the `Operator` and `GeneProduct` are in the green and brown boxes respectively.

Receiver Operator

The `Receiver` is an `Operator` that represents externally controlled expression through chemical supplementation. `Receiver` expression rate follows a Hill equation as a transfer function that maps `Supplement` concentration to `GeneProduct` expression rate.

$$\Phi(s) = \frac{\alpha_0 + \alpha_1 \left(\frac{s}{K}\right)^n}{1 + \left(\frac{s}{K}\right)^n} \tag{3.6}$$

Where Φ is the gene expression rate, α_0 is the unregulated or basal expression rate, α_1 is the regulated expression rate scalar, s is the `Supplement` concentration, K is the concentration of `Supplement` needed for half expression rate or switching concentration and n is the cooperativity degree.

An implementation example is the LuxR and HSL regulated promoter BBa_R0062, also known as pLux, from the iGEM distribution (Figure 3.4).

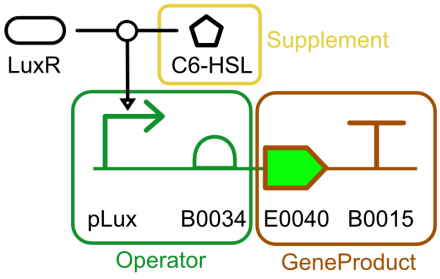


Fig. 3.4 SBOL visual diagram of a TU containing the HSL regulated promoter pLux. The SBOL components contained in the `Operator`, `GeneProduct` and `Supplement` are in the green, brown and yellow boxes respectively.

Hill1 Operator

The `Hill1` is an `Operator` that represents internally controlled expression through one transcriptional regulator. `Hill1` expression rate follows a Hill equation as a transfer function that maps `Regulator` concentration to `GeneProduct` expression rate.

$$\Phi(r) = \frac{\alpha_0 + \alpha_1 \left(\frac{r}{K}\right)^n}{1 + \left(\frac{r}{K}\right)^n} \tag{3.7}$$

Where Φ is the gene expression rate, α_0 is the unregulated or basal expression rate, α_1 is the regulated expression rate scalar, s is the `Regulator` concentration, K is the concentration of `Regulator` needed for half expression rate or switching concentration and n is the cooperativity degree. Depending on the parameters α_0 and α_1 this `Operator` may encode NOT logic ($\alpha_0 > \alpha_1$) or a Buffer ($\alpha_0 < \alpha_1$).

An implementation example is the LacI regulated promoter R0010, also known as pLac, from the iGEM distribution (Figure 3.5).

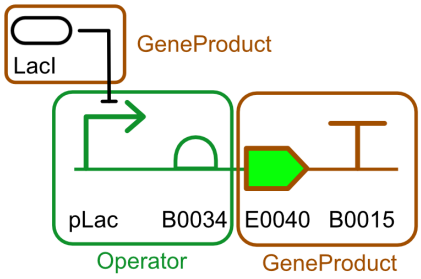


Fig. 3.5 SBOL visual diagram of a TU containing the repressible promoter pLac. The SBOL components contained in the `Operator` and `GeneProduct` are in the green and brown boxes respectively.

Hill2 Operator

The `Hill2` is an `Operator` that represents internally controlled expression through two transcriptional regulators. `Hill2` expression rate follows a two input modification of the Hill equation as transfer function that maps `Regulator` concentration to `GeneProduct` expression rate. Depending on the parameters α_0, α_1, α_2, and α_3 the `Operator` may encode a range of logic, including the NOR operation when $\alpha_0 > \alpha_1, \alpha_2, \alpha_3$.

$$\Phi(r_1, r_2) = \frac{\alpha_0 + \alpha_1 \left(\frac{r_1}{K_1}\right)^{n_1} + \alpha_2 \left(\frac{r_2}{K_2}\right)^{n_2} + \alpha_3 \left(\frac{r_1}{K_1}\right)^{n_1} \left(\frac{r_2}{K_2}\right)^{n_2}}{1 + \left(\frac{r_1}{K_1}\right)^{n_1} + \left(\frac{r_2}{K_2}\right)^{n_2} + \left(\frac{r_1}{K_1}\right)^{n_1} \left(\frac{r_2}{K_2}\right)^{n_2}} \tag{3.8}$$

An implementation example is the NOR gate, tandem promoters P_{Tac} and P_{PhiF} from the Cello collection [123](Figure 3.6).

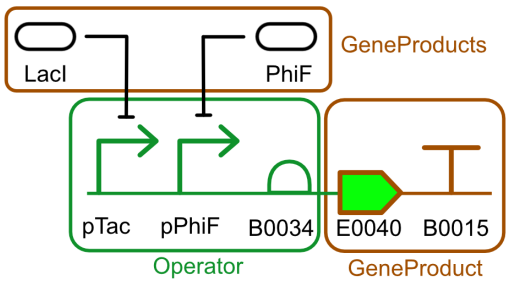


Fig. 3.6 SBOL visual diagram of a TU containing the tandem promoter P_{Tac} and P_{PhiF}. The SBOL components contained in the `Operator` and `GeneProduct` are in the green and brown boxes respectively.

The characterization for any `Operator` can be done automatically using LOICA characterise method which uses its two-way communication with Flapjack to get experimental data.

LOICA currently cannot represent networks with nodes with more than two inputs, but all `Operators` can drive multiple outputs `GeneProducts`. However, note that LOICA can be used to define an `Operator` as any operation that maps from input `Regulator` concentration to output synthesis rate, which may correspond to different genetic implementations than those described here. Thus, by expanding the range of `Operator` classes, in future LOICA could be extended to represent a larger range of genetic networks.

3.2.3 Software design and architecture

LOICA was developed in Python 3 as a package distributed through the Python package index (PyPI). Python is one of the most popular programming languages and is well known by engineers and scientists. The development was done using Git following the best practices for open source developers. The repository is publicly available at https://github.com/RudgeLab/LOICA under MIT license and the contributor covenant code of conduct. The docstrings were written following the NumPy style over reStructuredText and Google because its format is more amenable and readable, displaying information about input name, type and description. The documentation were built with Sphinx and is publicly available on https://loica.readthedocs.io/

The `Operator` and `GeneProduct` are abstract classes . `Operators` change their expression rate which modulates `GeneProducts` production. The `Source` is an `Operator` that changes its gene expression according to a profile of constitutive expression. The `Receiver`

is an Operator that changes its gene expression according to an input Supplement following a transfer function. The Hill1 is an Operator that changes its gene expression according to an input Regulator following a transfer function. The Hill2 is an Operator that changes its gene expression according to two inputs Regulators following a transfer function. GeneProducts have a fixed degradation rate that modulates GeneProduct degradation or destruction. The Regulator is a GeneProduct that can be the input of an Operator. The Reporter is a GeneProduct that can be measured or emits a signal. The GeneticNetwork class encapsulates Operators and GeneProducts. GeneticNetworks generate graph representations using NetworkX and SBOL representations using pySBOL3. It can also plot the graphs using NetworkX [68] visualisation tools in a full or contracted way. Default parameters for network visualisation were set by CV.

The Supplement is an experimentally controllable class that is the input of a Receiver. The Metabolism class changes the biomass and growth rate over time. The Sample class encapsulates GeneticNetwork, Metabolism and Supplements. The Assay class encapsulates Samples. Assay has two simulation algorithms, a deterministic simulation using ODEs with noise and a stochastic simulation using the Gillespie algorithm. The Colony class encapsulates GeneticNetwork. Similar to an Assay with one Sample it runs deterministic simulations of the spatio-temporal pattern of a growing colony using PDEs.

Furthermore, LOICA optionally can be connected to SBOL and therefore to SynBioHub. Operators and GeneProducts have an optional argument to add a SBOL3 Component GeneticNetwork can be converted to SBOL3 compiling TUs sequences from Operators and GeneProducts. Interactions and models are added to the SBOL document. Moreover, LOICA optionally can be connected to Flapjack. Reporter has an argument for Signal Flapjack ID. GeneticNetwork has an argument for Vector Flapjack ID. Assay, Sample and Supplement have an argument for their counterparts in Flapjack connected through IDs as well.

3.2.4 Encoding designs in SBOL3

LOICA generates a SBOL standardised representation of the designed genetic network. The SBOL representation is created using pySBOL3 and can append the created objects to an existing SBOL Document or create a new one. The only condition for this method is that all the Operators and GeneProducts contained in the GeneticNetwork must have a SBOL Component attribute. As the SBOL Components needed for Operators and GeneProducts are composites, builder functions were developed to facilitate and make it more intuitive creating them.

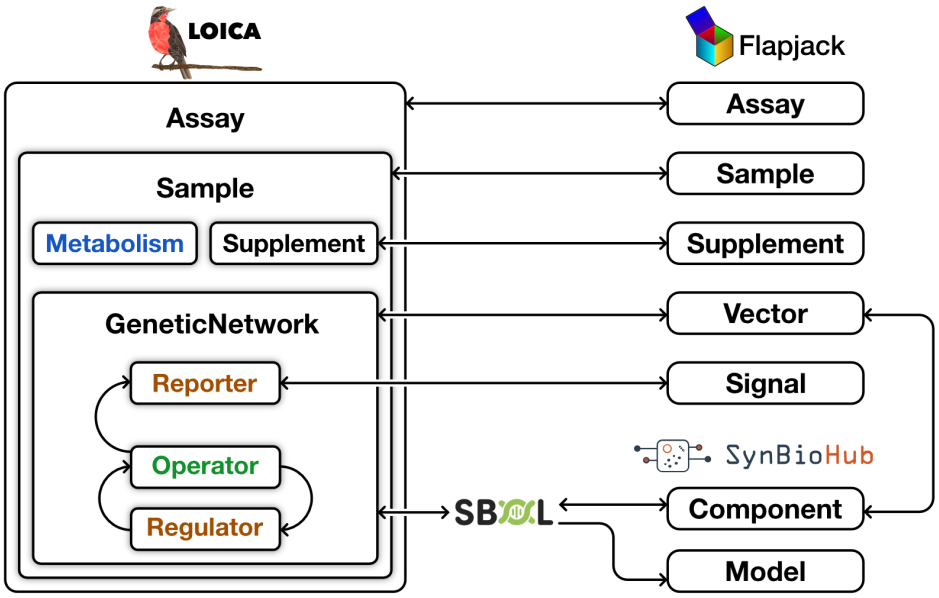


Fig. 3.7 Objects relationship. Diagram of an `Assay` encapsulating a `Sample` which in turn encapsulates `Metabolism`, `Supplement`, and `GeneticNetwork`. `GeneticNetwork` encapsulates the `Operator` and `Regulator` which generate a model through their interactions. On the right side the different interactions with the Flapjack and SBOL models are shown.

`GeneticNetwork` has the method to generate the SBOL representation. The whole genetic network is represented with a SBOL `Component` of type `SBO DNA` and role `SO ENGINEERED REGION`. For each `Operator` a TU SBOL `Component` of type `SBO DNA` and role `SO ENGINEERED REGION` is instantiated. The SBOL `Components` encapsulated by the `Operator` and the output `GeneProducts` are used to create SBOL `SubComponents` that are appended to the TU `Component` features. Then the order of the `Operator` followed by the `GeneProducts` is fixed. For each `GeneProduct` a molecular representation `Component` of type `SBO PROTEIN` or SBO RNA is created.

`GeneProducts` can be either a `Regulator` with the role `SO TRANSCRIPTION FACTOR` or a `Reporter` with the role `NCIT MEASURABLE`. Then a `Participation` of role `SBO TEMPLATE` is created with the `SubComponents` representing the `GeneProduct` DNA and a `Participation` of role `SBO PRODUCT` is created with the `SubComponents` representing the `GeneProduct` RNA or Protein. These two `Participations` are part of a `Interaction` of type `SBO GENETIC PRODUCTION` that is appended to the TU `Component` interactions. Then for each `Operator` input, its molecular representation is is used as participant of a `Participation` of role `SBO INHIBITOR` or `Operator` and the depending on the alpha parameter and the `Operator` DNA representation is used as the corresponding `Participation` of role `SBO INHIBITED` or `SBO STIMULATED`. An appropriate `Interaction` of type `SBO`

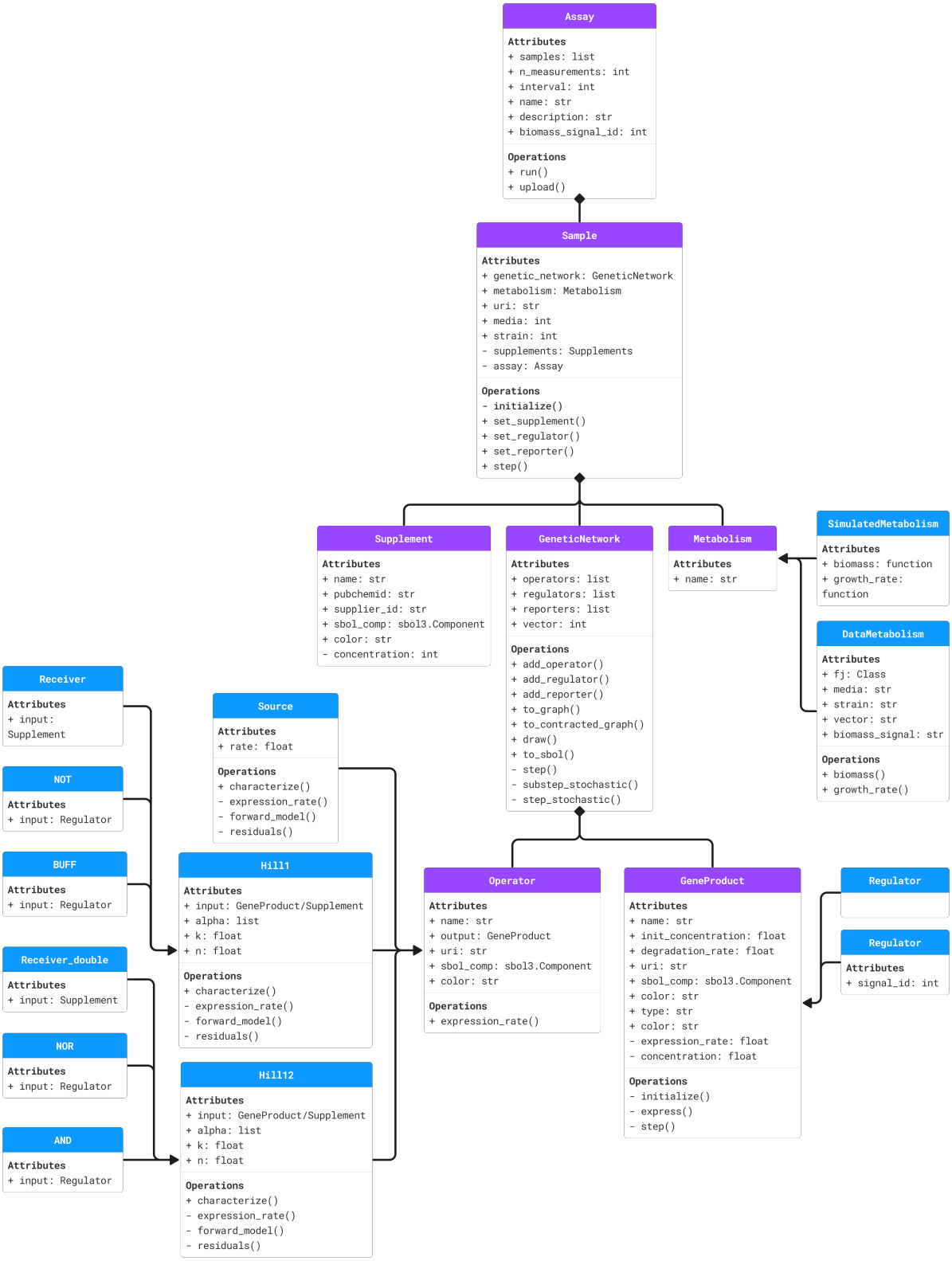


Fig. 3.7 Class Diagram. Purple boxes represent Abstract Class entities and blue boxes represent Class entities. Diamond end arrows represents composition relationships, and triangular end arrows represents inheritance relationships.

INHIBITION or SBO STIMULATION is created with these Participations and appended to the TU Component interactions.

Finally, a Model is created indicating the source, language and framework and appended to the TU Component models. All the TU Components are added as SubComponent to the genetic network Component and its position is fixed.

3.2.5 Genetic network design and simulation examples

Source

The simplest genetic network is one gene with constitutive expression. To design a genetic network of a constitutive TU a Genetic Network were instantiated. Then, a GFP Reporter was created and added to the Genetic Network. This component allows for the tracking of the molecular species concentration for visualisation. Finally, a Source was created, with the GFP Reporter as output, and added to the Genetic Network. This forms a TU that produces GFP in a constitutive fashion. Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networ tree/main/notebooks

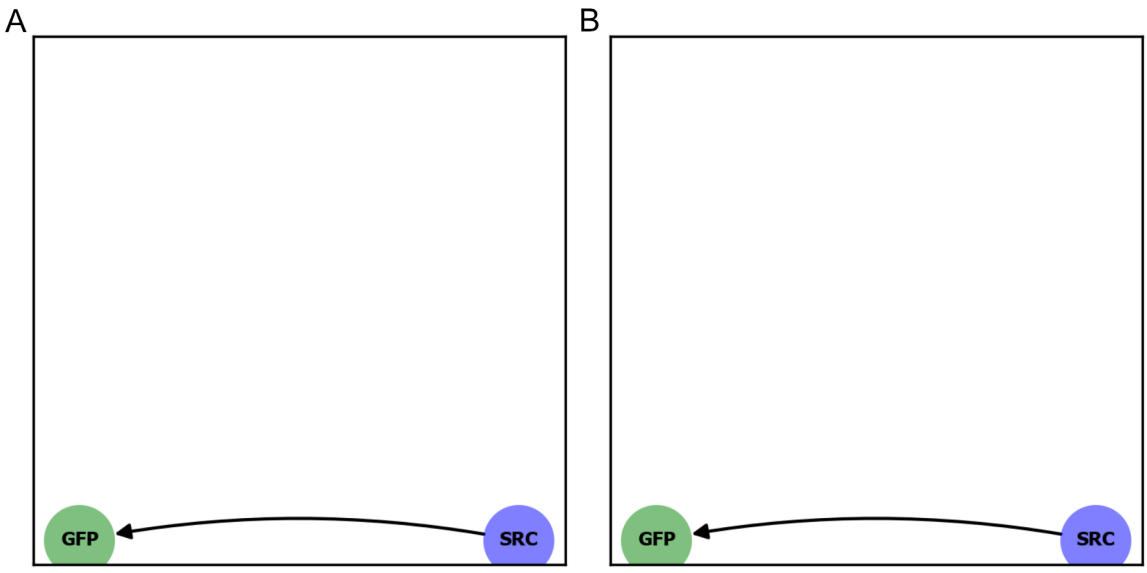


Fig. 3.8 Source genetic network. (A) Full genetic network diagram. (B) Contracted genetic network diagram. As Operators and Reporters are shown in the both, the visualzation is the same.

```
src = lc.GeneticNetwork(vector=vector.id[0])
```

```
gfp_rep = lc.Reporter(name='GFP', degradation_rate=1,
                      signal_id=gfp.id[0], sbol_comp=gp_gfp,
                      color='green')
src.add_reporter(gfp_rep)

j23101_src_gfp = lc.Source(output=gfp_rep, rate=10, sbol_comp=op_j23101)
src.add_operator(j23101_source_gfp)

src.draw()
```

The draw method plots the network representation of the graph generated by the `GeneticNetwork` (Figure 3.8 A) and has an argument to plot the contracted version of the graph (Figure 3.8 B).

To setup the assay for simulations of a well with bacteria a `SimulatedMetabolism` was created using the Gompertz growth model that describes the dynamics of a population of bacteria over time. Then, a `Sample` was created, containing the `GenenticNetwork`, `Metabolism`. Finally, an `Assay` was created containing the `Sample`, and defining the number of measurements and interval between them. Simulations were performed using the run method with defaults values (Figure 3.9 A) and using with NSR 10^{-3} (Figure 3.9 B). Single cell simulations were performed using a `SimulatedMetabolism` with growth rate 0 and biomass 1. Then, a `Sample` was created, containing the `GenenticNetwork` and `Metabolism`. Finally, an `Assay` was created containing the `Sample`, and defining the number of measurements and interval between them. Simulations were performed using the run method with stochastic equal True (Figure 3.9 C). Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

```
metab = lc.SimulatedMetabolism(name='LOICA metab',
            biomass=biomass, growth_rate=growth_rate)

sample = lc.Sample(genetic_network=src,
                   metabolism=metab,
                   media=media.id[0],
                   strain=strain.id[0]
                  )
assay = lc.Assay([sample],
                 n_measurements=100,
                 interval=0.25,
```

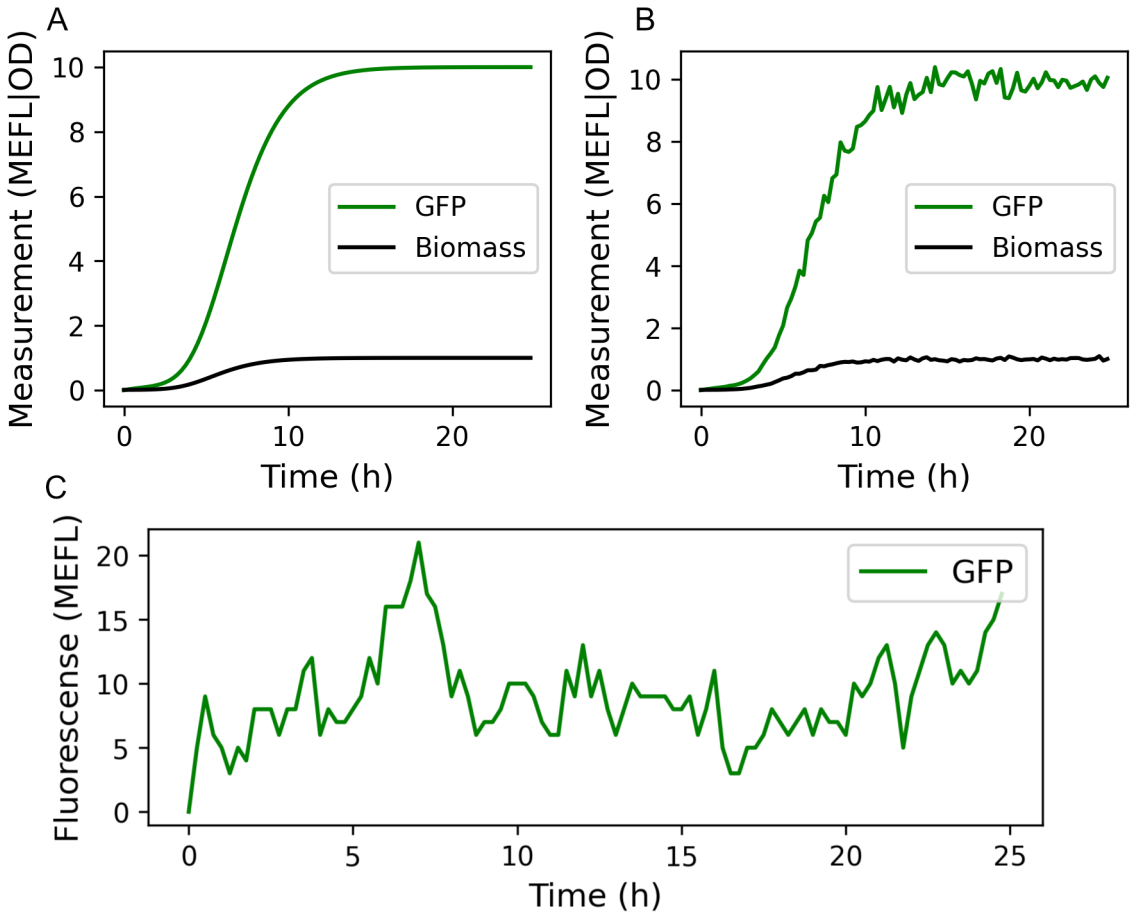


Fig. 3.9 Source simulations. (A) Source GFP and biomass profiles simulated with LOICA using Gompertz metabolism and ODEs. (B) Source GFP and biomass profiles simulated with LOICA using Gompertz metabolism and ODE NSR 10^{-3}. (C) Source GFP profile simulated with LOICA using growth rate 0, biomass 1 and the Gillespie algorithm.

```
            name='LOICA Source (SRC) simulation',
            description='Simulated constitutive expression',
            biomass_signal_id=biomass_signal.id[0]
        )
assay.run()
```

Receiver

A useful genetic network is a receiver, a gene that senses an experimentally accessible chemical and changes its gene expression in response. In this example the higher the concentration of the external chemical, the higher the gene expression of the receiver. To design a genetic network of a receiver TU a `Genetic Network` were instantiated. Then, a GFP `Reporter` was

created and added to the `Genetic Network`. This component allows for the tracking of the molecular species concentration for visualisation. An AHL inducer `Supplement` was created as the external chemical. Finally, a `Receiver` was created, with AHL `Supplement` as input and GFP `Reporter` as output, and added to the `Genetic Network`. This forms a TU that produces GFP in response to AHL. Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

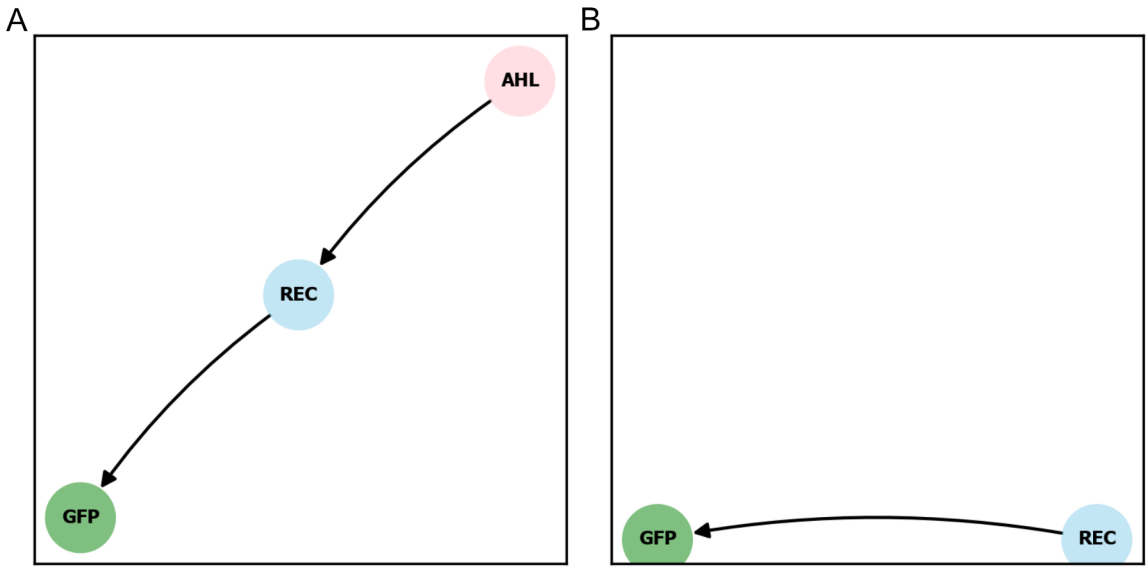


Fig. 3.10 Receiver genetic network.(A) Full genetic network diagram. (B) Contracted genetic network diagram. As `Operators` and `Reporters` are shown in the contracted network the only difference is the `Supplement`.

```
rec = lc.GeneticNetwork(vector=vector.id[0])

gfp_rep = lc.Reporter(name='GFP', degradation_rate=1,
                signal_id=gfp.id[0], sbol_comp=gp_gfp,
                color='green')
rec.add_reporter(gfp_rep)

ahl1 = lc.Supplement(name='AHL', sbol_comp=hsl_c4)

ahl1_REC_gfp = lc.Receiver(input=ahl1, output=gfp_rep,
                alpha=[100,0], K=1, n=2, sbol_comp=op_prhl)
rec.add_operator(ahl1_REC_gfp)
```

```
rec.draw()
```

The draw method plots the network representation of the graph generated by the `GeneticNetwork` (Figure 3.10 A) and has an argument to plot the contracted version of the graph (Figure 3.10 B).

To setup the assay for simulations of a well with bacteria a `SimulatedMetabolism` was created using the Gompertz growth model that describes the dynamics of a population of bacteria over time. Then, a list of `Samples` were created, containing the `GeneticNetwork`, `Metabolism` and `Supplement` at different concentrations. Finally, an `Assay` was created containing the set of `Samples`, and defining the number of measurements and interval between them. Simulations were performed using the run method with default values (Figures 3.11 and 3.12 A). Variants of the `Receiver` were done by changing their parameters and following the steps described before. The alpha parameters were changed from [0,100] to [20,200], this changed and extended the values for ON and OFF (Figure 3.12 B). The K parameter was changed from 1 to 10, this moves the response curve to the right as the switching point is at a higher concentration (Figure 3.12 C). The n parameter was changed from 2 to 4, this makes the transition between states to be faster or sharper (Figure 3.12 D). Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

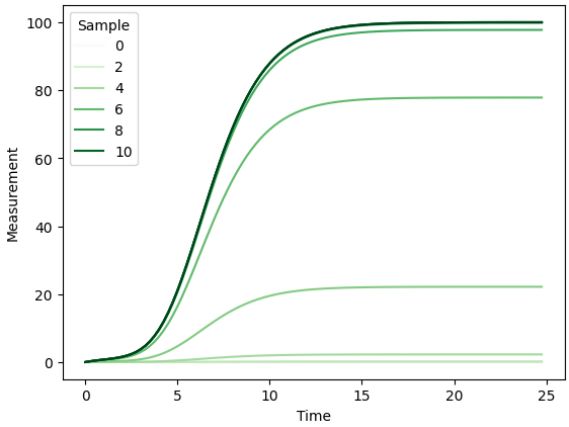


Fig. 3.11 Receiver expression profiles. Receiver GFP profiles simulated with LOICA using Gompertz metabolism and ODEs over different concentrations of inducer.

```
metab = lc.SimulatedMetabolism(name='LOICA metab',
            biomass=biomass, growth_rate=growth_rate)
```

```
samples = []
for conc1 in np.logspace(-3, 3, 12):
    sample = lc.Sample(genetic_network=rec, metabolism=metab,
                       media=media.id[0], strain=strain.id[0])
    sample.set_supplement(ahl1, conc1)
    samples.append(sample)

assay = lc.Assay(samples,
                 n_measurements=100,
                 interval=0.25,
                 name='LOICA Source (REC) simulation',
                 description='Simulated Receiver generated by LOICA',
                 biomass_signal_id=biomass_signal.id[0]
                )
assay.run()
```

Inverter

An inverter or NOT gate implements the logical negation. An inverter genetic network is a gene that senses an internal molecule and changes its gene expression in response. In this example the higher the concentration of the internal molecule, the lower the gene expression of the inverter. To design a genetic network of an inverter a `Genetic Network` were instantiated. An AHL inducer `Supplement` was created as the external chemical. Then, GFP and RFP `Reporters` were created and added to the `Genetic Network`. This component allows for the tracking of the molecular species concentration for visualisation. A `Receiver` were created, with AHL `Supplement` as input, and GFP `Reporter` and LacI `Regulator` as output. Both were added to the `Genetic Network`. This forms a TU that produces GFP and LacI in a bicistronic fashion induced by the AHL. Finally, a `Hill1` was created, with LacI `Regulator` as input and RFP `Reporter` as output. This forms a TU that produces RFP constitutively and gets repressed by LacI following the NOT logic. The `Receiver` and `Hill1` were added to the `Genetic Network`. Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

```
inv = lc.GeneticNetwork(vector=vector.id[0])
```

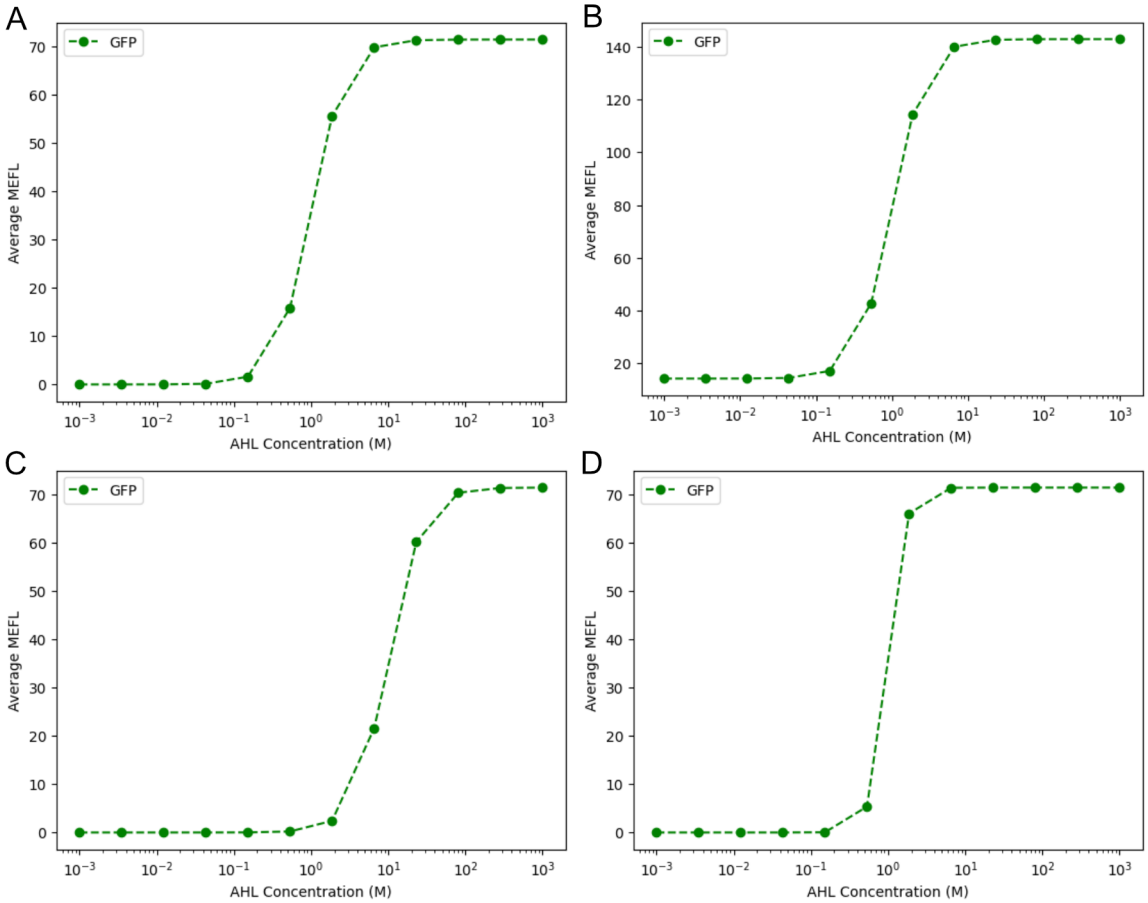


Fig. 3.12 Receiver transfer curves. Average MEFL over profile expression at simulated samples with different concentration of inducer. (A) Parameters used in the `Receiver`: alpha = [0,100], K = 1, n = 2. (B) Parameters used in the `Receiver`: alpha = [20,200], K = 1, n = 2. (C) Parameters used in the `Receiver`: alpha = [0,100], K = 10, n = 2. (D) Parameters used in the `Receiver`: alpha = [0,100], K = 1, n = 4.

```
ahl1 = lc.Supplement(name='AHL', sbol_comp=hsl_c4)

gfp_rep = lc.Reporter(name='GFP', degradation_rate=1, signal_id=gfp.id[0],
            sbol_comp=gp_gfp, color='green')
rfp_rep = lc.Reporter(name='RFP', degradation_rate=1, signal_id=rfp.id[0],
            sbol_comp=gp_rfp, color='red')
inv.add_reporter([gfp_rep, rfp_rep])

laci_reg = lc.Regulator(name='LacI', degradation_rate=1, init_concentration=5,
            sbol_comp=gp_laci)
```

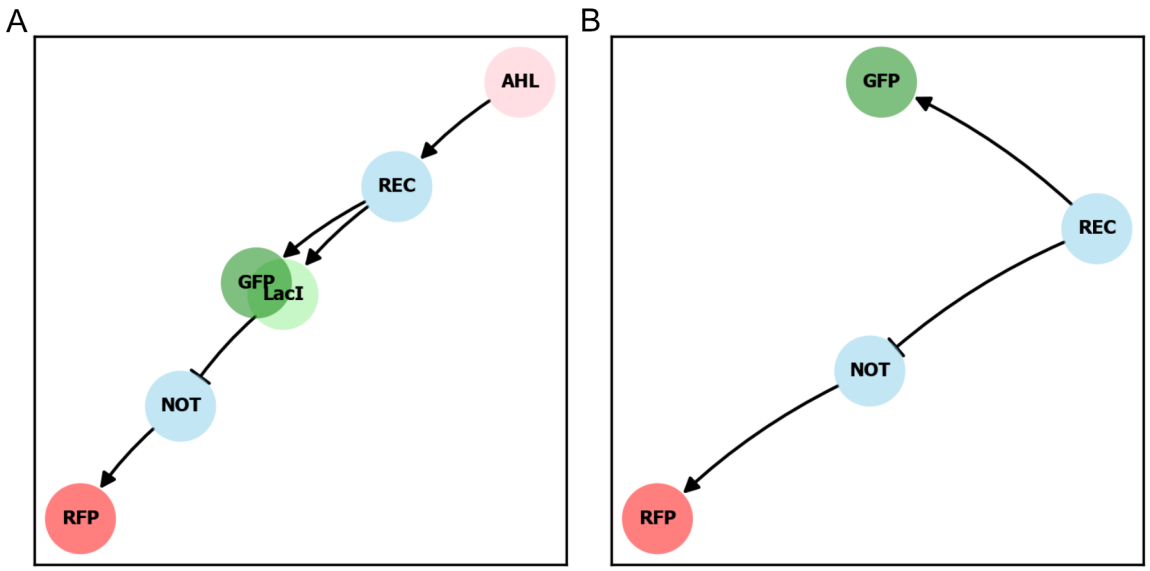


Fig. 3.13 Inverter genetic network. (A) Full genetic network diagram. (B) Contracted genetic network diagram.

```
inv.add_regulator(laci_reg)

ahl1_REC_laci_gfp = lc.Receiver(input=ahl1, output=[laci_reg, gfp_rep],
            alpha=[0,100], K=1, n=2, sbol_comp=op_prhl)
laci_NOT_rfp = lc.Hill1(name='NOT', input=laci_reg, output=rfp_rep,
            alpha=[100,0], K=1, n=2, sbol_comp=op_plac)
inv.add_operator([ahl1_REC_laci_gfp, laci_NOT_rfp])

inv.draw()
```

The draw method plots the network representation of the graph generated by the `GeneticNetwork` (Figure 3.13 A) and has an argument to plot the contracted version of the graph (Figure 3.13 B).

To setup the assay for simulations of a well with bacteria a `SimulatedMetabolism` was created using the Gompertz growth model that describes the dynamics of a population of bacteria over time. Then, a list of `Samples` were created, containing the `GeneticNetwork`, `Metabolism` and `Supplement` at different concentrations. Finally, an `Assay` was created containing the set of `Samples`, and defining the number of measurements and interval between them. Simulations were performed using the run method with default values (Figures 3.14 A,B and 3.15 A). The `Receiver` were the same across all the simulations, and variants of the `Hill1` were done by changing their parameters and following the steps described

before. The alpha parameters were changed from [0,100] to [20,200], this changed and extended the values for ON and OFF (Figure 3.15 B). The K parameter was changed from 1 to 10, this moves the response curve to the right as the switching point is at a higher concentration (Figure 3.15 C). The n parameter was changed from 2 to 4, this makes the transition between states to be faster or sharper, decreasing the dynamic range (Figure 3.15 D). Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

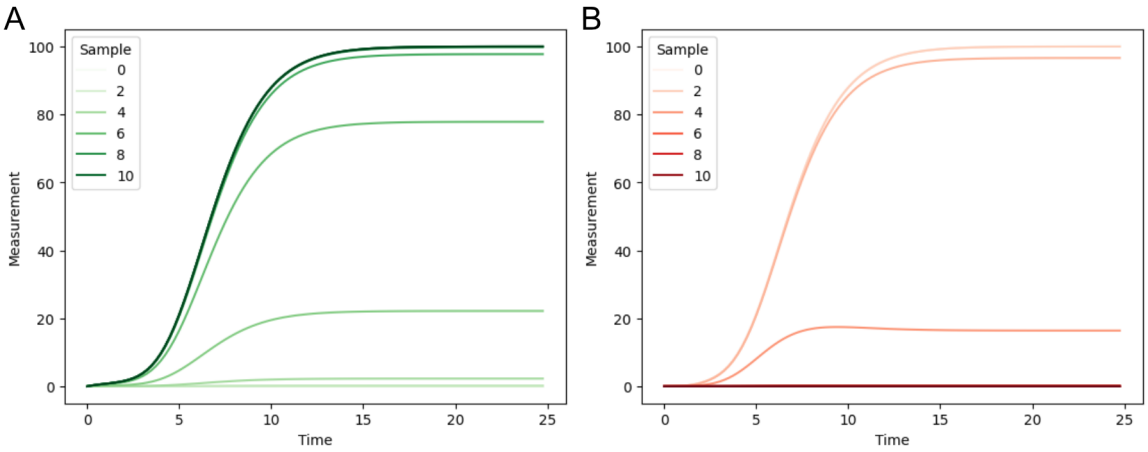


Fig. 3.14 Inverter expression profiles. (A). Receiver GFP profiles simulated with LOICA using Gompertz metabolism and ODEs over different concentrations of inducer. (B) Inverter RFP profiles simulated with LOICA using Gompertz metabolism and ODEs over different concentrations of inducer.

```
metab = lc.SimulatedMetabolism(name='LOICA metab', biomass=biomass, growth_rate=gro

samples = []
for conc1 in np.logspace(-3, 3, 12):
    sample = lc.Sample(genetic_network=inv, metabolism=metab,
                       media=media.id[0], strain=strain.id[0])
    sample.set_supplement(ahl1, conc1)
    samples.append(sample)

assay = lc.Assay(samples,
                 n_measurements=100,
                 interval=0.25,
                 name='LOICA Inverter simulation',
```

```
            description='Simulated Inverter generated by LOICA',
            biomass_signal_id=biomass_signal.id[0]
        )
assay.run()
```

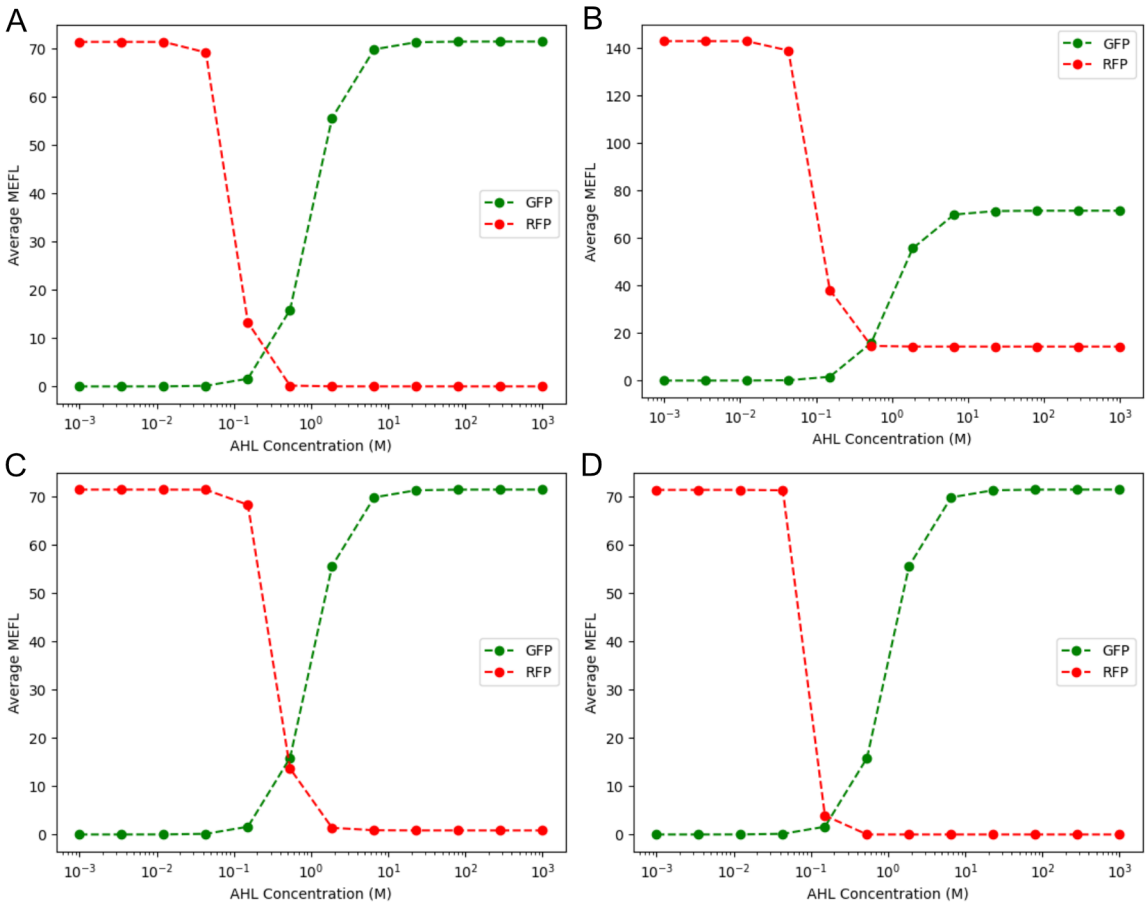


Fig. 3.15 Inverter transfer curves. Average green and red fluorescence profile expression at simulated samples with different concentrations of inducer. In this simulation a `Receiver` represents a TU that senses a biochemical signal and produces a the `Regulator` LacI and the `Reporter` GFP. The `Hill1` represents a TU that senses the `Regulator` LacI and in turn express the `Reporter` RFP. The `Receiver` parameters are fixed alpha = [0,100], K = 1, n = 2. (A) Parameters used in the `Hill1`: alpha = [0,100], K = 1, n = 2. (B) Parameters used in the `Hill1`: alpha = [20,200], K = 1, n = 2. (C) Parameters used in the `Hill1`: alpha = [0,100], K = 10, n = 2. (D) Parameters used in the `Hill1`: alpha = [0,100], K = 1, n = 4.

Toggle switch

A toggle switch is a bistable genetic network formed by two mutually repressing genes. A toggle switch genetic network remains in a state after a pulse stimulus which can be changed with another pulse encoding a memory module. To design a bistable genetic network a `Genetic Network` were instantiated. Two `Regulators` were created TetR and LacI, and added to the `Genetic Network`. For symmetry breaking the initial concentration of LacI was set to 5. Then, GFP and RFP `Reporters` were created and added to the `Genetic Network`. This component allows for the tracking of the molecular species concentration for visualisation. A `Hill1` encoding the NOT logic was created, with LacI `Regulator` as input, and TetR `Regulator` and GFP `Reporter` as output. This forms a TU that produces TetR and GFP constitutively in a bicistronic fashion, and gets repressed by LacI following the NOT logic. A `Hill1` encoding the NOT logic was created, with TetR `Regulator` as input, and LacI `Regulator` and RFP `Reporter` as output. This forms a TU that produces LacI and RFP constitutively in a bicistronic fashion, and gets repressed by TetR following the NOT logic. Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

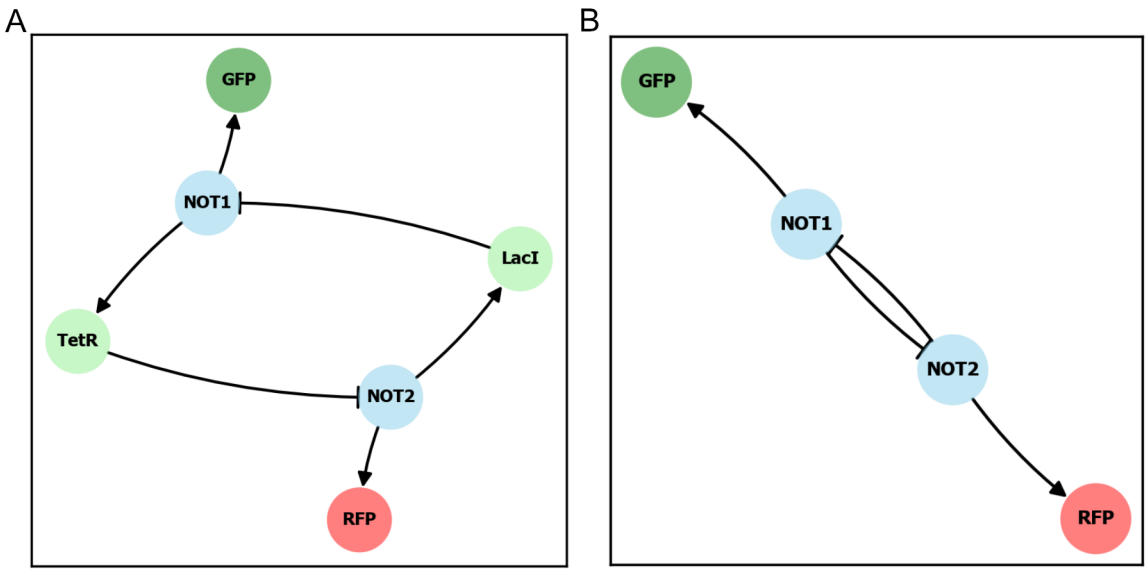


Fig. 3.16 Oscillator genetic network. (A) Full genetic network diagram. (B) Contracted genetic network diagram.

```
tgl = lc.GeneticNetwork(vector=vector.id[0])
```

```
tetr_reg = lc.Regulator(name='TetR', degradation_rate=1,
            sbol_comp=geneproducts[0])
laci_reg = lc.Regulator(name='LacI', degradation_rate=1,
            init_concentration=5, sbol_comp=geneproducts[1])
tgl.add_regulator([tetr_reg, laci_reg])

gfp_rep = lc.Reporter(name='GFP', degradation_rate=1, signal_id=gfp.id[0],
            sbol_comp=geneproducts[2], color='green')
rfp_rep = lc.Reporter(name='RFP', degradation_rate=1, signal_id=rfp.id[0],
            sbol_comp=geneproducts[3], color='red')
tgl.add_reporter([gfp_rep, rfp_rep])


laci_not_tetr_gfp = lc.Hill1(name='NOT1', input=laci_reg,
        output=[tetr_reg, gfp_rep], alpha=[10,0.01], K=1, n=2, sbol_comp=op_plac)
tetr_not_laci_rfp = lc.Hill1(name='NOT2', input=tetr_reg,
        output=[laci_reg, rfp_rep], alpha=[10,0.01], K=1, n=2, sbol_comp=op_ptet)
tgl.add_operator([laci_not_tetr_gfp, tetr_not_laci_rfp])

tgl.draw()
```

The draw method plots the network representation of the graph generated by the `GeneticNetwork` (Figure 3.16 A) and has an argument to plot the contracted version of the graph (Figure 3.16 B).

To setup the assay for simulations of a well with bacteria a `SimulatedMetabolism` was created using the Gompertz growth model that describes the dynamics of a population of bacteria over time. Then a `Sample` was created, containing `GenenticNetwork` and `Metabolism`. Finally, an `Assay` was created containing the `Sample`, and defining the number of measurements and interval between them. Simulations were performed using the run method with defaults values. Two genetic networks were constructed, one with LacI initial concentration set to 5 (Figure 3.17 A) and another with TetR initial concentration set to 5 (Figure 3.17 A). Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

```
metab = lc.SimulatedMetabolism(name='LOICA metab',
            biomass=biomass, growth_rate=growth_rate)
```

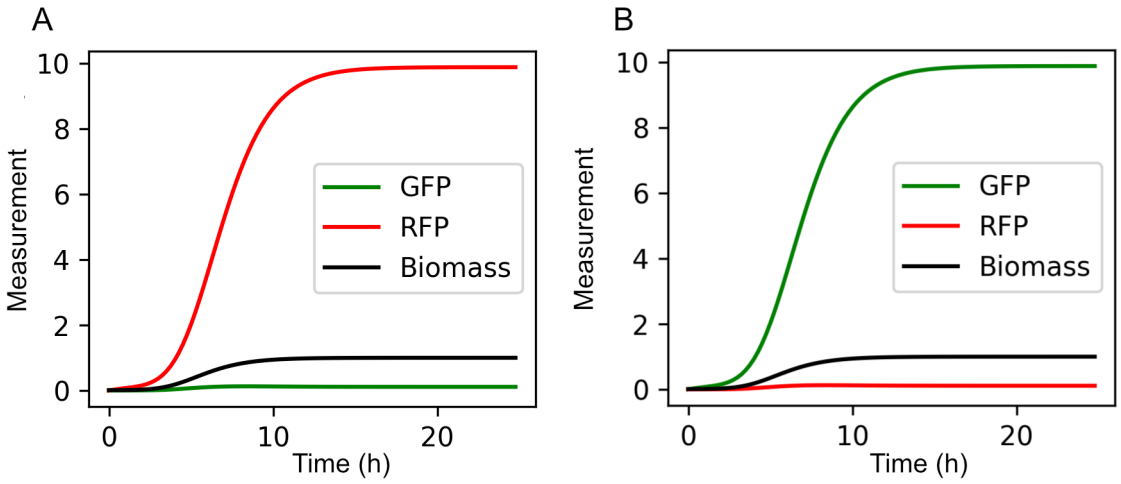


Fig. 3.17 Bistable genetic network simulations. (A) Bistable genetic network, simulated with LacI set to 5 as initial concentration. (B) Bistable genetic network, simulated with TetR set to 5 as initial concentration.

```
sample = lc.Sample(genetic_network=tgl,
                   metabolism=metab,
                   media=media.id[0],
                   strain=strain.id[0]
                  )
assay = lc.Assay([sample],
                 n_measurements=100,
                 interval=0.25,
                 name='LOICA toggle switch simulation',
                 description='Simulated bistable genetic network generated by LOICA',
                 biomass_signal_id=biomass_signal.id[0]
                )
assay.run()
```

Repressilator

A repressilator is a genetic network formed by three genes repressing each other in a ring fashion. A genetic network encoding a genetic ring oscillator similar to the repressilator can be done with three `Operators` repressed in a ring fashion. To design a genetic network of a ring oscillator a `Genetic Network` were instantiated. Three `Regulators` were created TetR, CI and LacI, and added to the `Genetic Network`. For symmetry breaking

the initial concentration of LacI was set to 5. Then, a GFP `Reporter` was created and added to the `Genetic Network`. This component allows for the tracking of the molecular species concentration for visualisation. A `Hill1` encoding the NOT logic was created, with TetR `Regulator` as input, and CI `Regulator` as output. This forms a TU that produces CI constitutively and gets repressed by TetR following the NOT logic. A `Hill1` encoding the NOT logic was created, with CI `Regulator` as input, and LacI `Regulator` as output. This forms a TU that produces CI constitutively and gets repressed by LacI following the NOT logic. A `Hill1` encoding the LacI NOT TetR logic was created, with LacI as input, and GFP `Reporter` and TetR `Regulator` as output. This forms a TU that produces TetR and GFP constitutively in a bicistronic fashion, and gets repressed by LacI following the NOT logic. Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

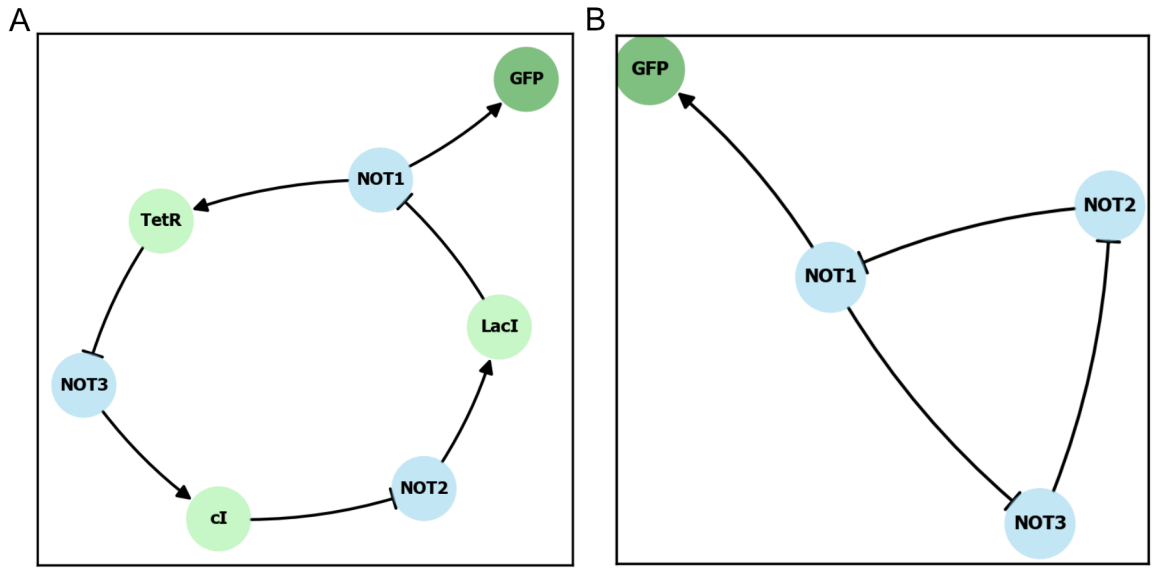


Fig. 3.18 Oscillator genetic network. (A) Full genetic network diagram. (B) Contracted genetic network diagram.

```
rep = lc.GeneticNetwork(vector=vector.id[0])

tetr_reg = lc.Regulator(name='TetR', degradation_rate=1,
            sbol_comp=geneproducts[1])
laci_reg = lc.Regulator(name='LacI', degradation_rate=1,
            init_concentration=5, sbol_comp=geneproducts[2])
ci_reg = lc.Regulator(name='cI', degradation_rate=1,
```

```
            sbol_comp=geneproducts[3])
rep.add_regulator([tetr_reg, laci_reg, ci_reg])

gfp_rep = lc.Reporter(name='GFP', degradation_rate=1, signal_id=gfp.id[0],
            sbol_comp=geneproducts[0], color='green')
rep.add_reporter(gfp_rep)

tetr_not_ci = lc.Hill1(name='NOT1', input=tetr_reg, output=ci_reg,
            alpha=[100,0], K=1, n=2, sbol_comp=op_ptet)
ci_not_laci = lc.Hill1(name='NOT2', input=ci_reg, output=laci_reg,
            alpha=[100,0], K=1, n=2, sbol_comp=op_plam)
laci_not_tetr_gfp = lc.Hill1(name='NOT3', input=laci_reg,
    output=[tetr_reg, gfp_rep], alpha=[100,0], K=1, n=2, sbol_comp=op_plac)
rep.add_operator([laci_not_tetr_gfp, ci_not_laci, tetr_not_ci])

rep.draw()
```

The draw method plots the network representation of the graph generated by the `GeneticNetwork` (Figure 3.18 A) and has an argument to plot the contracted version of the graph (Figure 3.18 B).

To setup the assay for simulations of a well with bacteria a `SimulatedMetabolism` was created using the Gompertz growth model that describes the dynamics of a population of bacteria over time. Then, a `Sample` was created, containing the `GeneticNetwork`, `Metabolism`. Finally, an `Assay` was created containing the `Sample`, and defining the number of measurements and interval between them. Simulations were performed using the run method with default values (Figure 3.19 A) and using NSR 10^{-3} (Figure 3.19 B). Single cell simulations were performed using a `SimulatedMetabolism` with growth rate 0 and biomass 1. Then, a `Sample` was created, containing the `GeneticNetwork` and `Metabolism`. Finally, an `Assay` was created containing the `Sample`, and defining the number of measurements and interval between them. Simulations were performed using the run method with stochastic equal True (Figure 3.19 C). Here is an example code, the full notebook is in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks

```
metab = lc.SimulatedMetabolism(name='LOICA metab',
            biomass=biomass, growth_rate=growth_rate)
```

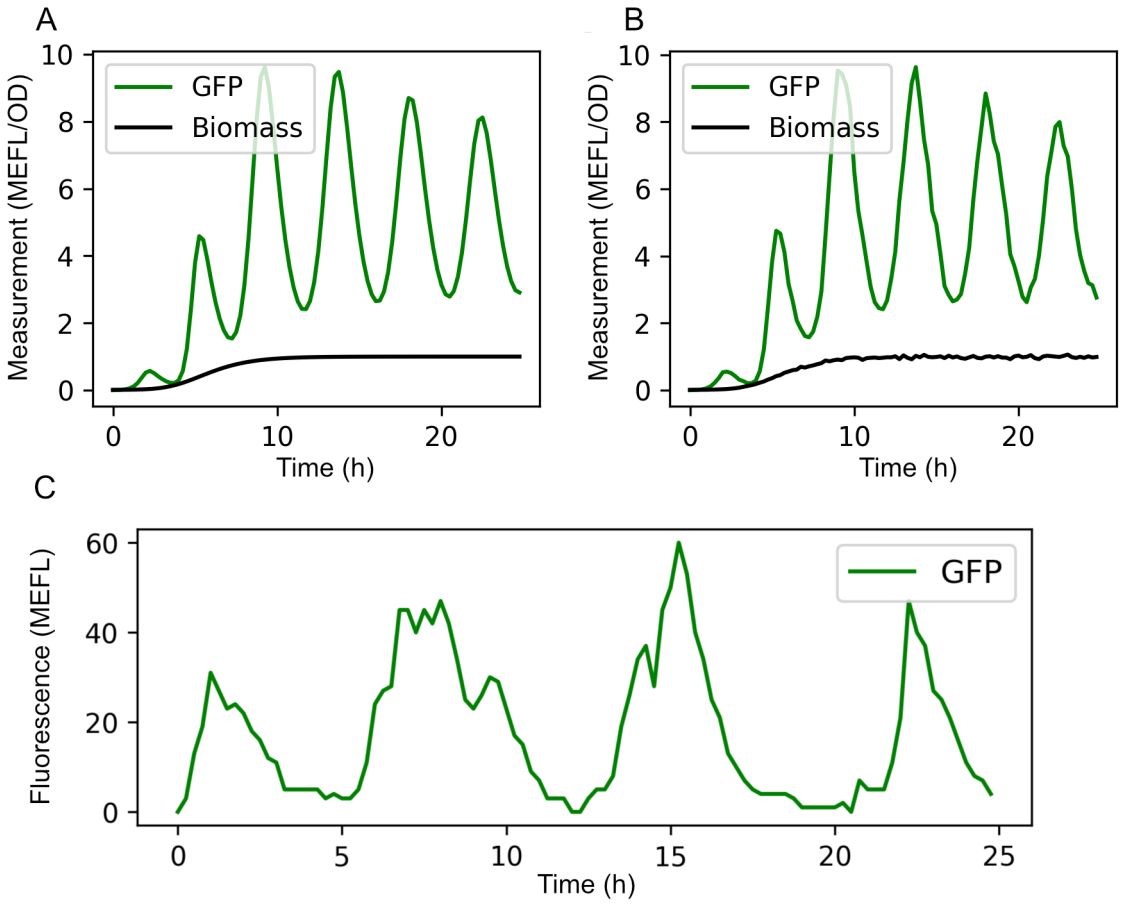


Fig. 3.19 Genetic oscillator simulations. (A) Oscillator GFP and biomass profiles simulated with LOICA using Gompertz metabolism and ODEs. (B) Oscillator GFP and biomass profiles simulated with LOICA using Gompertz metabolism and ODE NSR 10^{-3}. (C) Oscillator GFP profile simulated with LOICA using growth rate 0, biomass 1 and the Gillespie algorithm.

```
sample = lc.Sample(genetic_network=rep,
                   metabolism=metab,
                   media=media.id[0],
                   strain=strain.id[0]
                  )
assay = lc.Assay([sample],
                 n_measurements=100,
                 interval=0.25,
                 name='LOICA genetic oscillator simulation',
                 description='Simulated genetic oscillator generated by LOICA',
                 biomass_signal_id=biomass_signal.id[0]
```

```
            )
assay.run()
```

3.2.6 Operator model parameterization

To parameterize an `Operator` LOICA finds the values for the gene expression and growth rate models that better resemble experimental results using the characterise method. The characterise method is enabled by the two-way communication with Flapjack to access experimental data and analysis tools.

Instead of selecting the values for a profile, alpha, K and n, these can be extracted from the data using the characterise method. The `Receiver` characterise method uses information about the Flapjack instance, vector, media, strain, signal and biomass signal to request the data. It gets two DataFrames with background correction, the expression dataframe for the signal and the biomass dataframe for the biomass signal. These DataFrames are used as input for the residuals method that creates a function of a vector with the values for alpha0, alpha1, K and n for each `Sample`. Inside this loop the residuals method calls the forward model method creating an array of simulated data that is subtracted from an array with the experimental data and appended as residual. The residuals are used in the final part of the characterise method that uses least squares to obtain the alpha, K and n that better resembles the data.

The `Hill1` characterise method uses information about the Flapjack instance, `Receiver` vector, `Hill1` vector, media, strain, signal and biomass signal to request the data. It gets two DataFrames with background correction, the expression dataframe for the signal and the biomass dataframe for the biomass signal, and characterises a `Receiver`. These DataFrames and `Receiver` gene expression model values are used as input for the residuals method that creates a function of a vector with the values for alpha0, alpha1, K and n for each `Sample`. Inside this loop the residuals method calls the forward model method creating an array of simulated data that is subtracted from an array with the experimental data and appended as residual. The residuals are used in the final part of the characterise method that uses least squares to obtain the alpha, K and n that better resembles the data.

The `Hill2` characterise method uses information about the Flapjack instance, `Receiver` vector 1, `Receiver` vector 2, chemical 1, chemical 2, `Hill2` vector, media, strain, signal, gamma and biomass signal to request the data. It gets two DataFrames with background correction, the expression dataframe for the signal and the biomass dataframe for the biomass signal and characterises both `Receivers`. These dataframes, `Receivers` gene expression model values and gamma are used as input for the residuals method that creates a function of a vector with the values for alpha0, alpha1, K and n for each chemical and `Sample`. Inside

this loop the residuals method calls the forward model method creating an array of simulated data that is subtracted from an array with the experimental data and appended as residual. The residuals are used in the final part of the characterise method that uses least squares to obtain the alpha, K and n that better resembles the data.

I exemplify this process for a two-input *Operator* using simulated data (see example notebook https://github.com/RudgeLab/LOICA/blob/master/notebooks/Hill2.ipynb)

In order to characterise the two-input `Operator`, three auxiliary genetic networks are required. Two receiver `GeneticNetwork` composed of a `Receiver` with a `Supplement` as input, in this case C4 and C6 HSL respectively and a both with a GFP `Reporter` as output measurable signal. The last network includes the two previously described `Receivers` but now with a `Regulator` as output, instead of the GFP `Reporter`. In the example the `Receiver` whose input is C4 HSL outputs LacI and the `Receiver` which input is C6 HSL outputs TetR. These two `Regulators` are the input for the NOR `Hill2` which output is a GFP `Reporter` (Figure 3.20 A). LOICA were used to generate simulated kinetic, time-series data (Figure 3.20 B). Then, the simulated data were uploaded to Flapjack (Figure 3.20 C). Finally, the characterise method of the NOR `Hill2` were run, which in turn run the characterise method on each of the `Receivers` to parameterize its gene expression model (Figure 3.20 D). In Chapter 5 I develop a mathematical method to characterise microbial gene expression and growth rates.

3.3 Methods

3.3.1 Software development

LOICA was developed in Python 3, its dependencies are Numpy [70], Scipy [181], NetworkX [68], Pandas [111], pySBOL3 [117], Tyto [8], tqdm [39]. Software source code is publicly available under MIT licence at https://github.com/RudgeLab/LOICA. The documentation is available at https://loica.readthedocs.io and the package is distributed through PyPI https://pypi.org/project/loica. Designs and simulations were performed on a Apple MacBook Pro 15-inch, 2019, with a 2,6 GHz 6-Core Intel Core i7 processor, Radeon Pro 555X 4 GB and Intel UHD Graphics 630 1536 MB GPUs, and 16 GB 2400 MHz DDR4 RAM.

3.3.2 Noise in Kinetic Gene Expression and Biomass Simulations

Noise and background were added according to the following equations with $B' = 0.1$ and $y' = 0.1$,

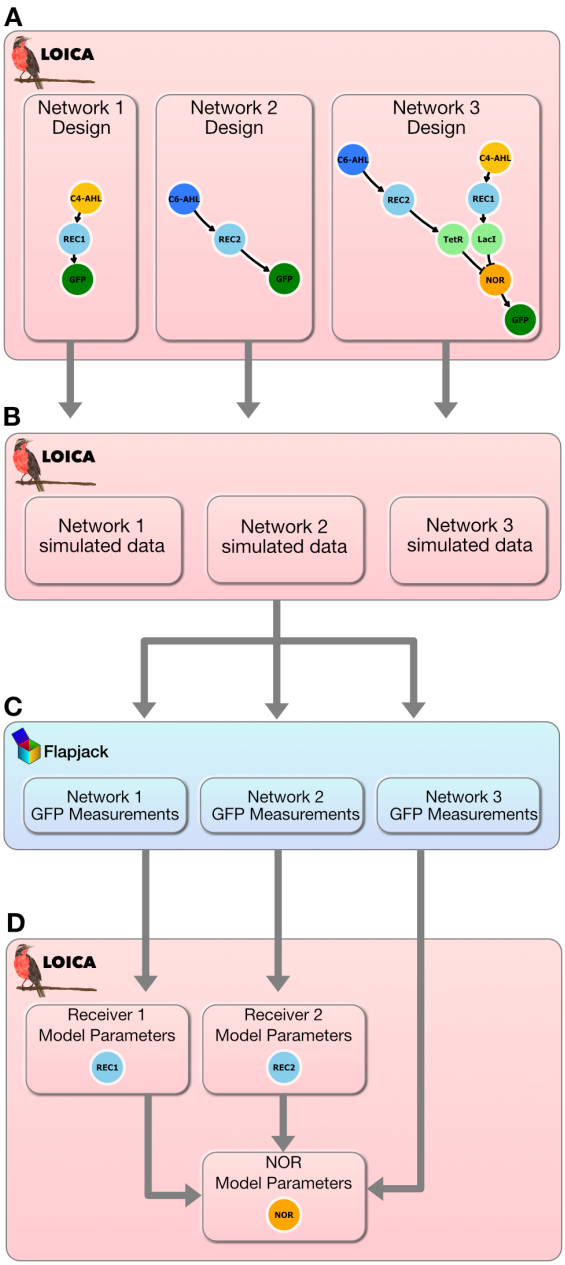


Fig. 3.20 Characterization from simulated data workflow. (A) Accessory genetic network designs in LOICA. (B) LOICA simulates experimental data. (C) Simulated data stored in Flapjack. (D) `Operators` model characterization.

$$B_t = (B(t) + B')(1 + \varepsilon_t) \tag{3.9}$$

$$y_t = (y(t) + y')(1 + \zeta_t), \tag{3.10}$$

where ε_t and ζ_t are uncorrelated white noise with variance σ^2, due to the measurement process. Simulated measurements were generated using LOICA [178] then uploaded to Flapjack [196], and analyzed using the API via Python.

3.4 Discussion and conclusions

In this chapter I developed a software tool to design synthetic genetic networks based on abstraction from Chapter 2. Standardised the output of the software tool from using SBOL3. Connected the software tool from to experimental data for characterization.

LOICA is a programmatic genetic network design tool, implemented as a Python package that generates network representations, simulations using ODE, ODE with noise and SSA, and can characterise genetic components from experimental data. LOICA makes accessible the genetic design to students and researchers with low to intermediate Python knowledge. This language is used by most biologists and engineers, for data analysis or software development, tackling the two most common backgrounds in synthetic biology and more. There is a good set of tools in Python on the SBOL ecosystem, like pySBOL3, sbol-utilities and TYTO and Flapjack has a Python package to access its API. This facilitates the development of new features or the maintenance of the software tools. Python is one of the most popular languages and has some of the most popular AI packages such as tensorflow and pytorch. Although LOICA alone is not a GDA tool it becomes one when used inside the Python environment in combination with other tools as demonstrated in 6.2.1. It's programmatic nature make it compatible with HPC, and has been used in this setup allowing for the simulation of many designs in parallel [145], in this case used to simulate parallel SSA for Monte Carlo simulations on more complex genetic networks. The designs includes genetic networks to perform phase-based arithmetic such as a majority gate, a full adder and a 4-bit phase-based genetic ripple adder [145].

LOICA is already impacting synthetic biology, presented in several conferences as poster and workshop. LOICA has been used for teaching synthetic biology modules at Pontifical Catholic University of Chile and Newcastle University. LOICA is already being extended by the community. A master student implemented diffusible signals and a new release is pending. Examples of the use of LOICA for genetic network design automation are available in Synthetic Biology Methods and Protocols [179].

As future work, I would like to connect LOICA to CellModeller to provide an easy way to access IBM for genetic network designs. With these capabilities, spatio-temporal patterns and emergent properties could be better studied. Also, I want to make more accurate predictions of expression profiles from sequence by studying effects of CDS swapping on

gene expression using techniques from AI. This would improve the assumption that gene expression remains the same after a reporter is swapped with our protein of interest. This can be analysed using LOICA characterization on a dataset of experimental data of swapped CDSs and LOICA could even be used for data augmentation. Furthermore, develop a GUI for LOICA to allow students and researchers without code experience to use it. I also worked in a Julia implementation of LOICA, but due to lack of time I had to put it in the back burner. The development of a connection between LOICA implementations in Python and Julia would cover one of the major problems in Python which is speed and would allow the use of stochastic rate parameter inference using the cross-entropy method (SPICE) for stochastic model parameterization [164]. The export of designs in SBOL does not just connect LOICA to the synthetic biology community but as SBOL can be converted into the Systems Biology Markup Language (SBML) [122] it connects LOICA to the systems biology community widening its impact.

Chapter 4

Standardizing and automating build planning

4.1 Introduction

When discussing build planning, synthetic biologists usually refer to different DNA sequences when referring to a part, it could be for example, a design for a part, that same design with fusion sites or that same design in a plasmid. A standard that serves as common language would help researchers to communicate effectively their assembly strategy or build plan and would help software tools to process these build plans in a reproducible manner. The build plan standardisation is a collaborative work where I participated as a member of the iGEM Engineering Committee. It was proposed as the SBOL Enhancement Proposal (SEP) 055 – Representation of Parts and Devices for Build Planning. This SEP was rejected because it does not modify the SBOL specification but was implemented as a Best Practice (BEP) 011 – Representation of Parts and Devices for Build Planning.

Lab automation tools have the capability to increase scientific throughput by reducing the time that researchers spend in the lab, reducing pipetting errors and its standard deviation, and increasing metadata capture, traceability and reproducibility. One of the first barriers for lab automation is the cost of liquid handling robots. This has been addressed by companies like Opentrons that have substantially reduced the cost of liquid handling robotics by making them open source. Although a lot of liquid handling robotics have been implemented in research and industrial laboratories there is still a challenge in the training of new users and the creation of new protocols. PUDU is a Python package for liquid handling robot control in synthetic biology workflows. It is composed of a set of classes that represent different common protocols from cloning such as DNA assembly, transformation, test plate setup and even calibration. These protocols are easy to modify and to adapt to different laboratory needs, reducing the barrier for new students and researchers to use the OT-2. Furthermore, PUDU connects to standards by accepting SBOL build designs as input and generating SBOL files of the protocol products as output.

4.2 Results

4.2.1 Encoding building plans in SBOL3

Note: *The work presented in this section was carried out in collaboration with Vinoo Salvaraja, Gael Chambonnier, Traci Haddock-Angelli, Alejandro Vignoni, Nicholas Roehner and Jacob Beal from iGEM and the iGEM Engineering Committee.*

The development of the terminology for build planning considered common digestion/ligation build workflows, for the "classic" restriction enzyme cloning and the Golden Gate assembly [52]. BioBricks [158] were considered as assembly standard for restriction

enzyme cloning, and MoClo [190], GoldenBraid [152] and PhytoBricks [26] were considered assembly standards for Golden Gate. Terms were developed with the aim of representing them as closely as possible to pre-existing jargon and patterns in descriptions and discussions of "parts" , "backbones" and "assembly" amongst practitioners, with adjustments to eliminate ambiguity. This terminology was circulated to collect feedback from other synthetic biology practitioners, privately in a first instance and then publicly and adjusted iteratively to address issues raised in comments.

The resulting terminology focused on the three main concepts of *part* (Table 4.1), *backbone* (Table 4.2), and *assembly* (Table 4.3). This terminology is compatible with functional synthetic biology [3] (Chapter 2) representing the build part of it. To make this terminology more concrete two common workflows that were used in this work are described.

In the domestication process a part recently designed or modified is inserted into a backbone compatible with an assembly method. In this example the design starts with a *Part core* that represents a new GFP CDS without stop codon, 4 bp where added as prefix AATG to join a CDS and as sufix AGCC to join a degradation tag to create a `Part insert` because now has a defined interface in an assembly. Along these sequences SapI *Restriction enzyme* recognition sites to create a `Part in backbone` which can be synthesised as double stranded linear DNA. These linear `Parts in backbone` can be used with the universal acceptor in an assembly. During the assembly the double stranded DNA is digested creating a `Part extract` and the universal acceptor is digested creating an `Open backbone`. The `Part extract` and `Open backbone` fusion sites or interfaces align by base complementarity and are ligated producing a `Part in backbone` that now is in a circular backbone and that can be used as level 0 part in Phytobrics and MoClo.

4.2.2 Implementing building plans using Python

Note: *This work was conducted as part of a SBOL Industrial internship supported by the BioDesign Automation Consortium (BDAC), and supervised by Jacob Beal.*

The concepts captured by the terminology in section 4.2.1, were explicitly represented in SBOL3 [112] using pySBOL3, sbol-utilities and pyDNA. This enables the development and use of software tools for working with parts and assembly plans *in-silico*. The BP011(see Figure 4.3) was implemented in sbol-utilities, components modules, as a set of functions and objects for build planning.

The `ed restriction enzyme` uses a name to build an `ExternallyDefined` with type `SBO PROTEIN`. The name must follow the standard restriction enzyme nomenclature, i.e. 'BsaI' to be used to construct a rebase URI that links to the canonical definition external to SBOL.

Terminology	Definition
Part	Design for a single contiguous linear DNA construct with a completely specified sequence.
Part Core	Any part that is not designed with reference to an assembly. In many cases a part core may also be a device with a function that can be defined simply (e.g., promoter, CDS, terminator), but part cores can potentially also be more complex devices, such as a whole functional unit or even an entire gene cluster (this is why "core" is used rather than "basic"). The distinction is in whether an assembly is referenced in the design (i.e., a composite part can be transformed into a part core by stripping associated assembly information).
Composite Part	A part that is designed as the composition of two or more other parts through an assembly. Note that samples of a composite part need not actually be produced by its designated assembly process: for example, the part might be implemented directly via synthesis, including the scars that would have been formed as part of assembly.
Assembled Part	A part, plus any 5' or 3' flanking scars, within the post-assembly context of a composite part.
Scar	A sequence that is produced by the combination of flanking sequences in an assembly.

Table 4.1 Part related terminology.

The `backbone` uses an identity, sequence, dropout location, fusion site length and a boolean to inform if it is linear or not to build a backbone `Component` and `Sequence`. To do this, first it builds a `dna component with sequence` using the identity and sequence provided and appends the role `SO DOUBLE STRANDED` to represent the backbone. Then, it creates two `SequenceFeatures`, one representing the dropout sequence with a role `SO.deletion` and another representing the insertion sites with a role `SO insertion site`. The last `SequenceFeature` to add is the open backbone and it is calculated differently depending if the backbone is linear or circular. If the backbone is linear the sequence from the beginning up to the start of the dropout sequence and after the dropout sequence up to the end are used to create the open backbone `SequenceFeature` these 2 segments have order 1 and 3 respectively to convey that there is something in between them. If the backbone is circular the sequence from the beginning up to the start of the dropout sequence and after the dropout sequence up to the end are used to create the open backbone `SequenceFeature` these 2 segments have order 2 and 1 respectively to convey that there is a continuity between them. Finally, the three `SequenceFeatures` are added as features to the backbone `Component`

Terminology	Definition
Backbone	A DNA construct into which parts are intended to be inserted at one or more designated insertion sites, in order to meet the requirements of an assembly. Precisely one part can be inserted at any given insertion site. In many cases, a backbone will be a circular plasmid with precisely one insertion site, but other types of vector are possible as well, such as linear plasmids, viral replicons, or non-replicating flanking adapters.
Drop-Out Sequence	A portion of a backbone at an insertion site that is removed when a part is inserted at that site. Some backbones include drop-out parts while others do not.
Open Backbone	A portion of a backbone that after a digestion processes is the complement of drop-out sequence. In most cases the open backbone will be a transient state during assembly which is the result of digesting a backbone. Open backbones can be ligated with extracted parts and is commonly the longest of the digested products carrying features such as antibiotic resistance and origin of replication.
Open Part in Backbone	A portion of a backbone that after a digestion processes is the complement of drop-out sequence in backbones that contains multiple insertions sites and others has been occupied by a part insert already.
Part Insert	A part, plus any 5' and 3' flanking sequences, that is placed into a designated insertion site of a backbone.
Part in Backbone	A backbone with at least one insertion site occupied by a part insert.
Part Extract	A part, plus any 5' or 3' flanking sequences, that has been extracted from a part in backbone as part of an assembly process. Note that the same extract can be produced from a backbone with flanking sequences and an insert without or a backbone without flanking sequences and an insert that includes them.

Table 4.2 Backbone related terminology

created at the beginning and a meet `Constraint` using the dropout sequence as subject and the open backbone as object is appended.

The `backbone from sbol` uses an identity, SBOL `Component`, dropout location, fusion site length and a boolean to inform if is linear or not to build a backbone `Component` and `Sequence`.

The `part in backbone` uses an identity, part core SBOL `Component`, backbone SBOL `Component` and a boolean to inform if is linear or not to build a part in backbone `Component`

Terminology	Definition
Assembly	A plan for combining a set of input parts in order to produce an output of either a single composite part or a library of composite parts. The inputs and output may or may not include backbones, depending on the specifics of the assembly. An assembly plan can be executed by appropriate laboratory protocols.

Table 4.3 Assembly terminology.

and `Sequence`. To do this, first it computes the two open backbone sequences before and after the dropout and extracts the part core sequence. If it is linear, the part in backbone sequence is compiled by adding the open backbone before the dropout sequence, the part core sequence and the open backbone after the dropout sequence. Also the topology type is set to `SO LINEAR`. If it is circular, the part in backbone sequence is compiled by adding the part core sequence, the open backbone after the dropout sequence and the open backbone before the dropout sequence. Also the topology type is set to `SO CIRCULAR`. Then, using the part in backbone sequence it builds a `dna component with sequence` with role `SO plasmid vector` and the topology type to represent the part in backbone. Finally, it creates a part core `SubComponent` with role `SO.engineered insert` and a backbone `SubComponent` and add them as features to the part in backbone `Component`.

The `part in backbone from sbol` uses an identity, SBOL `Component`, part location, part features, fusion site length and a boolean to inform if is linear or not to build a part in backbone `Component` and `Sequence`. To do this, first depending if the identity is a string or None it builds a new `dna component with sequence` using the using the identity and the sequence from the input `Component` or writes in top of the input `Component` maintaining its identity. Then, it appends the role `SO DOUBLE STRANDED` and creates two `SequenceFeatures` one for the part with role `SO.engineered insert` and other for the insertion sites with role `SO.insertion site`. If the part in backbone is linear the sequence from the beginning up to the start of the dropout sequence and after the dropout sequence up to the end are used to create the open backbone `SequenceFeature` these 2 segments have order 1 and 3 respectively to convey that there is something in between them. Also the topology type `SO LINEAR` and the role `SO ENGINEERED REGION` are appended. If the backbone is circular, the sequence from the beginning up to the start of the dropout sequence and after the dropout sequence up to the end are used to create the open backbone `SequenceFeature`. These 2 segments have order 2 and 1 respectively to convey that there is a continuity between them. Also the topology type `SO CIRCULAR` and the role `SO plasmid vector` are appended. Finally, the three `SequenceFeatures` are added as features to the backbone `Component`

created at the beginning and a meet `Constraint` using the dropout sequence as subject and the open backbone as object is appended.

The `digestion` uses a reactant, a restriction enzyme and an assembly plan to create `Extracted parts` and `Open backbones` and a cleavage interaction. Reactants can be either a `Part in backbone` or a `Backbone`, enzyme can be an externally defined enzyme and the assembly plan is the component that stores the products and interactions.

The `ligation` uses a list of reactants and an assembly plan to create a `Composite part` and a ligation interaction. Reactants can be `Part extracts` and `Open backbones` and the assembly plan is the component that stores the products and interactions.

The `Assembly plan composite in backbone single enzyme` is a class to orchestrate an assembly plan that produces a composite part in backbone using a single enzyme. The attribute part in backbone stores all parts in backbones to be used during the assembly. The attribute acceptor backbone stores all the backbones to be used during the assembly. The attribute restriction enzyme stores the restriction enzyme to be used during the assembly. This class is constrained to one restriction enzyme per reaction. The document stores the SBOL document that will contain all the components and interactions produced during the assembly. These classes can be used to format SBOL files created with LOICA, see Chapter 3

4.2.3 Creating build metadata in SBOL3

Recurrent reagents were recognized by iterating cloning and test setup. These recurrent reagents were abstracted and organised as a metadata model (Figure 4.1). The metadata model includes: `DNA`, `Strain`, `Chemical`, `Media`, `GMO`, `Supplement`, `Sample` and `Assay`. `DNA` represents any DNA with a sequence that is a plasmid like circular `Backbones` and `Part in backbones` from section 4.2.1. `Strain` represents any strain with its genome in a public repository like KEGG and may have a model. `Chemical` represents any chemical with ChEBI or PubChem ID. `Media` represents any media with a name and recipe. `GMO` represents any organism or `Strain` transformed with `DNA`. `Supplement` represents any `Chemical` that is used to supplement, or that is added to a `Sample`. `Sample` represents one well in a 96 well plate with a mix of `Media`, `GMO` and `Supplement`. `Assay` represents a loaded 96 well plate.

All these abstracted parts from the metadata model interact in the physical world through implementations. Implementations can be more specific and store specific information to track location, provider and reference information. A `Chemical` implementation can be an IPTG (CHEBI:61448) inducer solution in a -20 freezer. A `Media` implementation can be a Duran Schott bottle with LB media on shelf 1. A `Strain` implementation can be a DH5α in a cryogenic tube in a cryogenic box in a -80 freezer. A `GMO` implementation can be a DH5α, transformed with a plasmid, in a cryogenic tube in a cryogenic box in a -80 freezer. A `96`

well plate implementation can be a loaded 96 well plate ready to be measured. DNA can be in different forms during the Build stage. Usually researchers get DNA from bacteria stabs transformed with a plasmid (an Addgene order for example), having to perform culturing and extraction to get Extracted DNA, this process is the same for GMO created in situ at the laboratory. Another form in which DNA is obtained is as a synthesis product with an informed concentration, mass or moles (an IDT order for example). This synthesis product can be represented with Diluted DNA, which also can be used to represent dilutions made from Extracted DNA produced in the laboratory. Finally, DNA can be assembled at the lab and represented with Assembled DNA. DNA in any physical implementation can be used to produce new GMOs (Figure 4.1)

The metadata backbone proposed in this work can be customised to other laboratory needs. For this I created a set of functions to create components to track metadata that can be used along with the workflow.

- dna component with sequence uses an identity and a sequence to build a Sequence with encoding IUPAC DNA ENCODING and a Component of type SBO DNA.
- rna component with sequence uses an identity and a sequence to build a Sequence with encoding IUPAC RNA ENCODING and a Component of type SBO RNA.
- protein component with sequence uses an identity and a sequence to build a Sequence with encoding IUPAC PROTEIN ENCODING and a Component of type SBO PROTEIN.
- functional component uses an identity to build a Component of type SBO FUNCTIONAL ENTITY.
- promoter uses an identity and sequence to build a dna component with sequence and append the role SO PROMOTER.
- rbs uses an identity and sequence to build a dna component with sequence and append the role SO RBS.
- cds uses an identity and sequence to build a dna component with sequence and append the role SO CDS.
- terminator uses an identity and sequence to build a dna component with sequence and append the role SO TERMINATOR.
- protein stability element uses an identity and sequence to build a dna component with sequence and append the role SO protein stability element.

- `gene` uses an identity and sequence to build a `dna component with sequence` and append the role `SO GENE`.
- `operator` uses an identity and sequence to build a `dna component with sequence` and append the role `SO OPERATOR`.
- `engineered region` uses an identity and features as `Component` or `SubComponent` to build a `Component` of type `SBO DNA` and append the role `SO ENGINEERED REGION` and the features. `SubComponents` are added directly and `Components` are converted to `SubComponents` and then added. Features are added in the order of the input list and is fixed using `Constraints`.
- `mrna` uses an identity and sequence to build a `rna component with sequence` and append the role `SO MRNA`.
- `transcription factor` uses an identity and sequence to build a `protein component with sequence` and append the role `SO TRANSCRIPTION FACTOR`.
- `media` uses an identity and a recipe to build a `functional component` and append the role `NCIT Media`. The recipe is a dictionary having as key a `Component` of the ingredient and a `Measure` of the added amount.
- `strain` uses an identity and a recipe to build a `functional component` and append the role `NCIT Strain`.
- `ed simple chemical` uses a definition, a URI that links to a canonical definition external to SBOL (recommended repositories are ChEBI and PubChem) to build an `ExternallyDefined` with type `SBO SIMPLE CHEMICAL`.
- `ed protein` uses a definition, a URI that links to a canonical definition external to SBOL (the recommended repository is UniProt) to build an `ExternallyDefined` with type `SBO PROTEIN`.

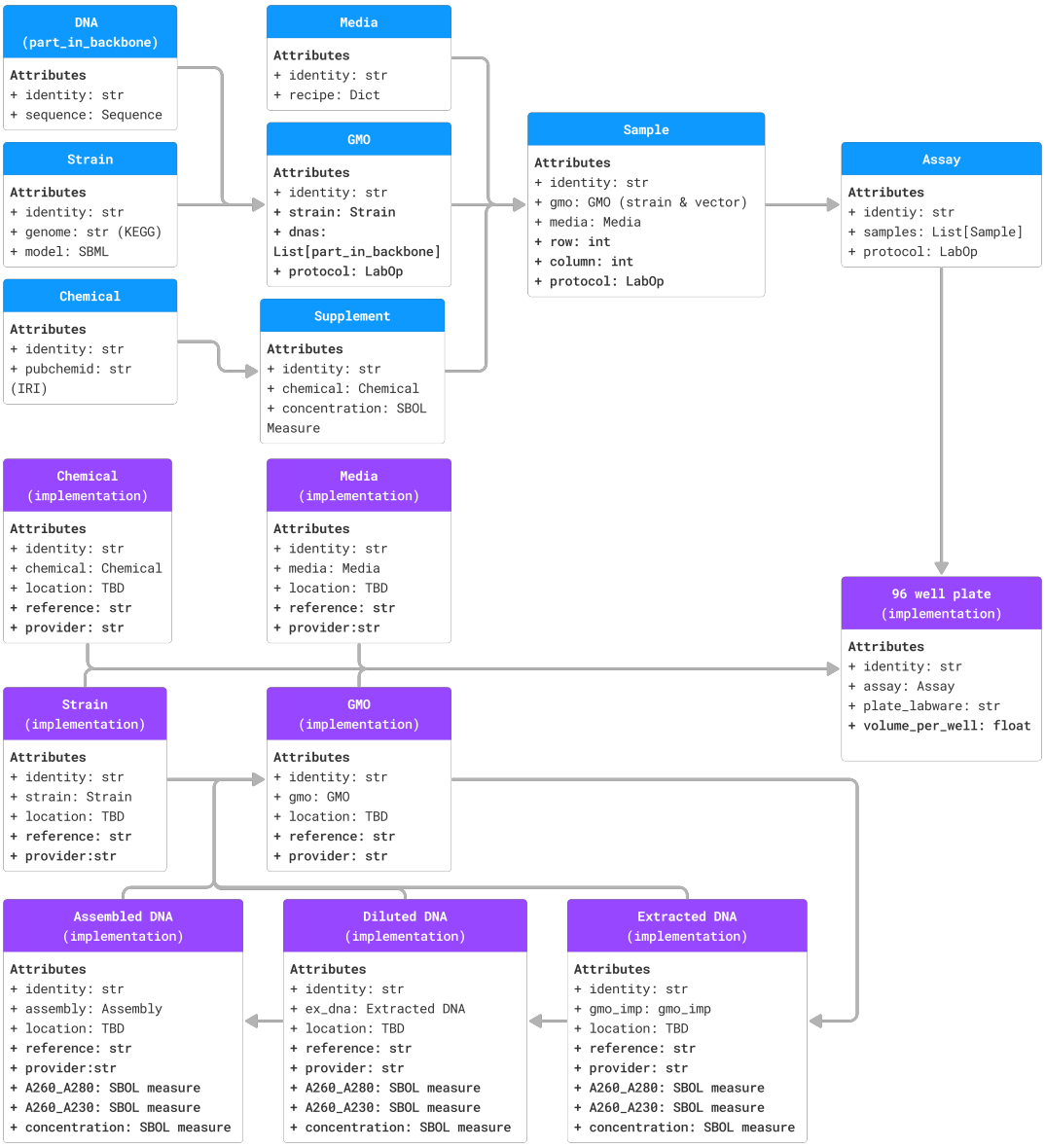


Fig. 4.1 Metadata representation flow diagram. SBOL Components (Blue) and representation of their physical implementation (Purple) used to represent the metadata produced at Build stage. The abstract elements `DNA` and `Strain` are encapsulated by `GMO`, `Chemical` is encapsulated by `Supplement`. `Media`, `GMO` and `Supplement` are encapsulated by `Sample` which in turn is encapsulated into `Assay`. The metadata from the physical implementations of these elements to prepare a test, such as sample design which includes `Chemical`, `Strain`, `Media` and `GMO`. In this context DNA implementations can be found in three forms, `Extracted DNA` represents DNA that has been extracted from a `GMO` through miniprep for example. `Diluted DNA` represents a DNA with a known concentration that can be product of diluting an `Extracted DNA` or synthesis product, it can be in fmol/μL and used for assemblies. `Assembled DNA` represents assembly products. All three can be used with a `Strain` to create new `GMO` and optionally have quality and sequencing information.

4.2.4 Automating build using Opentrons OT-2 liquid handling robot

Note: *The work presented in this section was carried out in collaboration with David Markham (DM) and Matt Burridge (MB) from ICOS. All experiments and coding presented below were performed by GV, except where explicitly noted as the work of a collaborator.*

Protocol Unified Design Unit (PUDU) provides a high-level abstraction for liquid handling robot control using a simple object-oriented programming approach in Python. Each object corresponds to a protocol template that can be easily customised changing its attributes. This allows the user to change the samples, volumes, pipette position, labware and more, but the protocol remains the same. To create a new class, or protocol template, users can inherit from existing ones and build on top of them making this process more straightforward, easy to update and reducing errors. For example, let's define a protocol as the series of steps that the human and the machine have to follow to achieve a goal. This goal can be to obtain a sequence of DNA, transform a cell or calibrate a plate reader. Then inside a protocol there are steps that a human has to perform and steps that the machine has to perform. The steps that the human has to perform are encoded in text. The steps that the machine has to perform are encoded in the script. To define a script, or OT-2 protocol, you need to create a run function that encodes the labware type and position, as well as the liquid transfers. PUDU simplifies this process of making and using scripts to just two lines of code. Inside the script's run function, the user needs to instantiate a PUDU class and then call its own run method.

For a typical PUDU protocol, the user creates a script and simulates it. The simulation will output a dictionary in the command line with information about reagent position. Then, the user can load the script into the Opentrons App and get information to set up the OT-2 deck and run the script. These scripts automate different stages of the cloning process, the test setup and calibration (Figure 4.2). All the code, protocols, examples and documentation are publicly available at https://github.com/RudgeLab/PUDU

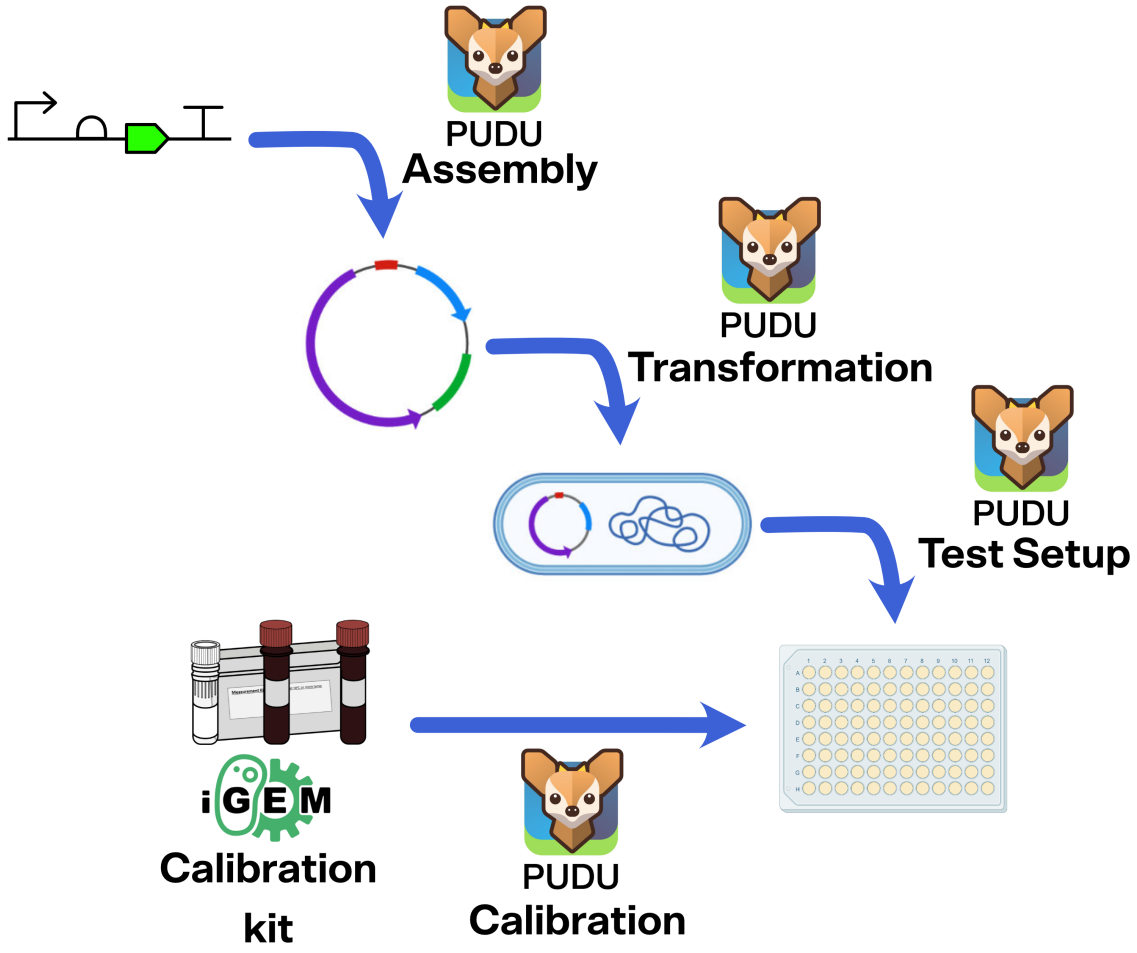


Fig. 4.2 PUDU workflow diagram. Synthetic biology workflow automated using PUDU. The DNA Assembly class automates domestication and the assembly using a design in SBOL or a list of parts using Loop. This process can be automated using a a DNA Assembly script. Then, the Transformation class automates the transformation of bacteria with the assembled DNA. Finally, the Test Setup class automates the setup of a 96 well plate with the transformed bacteria under different conditions. Furthermore, The Calibration class automates the preparation of a 96 well plate to obtain calibrated data.

DNA Assembly

The `DNA Assembly` class has three child classes, `SBOL DNA Assembly`, `Domestication` and `Loop DNA Assembly`. The `DNA Assembly` class is intended to be a template for default values and structure of other assembly classes, therefore it does not have a run method. The `SBOL DNA Assembly` class takes an assembly plan as input in SBOL format. This assembly plan must follow the representation of parts and devices for build planning best practice (https://github.com/SynBioDex/SBOL-examples/tree/main/SBOL/best-practices/BP011). This input provides the parts to mix and the restriction enzyme(s) to use. LOICA's SBOL output can be formatted with build plan best practices to connect Design and Build stages, see Chapter 3.

The `Domestication` class takes two inputs as either strings or SBOL Components. The first input is a list or dictionary of parts and the second is the acceptor backbone. It is designed to insert DNA parts from linear fragments (e.g. gBlocks) into plasmids (e.g. universal acceptor backbone pSB1C00) assuming the use of SapI.

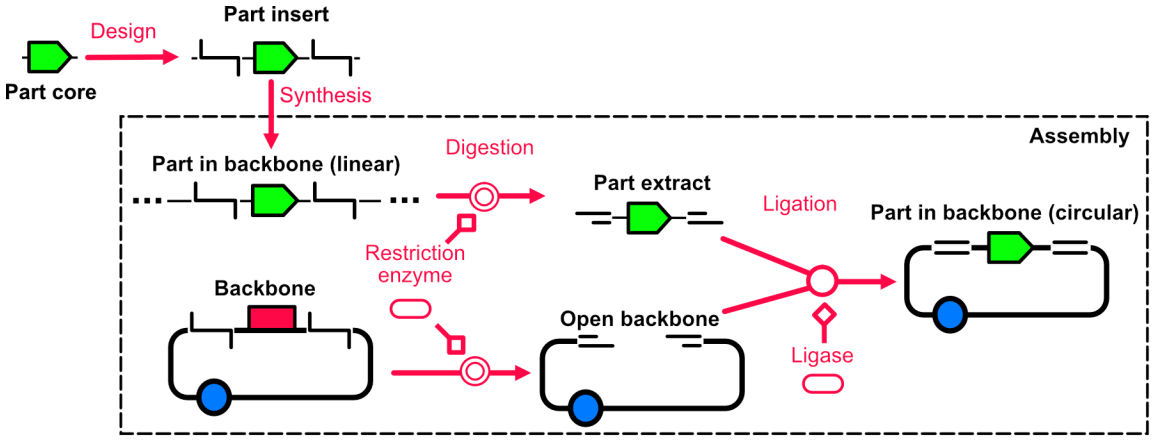


Fig. 4.3 Standard build plan representation of a Loop domestication. The design process starts with a new part core, in this example a new green CDS. Then its position on the assembly is decided and flanking regions are added to the design creating a part insert. The part insert then can be synthesised as a part in the linear backbone with the restriction enzyme recognition sites for part extraction. The backbone represents the universal acceptor. The part in the linear backbone and the backbone are digested using a restriction enzyme producing a part extract and an open backbone. These two digestion products then are ligated using a ligase to create a circular part in backbone that represents a level 0 part for Loop assembly.

The `Loop DNA Assembly` class takes a list of dictionaries as an input. Each dictionary describes the assembly of a combinatorial derivation of parts, for example: {"promoter" : ["j23101", "j23100"], "rbs" : "B0034", "cds" : "GFP", "terminator" : "B0015", "receiver" : "Odd 1"}. In this example the code takes the Cartesian product of values per each key,

building two transcriptional units, where the only difference between them is that one has J23101 and the other has J23100. Dictionary keys or roles can be defined by the user apart from the receiver, which always needs to be included. The receivers must start with Odd or Even to define the restriction enzyme to use in each assembly. All the `DNA Assembly` protocols can be performed with a p20 (Figure 4.5). The protocols were dry tested, using just tips and wet tested using food dyes.

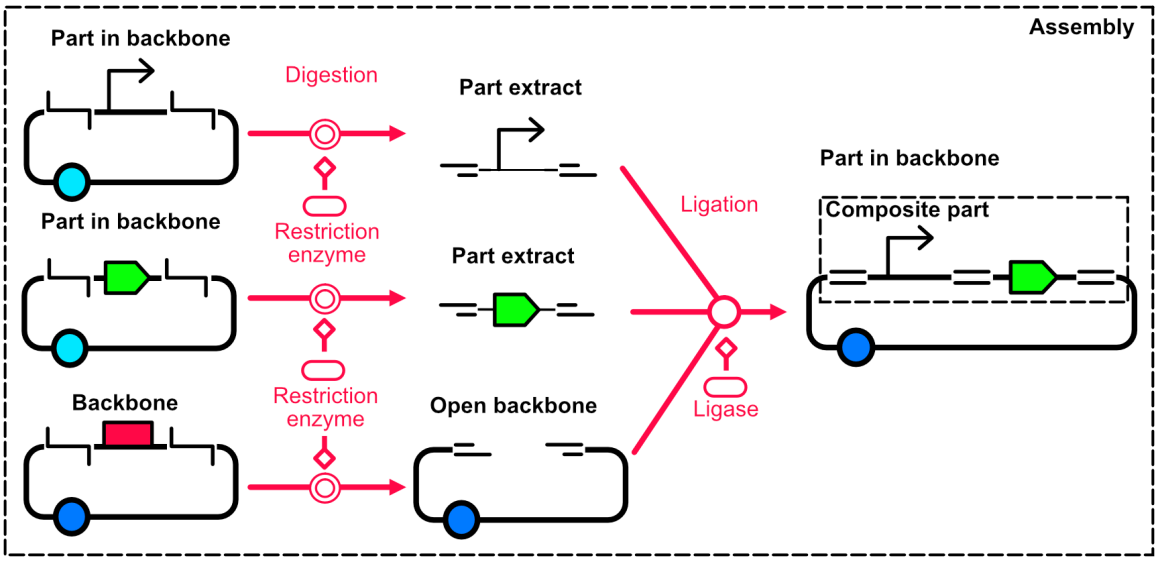


Fig. 4.4 Standard build plan representation of a level 1 Loop assembly. On the top left two parts in backbone are represented, one containing a promoter and other containing a green CDS. On the bottom left a backbone is represented containing a red CDS. These three plasmids are digested using a restriction enzyme, producing part extracts from the parts in backbone and an open backbone from the backbone. The three digestion products are then ligated using a ligase, producing a part in backbone where the part is a composite part of the two parts extract.

The `DNA Assembly` protocols use a tip rack, temperature module and thermocycler module. The tip rack is for a p20 pipette for liquid manipulation. Microcentrifuge tubes (1.5 mL or 2.0 mL) placed on the temperature module contain diluted plasmids, restriction enzyme, ligase and ligase buffer that are mixed in a set of wells on the thermocycler plate.

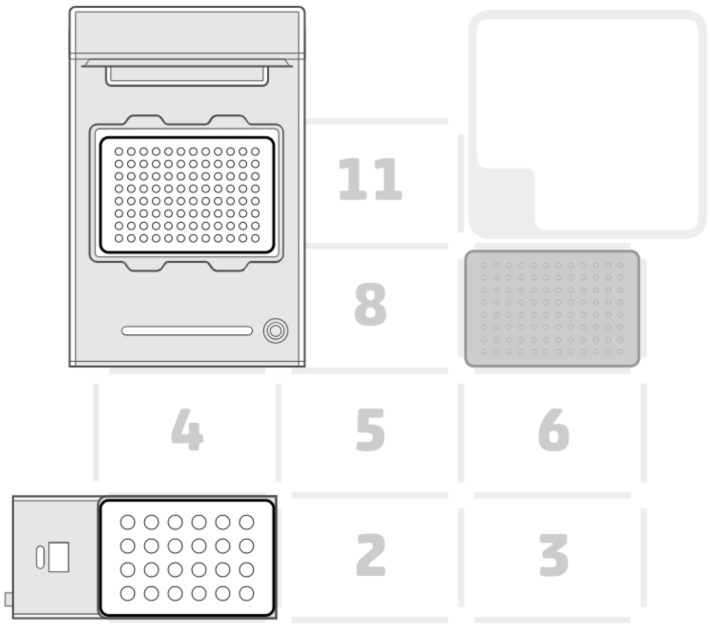


Fig. 4.5 OT-2 deck setup for running `DNA Assembly`. The temperature module with a 24 well aluminum block was placed in the slot 1. The p20 tip rack was placed in the slot 9. The thermocycler module was placed in the slots 7,8,10 and 11. The place of the reagents in the aluminum block and of products in the thermocycler is generated during the simulation and informed as a dictionary in the terminal.

Transformation

The `Transformation` class has one child class, `Chemical Transformation`. The `Chemical Transformation` class takes DNA and strain as needed inputs, but has several default values that can be changed if needed. This protocol starts by calculating the number of transformations to be done, multiplying the length of the provided list of DNAs by the number of replicates. The code assumes that you have enough DNA in each tube and uses a `volume dna` of 2 μL per transformation. Then the number of transformations per tube is calculated dividing the `volume competent cell per tube` by the `volume competent cell to add` to inform the user how many tubes the protocol will need. By default the `volume competent cell per tube` is considered to be 100 μ L and the `volume competent cell to add`

per transformation is 20 μ L. The number of transformations per media tube are also calculated dividing the `volume recovery media per tube` by `volume recovery media` to use per transformation. All these reagents are assumed to be in a 1.5 mL tube (Figure 4.6).

The DNA and competent cells are mixed in a 200 μ L well on the PCR well plate placed in the thermocycler module. Then a cold incubation profile is executed in the thermocycler, by default it starts with a cold incubation at 4° C for 30 minutes, followed by a heat shock at 42° C for 1 minute, and a second cold incubation at 4° C for 2 minutes. The thermocycler opens its lid and dispenses recovery media into the wells where transformation is being executed. The thermocycler closes its lid and starts executing a recovery incubation profile with incubation at 37° C for 60 minutes. At this point the researcher has to go to the OT-2 liquid handling robot to plate them. This class was implemented with the help of MB. The protocols were dry tested, using just tips and wet tested with food dyes.

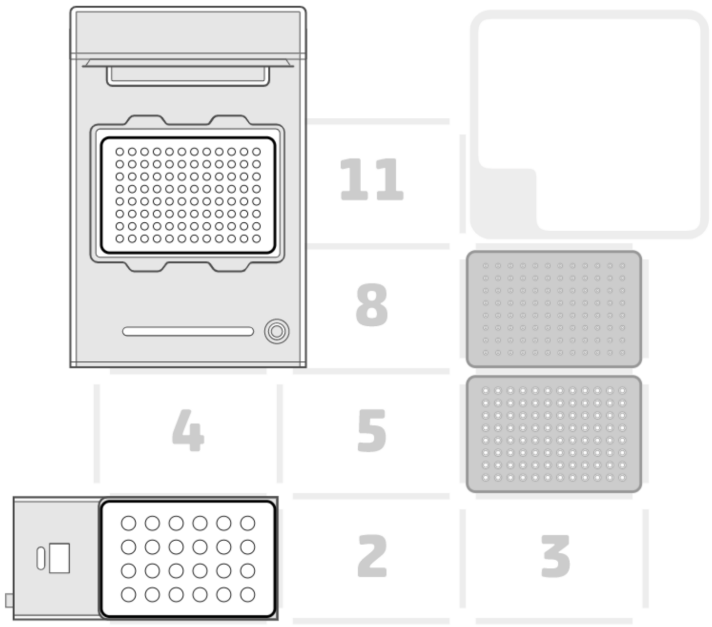


Fig. 4.6 OT-2 deck setup for running `Chemical Transformation`. The temperature module with a 24 well aluminum block was placed in the slot 1.The p200 tip rack was placed in the slot 6. The p20 tip rack was placed in the slot 9. The thermocycler module was placed in the slots 7,8,10 and 11. The place of the reagents in the aluminum block and of products in the thermocycler is generated during the simulation and informed as a dictionary in the terminal.

Test Setup

The `Test Setup` class has two child classes, `Plate Samples` and `Plate Supplement Setup`. The `Plate Samples` class takes the samples and dispenses them in a 96 well plate. The samples are 1.5 centrifuge tubes on the aluminium block with or without temperature module. By default it aspirates 200 μL from the sample and dispenses it in a well, this process is repeated 4 times creating 4 biological replicates of each well. Important samples to have are the Media control and the Strain control to inform the Learn stage for background correction and analysis (Figure 4.7).

The `Plate Supplement Setup` class takes the sample, inducer, initial mix inducer volume, initial mix sample volume, serial dilution volume and serial dilution steps. The samples are 1.5 mL centrifuge tubes, filled with 1.5 mL of liquid, on the aluminium block with or without temperature module. The protocol uses one tube of inducer and a variable amount of sample tubes. Inducer and sample information are used to store metadata and to inform the user during OT-2 desk setup. Starting from the second column, a number of wells with 200 uL of sample equal to serial dilution steps are prepared. It uses one row by serial dilution and starts at the first position with the initial mix, which is a combination of sample and inducer that will be used to start a serial dilution. The serial dilution proceeds by aspirating the serial dilution volume from the initial mix and dispensing it in the next well, from which it will take the same volume and dispense it in the next one maintaining its volume. This process repeats until the last well where the aspirated volume is dispensed in the initial mix well, which is used as a bin. Some optional attributes help to use it multiple times in the same plate are starting row and replicates. The protocols were dry tested, using just tips and wet tested with food dyes.

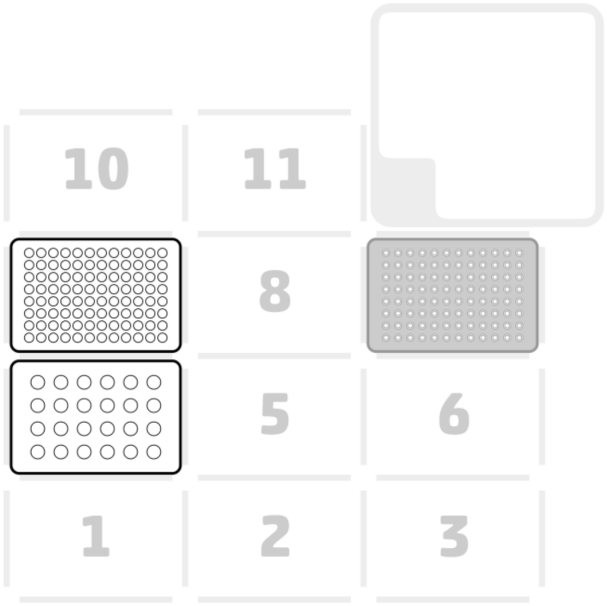


Fig. 4.7 OT-2 deck setup for running `Plate Samples`. The 24 well aluminum block was placed in the slot 4. The 96 well plate for test was placed in slot 7. The p200 tip rack was places in slot 9. The place of the reagents in the aluminum block and of samples in the 96 well plate is generated during the simulation and informed as a dictionary in the terminal.

Calibration

The `Calibration` class has two child classes, `iGEM GFP OD` and `iGEM RGB`. `iGEM GFP OD` is used to prepare a plate for calibration of green fluorescence [10] and the cell count from OD [9]. `iGEM RGB OD` is used to prepare a plate for calibration of red, green and blue fluorescence [11] and cell count from OD [9] (Figure 4.8). The calibration protocol was tested by GV and DM. The data obtained was processed using the iGEM calibration template from 2019 interlab obtaining a mean of $2,27 * 10^8$ particles per OD 600 absorbance and a mean of $2,46 * 10^9$ molecules of fluorescein per AU. This factors were used to calibrate measurements during this work.

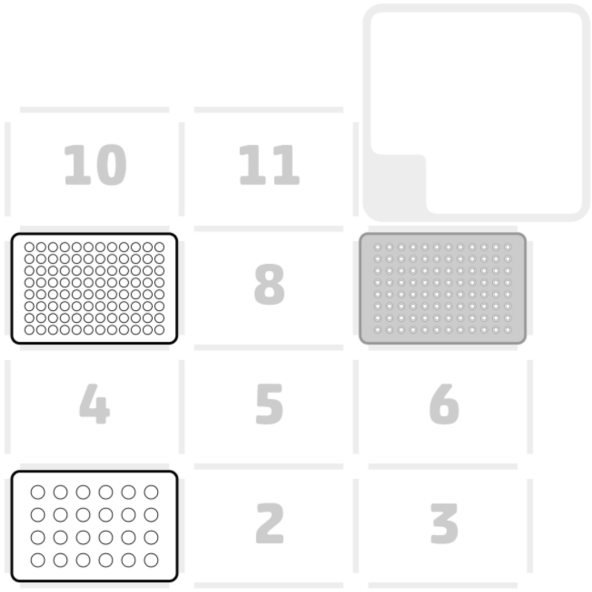


Fig. 4.8 OT-2 deck setup for running `iGEM RGB OD`. The 24 well aluminum block was placed in the slot 1. The 96 well plate for test was placed in slot 7. The p200 tip rack was places in slot 9. The place of reagents in the aluminum block is informed in a protocol.

4.3 Methods

4.3.1 Software development

PUDU were developed in Python 3, its dependencies are Opentrons API v2, pySBOL3 [117], xlsxwriter. Software source code is publicly available under MIT licence at https://github.com/RudgeLab/PUDU. The documentation is available at https://loica.readthedocs.io and the package is distributed through PyPI https://pypi.org/project/pudupy. Contributions to sbol-utilities were developed in Python 3. Software source code is publicly available under MIT licence at https://github.com/SynBioDex/SBOL-utilities. The documentation is available at https://sbol-utilities.readthedocs.io and the package is distributed through PyPI https://pypi.org/project/sbol-utilities.

Build plans and simulations were performed on a Apple MacBook Pro 15-inch, 2019, with a 2,6 GHz 6-Core Intel Core i7 processor, Radeon Pro 555X 4 GB and Intel UHD Graphics 630 1536 MB GPUs, and 16 GB 2400 MHz DDR4 RAM.

4.4 Discussion and conclusions

In this chapter I developed a standardised build plan representation and implemented them in SBOL3. I developed a set of functions to create experimental metadata using SBOL3. I developed a software tool that uses standardised build plans to automate build protocols using liquid handling robots.

In the future I would like to improve the connection with Design tools such as LOICA [178] and SynBioSuite [156]. This can be done by creating a function that takes a SBOL `Document` as input and formats it using BP011. I would also like to improve the connection to Learn tools by capturing more and better metadata in an semi-automated way. Although this process is difficult to fully automate as users will always be the ones that know what samples and reagents are provided, I think that creating better interfaces would better facilitate the process of metadata acquisition and standardisation. I would like to add chained actions such as creating an SBOL file, capturing metadata in a machine readable format, and/or adding the creation of a xlsx file, capturing relevant information for deck setup, position of reagents and products in a more amenable human readable format.

I collaborated with the Genetic Logic Lab to develop the experimental data connector (XDC) a software for experimental data and meta data standardization, upload and connection across SynBioHub and Flapjack [149]. This software tools uses an Excel workbook, one of the favorites GUI for wetlab researchers, to capture metadata and experimental data in a standardized format.

I aim to generate a digital plate, ready to be connected to tools like Flapjack [196] and SynBioHub [113]. Compared to other tools such as PyLabRobot [193] or the Opentrons API that help users to create protocols, PUDU has a set of protocols defined as classes where their arguments are required inputs from the user or small modifications, and PUDU captures metadata in a standardised format. For now just the `SBOL DNA Assembly` is the only function that can create metadata when the protocols is simulated or run in the OT-2. The only problem to implement PUDU with SBOL metadata capture capabilities is a dependency issue that does not allow for the installation of pySBOL3 and sbol-utilities in the OT-2 Raspberry Pi. Finally, I want to generalise the liquid handling control generating protocols in the Laboratory Open Protocol (LabOP) standard and expand the calibration scripts using absolute protein quantification [36].

I would like to continue working to extend PUDU capabilities including user friendliness, adding more protocols and making them more intuitive. The libre scripts allows wetlab users to try protocols without any installation. Although this is a good feature for adoption the intended use of PUDU is by installing the package in the controlling computer and the robot so it can use larger libraries such as pySBOL3 [117] to capture the metadata at the moment of the production of new physical entities. The functions to capture metadata from this work can be included in PUDU's workflow. Once the OT-2 gets updated to install pySBOL3 the `SBOL DNA Assembly` and updated functions capturing data with SBOL will be developed.

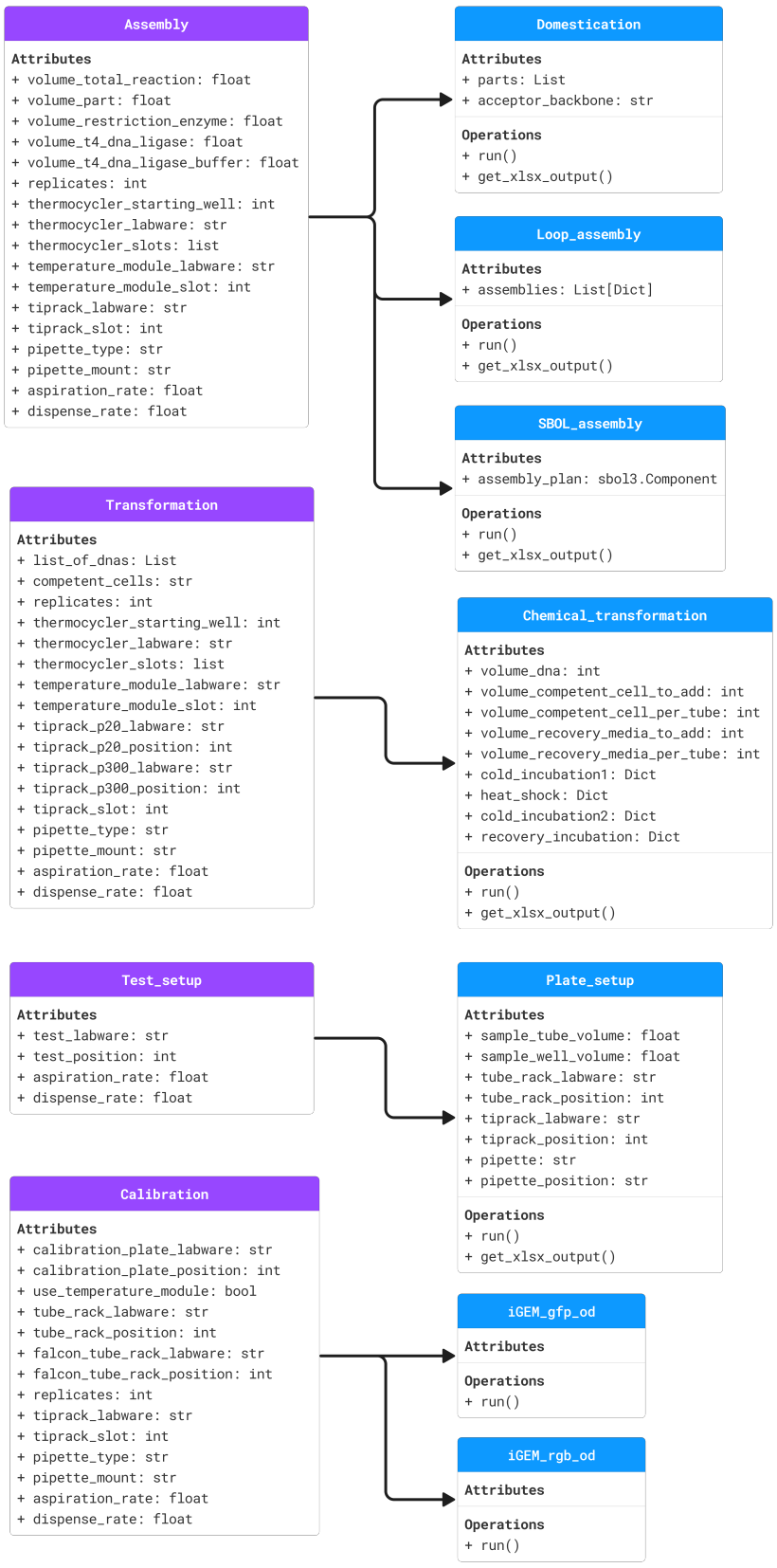


Fig. 4.9 Class Diagram. Purple boxes represent Abstract Class entities and blue boxes represent Class entities. Triangular end arrows represent inheritance relationships.

Chapter 5

Characterization of growth and gene expression rates

5.1 Introduction

Genetic networks are subject to variability due to stochastic events and the exposure to different contexts. Cells face changing internal state, the organism context, and environments, the environmental context, to which they sense and respond by changing their gene expression and growth rates. Furthermore, each gene in a genetic network operates in a composition of genes which may interact with each other and the host cell in complex ways. Short-range and long-range contexts can vary depending on composition of parts or TUs respectively. The context of genetic networks can therefore change gene expression and growth rates, and measuring their dynamics is essential to understanding natural and synthetic regulatory networks that give rise to functional phenotypes and traits. However, reconstruction of microbial gene expression and growth rate profiles from typical noisy measurements of cell populations is difficult due to the effects of noise at low cell densities after background correction among other factors. Here I describe a new mathematical method and a software implementation to estimate dynamic microbial gene expression rates and growth rates from noisy measurement data. Compared to the current state-of-the-art, our method significantly reduced the mean squared error of reconstructions from simulated data of growth and gene expression rates, improving the estimation of timing and magnitude of relevant profile shapes. The method was applied to characterise a triple-reporter plasmid library combining TUs in multiple compositions over different organism and environmental contexts in *E. coli*.

5.2 Results

Note: *The work presented in this section was carried out in collaboration with, Macarena Muñoz Silva (MMS), Carlos Castillo-Passi (CCP) and some of it was published in reference [177]. All simulations and analysis presented below were performed by GV, except where explicitly noted as the work of MMS or CCP.*

5.2.1 Gene Expression and Growth Dynamics

Genetic networks are composed by genes or TUs that interact with each other changing its gene expression rate through genetic products, i.e. RNA, proteins. The rate at which a protein is synthesised by a genetic network or circuit varies over time giving rise to dynamic gene expression rate profiles, which may include rich behaviours such as bistability [58] and oscillations [49]. To see the importance of measuring these gene expression rate profiles, consider a genetic network composed of TUs interacting through regulatory proteins. In the

typical case of short half-life mRNAs, assuming quasi-steady state and,

$$\frac{d\vec{p}}{dt} = \Psi(\vec{r})\eta(t) - \Gamma\vec{p} - \mu(t)\vec{p}, \tag{5.1}$$

for a genetic network with $\vec{p} = (p_0, p_1, ...p_{N-1})^T$ is the vector of `GeneProducts`, which includes different `Regulators` ($\vec{r} = (r_0, r_1, ...r_{M-1})^T$) and `Reporters` ($\vec{s} = (s_0, s_1, ...s_{N-M-1})^T$). The non-linear operator Ψ maps `Regulator` concentrations to `GeneProduct` synthesis rates. η is a function of time that modifies Ψ to create a profile or time-series. Γ is a diagonal matrix of `GeneProduct` degradation rates γ_i, and $\mu(t)$ is the instantaneous growth rate of the cells. Equation 3.1 shows the overall system where Ψ encodes the whole network and consists of a sum of individual LOICA `Operators` Φ_k (Equation 3.2), see Chapter 3. It is therefore essential to analysis of genetic network operation to estimate the gene expression rate profiles ϕ_i and growth rate profile μ, allowing parameterization of models such as those in equation 5.1.

The simplest genetic network consists of a single constitutive TU, in which case $\Phi_i(t)$ is only a function of time. To model the growth and reporter gene expression measurements from such a genetic network in a population of cells, the following equations were used:

$$\frac{\mathrm{d}B}{\mathrm{d}t} = \mu(t)B \tag{5.2}$$

$$\frac{\mathrm{d}y_i}{\mathrm{d}t} = B(t)\phi_i(t) - \gamma_i y_i \tag{5.3}$$

where B is a measure of sample biomass, $\mu(t)$ is the instantaneous relative growth rate, y_i is the intensity of reporter, $\phi_i(t)$ is the instantaneous expression rate, and γ_i is the reporter degradation rate for TU i. Assuming that reporter intensity $y_i = Bp_i$, with p_i the protein concentration per biomass. While this is a reasonable assumption in constant conditions [121], this relationship may not always hold leading to inaccuracy in reconstructed profiles, which is a general problem with gene expression reporters. More complex measurement models might be constructed given sufficient information about the transformation of gene expression level into reporter intensity. This is out of the scope of this work. From these equations the aim is to accurately and robustly estimate the growth rate $\mu(t)$ and the expression rates $\phi_i(t)$, given an estimate of the reporter degradation rates γ_i. Typically, reporter proteins are stable so it's reasonable to assume $\gamma_i = 0$ [4].

Note that the expression rate $\phi(t)$ is different from the rate of change of fluorescence $\mathrm{d}y/\mathrm{d}t$, and from the rate of change of fluorescence concentration $\mathrm{d}(y/B)/\mathrm{d}t$, which both depend on protein degradation and dilution due to growth (and may be negative). What is being estimated in this work is the underlying fluorescent protein synthesis rate $\Phi(t)$ (strictly

positive), which is independent of dilution and degradation processes, and which is assumed to be proportional to the underlying cellular gene expression rate.

5.2.2 Dynamics Estimation as an Inverse Problem

Reconstructing the functions $\mu(t)$ and $\phi(t)$ represents an inverse problem, which is underdetermined and ill-posed [12]. In order to reduce the dimensionality of the inverse problem, prior knowledge of the functions $\mu(t)$ and $\phi(t)$ were exploited to construct a simple basis as follows. Expression rates and growth rates may be reasonably assumed to be strictly positive, and smooth on typical time scales of transcription and translation. The following approximation is proposed, given a function $f(t)$ that meets our assumptions,

$$f(t) \approx \sum_{k=0}^{n-1} \hat{f}_k G_k(t), \tag{5.4}$$

with

$$G_k(t) = \exp\left(\frac{-(t-k\Delta)^2}{2\Delta}\right) \tag{5.5}$$

which represents a sum of n Gaussian curves G_k with weight $\hat{f}_k$, variance Δ and regularly spaced over time t at intervals Δ. Here Δ determines the time scale of variation or smoothness of the representation of the function $f(t)$. Since the algorithm is intended to be used for bulk culture experiments, gene expression bursting and noise in growth cannot be observed, and the relevant time scales are the gene expression reporter half-life and the culture doubling time, typically less than one hour. Choosing Δ greater than the sampling interval of the data makes the system overdetermined and regularised in the sense that it is constrained to be smooth. In this case $\Delta = 1$ hour was used, the typical time scale of protein synthesis, which is larger than the usual sampling interval of 10-15 minutes (see Methods). Examples of reconstructed growth rate and gene expression rate profiles in turquoise and the function to be fitted in dashed black in Figure 5.1A, B. The effect of this approximate Gaussian basis can be seen from the dependence on Δ of the maximum slope of the basis functions $G_k(t)$, which scales as $1/\sqrt{\Delta}$. This means that the sharpest change in expression rate and growth rate profiles that can be reconstructed by the method is determined by Δ.

The model given in equations 5.2 and 5.3 combined with the approximation of equations 5.4 and 5.5 represents the forward models of the inverse problems for reconstruction of $\mu(t) \approx \sum_k \hat{\mu}_k G_k(t)$ and $\phi(t) \approx \sum_k \hat{\phi}_k G_k(t)$. In practice the measurements used to estimate B and y are discrete, will contain background signal, and are subject to noise. The background signals B' and y' are typically estimated by measuring appropriate control samples containing

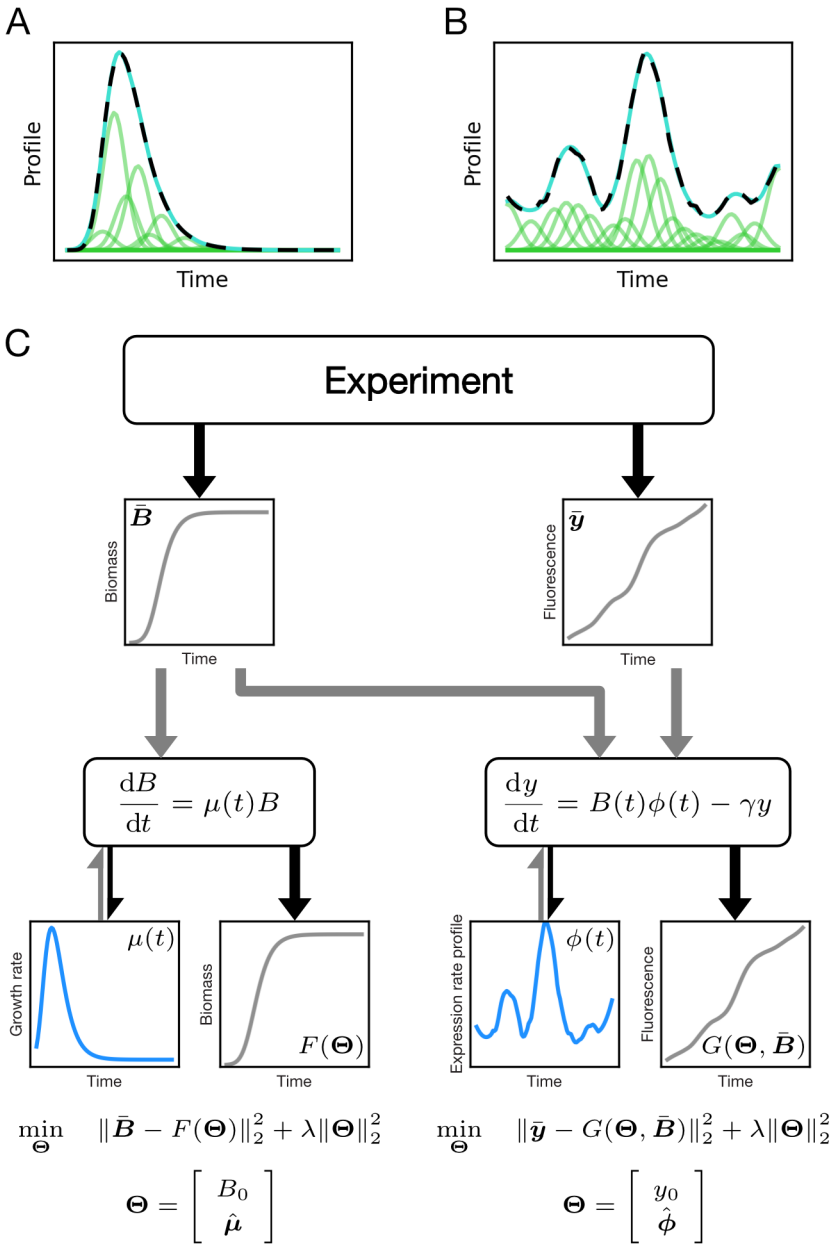


Fig. 5.1 Algorithm overview. **A, B** Growth and gene expression rates approximation using Gaussian basis. The light green curves are individual Gaussian curves that compose the basis. The turquoise line represents the sum of the Gaussian basis. The black dashed line represents the growth rate profile (A) or gene expression rate profile (B) to be fitted. **C** Inverse problem algorithm diagram. Once an experiment is performed, biomass data ($\bar{B}$) and fluorescence data ($\bar{y}$) are collected. These data are input to the models, where the biomass model requires only biomass as input and the fluorescence model requires both biomass and fluorescence. Finally, Growth rate ($\mu(t)$) and Expression rate ($\phi(t)$) profiles are generated from the biomass and fluorescence models, respectively. The gray arrows indicate inputs and black arrows indicate outputs.

no cells (B') and cells with no reporter expression (y') (see Methods). After subtracting these background measurements, the noisy estimates $\bar{B}$ and $\bar{y}$ are obtained. The the aim is to parameterize the forward models given by equations 5.2 and 5.3 such that $\|y-\bar{y}\|_2^2$ and $\|B-\bar{B}\|_2^2$ are minimized.

Note that while this approach uses a smooth basis to represent the reconstructed profiles, it does not filter or smooth the input data as in the indirect method. This method approximates the growth and gene expression rate profiles using a superposition of Gaussian functions to represent a continuous function as a discrete vector of parameters. The measurements and the estimated parameters are inputs of the forward model. The forward model then generates simulated measurements using different estimated parameters, and the algorithm computes the ones that minimise the difference between the measurements and the model.

5.2.3 Accurate Estimation of Growth Rate Dynamics

The inverse problem for reconstruction of the growth rate $\mu(t)$ can be stated as,

$$\min_{\Theta} \quad \|\bar{B}-F(\Theta)\|_2^2+\lambda\|\Theta\|_2^2 \tag{5.6}$$

with,

$$\Theta=\begin{bmatrix} B_0 \\ \hat{\mu} \end{bmatrix}. \tag{5.7}$$

It minimises the difference between noisy estimates of Biomass $\bar{B}$ and the result of the forward model $F(\Theta)$. Since this problem is ill-posed the Tikhonov penalty term λ was used for regularisation. Θ is the vector of parameters to optimise, containing the initial biomass B_0 and the Gaussian basis weights $\hat{\mu}_k$ to represent $\mu(t)\approx\sum_{k=0}^{n-1}\hat{\mu}_k G_k(t)$. The hyperparameter λ were chosen to minimise error within a reasonable range of values.

This problem is a nonlinear least squares optimization, which was solved using the trust region reflective algorithm [19]. Simulated data from equations 5.2 and 5.3 were generated using 100 randomly parameterized Gompertz growth models [202], and three different levels of measurement noise. The growth rate was characterised from these simulations using the inverse, direct and indirect methods. Hyperparameters for each method were optimised to minimise error within a reasonable range of values. A comparison between the results of the inverse method to the direct linear inversion method shows that the inverse method approach reduces mean squared error by more than 29-fold ($p<10^{-35}$, Welch's T-test) (Figure 5.2A).Then a comparison to the indirect method show that the inverse method reduced the mean squared error by more than 2-fold ($p<10^{-5}$, Welch's T-test) Figure 5.2A). Finally a comparison to a different indirect method that uses an anti-causal zero-

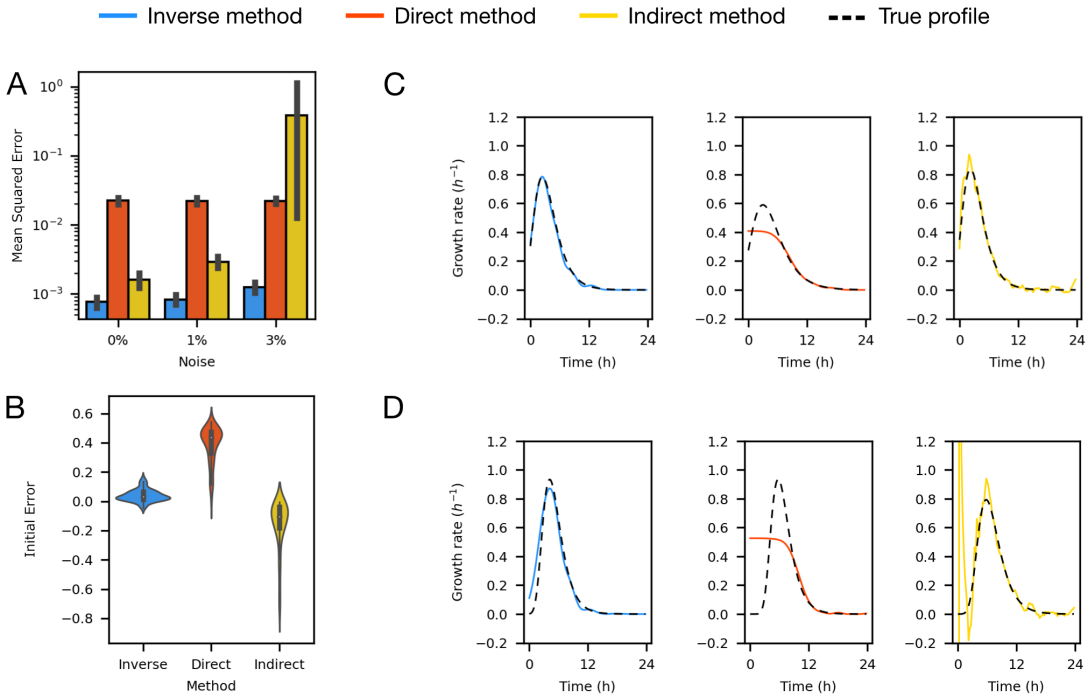


Fig. 5.2 Method comparison on growth rate simulations. **A** The errors corresponding to the inverse, direct and indirect methods are presented, the error of the inverse method were almost 30-fold lower than the direct method and two-fold lower than the indirect method, error bars represent 95% confidence interval. **B** Error of the initial growth rate over all noise levels. Growth rate, due to its nature, has a peak, which is not easily identified with the direct method. Since the initial growth rate should be low, the direct method overestimates and the indirect method underestimates the initial growth rate, showing that the inverse method is more accurate at low biomass. Error calculated for growth rate values at time 0 compared to the true profile. **C** Examples of accurate growth rate reconstructions from simulated data. Gompertz profiles are characterized using the inverse, direct and indirect methods (blue, red, and yellow lines respectively) compared to the true profile (black dashed line). **D** Examples of inaccurate growth rate reconstructions from simulated data. Gompertz profiles are characterized using the inverse, direct and indirect methods (blue, red, and yellow lines respectively) compared to the true profile (black dashed line). (n=100)

phase digital filter were performed and show that the inverse method reduced the mean squared error by more than a thousand fold. The indirect method that uses an anti-causal zero-phase digital filter were coded by CC in Matlab. The inverse method maintained the best performance at high noise levels. Further, the inverse method is more robust to noise in early biomass measurements, where the direct method overestimates the initial values and the indirect produces noisy reconstructions that usually goes below zero at the beginning (Figure 5.2B). The inverse method correctly reconstructed the lag phase, which is missing from the linear inversion solution. Our method also reconstructed the growth rate peak, effectively distinguishing between lag, exponential and stationary growth phases (Figure 5.2C,D).

5.2.4 Accurate Estimation of Gene Expression Rate Dynamics

In a similar way to growth rate, the aim is to find the optimal parameters,

$$\Theta = \begin{bmatrix} y_0 \\ \hat{\phi} \end{bmatrix}, \tag{5.8}$$

where Θ is the vector of parameters to optimise, containing the initial reporter intensity y_0 and the Gaussian basis with weights $\hat{\phi}_k$, and the forward model $G(\Theta, \bar{B})$. The problem is again a nonlinear least squares optimization,

$$\min_{\Theta} \quad \|\bar{y} - G(\Theta, \bar{B})\|_2^2 + \lambda \|\Theta\|_2^2 \tag{5.9}$$

which were solved using the same numerical procedure as for growth rate (See previous chapter). To test this approach, 100 random gene expression rate profiles were generated from smoothed lognormal random walks, with random Gompertz models for the biomass, and three measurement noise levels (see Methods). Again, the inverse method to the direct linear inversion method were compared and show that the mean squared error is more than four-fold lower ($p < 10^{-10}$, Welch's T-test) and close to three-fold lower than the indirect method ($p < 10^{-7}$ Welch's T-test) (Figure 5.3A). Finally the inverse method was compared to a different indirect method that uses an anti-causal zero-phase digital filter and showed that the inverse method reduced the mean squared error by more than a hundred fold. The gene expression simulations are not constrained to start with low values or to have an initial peak which make the three methods have similar initial errors (Figure 5.3B). The direct method does not correctly reconstruct early peaks in gene expression rate profiles, and the indirect method produces extremely noisy solutions (Figure 5.3C,D). The inverse method is the best performer capturing the shape of the expression rate profile, although this method suffers with

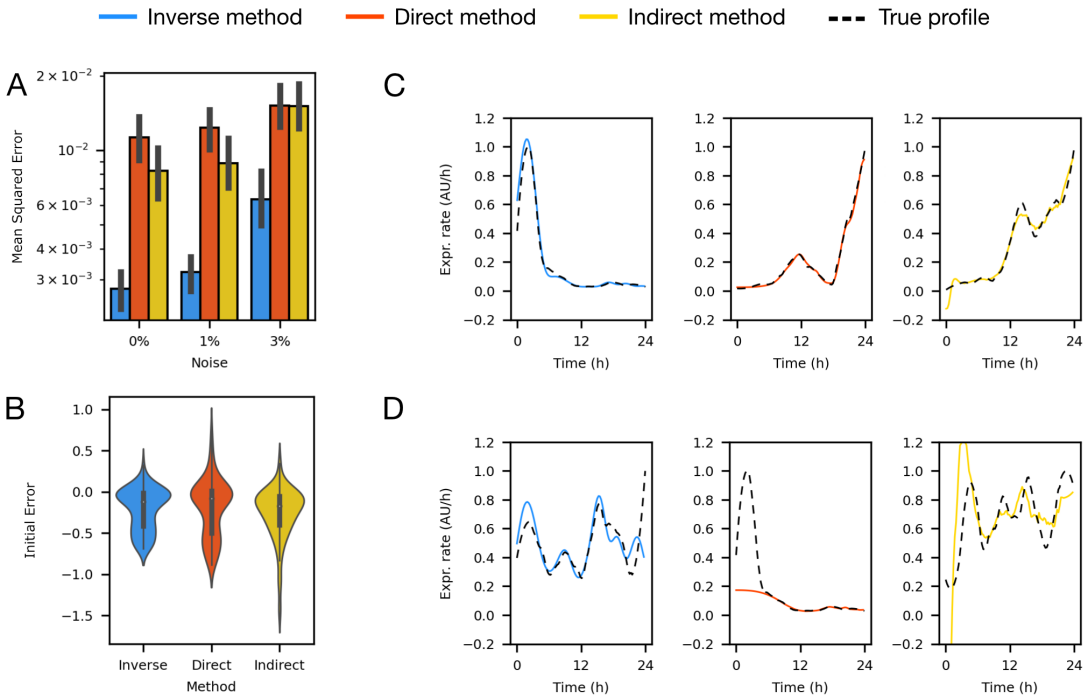


Fig. 5.3 Method comparison on gene expression rate simulations. **A** The errors corresponding to the inverse, the direct and the indirect method are presented; the inverse method has been shown to be four-fold better than the direct method and close to three-fold better than the indirect method, error bars represent 95% confidence interval. **B** Error at initial time. Since the initial gene expression rate simulations were not restricted to be low at the beginning the three methods show similar errors. Error calculated for expression rate values at time 0 compared to the true profile. **C** Representative examples of accurate gene expression reconstructions from simulated data. Random profiles characterised using the inverse, direct and indirect methods (blue, red, and yellow lines respectively) compared to the true profile (black dashed line). **D** representative examples of inaccurate gene expression rate reconstructions from simulated data. Random profiles characterised using the inverse, direct and indirect methods (blue, red, and yellow lines respectively) compared to the true profile (black dashed line). (n=100)

sharp changes due to its dependance with Δ (Figure 5.3C,D). The inverse method requires knowledge of the timescale of the process that you want to measure to set the hyperparameter Δ.

5.2.5 Characterizing Growth and Gene Expression Rate Dynamics in *Escherichia coli*

Using the inverse, direct and indirect methods growth and gene expression dynamics were reconstructed from experimental data of *E. coli* carrying a synthetic triple TU plasmid pAAA (Figure 5.4A, B). This plasmid contains three TUs with the same synthetic σ_{70} constitutive promoter J23101 [76] in different long-range DNA contexts determined by its position on the plasmid and in different short-range DNA contexts TU determined by different sets of promoter downstream elements RBS-CDS-Terminator (Figure 5.4A). Each CDS encoded a different fluorescent protein as a reporter. Assays were performed using a 96-well microplate reader to measure fluorescence in each reporter channel as a proxy for protein concentration and optical density as a proxy for biomass (see Methods). To assess the effects of organism context on growth and gene expression dynamics two strains of *E. coli* carrying the pAAA plasmid were measured, and to assess the effects of environmental context they were grown on two different carbon sources [196] (Figure 5.4B-E).

The inverse method captured the shape of the growth and gene expression rate profiles in a smooth way, consistent with population averages on the time-scale of protein synthesis, while the direct and indirect methods produced noisy solutions. The inverse method reconstructed a peak in growth rate consistent with the transition between lag, exponential, and stationary phase, as in the Gompertz growth model [63]. Furthermore, it captured a peak in the expression rate coincident with the growth rate peak, which is consistent with promoter dependence on σ_{70} and the abundance of this factor, as well as ribosomes, during peak growth (Figures 5.4B-E) [91, 155].

The gene expression rate profiles were different for TUs in the same plasmid under the control of the same promoter due to both different organism and long-range DNA contexts (Figures 5.4B-E). While in all cases peak gene expression rate coincided with peak growth rate, in some contexts multiple peaks were observed. The environmental (carbon source), organism context (strain) as well as the compositional context (promoter downstream elements, position and orientation in the plasmid), clearly change gene expression and growth dynamics, leading to different peak timing and overall shape. All growth rates characterised using the inverse method exhibited clear lag, exponential and stationary phases which are not

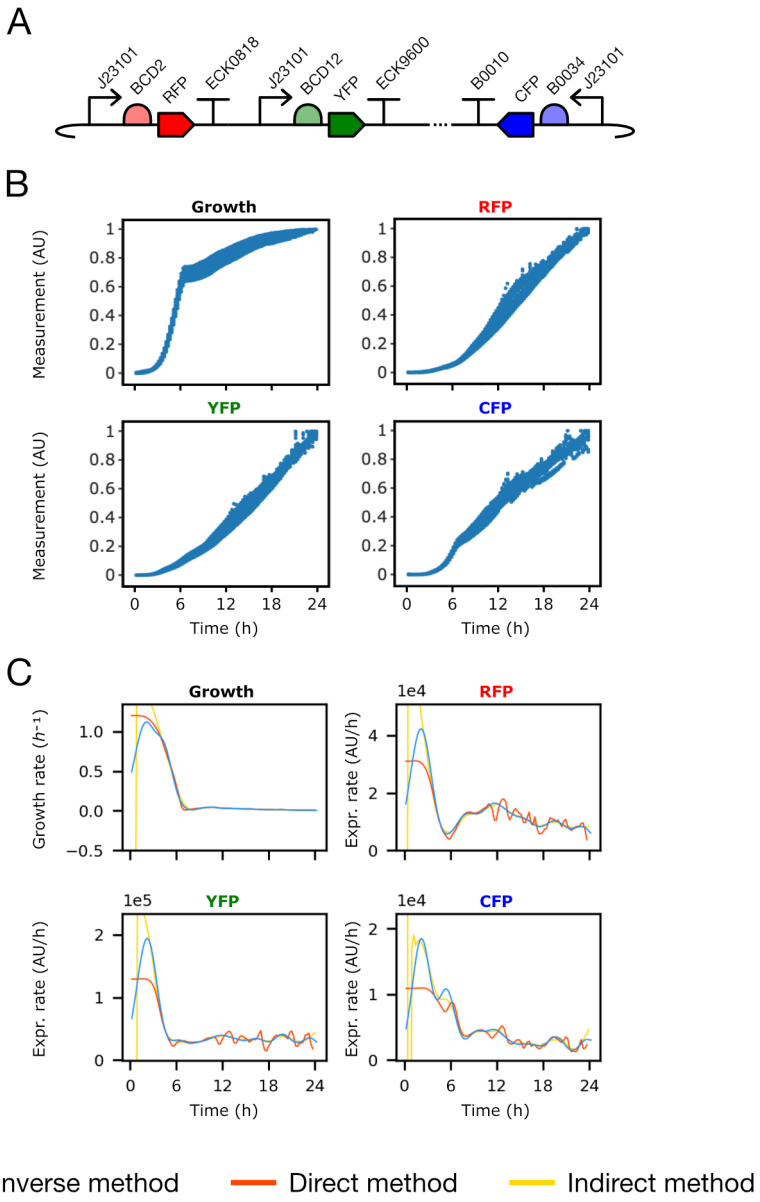


Fig. 5.4 Gene expression dynamics shape reconstructed from experimental data is different for transcription units with the same promoter. **A** Plasmid pAAA SBOL visual diagram. **B** Raw experimental data of OD and fluorescence. **C** Growth and gene expression rates reconstructed with the inverse, direct and indirect methods (blue, red and yellow lines respectively) on strain MG1655z1 in media M9-glucose. (n=30)

apparent with the direct method (Figures 5.4B-E). The timing of growth phase transitions were different in each organism context (Figures 5.4B-E).

5.2.6 Characterization of Gene Expression Rate Dynamics Relative to *in vivo* Reference

Gene expression magnitude has been characterised relative to a standard *in vivo* reference containing promoter J23101, in order to normalise for organism context [86]. Plasmid pAAA provides such a reference, with three TUs containing the same J23101 promoter in different compositional contexts. The hypothesis is that the reference plasmid could be used to characterise the dynamics of gene expression in a standardised fashion. Each TU in the pAAA plasmid presents a standard reference for a particular compositional context - the promoter downstream elements, position and orientation in the plasmid. The aim is to characterise TUs with arbitrary promoters relative to these reference TUs, allowing us to describe their dynamics in a concise way. In order to compare gene expression rate dynamics from different experiments, profiles were synchronised and normalised each one by subtracting its mean and dividing by its standard deviation. The timing of growth phase transitions is variable due to differences in initial conditions and experimental variability, which leads to differences in the timing of gene expression rate profiles. In order to correct for these differences, the reconstructed growth rate peak time t_0 was used to synchronise the expression rate profiles, shifting time to $\tau = t - t_0$, such that $\tau = 0$ is the time of peak growth rate.

This approach was tested on a collection of 14 combinatorial three-reporter plasmids, combining 10 different TUs which were each driven by one of seven promoters [160]. Each plasmid contained three TUs producing RFP, YFP and CFP, containing the same promoter downstream elements as the corresponding reference TU. The promoter downstream elements RBS, CDS and terminator for each reporter were maintained constant and referred using the reporter name. The CFP TU were maintained the same in all plasmids, to serve as a control [86, 143]. This collection of plasmids presented a variety of promoters in different TU compositional contexts, that is, with different promoter downstream elements. Each TU was assembled into multiple plasmid compositional contexts, that is, in the presence of different upstream or downstream TUs.

Three of the TUs in the collection contained a promoter from a family of constitutive promoters created by mutating a consensus sequence [76], which includes the reference TU promoter, J23101. The RFP reference TU matched the gene expression rate profile shape of the RFP TU with the constitutive promoter J23106 over different downstream TU contexts

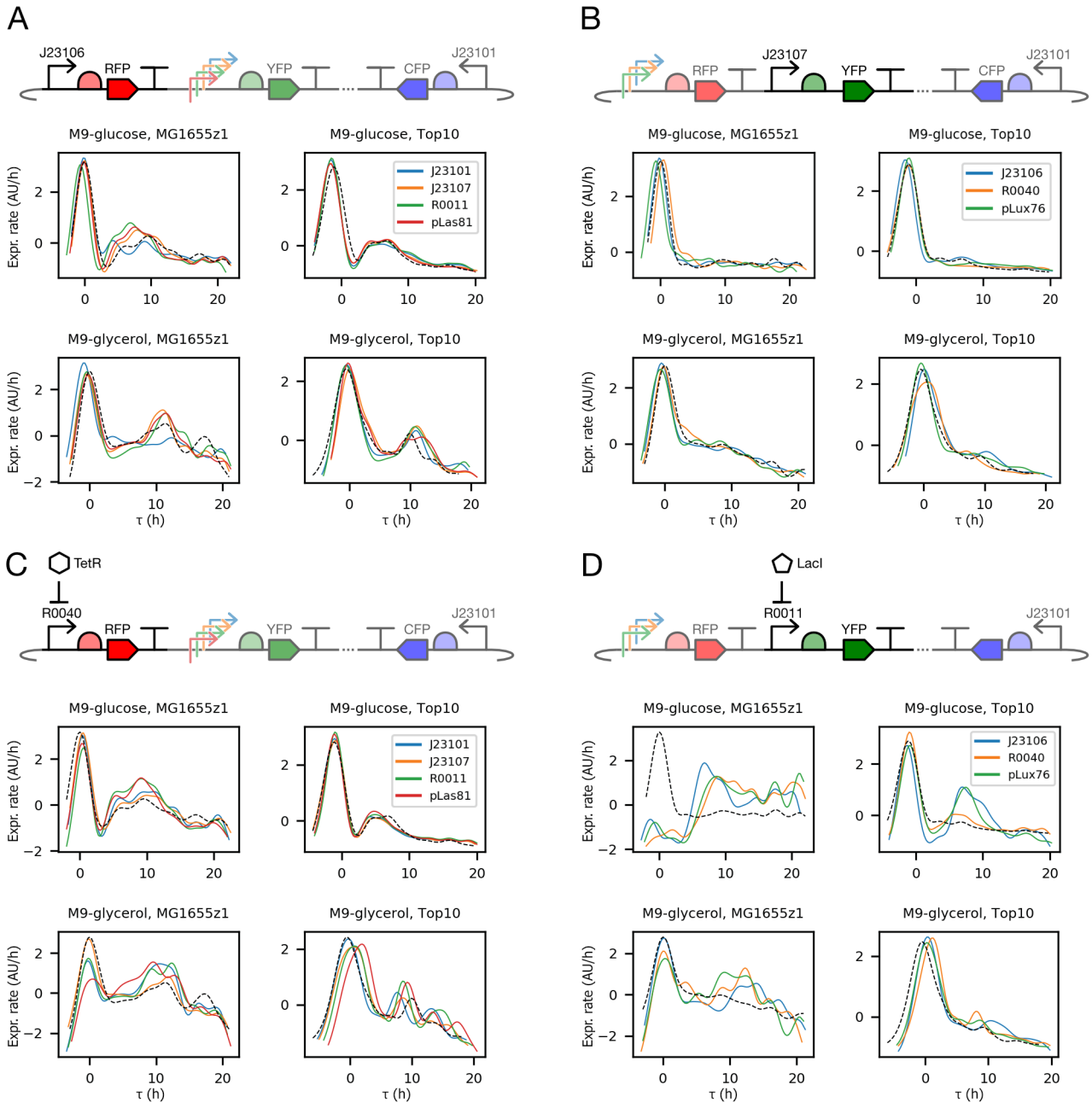


Fig. 5.5 The gene expression profile shape reconstructed from experimental data is similar for promoters under constitutive expression. TUs from pAAA were used as reference (black dashed line) for TUs within the same conditions but with different promoters (solid lines). **A** RFP reference TU compared to RFP TU with J23106 in different downstream TUs compositional context. **B** YFP reference TU compared to YFP TU with J23107 in different upstream TUs compositional contexts. **C** RFP reference TU compared to RFP TU with R0040 basal expression in different downstream TUs compositional contexts. **D** Reference YFP TU compared to YFP TU with R0011 with different upstream TUs compositional context. All profiles are shown normalised by mean and standard deviation.(n=30)

(Figure 5.5A). The YFP reference TU matched the gene expression profile shape of the YFP TU with the constitutive promoters J23107 (Figure 5.5B) and J23101 over different upstream TU contexts. The CFP TU were consistent across all compositional contexts. These results suggest that the dynamics are not affected by the tested plasmid compositional contexts.

Two of the TUs contained promoters that are repressible by a transcription regulating protein that binds to the promoter. In the case of the TetR responsive TU (containing promoter R0040), the RFP reference TU matched the gene expression rate profile over different downstream TU contexts, and over different organism contexts (Figure 5.5C). This is in spite of the fact that the genome of the strain MG1655z1 contains a constitutive TetR gene, while Top10 does not produce the protein. This result suggests that the dynamics of the TetR repressible TU are not affected by the action of the repressor.

The other repressible TU contained promoter R0011, regulated by LacI. LacI is produced by both strains and partly regulated by cAMP in Top10 [47]. The YFP reference TU matched the LacI repressible YFP TU gene expression rate profiles in M9-glycerol but did not match them in M9-glucose (Figure 5.5D). In MG1655z1 growing on glucose gene expression rates became negatively correlated with growth rate, and in Top10 on glucose they exhibited a second peak. These results suggest that transcription regulation can significantly affect the dynamics of gene expression compared to constitutive expression profiles.

The remaining two TUs contained promoters activated by different one-component signalling systems, measured in the absence of signal and regulator to study their basal expression. The RFP TU responsive to C6 homoserine lactone, containing promoter pLux76, were consistent with the constitutive reference TU in all contexts. However, for the YFP TU responsive to C12 homoserine lactone, containing promoter pLas81, the gene expression rate profile inverted its correlation with growth rate when downstream of the TU containing promoter R0040 and growing on glucose in strain MG1655z1. These results show the uncertainty that can be introduced in gene expression rate dynamics due to changing long-range sequence and organism context.

5.3 Methods

5.3.1 Software development

The inverse characterization method was developed in Python 3, its dependencies are Numpy [70], Scipy [181], Pandas [111]. Software source code is publicly available under MIT licence at https://github.com/RudgeLab/Inverse_Characterization.The method was integrated into Flapjack and is one of the standard analyses. Designs and simulations were

performed on a Apple MacBook Pro 15-inch, 2019, with a 2,6 GHz 6-Core Intel Core i7 processor, Radeon Pro 555X 4 GB and Intel UHD Graphics 630 1536 MB GPUs, and 16 GB 2400 MHz DDR4 RAM.

5.3.2 Kinetic Gene Expression and Growth Assays

Kinetic gene expression and growth assays were made culturing triple reporter plasmid pAAA containing bacteria (Figure 5.4A) using two different strains as well as two different carbon sources.

Monoclonal colonies of *E. coli* strain TOP10 or MG1655z1 transformed with plasmid DNA were picked and cultured for 14-15 hours overnight in M9 media with 50 µg/ml kanamycin, 0.2% w/v casaminoacids and 0.4% w/v glucose or 0.4% w/v glycerol. Overnight cultures were diluted 1000 times in 2 ml tubes. All the tubes were filled with 1996 µl of fresh M9 media, 2 µl of kanamycin and 2 µl of the bacteria liquid culture obtaining a final volume of 2 ml. In each well of a 96 well plate were added 200 µl, 4 wells with M9 media with the proper carbon source and kanamycin, 4 wells with non-transformed bacteria of the same strain and 10 wells of bacteria transformed with the appropriate plasmid to analyse from the previously prepared 2 mL tubes. Optical density and fluorescence in three channels (RFP, YFP and CFP) were measured approximately every 15 minutes for 24 hours in a Synergy HTX plate reader with Gen5 software. Each assay was repeated on 3 different days, with 10 replicates on each day, and each 96 well plate contained experiments with the same carbon source and strain. This method was performed by MMS.

5.3.3 Computational Methods

Computational methods were performed using Flapjack [196], Python [176], Numpy [70], Scipy [181], Pandas [111], Matplotlib [75], Plotly [133], Jupyter [92], Google Colaboratory [16], and Matlab. The direct methods were computed using the WellFARE package [201]. The indirect method with anti-causal zero-phase digital filters was computed using the Matlab function *butter* [25] by CC.

5.3.4 Web-based Software Implementation

Flapjack [196] (http://flapjack.rudge-lab.org) were extended to compute gene expression rate and growth rate profiles using the methods described above, using the WellFARE package [201] for the direct method. The indirect method is computed by Flapjack by filtering

the measured signals (biomass and reporter levels) using a Savitzky-Golay filter [153] and then differentiating the resulting smooth polynomial interpolation.

Flapjack is a systems and synthetic biology data storage and analysis tool, built as a web app that provides a user-friendly web interface and a REST and web socket API. The system allows upload of kinetic gene expression data from a variety of sources, and links it to metadata about experimental conditions. These data may then be queried and filtered and used to reconstruct gene expression and growth rates. Flapjack automatically subtracts background signal from both reporter and biomass measurements, taking the average of control samples (untransformed cells or media with no cells) at each time point.

5.3.5 Random Profile Generation

In order to generate a range of gene expression rate profiles for method comparison, with minimal assumptions about their form, lognormal random walks were generated,

$$\Phi(t) = \prod_{i=0}^{t} \xi_i \tag{5.10}$$

with $\log(\xi_i) \sim N(0, \sigma^2)$, and $\sigma^2 = 0.25$. The profiles ϕ_t were then smoothed using a second order Savitzky-Golay filter [153] with window size 21, and normalised to $[0,1]$. To generate random growth rate profiles, the Gompertz equation were used,

$$\log\left(\frac{B}{B_0}\right) = A \exp\left(-\exp\left(\frac{\mu^* \mathrm{e}}{A}(\lambda - t) + 1\right)\right) \tag{5.11}$$

with μ^* the maximal growth rate uniformly distributed on $[0.5,1]$ per hour, λ the lag phase length uniformly distributed in $[0,4]$ hours, $A = \log(B^*/B_0)$, where the maximal biomass $B^* = 1$ and minimum biomass $B_0 = 0.01$. The growth rate profile implied by this equation is given by,

$$\mu(t) = \mu^* \exp\left[\frac{\mu^* \mathrm{e} \cdot (\lambda - t)}{A} - \exp\left(\frac{\mu^* \mathrm{e} \cdot (\lambda - t)}{A} + 1\right) + 2\right]. \tag{5.12}$$

5.3.6 Simulation of Kinetic Gene Expression and Biomass Data

Equations 5.2 and 5.3 were solved using the forward Euler integration scheme with time step $\Delta t = 2.4 \times 10^{-2}$ hours for a period of 24 hours. Noise and background were added according to the following equations with $B' = 0.1$ and $y' = 0.1$,

$$B_t = (B(t) + B')(1 + \varepsilon_t) \tag{5.13}$$

$$y_t = (y(t) + y')(1 + \zeta_t), \tag{5.14}$$

where ε_t and ζ_t are uncorrelated white noise with variance σ^2, due to the measurement process. Simulated measurements were generated using LOICA [178] then uploaded to Flapjack [196], and analyzed using the API via Python.

5.3.7 Choice of hyperparameters

Each of the methods tested in the main text is dependent on one or more hyperparameters. In the case of the direct method this is the so-called insignificant value ε_L [201], for the indirect method it is the Savitzky-Golay filter window size, and for the inverse method they are Δ and λ. For reconstructions of simulated data these parameters were optimised by scanning a range of reasonable values and choosing the parameter that minimised the mean squared error. For experimental data, the value of λ were chosen using the L-curve method [69], ε_L were taken from the original paper [201], the value of Δ were fixed at 1 hour, and the Savitzky-Golay window size fixed at 11. For the anti-causal zero-phase digital filter, a second order Butterworth filter [25] with cutoff frequency 4/33 were used.

5.4 Discussion and conclusions

In this chapter, I created an algorithm for time-series estimation from plate reader typical fluorescence and absorbance measurements. Implemented the algorithm as a method for characterization on Flapjack data management tool that is available for LOICA (from Chapter 3) for model parameterization. Benchmarked the algorithm to previously described algorithms using synthetic data created with LOICA. Used the algorithm to characterise gene expression rate relative to an *in vivo* reference.

Accurate characterization of dynamic gene expression and growth rate profiles is essential for characterization of genetic circuits and inference of gene regulatory interactions in natural networks [172, 30, 7]. We have demonstrated an inverse problem approach to reconstructing dynamic gene expression rate and growth rate profiles from noisy kinetic measurement data. We compared our method to the current state-of-the-art algorithms, direct linear inversion [201], and indirect smoothing and differentiation [42]. Our approach reduced the mean squared error of reconstructions from simulated data of growth rate by almost thirty-fold and gene expression rate by more than four-fold with respect to the direct method.

The comparison showed that the direct method often fails to capture peaks at the beginning of the profiles. Indirect methods, even after filtering the noise, fail to reconstruct early stages of growth. This is likely because the biomass can have very low and even negative values

after background correction and dividing by these values can result in amplification of noise. This highlights that particular attention should be paid to reducing noise from experimental procedures and measurement techniques since all methods attain lower reconstruction errors with less noise. Surprisingly, our indirect method often performed better than the direct method, but our inverse problems approach improved on the indirect method by almost three-fold for gene expression rates, and more than two-fold for growth rate. Further, we were able to reconstruct features of both growth rate and gene expression rate profiles, such as exponential phase and peak growth, that were not apparent from the direct linear inversion method nor the indirect method. The growth rate peak is an important feature captured better with our method, allowing synchronisation of gene expression rate profiles.

While in terms of computation time our method is relatively slow, it is most intuitive to adjust by estimating the time-scale of dynamics to obtain Δ. The indirect methods require knowledge of the signal processing filters used, and the direct method requires tuning the "insignificant value" which is rather obscure.

Using our method we showed that the dynamic form of gene expression rates, not only their magnitude, is determined both by organism and compositional contexts. We examined two types of compositional context. Firstly, the short-range DNA context, corresponing to the composition of parts within a TU, not only the sequence of the promoter, determined the dynamics of gene expression rates. Secondly, the long-range DNA context, gene expression rate profiles were largely independent of the context in which the TU was placed, that is the upstream and downstream TUs. In most cases gene expression rates peaked in the exponential growth phase, with some promoters exhibiting a second peak in the stationary phase. This may be an effect of the abundance of different sigma factors in each growth phase and the sensitivity of the promoter to them. It is known that the binding sites for σ_{70}, most abundant in exponential growth phase, and σ_S, most abundant in stationary growth phase, are very similar [56]. Therefore a promoter with peaks in both exponential and stationary phases may be activated by both σ_{70} and σ_S to different extents and the peaks caused by variations in the sigma factor abundance in each growth phase. In some cases gene expression rate profiles inverted their correlation with growth rate, highlighting the uncertainty introduced by changing circuit composition. This uncertainty may be due to various mechanisms that modulate gene expression at sequence level such as DNA supercoiling [198].

Furthermore, our results also showed that the organism context, that is media and strain, changed dynamic gene expression and growth rate profiles. Our results suggest that while dynamic characterization for TUs under constitutive or leaky expression relative to an *in vivo* reference could be useful, uncertainty due to compositional and organism context must be taken into account. This highlights the need for strategies to mitigate compositional context

effects, such as gene expression load [116], five prime untranslated region [104], as well as techniques to predict interactions of genetic elements at sequence level [123, 165, 27].

Typically, measurements of promoter-reporter fusions are used as a proxy for transcription rates under various levels of transcription factors, external signals, and other determinants of gene expression [123, 169, 143, 159]. However, applying our method to multiple reporter TUs with the same promoter, we showed that gene expression rate profiles should not be taken as indicative of intrinsic characteristics of promoters, since they are affected by both the promoter and downstream genetic elements, gene position and orientation, and external factors such as carbon source and host strain.

Synthetic biology aims to design novel genetic networks or circuits from compositions of transcription units. It relies heavily on characterization of the functions of these TUs. Fundamentally, gene expression rates must be reconstructed from noisy measurement data in a range of conditions, including concentrations of inducer chemicals. The function of the circuit may then be mathematically modelled as in equation 5.1. Methods such as ours will enable such approaches for dynamical systems [49, 58] where the dynamic profile of gene expression rates is essential to the operation of the circuit.

In order to model circuit operation in this way, calibration of the fluorescence and biomass signals with respect to standard references [10, 9], or the use of relative (ratiometric) quantification [123, 143] are necessary, and can be easily incorporated into our workflow. Our results show that profiles for constitutive gene expression were consistent across a range of promoters in a range of contexts suggesting that an *in vivo* reference gene may be used to infer the expression profiles of other genes in a circuit without directly measuring them. The inverse problems approach provides a framework that could be extended easily to fit more complex models of gene expression and also for regulatory parameters (e.g. Hill functions) as well as dynamic profiles, providing an accurate and flexible characterization method for synthetic biology.

Chapter 6

Automated workflows to engineer synthetic genetic networks

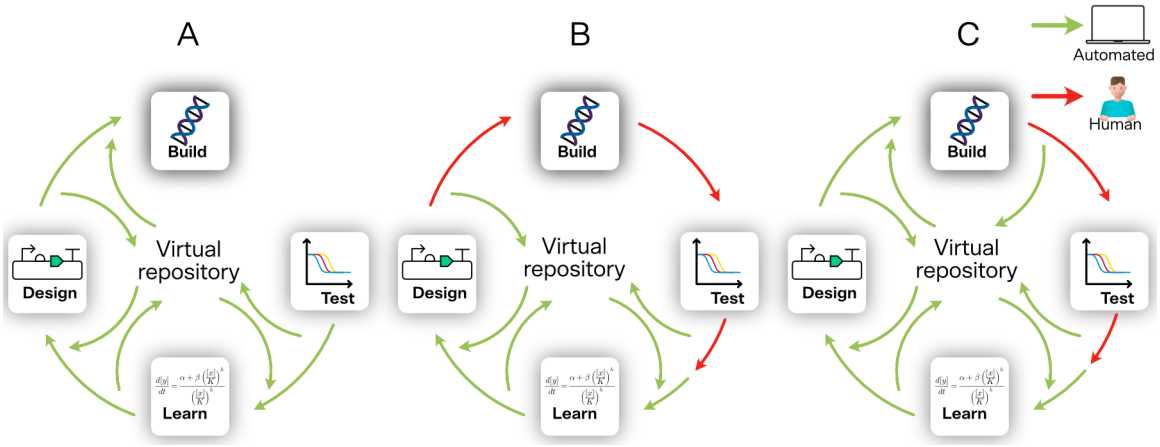


Fig. 6.1 DBTL workflows with increasing levels of connection and automation. (A) A virtual DBTL workflow with simulated data. (B) A DBTL workflow with manual build stage automated with software. (C) A DBTL workflow using lab automation and software.

6.1 Introduction

The software tool ecosystem in synthetic biology is still very new with a lot of dead software tools created by graduated students. Also, most tools read and produce data in incompatible formats making difficult the connection between tools and leaving an open DBTL cycle. Workflows are orchestrated repeatable patterns that allows for the systematic organization of resources into processes. Here I combined tools developed during this work with stable tools from the SBOL ecosystem that form modular parts of different DBTL workflows. I describe three DBTL workflows with increasing levels of connection and automation: a virtual workflow with simulated build and test (Figure 6.1 A), a workflow with manual build (Figure 6.1 B) and a workflow with automated build stage (Figure 6.1 C).

6.2 Results

6.2.1 Genetic design automation exploring the design space with simulated DBTL cycles

LOICA were used to generate repressilators given a set of NOT gate parameters. Hill function parameters were collected from the literature [159]. These parameters were used to create 18 `Operators`. Permutations of three `Operators` were instantiated to design 4896 oscillatory genetic networks. The inputs of each `Operator` were selected as the `Regulator` that represent the repressor of the promoter associated with that `Operator`. The output of each `Operator` were selected as the `Regulator` that represent the repressor of the promoter

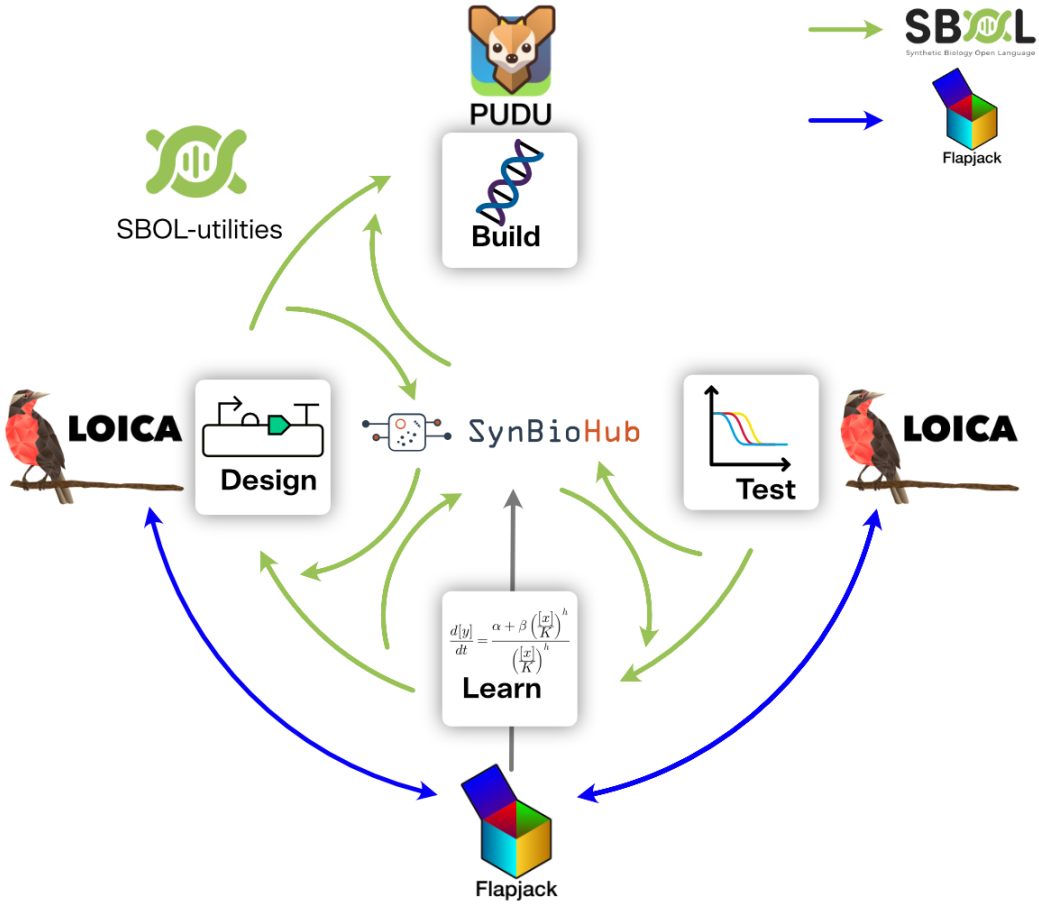


Fig. 6.2 Simulated DBTL workflow diagram. LOICA is used at the design stage, then the output from LOICA is formatted using sbol-utilities which works as input for PUDU. PUDU simulates the build and generates metadata. Simulated data from LOICA is uploaded to Flapjack where it can be processed and analysed.

associated with the next `Operator` of the permutation and a fluorescent protein `Reporter` to keep track of its expression. Deterministic simulations without noise were run using the Gompertz `SimulatedMetabolism`, with one `Sample` per `Assay`.

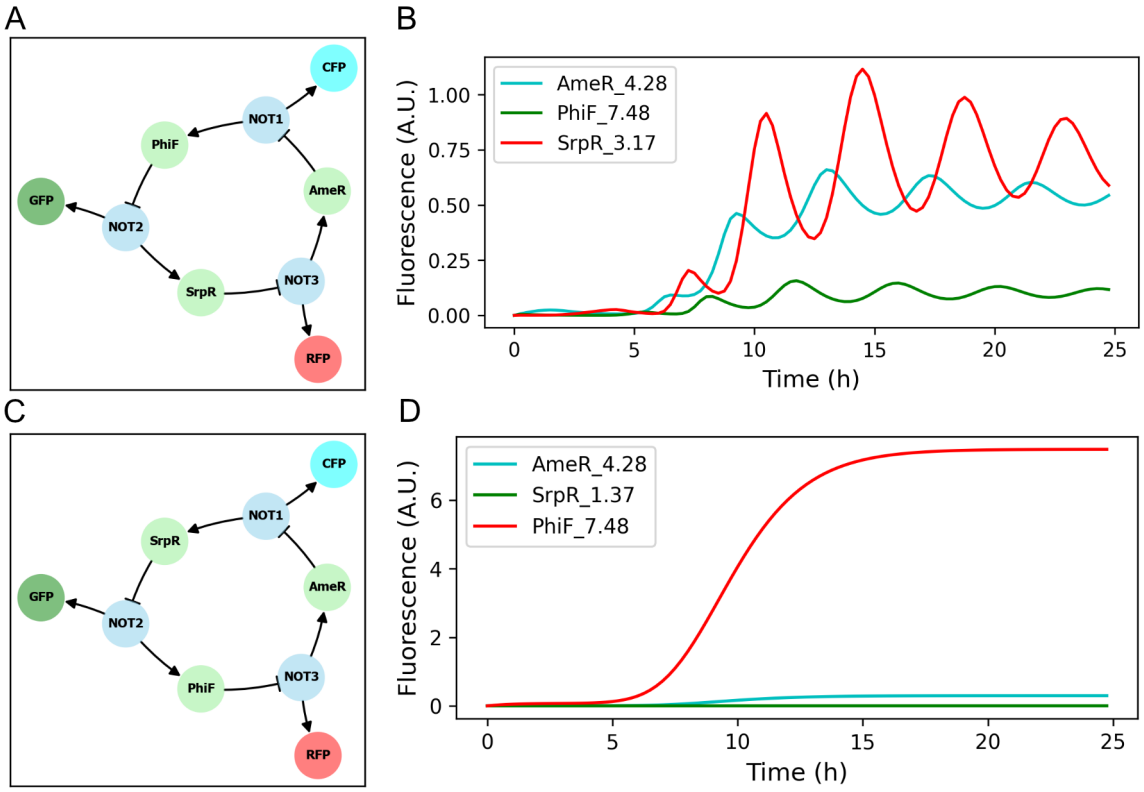


Fig. 6.3 Repressilator generator simulations. (A) Functional oscillatory `GeneticNetwork` diagram (B) Functional oscillatory `GeneticNetwork` simulation. (C) Non functional oscillatory `GeneticNetwork` diagram (D) Non functional oscillatory `GeneticNetwork` simulation.

A functional oscillator exhibits multiple peaks (Figure 6.3 A,B) in contrast to a non functional oscillator with no peaks (Figure 6.3 C, D). The fluorescence profiles of each `Reporter` were analysed looking for peaks. If a design has a number of peaks greater than 2 in the simulations the `GeneticNetwork` were stored as a likely functional genetic oscillator. With this the design space was reduced from 4896 to 2271 designs which have the desired behavior and can proceed to test. More examples of functional and non functional genetic oscillators along the code to design and simulate them are in https://github.com/Gonza10V/Automated-design-build-test-learn-workflows-to-engineer-synthetic-genetic-networks/tree/main/notebooks.

A simulated DBTL cycle can be iterated using the software tools for design, build and learn stages (Figure 6.5). In this workflow characterization data was obtained from the bibliography and used to generate LOICA simulations with different noise levels between what is

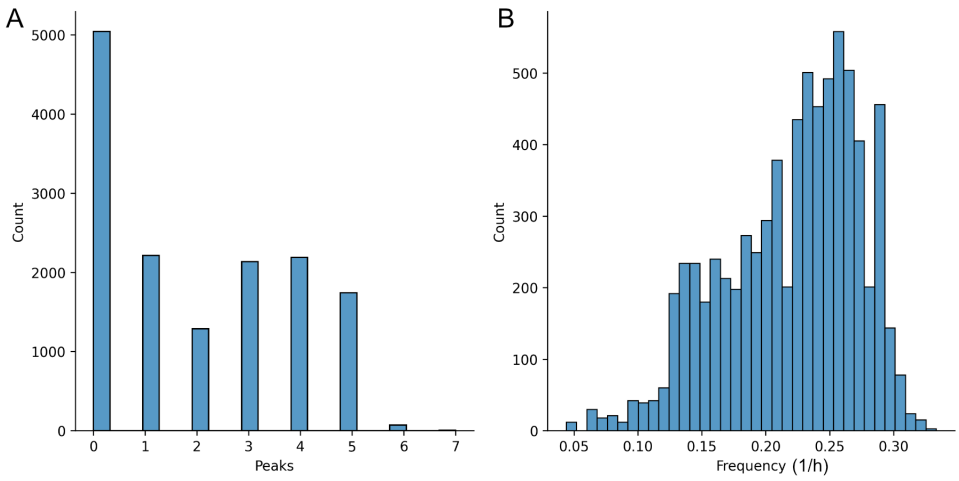


Fig. 6.4 Analysis on the count of number of peaks and frequency on simulated repressilators. (A) Count of peaks over samples and signals. (B) Count of frquencies over samples and signals.

common for typical plate reader data [159]. Then the simulated data were uploaded to Flapjack and LOICA were used to characterise their values (Figure 6.5). Designs assembly was simulated with PUDU 4.2.4 using `SBOL DNA Assembly`. Then, characterised `Operators` were used to simulate combinations to look in the design space for combinations of parts to form a functional repressilator. Finally, I calculated the frequency of each repressilator creating a dataset that links genetic designs with functional features.

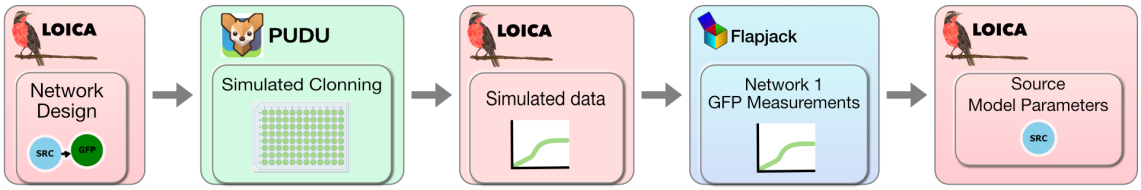


Fig. 6.5 Workflow execution with simulated build. LOICA was used to generate simulations from parameters from the literature. Simulated data from LOICA was uploaded to Flapjack. Data was analysed, looking for peaks to identify functional repressilators and calculate frequency.

6.2.2 Automating the DBTL cycle with manual build to characterise constitutive gene expression

A DBTL cycle with manual build were iterated using just software tools to automate the design and learn stages(Figure 6.6). The goal is to compare the constitutive expression of GFP with different degradation tags. The experimental design consists in building TUs

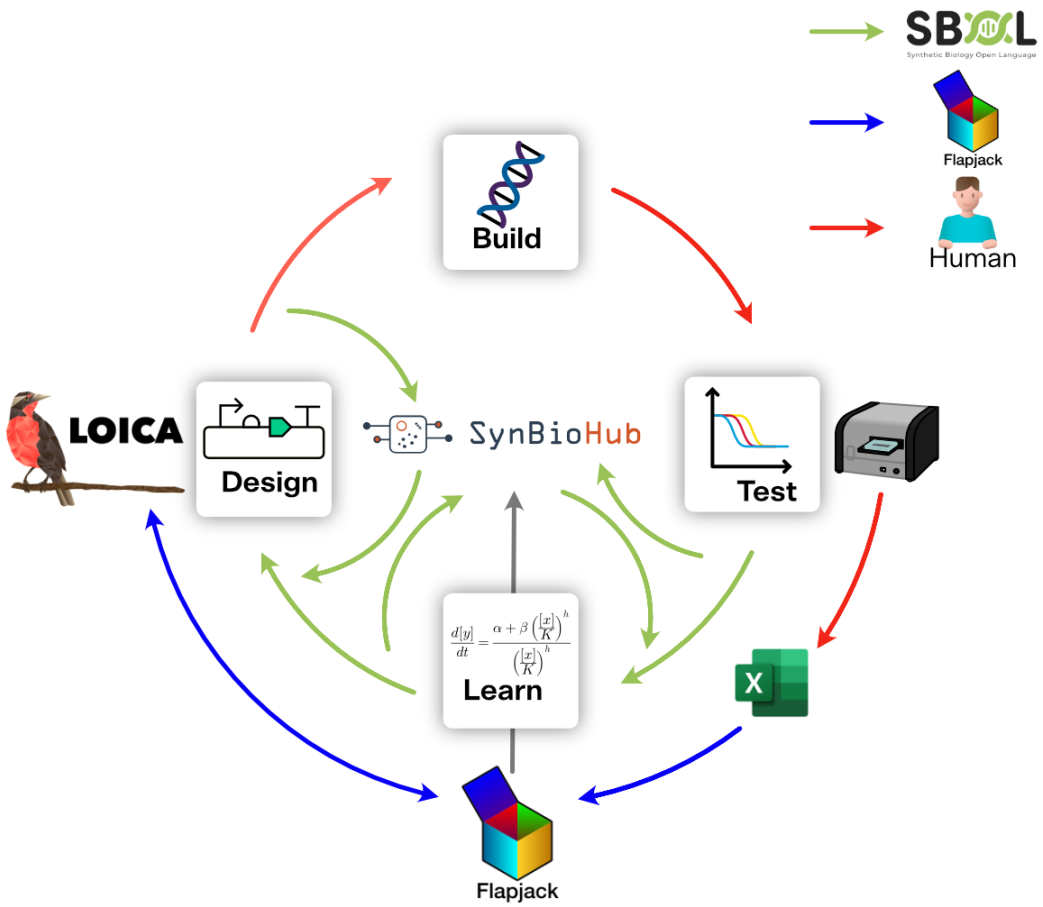


Fig. 6.6 Manual build workflow diagram. LOICA is used at the design stage, then the output from LOICA is formatted using sbol-utilities which works as input for PUDU. PUDU simulates the build and generates metadata. The build stage, which includes the final definition of the sequence to construct, the construction of DNA and the transformation of an organism with that DNA is done manually. Experimental data is uploaded to Flapjack where it can be processed and analysed.

composed of the standard promoter J23100, the standard RBS B0034, double terminator B0015 and a combination of GFPmut3 or sfGFP with 3 degradation tags. The degradation tags consisted in one with just a stop codon at the beginning or position 1, one with a random sequence and a stop codon at position 33, and the M0050 degradation tag with stop codon at position 33. The sequence design for the new DNA parts, GFP CDSs and degradation tags, were done in Benchling adding fusion sites for the MoClo universal acceptor pSB1C00 and SapI restriction enzyme recognition sites, following the assembly standard selected at Chapter 1, more details can be found in Methods. The DNA sequences were exported in GenBank format and used to order double stranded DNA fragments. The same GeneBank was used to create a virtual representation compliant with the SBOL data standard and best practices for build representation from Chapter 4. DNA sequences in SBOL were handled using pySBOL to create simple components for `Operators` and `GeneProducts`.

The `Source` were linked to a SBOL composition of J23100 the standard promoter from the Anderson collection and B0034 the standard RBS from Elowitz (Table 6.1) to create a standard expression `Source` (Table 6.2).

Glyph	**Description**
	Promoter J23100
	RBS B0034

Table 6.1 Parts for `Operators` for manual build workflow.

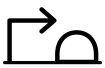
Glyph	**Description**
	Standard `Operator`

Table 6.2 `Operators` for manual build workflow.

The `Reporters` were linked to a SBOL composition of different GFP CDSs with a fusion sequence either a inert sequence or a degradation tag and the same terminator B0015 across all of them (Figure 6.3) to create 7 different `Reporters` (Table 6.4).

These components are an optional input in LOICA that is used to compose them into TUs and generate the SBOL representation of the whole genetic network. The genetic

Glyph	Description
	sfGFP non stop
	GFPmut3 non stop
	Dummy degradation tag with stop codon at position 1
	Dummy degradation tag with random srquence and stop codon at position 33
	Degradation tag M0050 with stop codon at position 33
	Terminator B0015

Table 6.3 Parts for `Reporters` on manual build workflow.

network SBOL representation can be exported and uploaded to SynBioHub to store the design metadata.

Then at the build stage, synthesised double stranded DNA fragments were domesticated using the universal acceptor pSB1C00 following the assembly strategy from Chapter 2. All the experimental work was performed manually which includes but is not limited to: assembly, transformation, colony selection and plate setup. The designs were manually assembled into the Odd1 Loop receiver. A digital representation of this process was created using sbol-utilities `Assembly plan composite in backbone single enzyme` from Chapter 4. The assembled DNA was manually introduced into *E. coli* DH5α using the manual transformation protocol. Colony selection was performed in plates with LB containing 50 ug/uL kanamycin looking for colonies with green fluorescence. Selected colonies were grown overnight, from each tube 500 uL were used to make stocks stored at -80 C and the rest 4.5 mL were used for DNA extraction. DNA was quantified using a nanodrop spectrophotometer and the elution buffer was used as blank, obtaining measurements of concentration and quality. Extracted and quantified DNA were diluted for sequencing. Upon sequence confirmation stocks were grown overnight and samples were prepared and distributed in a 96 well plate for measurement. All the build metadata can be captured in SBOL format using the functions from Chapter 4 and uploaded to SynBioHub.

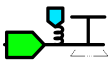
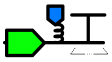
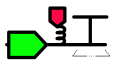
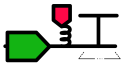
Glyph	Description
	sfGFP, dum0 and B0015 `Reporter`
	sfGFP, dum33 and B0015 `Reporter`
	sfGFP, M0050 and B0015 `Reporter`
	GFPmut3, dum0 and B0015 `Reporter`
	GFPmut3, dum33 and B0015 `Reporter`
	GFPmut3, M0050 and B0015 `Reporter`

Table 6.4 `Reporters` for manual build workflow.

The 96 well plates with samples produced at build were then introduced in a BMG Clariostar Plus plate reader to get measurements every 10 minutes over 25 hours. The measurements were acquired using a protocol to obtain OD600 absorbance and green fluorescence over time (more details in Methods). Then measurements obtained in a plate were used to fill the Flapjack Excel template to upload experimental data and metadata. The raw output from the plate reader was copied using one tab per measurement type, in this case one for OD and one for fluorescence. The metadata captured in this experiment were DNA, media and strain. After completing the Flapjack Excel template the data were uploaded to Flapjack using the BMG parser, one of the built-in Flapjack parsers. The data in Flapjack were queried, visualised and inspected using the front-end web app and the Python package pyFlapjack to communicate with the API in an programmatic way. After checking that there are no problems with the data, LOICA characterization method was used, see Chapter 3. Characterised `Sources` are represented by its expression rate profile which can be used in future `GeneticNetwork` designs.

Flapjack front-end was used to visualise the data and perform preliminary visualization of the raw data (Figure 6.7 A). The fluorescence is similar for all samples with some minor differences for GD0006 and GD0009 which are the ones with the M0050 degradation tag, the major difference can be seen in the OD where they grew slower (Figure 6.8). LOICA objects previously created were characterised and got an expression profile (Figure 6.7 B).

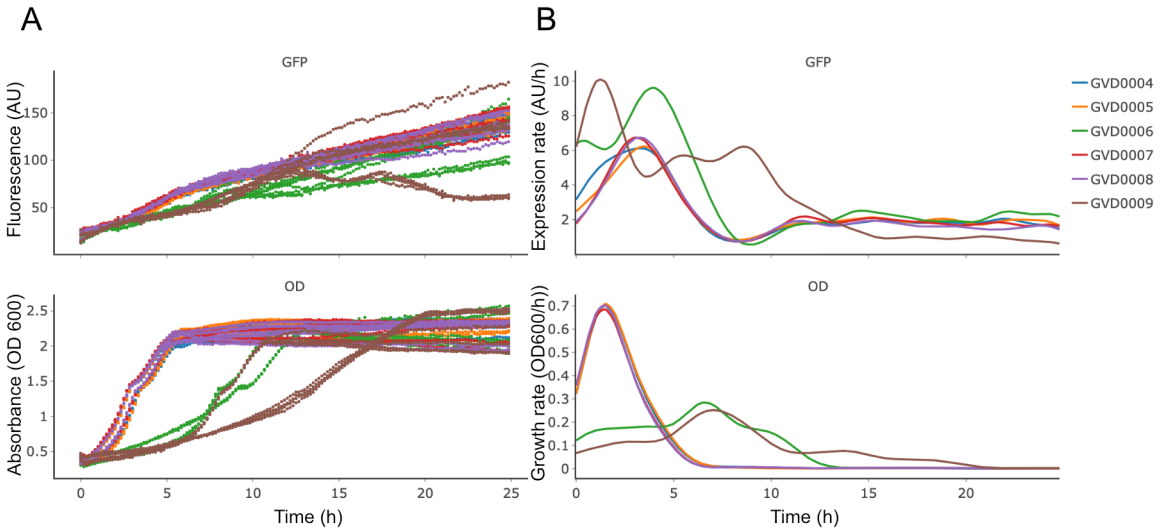


Fig. 6.7 Degradation tags characterization data and analysis on manual build workflow. (A) Experimental data points of green fluorescence and OD 600. (B) Gene expression rate and growth rate from experimental data.

The gene expression rate profiles of GD0006 and GD0009 are notably different from the rest suggesting that degradation tags have an effect on gene expression. More notable is that the growth rate peak is delayed and smaller in TUs with degradation tags compared to TUs with stop codons with or without random sequence. The expression profile then can be used in other simulations informing the design stage and closing the DBTL cycle (Figure 6.9).

6.2.3 Connecting software tools and liquid handling robots to automate the DBTL cycle

A DBTL cycle with automated build were iterated using the software tool to automate the design, build and learn stages and liquid handling robots at the build stage(Figure 6.10). The framework starts using LOICA at the design stage 6 TUs were defined combining 6 `Source Operator` with the `Reporter` sfGFP to create 6 simple `GeneticNetworks`. These `GeneticNetworks` were exported to SBOL and used to create standardised build plans with sbol-utilities.

The `Source` were linked to a SBOL composition of each promoter with the B0034 standard RBS from Elowitz (Table 6.1) to create a promoter working constitutively as a `Source` in the absence of its repressor.

The `Reporter` was linked to a SBOL composition of a sfGFP CDS with the B0015 terminator (Figure 6.3). This `Reporter` is combined with all the different `Sources`.

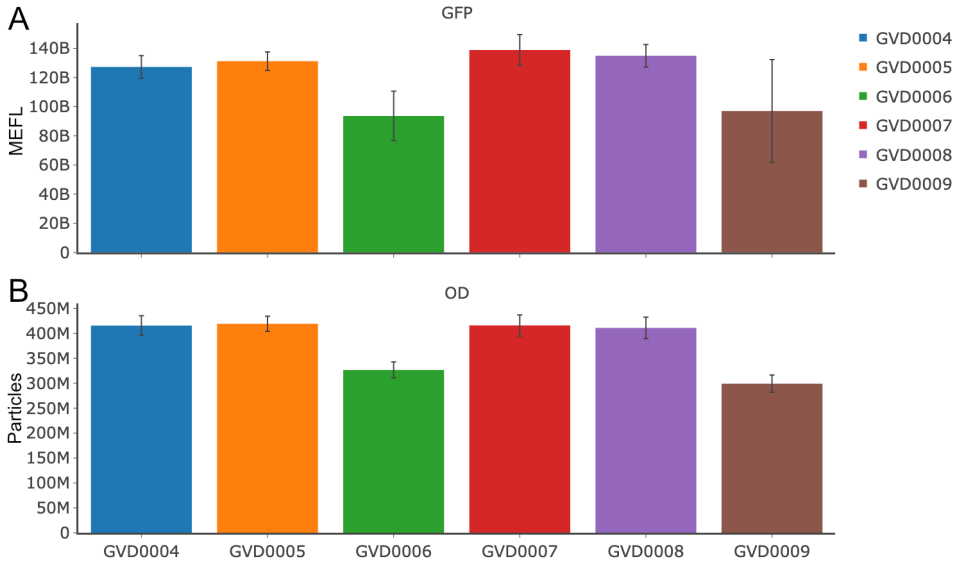


Fig. 6.8 Mean expression and biomass over different `Reporters on manual build workflow from experimental data`, bars represent SD, N=3. (A) Mean expression rate in MEFL, calibrated units. (B) Mean biomass in particles, calibrated units.

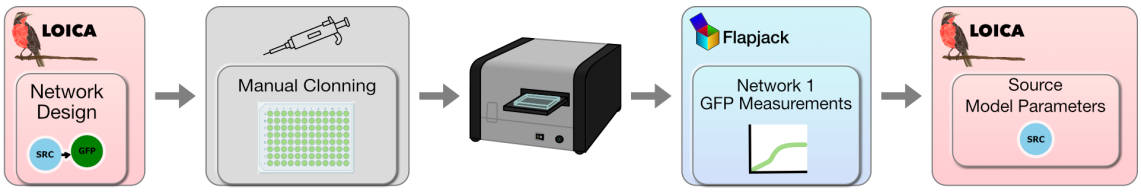


Fig. 6.9 Workflow with manual build execution. LOICA were used to generate simulations from parameters from the literature. Build were simulated using PUDU and performed manually. Experimental data were uploaded to Flapjack. Data were visualised and analysed in Flapjack.

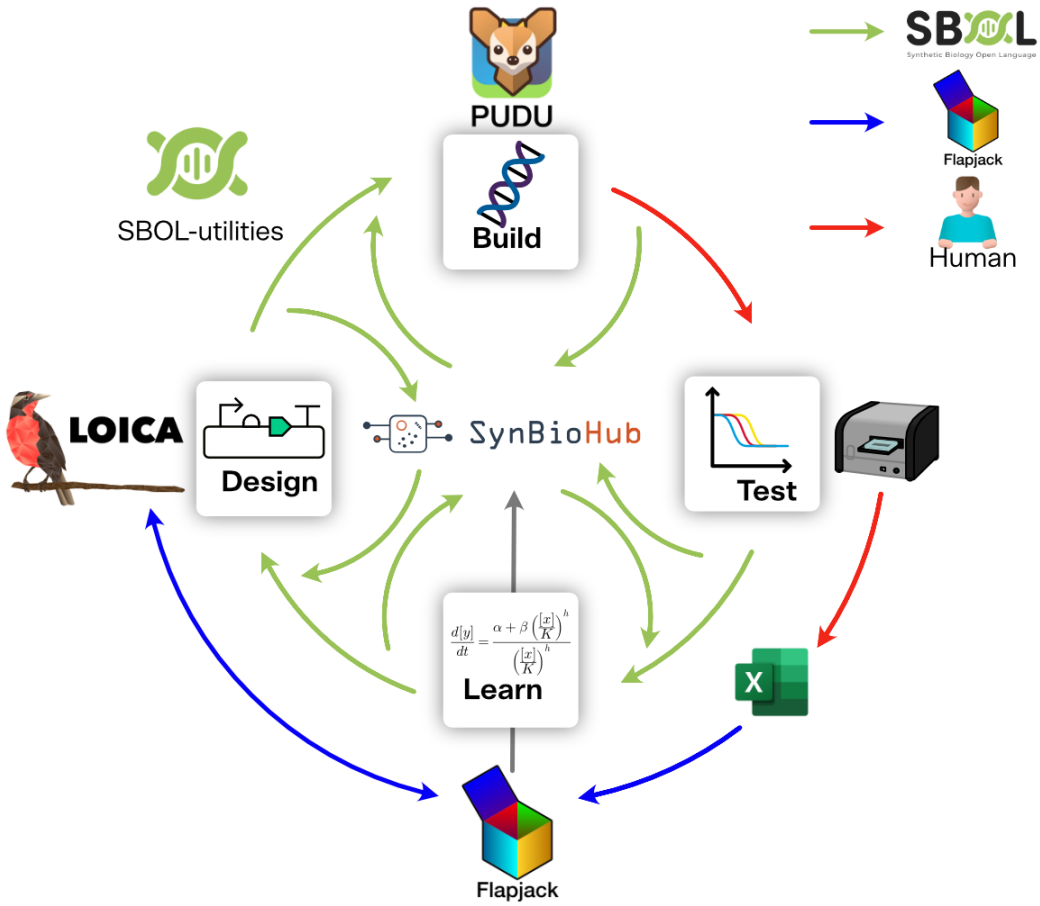


Fig. 6.10 Automated build workflow diagram. LOICA is used at the design stage, then the output from LOICA is formatted using sbol-utilities which works as input for PUDU. PUDU simulates the build and generates metadata. Build is done using PUDU to control lab automation. Experimental data is uploaded to Flapjack where it can be processed and analyzed.

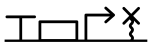
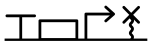
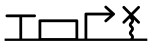
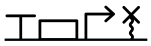
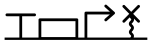
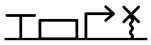
Glyph	Description
	Promoter GVP008, AmeR repressible
	Promoter GVP0010, BetI repressible
	Promoter GVP0012, BM3R1 repressible
	Promoter GVP0013, HlyllR repressible
	Promoter GVP0016, SrpR repressible
	Promoter GVP0017, LitR repressible
	RBS B0034

Table 6.5 Parts for `Operators` for automated build workflow.

Then, at the build stage PUDU were used to create an automated protocol to aid the cloning process. `Loop DNA Assembly` was used in its libre version to generate human and robot instructions to perform Golden Gate assembly. To use `Loop DNA Assembly` the user just need to provide the assemblies as a dictionary of parts per role.

```
assemblies = {"promoter":["GVP0008", "GVP0010", "GVP0012",
"GVP0013" "GVP0015", "GVP0016"],
"rbs":"B0034", "cds":"sfGFP", "terminator":"B0015",
"receiver":"Odd_1"}

def run(protocol= protocol_api.ProtocolContext):

    pudu_loop_assembly = Loop_assembly(assemblies=[assemblies])
    pudu_loop_assembly.run(protocol)
```

The user instructions printed in the command line where used to set up the deck and for labeling the assembles `Composites`.

Glyph	Description
	sfGFP
	Terminator B0015

Table 6.6 Parts for `Reporters` for automated build workflow.

```
Parts and reagents in temp_module
{'dd_h2o': 'A1', 't4_dna_ligase': 'A2', 't4_dna_ligase_buffer': 'A3',
'BsaI': 'A4', 'Odd_1': 'A5', 'GVP0012': 'A6', 'GVP0010': 'B1',
'sfGFP': 'B2', 'GVP0013': 'B3', 'GVP0008': 'B4', 'GVP0017': 'B5',
'B0034': 'B6', 'B0015': 'C1', 'GVP0016': 'C2'}

Assembled parts in thermocycler_module
{('GVP0008', 'B0034', 'sfGFP', 'B0015', 'Odd_1'): 'A1',
 ('GVP0010', 'B0034', 'sfGFP', 'B0015', 'Odd_1'): 'A2',
 ('GVP0012', 'B0034', 'sfGFP', 'B0015', 'Odd_1'): 'A3',
 ('GVP0013', 'B0034', 'sfGFP', 'B0015', 'Odd_1'): 'A4',
 ('GVP0016', 'B0034', 'sfGFP', 'B0015', 'Odd_1'): 'A5',
 ('GVP0017', 'B0034', 'sfGFP', 'B0015', 'Odd_1'): 'A6'}
```

The assembled DNA was then used to set up a `Chemical Transformation` to generate human and robot instructions to perform automated transformation. The devices assembled were GVD0011 composed of ('GVP0008', 'B0034', 'sfGFP', 'B0015', 'Odd 1'), GVD0013 composed of ('GVP0010', 'B0034', 'sfGFP', 'B0015', 'Odd 1'), GVD0015 composed of ('GVP0012', 'B0034', 'sfGFP', 'B0015', 'Odd 1'), GVD0016 composed of ('GVP0013', 'B0034', 'sfGFP', 'B0015', 'Odd 1'), GVD0019 composed of ('GVP0016', 'B0034', 'sfGFP', 'B0015', 'Odd 1'), and GVD0020 composed of ('GVP0017', 'B0034', 'sfGFP', 'B0015', 'Odd 1'). To use `Chemical Transformation` the user just need to provide a list of DNAs and competent cells to use.

```
def run(protocol= protocol_api.ProtocolContext):

    pudu_transformation = Chemical_transformation(list_of_dnas=['GVD0011',
    'GVD0013','GVD0015', 'GVD0016', 'GVD0019', 'GVD0020'],
```

```
    competent_cells = 'DH5alpha')
    pudu_transformation.run(protocol)
```

The user instructions printed in the command line where used to set up the deck and for labeling the GMOs.

```
Strain and media tube in temp_mod
{'GVD0011': 'A1', 'GVD0013': 'A2', 'GVD0015': 'A3', 'GVD0016': 'A4',
'GVD0019': 'A5', 'GVD0020': 'A6', 'Competent_cells_tube_0': 'B1',
'Competent_cells_tube_1': 'B2', 'Competent_cells_tube_2': 'B3',
'Media_tube_0': 'B4'}

Genetically modified organisms in thermocycler
{'Competent_cells_tube_0': ['A1', 'A2', 'A3', 'A4', 'A5'],
'Competent_cells_tube_1': ['A6', 'A7', 'A8', 'A9', 'A10'],
'Competent_cells_tube_2': ['A11', 'A12'], 'GVD0011': ['A1', 'A7'],
'GVD0013': ['A2', 'A8'], 'GVD0015': ['A3', 'A9'], 'GVD0016': ['A4', 'A10'],
'GVD0019': ['A5', 'A11'], 'GVD0020': ['A6', 'A12'],
'Media_tube_0': ['A1', 'A2', 'A3', 'A4', 'A5', 'A6',
'A7', 'A8', 'A9', 'A10', 'A11', 'A12']}
```

Transformed bacteria were manually plated into Petri dishes. The replicate was stored as a liquid sample.

Then, GMOs and culture media were used as input for `Simple Plate Setup` to generate human and robot instructions to distribute samples into a 96 well plate and a metadata containing a virtual representation of the plate which is added to the SBOL `Document`. To use `Simple Plate Setup` the user just needs to provide a list of samples that will be placed in 1.5 ml centrifuge tubes. The attribute starting slot allows for this code to be used to plate different sets of sample or to use a 96 well plate multiple times.

```
def run(protocol= protocol_api.ProtocolContext):

    pudu_plate_samples = Plate_samples(
    samples=['s1', 's2','s3', 's4', 's5', 's6'], starting_slot=13)
    pudu_plate_samples.run(protocol)
```

The user instructions printed in the command line where used to set up the deck and for labeling samples.

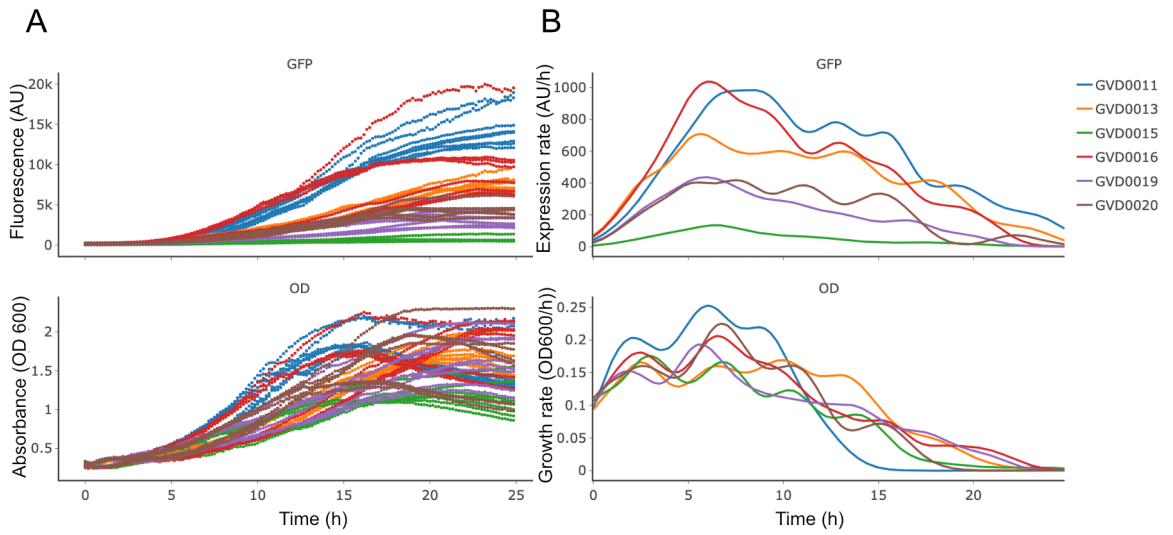


Fig. 6.11 Repressible promoter characterization data and analysis. (A) Fluorescence of GFP and OD raw experimental data. (B) Mean gene expression rate and growth rate of experimental data.

```
Samples in tube rack
{'s1': 'A1', 's2': 'A2', 's3': 'A3', 's4': 'A4', 's5': 'A5', 's6': 'A6'}

Samples in plate
{'s1': ['E4', 'F4', 'G4', 'H4'], 's2': ['E5', 'F5', 'G5', 'H5'],
's3': ['E6', 'F6', 'G6', 'H6'], 's4': ['E7', 'F7', 'G7', 'H7'],
's5': ['E8', 'F8', 'G8', 'H8'], 's6': ['E9', 'F9', 'G9', 'H9']}
```

The 96 well plates were placed into the plate reader to take absorbance and fluorescence measurement each 10 minutes for 25 hours. Then the measurements were exported as an Excel file and used in combination with the metadata to upload it into Flapjack using the BMG parser. Finally, at the learning stage I used Flapjack visualisation and analysis tools to inspect the raw data (Figure 6.11 A) and the data analysed using the inverse characterization method (Figure 6.11 B). I used Flapjack to get the mean expression and biomass of the devices, where the difference in fluorescence is related to the difference in biomass Figure 6.12). Then, I used LOICA to characterise `Source Operators`, closing the cycle (Figure 6.13).

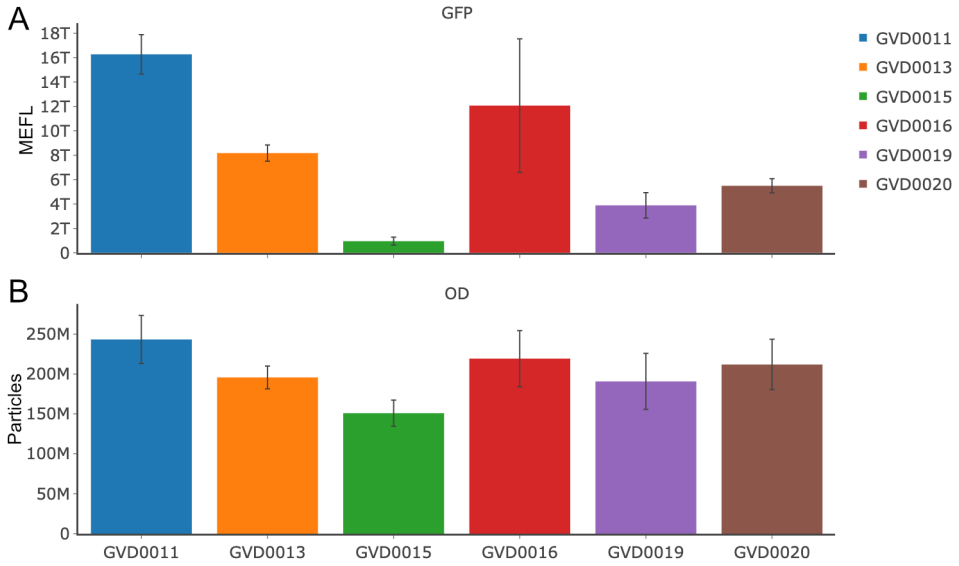


Fig. 6.12 Automated build workflow mean expression and biomass over different `Operators` in the automated build workflow experimental data, bars represent SD, N=3. (A) Mean expression rate in MEFL, calibrated units. (B) Mean biomass in particles, calibrated units.

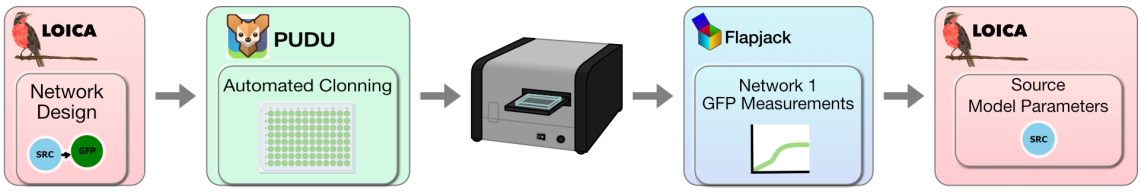


Fig. 6.13 Workflow execution with automated build. LOICA were used to generate simulations from parameters from the literature. Build were simulated using PUDU and performed with the help of lab automation tools. Experimental data were uploaded to Flapjack. Data were visualised and analysed in Flapjack.

6.3 Methods

6.3.1 Software Development

This framework was tested in Python 3, its dependencies are loica, pysbol3, sbol-utilities, pudupy and pyflapjack. Software source code is publicly available under MIT licence at https://github.com/RudgeLab. The packages are distributed through PyPI https://pypi.org. Tests and simulations were performed on an Apple MacBook Pro 15-inch, 2019, with a 2,6 GHz 6-Core Intel Core i7 processor, Radeon Pro 555X 4 GB and Intel UHD Graphics 630 1536 MB GPUs, and 16 GB 2400 MHz DDR4 RAM.

6.3.2 dsDNA design

To order dsDNA, that may be in the form of gBlocks, part cores were flanked with appropriate fusion sites (4 bp), a sequence with SapI recognition sites (30 bp) and optionally sequences to reduce complexity score allowing the synthesis (30 bp). The promoters designed have the following format: pre/extra30-prefix-partcore-suffix-suf/extra30 The CDSs designed have the following format: prefix-partcore-suffix. An outlier is the CDS LacI which has an extra sequence for complexity issues: pre/extra30-prefix-partcore-suffix (see Appendix for more information about the sequences)

6.3.3 Supplemented M9 media preparation

In a 50 ml Falcon tube, dispense: 5ml of 5X M9 media, 0.85 ml of Thiamine, 0.25 ml of glucose 40% w/v, 0.5 ml of cas amino acids solution 10%, 50μl of MgSO4, 2.5 μl CaCl2. Then add 13.575 ml or fill up to 25 ml.

6.3.4 Transformation solution 1

Chemical	**for 1 L**	**for 200 mL**
Potassium chloride (or rubidium chloride)	7.4 g	1.48 g
Potassium acetate	2.95 g	0.59 g
Calcium chloride dihydrate	1.5 g	0.3 g
Glycerol	150 g	30 g, 25 mL

Table 6.7 Transformation solution 1 recipe.

Make up the above (Table 6.7) to a final volume of 950 ml (for the 1L recipe) or 190 ml (for the 200 mL recipe). Adjust pH to 6.4 using acetic acid, dispense into 10 aliquots of 95 mL, and autoclave. Allow the solution to cool and then add 5 mL (per each 95 mL of solution) of 1M Manganese chloride tetrahydrate filter sterilised stock (19.8 g in 100 mL)

6.3.5 Transformation solution 2

Chemical	for 1 L	for 200 mL
Potassium chloride (or rubidium chloride)	0.74 g	0.148 g
Calcium chloride dihydrate	11 g	2.2 g
Glycerol	150 g	30 g, 25 mL

Table 6.8 Transformation solution 2 recipe.

Make up the above (Table 6.8) to a final volume of 980 ml (for the 1L recipe) or 196 ml (for the 200 mL recipe). Dispense into 10 aliquots of 98 mL, and autoclave. Allow the solution to cool and then add 2 mL (per each 98 mL of solution) of 0.5M MOPS buffer (10.47 g in 100 mL). Adjust pH to 6.8 using 5M KOH and filter sterilise.

6.3.6 Competent cell preparation

Scrape a few cells from an *E.coli* glycerol stock (i.e. DH5α) straight from the $-80°C$ freezer onto a marked space on an LB-agar plate (without antibiotics). Streak out three times with a wire or combi loop and grow overnight at $37°C$. Alternatively plate out 50 mL from an aliquot of previously made competent cells onto an LB-agar plate. Inoculate 5 mL LB, in a 50 mL Falcon tube with one colony and incubate overnight at $37°C$ in an orbital shaker at 200 rpm. Inoculate 40 mL of LB in a 250 mL conical flask with 1 mL of the overnight culture. Grow at $37°C$ in an orbital shaker at 200 rpm. Check OD an hour after and until it reaches OD_{600} around 0.4 and 0.5. Transfer the culture to a 50 mL Falcon tube and harvest by centrifugation at 5,000 rpm for 10 minutes at $4°C$. Drain the pellet and re-suspend cells into 8 mL of transformation buffer, TF-1. Place on ice for 15 minutes and then centrifuge at 5,000 rpm for 10 minutes, as above. Thoroughly drain the pellet and re-suspend in 4 mL of transformation buffer, TF-2. Create 100 μL aliquots in 2 mL Eppendorfs and immediately freeze in liquid nitrogen. Store aliquots in a labelled box in a $-80°C$ freezer.

6.3.7 DNA extraction

DNA extraction was performed using the mini prep Monarch kit (NEB).

6.3.8 DNA assembly

All plasmids were constructed by Golden Gate using T4 ligase and its buffer (NEB) and a restriction enzyme either BsaI v2 (NEB) or SapI (NEB) depending on the level of the assembly. Domestications were constructed by Golden Gate, where a part is extracted from a gBlock and inserted in the universal acceptor pSB1C00. This methods were performed mixing 1μl of T4 buffer, 1μl of T4 ligase, 1μl of SapI, 0.5μl of the gBlock at 20 fmol/μl, 0.5 μl of extracted universal acceptor at 20 fmol/μl and 6μl of milli-Q water. In level Odd assemblies, domesticated parts are composed into a TU in an Odd receiver. These methods were performed mixing 1μl of T4 buffer, 1μl of T4 ligase, 1μl of BsaI, 0.5μl of each part in backbone at 20 fmol/μl, 0.5 μl of Odd receiver at 20 fmol/μl and milli-Q water to complete 10μl of reaction volume. In level Even assemblies, TUs are composed into networks in an Even receiver. This methods were performed mixing 1μl of T4 buffer, 1μl of T4 ligase, 1μl of SapI, 0.5μl of each part in backbone at 20 fmol/μl, 0.5 μl of Even receiver at 20 fmol/μl and milli-Q water to complete 10μl of reaction volume.

6.3.9 Transformation

Eppendorf 2 ml tubes with 100 μl of competent bacteria *E. coli* DH5α stored at -80 °C were thawed in ice, then 5 μl of assembly reaction or 2 μl of extracted plasmid were added in the tube with competent bacteria and leave in cold incubation in ice for 30 min. After the cold incubation a heat shock was performed, the tubes were placed in a water bath at 42 °C for 1 minute and returned to the ice for 2 min. Then, 200 μl LB media without antibiotics were added to the tube with competent cells and incubated at 37 °C in an orbital shaker for 1 hour for their recovery. After the recovery incubation, 100 μl were plated in a plate with solid LB and the appropriate antibiotic resistance using 50mg/ml for kanamycin and 100mg/ml for ampicillin.

6.3.10 Automated DNA assembly

Automated DNA assembly implemented the same method used in DNA assembly, but automated using an OT-2. The instructions for deck setup were generated by PUDU as well as the script used to run the protocol.

6.3.11 Automated Transformation

Automated transformation implemented the similar method used in transformation, but automated using an OT-2. The instructions for deck setup were generated by PUDU as well as the script used to run the protocol.

6.3.12 Automated test setup

Automated test setup was used to prepare 96 well plates for experiments. Samples in 2 ml Eppendorf tubes were distributed in 4 wells in a 96 well plate. The instructions for deck setup were generated by PUDU as well as the script used to run the protocol.

6.3.13 Calibration

Calibrations were performed following iGEM technology multicolor fluorescence per particle calibration protocol, described in [11]. The fluorescence calibrants used were fluorescein, sulforamide 101 and cascade blue at an initial concentration of 10 μM. The particle calibrant used was silica microspheres 0.961 μm at 3 10^8 initial particles per well. The calibrants at initial concentration were placed in two wells, fluorescein in A1 and B1, sulforamide 101 in C1 and D1, cascade blue in E1 and F1, and silica microspheres in G1 and H1. From the initial concentrations the calibrants were diluted in half on the well to the right until the column 11. Fluorescein and sulforamide 101 were diluted in PBS, and cascade blue and silica beads were diluted in water. Column 12 was used for water and PBS, 4 wells each. The analysis was performed using the iGEM interlab 2019 template.

6.3.14 Kinetic gene expression and growth assays

Kinetic gene expression and growth assays were made culturing bacteria with the appropriate plasmid. Monoclonal colonies of *E. coli* strain DH5α transformed with plasmid DNA were picked and cultured for 14-16 hours overnight in LB media with 50 µg/ml kanamycin. Overnight cultures were diluted 1000 times in 2 ml tubes with supplemented M9 media. All the tubes were filled with 1998 µl of fresh supplemented M9 media with 50 µg/ml kanamycin and 2 µl of the bacteria liquid culture obtaining a final volume of 2 ml. From each 2 ml tube, 4 samples of 200 µl were distributed, one in each well of a 96 well plate. 4 wells with non-transformed bacteria of the same strain and 4 wells of bacteria transformed with the appropriate plasmid to analyse from the previously prepared 2 mL tubes. Optical density and fluorescence in three channels (RFP, GFP and CFP) were measured approximately every 10

minutes for 25 h in a BMG Clariostar Plus plate reader with Mars software. Each assay was repeated on 3 different days, with 4 replicates on each day.

6.4 Discussion and conclusions

In this chapter three different DBTL workflows were developed, connected and exercised. The simulated framework can be used as a teaching tool and to plan assemblies as the only difference between the simulated and automated framework is simulated data versus real data produced with lab automation.

The communication between the design and build stages is a key feature of the automated framework and this work not just provides tools for design and build that can be connected but also an interface for the easy integration of other software tools to the framework. The connection between the GDA and the liquid handling robot can play a major role in democratizing synthetic biology as the Design stage is less expensive. Laboratories with less resources can dedicate their efforts to design and collaborators or cloud labs can perform the build stage.

The workflow with manual build is what most synthetic biology labs can implement. The main advantage is the connection between the design and learning stages. As well as the easy upload of data and metadata to be used by the learn tool.

The workflow with automated build is the most reproducible and automated. Although lab automation tools are expensive, the liquid handling robot used in this work is in the 10,000 USD range being available to a large number of researchers. The automation of the physical process and meta data capturing at Build stage helps to generate more reproducible experiments, and better quality data and metadata.

The framework is modular and compatible with the SBOL ecosystem, providing value for the community as the tools can be used in other workflows or can be adapted by swapping tools 6.14.

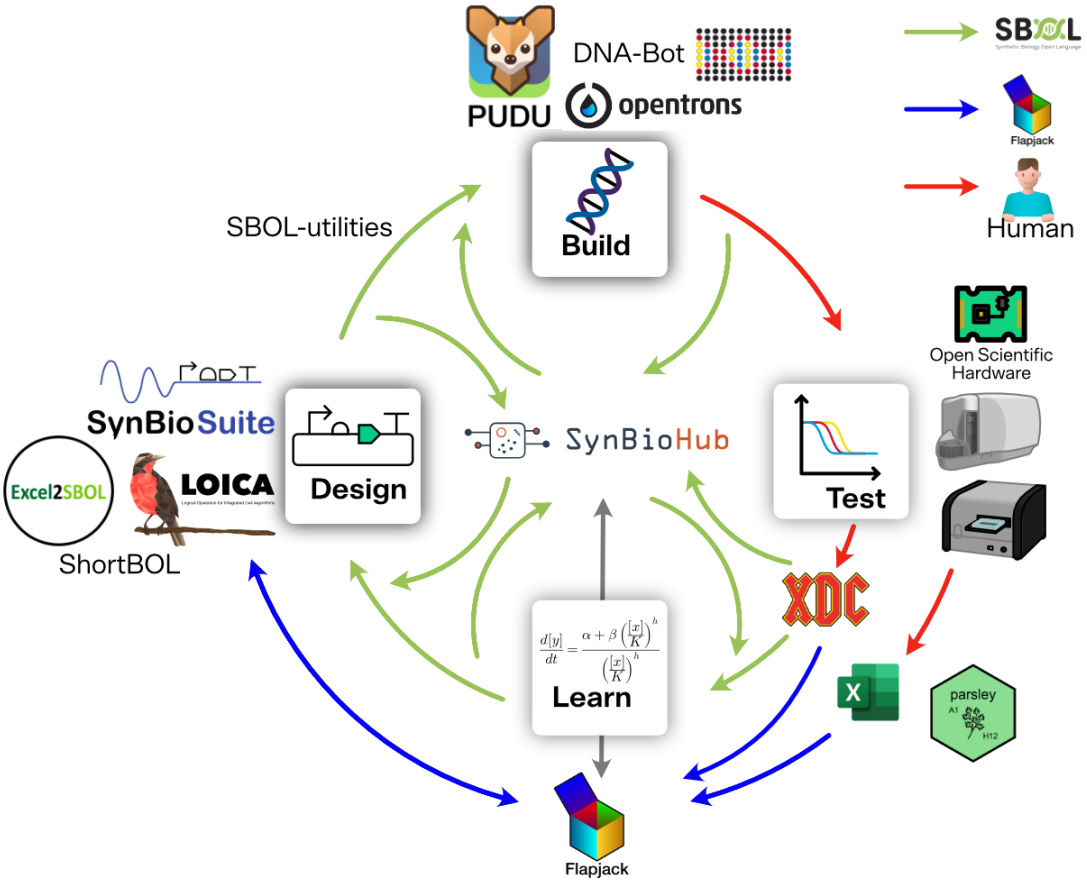


Fig. 6.14 Extended framework diagram. The framework developed in this work can be easily extended with tools connected to standards.

Chapter 7

Conclusion

In this work, in objective A: Create a design tool for synthetic genetic networks. I have created a design abstraction compatible with functional synthetic biology and the Loop hierarchical assembly. I developed LOICA, a Design software tool that uses genetic devices to create genetic networks. Genetic networks were visualised and simulated in different contexts, single cells, bulk cultures and growing colonies. Designs with the desired function were exported as SBOL files. Then, in objective B: Automate the build stage with software and robotics. LOICA SBOL output were formatted as standard build plans using the builders that I contributed to the sbol-utilities package. I developed PUDU, a Build software tool that uses SBOL files to create automated protocols. Standard build plans were used to create assembly instructions as liquid handling robot script and human readable protocol information. Automated and manual protocols were used to insert the assembled DNA into hosts creating genetically modified organisms and to arrange samples containing those organisms into well plates to test its behaviour. Also, calibration plates were made to calibrate plate readers and obtain data in units of fluorescence per particle. Experimental data and related metadata were uploaded to Flapjack. I contributed to Flapjack, a Learn software with data management, visualisation and analysis tools. In objective C: Develop and implement a method for characterising genetic networks to connect Learn and Design stages. I developed a method that uses inverse problems to characterise gene expression and growth rate, and implemented it in Flapjack. LOICA can fetch data and use analysis algorithms from Flapjack to characterise genetic networks to be used in new designs, closing the cycle.

Finally in objective D: Demonstrate DBTL workflows. I exercised three workflows to design and characterise small genetic networks in *Escherichia coli* with simulated, manual and automated build. The workflows developed in this work provides novel teaching and research tools available for different needs. The software tools developed in this work are modular allowing for its combination with other tools from the SBOL ecosystem creating multiple possible workflows. The software is open source and freely available for its use with an MIT license, providing an useful resource for researchers studying gene expression that is already used in labs at the UK, US, Europe and South America.

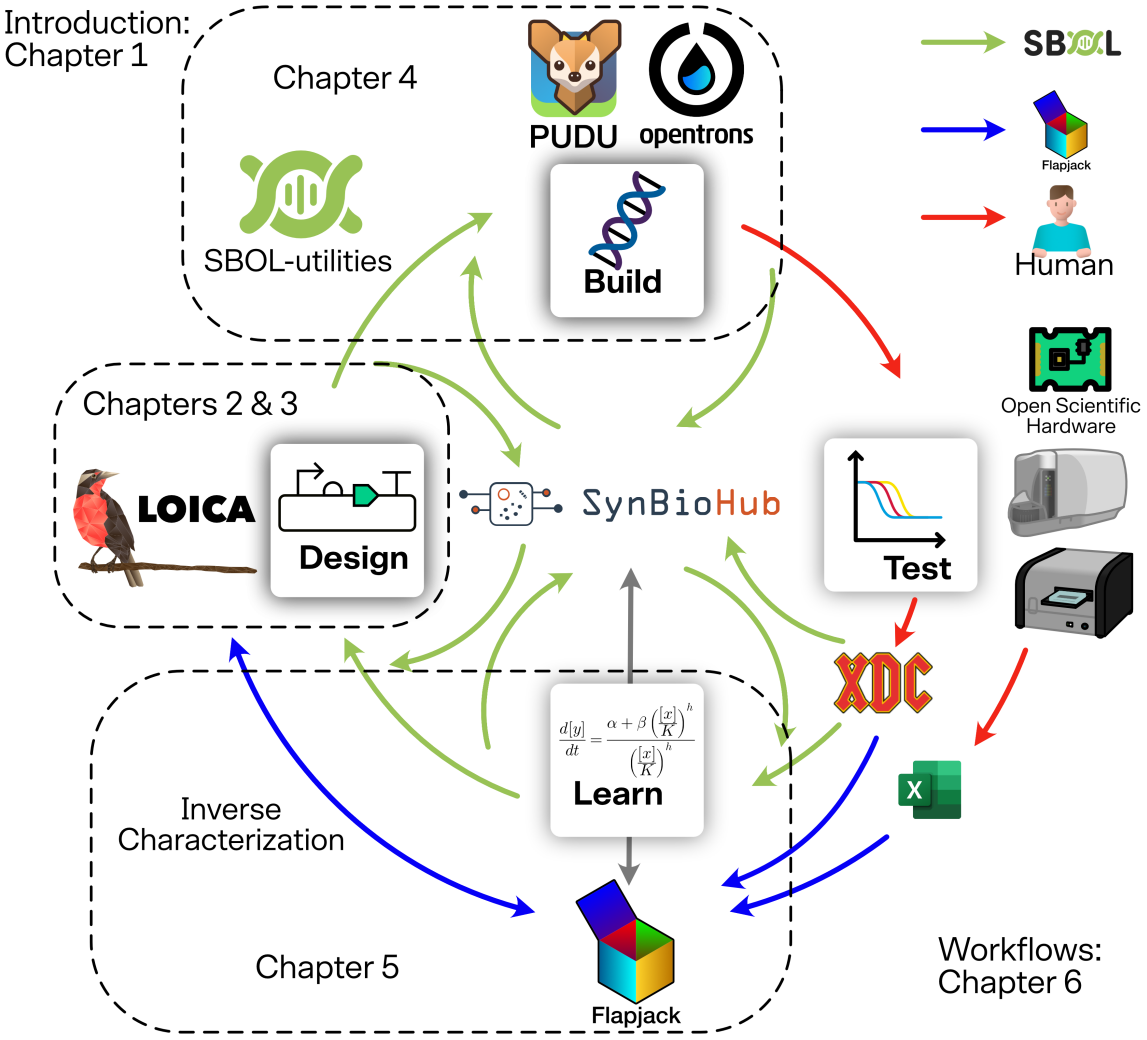


Fig. 7.1 Graphical abstract with software tools.

Chapter 8

Discussion

The workflows developed in this thesis are just an example of the many different workflows that can be made leveraging standards. Using standards inside the different stages of the DBTL cycle and agreeing in standardized interfaces more tools can be added in a plug and play fashion (Figure 6.14). This will enable the addition of more tools by other researchers and its adaptation for its use in different laboratories and for diverse experiments. Although this work is oriented to synthetic biologists and related disciplines with the ability to code, its continuation may produce better user interfaces. A GUI would make the workflows even more accessible widening the impact of these software tools, standards and mathematical methods. Also, programming languages are more and more frequent in the formation of scientists, being Python and R one of the most used tools for data analysis. Scientific breakthroughs like large language models (LLMs) will make coding easier or unnecessary, as code might be produced through natural language.

As future work I would like to exercise this workflow in different laboratories and characterise inverters to run the repressilator generator script with the data of my NOT gates so I would be able to build the repressilators. Functional repressilators can be tested imaging the colony and looking for expression rings [197]. Then, repressilators can be characterised and used as basis to synthesis functions in an analog way to Fourier synthesis. I would like to use optogenetic promoters as `Receivers`, changing the expression level using light intensity and duration for stimulation. It will also be capable of synthesising any other `Operator` profile or response function and will diminish the expenses in chemical inducers or time to load them into the samples. Although these workflows were developed with the bacterial model in mind and tested with *E. coli*, they could be easily adapted to use yeast or other unicellular organisms being the design tool the most challenging to adapt. I would like to extend the workflows to support the work with fungi and plant. With this it will be fun to research beneficial associations between bacteria, fungi and plants to develop plants to make them more resilient, adapted better to new environments in earth, space and mars, that produces more food or air, among others.

Functional synthetic biology and the assembly strategies from this work will enable the use of a software engineering abstraction on synthetic biology. This will allow for the use of genetic parts, devices and systems as objects from a package that can be tested and updated decoupling sequence from function.

The GDA tools developed in this work, LOICA, is also compatible with this abstraction, but still needs a pipeline with more functionalities to update the sequence with a new characterization data. Quality controls and curation are also needed in this pipeline to reduce the risk of using bad characterization data. LOICA holds a great potential to aid in the design of genetic networks and its simulation, as its parameters are experimentally

accesible improving the connection between Test, Learn and Design. In version 1 LOICA included the features to use ODE and SSA for simulations and in version 1.2 incorporated partial differential equations (PDE) for spatio-temporal behaviours. With this LOICA covers the simulation of single cells with SSA, bulk cultures with ODE and growing bacterial colonies with PDE. The simulation of bacterial colonies is still not validated experimentally but previous theoretical work points that it would work in small monolayer colonies [197]. Advances in the incorporation of proteins and production of chemicals have been proposed but needs to be reviewed to be merged in a new version. A major point for this version is to create a clear interface between LOICA and the systems biology markup language (SBML). This connection would allow the simulation of biochemical networks using the plethora of tools already available on the SBML ecosystem to simulate Petri nets [21], flux balance analysis [125] and parameter estimation [154]. Another feature that I would like to develop is the connection between LOICA and CellModeller [144] to provide an easy way to access IBM for genetic network designs. LOICA would provide the design and simulation functionalities and CellModeller would provide a biophysics model that is implemented for bacteria [144] and plants [46]. A recent advantage of connecting to CellModeller is that recently it released a web platform to run simulations in a remote computer server, reducing the hardware needs from the user [132]. With these capabilities spatio-temporal patterns and emergent properties could be better studied. LOICA's programmatic nature make it compatible with HPC, and has been used in this setup allowing for the simulation of many designs in parallel [145], and has been used to simulate parallel SSA for Monte Carlo simulations on complex genetic networks. The designs includes genetic networks to perform phase-based arithmetic such as a majority gate, a full adder and a 4-bit phase-based genetic ripple adder [145]. Moreover, the extension of LOICA and the whole workflow to host organisms of other life kingdoms is an evident need. It's adaptation to be used in fungi and plants are in the development road map as well as its work for cell-free protein expression systems.

The liquid handling robot control tool developed in this work, PUDU, provides a good starting point for automating cloning. PUDU still needs some changes on the Opentrons software to provide support for the creation of SBOL metadata. This would allow an automated production of metadata following best practices improving its quality and removing this burden from the experimentalist. I aim to generate a digital plate, ready to be connected to tools like Flapjack [196] and SynBioHub [113]. Compared to other tools such as PyLabRobot [193] or the Opentrons API that help users to create protocols, PUDU has a set of protocols defined as classes where their arguments are required inputs from the user or small modifications, and PUDU captures metadata in a standardised format. For now just

the `SBOL DNA Assembly` is the only function that can create metadata when the protocols is simulated or run in the OT-2. The only problem to implement PUDU with SBOL metadata capture capabilities is a dependency issue that does not allow for the installation of pySBOL3 and sbol-utilities in the OT-2 Raspberry Pi. Finally, I want to generalise the liquid handling control generating protocols in the Laboratory Open Protocol (LabOP) standard and expand the calibration scripts using absolute protein quantification [36].

To improve Flapjack capabilities its development road map includes the modification of the data objects. For example a sample design object would allow for a simplified way to inform sample metadata, the change on strains to have a many-to-many relationship will allow for the storage of consortia data. Also, the separation of the back-end between data repository and analysis tool would simplify Flapjack's operation. The analysis tools would be organized in a Python package where the characterization method based inverse problems developed in this work can be included.

As synthetic biology aims to design novel genetic networks or circuits from compositions of transcription units, it relies heavily on characterization of the functions of these TUs. Fundamentally, gene expression rates must be reconstructed from noisy measurement data in a range of conditions, including concentrations of inducer chemicals. The function of the network may then be mathematically modelled as in equation 5.1. Methods such as this will enable such approaches for dynamical systems [49, 58] where the dynamic profile of gene expression rates is essential to the operation of the network. I would like to improve the assumption that the expression profile is maintained when the CDS is swapped using deep learning algorithms. This can be analysed using LOICA characterization on a dataset of experimental data of swapped CDSs and LOICA could even be used for data augmentation.

Finally, in this maturation period synthetic biology is defining abstractions, standards and how to decouple the work at different levels. Following the idea of abstraction hierarchy that supports the engineering of integrated genetic system, the decoupling of the work in synthetic biology can be defined. At the sequence level, Sequence engineering tackles DNA structure and function to build genetic parts. For example a bioinformatician can build a promoter and a protein engineer a CDS. Expression engineering knows how these parts interact with each other to create compositions of parts or small genetic devices, tuning expression of one protein by changing the RBS for example. At the organism level, Genetic Network Engineering uses genetic devices to compose genetic regulatory networks, genetic circuits or metabolic pathways to create the "software" for genetically modified organisms. For example, this area will tune protein expression, modify the amplitude and frequency of oscillatory networks and use blue light to control an optogenetic promoter. Host engineering modifies the host organism, its genome and media or cell free reaction to create the "hardware" for

genetically modified organisms. This area will modify or create organisms for determined uses, for example with synthetic genomes, adding new genome insertions or creating new cell-free extracts. At the ecosystem level, Ecosystem engineering creates artificial ecosystems to modify natural ones in a defined geographical location. This area will create consortia, place a set of organisms in a greenhouse for food production or deploy predators to make an ecosystem more robust. Planet engineering combines ecosystems to create or modify the biosphere of a planet, satellite or other cosmic object. This area will be in charge of planet sustainability, colonisation and terraforming. For now planets are the biggest units of life and long term colonisation of other planets will require humans with this expertise. Planet engineering is still in its early stages, with significant challenges to overcome before it becomes a reality. Synthetic biology can play a crucial role in planet engineering by offering innovative solutions to some of these challenges. In bioremediation, synthetic organisms can be engineered to clean up pollutants and toxins, making environments more habitable. In atmospheric modification, engineered microbes could be used to produce gases that alter the atmosphere, potentially making it more suitable for human life. In resource production, synthetic biology can help create sustainable sources of food, fuel, and materials, essential for long-term habitation. In climate control, synthetic organisms could be designed to regulate temperature and weather patterns, aiding in climate stabilization. Although, this work dont tackle these in specific, it looks for automating and advancing synthetic biology to speed up scientific discovery and engineering.

In this work I hope I contributed with my little grain of sand and my vision towards using the engineering method in synthetic biology even though I'm not an engineer myself, I'm just a humble biochemist.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Adhya, S. and Gottesman, M. (1978). Control of transcription termination. *Annual review of biochemistry*, 47(1):967–996.

[3] Aldulijan, I., Beal, J., Billerbeck, S., Bouffard, J., Chambonnier, G., Ntelkis, N., Guerreiro, I., Holub, M., Ross, P., Selvarajah, V., et al. (2023). Functional synthetic biology. *Synthetic Biology*, 8(1):ysad006.

[4] Andersen, J. B., Sternberg, C., Poulsen, L. K., Bjørn, S. P., Givskov, M., and Molin, S. (1998). New unstable variants of green fluorescent protein for studies of transient gene expression in bacteria. *Applied and environmental microbiology*, 64(6):2240–2246.

[5] Arkin, A. (2008). Setting the standard in synthetic biology. *Nature biotechnology*, 26(7):771–774.

[6] Baig, H., Fontanarossa, P., McLaughlin, J., Scott-Brown, J., Vaidyanathan, P., Gorochowski, T., Misirli, G., Beal, J., and Myers, C. (2021). Synthetic biology open language visual (sbol visual) version 3.0. *Journal of integrative bioinformatics*, 18(3):20210013.

[7] Bansal, M., Gatta, G. D., and Bernardo, D. D. (2006). Inference of gene regulatory networks and compound mode of action from time course gene expression profiles. *Bioinformatics*, 22(7):815–822.

[8] Bartley, B. A. (2022). Tyto: A python tool enabling better annotation practices for synthetic biology data-sharing. *ACS Synthetic Biology*, 11(3):1373–1376.

[9] Beal, J., Farny, N. G., Haddock-Angelli, T., Selvarajah, V., Baldwin, G. S., Buckley-Taylor, R., Gershater, M., Kiga, D., Marken, J., Sanchania, V., and Others (2020). Robust estimation of bacterial cell count from optical density. *Communications biology*, 3(1):1–29.

[10] Beal, J., Haddock-Angelli, T., Baldwin, G., Gershater, M., Dwijayanti, A., Storch, M., De Mora, K., Lizarazo, M., Rettberg, R., and with the iGEM Interlab Study Contributors

(2018). Quantification of bacterial fluorescence using independent calibrants. *PloS one*, 13(6):e0199432.

[11] Beal, J., Telmer, C. A., Vignoni, A., Boada, Y., Baldwin, G. S., Hallett, L., Lee, T., Selvarajah, V., Billerbeck, S., Brown, B., et al. (2022). Multicolor plate reader fluorescence calibration. *Synthetic Biology*, 7(1):ysac010.

[12] Beck, J. V. and Woodbury, K. A. (1998). Inverse problems and parameter estimation: integration of measurements and analysis. *Measurement Science and Technology*, 9(6):839.

[13] Bell, S. P. and Dutta, A. (2002). Dna replication in eukaryotic cells. *Annual review of biochemistry*, 71(1):333–374.

[14] Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Brown, C. G., Hall, K. P., Evers, D. J., Barnes, C. L., Bignell, H. R., et al. (2008). Accurate whole human genome sequencing using reversible terminator chemistry. *nature*, 456(7218):53–59.

[15] Bird, J. E., Marles-Wright, J., and Giachino, A. (2022). A user's guide to golden gate cloning methods and standards. *ACS Synthetic Biology*, 11(11):3551–3563.

[16] Bisong, E. (2019). Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 59–64. Springer.

[17] Blakes, J., Twycross, J., Romero-Campero, F. J., and Krasnogor, N. (2011). The infobiotics workbench: an integrated in silico modelling platform for systems and synthetic biology. *Bioinformatics*, 27(23):3323–3324.

[18] Bradshaw, R. A. and Stahl, P. D. (2015). *Encyclopedia of cell biology*. Academic Press.

[19] Branch, M. A., Coleman, T. F., and Li, Y. (1999). A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems. *SIAM Journal on Scientific Computing*, 21(1):1–23.

[20] Brenner, K., You, L., and Arnold, F. H. (2008). Engineering microbial consortia: a new frontier in synthetic biology. *Trends in biotechnology*, 26(9):483–489.

[21] Brinkrolf, C., Ochel, L., and Hofestädt, R. (2021). Vanesa: an open-source hybrid functional petri net modeling and simulation environment in systems biology. *Biosystems*, 210:104531.

[22] Browning, D. F. and Busby, S. J. (2004). The regulation of bacterial transcription initiation. *Nature Reviews Microbiology*, 2(1):57–65.

[23] Buecherl, L., Mitchell, T., Scott-Brown, J., Vaidyanathan, P., Vidal, G., Baig, H., Bartley, B., Beal, J., Crowther, M., Fontanarrosa, P., et al. (2023). Synthetic biology open language (sbol) version 3.1. 0. *Journal of integrative bioinformatics*, 20(1):20220058.

[24] Burn, S. F. (2012). Detection of β-galactosidase activity: X-gal staining. *Kidney Development: Methods and Protocols*, pages 241–250.

[25] Butterworth, S. et al. (1930). On the theory of filter amplifiers. *Wireless Engineer*, 7(6):536–541.

[26] Cai, Y.-M., Carrasco Lopez, J. A., and Patron, N. J. (2020). Phytobricks: manual and automated assembly of constructs for engineering plants. *DNA cloning and assembly: Methods and protocols*, pages 179–199.

[27] Carr, S. B., Beal, J., and Densmore, D. M. (2017). Reducing dna context dependence in bacterial promoters. *PloS one*, 12(4):e0176013.

[28] Caruthers, M. H. (1985). Gene synthesis machines: Dna chemistry and its uses. *Science*, 230(4723):281–285.

[29] Caruthers, M. H. (1991). Chemical synthesis of dna and dna analogs. *Accounts of chemical research*, 24(9):278–284.

[30] Chan, T. E., Stumpf, M. P. H., and Babtie, A. C. (2017). Gene Regulatory Network Inference from Single-Cell Data Using Multivariate Information Measures. *Cell systems*, 5(3):251.

[31] Chargaff, E., Magasanik, B., Vischer, E., Green, C., Doniger, R., and Elson, D. (1950). Nucleotide composition of pentose nucleic acids from yeast and mammalian tissues. *Journal of Biological Chemistry*, 186(1):51–67.

[32] Chen, H., Lei, P., Ji, H., Yang, Q., Peng, B., Ma, J., Fang, Y., Qu, L., Li, H., Wu, W., et al. (2023). Advances in escherichia coli nissle 1917 as a customizable drug delivery system for disease treatment and diagnosis strategies. *Materials Today Bio*, 18:100543.

[33] Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

[34] Cleaves II, H. J. (2010). The origin of the biologically coded amino acids. *Journal of Theoretical biology*, 263(4):490–498.

[35] Cline, J., Braman, J. C., and Hogrefe, H. H. (1996). Pcr fidelity of pfu dna polymerase and other thermostable dna polymerases. *Nucleic acids research*, 24(18):3546–3551.

[36] Csibra, E. and Stan, G.-B. (2022). Absolute protein quantification using fluorescence measurements with fpcountr. *Nature Communications*, 13(1):6600.

[37] Cubitt, A. B., Heim, R., Adams, S. R., Boyd, A. E., Gross, L. A., and Tsien, R. Y. (1995). Understanding, improving and using green fluorescent proteins. *Trends in biochemical sciences*, 20(11):448–455.

[38] Curtiss III, R., Inoue, M., Pereira, D., Hsu, J. C., Alexander, L., and Rock, L. (1977). Construction and use of safer bacterial host strains for recombinant dna research. *Molecular cloning of recombinant DNA*, pages 99–114.

[39] da Costa-Luis, C. O. (2019). tqdm: A fast, extensible progress meter for python and cli. *Journal of Open Source Software*, 4(37):1277.

[40] Danino, T., Mondragón-Palomino, O., Tsimring, L., and Hasty, J. (2010). A synchronized quorum of genetic clocks. *Nature*, 463(7279):326–330.

[41] Davenport, D. and Nicol, J. C. (1955). Luminescence in hydromedusae. *Proceedings of the Royal Society of London. Series B-Biological Sciences*, 144(916):399–411.

[42] De Jong, H., Ranquet, C., Ropers, D., Pinel, C., and Geiselmann, J. (2010). Experimental and computational validation of models of fluorescent and luminescent reporter genes in bacteria. *BMC systems biology*, 4(1):1–17.

[43] De Meeûs, T., Prugnolle, F., and Agnew, P. (2007). Asexual reproduction: genetics and evolutionary aspects. *Cellular and Molecular Life Sciences*, 64(11):1355–1372.

[44] Dever, T. E. and Green, R. (2012). The elongation, termination, and recycling phases of translation in eukaryotes. *Cold Spring Harbor perspectives in biology*, 4(7):a013706.

[45] Dower, W. J., Miller, J. F., and Ragsdale, C. W. (1988). High efficiency transformation of e. coli by high voltage electroporation.

[46] Dupuy, L., Mackenzie, J., Rudge, T., and Haseloff, J. (2008). A system for modelling cell–cell interactions during plant morphogenesis. *Annals of botany*, 101(8):1255–1265.

[47] Durfee, T., Nelson, R., Baldwin, S., Plunkett III, G., Burland, V., Mau, B., Petrosino, J. F., Qin, X., Muzny, D. M., Ayele, M., et al. (2008). The complete genome sequence of escherichia coli dh10b: insights into the biology of a laboratory workhorse. *Journal of bacteriology*, 190(7):2597.

[48] Elahi, E. and Ronaghi, M. (2004). Pyrosequencing: a tool for dna sequencing analysis. *Bacterial Artificial Chromosomes: Volume 1 Library Construction, Physical Mapping, and Sequencing*, pages 211–219.

[49] Elowitz, M. B. and Leibler, S. (2000a). A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338.

[50] Elowitz, M. B. and Leibler, S. (2000b). A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338.

[51] Endy, D. (2005). Foundations for engineering biology. *Nature*, 438(7067):449–453.

[52] Engler, C., Kandzia, R., and Marillonnet, S. (2008). A one pot, one step, precision cloning method with high throughput capability. *PloS one*, 3(11):e3647.

[53] Estrem, S. T., Ross, W., Gaal, T., Chen, Z. S., Niu, W., Ebright, R. H., and Gourse, R. L. (1999). Bacterial promoter architecture: subsite structure of up elements and interactions with the carboxy-terminal domain of the rna polymerase α subunit. *Genes & development*, 13(16):2134–2147.

[54] Filo, M., Kumar, S., and Khammash, M. (2022). A hierarchy of biomolecular proportional-integral-derivative feedback controllers for robust perfect adaptation and dynamic performance. *Nature communications*, 13(1):2119.

[55] Fontanarrosa, P., Doosthosseini, H., Borujeni, A. E., Dorfan, Y., Voigt, C. A., and Myers, C. (2020). Genetic circuit dynamics: hazard and glitch analysis. *ACS Synthetic Biology*, 9(9):2324–2338.

[56] Gaal, T., Ross, W., Estrem, S. T., Nguyen, L. H., Burgess, R. R., and Gourse, R. L. (2001). Promoter recognition and discrimination by es rna polymerase. *Molecular Microbiology*, 42(4):939–954.

[57] Gapp, K. and Miska, E. A. (2016). trna fragments: novel players in intergenerational inheritance. *Cell Research*, 26(4):395–396.

[58] Gardner, T. S., Cantor, C. R., and Collins, J. J. (2000). Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342.

[59] Gibson, D. G., Young, L., Chuang, R.-Y., Venter, J. C., Hutchison III, C. A., and Smith, H. O. (2009). Enzymatic assembly of dna molecules up to several hundred kilobases. *Nature methods*, 6(5):343–345.

[60] Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of computational physics*, 22(4):403–434.

[61] Gillespie, D. T. (2007). Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.*, 58:35–55.

[62] Gold, L., Pribnow, D., Schneider, T., Shinedling, S., Singer, B. S., and Stormo, G. (1981). Translational initiation in prokaryotes. *Annual Reviews in Microbiology*, 35(1):365–403.

[63] Gompertz, B. (1825). On the Nature of the Function Expressive of the Law of Human Mortality, and on a New Mode of Determining the Value of Life Contingencies. *Philosophical Transactions of the Royal Society*, 115:513–583.

[64] Goodwin, S., McPherson, J. D., and McCombie, W. R. (2016). Coming of age: ten years of next-generation sequencing technologies. *Nature reviews genetics*, 17(6):333–351.

[65] Green, A. A., Kim, J., Ma, D., Silver, P. A., Collins, J. J., and Yin, P. (2017). Complex cellular logic computation using ribocomputing devices. *Nature*, 548(7665):117–121.

[66] Griffith, F. (1928). The significance of pneumococcal types. *Epidemiology & Infection*, 27(2):113–159.

[67] Grozinger, L., Amos, M., Gorochowski, T. E., Carbonell, P., Oyarzún, D. A., Stoof, R., Fellermann, H., Zuliani, P., Tas, H., and Goñi-Moreno, A. (2019). Pathways to cellular supremacy in biocomputing. *Nature communications*, 10(1):5250.

[68] Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[69] Hansen, P. C. and O'Leary, D. P. (1993). The use of the L-curve in the regularization of discrete ill-posed problems. *SIAM journal on scientific computing*, 14(6):1487–1503.

[70] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[71] Haseloff, J. and Amos, B. (1995). Gfp in plants.

[72] Hatch, B., Meng, L., Mante, J., McLaughlin, J. A., Scott-Brown, J., and Myers, C. J. (2021). Visbol2—improving web-based visualization for synthetic biology designs. *ACS Synthetic Biology*, 10(8):2111–2115.

[73] Haugen, S. P., Berkmen, M. B., Ross, W., Gaal, T., Ward, C., and Gourse, R. L. (2006). rrna promoter regulation by nonoptimal binding of σ region 1.2: an additional recognition element for rna polymerase. *Cell*, 125(6):1069–1082.

[74] Hérisson, J., Duigou, T., Du Lac, M., Bazi-Kabbaj, K., Sabeti Azad, M., Buldum, G., Telle, O., El Moubayed, Y., Carbonell, P., Swainston, N., et al. (2022). The automated galaxy-synbiocad pipeline for synthetic biology design and engineering. *Nature Communications*, 13(1):5082.

[75] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

[76] IGEM (2006). Registry of Standard Biological Parts.

[77] Inouye, S. and Tsuji, F. I. (1994). Aequorea green fluorescent protein: Expression of the gene and fluorescence characteristics of the recombinant protein. *FEBS letters*, 341(2-3):277–280.

[78] Iverson, S. V., Haddock, T. L., Beal, J., and Densmore, D. M. (2016). Cidar moclo: improved moclo assembly standard and new e. coli part library enable rapid combinatorial design for synthetic and traditional biology. *ACS synthetic biology*, 5(1):99–103.

[79] Jacob, F. and Monod, J. (1961). Genetic regulatory mechanisms in the synthesis of proteins. *Journal of molecular biology*, 3(3):318–356.

[80] Jain, M., Koren, S., Miga, K. H., Quick, J., Rand, A. C., Sasani, T. A., Tyson, J. R., Beggs, A. D., Dilthey, A. T., Fiddes, I. T., et al. (2018). Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4):338–345.

[81] Jakel, A. C., Aufinger, L., and Simmel, F. C. (2022). Steady-state operation of a cell-free genetic band-detection circuit. *ACS Synthetic Biology*, 11(10):3273–3284.

[82] Jones, T. S., Oliveira, S. M., Myers, C. J., Voigt, C. A., and Densmore, D. (2022). Genetic circuit design automation with cello 2.0. *Nature protocols*, 17(4):1097–1113.

[83] Josaitis, C. A., Gaal, T., and Gourse, R. L. (1995). Stringent control and growth-rate-dependent control have nonidentical promoter sequence requirements. *Proceedings of the National Academy of Sciences*, 92(4):1117–1121.

[84] Kazemi, A. S., NASRESFAHANI, M. H., HOSSEINI, S. M., Moulavi, F., Hajian, M., Forouzanfar, M., Abedi, P., Memar, M., REZAZADEH, V. M., Gourabi, H., et al. (2008). Royana: successful experience in cloning the sheep.

[85] Keilty, S. and Rosenberg, M. (1987). Constitutive function of a positively regulated promoter reveals new sequences essential for activity. *Journal of Biological Chemistry*, 262(13):6389–6395.

[86] Kelly, J. R., Rubin, A. J., Davis, J. H., Ajo-Franklin, C. M., Cumbers, J., Czar, M. J., de Mora, K., Glieberman, A. L., Monie, D. D., and Endy, D. (2009). Measuring the activity of BioBrick promoters using an in vivo reference standard. *Journal of Biological Engineering*, 3(1):4.

[87] Khammash, M. H. (2021). Perfect adaptation in biology. *Cell Systems*, 12(6):509–521.

[88] Khorana, H. G., Agarwal, K., Besmer, P., Büchi, H., Caruthers, M., Cashion, P., Fridkin, M., Jay, E., Kleppe, K., Kleppe, R., et al. (1976). Total synthesis of the structural gene for the precursor of a tyrosine suppressor transfer rna from escherichia coli. 1. general introduction. *Journal of Biological Chemistry*, 251(3):565–570.

[89] Klebe, R. J., Harriss, J. V., Sharp, Z. D., and Douglas, M. G. (1983). A general method for polyethylene-glycol-induced genetic transformation of bacteria and yeast. *Gene*, 25(2-3):333–341.

[90] Kleppe, K., Ohtsuka, E., Kleppe, R., Molineux, I., and Khorana, H. (1971). Studies on polynucleotides: Xcvi. repair replication of short synthetic dna's as catalyzed by dna polymerases. *Journal of molecular biology*, 56(2):341–361.

[91] Klumpp, S. and Hwa, T. (2008). Growth-rate-dependent partitioning of RNA polymerases in bacteria. *Proceedings of the National Academy of Sciences of the United States of America*, 105(51):20245–20250.

[92] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., et al. (2016). *Jupyter Notebooks-a publishing format for reproducible computational workflows.*, volume 2016.

[93] Knight, T. (2003). Idempotent vector design for standard assembly of biobricks.

[94] Konur, S., Gheorghe, M., and Krasnogor, N. (2023). Verifiable biology. *Journal of the Royal Society Interface*, 20(202):20230019.

[95] Konur, S., Mierla, L., Fellermann, H., Ladroue, C., Brown, B., Wipat, A., Twycross, J., Dun, B. P., Kalvala, S., Gheorghe, M., et al. (2021). Toward full-stack in silico synthetic biology: integrating model specification, simulation, verification, and biological compilation. *ACS Synthetic Biology*, 10(8):1931–1945.

[96] Kormanec, J. (2022). Bacterial regulatory proteins.

[97] Kornberg, A., Lehman, I., Bessman, M. J., and Simms, E. (1956). Enzymic synthesis of deoxyribonucleic acid. *Biochimica et Biophysica Acta*, 21(1):197–198.

[98] Korzheva, N., Mustaev, A., Kozlov, M., Malhotra, A., Nikiforov, V., Goldfarb, A., and Darst, S. A. (2000). A structural model of transcription elongation. *Science*, 289(5479):619–625.

[99] LaFleur, T. L., Hossain, A., and Salis, H. M. (2022). Automated model-predictive design of synthetic promoters to control transcriptional profiles in bacteria. *Nature communications*, 13(1):5159.

[100] Lehman, I., Zimmerman, S. B., Adler, J., Bessman, M. J., Simms, E., and Kornberg, A. (1958). Enzymatic synthesis of deoxyribonucleic acid. v. chemical composition of enzymatically synthesized deoxyribonucleic acid. *Proceedings of the National Academy of Sciences*, 44(12):1191–1196.

[101] Liljeruhm, J., Funk, S. K., Tietscher, S., Edlund, A. D., Jamal, S., Wistrand-Yuen, P., Dyrhage, K., Gynnå, A., Ivermark, K., Lövgren, J., et al. (2018). Engineering a palette of eukaryotic chromoproteins for bacterial synthetic biology. *Journal of biological engineering*, 12:1–10.

[102] Liu, L., Li, Y., Li, S., Hu, N., He, Y., Pong, R., Lin, D., Lu, L., and Law, M. (2012). Comparison of next-generation sequencing systems. *J Biomed Biotechnol*, 2012(251364):251364.

[103] Losick, R. and Pero, J. (1981). Cascades of sigma factors. *Cell*, 25(3):582–584.

[104] Lou, C., Stanton, B., Chen, Y.-J., Munsky, B., and Voigt, C. A. (2012). Ribozyme-based insulator parts buffer synthetic circuits from genetic context. *Nature biotechnology*, 30(11):1137–1142.

[105] Magasanik, B., Vischer, E., Doniger, R., Elson, D., and Chargaff, E. (1950). The separation and estimation of ribonucleotides in minute quantities. *Journal of Biological Chemistry*, 186(1):37–50.

[106] Majeed, A. H., Zainal, M. S. B., Alkaldy, E., and Nor, D. M. (2020). Full adder circuit design with novel lower complexity xor gate in qca technology. *Transactions on Electrical and Electronic Materials*, 21:198–207.

[107] Mante, J. and Myers, C. J. (2023). Advancing reuse of genetic parts: progress and remaining challenges. *nature communications*, 14(1):2953.

[108] Marians, K. J. (1992). Prokaryotic dna replication. *Annual review of biochemistry*, 61(1):673–715.

[109] Maxam, A. M. and Gilbert, W. (1977). A new method for sequencing dna. *Proceedings of the National Academy of Sciences*, 74(2):560–564.

[110] Mayer, G. (2009). The chemical biology of aptamers. *Angewandte Chemie International Edition*, 48(15):2672–2689.

[111] McKinney, W. (2010). Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56–61.

[112] McLaughlin, J. A., Beal, J., Mısırlı, G., Grünberg, R., Bartley, B. A., Scott-Brown, J., Vaidyanathan, P., Fontanarrosa, P., Oberortner, E., Wipat, A., et al. (2020). The synthetic biology open language (sbol) version 3: Simplified data exchange for bioengineering. *Frontiers in Bioengineering and Biotechnology*, 8.

[113] McLaughlin, J. A., Myers, C. J., Zundel, Z., Mısırlı, G., Zhang, M., Ofiteru, I. D., Goni-Moreno, A., and Wipat, A. (2018). Synbiohub: a standards-enabled design repository for synthetic biology. *ACS synthetic biology*, 7(2):682–688.

[114] Mejía-Almonte, C., Busby, S. J., Wade, J. T., van Helden, J., Arkin, A. P., Stormo, G. D., Eilbeck, K., Palsson, B. O., Galagan, J. E., and Collado-Vides, J. (2020). Redefining fundamental concepts of transcription initiation in bacteria. *Nature Reviews Genetics*, 21(11):699–714.

[115] Mello, C. and Fire, A. (1995). Dna transformation. *Methods in cell biology*, 48:451–482.

[116] Mishra, D., Rivera, P. M., Lin, A., Del Vecchio, D., and Weiss, R. (2014). A load driver device for engineering modularity in biological networks. *Nature biotechnology*, 32(12):1268.

[117] Mitchell, T., Beal, J., and Bartley, B. (2022). pysbol3: Sbol3 for python programmers. *ACS Synthetic Biology*, 11(7):2523–2526.

[118] Mullis, K. B. and Faloona, F. A. (1987). [21] specific synthesis of dna in vitro via a polymerase-catalyzed chain reaction. In *Methods in enzymology*, volume 155, pages 335–350. Elsevier.

[119] Nakamura, Y., Ito, K., and Isaksson, L. A. (1996). Emerging understanding of translation termination. *Cell*, 87(2):147–150.

[120] Naylor, J., Fellermann, H., Ding, Y., Mohammed, W. K., Jakubovics, N. S., Mukherjee, J., Biggs, C. A., Wright, P. C., and Krasnogor, N. (2017). Simbiotics: a multiscale integrative platform for 3d modeling of bacterial populations. *ACS Synthetic Biology*, 6(7):1194–1210.

[121] Naylor, L. H. (1999). Reporter gene technology: the future looks bright. *Biochemical Pharmacology*, 58(5):749–757.

[122] Nguyen, T., Roehner, N., Zundel, Z., and Myers, C. J. (2016). A converter from the systems biology markup language to the synthetic biology open language. *ACS synthetic biology*, 5(6):479–486.

[123] Nielsen, A. A., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., and Voigt, C. A. (2016). Genetic circuit design automation. *Science*, 352(6281).

[124] Nielsen, J. and Keasling, J. D. (2011). Synergies between synthetic biology and metabolic engineering. *Nature biotechnology*, 29(8):693–695.

[125] Olivier, B. G. and Bergmann, F. T. (2018). Sbml level 3 package: flux balance constraints version 2. *Journal of integrative bioinformatics*, 15(1):20170082.

[126] Pardee, A. B., Jacob, F., and Monod, J. (1959). The genetic control and cytoplasmic expression of "inducibility" in the synthesis of β-galactosidase by e. coli. *Journal of Molecular Biology*, 1(2):165–178.

[127] Pardee, K., Slomovic, S., Nguyen, P. Q., Lee, J. W., Donghia, N., Burrill, D., Ferrante, T., McSorley, F. R., Furuta, Y., Vernet, A., et al. (2016). Portable, on-demand biomolecular manufacturing. *Cell*, 167(1):248–259.

[128] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

[129] Peccoud, J., Anderson, J. C., Chandran, D., Densmore, D., Galdzicki, M., Lux, M. W., Rodriguez, C. A., Stan, G.-B., and Sauro, H. M. (2011). Essential information for synthetic dna sequences. *Nature Biotechnology*, 29(1):22–22.

[130] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[131] Peña, G. A. V., Vidal-Céspedes, C. I., et al. (2021). Accurate reconstruction of dynamic gene expression and growth rate profiles from noisy measurements. *bioRxiv*.

[132] Philippou, J., Yanez Feliu, G., and Rudge, T. J. (2024). Webcm: A web-based platform for multiuser individual-based modeling of multicellular microbial populations and communities. *ACS Synthetic Biology*.

[133] Plotly Technologies Inc. (2015). Collaborative data science.

[134] Pollak, B., Cerda, A., Delmans, M., Álamos, S., Moyano, T., West, A., Gutiérrez, R. A., Patron, N. J., Federici, F., and Haseloff, J. (2019). Loop assembly: a simple and open system for recursive fabrication of dna circuits. *New Phytologist*, 222(1):628–640.

[135] Pollak, B., Matute, T., Nuñez, I., Cerda, A., Lopez, C., Vargas, V., Kan, A., Bielinski, V., von Dassow, P., Dupont, C. L., et al. (2020). Universal loop assembly: open, efficient and cross-kingdom dna fabrication. *Synthetic Biology*, 5(1):ysaa001.

[136] Potvin-Trottier, L., Lord, N. D., Vinnicombe, G., and Paulsson, J. (2016). Synchronous long-term oscillations in a synthetic gene circuit. *Nature*, 538(7626):514–517.

[137] Prasher, D. C., Eckenrode, V. K., Ward, W. W., Prendergast, F. G., and Cormier, M. J. (1992). Primary structure of the aequorea victoria green-fluorescent protein. *Gene*, 111(2):229–233.

[138] Rajkovic, A. and Ibba, M. (2017). Elongation factor p and the control of translation elongation. *Annual Review of Microbiology*, 71:117–131.

[139] Rechavi, O., Minevich, G., and Hobert, O. (2011). Transgenerational inheritance of an acquired small rna-based antiviral response in c. elegans. *Cell*, 147(6):1248–1256.

[140] Ritchie, M. D., Holzinger, E. R., Li, R., Pendergrass, S. A., and Kim, D. (2015). Methods of integrating data to uncover genotype–phenotype interactions. *Nature Reviews Genetics*, 16(2):85–97.

[141] Roberts, R. J., Carneiro, M. O., and Schatz, M. C. (2013). The advantages of smrt sequencing. *Genome biology*, 14(6):1–4.

[142] Rodnina, M. V. (2018). Translation in prokaryotes. *Cold Spring Harbor perspectives in biology*, 10(9):a032664.

[143] Rudge, T. J., Brown, J. R., Federici, F., Dalchau, N., Phillips, A., Ajioka, J. W., and Haseloff, J. (2016). Characterization of intrinsic properties of promoters. *ACS synthetic biology*, 5(1):89–98.

[144] Rudge, T. J., Steiner, P. J., Phillips, A., and Haseloff, J. (2012). Computational modeling of synthetic microbial biofilms. *ACS synthetic biology*, 1(8):345–352.

[145] Rudge, T. J. and Vidal, G. (2022). Phase-based genetic logic circuits. *bioRxiv*, pages 2022–12.

[146] Ruff, E. F., Record Jr, M. T., and Artsimovitch, I. (2015). Initial events in bacterial transcription initiation. *Biomolecules*, 5(2):1035–1062.

[147] Salis, H. M. (2011). The ribosome binding site calculator. In *Methods in enzymology*, volume 498, pages 19–42. Elsevier.

[148] Sambrook, J., Fritsch, E. F., Maniatis, T., et al. (1989). *Molecular cloning: a laboratory manual*. Number Ed. 2. Cold spring harbor laboratory press.

[149] Samineni, S. P., Vidal, G., Vitalis, C., Feliú, G. Y., Rudge, T. J., Myers, C. J., and Mante, J. (2023). Experimental data connector (xdc): Integrating the capture of experimental data and metadata using standard formats and digital repositories. *ACS Synthetic Biology*, 12(4):1364–1370.

[150] Sanassy, D., Widera, P., and Krasnogor, N. (2015). Meta-stochastic simulation of biochemical models for systems and synthetic biology. *ACS synthetic biology*, 4(1):39–47.

[151] Sanger, F., Nicklen, S., and Coulson, A. R. (1977). Dna sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467.

[152] Sarrion-Perdigones, A., Falconi, E. E., Zandalinas, S. I., Juárez, P., Fernández-del Carmen, A., Granell, A., and Orzaez, D. (2011). Goldenbraid: an iterative cloning system for standardized assembly of reusable genetic modules. *PloS one*, 6(7):e21622.

[153] Savitzky, A. and Golay, M. J. E. (1964). Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639.

[154] Schälte, Y., Fröhlich, F., Jost, P. J., Vanhoefer, J., Pathirana, D., Stapor, P., Lakrisenko, P., Wang, D., Raimúndez, E., Merkt, S., et al. (2023). pypesto: a modular and scalable tool for parameter estimation for dynamic models. *Bioinformatics*, 39(11):btad711.

[155] Scott, M., Gunderson, C. W., Mateescu, E. M., Zhang, Z., and Hwa, T. (2010). Interdependence of cell growth and gene expression: origins and consequences. *Science*, 330(6007):1099–1102.

[156] Sents, Z., Stoughton, T. E., Buecherl, L., Thomas, P. J., Fontanarrosa, P., and Myers, C. J. (2023). Synbiosuite: A tool for improving the workflow for genetic design and modeling. *ACS Synthetic Biology*, 12(3):892–897.

[157] Shaner, N. C., Steinbach, P. A., and Tsien, R. Y. (2005). A guide to choosing fluorescent proteins. *Nature methods*, 2(12):905–909.

[158] Shetty, R., Lizarazo, M., Rettberg, R., and Knight, T. F. (2011). Assembly of biobrick standard biological parts using three antibiotic assembly. In *Methods in enzymology*, volume 498, pages 311–326. Elsevier.

[159] Shin, J., Zhang, S., Der, B. S., Nielsen, A. A., and Voigt, C. A. (2020). Programming escherichia coli to function as a digital display. *Molecular systems biology*, 16(3):e9401.

[160] Silva, M. A. M., Matute, T., Nuñez, I., Valdes, A., Ruiz, C. A., Peña, G. A. V., Federici, F., and Rudge, T. J. (2019). Phase space characterization for gene circuit design. *BioRxiv*, page 590299.

[161] Smolke, C. D. (2009). Building outside of the box: igem and the biobricks foundation. *Nature biotechnology*, 27(12):1099–1102.

[162] Solomatine, D., See, L. M., and Abrahart, R. (2008). Data-driven modelling: concepts, approaches and experiences. *Practical hydroinformatics: Computational intelligence and technological developments in water applications*, pages 17–30.

[163] Song, K.-M., Lee, S., and Ban, C. (2012). Aptamers and their biological applications. *Sensors*, 12(1):612–631.

[164] Stoica, P. and Babu, P. (2012). Spice and likes: Two hyperparameter-free methods for sparse-parameter estimation. *Signal Processing*, 92(7):1580–1590.

[165] Storch, M., Casini, A., Mackrow, B., Fleming, T., Trewhitt, H., Ellis, T., and Baldwin, G. S. (2015). Basic: a new biopart assembly standard for idempotent cloning provides accurate, single-tier dna assembly for synthetic biology. *ACS synthetic biology*, 4(7):781–787.

[166] Storch, M., Haines, M. C., and Baldwin, G. S. (2020). Dna-bot: a low-cost, automated dna assembly platform for synthetic biology. *Synthetic Biology*, 5(1):ysaa010.

[167] Taketo, A. (1972). Sensitivity of escherichia coli to viral nucleic acid: v. competence of calcium-treated cells. *The Journal of Biochemistry*, 72(4):973–979.

[168] Tamsir, A., Tabor, J. J., and Voigt, C. A. (2011). Robust multicellular computing using genetically encoded nor gates and chemical 'wires'. *Nature*, 469(7329):212–215.

[169] Tas, H., Grozinger, L., Stoof, R., de Lorenzo, V., and Goñi-Moreno, Á. (2021). Contextual dependencies expand the re-usability of genetic inverters. *Nature communications*, 12(1):1–9.

[170] Terry, L., Earl, J., Thayer, S., Bridge, S., and Myers, C. J. (2021). Sbolcanvas: A visual editor for genetic designs. *ACS Synthetic Biology*.

[171] Thomas, D. E. and Moorby, P. R. (1990). *The Verilog® Hardware Description Language*.

[172] Toni, T., Welch, D., Strelkowa, N., Ipsen, A., and Stumpf, M. P. H. (2009). Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface*, 6(31):187–202.

[173] Torella, J. P., Lienert, F., Boehm, C. R., Chen, J.-H., Way, J. C., and Silver, P. A. (2014). Unique nucleotide sequence–guided assembly of repetitive DNA parts for synthetic biology applications. *Nature protocols*, 9(9):2075–2089.

[174] Travers, A. and Muskhelishvili, G. (2015). Dna structure and function. *The FEBS journal*, 282(12):2279–2295.

[175] TURING, A. M. (1950). I.—computing machinery and intelligence. *Mind*, LIX(236):433–460.

[176] Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[177] Vidal, G., Vidal-Céspedes, C., Muñoz Silva, M., Castillo-Passi, C., Yáñez Feliú, G., Federici, F., and Rudge, T. J. (2022a). Accurate characterization of dynamic microbial gene expression and growth rate profiles. *Synthetic Biology*, 7(1):ysac020.

[178] Vidal, G., Vidal-Céspedes, C., and Rudge, T. J. (2022b). Loica: Integrating models with data for genetic network design automation. *ACS Synthetic Biology*.

[179] Vidal, G., Vitalis, C., Matúte, T., Núñez, I., Federici, F., and Rudge, T. J. (2024). Genetic network design automation with loica. In *Synthetic Biology: Methods and Protocols*, pages 393–412. Springer.

[180] Vilanova, C. and Porcar, M. (2014). igem 2.0—refoundations for engineering biology. *Nature Biotechnology*, 32(5):420–424.

[181] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, , Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors, S. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272.

[182] Vischer, E. and Chargaff, E. (1948). The separation and quantitative estimation of purines and pyrimidines in minute amounts. *Journal of Biological Chemistry*, 176(2):703–714.

[183] Vitalis, C., Feliú, G. Y., Vidal, G., Silva, M. M., Matúte, T., Núñez, I., Federici, F., and Rudge, T. J. (2024). Flapjack: Data management and analysis for genetic circuit characterization. In *Synthetic Biology: Methods and Protocols*, pages 413–434. Springer.

[184] Wang, X. and Harrison, A. (2021). A general principle for spontaneous genetic symmetry breaking and pattern formation within cell populations. *Journal of Theoretical Biology*, 526:110809.

[185] Wang, Y., Wang, H., Wei, L., Li, S., Liu, L., and Wang, X. (2020). Synthetic promoter design in escherichia coli based on a deep generative network. *Nucleic Acids Research*, 48(12):6403–6412.

[186] Wang, Y., Zhao, Y., Bollas, A., Wang, Y., and Au, K. F. (2021). Nanopore sequencing technology, bioinformatics and applications. *Nature biotechnology*, 39(11):1348–1365.

[187] Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

[188] Watanabe, L., Nguyen, T., Zhang, M., Zundel, Z., Zhang, Z., Madsen, C., Roehner, N., and Myers, C. (2018). ibiosim 3: a tool for model-based genetic circuit design. *ACS synthetic biology*, 8(7):1560–1563.

[189] Watson, J. D. and Crick, F. H. (1953). Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738.

[190] Weber, E., Engler, C., Gruetzner, R., Werner, S., and Marillonnet, S. (2011). A modular cloning system for standardized assembly of multigene constructs. *PloS one*, 6(2):e16765.

[191] Westfall, P. J. and Gardner, T. S. (2011). Industrial fermentation of renewable diesel fuels. *Current opinion in biotechnology*, 22(3):344–350.

[192] Wickham, H. (2014). Tidy data. *Journal of statistical software*, 59:1–23.

[193] Wierenga, R. P., Golas, S. M., Ho, W., Coley, C. W., and Esvelt, K. M. (2023). Pylabrobot: An open-source, hardware-agnostic interface for liquid-handling robots and accessories. *Device*, 1(4).

[194] Wong, A., Wang, H., Poh, C. L., and Kitney, R. I. (2015a). Layering genetic circuits to build a single cell, bacterial half adder. *BMC biology*, 13(1):1–16.

[195] Wong, S. P., Argyros, O., and Harbottle, R. P. (2015b). Sustained expression from dna vectors. *Advances in genetics*, 89:113–152.

[196] Yáñez Feliú, G., Earle Gómez, B., Codoceo Berrocal, V., Muñoz Silva, M., Nuñez, I. N., Matute, T. F., Arce Medina, A., Vidal, G., Vidal Céspedes, C., Dahlin, J., et al. (2020a). Flapjack: Data management and analysis for genetic circuit characterization. *ACS Synthetic Biology*, 10(1):183–191.

[197] Yáñez Feliú, G., Vidal, G., Muñoz Silva, M., and Rudge, T. J. (2020b). Novel tunable spatio-temporal patterns from a simple genetic oscillator circuit. *Frontiers in bioengineering and biotechnology*, 8:893.

[198] Yeung, E., Dy, A. J., Martin, K. B., Ng, A. H., Del Vecchio, D., Beck, J. L., Collins, J. J., and Murray, R. M. (2017). Biophysical constraints arising from compositional context in synthetic gene networks. *Cell systems*, 5(1):11–24.

[199] Yokobayashi, Y., Weiss, R., and Arnold, F. H. (2002). Directed evolution of a genetic circuit. *Proceedings of the National Academy of Sciences*, 99(26):16587–16591.

[200] Zhang, Y., Feng, Y., Chatterjee, S., Tuske, S., Ho, M. X., Arnold, E., and Ebright, R. H. (2012). Structural basis of transcription initiation. *Science*, 338(6110):1076–1080.

[201] Zulkower, V., Page, M., Ropers, D., Geiselmann, J., and de Jong, H. (2015). Robust reconstruction of gene expression profiles from reporter gene data using linear inversion. *Bioinformatics*, 31(12):71–79.

[202] Zwietering, M. H., Jongenburger, I., Rombouts, F. M., and van 't Riet, K. (1990). Modeling of the Bacterial Growth Curve. *Applied and Environmental Microbiology*, 56(6):1875–1881.

Appendix A

Automated design build test learn workflows to engineer synthetic genetic networks

Designed DNA sequences

Part name	Sequence
L3S2P11-UPA20-Prhl-RiboJ10	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagagacgctttcgagcgtctttttttcgttttggtccgtg cctactctggaaaatcttcctgtgaaatctggcagttaccgttagctttcgaattggctaaa aagtgttcagcgctcaacgggtgtgcttcccgttctgatgagtccgtgaggacgaaagcgcc tctacaaataatttttgtttaatactcgagagaagagccacgtagtgggccatcgccagctgt cagcactactaacttgcggtcagt
L3S2P21-UPA20-Plux-RiboJ	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagaggcctcccgaaaggggggccttttttcgttttggtc cgtgcctactctggaaaatctacctgtaggatcgtacaggtttacgcaagaaaatggtttgt tatagtcgaataaaagctgtcaccggatgtgctttccggtctgatgagtccgtgaggacgaa acagcctctacaaataatttttgtttaatactcgagagaagagccacgtagtgggccatcgcc agctgtcagcactactaacttgcggtcagt
L3S1P13-UPA20-Ptra-RiboJ	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg aggacgaacaataaggcctccctaacgggggggccttttttattgataacaaaagtgcctact ctggaaaatctgcacgtgcagatctgcacatttacgcaagaaaatggtttgttatagtcgaa tatagctgtcaccggatgtgctttccggtctgatgagtccgtgaggacgaaacagcctctac aaataatttttgtttaatactcgagagaagagccacgtagtgggccatcgccagctgtcagca ctactaacttgcggtcagt

Table A.1 Receiver receptor promoters 1.

Part name	Sequence
L3S2P11-UPA20-Plas-RiboJ10	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagagacgctttcgagcgtctttttcgttttggtccgtg cctactctggaaaatctcctttccgaaacgaaacaagttggattttgcacctaccagaactg gtagttctgacctgtggctatcttcgaaggcatcgatattatgcacattggaactcttcatg acataacgccgagagcgctcaacgggtgtgcttcccgttctgatgagtccgtgaggacgaaa gcgcctctacaaataattttgtttaatactcgagagaagagccacgtagtgggccatcgcca gctgtcagcactactaacttgcggtcagt
L3S1P13-UPA20-Pcin-RiboJ51	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg aggacgaacaataaggcctccctaacgggggggccttttttattgataacaaaagtgcctact ctggaaaatctgggggcctatctgagggaatttacttccctatcagtgatagagatactgag cacatccctatcagtgatagagatagtagtcaccggctgtgcttgccggtctgatgagcctg tgaaggcgaaactacctctacaaataattttgtttaatactcgagagaagagccacgtagtg ggccatcgccagctgtcagcactactaacttgcggtcagt
L3S2P21-UPA20-Prpa-RiboJ51	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagaggcctcccgaaaggggggccttttttcgttttggtc cgtgcctactctggaaaatctgcacctgtccgatcggacagtattacgcaagaaaatggttt gttatagtcgaatatagtagtcaccggctgtgcttgccggtctgatgagcctgtgaaggcga aactacctctacaaataattttgtttaatactcgagagaagagccacgtagtgggccatcgc cagctgtcagcactactaacttgcggtcagt

Table A.2 Receiver receptor promoters 2.

Part name	Sequence
CinR	tgcccgctttccagtcggggctcttcatcgaatgattgagaatacctatagcgaaaagttcg agtccgcgttc-gaacagatcaaggcggcggccaacgtggatgccgccatccgtattctccag gcggtatataacctc-gatttcgtcacctaccatctcgcccagacgatcgcgagcaagatcga tccgcccttcgtgcgcac-cacctatccggatgcctgggtttcccgctacctcctcaacagct atgtgaaggtcgatccgatcgt-caagcagggcttcgaacgccagctgcccttcgactggagc gaggtcgaaccgacgccggaggcc-tatgccatgctggtcgacgcccagaaacacggcatcgg tggcaatggctactccatccccgtcgccga-caaggcgcagcgccgcgccctgctgtcgctga atgcccgtataccggccgacgaatggaccgagcttgt-gcgccgctgccgcaacgagtggatc gagatcgcccatctgatccaccgcaaggccgtctatgagctg-catggcgaaaacgatccggt gccggcattgtcgccgcgcgagatcgagtgtctgcactggaccgcc-ctcggcaaggattaca aggatatttcggtcatcctgggcatatcagagcataccacacgcgattacctgaa-gaccgcc cgcttcaagctcggctgcgccacgatctcggccgccgcgtcgcgggctgttcaattgcgcat catcaatcccggagcccgagagaagagccacgtagtgggccatcgcc
LasR	tgcccgctttccagtcggggctcttcatcgaatggccttggttgacggttttcttgagctgg aacgctcaagtg-gaaaattggagtggagcgccatcctccagaagatggcgagcgaccttgga ttctcgaagatcctgttcggc-ctgttgcctaaggacagccaggactacgagaacgccttcat cgtcggcaactacccggccgcctggcgc-gagcattacgaccgggctggctacgcgcgggtcg acccgacggtcagtcactgtacccagagcgtact-gccgattttctgggaaccgtccatctac cagacgcgaaagcagcacgagttcttcgaggaagcctcggc-cgccggcctggtgtatgggct gaccatgccgctgcatggtgctcgcggcgaactcggcgcgctgagc-ctcagcgtggaagcgg aaaaccgggccgaggccaaccgtttcatagagtcggtcctgccgaccctgtg-gatgctcaag gactacgcactgcaaagcggtgccggactggccttcgaacatccggtcagcaaaccg-gtggt tctgaccagccgggagaaggaagtgttgcagtggtgcgccatcggcaagaccagttgggaga tatcggttatctgcaactgctcggaagccaatgtgaacttccatatgggaaatattcggcgg aagttcggt-gtgacctcccgccgcgtagcggccattatggccgttaatttgggtcttattac tctcggagcccgagagaa-gagccacgtagtgggccatcgcc
LuxR	tgcccgctttccagtcggggctcttcatcgaatgaaaaacataaatgccgacgacacataca gaataat-taataaaattaaagcttgtagaagcaataatgatattaatcaatgcttatctgat atgactaaaatggtacattgt-gaatattatttactcgcgatcatttatcctcattctatggt taaatctgatatttcaatcctagataattaccc-taaaaaatggaggcaatattatgatgacg ctaatttaataaaatatgatcctatagtagattattctaactc-caatcattcaccaattaat tggaatatatttgaaaacaatgctgtaaataaaaaatctccaaatgtaattaaa-gaagcgaa aacatcaggtcttatcactgggtttagtttccctattcatacggctaacaatggcttcggaa tgcttagttttgcacattcagaaaaagacaactatatagatagtttatttttacatgcgtgt atgaacataccat-taattgttccttctctagttgataattatcgaaaaataaatatagcaaa taataaatcaaacaacgatttaac-caaaagagaaaaagaatgtttagcgtgggcatgcgaag gaaaaagctcttgggatatttcaaaaatattag-gttgcagtgagcgtactgtcactttccat ttaaccaatgcgcaaatgaaactcaatacaacaaaccgctgc-caaagtatttctaaagcaat tttaacaggagcaattgattgcccatactttaaaaatggagcccgagagaa-gagccacgtag tgggccatcgcc

Table A.3 Receiver receptor CDS 1.

Part name	Sequence
RpaR	tgcccgctttccagtcggggctcttcatcgaatgatcgtcggcgaagatcagctttggggac ggcgtgcgctggagttcgtcgattccgtcgaacggctcgaggcgccggcgctgatcagccgg ttcgaatcgctgatcgcgagctgcggatttaccgcctacatcatggccggcctgccgtcgcg caatgccggactaccggagctgacgctggccaatggctggccgcgagactggttcgatctgt atgtcagcgaaaacttcagcgcggtcgatccggtgccgcgccacggcgctaccacggttcat cctttcgtatggtccgatgcaccctacgaccgcgaccgtgatccggccgcccaccgggtcat gacccgggcggcggagttcggactggtcgagggttactgcattccgctgcactacgacgacg gtagcgccgcgatcagcatggccggcaaggatccggacctcagcccggccgcgcgcggcgcg atgcagctggtcagcatctacgcgcatagtcgcctgcgcgcactcagccggccaaagccgat ccggcgcaaccggctcacgccgcgcgagtgcgagatcctgcaatgggcagcgcagggcaaga ccgcctgggaaatctcggtaatcctctgcatcaccgaacgcacggtgaaattccatctgatc gaagccgcccgcaagctcgacgccgccaaccgcaccgcggcggttgccaaggcattgacgct cggattgatccgtttgggagcccgagagaagagccacgtagtgggccatcgcc
TraR	tgcccgctttccagtcggggctcttcatcgaatgcagcactggctggacaagctgactgatc ttgccgcgatcgaaggcgatgagtgcatcctgaagaccgggctggcggacatcgccgaccat ttcggcttcaccggctatgcctaccttcatatccagcacaggcacatcaccgccgttaccaa ctatcaccgccaatggcaatcaacctacttcgacaagaagttcgaagcgctcgatccggtcg tcaaacgcgcgaggtcccggaagcacatcttcacctggtcgggcgagcacgagcggccgacg ctgtcgaaggacgagcgtgccttctatgaccacgcatccgatttcggcatccgctccggcat cacaatacccatcaagaccgccaacggctttatgtcgatgttcacgatggcatcggacaagc cggtgatcgatctcgatcgggagatcgatgcagtcgcagccgctgcaaccatcgggcagatc catgcccgcatctcattccttcgcaccacccctaccgcggaagatgccgcatggctcgatcc gaaggaggccacctatctgagatggattgccgtcggcaagacgatgtgggagatcgccgacg tcgaaggggtcaagtacaacagcgtccgcgtcaagctacgcgaagccatgaagcgcttcgac gtccgcagcaaggcccatcttaccgcgctcgccatccggcggaaactcatcggagcccgaga gaagagccacgtagtgggccatcgcc

Table A.4 Receiver receptor CDS 2.

Part name	Sequence
CinI	tgcccgctttccagtcggggctcttcatcgaatgcagcactggctggacaagctgactgatc ttgccgc-gatcgaaggcgatgagtgcatcctgaagaccgggctggcggacatcgccgaccat ttcggcttcaccg-gctatgcctaccttcatatccagcacaggcacatcaccgccgttaccaa ctatcaccgccaatggcaat-caacctacttcgacaagaagttcgaagcgctcgatccggtcg tcaaacgcgcgaggtcccggaagca-catcttcacctggtcgggcgagcacgagcggccgacg ctgtcgaaggacgagcgtgccttctatgac-cacgcatccgatttcggcatccgctccggcat cacaatacccatcaagaccgccaacggctttatgtcgat-gttcacgatggcatcggacaagc cggtgatcgatctcgatcgggagatcgatgcagtcgcagccgctg-caaccatcgggcagatc catgcccgcatctcattccttcgcaccacccctaccgcggaagatgccgcatg-gctcgatcc gaaggaggccacctatctgagatggattgccgtcggcaagacgatgtgggagatcgcc-gacg tcgaaggggtcaagtacaacagcgtccgcgtcaagctacgcgaagccatgaagcgcttcgac gtccgcagcaaggcccatcttaccgcgctcgccatccggcggaaactcatcggagcccgaga gaa-gagccacgtagtgggccatcgcc
LasI	tgcccgctttccagtcggggctcttcatcgaatgatcgttcagatcggtcgtcgtgaagagt tcga-caaaaaactgctgggtgaaatgcacaaactgcgtgctcaggttttcaaagaacgtaaa ggttgggacgtttc-cgttatcgacgaaatggaaatcgacggttacgacgctctgtccccgta ctacatgctgatccaggaaga-caccccggaagctcaggttttcggttgctggcgtatcttcg acaccaccggtccgtacatgctgaaaaacac-cttcccggaactgctgcacggtaaagaagct ccgtgctccccgcacatctgggaactgtcccgtttcgctat-caactccggtcagaaaggttc cctgggtttctccgactgcaccctggaagctatgcgtgctctggctcgt-tactccttgcaga acgacatccagaccctggttaccgttaccaccgttggtgttgaaaaaatgatgatccgt-gct ggtctggacgtttcccgtttcggtccgcacctgaaaatcggtatcgaacgtgctgttgctct gcgtatc-gaactgaacgctaaaacccagatcgctctgtacggtggtgttctggttgaacagc gtctggctgtttccg-gagcccgagagaagagccacgtagtgggccatcgcc
LuxI	tgcccgctttccagtcggggctcttcatcgaatgactataatgataaaaaaatcggattttt tggcaattc-catcggaggagtataaaggtattctaagtcttcgttatcaagtgtttaagcaa agacttgagtgggact-tagttgtagaaaataaccttgaatcagatgagtatgataactcaaa tgcagaatatatttatgcttgtgatgat-actgaaaatgtaagtggatgctggcgtttattac ctacaacaggtgattatatgctgaaaagtgtttttc-ctgaattgcttggtcaacagagtgct cccaaagatcctaatatagtcgaattaagtcgttttgctgtag-gtaaaaatagctcaaagat aaataactctgctagtgaaattacaatgaaactatttgaagctatatataaa-cacgctgtta gtcaaggtattacagaatatgtaacagtaacatcaacagcaatagagcgatttttaaagcgt attaaagttccttgtcatcgtattggagacaaagaaattcatgtattaggtgatactaaatc ggttgtattgtc-tatgcctattaatgaacagtttaaaaaagcagtcttaaatggagcccgag agaagagccacgtagtgggc-catcgcc
RhlI	tgcccgctttccagtcggggctcttcatcgaatgatcgaactgctgtccgaatccctggaag gtctgtccgct-gctatgatcgctgaactgggtcgttaccgtcaccaggttttcatcgaaaaa ctgggttgggacgttgtttccac-ctcccgtgttcgtgaccaggagttcgaccagttcgacca cccgcagacccgttacatcgttgctatgtcc-cgtcagggtatctgcggttgcgctcgtctgc tgccgaccaccgacgcttacctgctgaaagacgttttcgct-tacctgtgctccgaaaccccg ccgtccgacccgtccgtttgggaactgtcccgttacgctgcttccgctgct-gacgacccgca gctggctatgaaaatcttctggtcctccctccagtgcgcttggtacctgggtgcttcctccg ttgttgctgttaccaccaccgctatggaacgttacttcgttcgtaacggtgttatcctccag cgtctgggtccgc-cgcagaaagttaaaggtgaaaccctggttgctatctccttcccggctta ccaggaacgtggtctggaaat-gctgctgcgttaccacccggaatggctccagggtgttccgc tgtccatggctgttggagcccgagagaa-gagccacgtagtgggccatcgcc

Table A.5 Receiver sender CDS.

Part name	Sequence
L3S1P13-UPA20-plambda-RiboJ51	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg aggacgaacaataaggcctccctaacgggggggccttttttattgataacaaaagtgcctact ctggaaaatctccgccgccctagacctagctgcaggtcgaggataaatatctaacaccgtgc gtgttgactattttacctctggcggtgataatggttgcatgtactagaattcattagtagtc accggctgtgcttgccggtctgatgagcctgtgaaggcgaaactacctctacaaataatttt gtttaatactcgagagaagagccacgtagtgggccatcgccagctgtcagcactactaactt gcggtcagt
L3S1P13-UPV1-pAmeR-Riboj51	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg aggacgaacaataaggcctccctaacgggggggccttttttattgataacaaaatcgtcacta gagggcgatagtgacaaacttgacaactcatcacttcctaggtataatgctagcagtagtca ccggctgtgcttgccggtctgatgagcctgtgaaggcgaaactacctctacaaataatttttg tttaatactcgagagaagagccacgtagtgggccatcgccagctgtcagcactactaacttg cggtcagt
L3S1P13-UPV2-pAmtR-RiboJ10	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg aggacgaacaataaggcctccctaacgggggggccttttttattgataacaaaacttgtccaa ccaaatagcgctcaacgggtgtgcttcccgttctgatgagtccgtgaggacgaaagcgcctc tacaaataatttttgtttaatactcgagagaagagccacgtagtgggccatcgcc
L3S1P13-UPV3-pBetI-RiboJ	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg aggacgaacaataaggcctccctaacgggggggccttttttattgataacaaaaagcgcgggt gagagggattcgttaccaattgacaattgattggacgttcaatataatgctagcagctgtca ccggatgtgctttccggtctgatgagtccgtgaggacgaaacagcctctacaaataatttttg tttaatactcgagagaagagccacgtagtgggccatcgccagctgtcagcactactaacttg cggtcagt
L3S2P11-UPA20-pLac-RiboJ10	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagagacgctttcgagcgtctttttcgttttggtccgtg cctactctggaaaatctcctttcgtcttcacctcgagaattgtgagcggataacaattgaca ttgtgagcggataacaagatactgagcacatcagcaggacgcactgaccgaattcattagcg ctcaacgggtgtgcttcccgttctgatgagtccgtgaggacgaaagcgcctctacaaataat tttgtttaatactcgagagaagagccacgtagtgggccatcgccagctgtcagcactactaa cttgcggtcagt
L3S2P11-UPV4-pBM3R1-RiboJ51	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagagacgctttcgagcgtctttttcgttttggtccaat ccgcgtgataggtctgattcgttaccaattgacggaatgaacgttcattccgataatgctag cagtagtcaccggctgtgcttgccggtctgatgagcctgtgaaggcgaaactacctctacaa ataattttgtttaatactcgagagaagagccacgtagtgggccatcgccagctgtcagcact actaacttgcggtcagt

Table A.6 Repressible promoters 1.

Part name	Sequence
L3S2P11-UPV5-pHlyllR-RiboJ	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagagacgctttcgagcgtctttttttcgttttggtccacc aggaatctgaacgattcgttaccaattgacatatttaaaattcttgtttaaaatgctagcag ctgtcaccggatgtgctttccggtctgatgagtccgtgaggacgaaacagcctctacaaata attttgtttaatactcgagagaagagccacgtagtgggccatcgccagctgtcagcactact aacttgcggtcagt
L3S2P21-UPA20-pTetR-RiboJ	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagaggcctcccgaaaggggggccttttttcgttttggtc cgtgcctactctggaaaatcttccctatcagtgatagagattgacatccctatcagtgatag agatactgagcacatcagcaggacgcactgaccagctgtcaccggatgtgctttccggtctg atgagtccgtgaggacgaaacagcctctacaaataatttttgtttaatactcgagagaagagc cacgtagtgggccatcgccagctgtcagcactactaacttgcggtcagt
L3S2P21-UPV6-pPhiF-RiboJ51	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagaggcctcccgaaaggggggccttttttcgttttggtc ccgacgtacggtggaatctgattcgttaccaattgacatgatacgaaacgtaccgtatcgtt aaggtagtagtcaccggctgtgcttgccggtctgatgagcctgtgaaggcgaaactacctct acaaataatttttgtttaatactcgagagaagagccacgtagtgggccatcgccagctgtcag cactactaacttgcggtcagt
L3S2P21-UPV7-pSrpR-RiboJ10	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagaggcctcccgaaaggggggccttttttcgttttggtc ctctatgattggtccagattcgttaccaattgacagctagctcagtcctaggtatatacata catgcttgtttgtttgtaaacagcgctcaacgggtgtgcttcccgttctgatgagtccgtga ggacgaaagcgcctctacaaataatttttgtttaatactcgagagaagagccacgtagtgggc catcgccagctgtcagcactactaacttgcggtcagt
L3S2P21-UPV8-pLitR-RiboJ10	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcggg agctcggtaccaaattccagaaaagaggcctcccgaaaggggggccttttttcgttttggtc ccgagcgtagagctagattcgttaccaattgacaaatttataaattgtcagtataatgcta gcagcgctcaacgggtgtgcttccgttctgatgagtccgtgaggacgaaagcgcctctaca aataattttgtttaatactcgagagaagagccacgtagtgggccatcgccagctgtcagcac tactaacttgcggtcagt

Table A.7 Repressible promoters 2.

Part name	Sequence
AmeR	tgcccgctttccagtcggggctcttcatcgaatgaacaaaaccattgatcaggtgcgt aaaggtgatcg-taaaagcgatctgccggttcgtcgtcgtccgcgtcgtagtgccgaaga aacccgtcgtgatattctg-gcaaaagccgaagaactgtttcgtgaacgtggttttaatgca gttgccattgcagatattgcaagcgcact-gaatatgagtccgg caaatgtgtttaaacattttagcagcaaaaacgcactggttgatgcaattggttttg-gtcag attggtgtttttgaacgtcagatttgt ccgctggataaaagccatgcaccgctggatcgtctgcgt-catctggcacgtaatctgatgga acagcatcatcaggatcatttcaaacacatacgggtttttattcagatc-ctgatgaccgcca aacaggatatgaaatgtggcgattattacaaaagcgtgattgcaaaactgctggc-cgaaatt attcgtgatggtgttgaagcaggtctgtatattgcaaccgatattccggttctggcagaaac cgttctgcatgcactgaccagcgttattcatccggttctgattgcacaagaagatattggta atctggcaacc-cgttgtgatcagctggttgatctgattgatgcaggtctgcgtaatccgctg gcaaaaggagcccgagagaa-gagccacgtagtgggccatcgcc
AmtR	tgcccgctttccagtcggggctcttcatcgaatggcaggcgcagttggtcgtccgcgtcgta gtg-caccgcgtcgtgcaggtaaaaatccgcgtgaagaaattctggatgcaagcgcagaactg tttacc-cgtcagggttttgcaaccaccagtacccatcagattgcagatgcagttggtattcg tcaggcaagcct-gtattatcattttccgagcaaaaccgaaatctttctgaccctgctgaaaa gcaccgttgaaccgagcac-cgttctggcagaagatctgagcaccctggatgcaggtccggaa atgcgtctgtgggcaattgttgcaagc-gaagttcgtctgctgctgagcaccaaatggaatgt tggtcgtctgtatcagctgccgattgttggtagcgaa-gaatttgcagaatatcatagccagc gtgaagcactgaccaatgtttttcgtgatctggcaaccgaaattgttg-gtgatgatccgcgt gcagaactgccgtttcatattaccatgagcgttattgaaatgcgtcgcaatgatg-gtaaaat tccgagtccgctgagcgcagatagcctgccggaaaccgcaattatgctggcagatgcaagcc tggcagttctgggtgcaccgctgcctgcagatcgtgttgaaaaaaccctggaactgattaaa caggcagat-gcaaaaggagcccgagagaagagccacgtagtgggccatcgcc

Table A.8 Repressor CDS 1.

Part name	Sequence
BetI	tgcccgctttccagtcggggctcttcatcgaatgccgaaactgggtatgcagagcattcgtc gtcgtcagct-gattgatgcaaccctggaagcaattaatgaagttggtatgcatgatgcaacc attgcacagattgcacgtcgt-gccggtgttagcaccggtattattagccattatttccgcga taaaaacggtctgctggaagcaaccatgcgt-gatattaccagccagctgcgtgatgcagttc tgaatcgtctgcatgcactgccgcagggtagcgca-gaacagcgtctgcaggcaattgttggt ggtaattttgatgaaacccaggttagcagcgcagcaat-gaaagcatggctggcattttgggc aagcagcatgcatcagccgatgctgtatcgtctgcagcaggt-tagcagtcgtcgtctgctga gcaatctggttagcgaatttcgtcgtgaactgcctcgtgaacaggcacaa-gaggcaggttat ggtctggcagcactgattgatggtctgtggctgcgtgcagcactgagcggtaaac-cgctgga taaaacccgtgcaaatagcctgacccgtcattttatcacccagcatctgccgaccgatggag cccgagagaagagccacgtagtgggccatcgcc
BM3R1	tgcccgctttccagtcggggctcttcatcgaatggaaagcaccccgaccaaacagaaagcaa tttt-tagcgcaagcctgctgctgtttgcagaacgtggttttgatgcaaccaccatgccgatg attgcagaaaat-gcaaaagttggtgcaggcaccatttatcgctatttcaaaaacaaagaaag cctggtgaacgaact-gtttcagcagcatgttaatgaatttctgcagtgtattgaaagcggtc tggcaaatgaacgtgatggttatcgt-gatggctttcatcacatttttgaaggtatggtgacc tttaccaaaaatcatccgcgtgcactgggttttat-caaaacccatagccagggcacctttct gaccgaagaaagccgtctggcatatcagaaactggttgaatttgt-gtgcacctttttttcgtg aaggtcagaaacagggtgtgattcgtaatctgccggaaaatgcactgattg-caattctgttt ggcagctttatggaagtgtatgaaatgatcgagaacgattatctgagcctgaccgatgaact gctgaccggtgttgaagaaagcctgtgggcagcactgagccgtcagagcggagcccgagaga agagc-cacgtagtgggccatcgcc

Table A.9 Repressor CDS 2.

Part name	Sequence
HlyllR	tgcccgctttccagtcggggctcttcatcgaatgaaatacatcctgtttgaggtgtgcgaaa tgggtaaaagc-cgtgaacagaccatggaaaatattctgaaagcagccaaaaagaaattcggc gaacgtggttatgaag-gcaccagcattcaagaaattaccaaagaagccaaagttaacgttgc aatggccagctattactttaatg-gcaaagagaacctgtactacgaggtgttcaaaaaatacg gtctggcaaatgaactgccgaactttctg-gaaaaaaaccagtttaatccgattaatgccctg cgtgaatatctgaccgtttttaccacccacattaaa-gaaaatccggaaattggcaccctggc ctatgaagaaattatcaaagaaagcgcacgcctggaaaaaat-caaaccgtattttatcggca gcttcgaacagctgaaagaaattctgcaagagggtgaaaaacagggtgt-gtttcacttttt agcatcaaccataccatccattggattaccagcattgttctgtttccgaaattcaaaaaatt catcgatagcctgggtccgaatgaaaccaatgataccaatcatgaatggatgccggaagatc tggttagc-cgtattattagcgcactgaccgataaaccgaacattggagcccgagagaagagc cacgtagtgggc-catcgcc
LacI	cactagctcagattcagagtcggggctcttcatcgaatggtgaatgtgaaaccagtaacgtt atacgat-gtcgcagagtatgccggtgtctcttatcagaccgtttcccgcgtggtgaaccagg ccagccacgtttct-gcgaaaacgcgggaaaaagtggaagcggcgatggcggagctgaattac attcccaaccgcgtggca-caacaactggcgggcaaacagtcgttgctgattggcgttgccac ctccagtctggccctgcacgcgc-cgtcgcaaattgtcgcggcgattaaatctcgcgccgatc aactgggtgccagcgtggtggtgtcgatggta-gaacgaagcggcgtcgaagcctgtaaagcg gcggtgcacaatcttctcgcgcaacgcgtcagtgggct-gatcattaactatccgctggatga ccaggatgccattgctgtggaagctgcctgcactaatgttccg-gcgttatttcttgatgtct ctgaccagacacccatcaacagtattattttctcccatgaagacggtacgc-gactgggcgtg gagcatctggtcgcattgggtcaccagcaaatcgcgctgttagcgggcccat-taagttctgt ctcggcgcgtctgcgtctggctggctggcataaatatctcactcgcaatcaaattcagc-cga tagcggaacgggaaggcgactggagtgccatgtccggttttcaacaaaccatgcaaatgctg aat-gagggcatcgttcccactgcgatgctggttgccaacgatcagatggcgctgggcgcaat gcgcgccat-taccgagtccgggctgcgcgttggtgcggatatctcggtagtgggatacgacg ataccgaagacagct-catgttatatcccgccgttaaccaccatcaaacaggattttcgcctg ctggggcaaaccagcgtggac-cgcttgctgcaactctctcagggccaggcggtgaagggcaa tcagctgttgcccgtctcactggtgaaaa-gaaaaaccaccctggcgcccaatacgcaaaccg cctctccccgcgcgttggccgattcattaatgcagctg-gcacgacaggtttcccgactggaa agcgggcagggagcccgagagaagagccacgtatagaccgct-gttgc

Table A.10 Repressor CDS 3.

Part name	**Sequence**
LacI	cactagctcagattcagagtcggggctcttcatcgaatggtgaatgtgaaaccagtaacgtt atacgatgtcgcagagtatgccggtgtctcttatcagaccgtttcccgcgtggtgaaccagg ccagccacgtttctgcgaaaacgcgggaaaaagtggaagcggcgatggcggagctgaattac attcccaaccgcgtggcacaacaactggcgggcaaacagtcgttgctgattggcgttgccac ctccagtctggccctgcacgcgccgtcgcaaattgtcgcggcgattaaatctcgcgccgatc aactgggtgccagcgtggtggtgtcgatggtagaacgaagcggcgtcgaagcctgtaaagcg gcggtgcacaatcttctcgcgcaacgcgtcagtgggctgatcattaactatccgctggatga ccaggatgccattgctgtggaagctgcctgcactaatgttccggcgttatttcttgatgtct ctgaccagacacccatcaacagtattattttctcccatgaagacggtacgcgactgggcgtg gagcatctggtcgcattgggtcaccagcaaatcgcgctgttagcgggcccattaagttctgt ctcggcgcgtctgcgtctggctggctggcataaatatctcactcgcaatcaaattcagccga tagcggaacgggaaggcgactggagtgccatgtccggttttcaacaaaccatgcaaatgctg aatgagggcatcgttcccactgcgatgctggttgccaacgatcagatggcgctgggcgcaat gcgcgccattaccgagtccgggctgcgcgttggtgcggatatctcggtagtgggatacgacg ataccgaagacagctcatgttatatcccgccgttaaccaccatcaaacaggattttcgcctg ctggggcaaaccagcgtggaccgcttgctgcaactctctcagggccaggcggtgaagggcaa tcagctgttgcccgtctcactggtgaaaagaaaaaccaccctggcgcccaatacgcaaaccg cctctccccgcgcgttggccgattcattaatgcagctggcacgacaggtttcccgactggaa agcgggcagggagcccgagagaagagccacgtatagaccgctgttgc
CI	tgcccgctttccagtcggggctcttcatcgaatgagcacaaaaaagaaaccattaacacaag agcagcttgaggacgcacgtcgccttaaagcaatttatgaaaaaaagaaaaatgaacttggc ttatccaggaatctgtcgcagacaagatggggatggggcagtcaggcgttggtgctttatt taatggcatcaatgcattaaatgcttataacgccgcattgcttgcaaaaattctcaaagtta gcgttgaagaatttagccccttcaatcgccagagaaatctacgagatgtatgaagcggttagt atgcagccgtcacttagaagtgagtatgagtaccctgttttttctcatgttcaggcagggat gttctcacctgagcttagaacctttaccaaaggtgatgcggagagatgggtaagcacaacca aaaaagccagtgattctgcattctggcttgaggttgaaggtaattccatgaccgcaccaaca ggctccaagccaagctttcctgacggaatgttaattctcgttgaccctgagcaggctgttga gccaggtgatttctgcatagccagacttgggggtgatgagtttaccttcaagaaactgatca gggatagcggtcaggtgtttttacaaccactaaacccacagtacccaatgatcccatgcaat gagagttgttccgttgtggggaaagttatcgctagtcagtggcctgaagagacgtttggcgg agcccgagagaagagccacgtagtgggccatcgcc

Table A.11 Repressor CDS 4.

Part name	Sequence
LitR	tgcccgctttccagtcggggctcttcatcgaatggataccattcagaaacgtccgcgtaccc gtctgagtc-cggaaaaacgtaaagaacagctgctggatattgccattgaagtttttagccag cgtggtattggtcgtg-gtggtcatgcagatattgcagaaattgcacaggttagcgttgcaac cgtgtttaactattttccgacccgt-gaagatctggttgatgatgttctgaacaaagtggaaa acgagtttcaccagttcatcaataacagcattagc-ctggatctggatgttcgtagcaatctg aataccctgctgctgaacattattgatagcgttcagaccggcaa-caaatggattaaagtttg gtttgaatggtcaaccagcacccgtgatgaagtttggcctctgtttctgagcac-ccatagca ataccaatcaggtgatcaaaaccatgtttgaagagggtattgaacgcaatgaagtgtgcaat gatcatacaccggaaaatctgaccaaaatgctgcatggtatttgctatagcgtgtttattca ggccaatcg-taatagcagcagcgaagaaatggaagaaaccgcaaattgctttctgaatatgc tgtgcatctacaaag-gagcccgagagaagagccacgtagtgggccatcgcc
PhiF	tgcccgctttccagtcggggctcttcatcgaatggcacgtaccccgagccgtagcagcattg gtagc-ctgcgtagtccgcatacccataaagcaattctgaccagcaccattgaaatcctgaaa gaatgtggt-tatagcggtctgagcattgaaagcgttgcacgtcgtgccggtgcaagcaaacc gaccatttatcgttg-gtggaccaataaagcagcactgattgccgaagtgtatgaaaatgaaa gcgaacaggtgcgtaaatttc-cggatctgggtagctttaaagccgatctggattttctgctg cgtaatctgtggaaagtttggcgtgaaac-catttgtggtgaagcatttcgttgtgttattgc agaagcacagctggaccctgcaaccctgacccagctgaaa-gatcagtttatggaacgtcgtc gtgagatgccgaaaaaactggttgaaaatgccattagcaatggtgaact-gccgaaagatacc aatcgtgaactgctgctggatatgatttttggtttttgttggtatcgcctgctgaccgaaca gctgaccgttgaacaggatattgaagaatttaccttcctgctgattaatggtgtttgtccgg gtacacagcgtg-gagcccgagagaagagccacgtagtgggccatcgcc

Table A.12 Repressor CDS 5.

Part name	Sequence
SrpR	tgcccgctttccagtcggggctcttcatcgaatggcacgtaaaaccgcagcagaagcagaag aaacccgtcagcgtattattgatgcagcactggaagtttttgttgcacagggtgttagtgat gcaaccctggatcagattgcacgtaaagccggtgttacccgtggtgcagtttattggcattt taatggtaaactggaagttctgcaggcagttctggcaagccgtcagcatccgctggaactgg attttacaccggatctgggtattgaacgtagctgggaagcagttgttgttgcaatgctggat gcagttcatagtccgcagagcaaacagtttagcgaaattctgatttatcagggtctggatga aagcggtctgattcataatcgtatggttcaggcaagcgatcgttttctgcagtatattcatc aggttctgcgtcatgcagttacccagggtgaactgccgattaatctggatctgcagaccagc attggtgtttttaaaggtctgattaccggtctgctgtatgaaggtctgcgtagcaaagatca gcaggcacagattatcaaagttgcactgggtagcttttgggcactgctgcgtgaaccgcctc gttttctgctgtgtgaagaagcacagattaaacaggtgaaatccttcgaaggagcccgagag aagagccacgtagtgggccatcgcc
TetR	tgcccgctttccagtcggggctcttcatcgaatgtccagattagataaaagtaaagtgatta acagcgcattagagctgcttaatgaggtcggaatcgaaggtttaacaacccgtaaactcgcc cagaagctaggtgtagagcagcctacattgtattggcatgtaaaaaataagcgggctttgct cgacgccttagccattgagatgttagataggcaccatactcacttttgccctttagaagggg aaagctggcaagatttttttacgtaataacgctaaaagttttagatgtgctttactaagtcat cgcgatggagcaaaagtacatttaggtacacggcctacagaaaaacagtatgaaactctcga aaatcaattagccttttttatgccaacaaggtttttcactagagaatgcattatatgcactca gcgctgtggggcattttactttaggttgcgtattggaagatcaagagcatcaagtcgctaaagaagaaagggaaacacctactactgatagtatgccgccattattacgacaagctatcgaatt atttgatcaccaaggtgcagagccagccttcttattcggccttgaattgatcatatgcggat tagaaaaacaacttaaatgtgaaagtgggtctggagcccgagagaagagccacgtagtgggc catcgcc

Table A.13 Repressor CDS 6.

Part name	Sequence
GFP mut3 non stop	cactagctcagattcagtagaccgctgttgtgcccgctttccagtcggggctcttcatcgaa tgcgtaaaggagaagaacttttcactggagttgtcccaattcttgttgaattagatggtgat gttaatgggcacaaattttctgtcagtggagagggtgaaggtgatgcaacatacggaaaact taccttaaatttatttgcactactggaaaactacctgttccatggccaacacttgtcacta ctttcggttatggtgttcaatgctttgcgagataccagatcatatgaaacagcatgacttt ttcaagagtgccatgcccgaaggttatgtacaggaaagaactatatttttcaaagatgacgg gaactacaagacacgtgctgaagtcaagtttgaaggtgataccttgttaatagaatcgagt taaaaggtattgattttaaagaagatggaaacattcttggacacaaattggaatacaactat aactcacacaatgtatacatcatggcagacaaacaaaagaatggaatcaaagttaacttcaa aattagacacaacattgaagatggaagcgttcaactagcagaccattatcaacaaaatactc caattggcgatggccctgtccttttaccagacaaccattacctgtccacacaatctgccctt tcgaaagatcccaacgaaaagcgcgaccacatggtccttcttgagtttgtaacagctgctgg gattacacatggcatggatgaactatacaaaggagcccgagagaagagccacgtagtgggcc atcgcccaacagcggtctactgaatctgagctagtg
sfGFP non stop	tgcccgctttccagtcggggctcttcatcgaatgcgtaaaggcgaggaactgttcactggtg tcgtccctattctggtggaactggatggtgatgtcaacggtcataagttttccgtgcgtggc gagggtgaaggtgacgcaactaatggtaaactgacgctgaagttcatctgtactactggtaa actgccggtaccttggccgactctggtaacgacgctgacttatggtgttcagtgctttgctc gttatccggaccatatgaagcagcatgacttcttcaagtccgccatgccggaaggctatgtg caggaacgcacgatttcctttaaggatgacggcacgtacaaaacgcgtgcggaagtgaaatt tgaaggcgataccctggtaaaccgcattgagctgaaaggcattgactttaaagaagacggca atatcctgggccataagctggaatacaattttaacagccacaatgtttacatcaccgccgat aaacaaaaaaatggcattaaagcgaatttaaaattcgccacaacgtggaggatggcagcgt gcagctggctgatcactaccagcaaaacactccaatcggtgatggtcctgttctgctgccag acaatcactatctgagcacgcaaagcgttctgtctaaagatccgaacgagaaacgcgatcat atggttctgctggagttcgtaaccgcagcgggcatcacgcatggtatggatgaactgtacaa aggagcccgagagaagagccacgtagtgggccatcgcc

Table A.14 Reporters 1.

Part name	Sequence
RFP non stop	tgcccgctttccagtcggggctcttcatcgaatggcttcctccgaggatgttatcaaagagt tcatgcgtttcaaagttcgtatggaaggttccgttaacggtcacgagttcgaaatcgaaggt gaaggtgaaggtcgtccgtacgaaggtacccagaccgctaaactgaaagttaccaaaggtgg tccgctgccgttcgcttgggacatcctgtccccgcagttccagtacggttccaaagcttacg ttaaacacccggctgacatcccggactacctgaaactgtccttcccggaaggtttcaaatgg gaacgtgttatgaacttcgaagatggtggtgttgttaccgttacccaggactcctccctgca agacggtgagttcatctacaaagttaaactgcgtggtaccaacttcccgtccgacggtccgg ttatgcagaaaaaaaccatgggttgggaagcttccaccgaacgtatgtacccggaggatggt gctctgaaaggtgaaatcaaaatgcgtctgaaactgaaagacggtggtcactacgacgctga agttaaaaccacctacatggctaaaaaaccggttcagctgccgggtgcttacaaaaccgaca tcaaactggacatcacctcccacaacgaggactacaccatcgttgaacagtacgaacgtgct gaaggtcgtcactccaccggtgctggagcccgagagaagagccacgtagtgggccatcgcc
YFP non stop	tgcccgctttccagtcggggctcttcatcgaatggtgagcaagggcgaggagctgttcaccg gggtggtgcccatcctggtcgagctggacggcgacgtaaacggccacaagttcagcgtgtcc ggcgagggcgagggcgatgccacctacggcaagctgaccctgaagttcatctgcaccaccgg caagctgcccgtgccctggcccaccctcgtgaccaccttcggctacggcctgcaatgcttcg cccgctaccccgaccacatgaagctgcacgacttcttcaagtccgccatgcccgaaggctac gtccaggagcgcaccatcttcttcaaggacgacggcaactacaagacccgcgccgaggtgaa gttcgagggcgacaccctggtgaaccgcatcgagctgaagggcatcgacttcaaggaggacg gcaacatcctggggcacaagctggagtacaactacaacagccacaacgtctatatcatggcc gacaagcagaagaacggcatcaaggtgaacttcaagatccgccacaacatcgaggacggcag cgtgcagctcgccgaccactaccagcagaacacccccatcggcgacggccccgtgctgctgc ccgacaaccactacctgagctaccagtccgccctgagcaaagaccccaacgagaagcgcgat cacatggtcctgctggagttcgtgaccgccgccgggatcactctcggcatggacgagctgta caagggagcccgagagaagagccacgtagtgggccatcgcc
CFP non stop	tgcccgctttccagtcggggctcttcatcgaatggtgagcaagggcgaggagctgttcaccg gggtggtgcccatcctggtcgagctggacggcgacgtgaacggccacaagttcagcgtgtccggc gagggcgagggcgatgccacctacggcaagctgaccctgaagttcatctgcaccaccggcaagct gcccgtgccctggcccaccctcgtgaccaccctgacctggggcgtgcagtgcttcagccgctacc ccgaccacatgaagcagcacgacttcttcaagtccgccatgcccgaaggctacgtccaggagcgca ccatcttcttcaaggacgacggcaactacaagacccgcgccgaggtgaagttcgagggcgacaccc tggtgaaccgcatcgagctgaagggcatcgacttcaaggaggacggcaacatcctggggcacaagc tggagtacaactacatcagccacaacgtctatatcaccgccgacaagcagaagaacggcatcaagg ccaacttcaagatccgccacaacatcgaggacggcagcgtgcagctcgccgaccactaccagcaga acacccccatcggcgacggccccgtgctgctgcccgacaaccactacctgagcacccagtccgccc tgagcaaagaccccaacgagaagcgcgatcacatggtcctgctggagttcgtgaccgccgccggga tcactctcggcatggacgagctgtacaagggagcccgagagaagagccacgtagtgggccatcgcc

Table A.15 Reporters 2.