Lecture 5 –
Specification - Overview, Approaches, Challenges

Prof. Douglas Densmore

EC/BE552

Computational Synthetic Biology for Engineers

Lecture Objectives

- Understand what “specification” would mean in synthetic biology.
- Outline the challenges
- Outline the current approaches

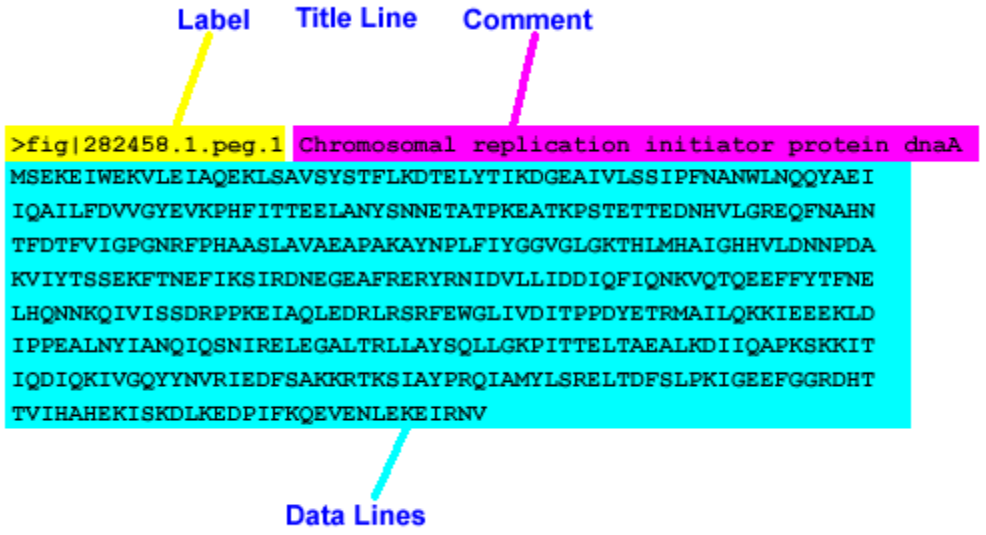
Specification Concepts

- Specifications
  - What vs. how vs. how well (orthogonal concerns)
  - Function, Structure, Performance
- Syntax vs. semantics
- Execution/simulation?

```
>yneN
TTAATGCCTCTTCTCATTTCTTCTGCTGTCATCCGCACAGCAGAAGAATTCCTCATTGAC
TATTATTTCGCAATTTGCTCACATGGATTAAATTAAACTACATACTATAAGATATAAACT
TCTGCCTACAGCTGTAAGAAACTCCGCTCAGTACTGAAGCACCAGTCCTATTTCCTCTTT
TCTCCAGCCTGTTATATTAAGCATACTGATTAACGATTTTTAACGTTATCCGCTAAATAA
ACATATTTGAAATGCATGCGACCACAGTGAAAAACAAAATCACGCAAAGAGACAACTATA
A
>yegR
ACTAACGGCTGCCACCGATAAATTTCAAAAAAGAGCATATACCTAATATTCAACTAAACA
GTGGCATCTTCAATATAATATATTAAAGCCCCCATGGAGTTACCCTGAAGGGCCTCAATG
TCCGTAATTCCTACTTATGTAGGAAATGTTGTACAGAACATTTATTATAATCCTATTCAA
TTATAATAATCATGCCATTATTATATTTAAACACTAGAGAGTGTCGTTGGTATTTAATGG
GGGAAGGTGAGATGAAAAAGATAGCTGCTATATCATTAATTAGTATTTTTATTATGTCTG
G
>emrK
AAATCAGGGATTGTACCGATGATTTATAGTTTCAAGTTGGCACTATAAGTCTTCTTACTA
ATCCTACAGGCGTAAGAATTGTATTGCAAAAGCCACGGTTTAGTCCTCTGTTGTTTTTTT
TGCACCTCATTTAAATTAGGCCTCCAACGTTCCTGGGATAATGTGCAACACATGCACTGT
GTTTGATATGAAGAATGAATGCTCTTTTCATTCAATTCATAAATTTCATCTATGAGAAAT
GAGAGATAATAGTGGAACAGATTAATTCAAATAAAAAACATTCTAACAGAAGAAAATACT
T
>evgA
AATACAATTCTTACGCCTGTAGGATTAGTAAGAAGACTTATAGTGCCAACTTGAAACTAT
AAATCATCGGTACAATCCCTGATTTTATTGTTGACATTTCATTTATGCCGACTATTTATA
TGGTATACTTGTCGAATTATCTTAAAGGAAGCTCAGATTTTCTTATTTTTATTGAGAAAA
TGAGATGACGCCTTATGTCTGTATTACTACAGGGAGAAGGGAGATGCTTCATTGCAAAGG
GAATAATCTATGAACGCAATAATTATTGATGACCATCCTCTTGCTATCGCAGCAATTCGT
>yfdX
TGGCTGTATTTACATTTAATTAATCAGT
GTGGTATATAAGAATAGTTCTCTGCGA
ATACTTCTTGCGATAATAACTACAACTC
CTGATTTCTCATTTCATGCTCACCCAA
ATGAGGTAAGTATGAAACGTTTAATTAT
C
```

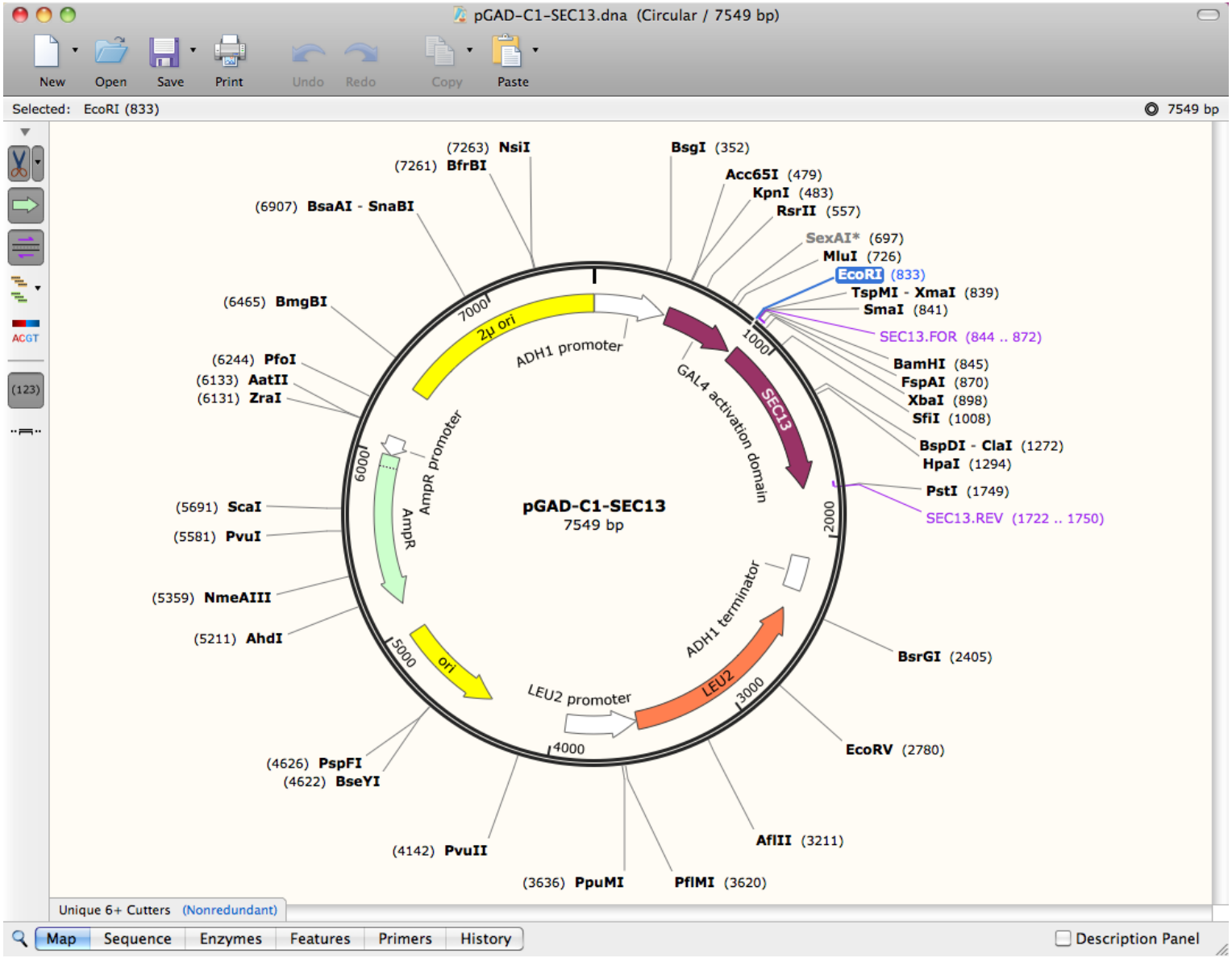
FASTA – most basic description of a DNA sequence

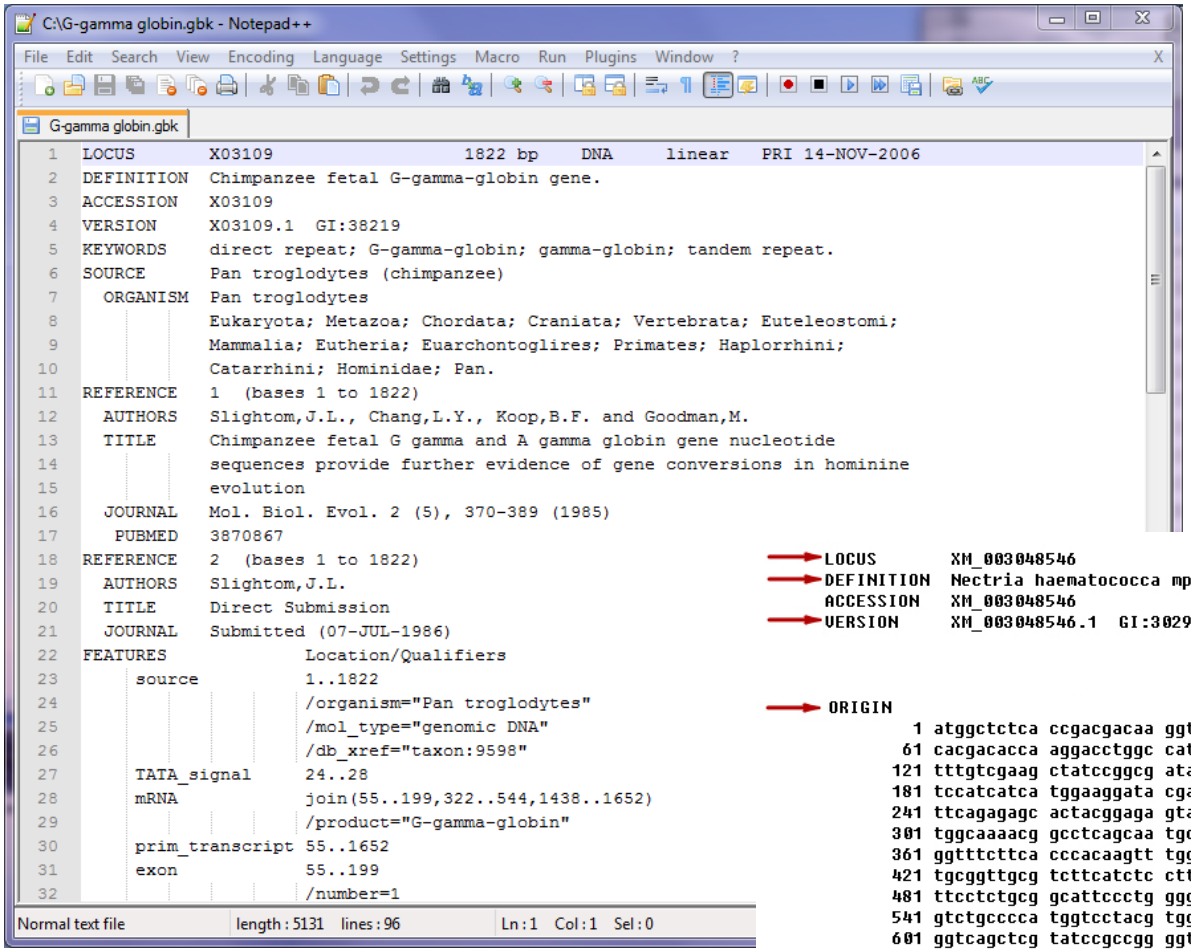
What information is this capturing? Explicitly? Implicitly?



“Plasmid Editor” – Snap Gene

What information is this capturing? Explicitly? Implicitly?





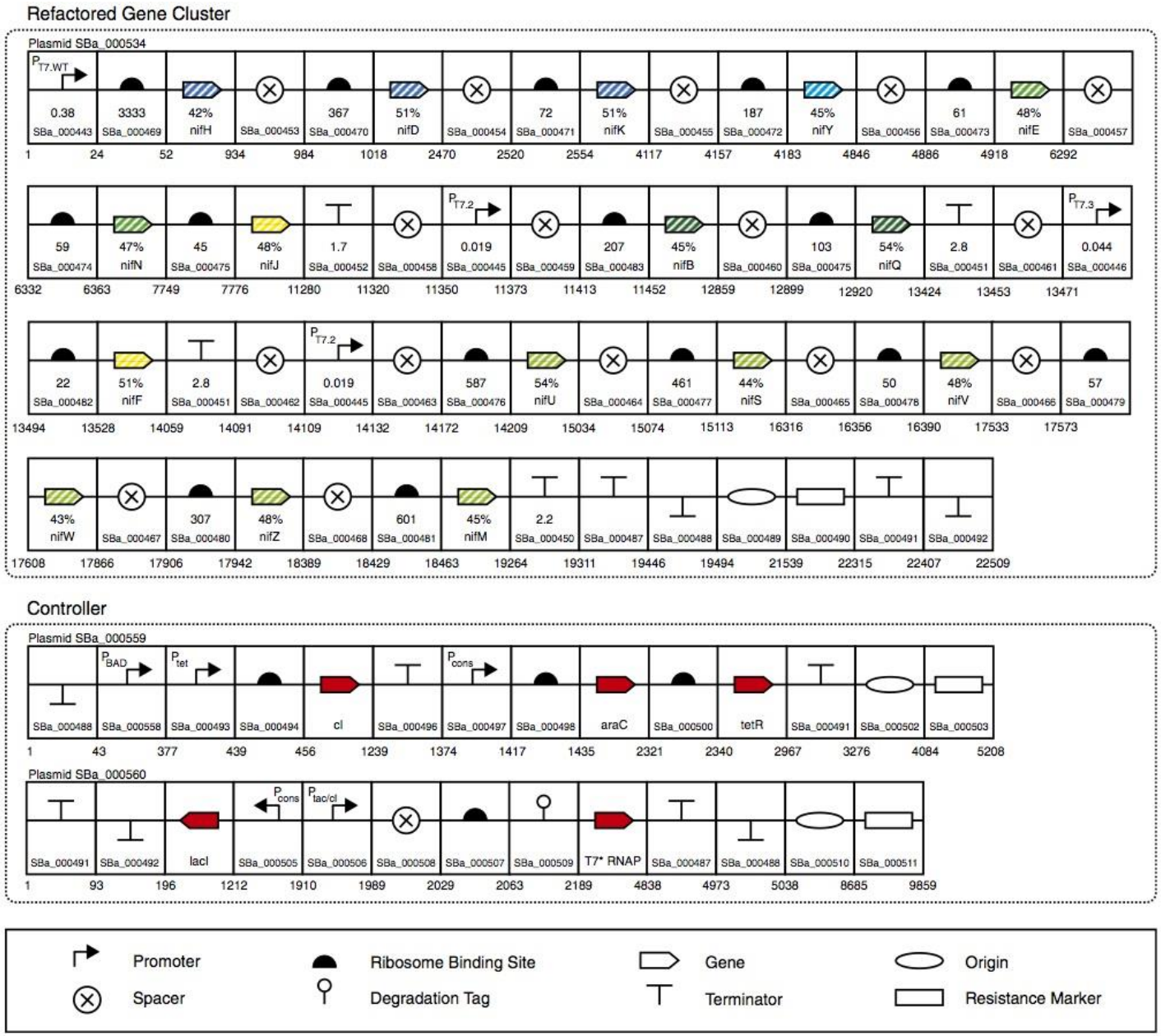
Genbank – DNA sequence, features, "meta-data"

What information is this capturing? Explicitly? Implicitly?

```
LOCUS       XM_003048546            1614 bp    mRNA    linear   PLN 14-AUG-2010
DEFINITION  Nectria haematococca mpVI 77-13-4 predicted protein, mRNA.
ACCESSION   XM_003048546
VERSION     XM_003048546.1  GI:302902148


ORIGIN
        1 atggctctca ccgacgacaa ggtgaccagg gtcagccaag atgatcaccc tcacgactat
       61 cacgacacca aggacctggc catcgccaag caggcagccg aggaagagca tcgactcacc
      121 tttgtcgaag ctatccggcg ataccctatg gccgtcatgt ggtccgtcct actgtcgacc
      181 tccatcatca tggaaggata cgacatcgtc ctgatctcct ccttcttcgc ccagccctcc
      241 ttcagagagc actacggaga gtacatttcc gagtccaaca cgtatcagat caccgcctcc
      301 tggcaaaacg gcctcagcaa tgccgtcagc gtggggacca tcattggcgc ctttgccaac
      361 ggtttcttca cccacaagtt tggttaccgc aaggtcctcc tcgcctcgct cgtcgctatc
      421 tgcggttgcg tcttcatctc cttctttttt cccaacctcc ccgttctatt ggtcggacaa
      481 ttcctctgcg gcattccctg gggagttttt gccaccatgg ccccggccta cgcatccgag
      541 gtctgcccca tggtcctacg tggctatctc actgtctatg tgaacctctg ctgggccctt
      601 ggtcagctcg tatccgccgg ggtgcaggct gggttctctg ataggcaagg ccaatggtct
      661 taccgtgttc cgttcgctat ccagtgggct tggccagttc ccctttttcat cgtgctctgg
      721 ttcgcccctg aatctccttg gtactttgtc cgcactggaa actatgacga agccgaaaag
      781 tctgtcactc gccttggctc tacgtccaaa ggcgtcaacg ccaagcaaac cgtcgccatg
      841 atgattcaca ccaacgagat tgagaagtct atcgaccaag gcacttcgta tatcgattgc
      901 ttccgcgggg tcgacctcca ccgaactgag atcgcctgca tggcatttgc cgctcagccc
      961 ttttgcggtt cggccatggg cggcactcct acatacttct tcgttcaagc cggccttccc
     1021 gagtcaatct ctttccgcat gtccgtgggt ggactcggaa ttgcttctgt tggcaccgtt
     1081 atctcctggt ggttgattca ccccttttggc tgccggaccc tgtacctctg gggtctcggt
     1141 ctcctaactg ccattcttct agccgtcggc ttcattagcg tcggggctag taactcggaa
     1201 ggtggtaact acgcgcaggc cagcatgatg cttctgtggc ttggcgttta ctacttaact
     1261 gtcggtcctg tctgctacgc catcatctct gaagtttcct cgacccgcct ccggaacaag
     1321 agcatttgcc tcagccgtat tgtctattat gtggctcaga tcatttgcaa cgtcatcaac
     1381 ccgtacatgc tgaaccccac cgccggtaac tggcgtggca aaactggttt cttctgggcg
     1441 gggtgtgcct ttgtcttctt tgtatggacc tttttccgtc ttccagagac taaggataag
     1501 accttcgaag aacttgatat tttgtttgct cacggcgtca aggctcgtga attcgcccag
     1561 taccgcgtcg acgcctacgc tgaaagcggc gatgttttga ccaaggaggg ctga
//
```

http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html

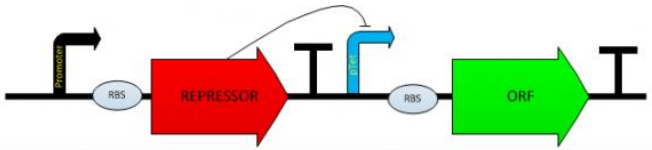
“Synthetic Biology Device”



What information is this capturing? Explicitly? Implicitly?

www.eugenecad.org

Structural Specification



A CONTAINS B	Inverter CONTAINS Repressor	
A BEFORE B	Promoter BEFORE RBS	
A AFTER B	Terminator AFTER ORF	
A STARTSWITH B	Inverter STARTSWITH Promoter	
A ENDSWITH B	Inverter ENDSWITH Terminator	
A WITH B	Repressor WITH pTet	
A THEN B	Repressor THEN pTet	

```
Property sequence(txt);
Property strength(num);
Property toxicity(txt);
Property uniqueID(num);

Part Promoter(sequence);
Part ORF(sequence, toxicity);
Part Terminator(sequence, strength);
Part RBS(sequence, uniqueID);

Rule rule2a(p BEFORE r1);
Rule rule2b(p NEXTTO r1);
Rule rule2c(p BEFORE r2);
Rule rule2d(p NEXTTO r2);
Note((rule2a AND rule2b) OR (rule2c
AND rule2d);

//Device Generation
permute(deviceType4, 25, strict);
product(deviceType5, strict)
```

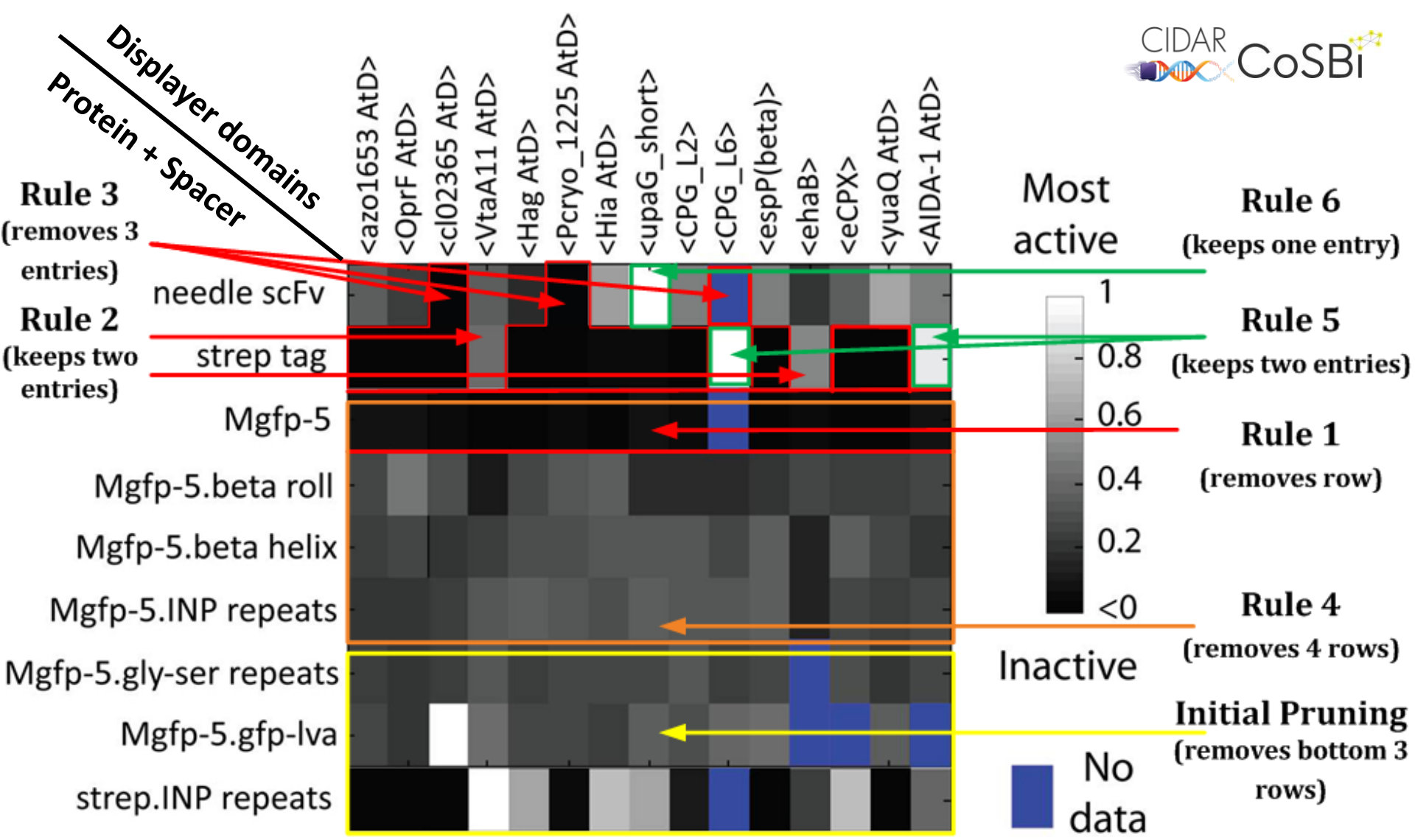

Journals

1. E. Oberortner, S. Bhatia, E. Lindgren, and **D. Densmore**, "A Rule-based Design Specification Language for Synthetic Biology," ***ACM JETC***, 2014.
2. L. Bilitchenko, A. Liu, and **D. Densmore**, "The Eugene language for synthetic biology," ***Meth. Enzymol.***, 2011.
3. L. Bilitchenko, A. Liu, S. Cheung, E. Weeding, B. Xia, M. Leguia, J. C. Anderson, and **D. Densmore**, "Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems," ***PLoS ONE***, 2011.

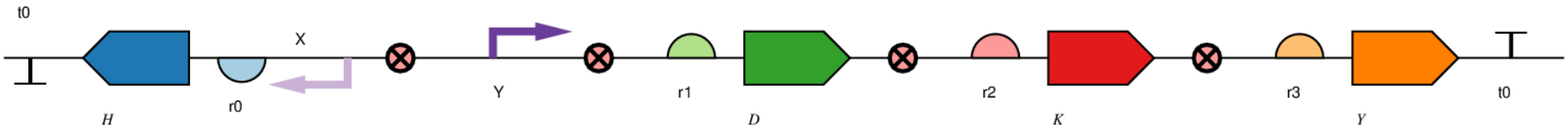
Conference

1. **D. Densmore**, J. T. Kittleson, L. Bilitchenko, A. Liu, and J. C. Anderson, "Rule based constraints for the construction of genetic devices," in ***ISCAS***, 2010.

Iteration 1:



Rule Rule1((PassMgfp WITH SpacerBeta-Roll) OR (PassMgfp WITH SpacerBeta-Helix) OR (PassMgfp WITH SpacerINP));
Rule Rule2((PassStrep WITH Disp_Vta) OR (PassStrep WITH Disp_ehaB) OR (PassStrep WITH Disp_CPG6) OR (PassStrep WITH Disp_AIDA));
Rule Rule3((PassNeedle NOTWITH Disp_cl) AND (PassNeedle NOTWITH Disp_Pcryo) (PassNeedle NOTWITH Disp_CPG6));
Rule Rule4(NOTCONTAINS PassMgfp);
Rule Rule5((PassStrep WITH Disp_CPG6) OR (PassStrep WITH Disp_AIDA));
Rule Rule6(PassNeedle WITH Disp_upaG);



6. There is no terminator between NifD and K

$$\forall x_0 \forall x_1 \forall x_2 (x_2 < x_1) \wedge (x_2 \text{ is a Terminator})$$

$$\begin{aligned}
&\text{t0}(0) \wedge \text{Terminator}(0) \wedge \text{reverse}(0) \wedge \\
&\text{H}(1) \wedge \text{CDS}(1) \wedge \text{reverse}(1) \wedge \\
&\text{r0}(2) \wedge \text{5UTR}(2) \wedge \text{reverse}(2) \wedge \\
&\text{X}(3) \wedge \text{Promoter}(3) \wedge \text{reverse}(3) \wedge \\
&\text{s0}(4) \wedge \text{Spacer}(4) \wedge \text{both}(4) \wedge \\
&\text{Y}(5) \wedge \text{Promoter}(5) \wedge \text{forward}(5) \wedge \\
&\text{s0}(6) \wedge \text{Spacer}(6) \wedge \text{both}(6) \wedge \\
&\text{r1}(7) \wedge \text{5UTR}(7) \wedge \text{forward}(7) \wedge \\
&\text{D}(8) \wedge \text{CDS}(8) \wedge \text{forward}(8) \wedge \cdots
\end{aligned}$$

7. There is

Similar to 6 above.

8. There is no promoter between nif K and Y

Similar to 7 above.

9. All genes are oriented in the same direction

$$\forall x_0 \forall x_1 \ (\text{cds}(x_0) \wedge \text{cds}(x_1)) \implies ((\text{forward}(x_0) \wedge \text{forward}(x_1)) \vee (\text{reverse}(x_0) \wedge \text{reverse}(x_1)))$$

4. NifH has RBS z

11. Not all genes have a promoter upstream of their RBS

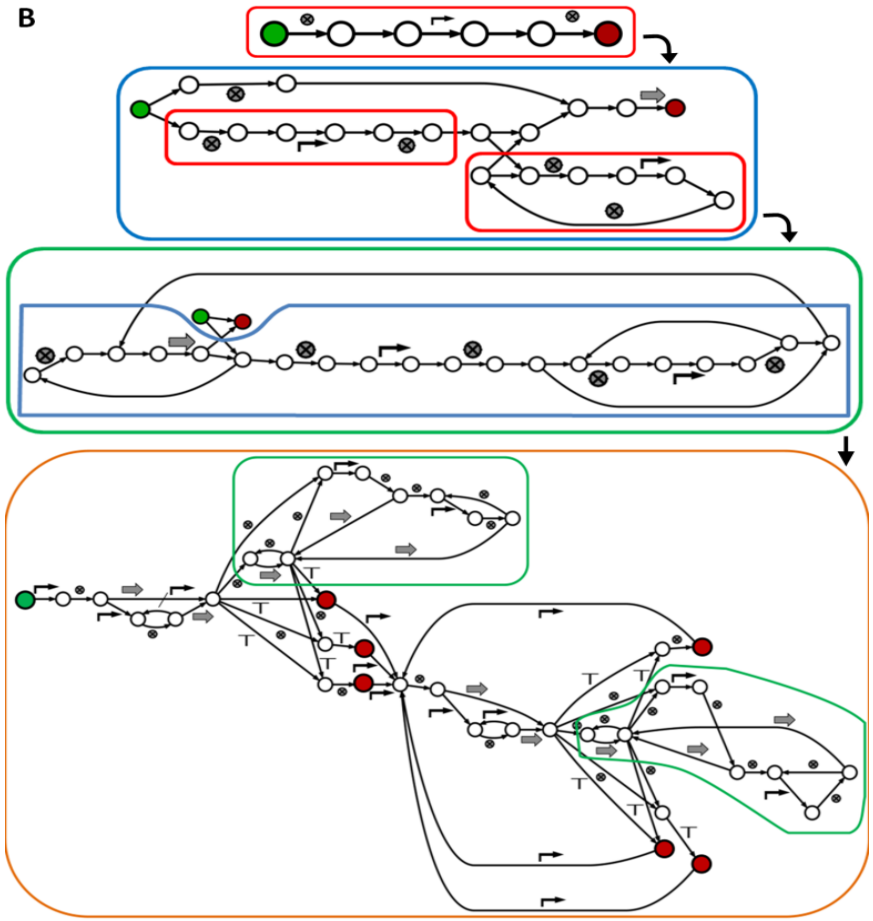
12. Not all genes are followed immediately by a terminator

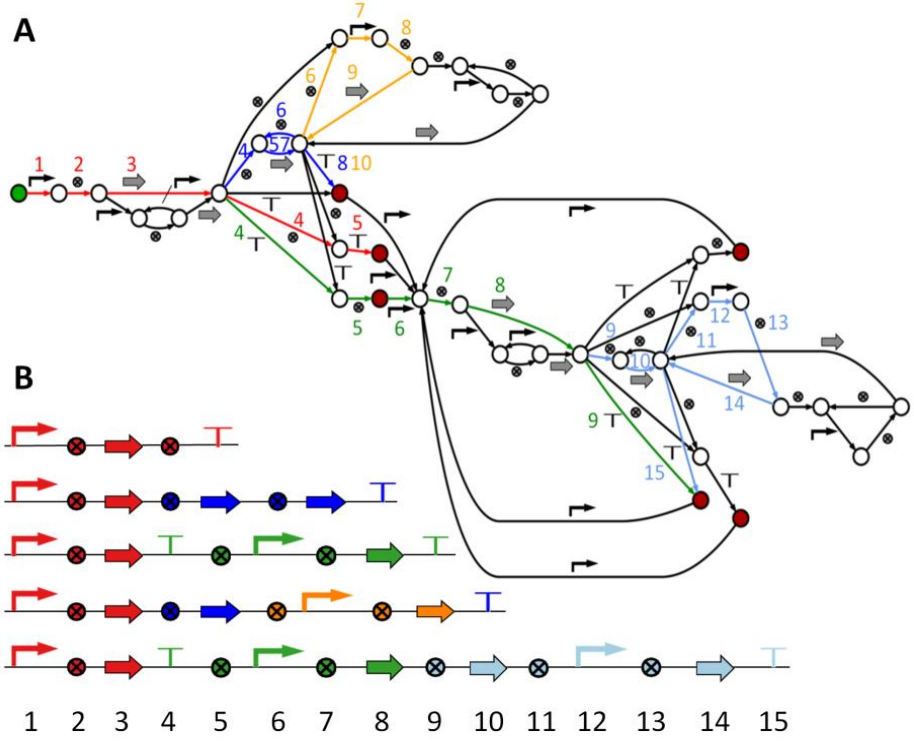
$$\exists x_0 \forall x_1 \forall x_2 \ \text{NifH}(x_1) \implies \text{RBSZ}(x_0) \wedge (x_0 < x_1) \wedge (\neg(x_0 < x_2 \wedge x_2 < x_1))$$

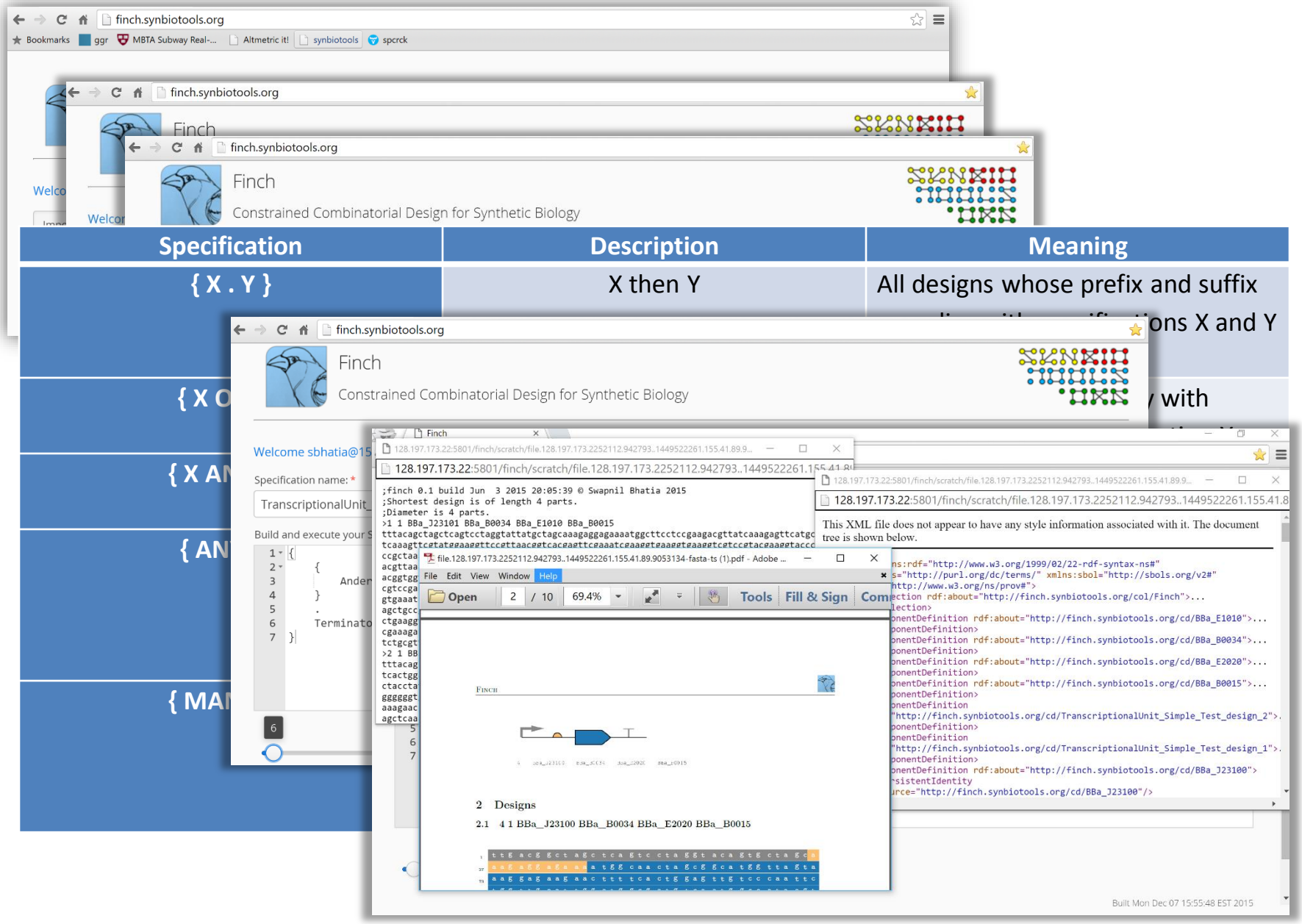
A

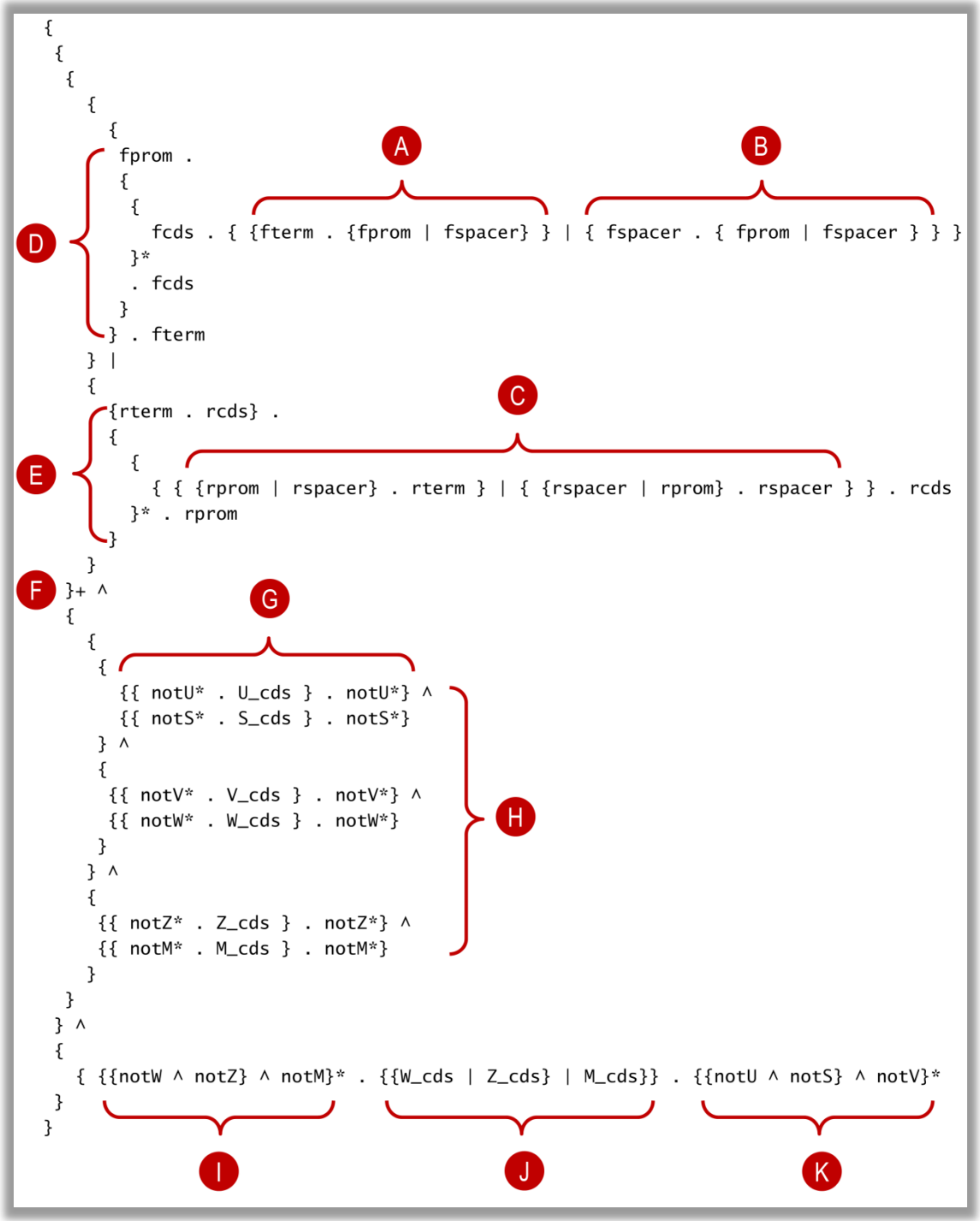
```
one-or-more(
   one-or-more(prom_f then spacer_f) then cds_f then
      zero-or-more(
      spacer_f or one-or-more(
            spacer_f then prom_f then spacer_f
      ) then cds_f
   ) then
   term_f or (term_f then spacer_f) or (spacer_f then term_f)
)
```

B






finch.synbiotools.org
Finch
Constrained Combinatorial Design for Synthetic Biology
Specification
Description
Meaning
{ X . Y }
X then Y
All designs whose prefix and suffix
Welcome sbhatia@15
Specification name: *
TranscriptionalUnit_
Build and execute your S
;finch 0.1 build Jun 3 2015 20:05:39 © Swapnil Bhatia 2015
;Shortest design is of length 4 parts.
;Diameter is 4 parts.
>1 1 BBa_J23101 BBa_B0034 BBa_E1010 BBa_B0015
>2 1 BB
This XML file does not appear to have any style information associated with it. The document tree is shown below.
2 Designs
2.1 4 1 BBa_J23100 BBa_B0034 BBa_E2020 BBa_B0015
Built Mon Dec 07 15:55:48 EST 2015



The specification for the nif six-gene subcluster design space is shown. For brevity, we use symbols **.**, **^**, **|**, *, and **+** for the operators **then**, **and**, **or**, **zero-or-more**, and **one-or-more**. **(A)** In an operon, a CDS may be followed by a terminator and promoter or a terminator and a spacer. **(B)** Alternatively, it may be followed by a promoter flanked by spacers. **(C)** Analogous constraints apply if the operon is oriented in the reverse direction. **(D)** A forward oriented operon comprises zero of more CDS with terminators or promoters and spacers as specified above, followed by a CDS and a terminator, and preceded by a promoter. **(E)** The structure of a reverse oriented operon is analogous. **(F)** A design may have one or more such forward or reverse oriented operons. **(G)** A design must contain exactly one copy of the nifU CDS. **(H)** Single copy constraints are specified for all six genes. **(I)** The design must contain a prefix that is free of the nifW, nifZ, and nifM genes. **(J)** This prefix must be followed by a segment containing the nifW, nifZ, or nifM genes. **(K)** The design must end with a suffix not containing the nifU, nifS, and nifV genes. Thus, in effect, the USV genes must not follow the WZM genes.

Functional Specification Approach

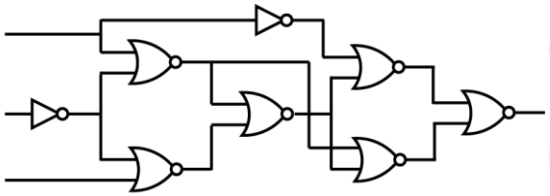
JavaScript/HTML:
Cello web application

Verilog:
high-level input

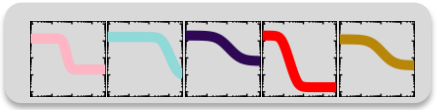
Computational approach

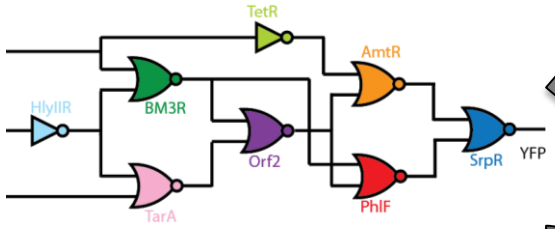
Experimental approach

ESPRESSO, ABC, Java:
logic synthesis

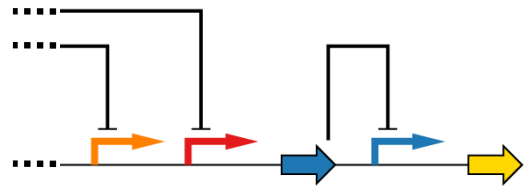


MySQL:
genetic parts database

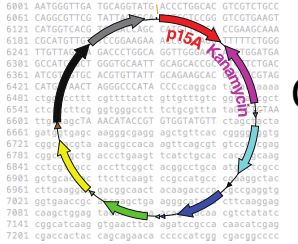




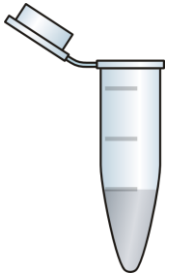
Java:
GENETIC COMPILER



Eugene:
combinatorial plasmid design



Geneart:
Commercial DNA synthesis of circuit designs



Flow cytometry and **RNA-Seq**: circuit measurements

High-level user input

Verilog for high-level user input:

- Hardware description language for compact description of large circuits
- Mature language, used for decades
- Genetic knowledge not required
- Avenue for electronic design experts to enter synthetic biology

Any combinational logic function can be converted into a truth table

```
module AND2
          2'b00: out = 1'b0;
          2'b01: out = 1'b0;
          2'b10: out = 1'b0;
          2'b11: out = 1'b1;
```

Case

```
module AND2
          assign out = in1 &
in2;
```

Assign

```
module AND2
          if (in1 & in2)
begin
                    out =
1'b1;
          end
          else begin
                    out =
1'b0;
          end
```

If/Else

```
module AND2
          NOT (in1, wire1);
          NOT (in2, wire2);
          NOR (out, wire1,
wire2)
```

Purely structural

Logic synthesis (NOT/NOR)

Truth table

Inputs		Output
A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

(A’•B) + (A•B’)

(x + y) = (x’ • y’)’

((A’•B)’•(A•B’)’)’

S.O.P.

A.I.G.

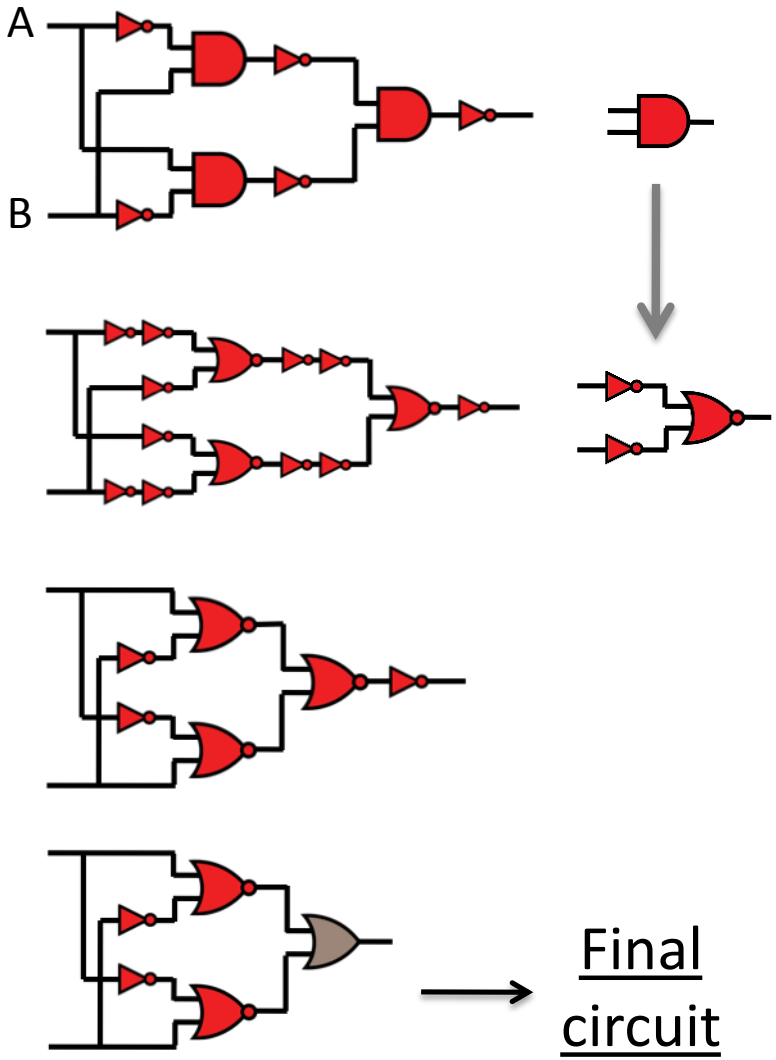
(A’•B)
+
(A•B’)

AND-Inverter Graph

Map to NOR/NOT

Remove double inv.

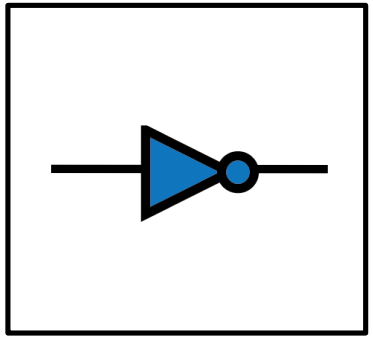
OUTPUT_OR



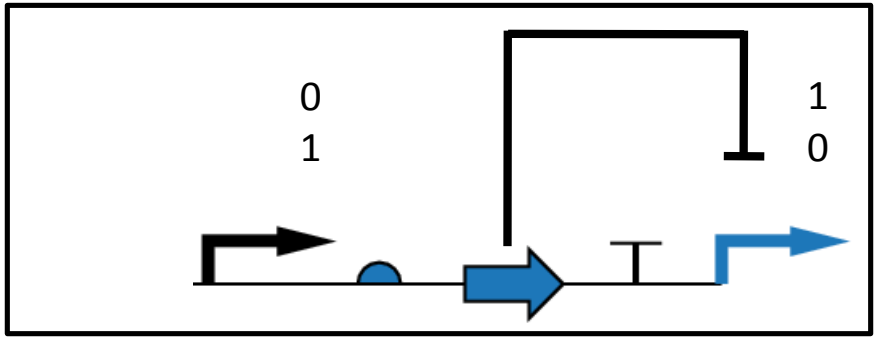
Final circuit

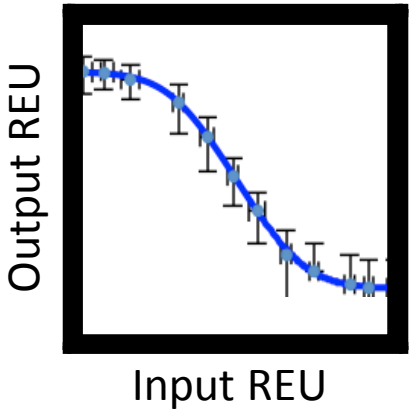
Genetic NOT/NOR gates

NOT

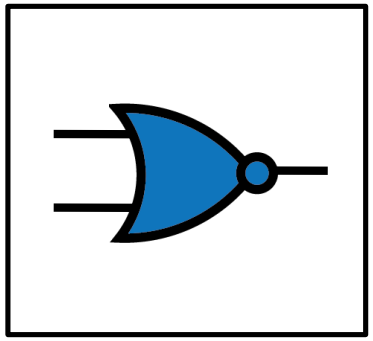


Genetic NOT gate

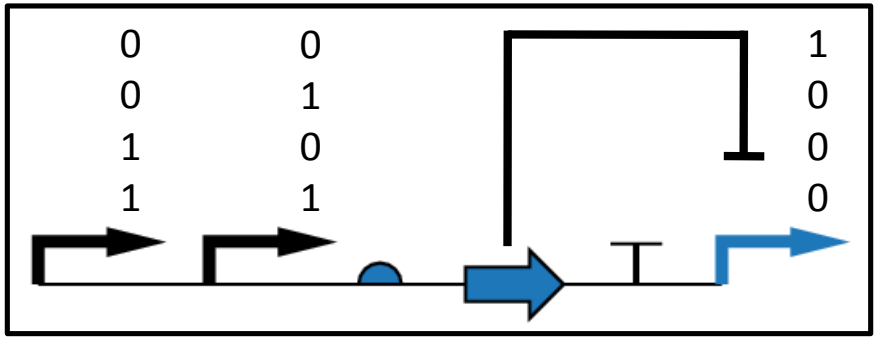


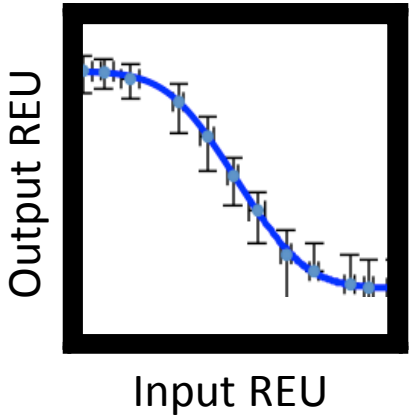


NOR

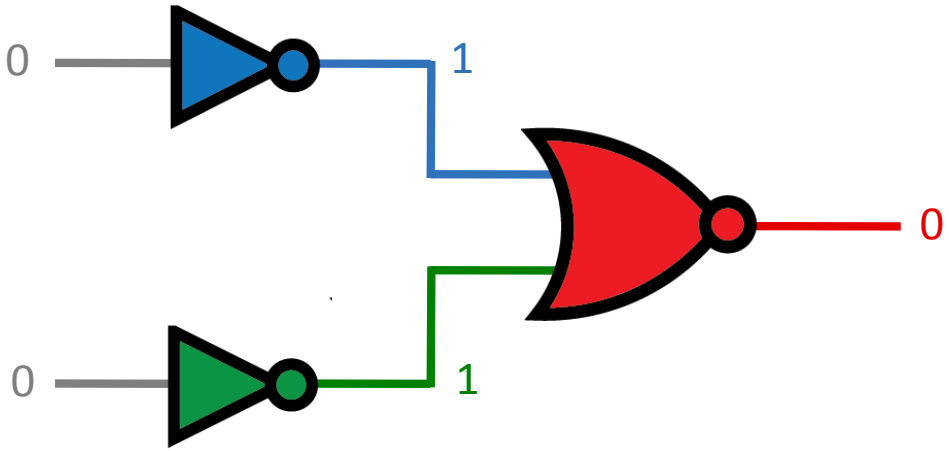


Genetic NOR gate

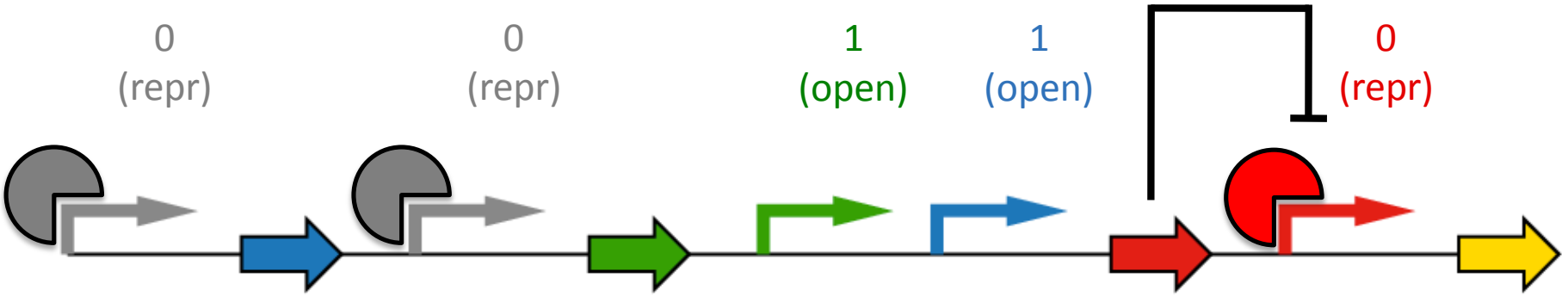




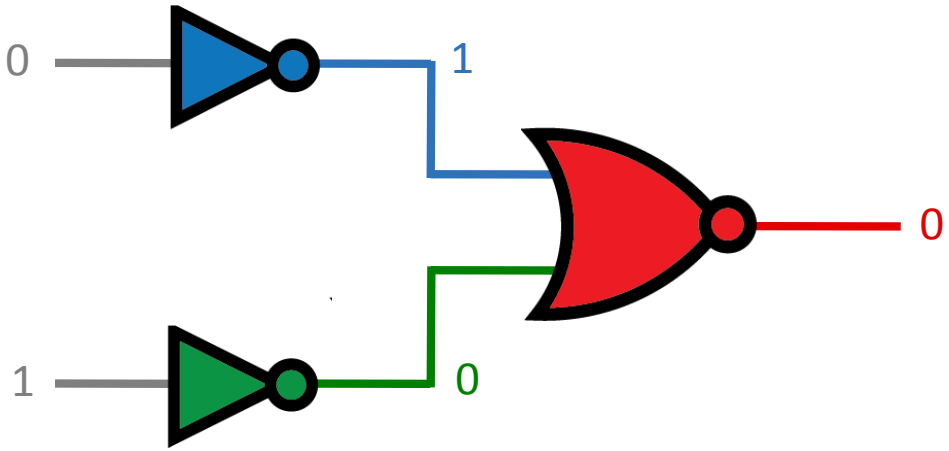
Digital abstraction for transcriptional logic



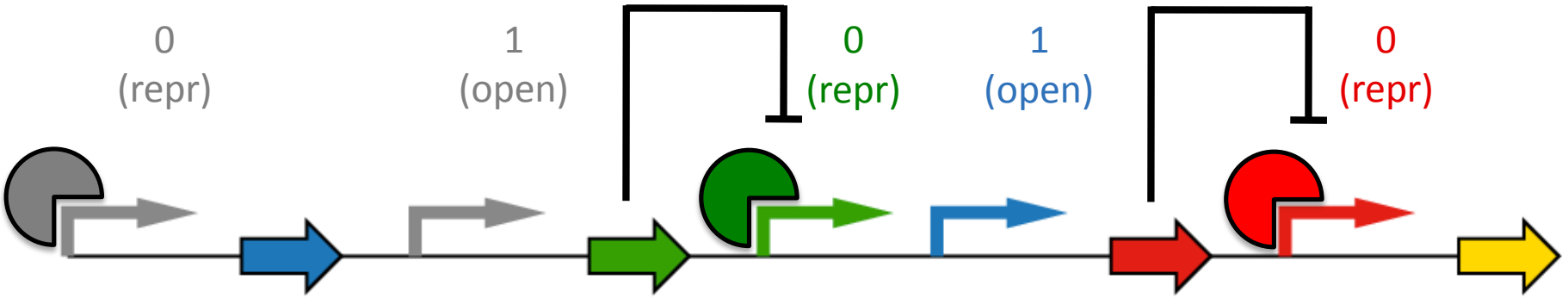
A	B	Out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



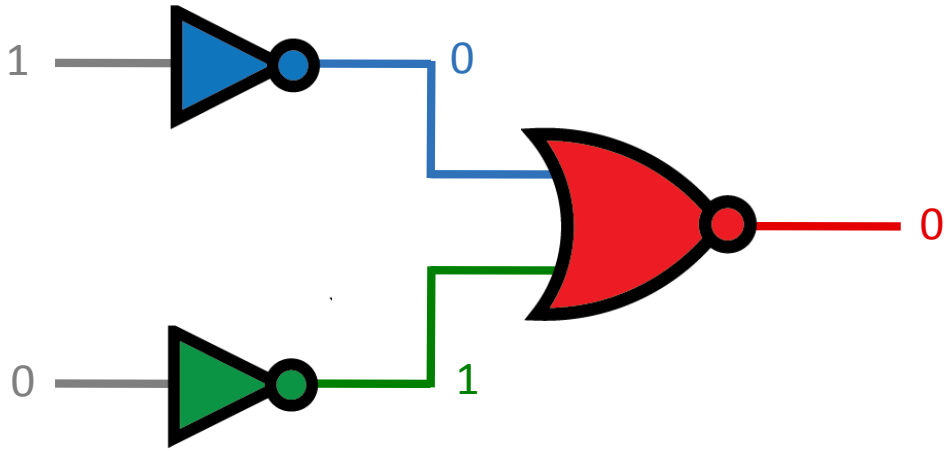
Digital abstraction for transcriptional logic



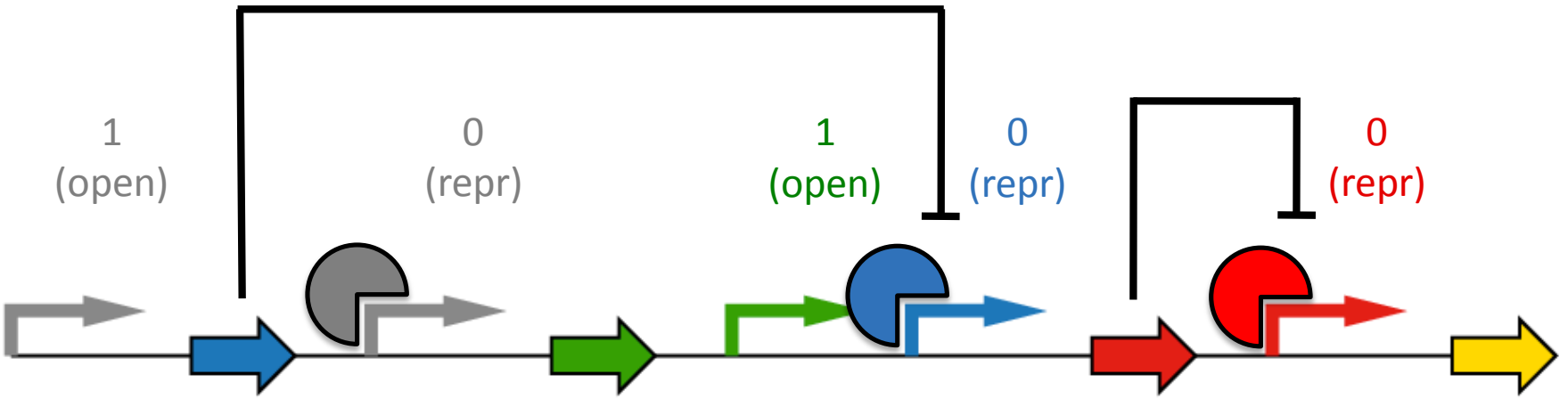
A	B	Out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



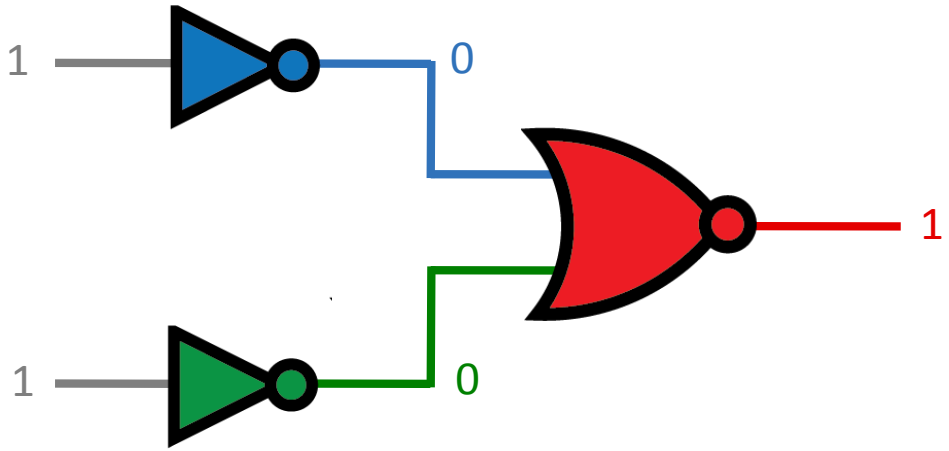
Digital abstraction for transcriptional logic



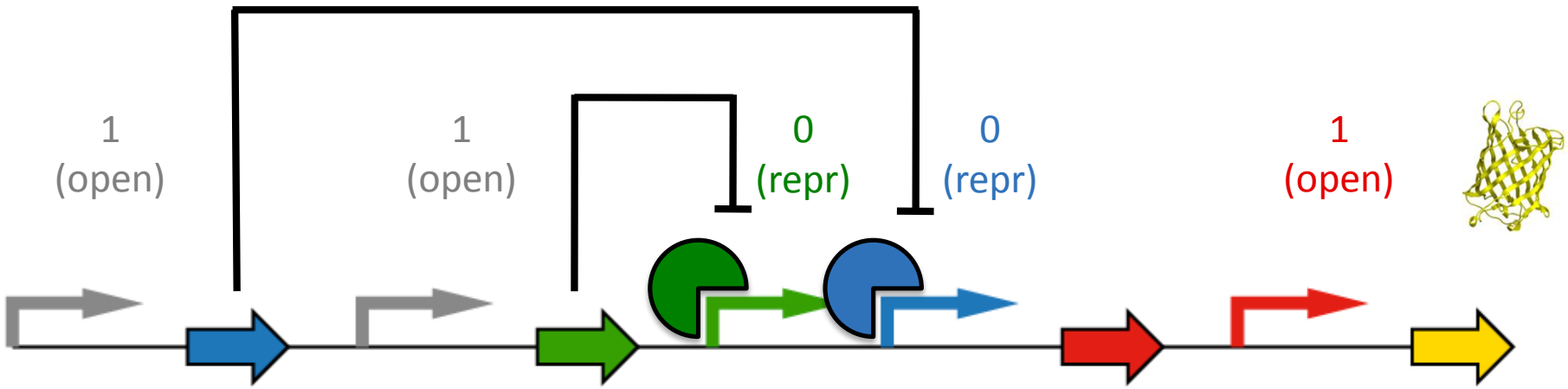
A	B	Out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

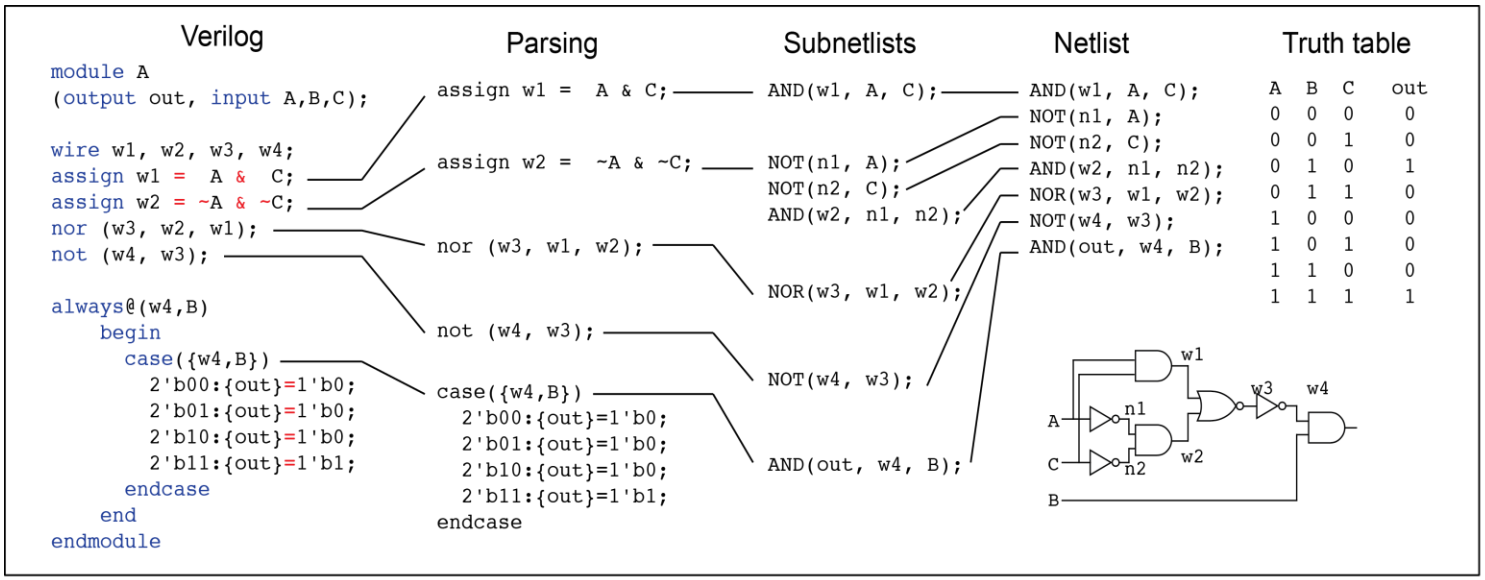


Digital abstraction for transcriptional logic



A	B	Out
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



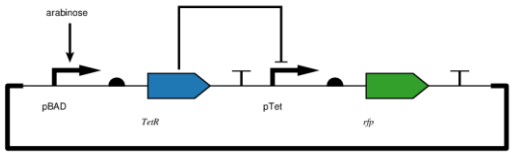

Verilog
Parsing
Subnetlists
Netlist
Truth table
module A
(output out, input A,B,C);
wire w1, w2, w3, w4;
assign w1 = A & C;
assign w2 = ~A & ~C;
nor (w3, w2, w1);
not (w4, w3);
always@(w4,B)
begin
case({w4,B})
2'b00:{out}=1'b0;
2'b01:{out}=1'b0;
2'b10:{out}=1'b0;
2'b11:{out}=1'b1;
endcase
end
endmodule
assign w1 = A & C;
assign w2 = ~A & ~C;
nor (w3, w1, w2);
not (w4, w3);
case({w4,B})
2'b00:{out}=1'b0;
2'b01:{out}=1'b0;
2'b10:{out}=1'b0;
2'b11:{out}=1'b1;
endcase
AND(w1, A, C);
NOT(n1, A);
NOT(n2, C);
AND(w2, n1, n2);
NOR(w3, w1, w2);
NOT(w4, w3);
AND(out, w4, B);
AND(w1, A, C);
NOT(n1, A);
NOT(n2, C);
AND(w2, n1, n2);
NOR(w3, w1, w2);
NOT(w4, w3);
AND(out, w4, B);
A B C out
0 0 0 0
0 0 1 0
0 1 0 1
0 1 1 0
1 0 0 0
1 0 1 0
1 1 0 0
1 1 1 1
w1
w3
w4
n1
A
w2
C
n2
B

Performance Specification

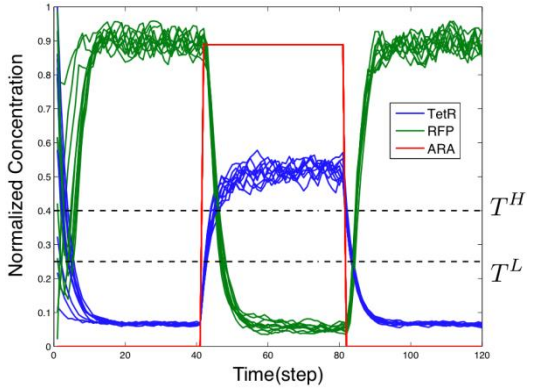
- Signal Temporal Logic (STL) is a *formal method* to study system behavior with respect to complex specifications
- With STL, we can formally specify desired function and performance

EXAMPLE

- *Function*: Inverter (NOT gate)



- *English Specification*: If the *TetR* signal is higher than T_H, the *rfp* signal is expected to become lower than T_L within a time period of k_1 and persist for a time period of k_2.



- *STL Specification*: $(x_{TetR} > T_H) \mathrel{\mathrm{P}} (F_{[0,k_1)} G_{[0,k_2)} x_{rfp} < T_L)$

- Other typical functions such as ‘toggle switch’ and ‘oscillator’ can also be translated into STL formulae.

Metrics Summary

Name	Type	Description	Infrastructure
Evolutionary Stability	Quality	Number of generations over which the output stays within 3-fold of the output at time t=0.	eVOLVER and flow cytometry
Signal to Noise Ratio	Quality	Separation of high and low output.	Cello and Phoenix
Propagation Delay	Time	Time between valid inputs and outputs.	Phoenix
DNA Design Size	Space	Number of DNA base pairs in a design.	Phoenix, Raven, Finch
DNA Delivery Size	Space	Number of designs introduced into the host organism.	Phoenix, Raven, Finch
Regulatory Interactions	Complexity	Number of interactions between regulatory regions of DNA.	BioCompiler, Phoenix
Functional Units	Complexity	Number of DNA functional units required for a design.	Cello, BioCompiler, Phoenix
Gene Products	Complexity	Number of genes translated in the course of a design execution.	Phoenix
Algorithm Complexity	Complexity	Asymptotic complexity as a function of the number of modules and parameters.	TBD
Load Driver Count	Complexity	Number of inserted load drivers based on signal transduction systems.	TBD
Degradation	Quality	Expected time over which the probability is at least p that a memory element will retain original value.	Flow Cytometry, Phoenix
Data Rate	Quality	Amount of biological data transmitted per unit time.	Phoenix
Response Time	Quality	Time from signal reception to phenotypic output response.	Flow Cytometry, Phoenix
Message Specificity	Quality	Ability to individually address individual biological elements.	Microfluidic test environment, Phoenix

Lecture 6

Specification – Structure, Constraints, Design Space

Prof. Douglas Densmore

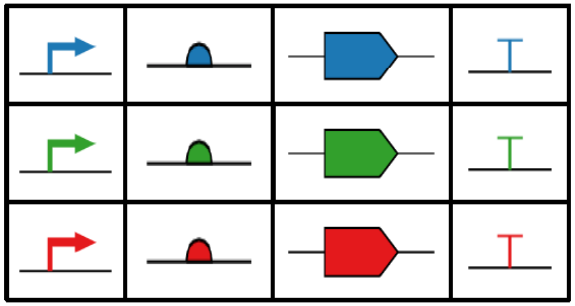
EC/BE552

Computational Synthetic Biology for Engineers

Lecture Objectives

- Describe a basic structural design enumeration environment
  - Eugene and Eugene Lab
- Describe a structural design space creation and generation environment
  - Finch
  - Knox

Design Space

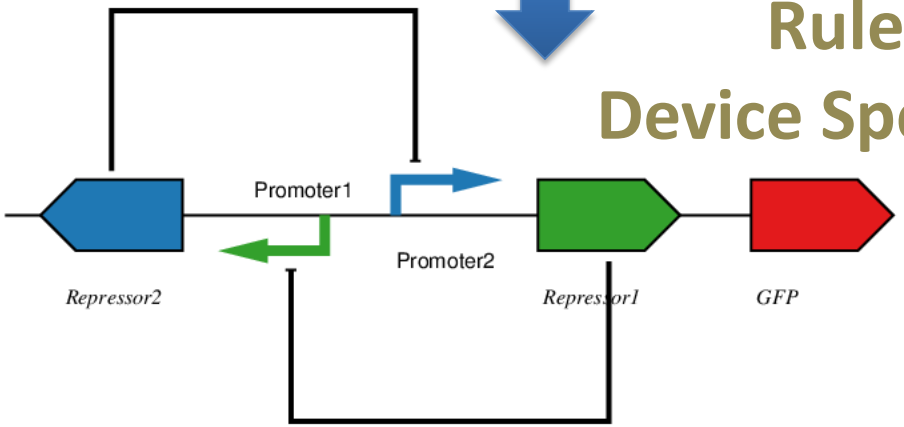


Rules/Constraints

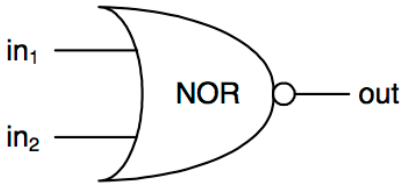
```
Device ToggleSwitch(
    Repressor, Promoter,
    Promoter, Repressor,
    Reporter);

Repressor1.represses == Promoter1
Repressor2.represses == Promoter2

Promoter1.direction == "backward"
```

Eugene is a design language for synthetic biology that defines **devices, part types**, **part properties**, and constrains their composition.

Rule-Compliant Device Specifications

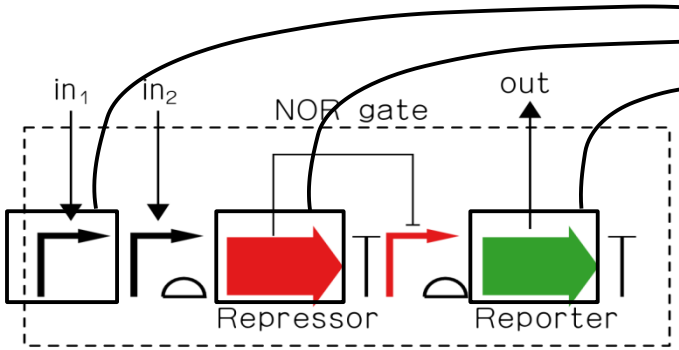


Biological Components



in_1	in_2	out
0	0	1
0	1	0
1	0	0
1	1	0

logical NOR Gate



genetic NOR Gate

// Properties
Property Name(txt);
Property Sequence(txt);
Property Represses(txt);
Property RepressedBy(txt);

// Part Types
Part InduciblePromoter(Name, Sequence);
Part RepressiblePromoter(Name, Sequence, RepressedBy);
Part Repressor(Name, Sequence, Represses);
. . .

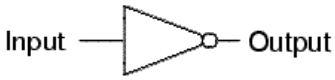
// Device
Device NORGate(
  InduciblePromoter,
  InduciblePromoter, . . .,
  Repressor, . . .);

// Design Space
include DesignSpace.h;

`DesignSpace.h`

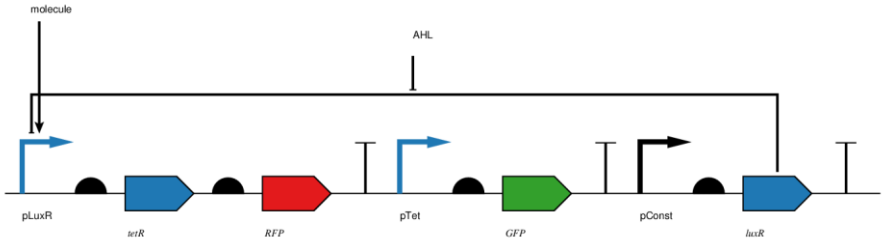
```
InduciblePromoter  pBAD(.Name("BBa_K206000"),.Sequence("acatt...tagc"));

Repressor cI(.Name("BBa_C0051"),.Sequence("atgag...aataa"),.Represses("pCI"));

RBS rbs61100("J61100","tctagaGAAAGAGGGGACAAactagt");

RepressiblePromoter  pCI("BBa_R0051","taaca...gttgc");

Reporter GFP("BBa_I13522","tccct...ttata");

Terminator BBa_B0010("BBa_B0010","nnnnc...");
```

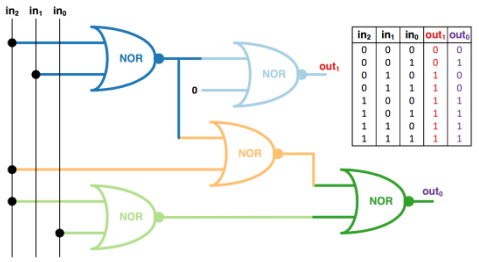
Logic and Genetic Gates/Circuits



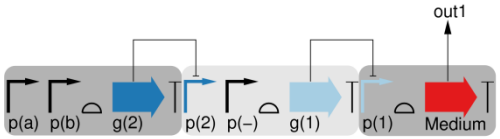
Inverter

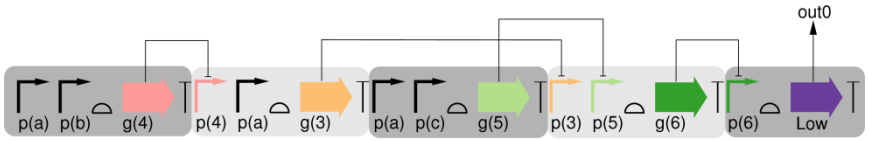
Input	Output
1	0
0	1

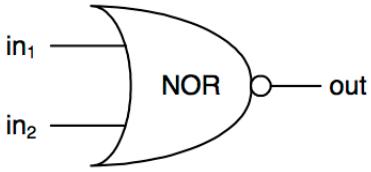




Priority Encoder

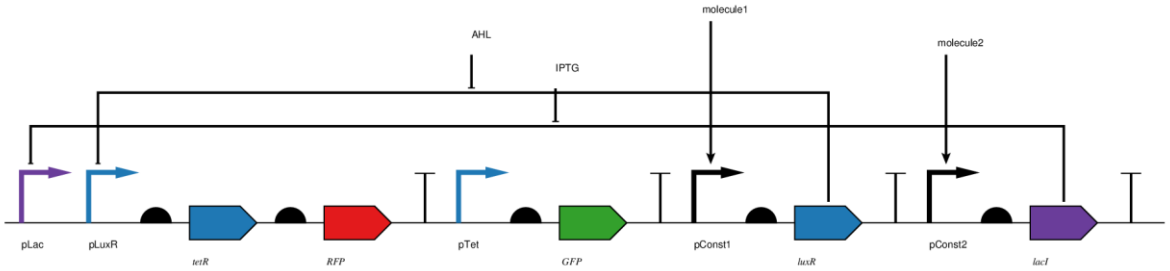






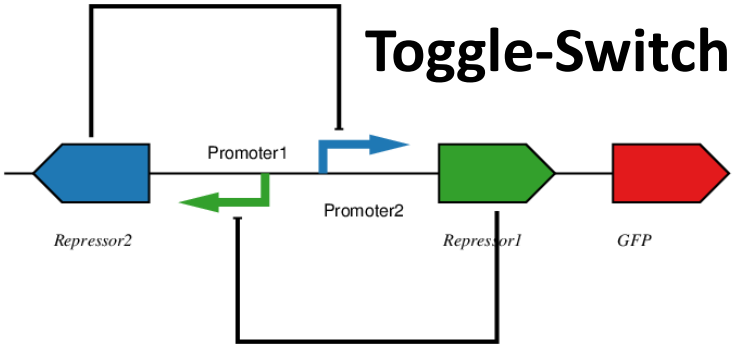
in_1	in_2	out
0	0	1
0	1	0
1	0	0
1	1	0

NOR Gate

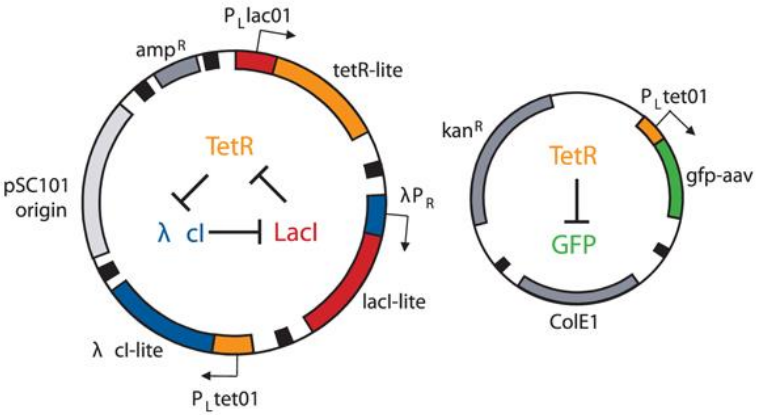


Designs from the Literature

Toggle-Switch

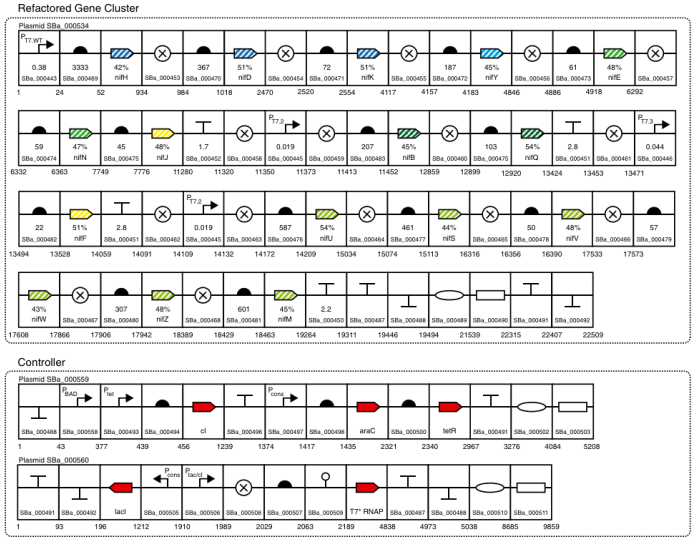


Repressilator



Nitrogen Fixation

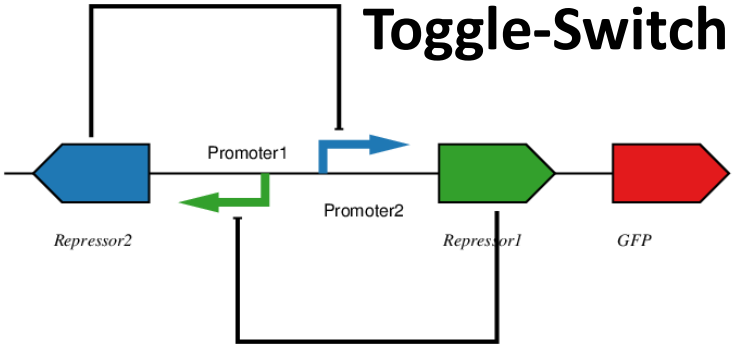
(doi: 10.1073/pnas.1120788109)



Eugene Example:

Design Space

```
// Specification of Properties
Property name(txt);
Property sequence(txt);
Property represses(txt);

// Specification of Part Types
Part Promoter(name);
Part Repressor(name, represses);
Part Reporter(name);

// Specification of Parts (Instances of Part Types)
Repressor repressor2("Repressor2", "Promoter2");
Promoter promoter1("Promoter1", "Repressor1");
Promoter promoter2("Promoter2", "Repressor2");
Repressor repressor1("Repressor1", "Promoter1");
Reporter GFP("GFP");
```




```
// Design of a Toggle Switch
Device ToggleSwitch(Repressor, Promoter, Promoter, Repressor, Reporter);

// Rules
Rule R01(ToggleSwitch[0].Represses EQUALS ToggleSwitch[2].Name);
Rule R02(ToggleSwitch[3].Represses EQUALS ToggleSwitch[1].Name);

// Combinatorial Funtion (cartesian product)
Device[] lstToggleSwitches = product(ToggleSwitch, strict);
```

Eugene's Features

Data Exchange:

```
Device sbolNorGate = SBOL.import("nor-gate.sbol");

// import data from MIT's part registry
Device gbNorGate = Genbank.import("nor-gate.gb");

// import a genetic part from MIT's partsregistry
Part I0500 = Registry.import("BBa_I0500");

// tell Eugene to save the gbNORGate device
Genbank.export(gbNORGate)

// export data to a SBOL
SBOL.export(gbNORGate);
```

Function Prototyping:

```
// function prototype
function Device assembleParts(Part[] arrParts) {
    Device d;
    for(num i=0;i<arrParts.size(); i++) {
        d.add(arrParts[i]);
    }
    return d;
}
```

```
// function call
Device d = assembleParts([pBAD, pCI, LuxR]);
```

Conditional Statements:

```
// RULE DECLARATION
Rule r (d[3].Represses == d[5].Name);
if (r) {
    return true;
} else {
    return false;
}
```

Loops:

```
// FOR
for(num i=0; i<lstProduct.size(); i++) {
    print(lstProduct[i]);
}
```

Combinatorial Algorithms:

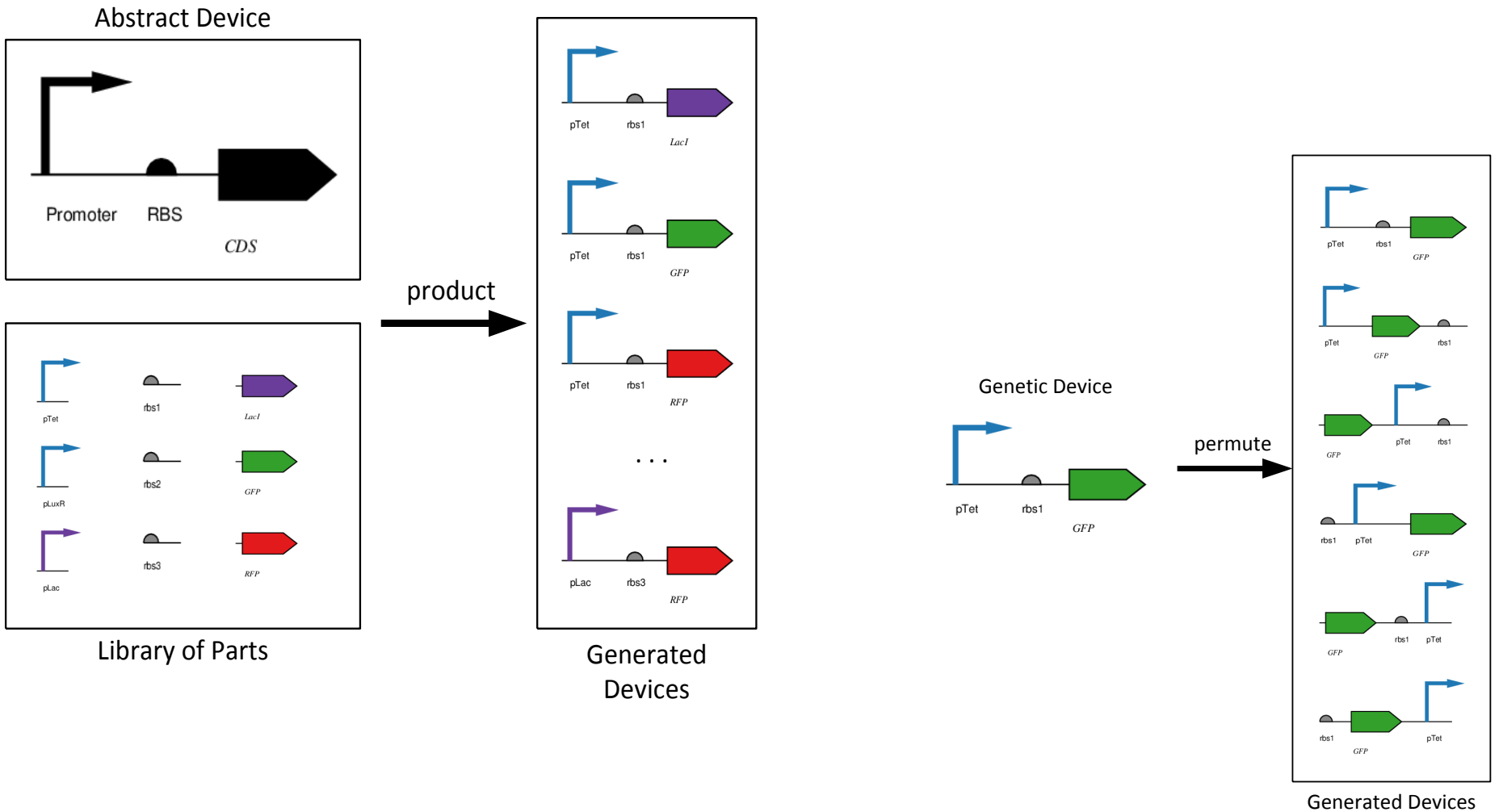
```
// cartesian product
Device[] lstProduct = product(NORGate);
```

```
// permutations
Device[] lstPermute = permute(NORGate);
```

RULES:

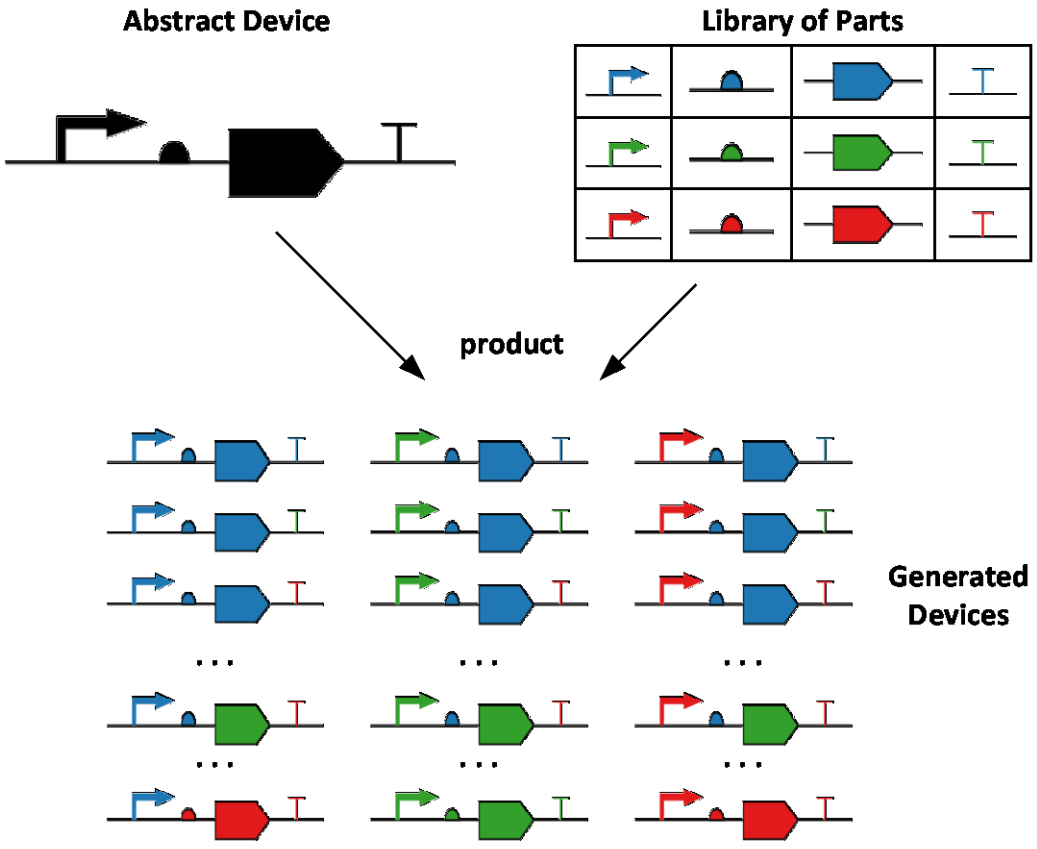
RULE OPERATOR
<left> **CONTAINS** <right>
<left> **AFTER** <right>
<left> **BEFORE** <right>
<left> **STARTSWITH** <right>
<left> **ENDSWITH** <right>
<left> **WITH** <right>
<left> **THEN** <right>
<left> **NEXTTO** <right>
<left> **MORETHAN** N
<left> ==, !=, >, >=, =<, < <right>

Combinatorial Functions



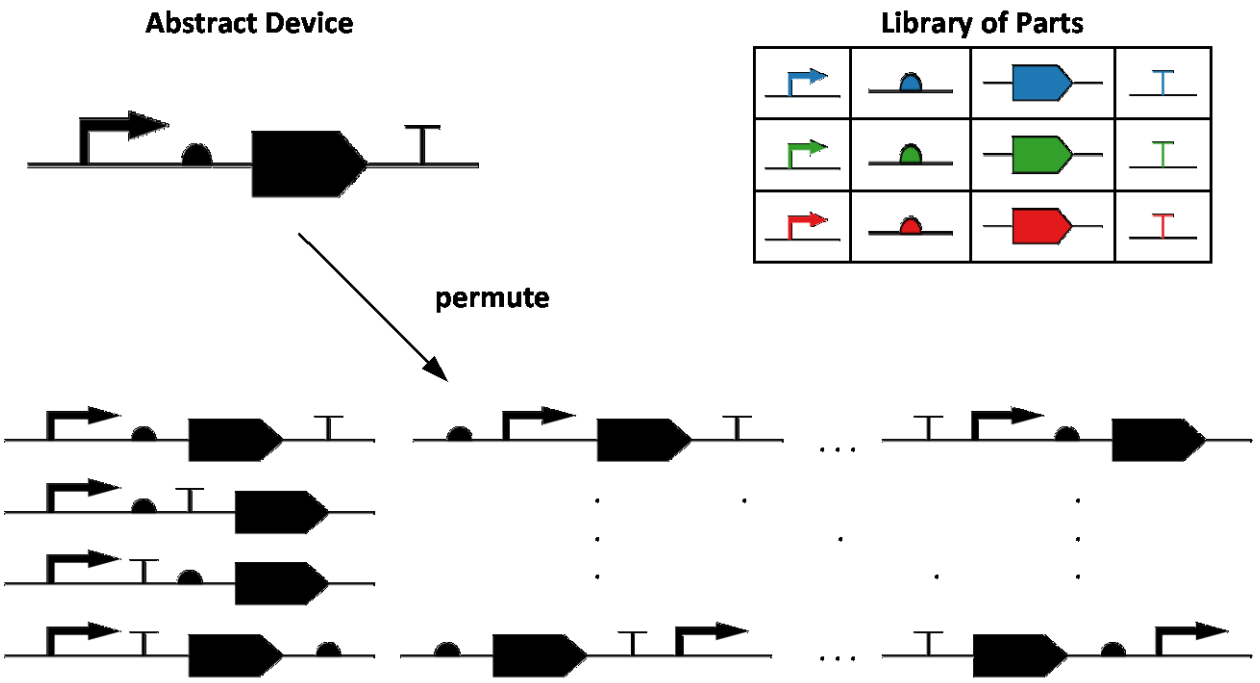
Challenges

- **Algorithm: Cartesian Product**



Challenges

- **Algorithm: Permutations**



Current Evaluation Results

Design Space:

- 5 Promoters
- 3 Repressors
- 8 RBSs
- 3 Reporters (xFPs)
- 4 Terminators

Evaluation Results:

Genetic Design	Size	Possible Designs	Rules	Valid Designs	LOC	Time [sec]
Inverter	14	230 400	4	16 128		4.78
NOR Gate	19	1 152 000	5	64 512		86.2
Priority Encoder	38	552 960 000				
Toggle Switch	5	675	2	12		0.5
Repressilator	8	50 625	3	24		1.57
Nitrogen Fixation	27					

Design Space Definition

Parts Library

C = { { pTac, pTet }, {s1, s2} } = { prom_f, spacer_f }

Swapnil Bhatia

Nicholas Roehner

Design Space Definition

Parts Library

C = { { pTac, pTet }, {s1, s2} } = { prom_f, spacer_f }

Abstract Design

prom_f,

spacer_f

Part Sets

Design Space Definition

Parts Library

C = { { pTac, pTet }, {s1, s2} } = { prom_f, spacer_f }

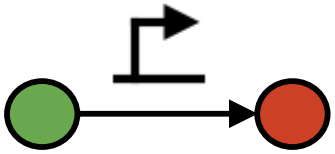
Abstract Design

prom_f, spacer_f *Part Sets*

or, **and then**, **one-or-more**, **zero-or-more**, *Operators*

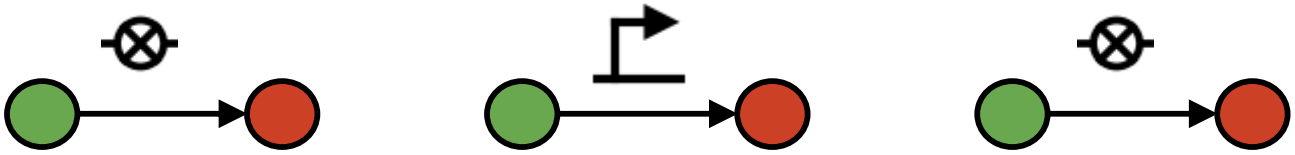
Design Space Construction

prom_f



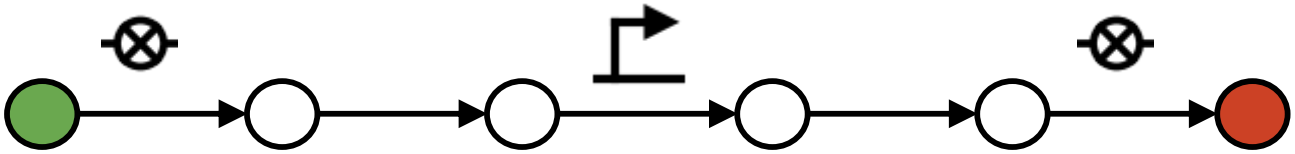
then Operator

spacer_f **then** prom_f **then** spacer_f



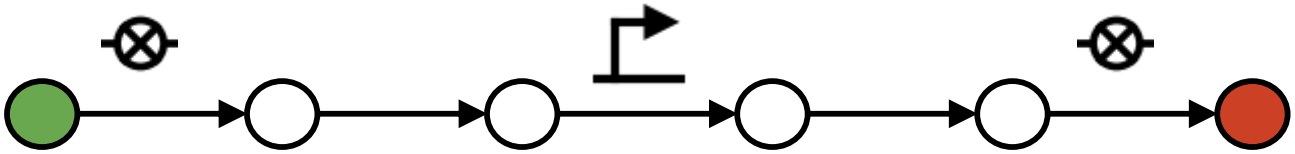
then Operator

spacer_f **then** prom_f **then** spacer_f



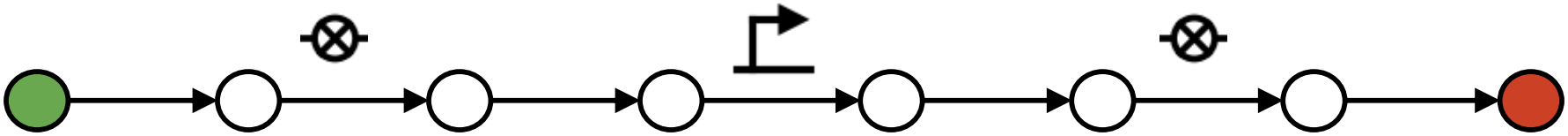
zero-or-more operator

zero-or-more (spacer_f **then** prom_f **then** spacer_f)



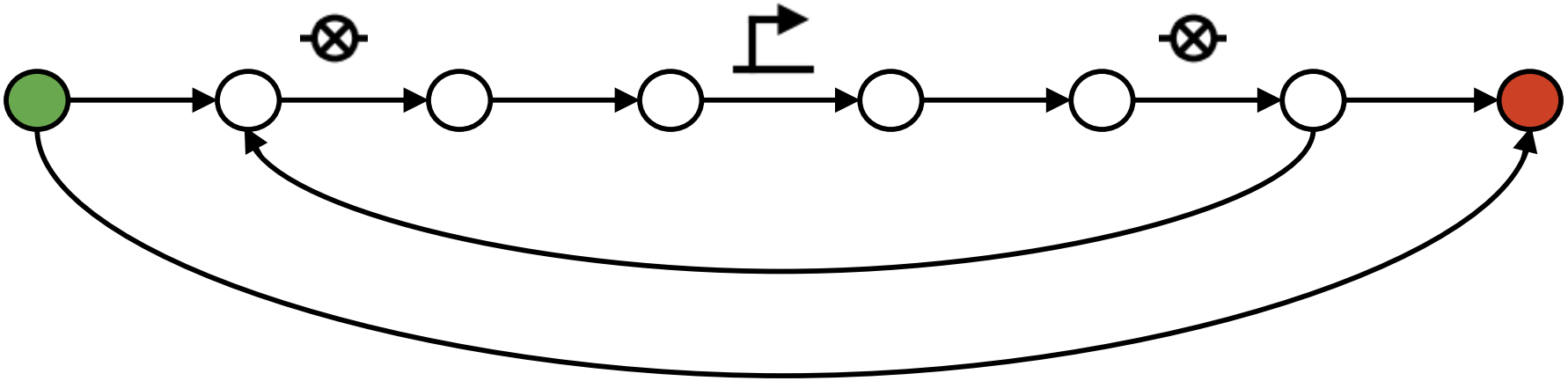
zero-or-more operator

zero-or-more (spacer_f **then** prom_f **then** spacer_f)



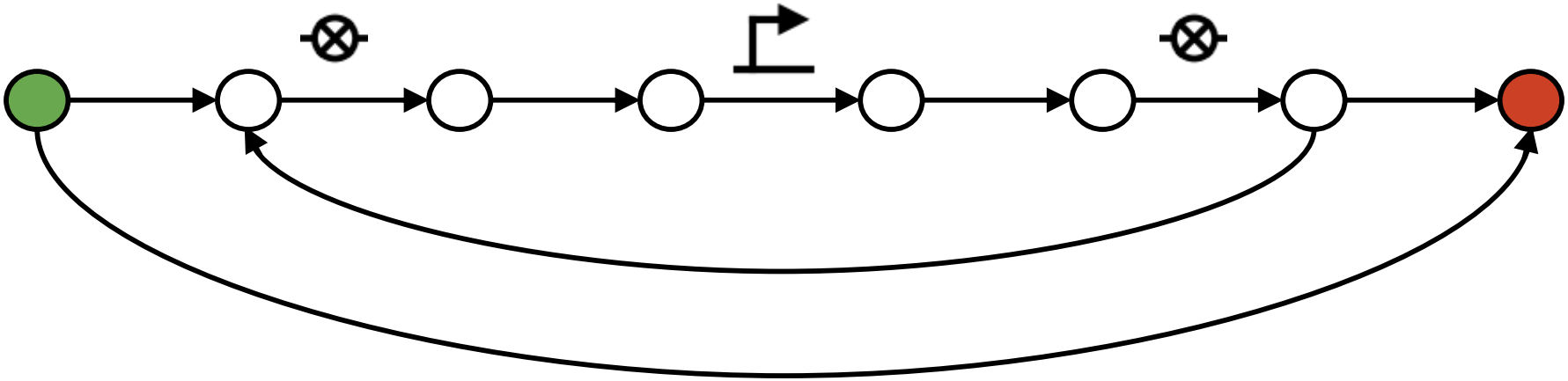
zero-or-more operator

zero-or-more (spacer_f **then** prom_f **then** spacer_f)

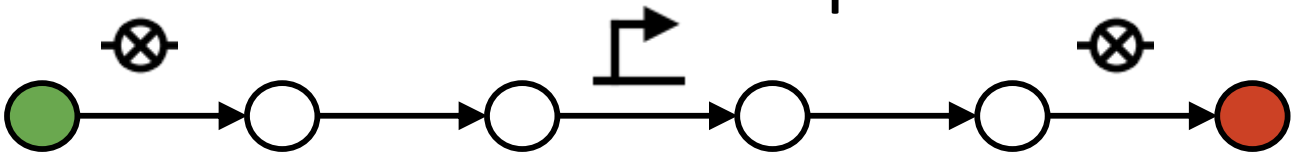


one-or-more operator

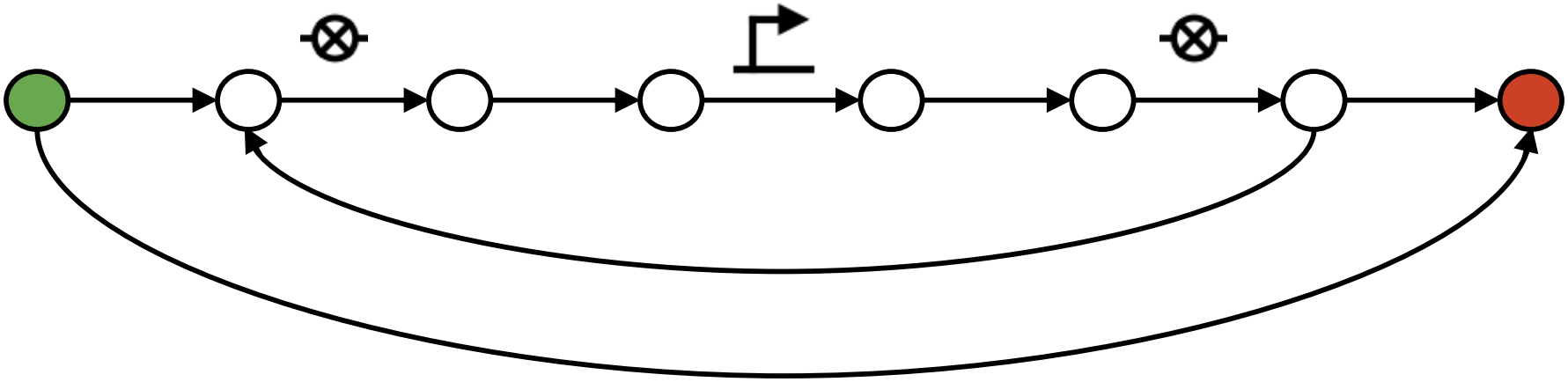
one-**or-more** (spacer_f **then** prom_f **then** spacer_f)



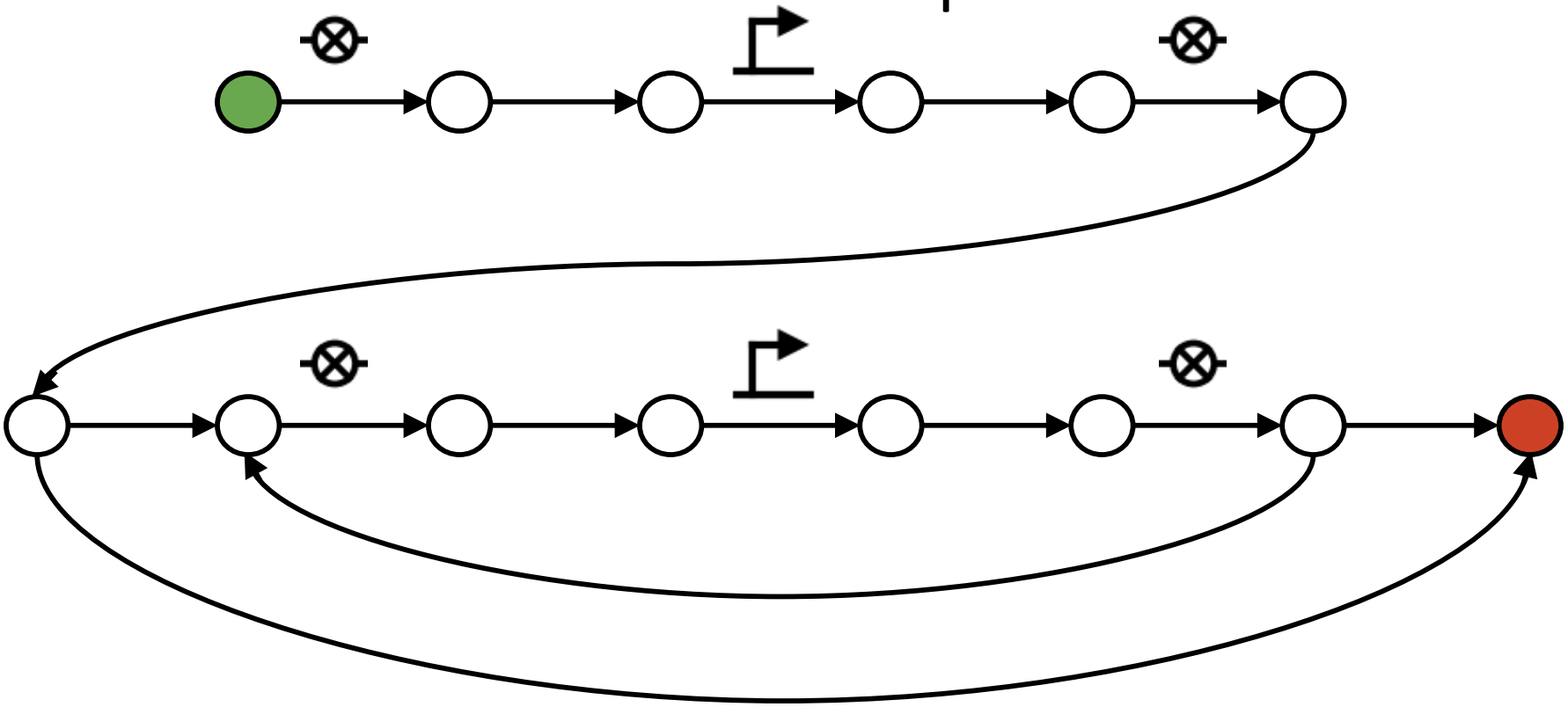
one-or-more operator



one-or-more (spacer_f **then** prom_f **then** spacer_f)



one-or-more operator

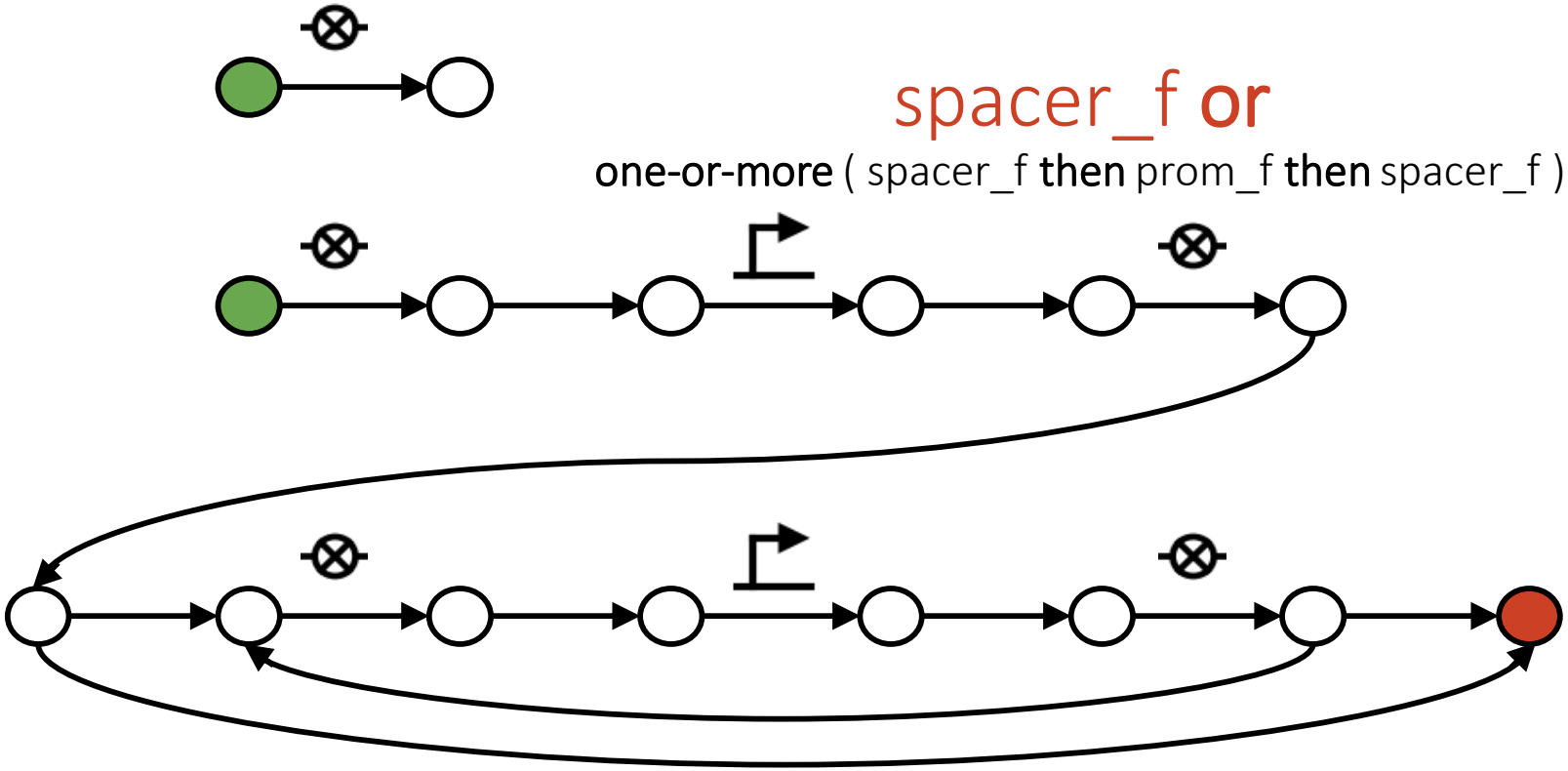


one-or-more (spacer_f **then** prom_f **then** spacer_f)

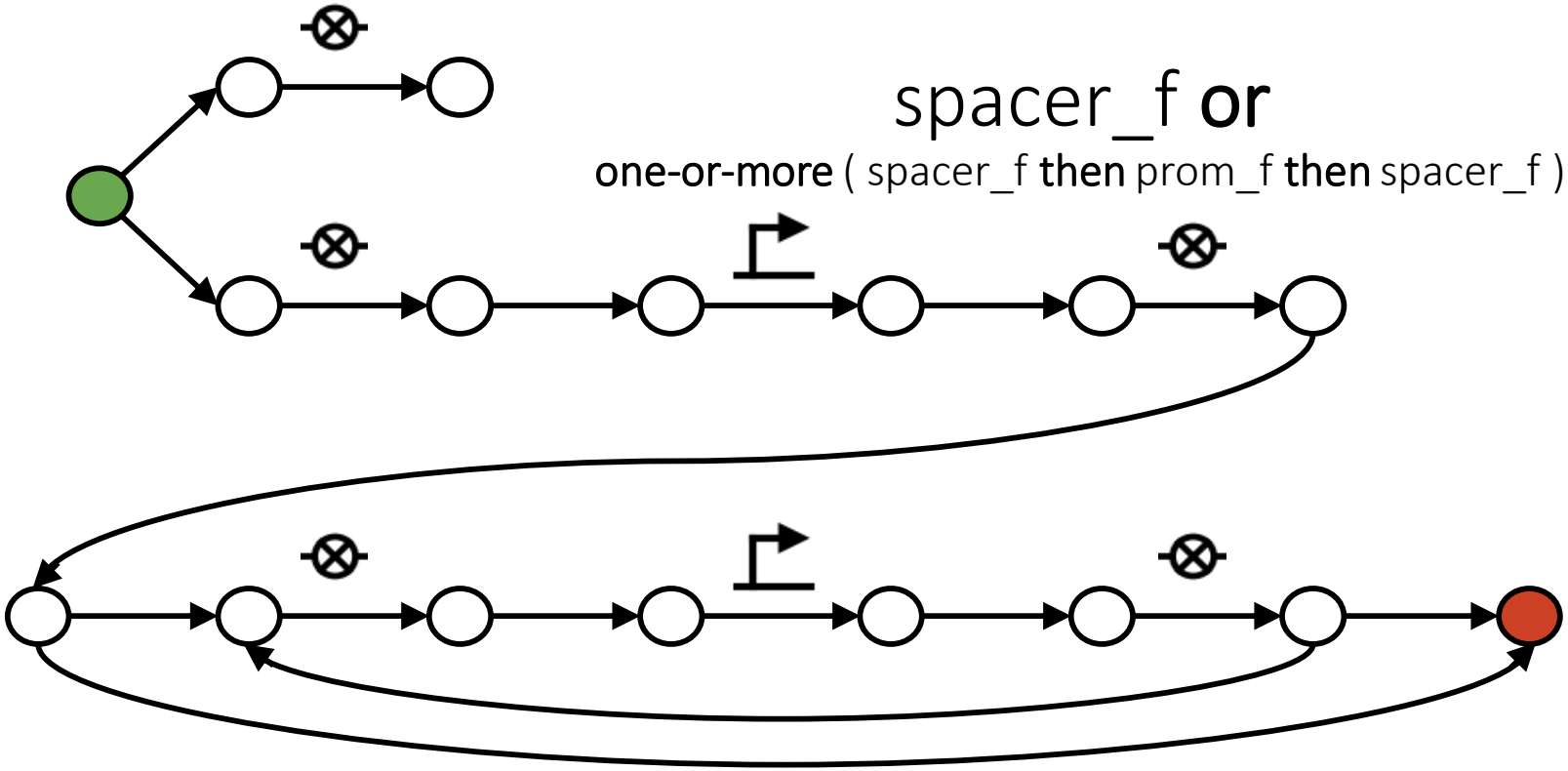
or operator

spacer_f **or**

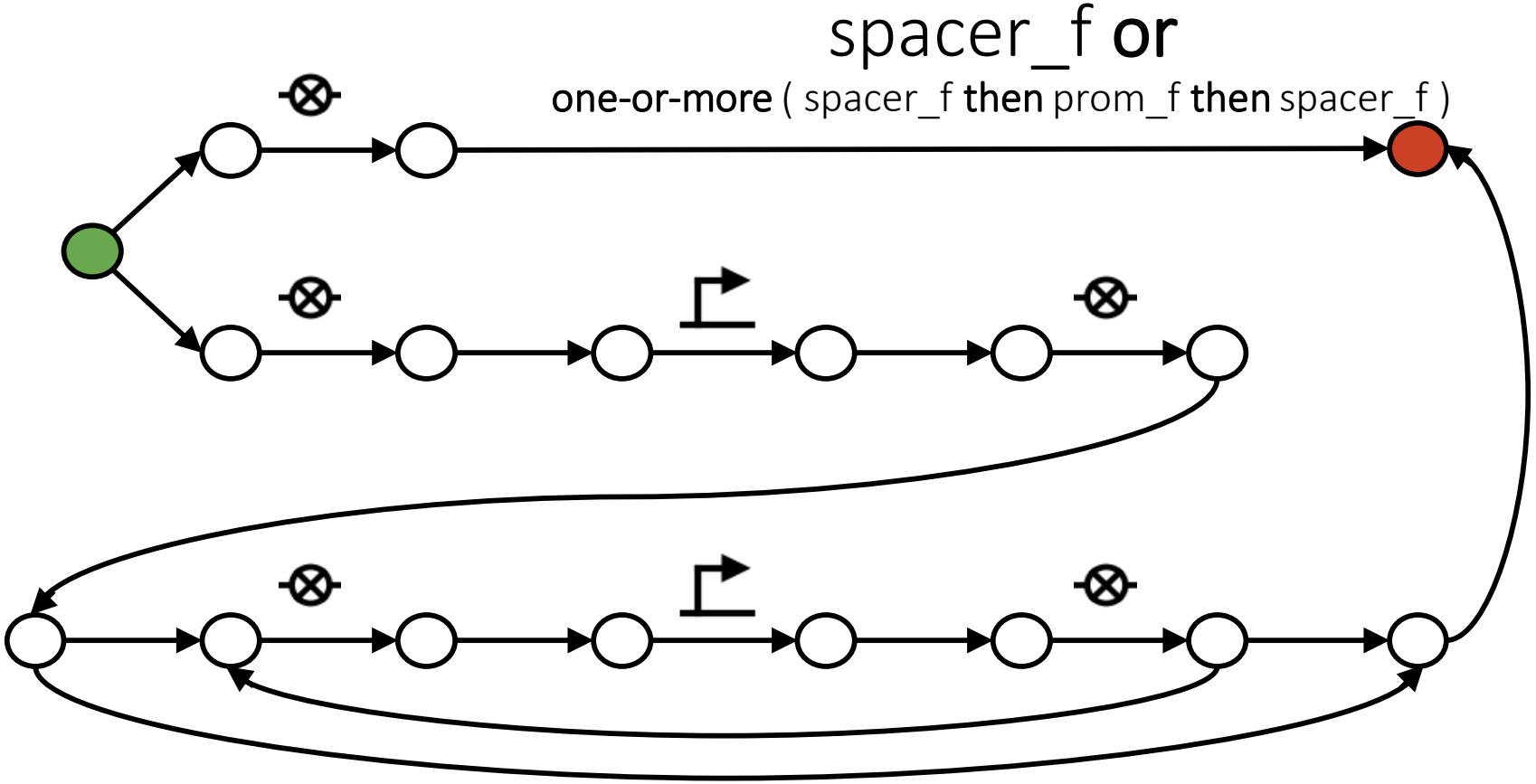
one-or-more (spacer_f **then** prom_f **then** spacer_f)



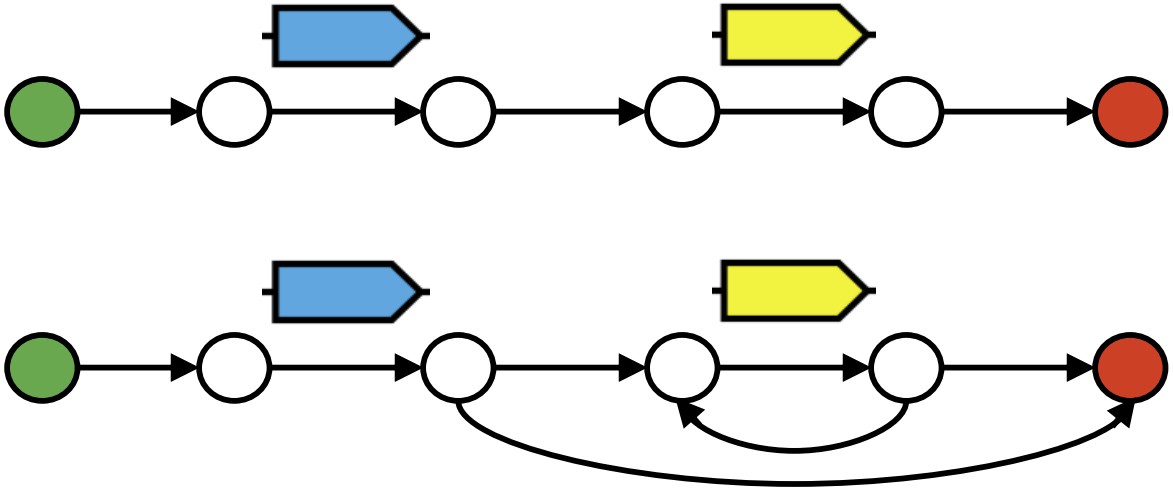
or operator



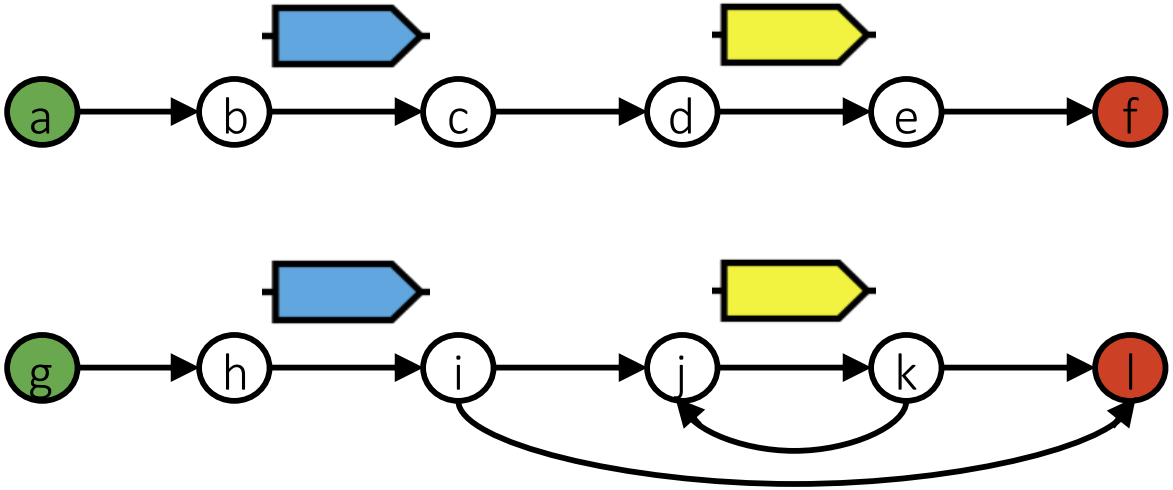
or operator



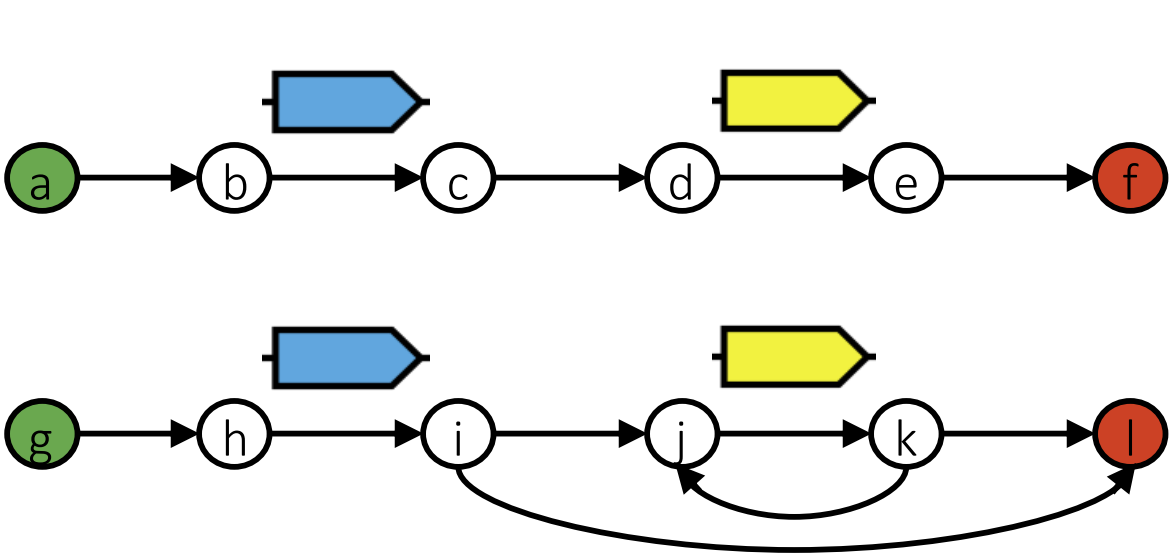
and operator (tensor product of graphs)

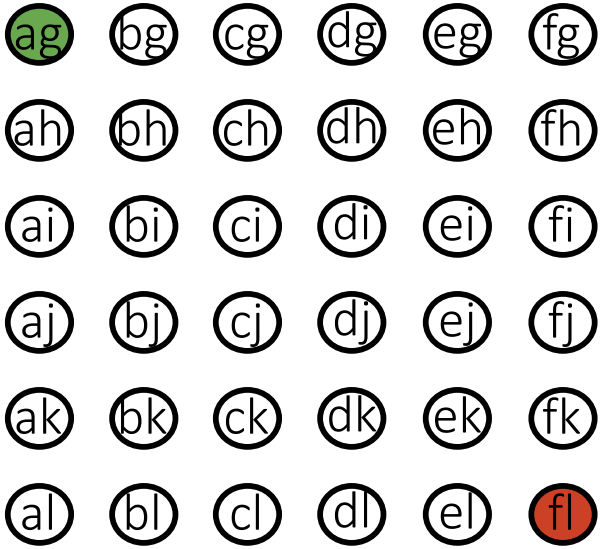


and operator (tensor product of graphs)

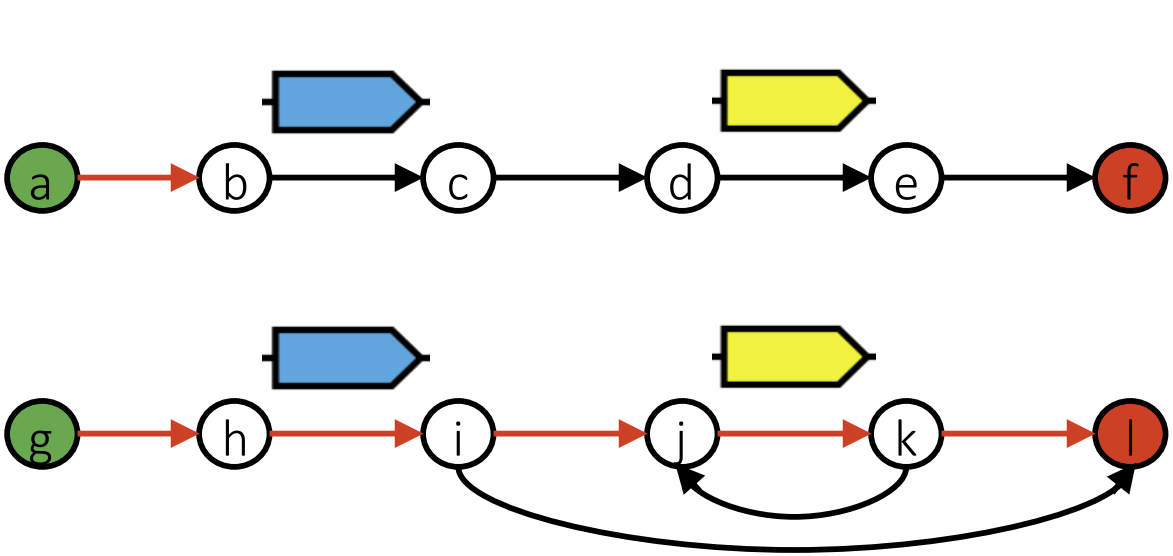


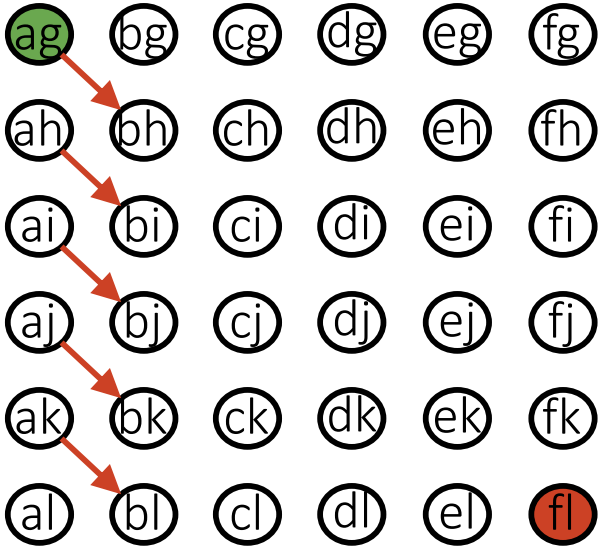
and operator (tensor product of graphs)



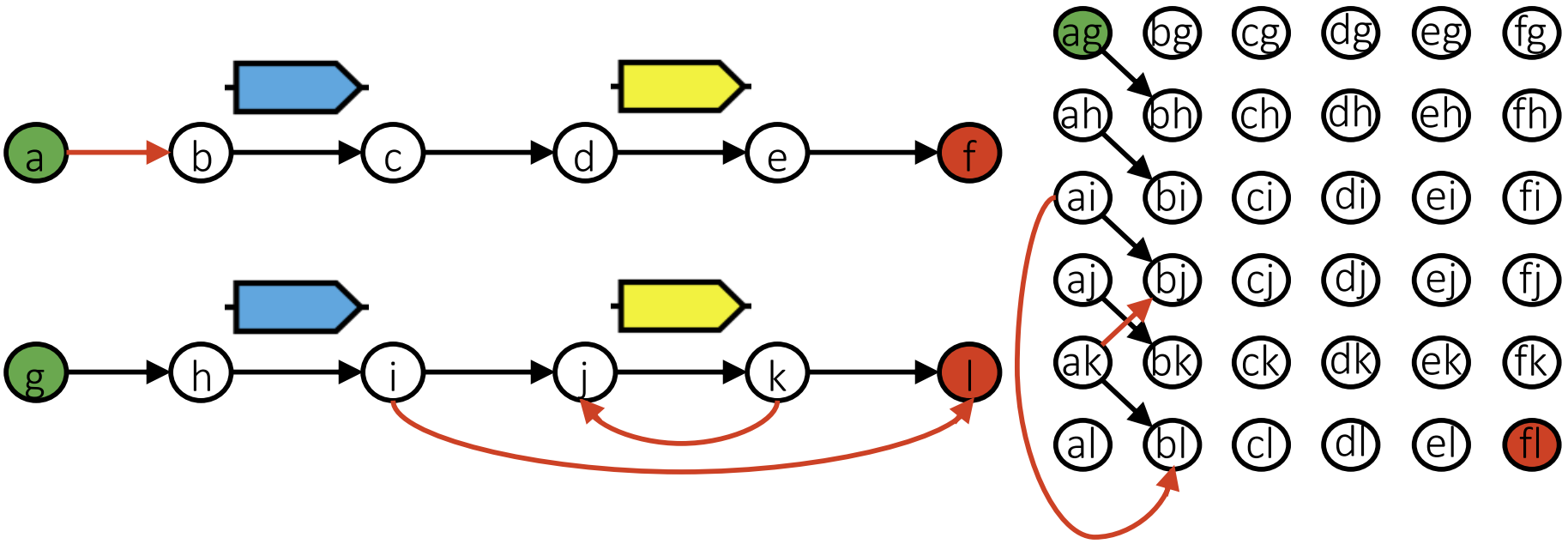


and operator (tensor product of graphs)

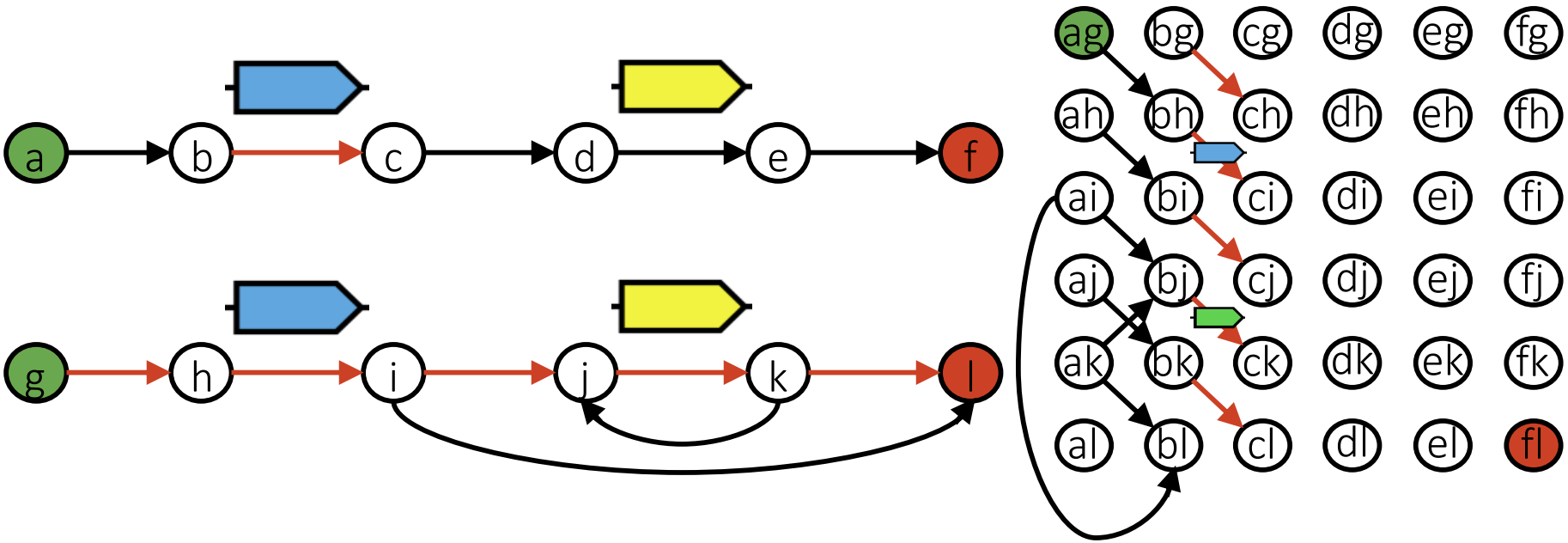




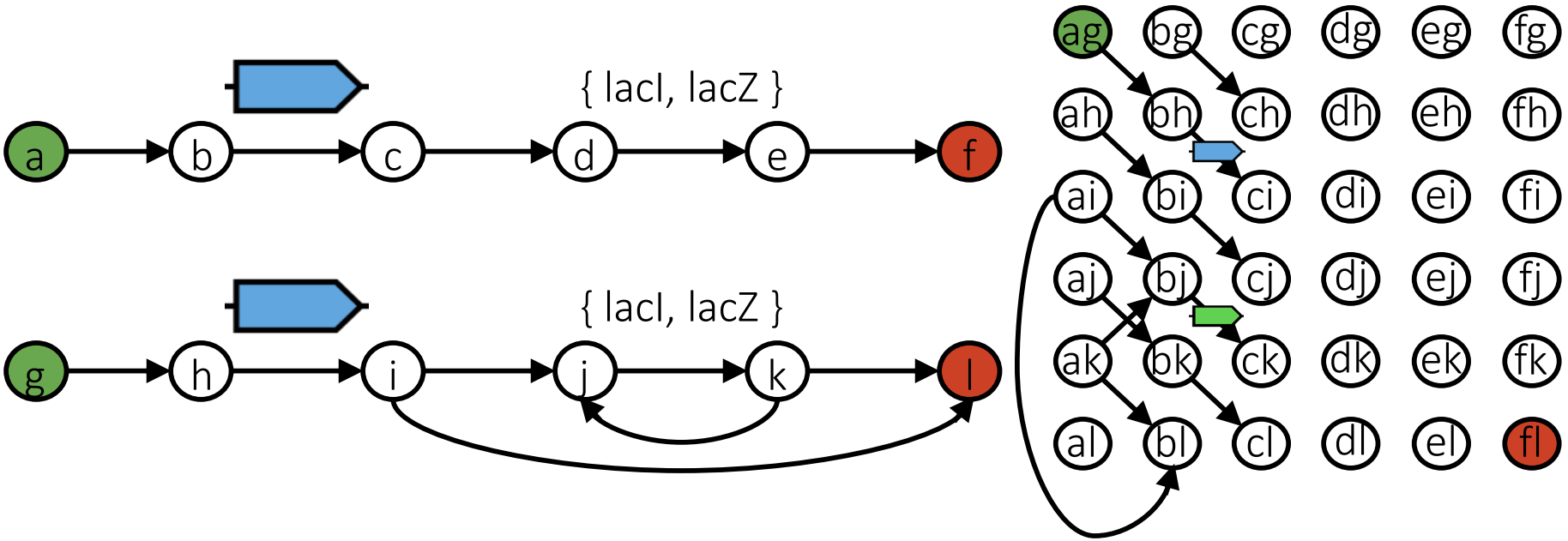
and operator (tensor product of graphs)



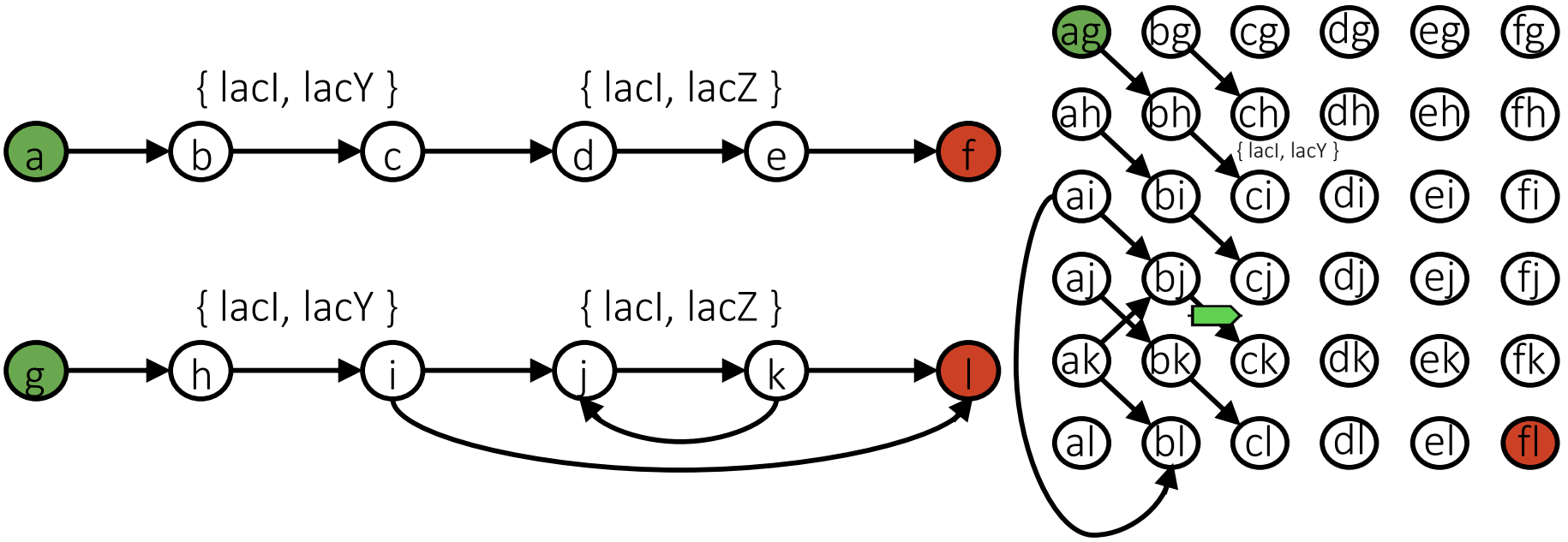
and operator (tensor product of graphs)



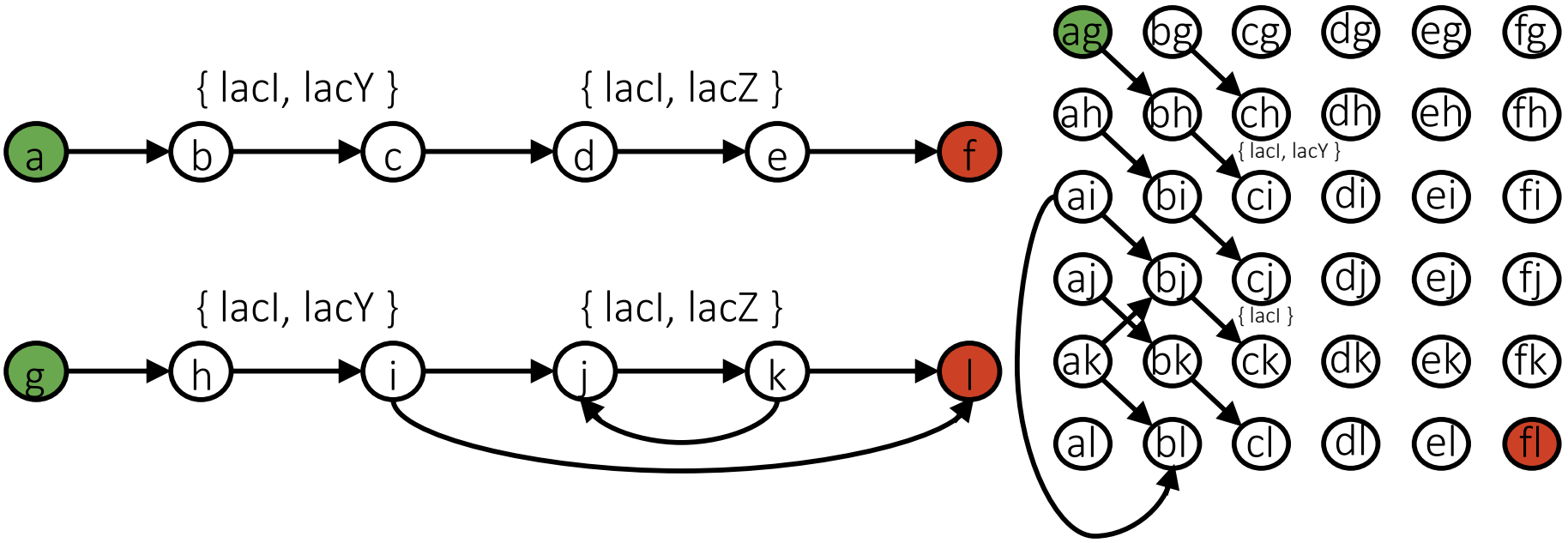
and operator (tensor product of graphs)



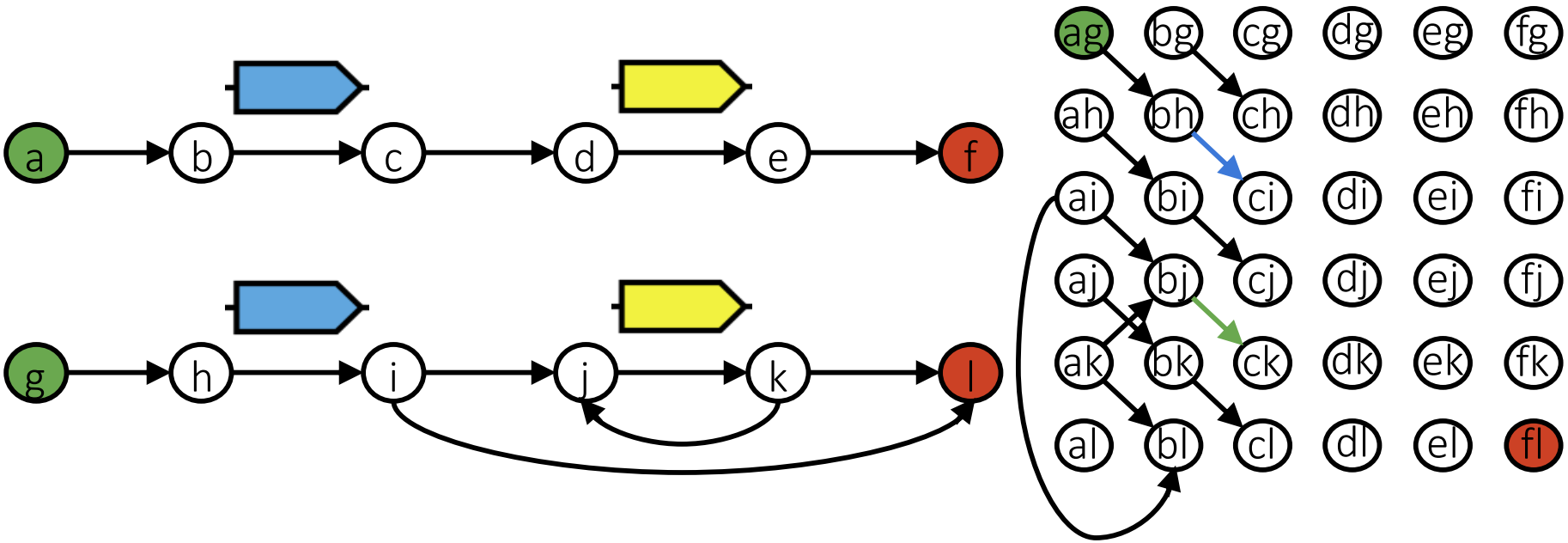
and operator (tensor product of graphs)



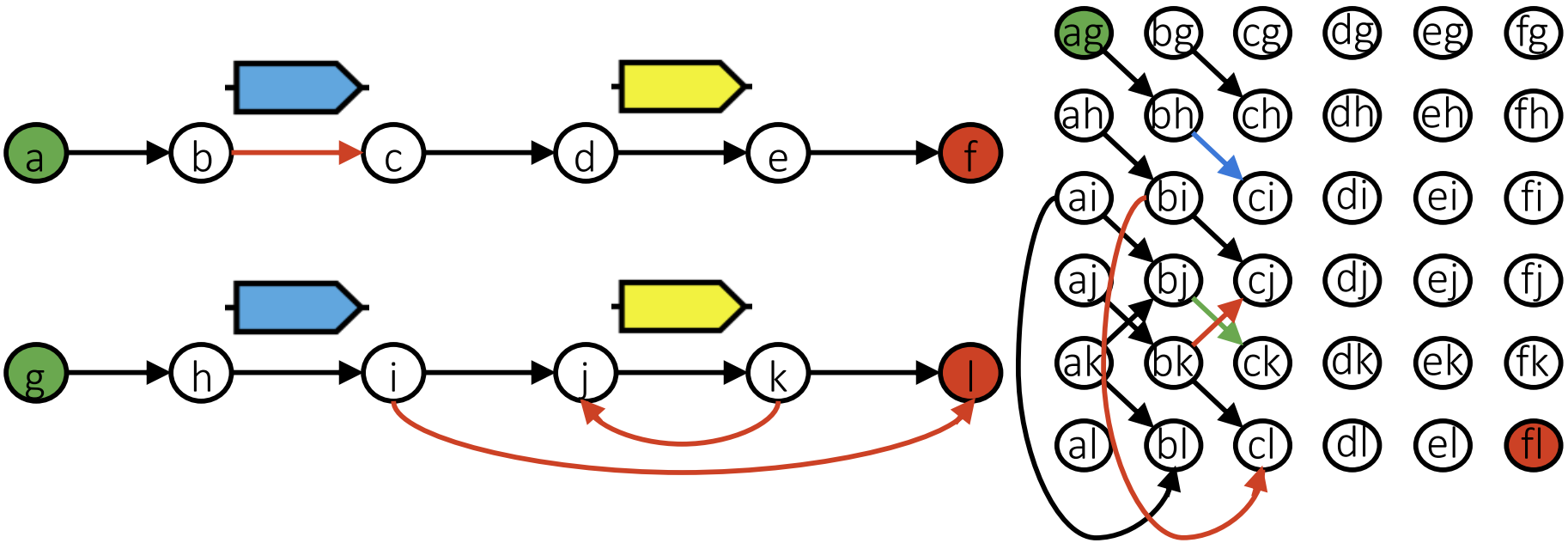
and operator (tensor product of graphs)



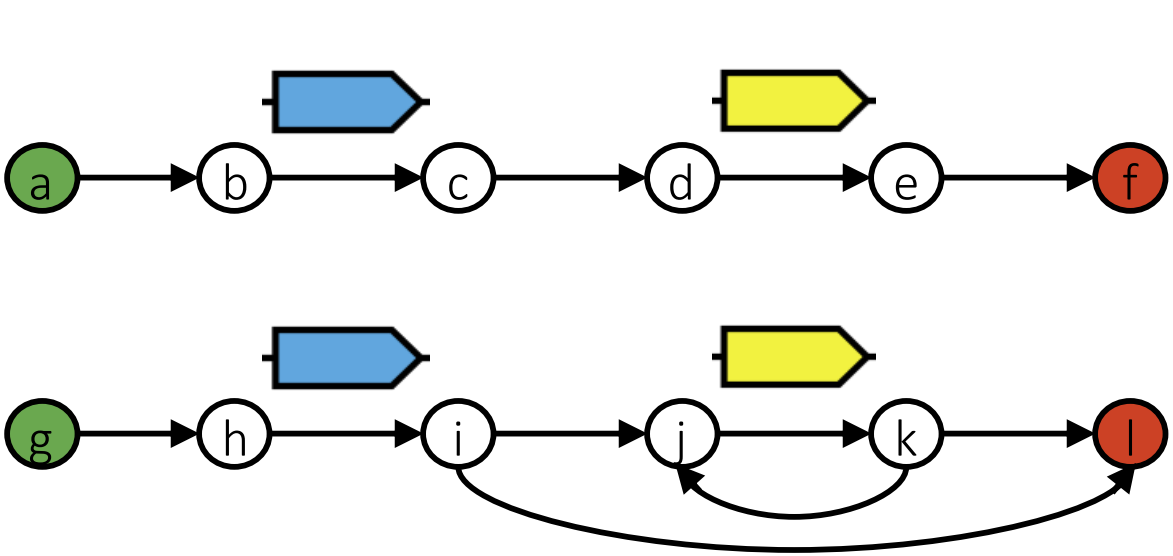
and operator (tensor product of graphs)

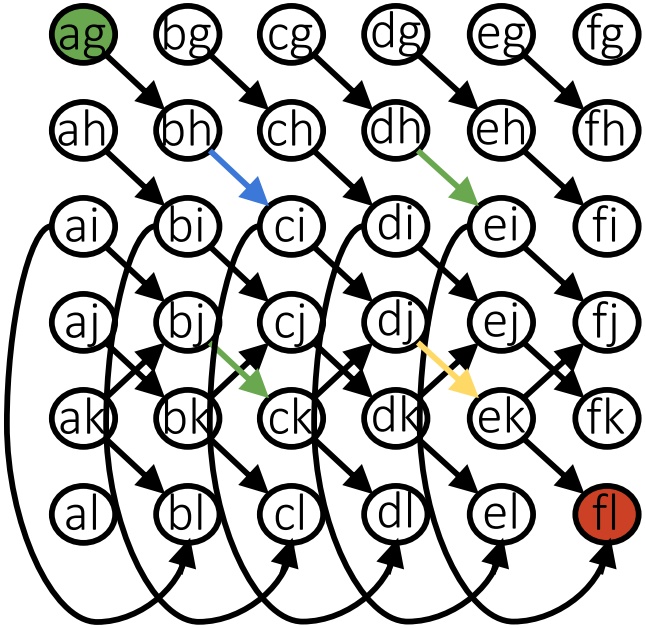


and operator (tensor product of graphs)

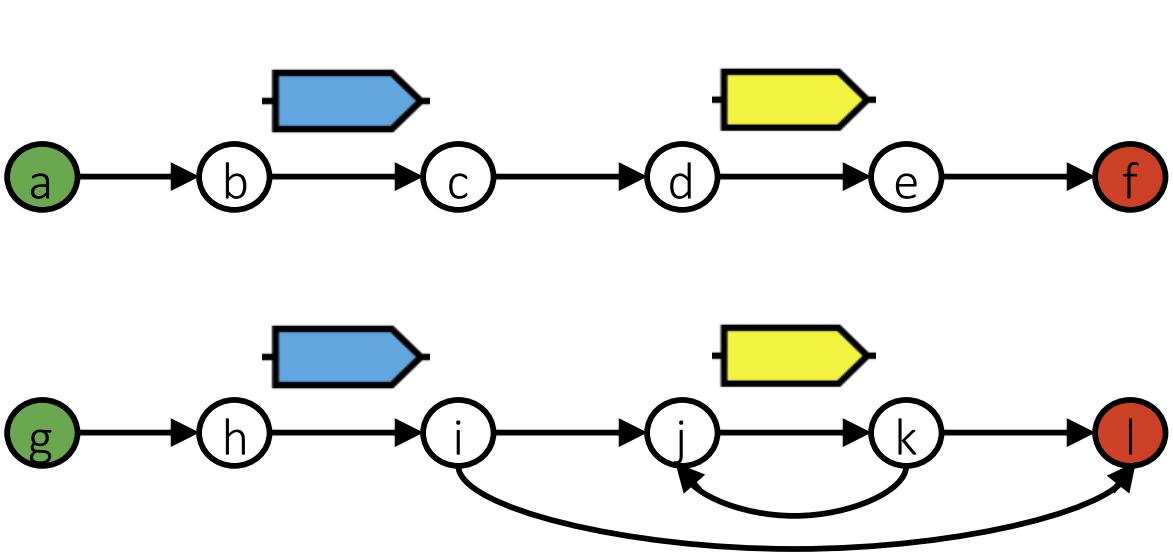


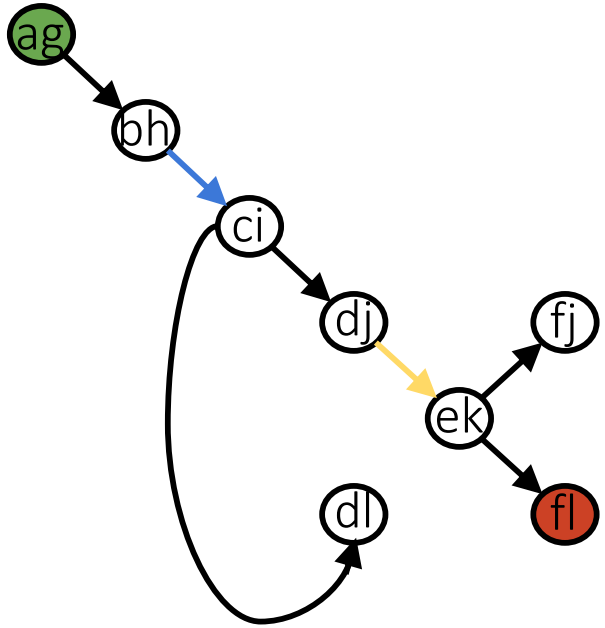
and operator (tensor product of graphs)



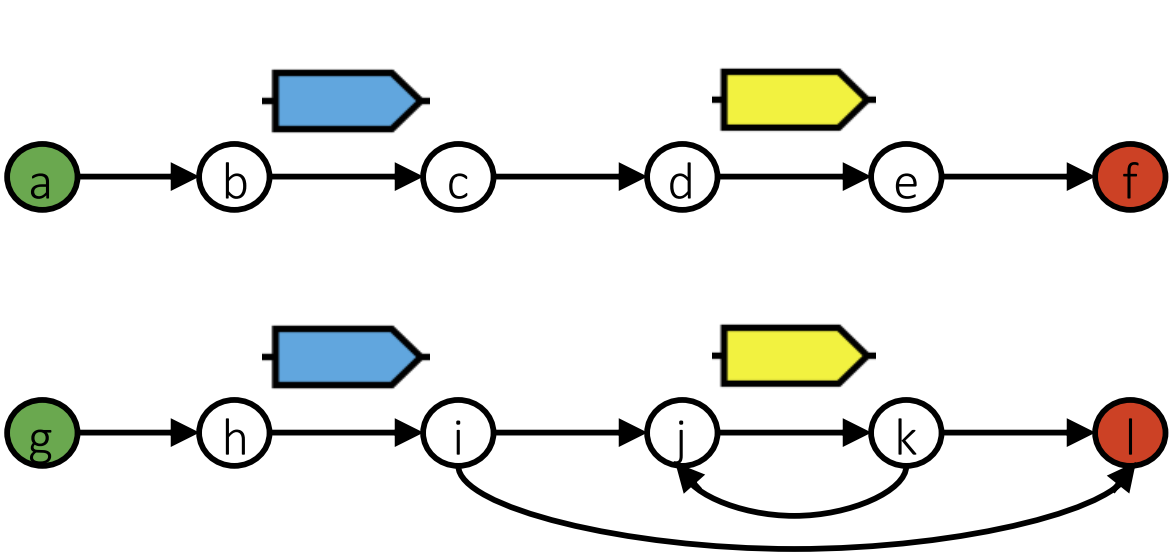


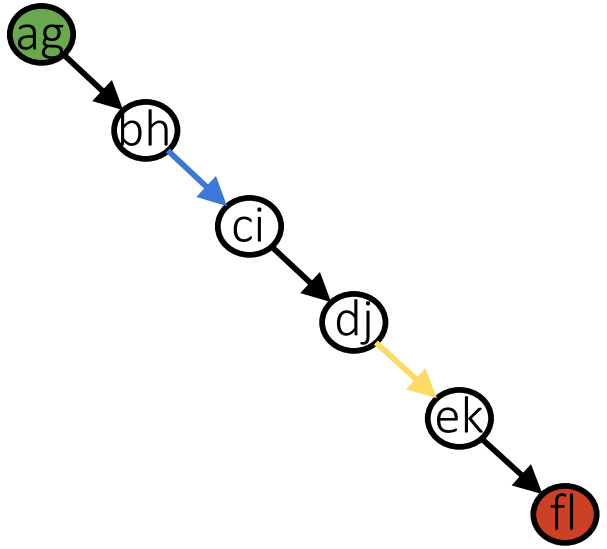
and operator (tensor product of graphs)





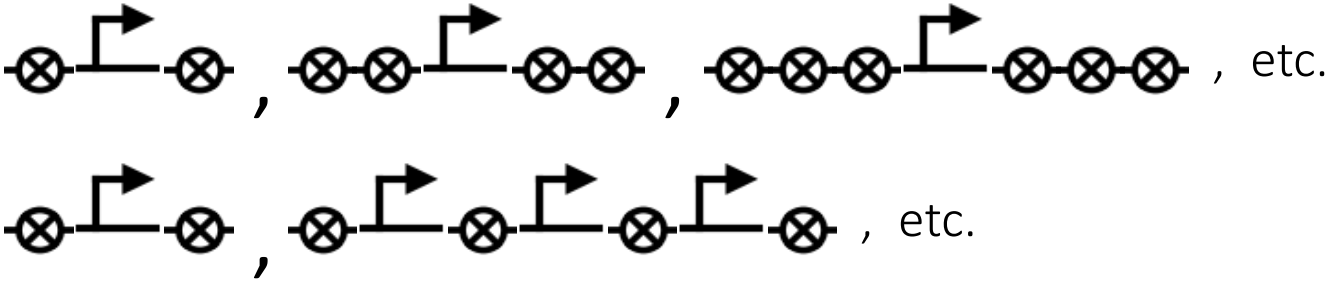
and operator (tensor product of graphs)



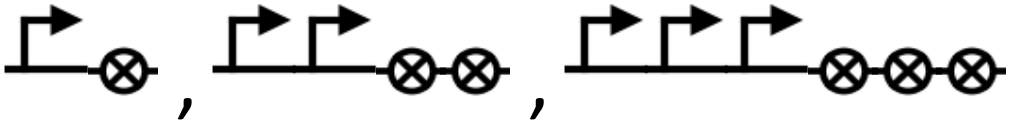


Unspecifiable Patterns

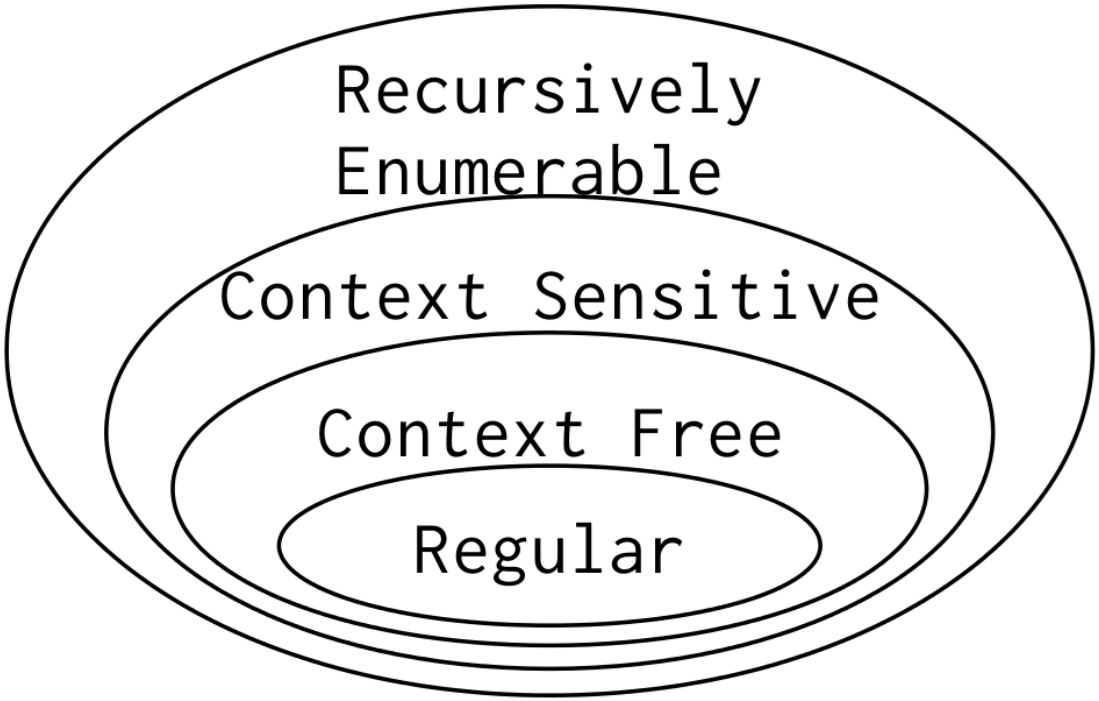
Variable Length Palindromes



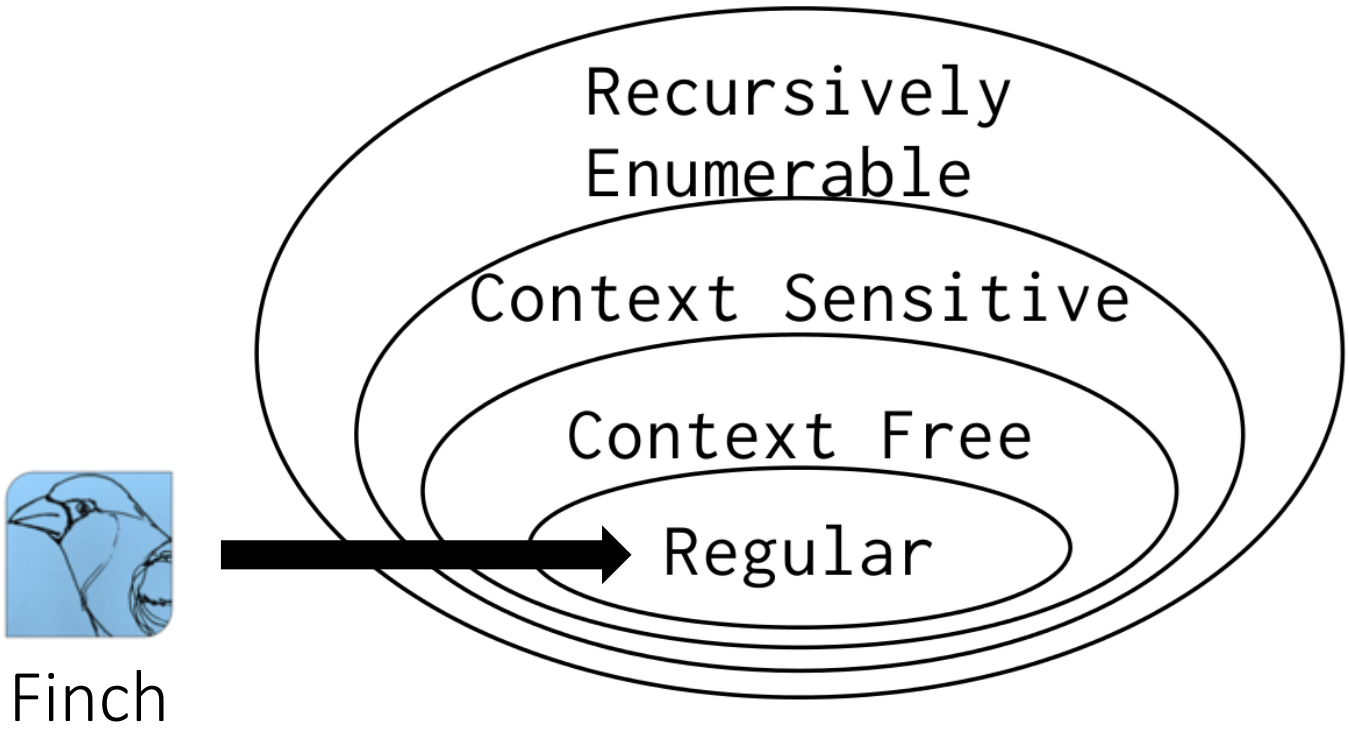
A^nB^n



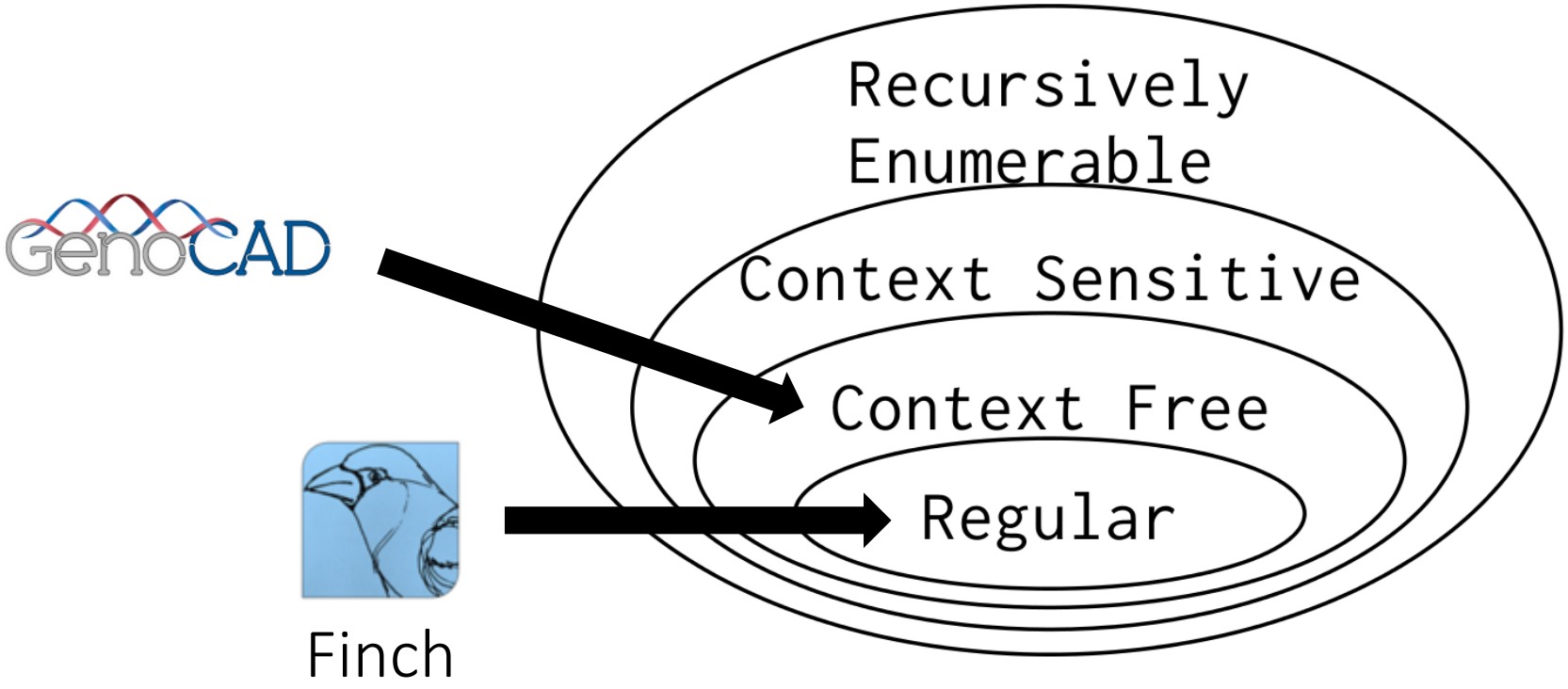
Expressive Power, Languages



Expressive Power, Languages

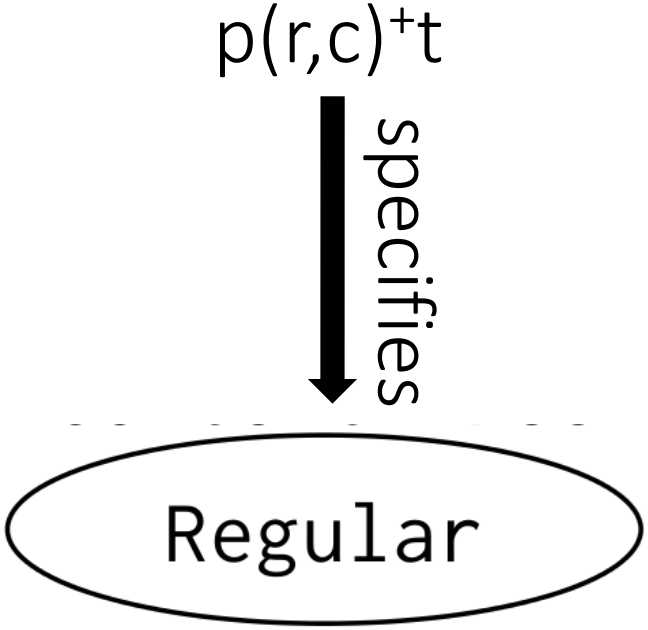


Expressive Power, Languages



Regular Languages

Regular Expression



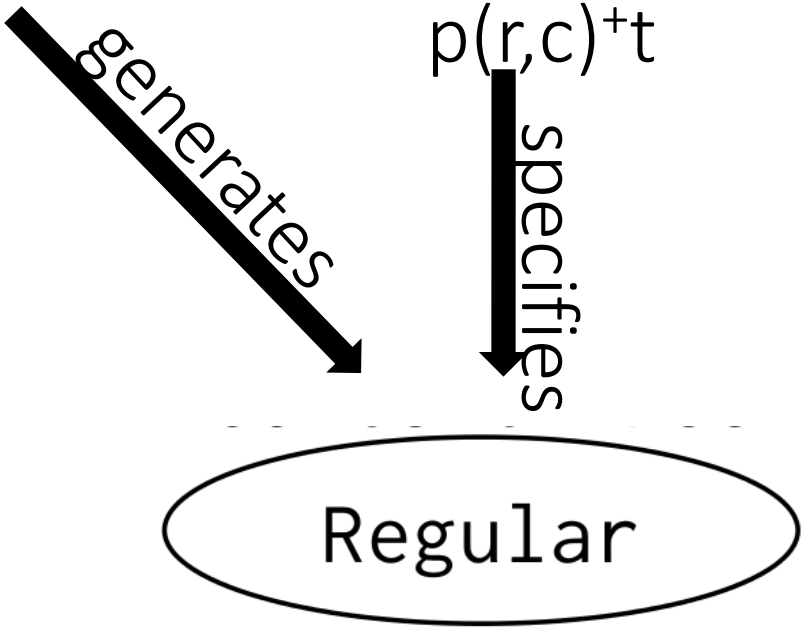
Regular Languages

Regular Grammar

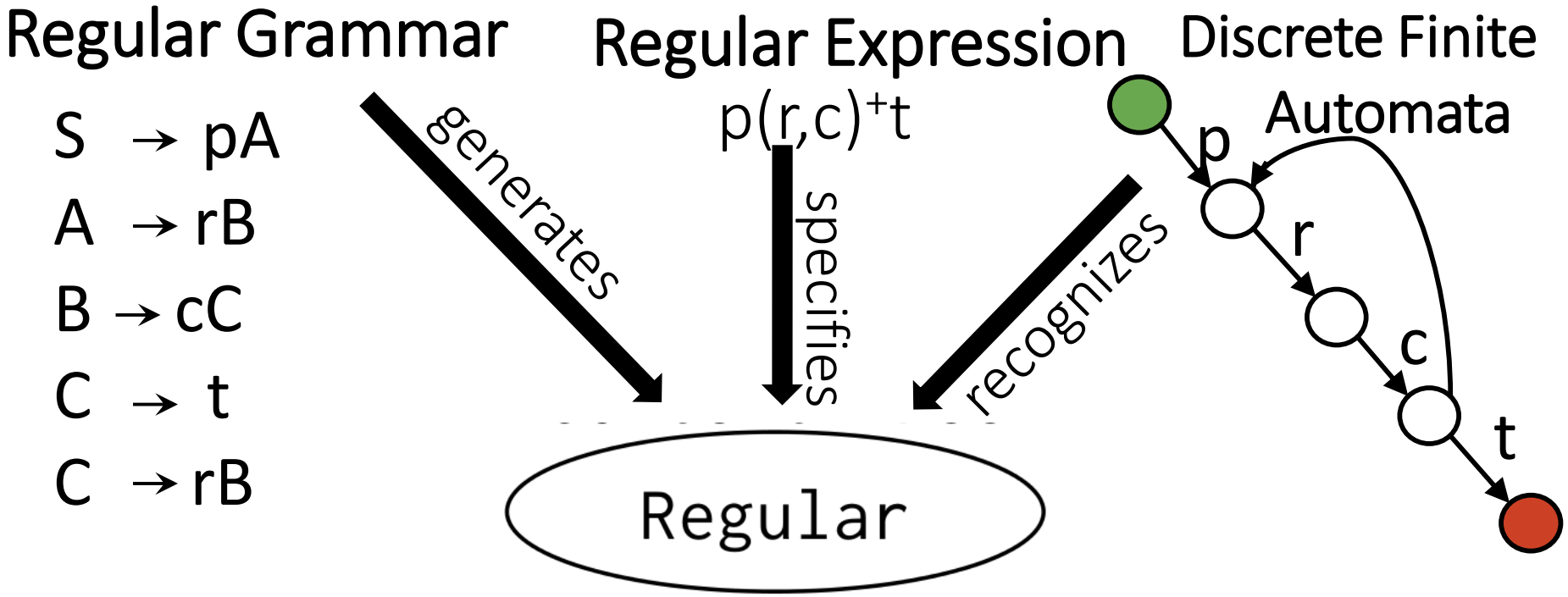
S → pA
A → rB
B → cC
C → t
C → rB

Regular Expression

$p(r,c)^{+}t$



Regular Languages



Lecture 7
Performance Specification– STL

Prof. Douglas Densmore

EC/BE552

Computational Synthetic Biology for Engineers

Material from Prashant Vaidyanathan, Curtis Madsen

Lecture Objectives

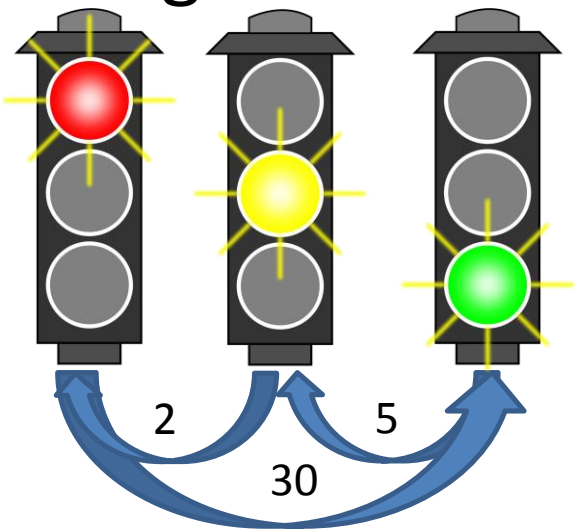
- What is Performance Specification
  - Examples
- Temporal Logic
  - Propositional Logic
  - Syntax Semantics
  - STL & Robustness
- STL in Biology
- Current Implementation

Performance Specification

- A specification that is primarily focused on the product of a process and what goods or services are required to produce it.
- These types of specifications, however, are not concerned with how the goods or services are used.
- For instance, a performance specification would be concerned with the amount of time required to complete a particular task but not how the time is spent completing it.

Performance Specification (Example)

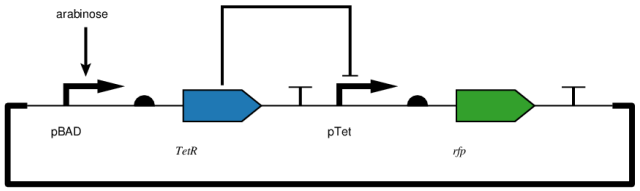
- A traffic light should remain green *until* a pedestrian requests a walk signal. *Within 5 seconds* of receiving the request, the traffic light should change to yellow for *2 seconds*, and then change to red for *30 seconds* before switching back to green.



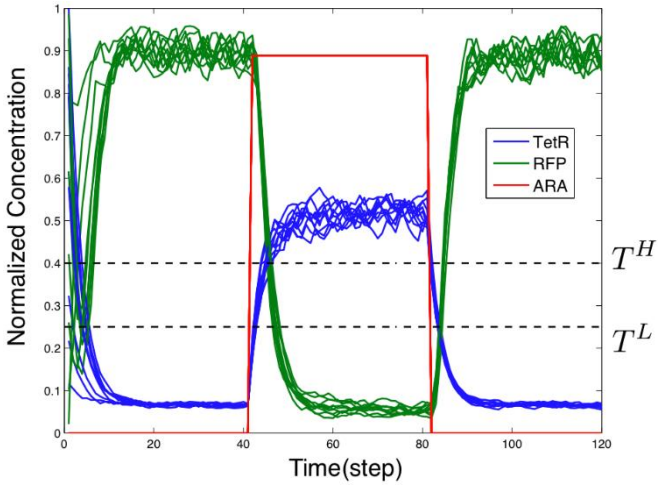
Performance Specification in Biology

- In synthetic biology, we are concerned with designing circuits that produce proteins for various tasks.
- A performance specification would be concerned with what proteins the circuit produces, how long it takes to produce the proteins, and what order the proteins are produced.

Gene Circuit Performance Specification



- *Function*: Inverter (NOT gate)
- *Specification*: If the *TetR* signal is higher than 0.4 (T_H), the *rfp* signal is expected to become lower than 0.25 (T_L) within a time period of 42 (k_1) and persist for a time period of 83 (k_2).



Propositional Logic

- Study of propositions (whether a statement is true or false)
  - Propositions and logical connectives
  - The value of propositions depends on the truth value of the respective components.

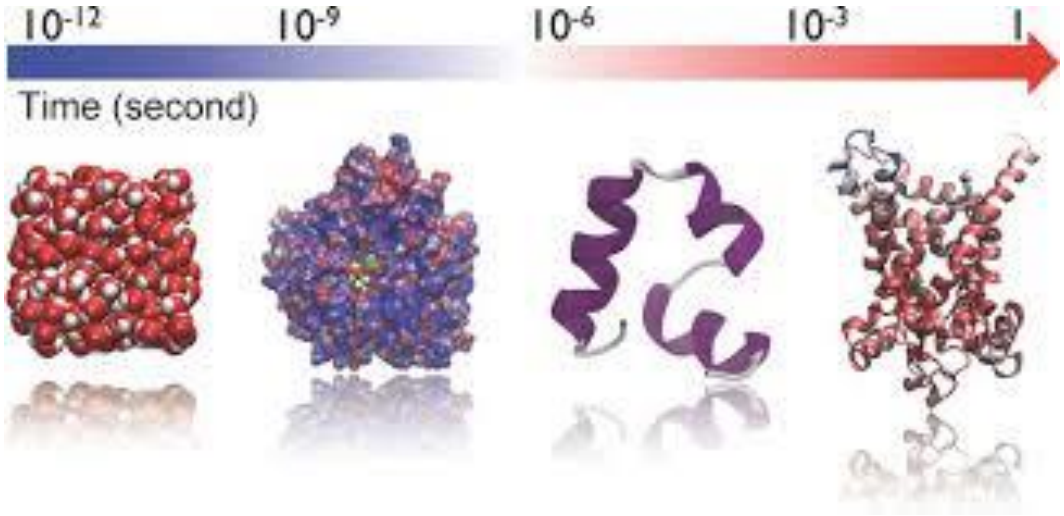
- Premise 1: If it snows, it will be cold
- Premise 2: It is snowing
- Conclusion: It is cold.

Temporal Logic

- A system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time.
- We can talk about how a statement is always true or how an event will eventually occur.
- For example, “The light is always on.” or “This class will eventually be over.”

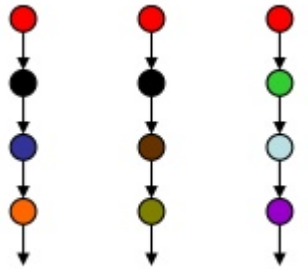
Why Temporal Logic?

- Time is essential to biological systems.
- Any circuit designed to operate in a cell will need to take into account the different time scales of biological processes.

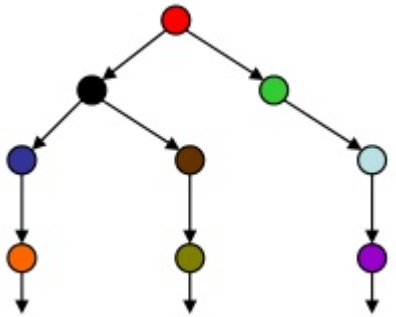


Types of Temporal Logic

- Linear Time:
  - Every moment has a unique successor.
  - Infinite sequences.
  - Linear Time Temporal Logic (LTL).



- Branching Time:
  - Every moment has several successors.
  - Infinite tree.
  - Computation Tree Logic (CTL).



LTL vs CTL

- Due to its branching structure, computations utilizing CTL are more efficient than those using LTL.
- LTL is, however, better for expressing properties dealing with fairness and stability.
- Which logic to use really depends on the application.

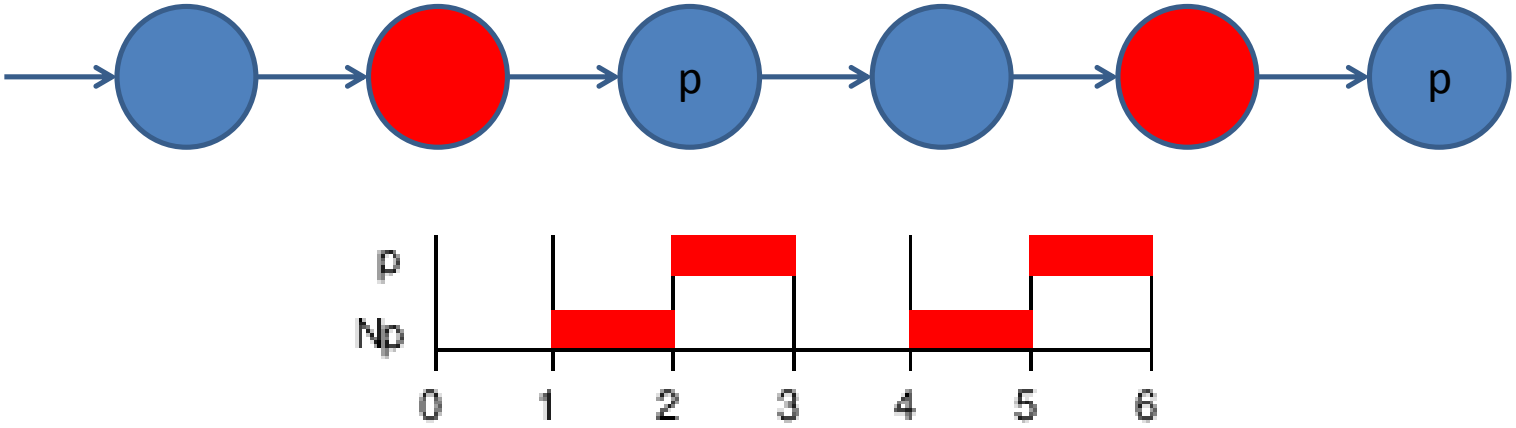
Temporal Logic Syntax

- Logical Operators:
  - Conjunction: $\phi \wedge \psi$
  - Disjunction: $\phi \vee \psi$
  - Implication: $\phi \rightarrow \psi$
  - Negation: $\neg\phi$
- Temporal Operators:
  - Next: N ϕ (or $\bigcirc\phi$)
  - Until: ϕ U ψ
  - Future (Eventually): F ϕ (or $\Diamond\phi$)
  - Globally: G ϕ (or $\Box\phi$)

Next Operator

N p

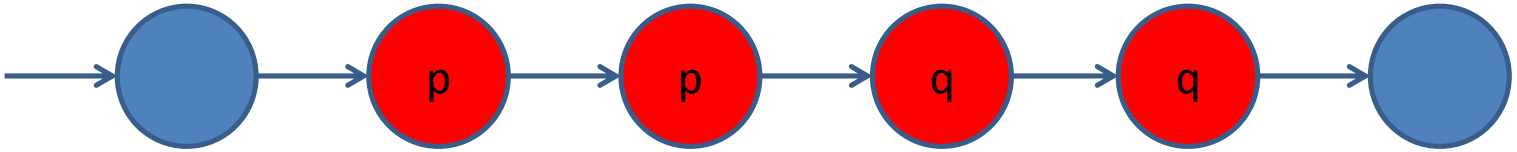
- **N**ext: p has to hold at the next state.

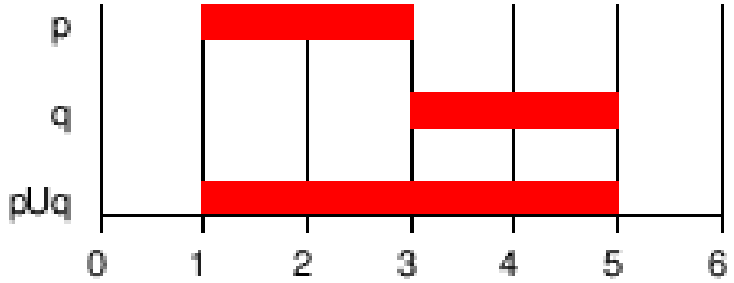


Until Operator

p U q

- q holds at the current or a future position, and p has to hold until that position. At that position p does not have to hold any more.

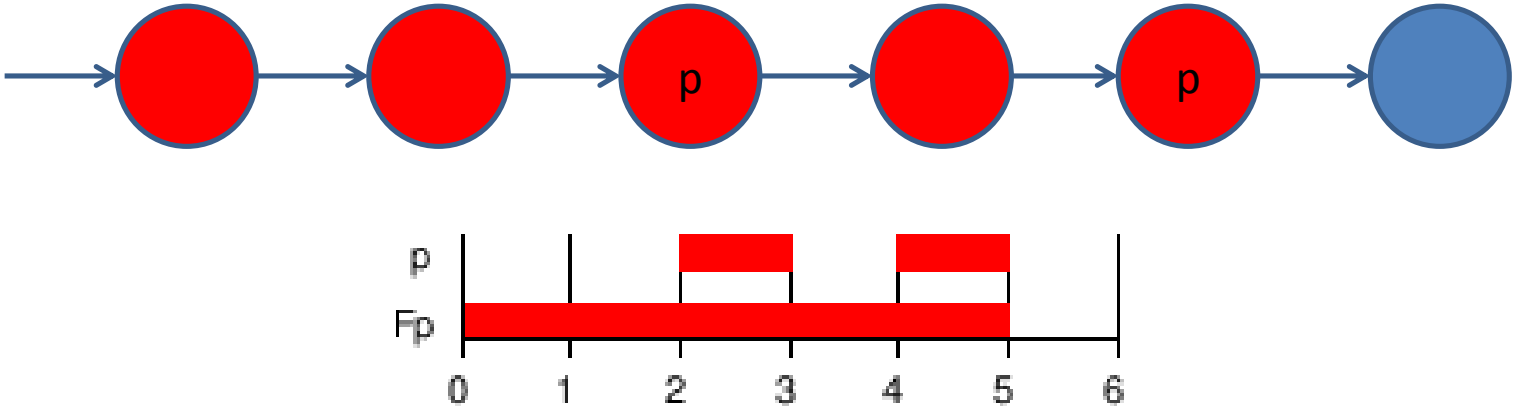




Future (Eventually) Operator

F p

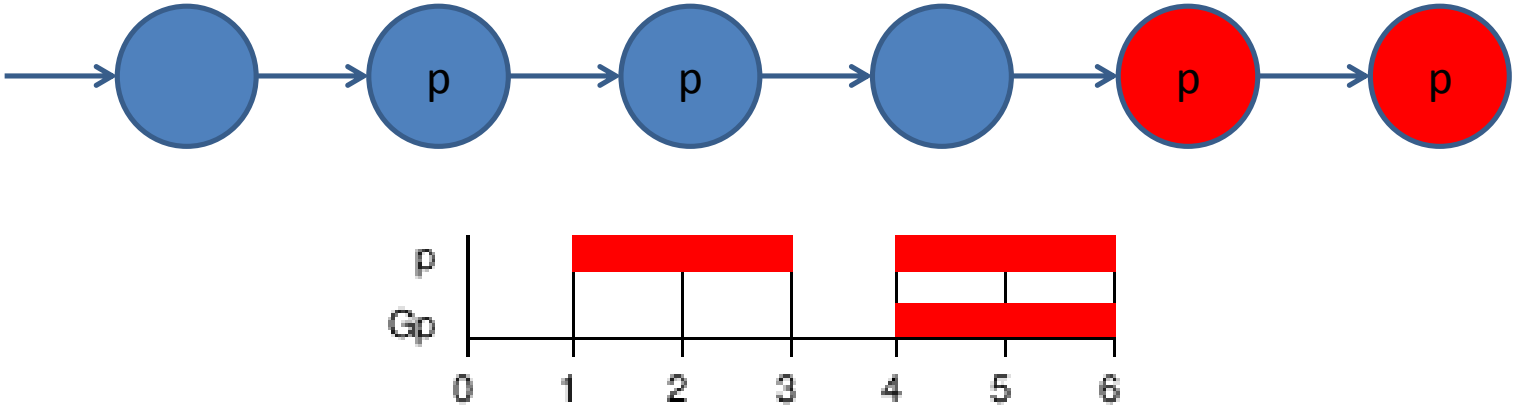
- **F**uture: p eventually has to hold (somewhere on the subsequent path).



Globally Operator

G p

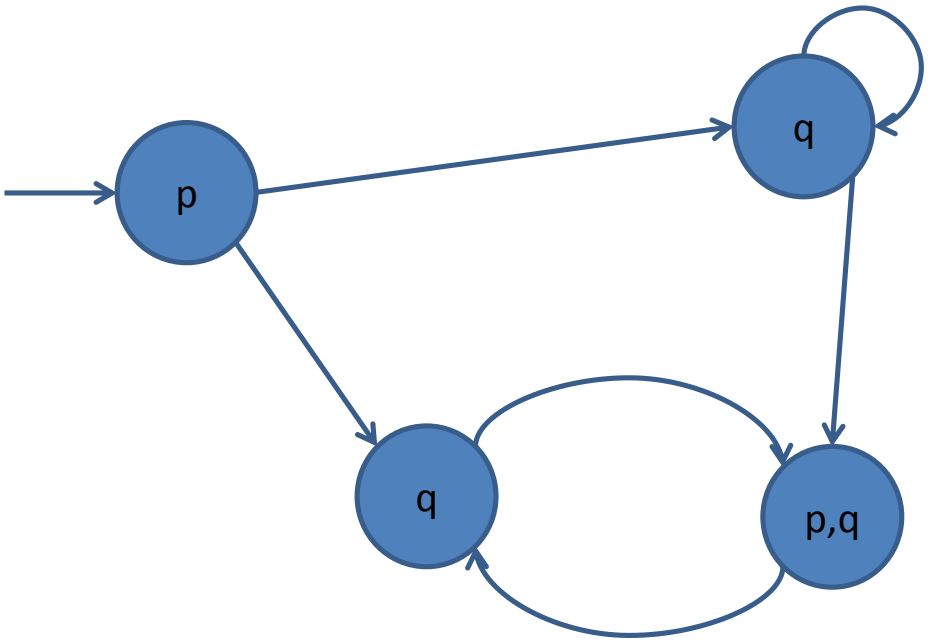
- **G**lobally: p has to hold on the entire subsequent path.



Temporal Logic Semantics

- For a state in a system, a formula ϕ is satisfied if all paths exiting that state satisfy the formula.
- If the initial states of a system satisfy a formula, then the system as a whole satisfies the system.

Example



- F q
  - True
- p U q
  - True
- FG q ∧ ¬p
  - False
- N p
  - False

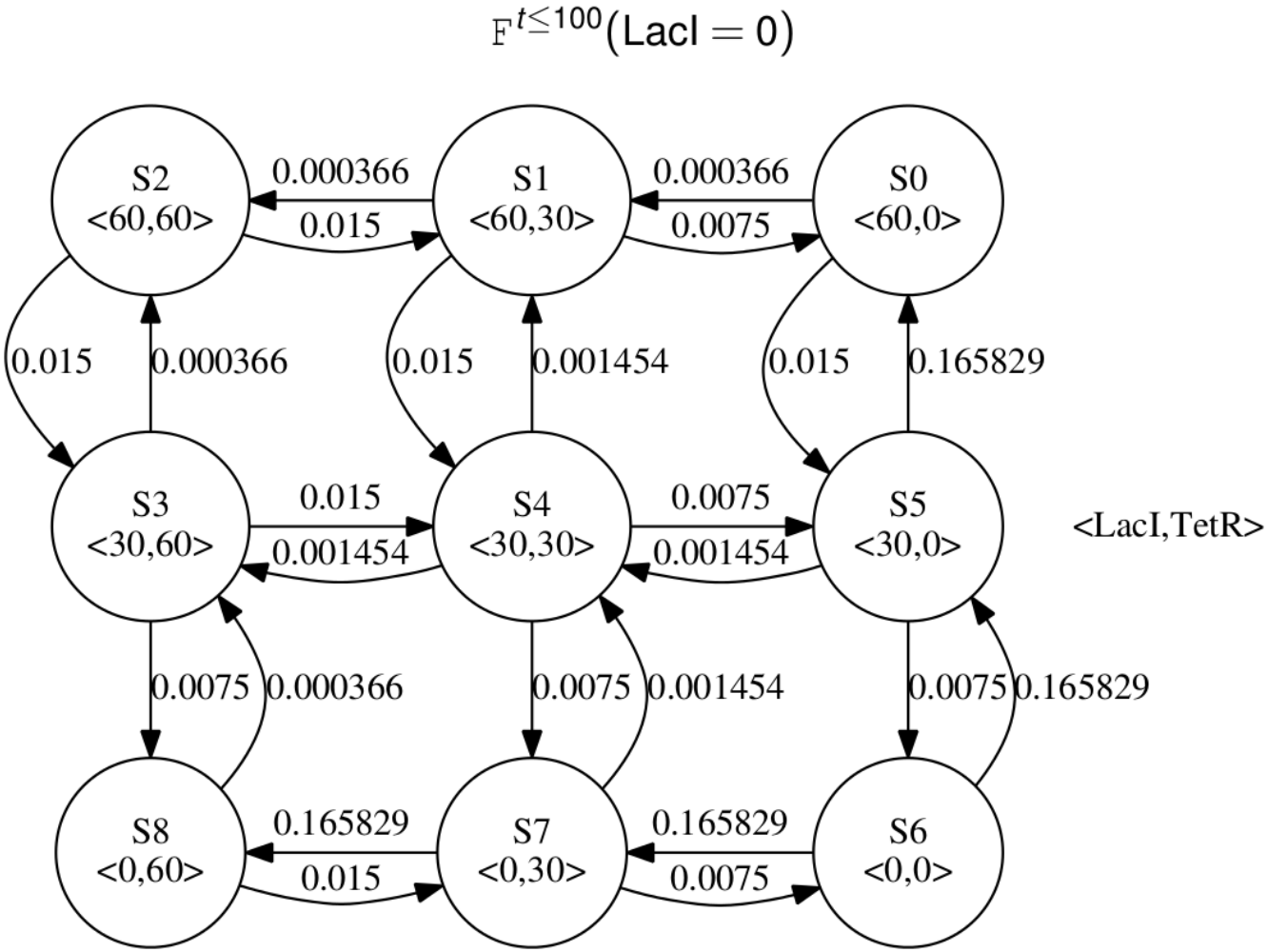
Probabilistic Temporal Logics

- In order to ask questions about the likelihood of certain events, we need ways of expressing probability in temporal logics.
- Can introduce the probabilistic operator P.
- For example, $P \geq 0.95\ (\phi)$ asks, “Is the likelihood that ϕ is true greater than 95%?”

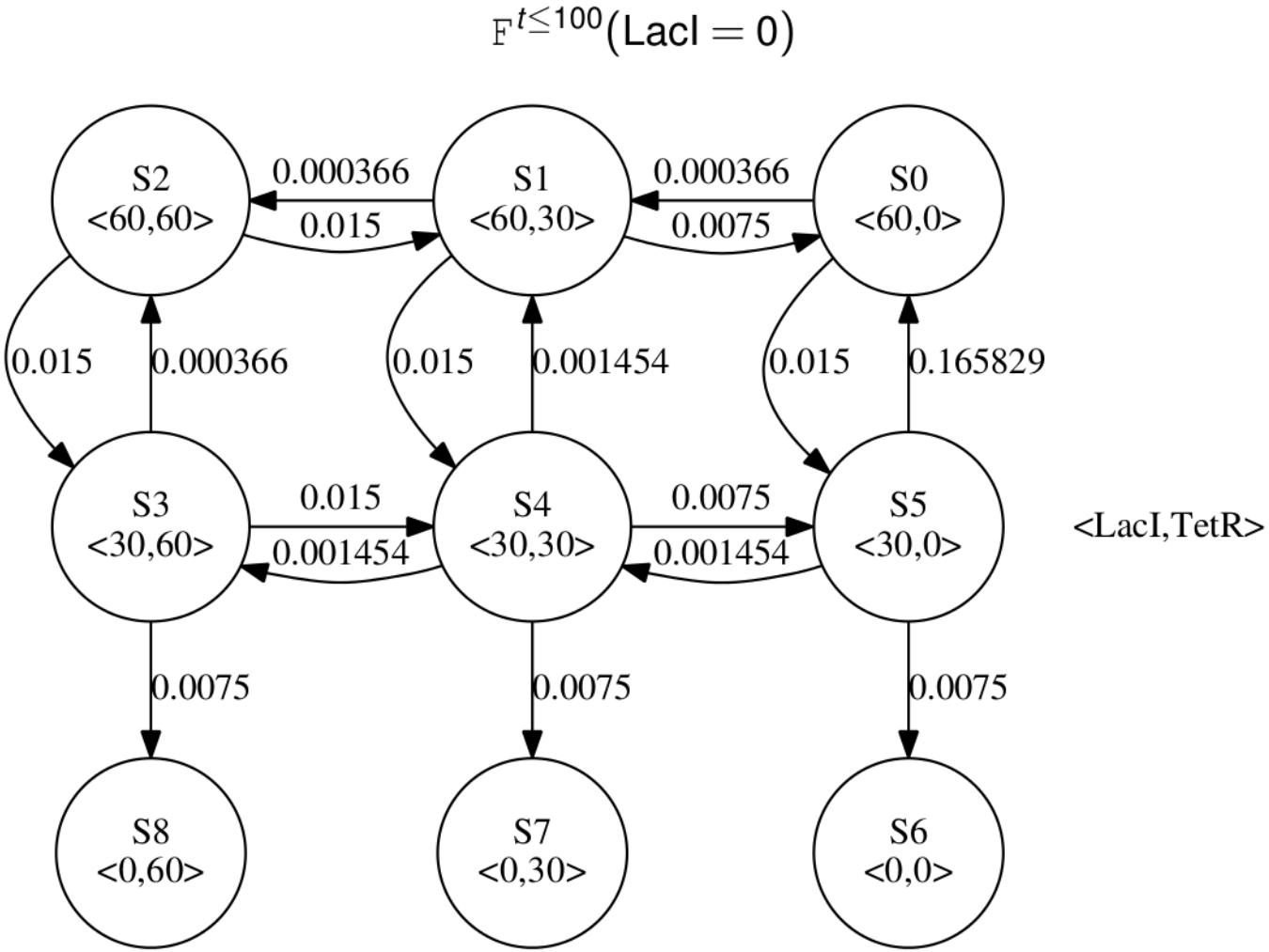
Computations with Probability

- There are two types of approaches used to compute the likelihood that a property is true: statistical and numerical based techniques.
- Statistical techniques involve simulating a system a large number of times and terminating whenever a property is shown to be true or false.
- Numerical methods attempt to determine the likelihood of a property in a more direct method utilizes techniques such as Markov chain analysis.

Probabilistic Example

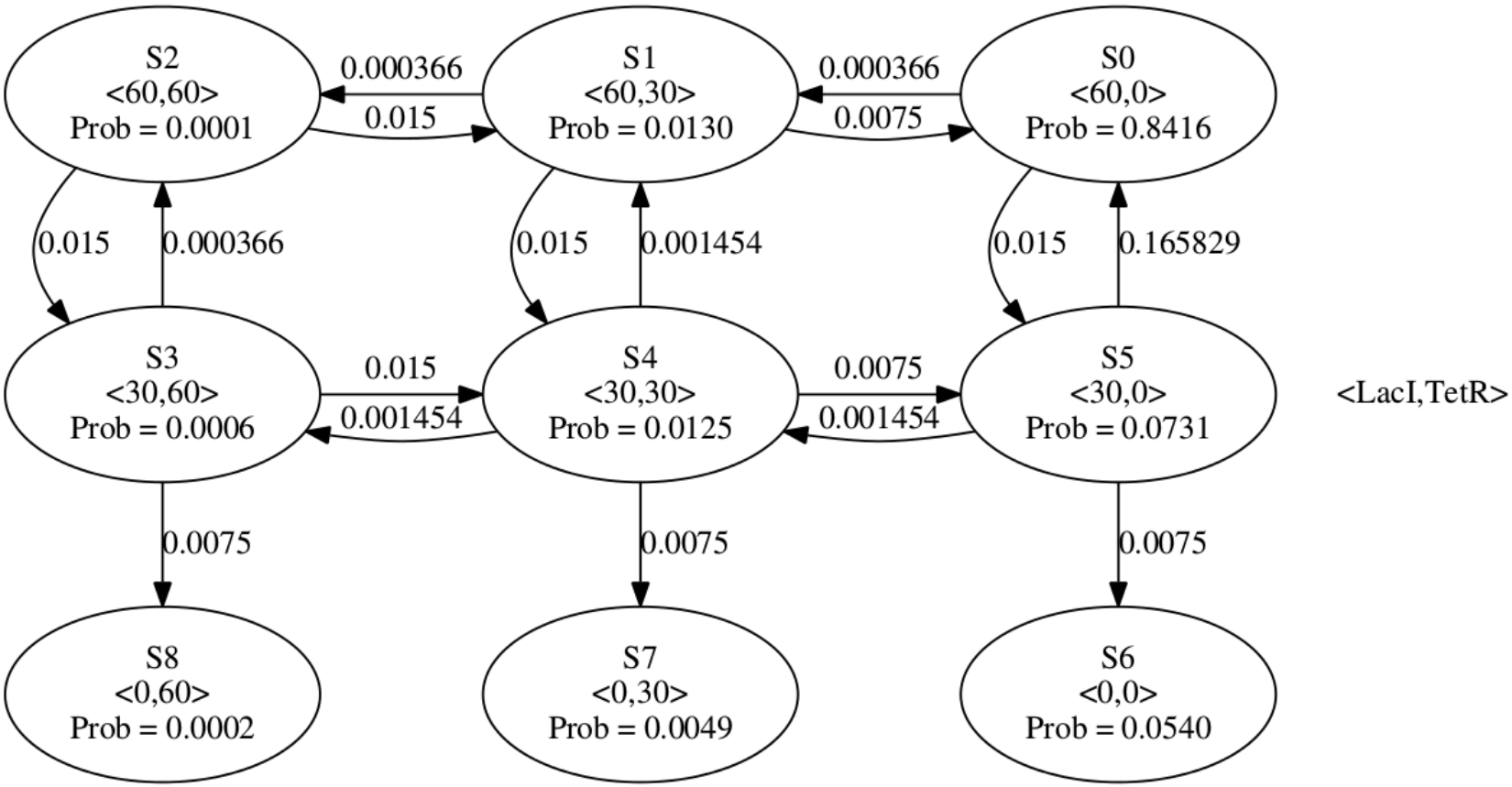


Probabilistic Example



Probabilistic Example

$$\mathbb{F}^{t \leq 100}(\mathsf{LacI} = 0) \approx 0.0591$$



Signal Temporal Logic (STL)

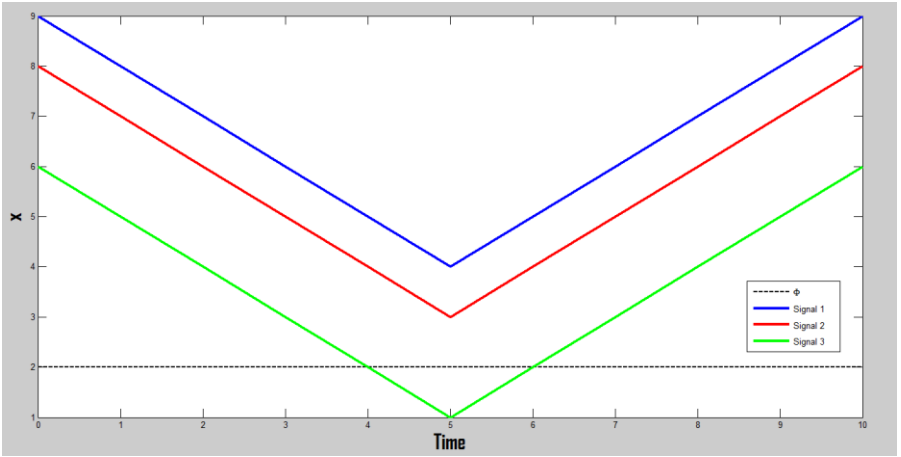
- Extension of LTL with real-time and real-valued constraints.
- Can express properties of “signals” in the system over time.
- For instance, $G_{[0,10)}(x > 2)$ represents that between 0s and 10s the signal, x, is greater than 2.

Robustness

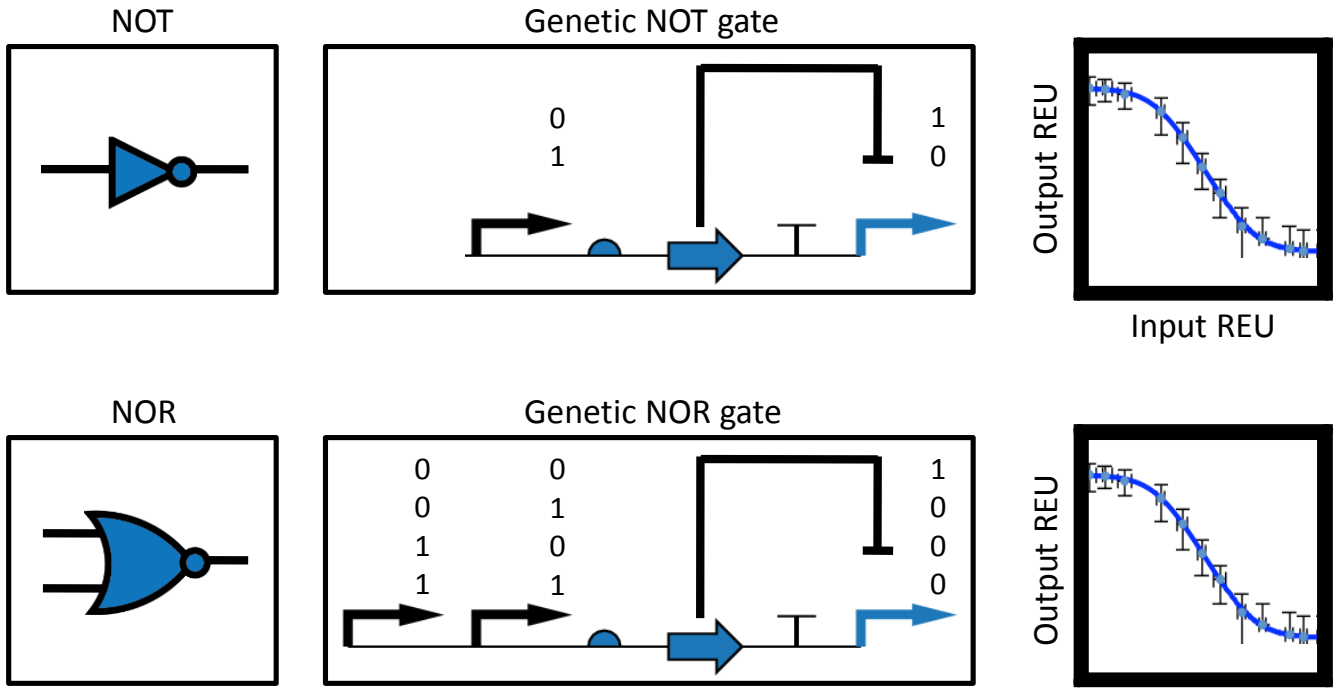
- The definition of STL semantics includes *quantitative semantics* that can recursively compute the *robustness* or *degree of satisfaction*
- *Robustness* is used as a measure of how well a given trace satisfies or violates a specification

EXAMPLE

- *STL Specification*: $\mathsf{F} = G_{[0,10)}(x > 2)$
- Signal$_1$ is satisfying with: $r(s_1, \mathsf{F}) = 2$
- Signal$_2$ is satisfying with: $r(s_2, \mathsf{F}) = 1$
- Signal$_3$ is violating with: $r(s_3, \mathsf{F}) = -1$

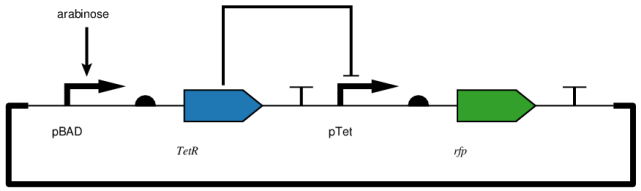


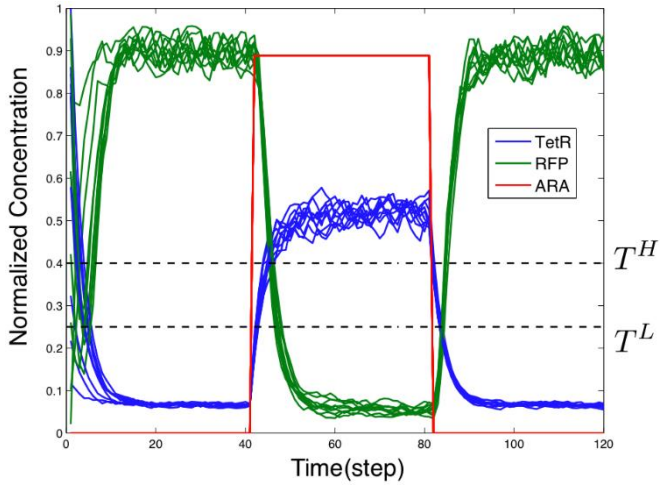
Functional Specification – Boolean Algebra



STL – Functional Richness

- *Function*: Inverter (NOT gate)
- *English Specification*: If the *TetR* signal is higher than 0.4 (T_H), the *rfp* signal is expected to become lower than 0.25 (T_L) within a time period of 42 (k_1) and persist for a time period of 83 (k_2).
- *STL Specification*: $(x_{TetR} > 0.4) \Rightarrow (F_{[0,42\ )} G_{[0,83\ )} x_{rfp} < 0.25\ )$

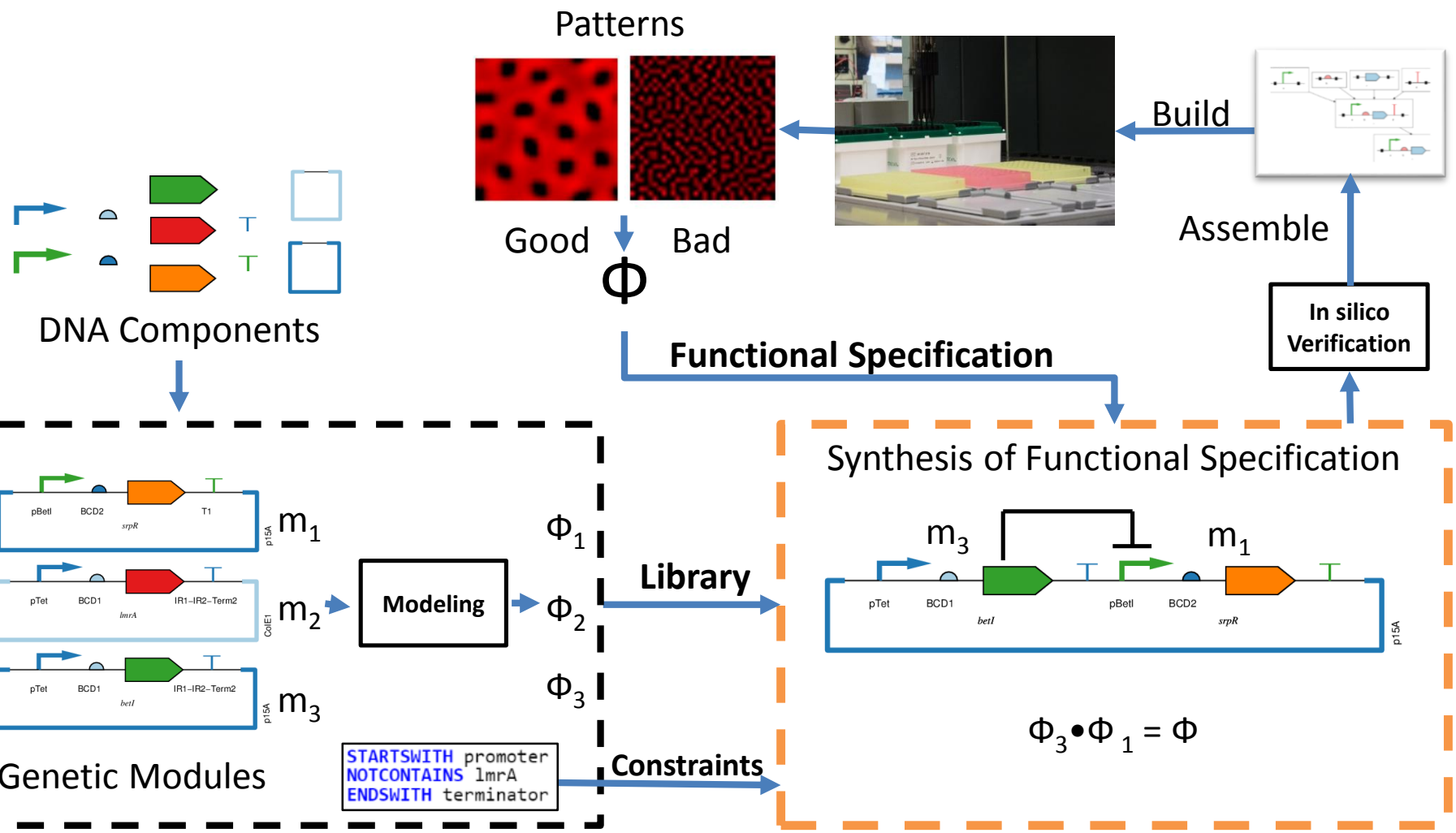




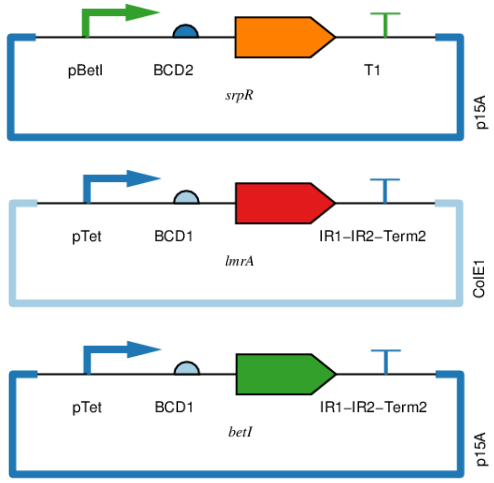
Challenges

- Biological systems often behave stochastically instead of deterministically which may require the use of probabilistic logics.
- In synthetic biology, we are often interested in composing modules together and this is not readily available in temporal logics like STL.

Implementation

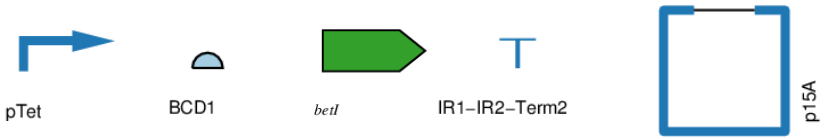


Library

Genetic Modules	Name	STL Formula
	m_1	Φ_1
	m_2	Φ_2
	m_3	Φ_3
	⋮	

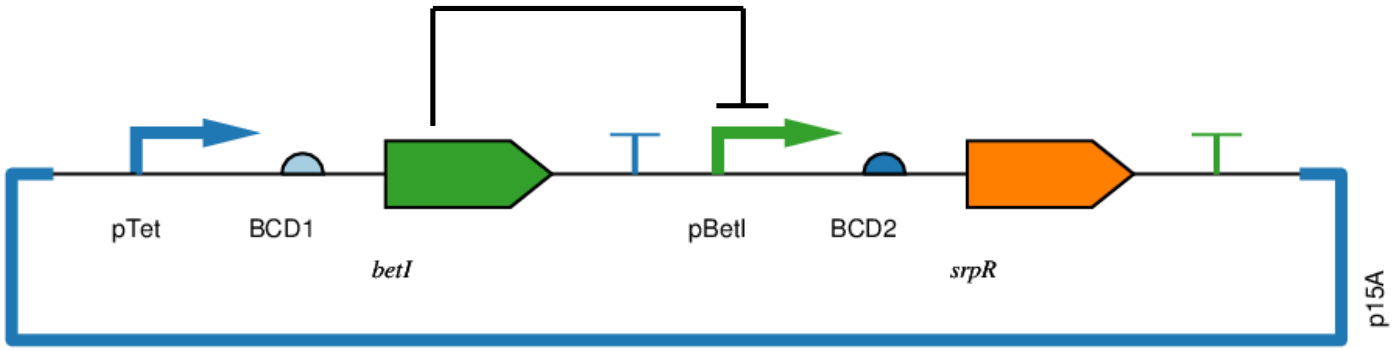
Composability

DNA Components

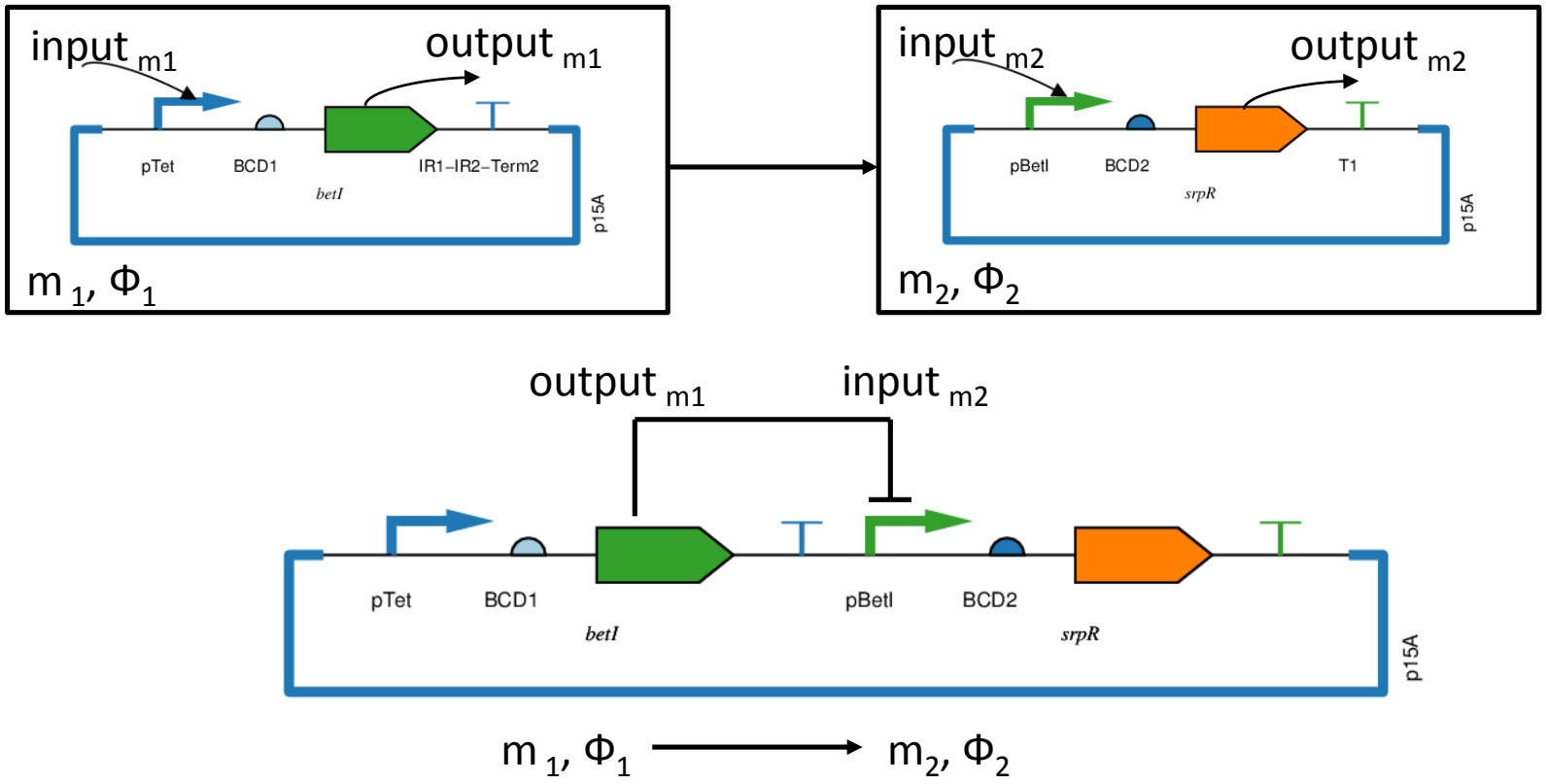


Genetic Modules

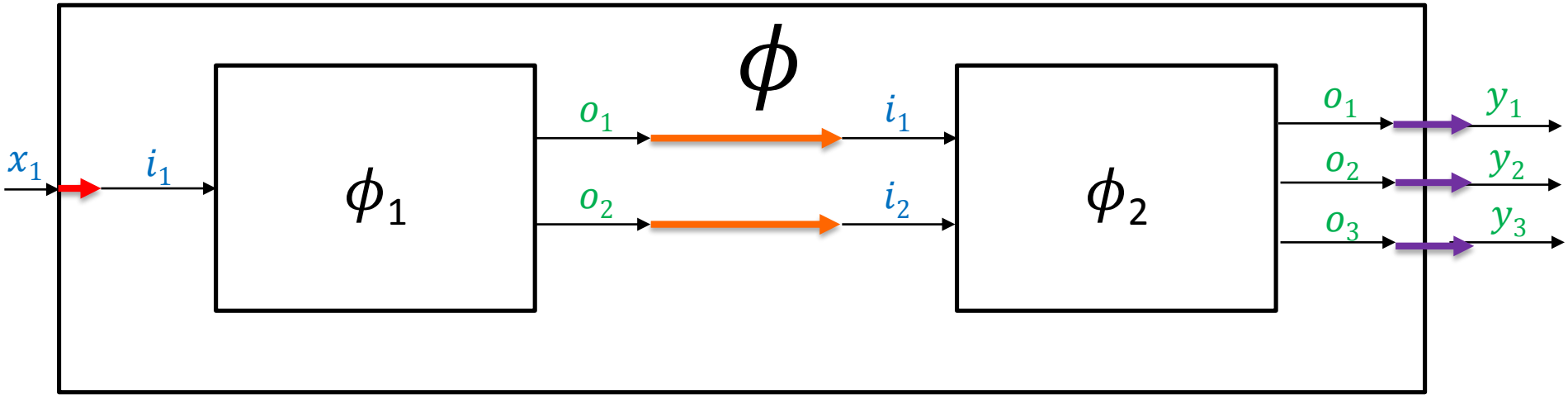
Genetic Circuit



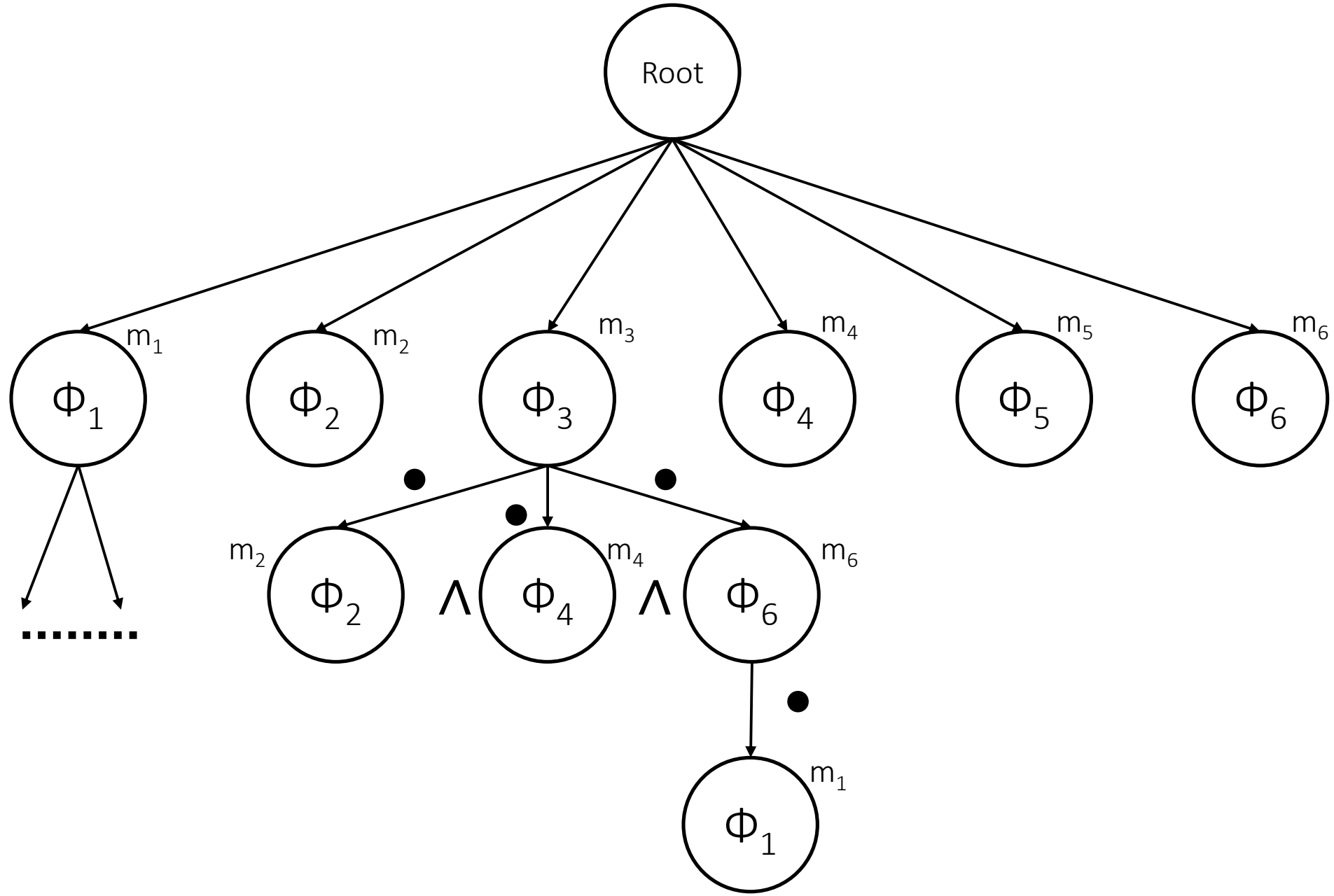
Composability

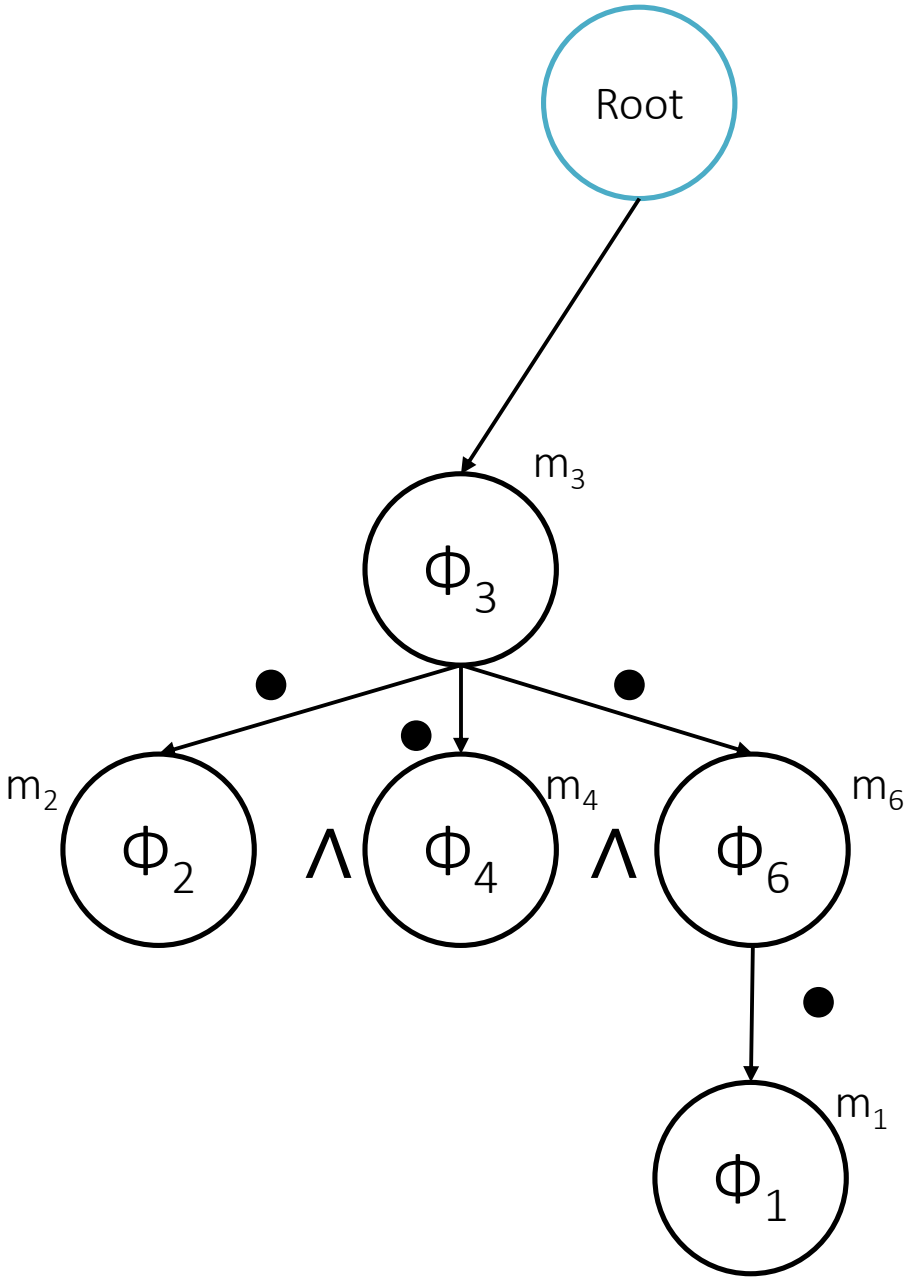


- For instance, ϕ is composed of ϕ_1 and ϕ_2:
  - Concatenation – $\phi(x_1, y_1, y_2, y_3) = \phi_1(i_1, o_1, o_2) \bullet \phi_2(i_1, i_2, o_1, o_2, o_3)$.
  - Input mapping – $(\phi: x_1 = \phi_1: i_1)$.
  - Output mapping – $(\phi: y_1 = \phi_2: o_1) \wedge (\phi: y_2 = \phi_2: o_2) \wedge (\phi: y_3 = \phi_2: o_3)$.
  - Internal mapping – $(\phi_1: o_1 = \phi_2: i_1) \wedge (\phi_1: o_2 = \phi_2: i_2)$.



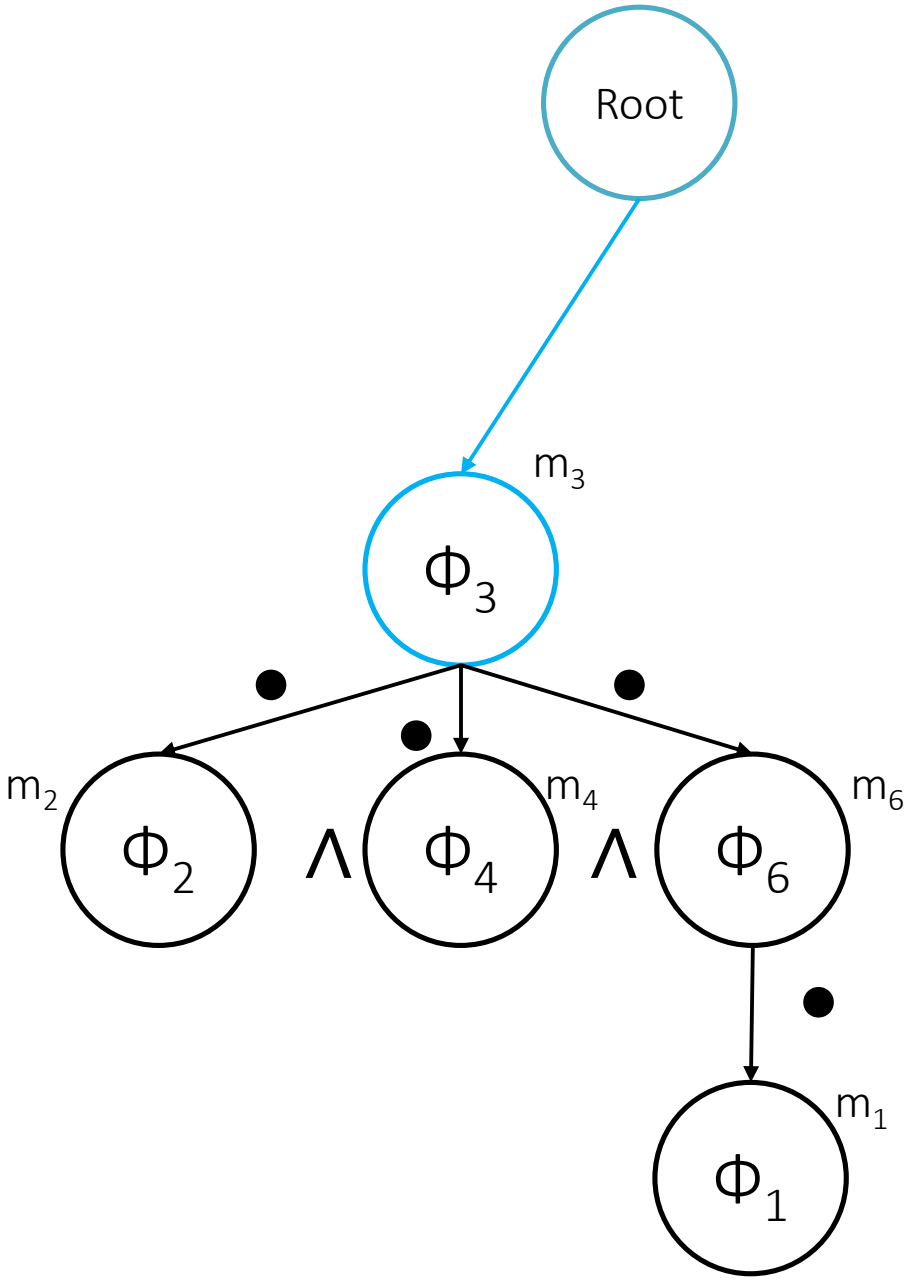
Note: The mapping can be applied to other STL operators, not just concatenation.


Root
m1
Φ1
m2
Φ2
m3
Φ3
m4
Φ4
m5
Φ5
m6
Φ6
m2
Φ2
Λ
m4
Φ4
Λ
m6
Φ6
m1
Φ1

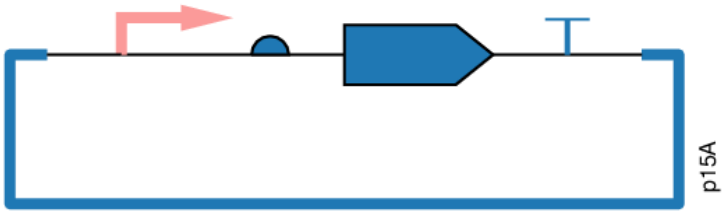


Design

STL Formula

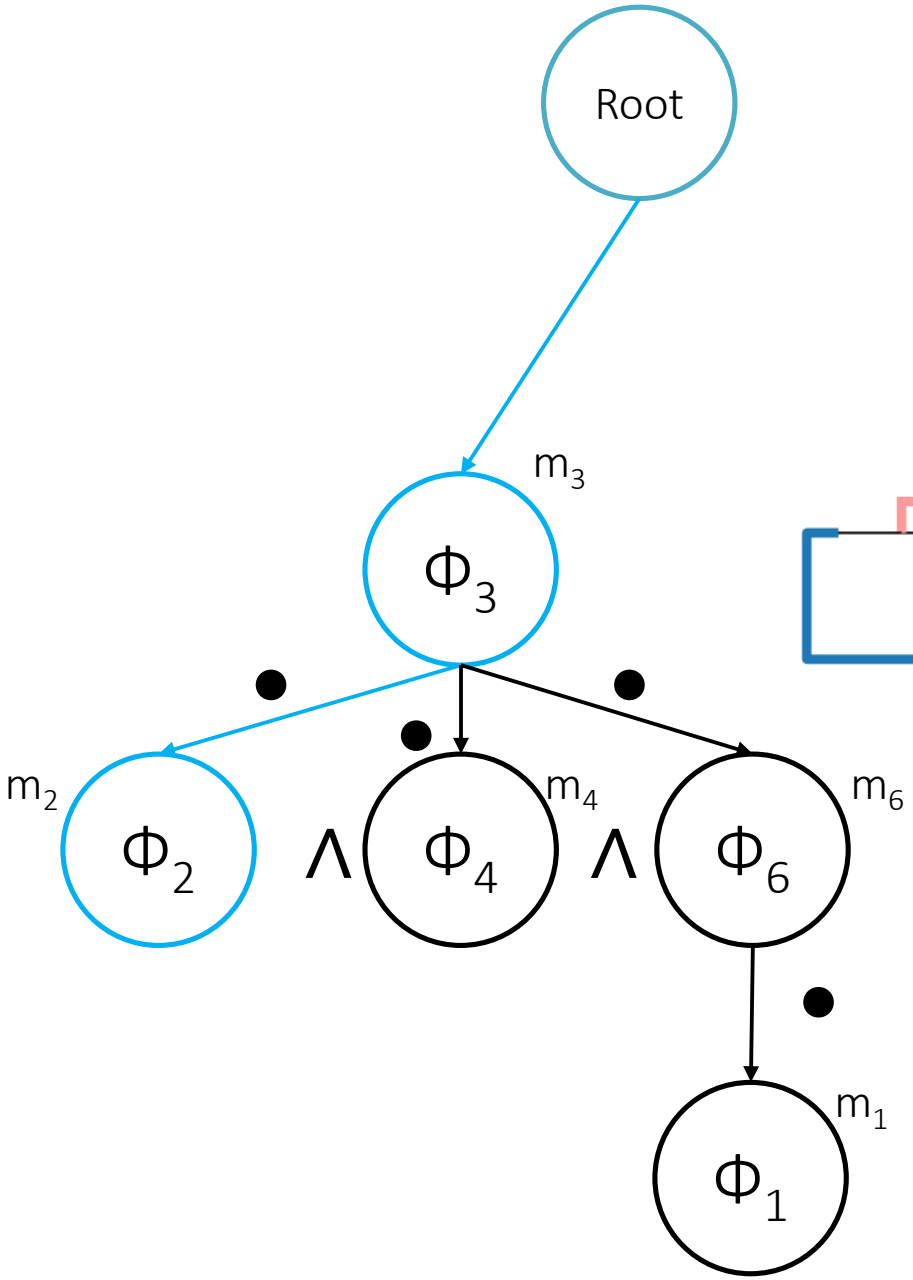


Design

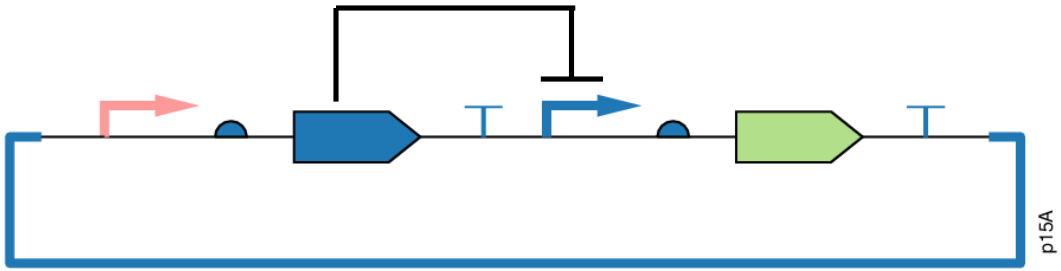


STL Formula

Φ_3

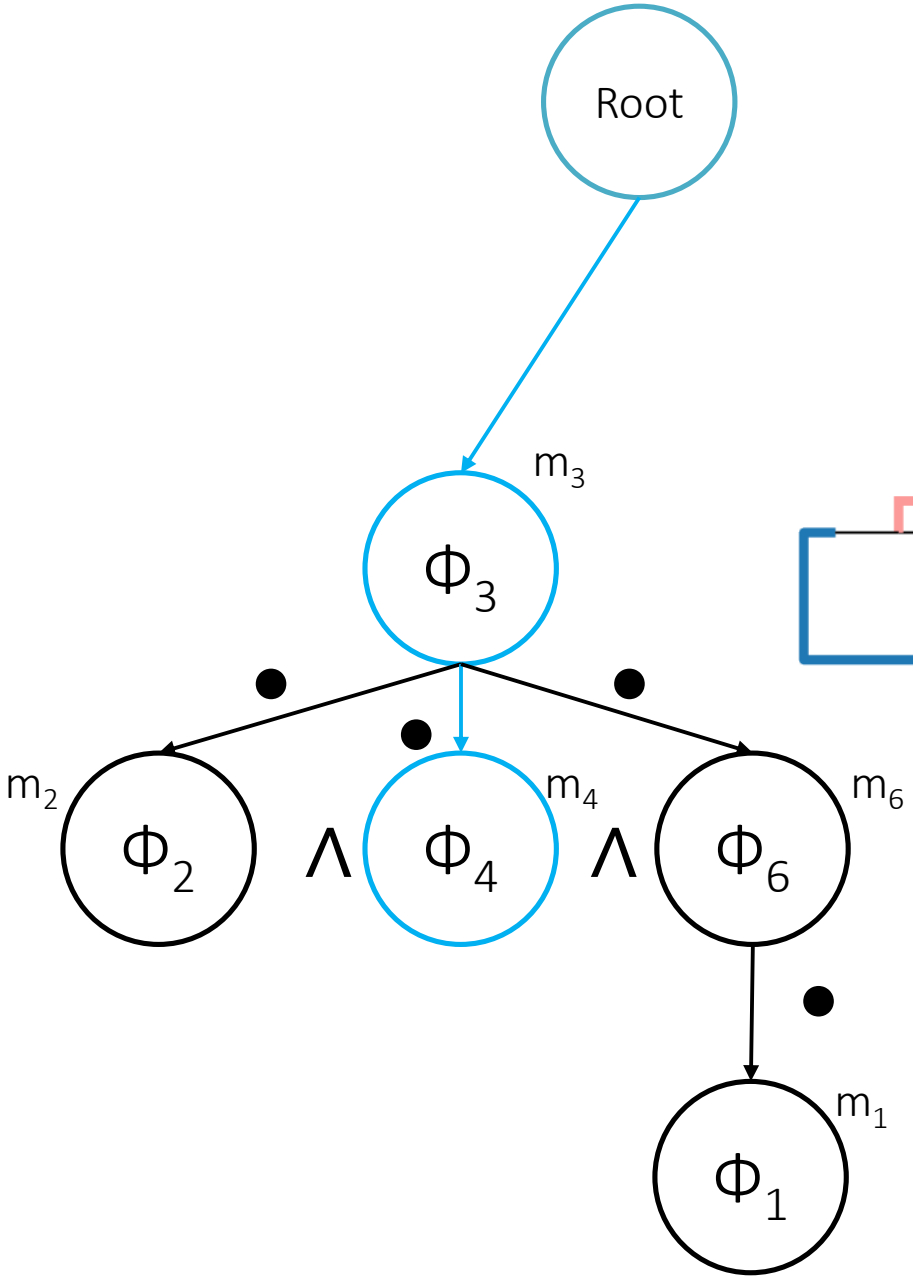


Design

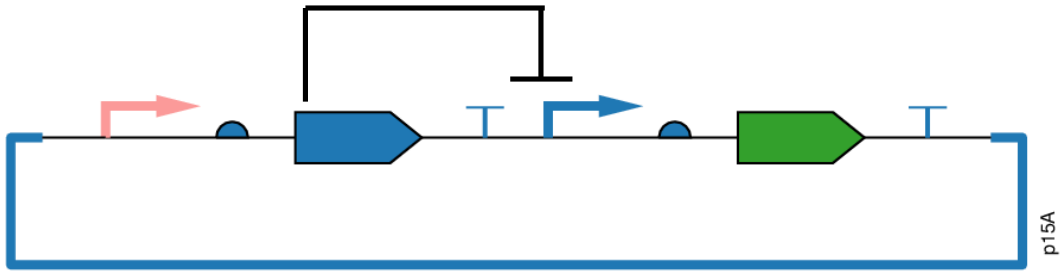


STL Formula

$\Phi_3 \bullet \Phi_2$

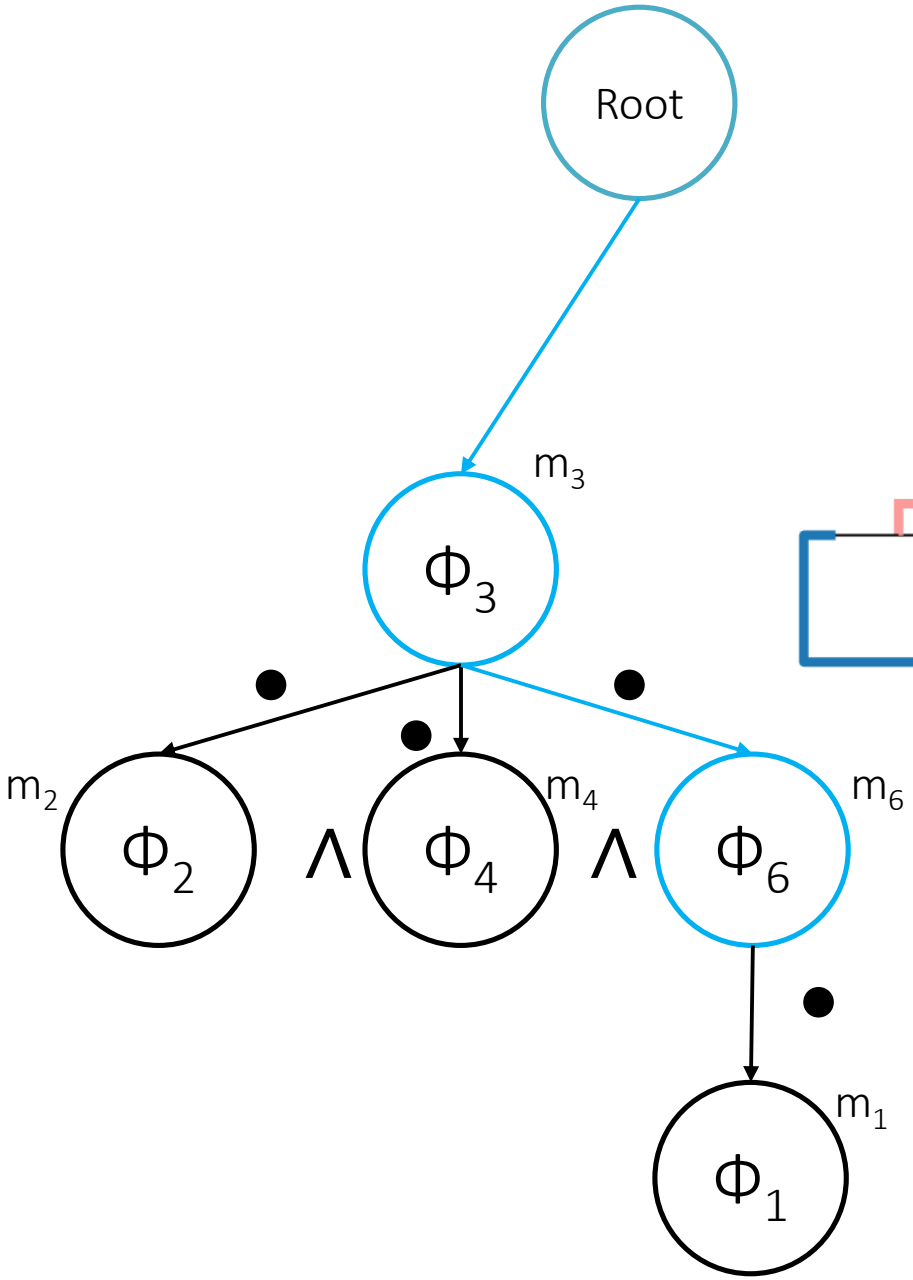


Design

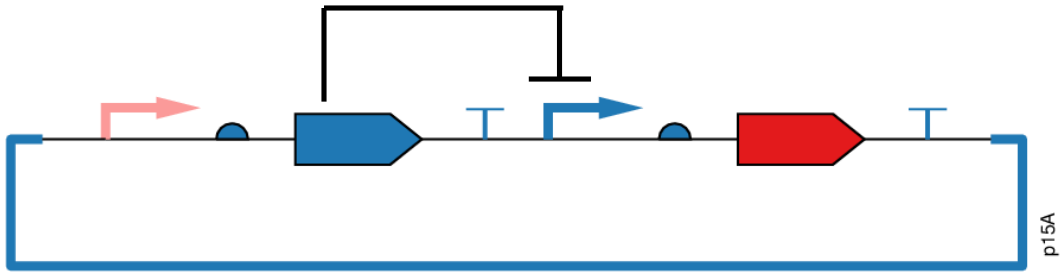


STL Formula

$$\Phi_3 \bullet \Phi_4$$

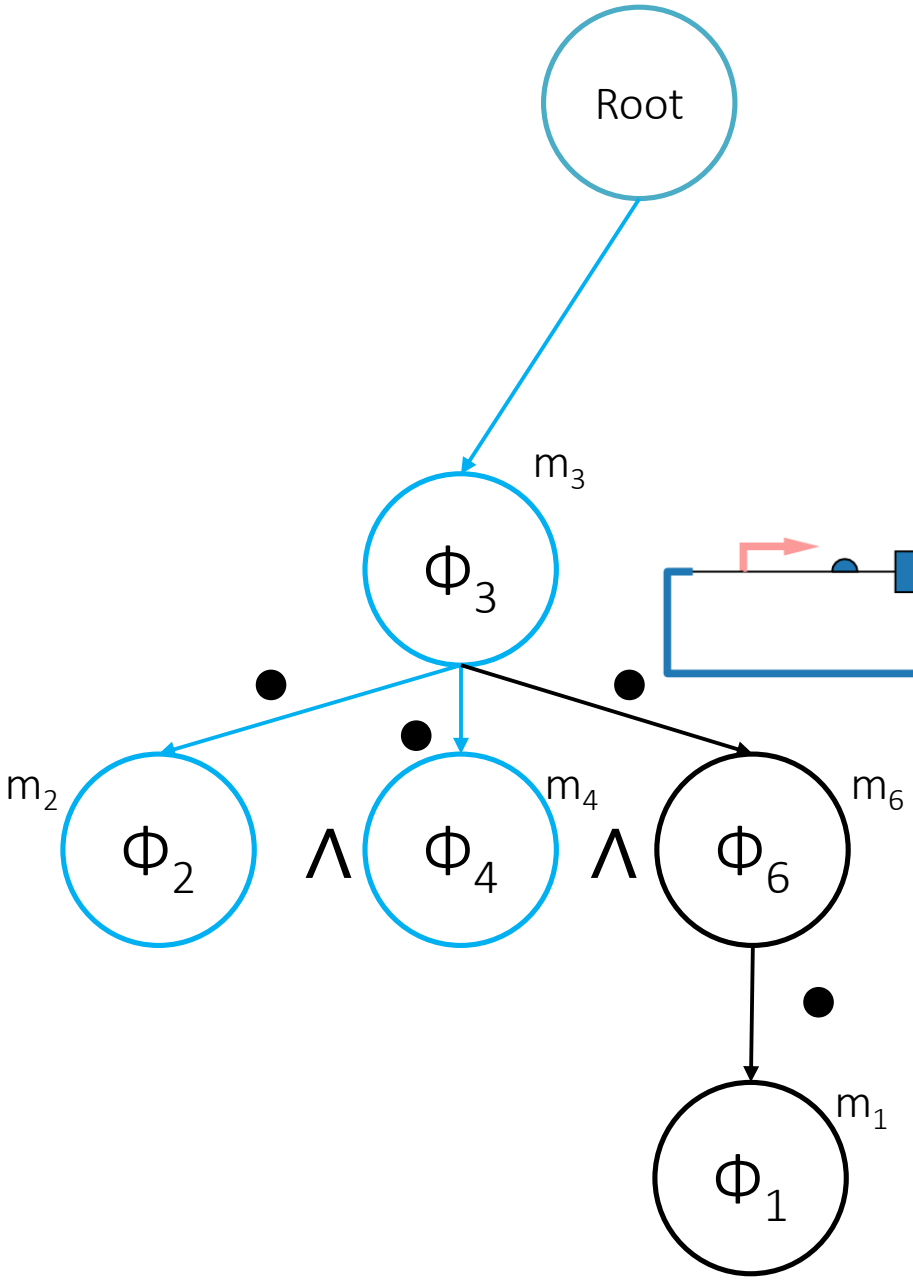


Design

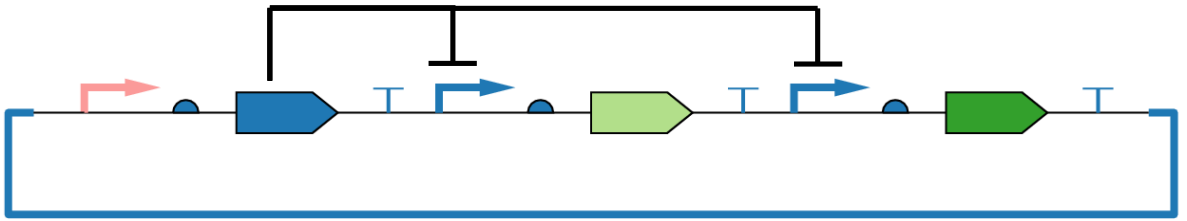


STL Formula

$$\Phi_3 \bullet \Phi_6$$

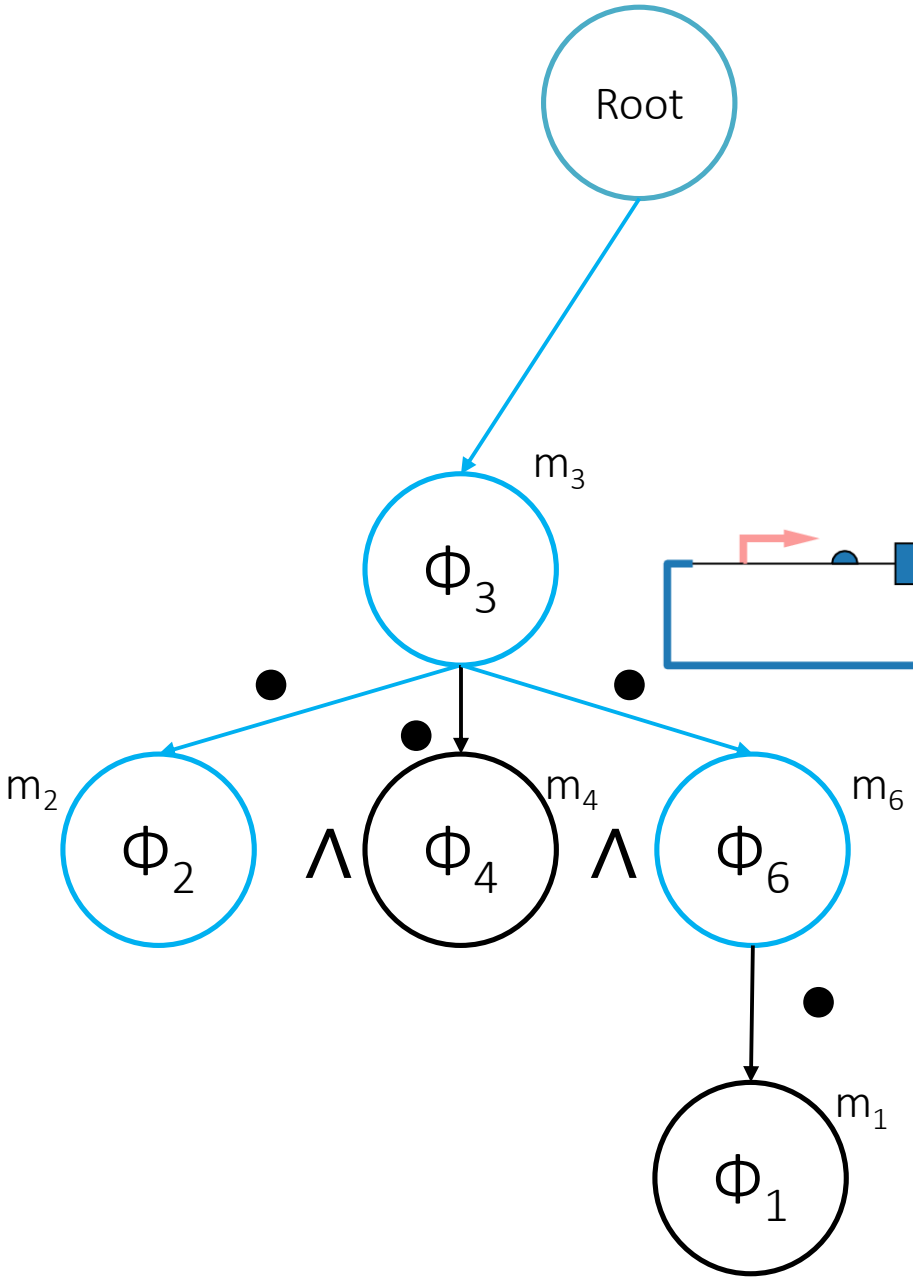


Design

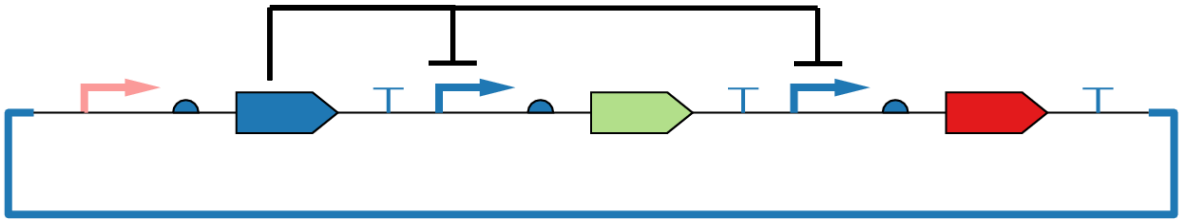


STL Formula

$$\Phi_3 \bullet (\Phi_2 \wedge \Phi_4)$$

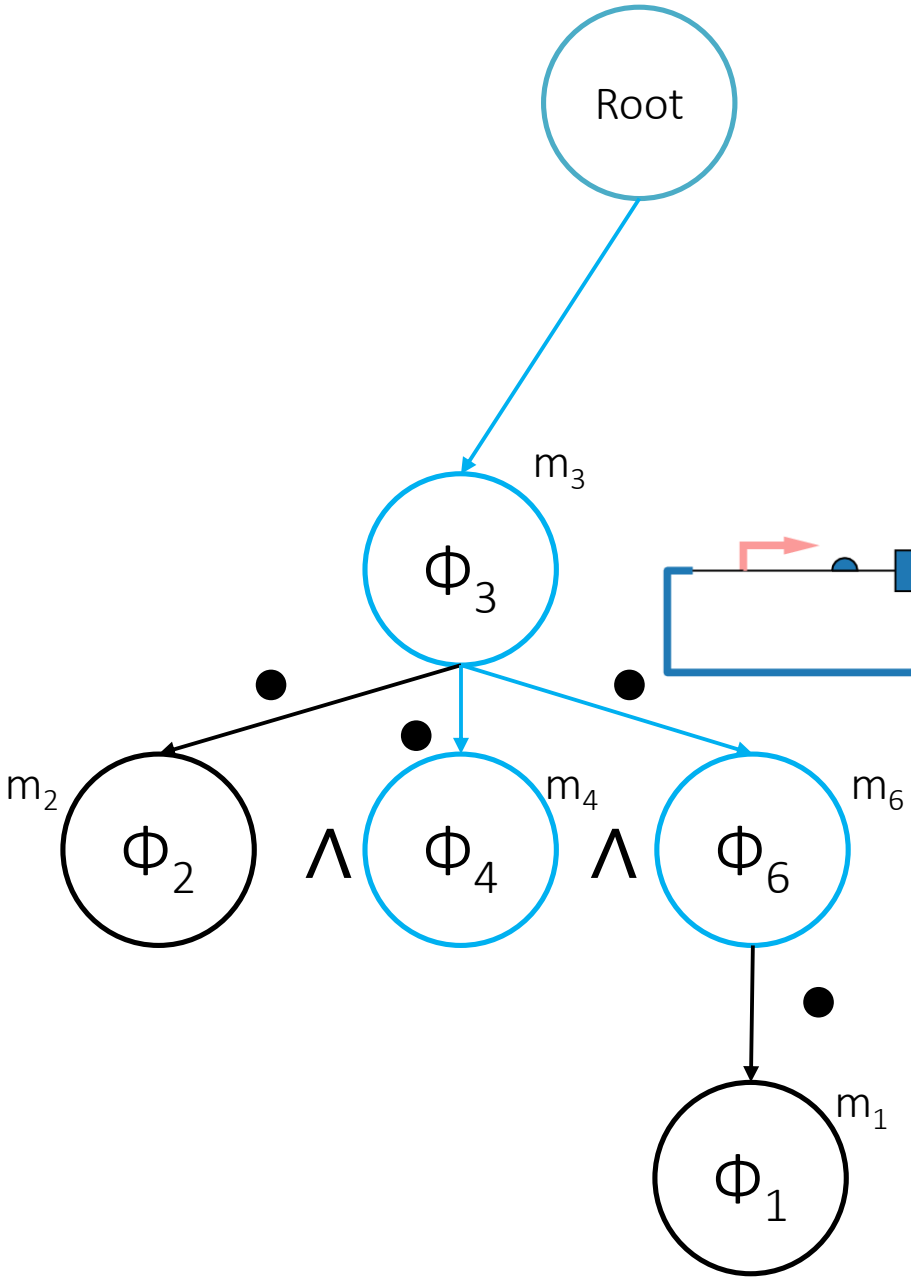


Design

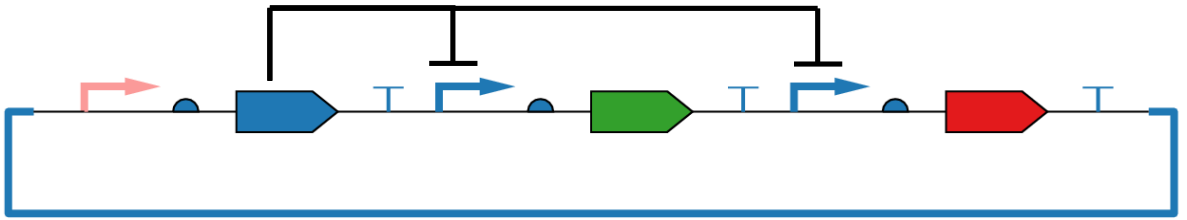


STL Formula

$$\Phi_3 \bullet (\Phi_2 \wedge \Phi_6)$$

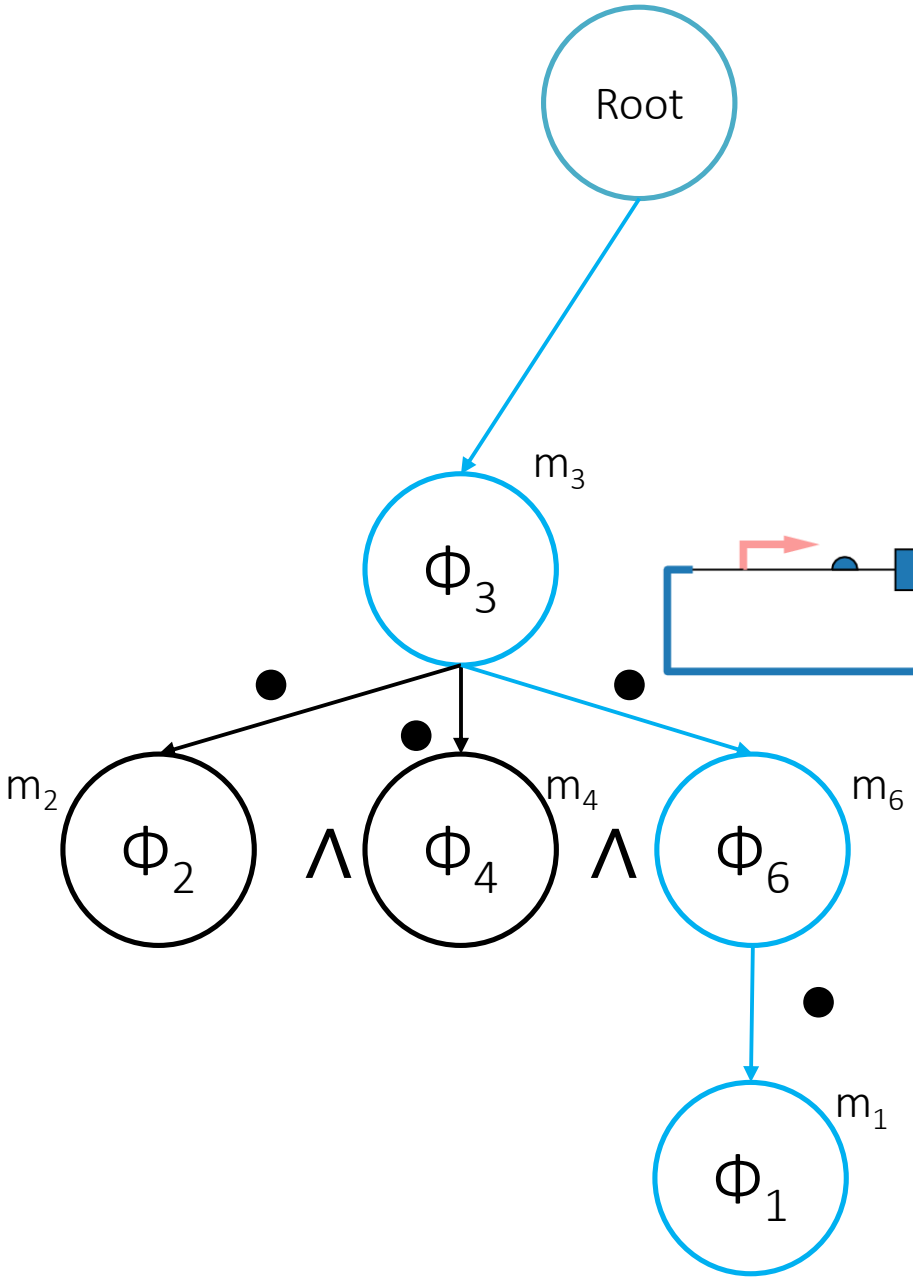


Design

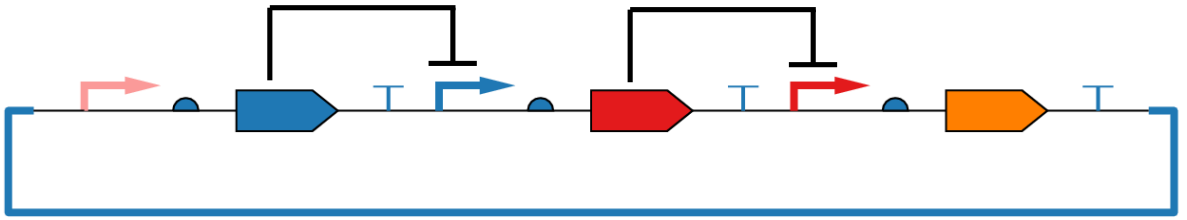


STL Formula

$$\Phi_3 \bullet (\Phi_4 \wedge \Phi_6)$$

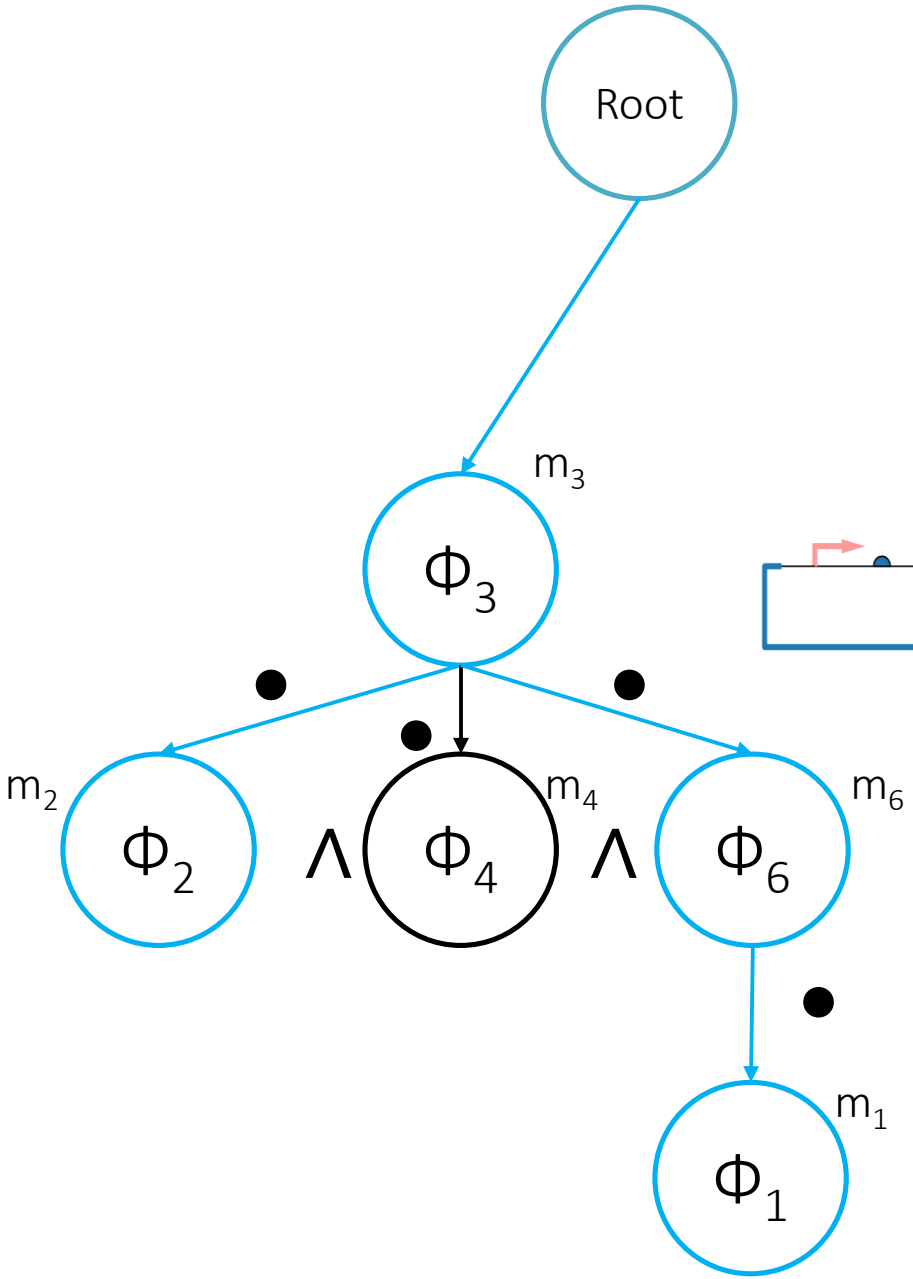


Design

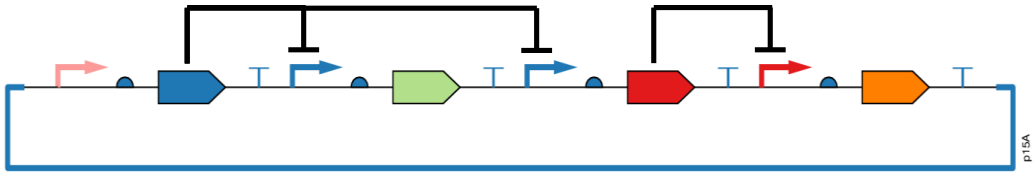


STL Formula

$$\Phi_3 \bullet (\Phi_6 \bullet \Phi_1)$$

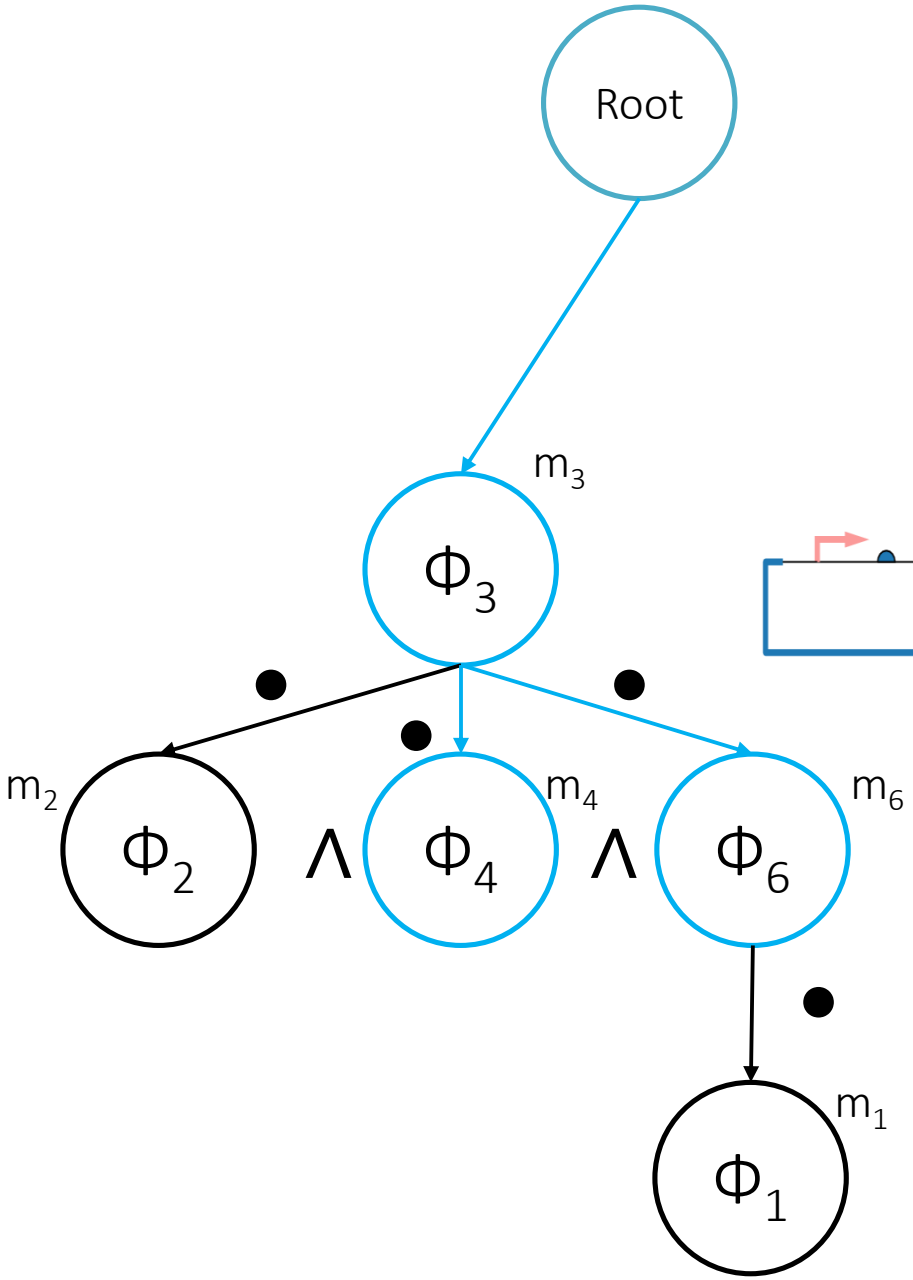


Design

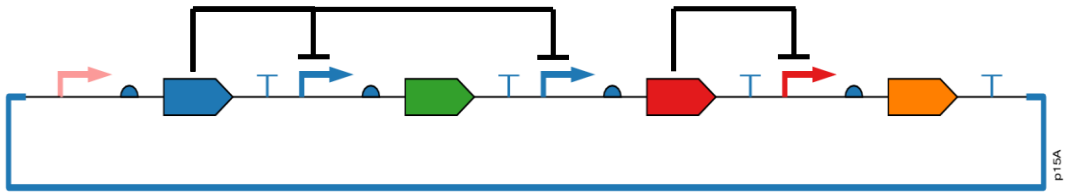


STL Formula

$$\Phi_3 \bullet (\Phi_2 \wedge (\Phi_6 \bullet \Phi_1))$$

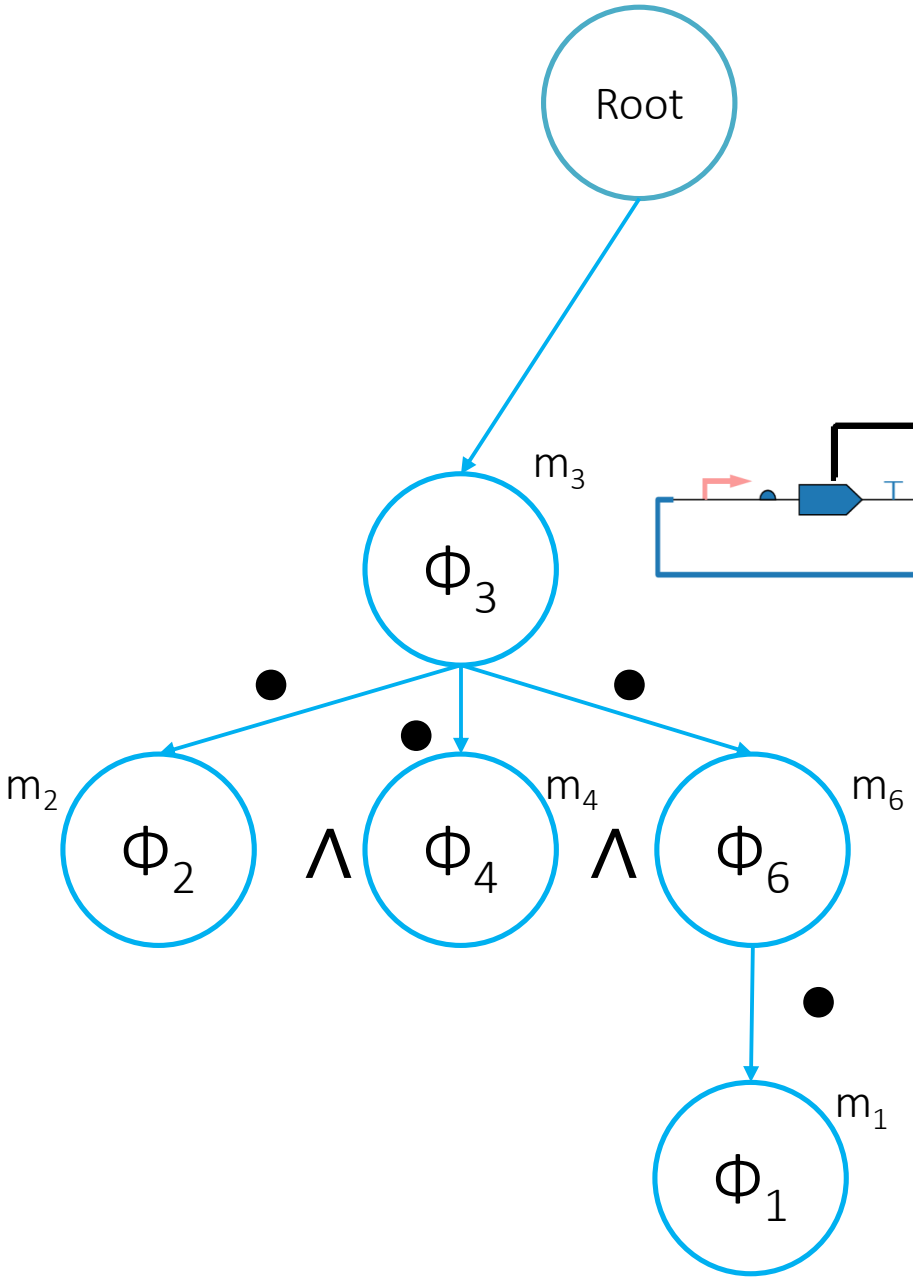


Design

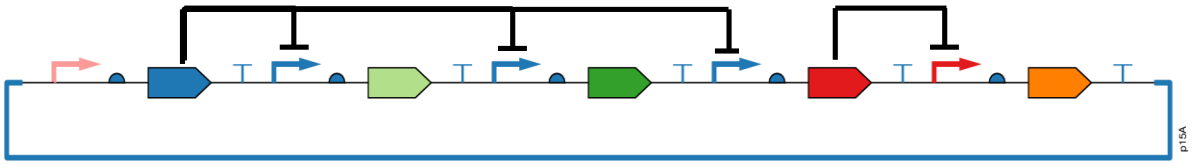


STL Formula

$$\Phi_3 \bullet (\Phi_4 \wedge (\Phi_6 \bullet \Phi_1))$$

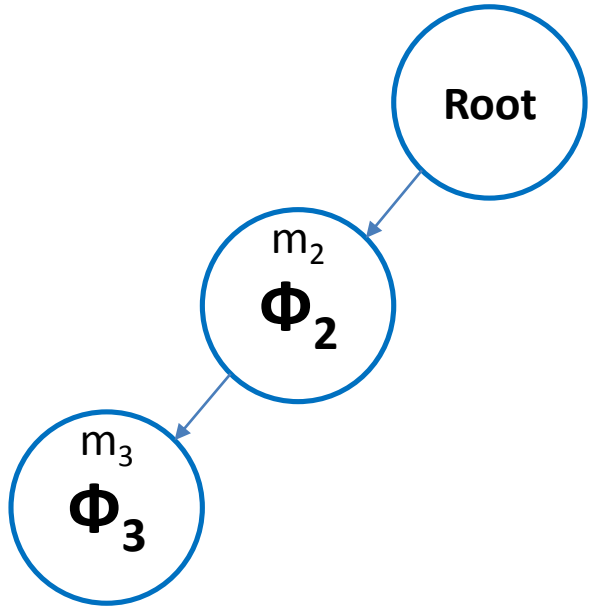


Design

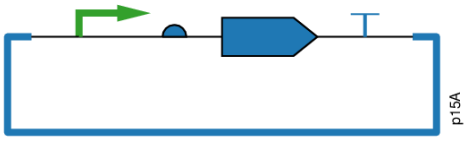


STL Formula

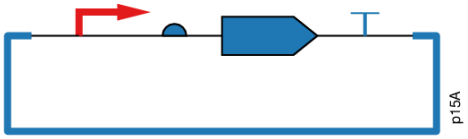
$$\Phi_3 \bullet (\Phi_2 \wedge \Phi_4 \wedge (\Phi_6 \bullet \Phi_1))$$



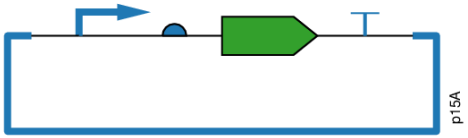
Library



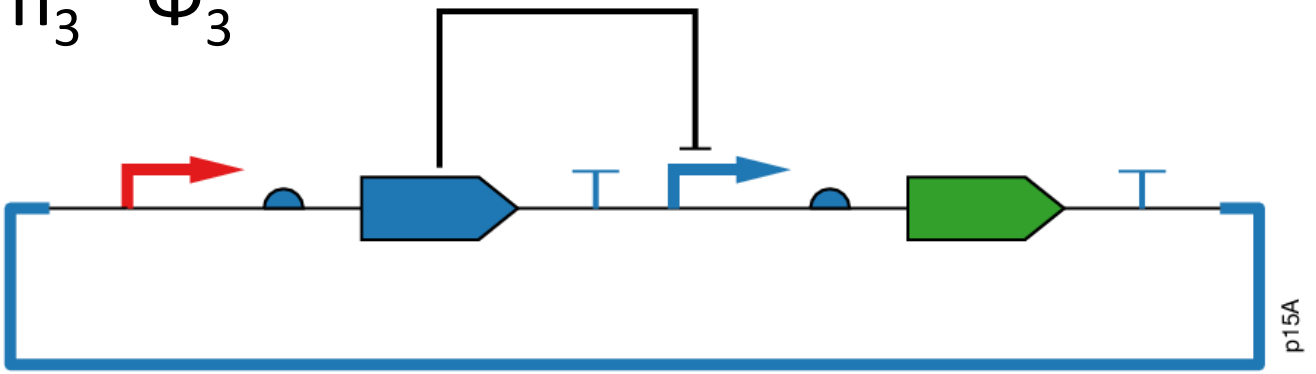
m_1 Φ_1



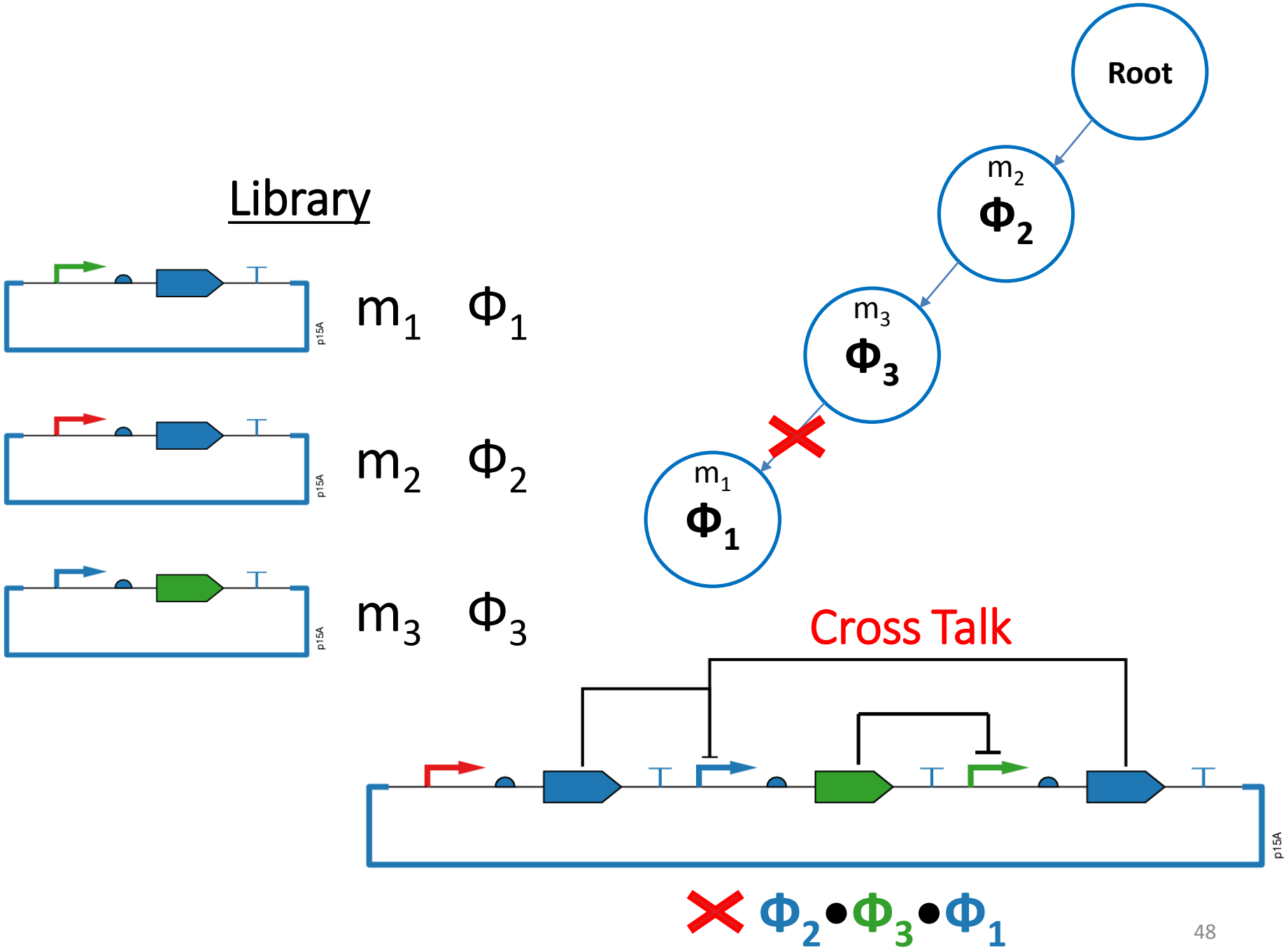
m_2 Φ_2



m_3 Φ_3



$\Phi_2 \bullet \Phi_3$


Library
p15A
m_1 Φ_1
p15A
m_2 Φ_2
p15A
m_3 Φ_3
Root
m_2
Φ_2
m_3
Φ_3
m_1
Φ_1
Cross Talk
p15A
$\Phi_2 \bullet \Phi_3 \bullet \Phi_1$

Functional Synthesis