A Genetic Circuit Compiler

Supplementary Materials

William Waites,[*,†] Göksel Mısırlı,[‡] Matteo Cavaliere,[¶] Vincent Danos,[§,†] and Anil Wipat[‖]

†*School of Informatics, University of Edinburgh*

‡*School of Computing and Mathematics, Keele University*

¶*School of Computing & Mathematics, Manchester Metropolitan University*

§*École Normale Supérieure, Paris, CNRS*

‖*School of Computing Science, Newcastle University*

Extended Output Representation Discussion

Generic Agents

The behaviour of each kind of genetic part can be specified with rules, examples of which are given below. Fundamentally these rules operate on representations of DNA, RNA and proteins. Since each part can be linearly adjacent to others, there must be sites to stand for this linkage. These will be called `us` and `ds` for "upstream" and "downstream" respectively. There is also a need for a site to stand for the binding of protein or RNA polymerase to DNA, or the ribosome to RNA. This will be called `bs` for "binding site".

We immediately arrive at a modelling choice: the specific part, for example an operator to which the Lac repressor binds, could be represented as distinct kind of agent with DNA, RNA and protein variants (Figure 1a) or it could be represented as a label or tag on a generic DNA, RNA and protein agents (Figure 1b). We choose the latter because not only does it

```
%agent: D-LacI(us, ds, bs)
%agent: R-LacI(us, ds, bs)
%agent: P-LacI(us, ds, bs)
```

(a) Distinct agents for each variant of part.

```
%agent: DNA(us, ds, bs, type{LacI})
%agent: RNA(us, ds, bs, type{LacI})
%agent: Protein(us, ds, bs, type{LacI})
```

(b) Generic agents for each variant with part indicated by the `type` site.

Figure 1: Dual representations of parts as agents.

remove the need for having a large number of agents and inventing names for each DNA and RNA variant, but it greatly simplifies the rules. As we shall see the generic representation means that rules can easily be written where it only matters that a part is adjacent to *some* other part without specifying which one in particular. This is simply done by not specifying the `type` site. This is not possible with distinct agents because the Kappa language does not allow for unspecified or wild card agents.

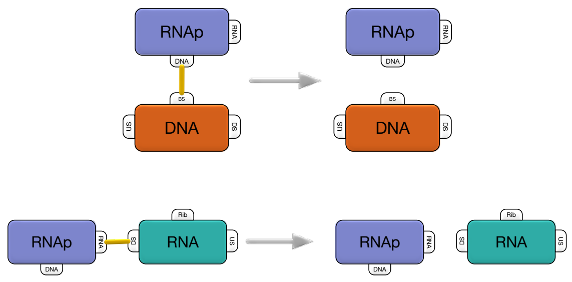
```
%agent: RNAp(dna, rna)
%agent: Ribosome(rna, protein)
```

Figure 2: RNA polymerase and ribosome agents.

These constructs, with their upstream and downstream linkages are enough to form the "rails" along which transcription and translation happen but we still require agents to join these together, namely RNA polymerase and the ribosome. These agents have two sites, one for each rail that they straddle (Figure 2).

Unbinding Rules

To understand how this works in practice, consider the simplest kind of rule, the unbinding rule. Those for transcription and translation are shown in Figure 3. This does not yet use



```
'transcription-termination'
    DNA(bs[1]), RNAp(dna[1])
->
    DNA(bs[.]), RNAp(dna[.])
@k

'translation-termination'
    RNA(ds[1]), RNAp(rna[1])
->
    RNA(ds[.]), RNAp(rna[.])
@k
```

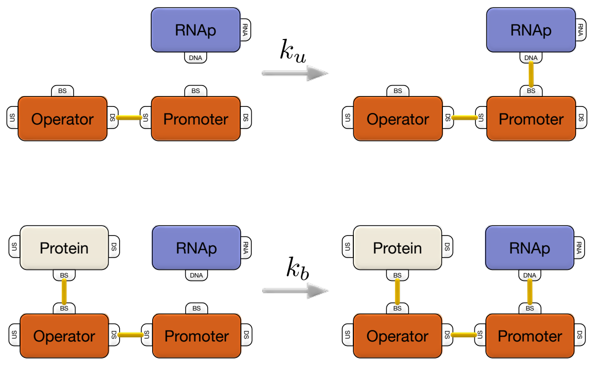
Figure 3: Termination rules: transcription and translation

any of the features that motivated our choice of agent representation, but does already show the "don't care, don't write" way of the KaSim dialect of Kappa: those sites that are not necessary for the operation of the rule do not appear. This brevity is a great boon.

An unbinding rule of the same form exists for each DNA part. Particularly significant among these is the unbinding of a protein from an operator.

Binding Rules with Context

The simplest kind of binding rule is just the same as unbinding with the direction of the arrow reversed. Such rules appear for the initiation of translation — the binding of a ribosome onto a ribosome binding site — as well as for the activation of an operator. These are not reproduced here. Instead, we consider binding rules with context, as in Figure 4.



```
'RNAp-binding-unbound'
    DNA(type{operator}, ds[1], bs[.]),
    DNA(type{promoter}, us[1], bs[.]),
    RNAp(dna)
->
    DNA(type{operator}, ds[1], bs[.]),
    DNA(type{promoter}, us[1], bs[2]),
    RNAp(dna[2])
@k_u

'RNAp-binding-bound'
    DNA(type{operator}, ds[1], bs[_]),
    DNA(type{promoter}, us[1], bs[.]),
    RNAp(dna)
-> \
    DNA(type{operator}, ds[1], bs[_]),
    DNA(type{promoter}, us[1], bs[2]),
    RNAp(dna[2])
@k_b
```

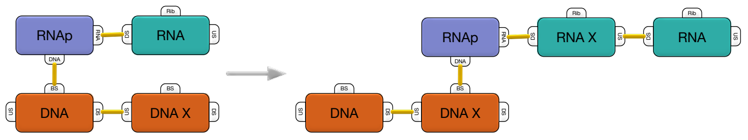
Figure 4: Binding of RNA polymerase to a promoter with different rates, k_u and k_b according to context given by operator state.

The explicit context, with the operator adjacent to the promoter being bound to a protein, or not, allows for the modelling of inducible or repressible promoter architectures. The transcription process begins with the binding of RNA polymerase and the rate at which this happens depends on the state of the operators as illustrated in Figure 4. This is the simple case with only one operator but there is no restriction on the number of operators; we allow for upstream and downstream context of arbitrary size.

This example is illustrative in that rules are posed in terms of a "main" part that becomes bound or unbound and in principle it is possible to provide arbitrary amounts of context for *any* rule. This is supported by the low-level language here, but however it is only implemented in the compiler for the particular family of rules depicted in Figure 4, the activation of promoters through the binding of RNA polymerase. This is sufficient for models involving complex promoter architectures, but an extension allowing for context everywhere is not difficult.

Sliding Rules

A somewhat more complicated sliding rule than the one presented in the *Output Representation* section of the main text is used to implement transcription, as shown in Figure 5. This shares the feature of the translation rule above where there is a part that is central to this rule, part X, and there is an adjacent part whose type does not matter. Here, the RNA



```
'transcription-elongation'
  DNA(ds[2], bs[1]),
  DNA(type{X}, us[2], bs[.]),
  RNAp(dna[1], rna[3]),
  RNA(ds[3]),
  .
->
  DNA(ds[2], bs[.]),
  DNA(type{X}, us[2], bs[1]),
  RNAp(dna[1], rna[3]),
  RNA(ds[4]),
  RNA(type{X}, us[4], ds[3], bs[.])
@k
```

Figure 5: Transcription, production of an RNA sequence from DNA

polymerase starts off bound to the adjacent DNA part, whose type does not matter and so is not specified, and slides onto the central part of type X. In the process, an RNA part of type X is inserted into the growing chain.

Other rules are necessary, of course. The rule in Figure 5, for example, cannot operate without a piece RNA bound to the polymerase. Chains of RNA cannot be produced before the first link has been added. The rule that does that is exactly analogous to that of Figure 8 in the main text. And similarly in the other direction, there is a rule to produce protein chains where a protein already exists and a coding sequence is slid across. This is almost identical to making an RNA chain. All of the other core rules are simply variations on those given above.

The Genetic Circuit Compiler Language

```
# -*- n3 -*-
@prefix dct:  <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix rbmo: <http://purl.org/rbm/rbmo#>.
@prefix gcc: <http://purl.org/rbm/comp#>.
@prefix rbmt: <http://purl.org/rbm/templates/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.

gcc:part a gcc:Token; skos:prefLabel "name".
gcc:Part a owl:Class;
    gcc:tokens gcc:part.

gcc:next a gcc:Token; skos:prefLabel "next".

gcc:transcriptionFactor a gcc:Token;
    skos:prefLabel "transcriptionFactor";
    gcc:default 1.0.
gcc:transcriptionFactorBindingRate a gcc:Token;
    skos:prefLabel "transcriptionFactorBindingRate";
    gcc:default 1.0.
gcc:transcriptionFactorUnbindingRate a gcc:Token;
    skos:prefLabel "transcriptionFactorUnbindingRate";
    gcc:default 1.0.

gcc:rnapBindingRate a gcc:Token;
    skos:prefLabel "rnapBindingRate";
    gcc:default 1.0.
gcc:rnapDNAUnbindingRate a gcc:Token;
    skos:prefLabel "rnapDNAUnbindingRate";
    gcc:default 1.0.
gcc:rnapRNAUnbindingRate a gcc:Token;
    skos:prefLabel "rnapRNAUnbindingRate";
    gcc:default 1.0.

gcc:ribosomeBindingRate a gcc:Token;
    skos:prefLabel "ribosomeBindingRate";
    gcc:default 1.0.
```

```
gcc:ribosomeRNAUnbindingRate a gcc:Token;
    skos:prefLabel "ribosomeRNAUnbindingRate";
    gcc:default 1.0.
gcc:ribosomeProteinUnbindingRate a gcc:Token;
    skos:prefLabel "ribosomeProteinUnbindingRate";
    gcc:default 1.0.

gcc:transcriptionInitiationRate a gcc:Token;
    skos:prefLabel "transcriptionInitiationRate";
    gcc:default 1.0.
gcc:transcriptionElongationRate a gcc:Token;
    skos:prefLabel "transcriptionElongationRate";
    gcc:default 1.0.

gcc:translationElongationRate a gcc:Token;
    skos:prefLabel "translationElongationRate";
    gcc:default 1.0.

gcc:rnaDegradationRate a gcc:Token;
    skos:prefLabel "rnaDegradationRate";
    gcc:default 1.0.
gcc:proteinDegradationRate a gcc:Token;
    skos:prefLabel "proteinDegradationRate";
    gcc:default 1.0.

gcc:overlaps
    rdfs:domain gcc:Part;
    rdfs:range gcc:Part.

gcc:Operator rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:operator.ka;
    gcc:bnglTemplate rbmt:operator.bngl;
    gcc:tokens
        gcc:transcriptionFactor,
        gcc:transcriptionFactorBindingRate,
        gcc:transcriptionFactorUnbindingRate,
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionInitiationRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.

gcc:Promoter rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:promoter.ka;
    gcc:bnglTemplate rbmt:promoter.bngl;
    gcc:tokens
        gcc:next,
        gcc:rnapBindingRate,
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionInitiationRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.

gcc:RibosomeBindingSite rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:rbs.ka;
    gcc:bnglTemplate rbmt:rbs.bngl;
    gcc:tokens
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeBindingRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.

gcc:protein a gcc:Token;
    skos:prefLabel "protein".

gcc:CodingSequence rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:cds.ka;
    gcc:bnglTemplate rbmt:cds.bngl;
    gcc:tokens
        gcc:protein,
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.

gcc:Terminator rdfs:subClassOf gcc:Part;
    gcc:kappaTemplate rbmt:generic.ka;
    gcc:bnglTemplate rbmt:generic.bngl;
    gcc:tokens
        gcc:rnapDNAUnbindingRate,
        gcc:rnapRNAUnbindingRate,
        gcc:transcriptionElongationRate,
        gcc:ribosomeRNAUnbindingRate,
        gcc:ribosomeProteinUnbindingRate,
        gcc:translationElongationRate,
        gcc:rnaDegradationRate,
        gcc:proteinDegradationRate.
```

Additional Inference Rules for GCC

```
# -*- n3 -*-
@prefix dct:  <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix rbmo: <http://purl.org/rbm/rbmo#>.
@prefix gcc: <http://purl.org/rbm/comp#>.
@prefix rbmt: <http://purl.org/rbm/templates/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.

## The preferred label of a part is it's part slug
{ ?part gcc:part ?label } => { ?part skos:prefLabel ?label }.

## Derivation of templates
{ ?part a [ gcc:kappaTemplate ?template ] } => { ?part gcc:kappaTemplate ?template }.
{ ?part a [ gcc:bnglTemplate ?template ] } => { ?part gcc:bnglTemplate ?template }.

## Translation of special predicates to replacement instructions
{ ?kind gcc:tokens ?token .
  ?token skos:prefLabel ?label .
  ?part a ?kind; ?token ?value } =>
{ ?part gcc:replace [ gcc:string ?label; gcc:value ?value ] }.

## overlaps is a symmetric relation
{ ?a gcc:overlaps ?b } => { ?b gcc:overlaps ?a }.
```

Complete Model of the Elowitz Repressilator

```
# -*- n3 -*-
@prefix : <http://id.inf.ed.ac.uk/rbm/examples/repressilator#>.
@prefix dct:  <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix rbmo: <http://purl.org/rbm/rbmo#>.
@prefix gcc: <http://purl.org/rbm/comp#>.
@prefix rbmt: <http://purl.org/rbm/templates/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix skos: <http://www.w3.org/2004/02/skos/core#>.

## Top-level model description.
:m a rbmo:Model;
    ## bibliographic metadata
    dct:title "The Elowitz repressilator constructed from BioBrick parts";
    dct:description "Transcription of the treatment of the Elowitz repressilator given in the Kappa
        BioBricks Framework book chapter";
    rdfs:seeAlso <http://link.springer.com/protocol/10.1007/978-1-4939-1878-2_6>;
    gcc:prefix <http://id.inf.ed.ac.uk/rbm/examples/repressilator#>;
    ## include the host environment
    gcc:include <host.ka>;
    ## The expression of the model as a genetic circuit
    gcc:circular (
        :R0040o :R0040p :B0034a :C0051 :B0011a
        :R0051o :R0051p :B0034b :C0012 :B0011b
        :R0010o :R0010p :B0034c :C0040 :B0011c
    ).

:P0040 a gcc:Protein;
    skos:prefLabel "P0040";
    rdfs:label "TetR".

:P0051 a gcc:Protein;
    skos:prefLabel "P0051";
    rdfs:label "lambda-Cl".

:P0010 a gcc:Protein;
    skos:prefLabel "P0010";
    rdfs:label "LacI".
```

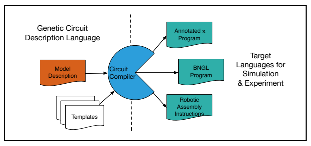
```
:C0051 a gcc:CodingSequence;
    rdfs:label "Coding sequence for lambda-Cl";
    gcc:part "C0051";
    gcc:protein :P0051;
    gcc:proteinDegradationRate 0.0001.

:C0012 a gcc:CodingSequence;
    gcc:label "Coding sequence for LacI";
    gcc:part "C0012";
    gcc:protein :P0010;
    gcc:proteinDegradationRate 0.0001.

:C0040 a gcc:CodingSequence;
    gcc:label "Coding sequence for TetR";
    gcc:part "C0040";
    gcc:protein :P0040;
    gcc:proteinDegradationRate 0.0001.

:B0034a a gcc:RibosomeBindingSite;
    rdfs:label "Ribosome binding site";
    gcc:part "B0034a".

:B0011a a gcc:Terminator;
    rdfs:label "Terminator, stop codon";
    gcc:part "B0011a".

:B0034b a gcc:RibosomeBindingSite;
    rdfs:label "Ribosome binding site";
    gcc:part "B0034b".

:B0011b a gcc:Terminator;
    rdfs:label "Terminator, stop codon";
    gcc:part "B0011b".

:B0034c a gcc:RibosomeBindingSite;
    rdfs:label "Ribosome binding site";
    gcc:part "B0034c".

:B0011c a gcc:Terminator;
    rdfs:label "Terminator, stop codon";
    gcc:part "B0011c".

:R0040o a gcc:Operator;
```

```
    rdfs:label "TetR activated operator";
    gcc:part "R0040o";
    gcc:transcriptionFactor :P0040;
    gcc:transcriptionFactorBindingRate 0.01;
    gcc:transcriptionFactorUnbindingRate 0.01.

:R0040p a gcc:Promoter;
    rdfs:label "TetR repressible promoter";
    gcc:part "R0040p";
    gcc:next "B0034a";
    gcc:rnapBindingRate [
      gcc:upstream ( [a rbmo:BoundState; rbmo:stateOf :R0040o] );
      gcc:value 7e-7
    ], [
      gcc:upstream ( [a rbmo:UnboundState; rbmo:stateOf :R0040o] );
      gcc:value 0.0007
    ].

:R0051o a gcc:Operator;
    rdfs:label "lambda-Cl activated operator";
    gcc:part "R0051o";
    gcc:transcriptionFactor :P0051;
    gcc:transcriptionFactorUnbindingRate 0.01;
    gcc:transcriptionFactorBindingRate 0.01.

:R0051p a gcc:Promoter;
    rdfs:label "lambda-Cl repressible promoter";
    gcc:part "R0051p";
    gcc:next "B0034b";
    gcc:rnapBindingRate [
        gcc:upstream ( [a rbmo:BoundState; rbmo:stateOf :R0051o] );
        gcc:value 7e-7
    ], [
        gcc:upstream ( [a rbmo:UnboundState; rbmo:stateOf :R0051o] );
        gcc:value 0.0007
    ].

:R0010o a gcc:Operator;
    rdfs:label "LacI activated operator";
    gcc:part "R0010o";
    gcc:transcriptionFactor :P0010;
    gcc:transcriptionFactorBindingRate 0.01;
    gcc:transcriptionFactorUnbindingRate 0.01.
```

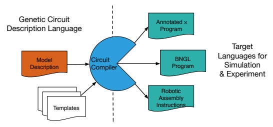
```
:R0010p a gcc:Promoter;
    rdfs:label "LacI repressible promoter";
    gcc:part "R0010p";
    gcc:next "B0034c";
    gcc:rnapBindingRate [
        gcc:upstream ( [a rbmo:BoundState; rbmo:stateOf :R0010o] );
        gcc:value 7e-7
    ], [
        gcc:upstream ( [a rbmo:UnboundState; rbmo:stateOf :R0010o] );
        gcc:value 0.0007
    ].
```

For Table of Contents Use Only

With a box:



Without a box:



This graphic is `figures/CompilerFlow.pdf`