A brief description of the CRISPR circuit using SBOL 2.0 data model

We first give a brief description of the CRISPR-based repression module. We use bold font in the following text and figure captions to mark available data model in SBOL 2.0. Detailed description of properties of the data model is available in the Specification (Data Model 2.0).

First, consider the CRISPR-based Repression Template **ModuleDefinition** shown in the center of Figure 1. It provides a generic description of CRISPR-based repression behavior. Namely, it includes generic *Cas9*, *guide RNA* (gRNA), and *target* DNA **FunctionalComponent** instances. It also includes a *genetic production* **Interaction** that expresses a generic target gene product. Finally, it includes a *non-covalent binding* **Interaction** that forms the Cas9/gRNA complex (shown as dashed arrows), which in turn participates in an *inhibition* **Interaction** to repress the target gene product production (shown with a tee-headed arrow). The CRISPR-based Repression Template is then instantiated to test a particular CRISPR-based repression device, CRPb, by the outer CRPb Characterization Circuit **ModuleDefinition**. This outer characterization circuit includes gene **FunctionalComponents** to produce specific products (i.e., mKate, Gal4VP16, cas9m_BFP, gRNA_b, and EYFP), as well as **FunctionalComponents** for the products themselves. Next, it includes *genetic production* **Interactions** connecting the genes to their products, and it has a *stimulation* **Interaction** that indicates that Gal4VP16 stimulates production of EYFP. Finally, it uses **MapsTo** objects (shown as dashed lines) to connect the generic **FunctionalComponents** in the template to the specific objects in the outer **ModuleDefinition**. For example, the outer module indicates that the target protein is EYFP, while the cas9_gRNA complex is cas9m_BFP_gRNA_b.

Modeling CRISPR repression using `pysbol`

Creating SBOL Document

All SBOL data objects are organized within an **SBOLDocument** object. The **SBOLDocument** provides a rich set of methods to create, access, update, and delete each type of **TopLevel** object (i.e., **Collection**, **ModuleDefinition**, **ComponentDefinition**, **Sequence**, **Model**, or **GenericTopLevel**). Every SBOL object has a *uniform resource identifier* (URI) and consists of properties that may refer to other objects, including non-**TopLevel** objects such as SequenceConstraint and Interaction objects. `pySBOL` organizes the URI collections to enable efficient access

Adding CRISPR-based Repression Template module

Creating TopLevel objects

We first create the CRISPR-based Repression Template module shown in Figure 1. In this template, we include definitions for generic *Cas9*, *guide RNA* (gRNA), and *target* DNA **FunctionalComponent** instances. They are encoded as **ComponentDefinition** objects. Creation of the generic Cas9 (line 3) **ComponentDefinition** is done by passing its *displayId* "cas9_generic", *version* specified by the `version` string, and *type* to the `ComponentDefinition` constructor. Every **ComponentDefinition** must contain one or more types, each of which is specified by a URI. A type specifies the component's category of biochemical or physical entity (for example DNA, protein, or small molecule). The generic Cas9's type is `PROTEIN`, which is defined as the BioPAX ontology term for protein (`http://www.biopax.org/release/biopax-level3.owl#Protein`). Other **ComponentDefinition** objects shown below are created in the same way. A **ComponentDefinition** object can optionally have one or more roles, also in the form of URIs. The `gRNA_generic` has a role of `SGRNA` (line 11 below), defined as the *Sequence Ontology* (SO) term "SO:0001998" (`http://identifiers.org/so/SO:0001998`) in the library. Similarly, the `target_gene` on line 21 below has a role of `PROMOTER`, defined as SO term "SO:0000167" (`http://identifiers.org/so/SO:0000167`). We then create the **ModuleDefinition** template with the displayId "CRISPR_Template" and `version`.

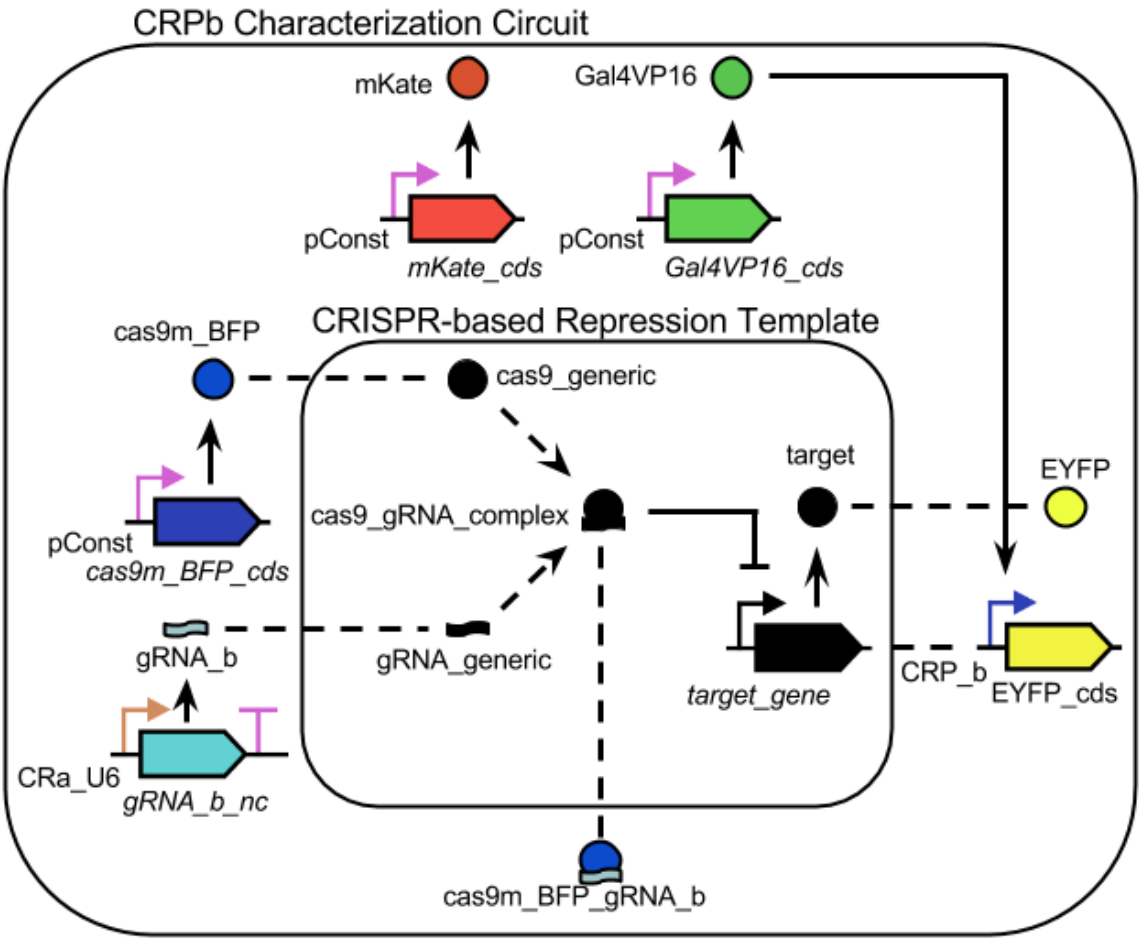

Figure 1: Illustration of a hierarchical CRISPR-based repression module represented in SBOL 2.0 (adapted from Figure 1a in [1]). The CRISPR-based Repression Template **ModuleDefinition** describes a generic CRISPR repression circuit that combines a Cas9 protein with a gRNA to form a complex (represented by the dashed arrows) that represses a target gene (represented by the arrow with the tee arrowhead). These relationships between these **FunctionalComponents** (instances of **ComponentDefinitions**) are represented in SBOL 2.0 using **Interactions**. This **Module** is instantiated in the outer CRPb Characterization Circuit **ModuleDefinition** in order to specify the precise (including **Sequences** when provided) **FunctionalComponents** used for each generic **FunctionalComponent**. The undirected dashed lines going into the template **Module** represent **MapsTo** objects that specify how specific **FunctionalComponents** replace the generic ones.

```
1
2  from sbol import *
3  v = "1.0";
4  ns = "sys-bio.org"
5
6  cas9_generic = ComponentDefinition("cas9_generic", BIOPAX_PROTEIN, v)
7  gRNA_generic = ComponentDefinition("gRNA_generic", BIOPAX_RNA, v)
8  cas9_gRNA_complex = ComponentDefinition("cas9_gRNA_complex", BIOPAX_COMPLEX, v)
9  generic_target = ComponentDefinition("generic_target", BIOPAX_DNA, v)
10 generic_target_gene = ComponentDefinition("generic_target_gene", BIOPAX_PROTEIN, v
       )
11
12
13 CRISPRTemplate = ModuleDefinition(ns, "CRISPRTemplate", v)
```

Each of create methods in the library creates a *compliant URI* with the following form:

http://⟨prefix⟩/⟨displayId⟩/⟨version⟩

using the default URI prefix and provided displayId and version. The ⟨*prefix*⟩ represents a URI for a namespace (for example, `www.sbols.org/CRISPR_Example`). The author of a **TopLevel** object, such as the **ModuleDefinition** object we just created, should use a URI prefix that either they own or an organization of which they are a member owns. When using compliant URIs, the owner of a prefix must ensure that the URI of any unique **TopLevel** object that contains the prefix also contains a unique ⟨*displayId*⟩ or ⟨*version*⟩ portion. Multiple versions of an SBOL object can exist and would have compliant URIs that contain identical prefixes and displayIds, but each of these URIs would need to end with a unique version. Lastly, the compliant URI of a non-**TopLevel** object is identical to that of its parent object, except that its displayId is inserted between its parent's displayId and version. This form of compliant URIs is chosen to be easy to read, facilitate debugging, and support a more efficient means of looking up objects and checking URI uniqueness.

Specifying Interactions

We are now ready to specify the interactions in the repression template. The first one is the complex formation interaction for `cas9_generic` and `gRNA_generic`. We first create an **Interaction** object `Cas9Complex_Formation` in `CRISPR_Template`, with the displayId "cas9_complex_formation" and a non-covalent binding type (line 1 to 4). It is recommended that terms from the *Systems Biology Ontology* (SBO) [2] are used specify the types for interactions. Table 11 of the Specification (Data Model 2.0) document provides a list of possible SBO terms for the types property and their corresponding URIs.

Next, we create three participants to this interaction object. Each participant is modeled as a **Participation** object and it refers to the corresponding **FunctionalComponent** object. For example, the `participant_cas9_generic` **Participation** object refers to the `cas9_generic` **FunctionalComponent** object. Note that such an object does not exist yet, but since there exists a `cas9_generic` **ComponentDefinition** object, and the "createDefaults" of this **SBOLDocuemnt** is set to **`true`**, a **FunctionalComponent** with the same displayId "cas9_generic" is automatically created and it connects the reference from the `participant_cas9_generic` **Participation** to the `cas9_generic` **ComponentDefinition**. We also assign a `REACTANT` role to it. The remaining two participants are created in a similar fashion.

```
Cas9Complex_Formation= Interaction(ns, "Cas9Complex_Formation", v,
    SBO_NONCOVALENT_BINDING)
CRISPRTemplate.interactions.add(Cas9ComplexFormation)

participant_cas9_generic = Participation(ns, "participant_cas9_generic", v, "sys-
    bio.org/FunctionalComponent/participant_cas9_generic/1.0.0")

participant_gRNA_generic  = Participation(ns,"participant_gRNA_generic ", v, "sys-
    bio.org/FunctionalComponent/participant_gRNA_generic/1.0.0")

participant_cas9_gRNA_complex = Participation(ns,"participant_cas9_gRNA_complex",
    v, "sys-bio.org/FunctionalComponent/participant_cas9_gRNA_complex/1.0.0")

participant_cas9_generic.roles.set( SBO_REACTANT );
participant_gRNA_generic.roles.set( SBO_REACTANT );
participant_cas9_gRNA_complex.roles.set( SBO_PRODUCT );

Cas9Complex_Formation.participations.add(participant_cas9_generic);
Cas9Complex_Formation.participations.add(participant_gRNA_generic);
Cas9Complex_Formation.participations.add(participant_cas9_gRNA_complex);
```

The remaining two interactions, namely the genetic production of the target protein from the `target_gene` and the inhibition of the target protein by the `cas9_gRNA_complex`, are specified using the same method calls.

```
// Production of target from target gene
EYFP_production= Interaction(ns, "EYFP_production", v, SBO_GENETIC_PRODUCTION)

participant_target_gene = Participation(ns, "participant_target_gene", v, "sys-bio
    .org/FunctionalComponent/participant_target_gene/1.0.0")
participant_target_gene.roles.set( SBO_PROMOTER )

participant_target = Participation(ns, "participant_target ", v, "sys-bio.org/
    FunctionalComponent/participant_target/1.0.0")
participant_target.roles.set( SBO_PRODUCT )

EYFP_production.participations.addparticipant_target_gene)
EYFP_production.participations.add(participant_target)

target_generic_gene_inhibition= Interaction(ns, "target_generic_gene_inhibition",
    v, SBO_INHIBITION)

participant_cas9_gRNA_complex = Participation(ns, "participant_cas9_gRNA_complex",
     v, "sys-bio.org/FunctionalComponent/participant_cas9_gRNA_complex/1.0.0")

participant_cas9_gRNA_complex.roles.set( SBO_INHIBITOR);

participant_target_gene = Participation(ns, "participant_target_gene", v, "sys-bio
    .org/FunctionalComponent/participant_target_gene/1.0.0")

participant_target_gene.roles.set( SBO_PROMOTER );

target_generic_gene_inhibition.participations.add(participant_cas9_gRNA_complex);
target_generic_gene_inhibition.participations.add(participant_target_gene);
```

Creating CRPb Characterization Circuit

So far, we have completed the repression template. In order to build the CRPb Characterization Circuit, we need to add precise (including **Sequences** when provided) **FunctionalComponents** used for each generic **FunctionalComponent** in the template. We first create **Sequence** objects for those provided in [1][1] as shown in the code below. For example, to create the sequence for the CRP_b promoter, we call the `createSequence` method, as shown on line 57, with the displayId "CRP_b_seq", `version`, the sequence specified by `CRP_b_seq_elements`, and the IUPAC encoding for DNA, which is defined as a URI in the **Sequence** class, referencing `http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html`.

[1] Unfortunately, as usual, not all sequences are provided in the paper.

```
// Create Sequence for CRa_U6 promoter
CRa_U6_seq_elements = "GGTTTACCGAGCTCTTATTGGTTTTCAAACTTCATTGACTGTGCC" +
                "AAGGTCGGGCAGGAAGAGGGCCTATTTCCCATGATTCCTTCATAT" +
                "TTGCATATACGATACAAGGCTGTTAGAGAGATAATTAGAATTAAT" +
                "TTGACTGTAAACACAAAGATATTAGTACAAAATACGTGACGTAGA" +
                "AAGTAATAATTTCTTGGGTAGTTTGCAGTTTTAAAATTATGTTTT" +
                "AAAATGGACTATCATATGCTTACCGTAACTTGAAATATAGAACCG" +
                "ATCCTCCCATTGGTATATATTATAGAACCGATCCTCCCATTGGCT" +
                "TGTGGAAAGGACGAAACACCGTACCTCATCAGGAACATGTGTTTA" +
                "AGAGCTATGCTGGAAACAGCAGAAATAGCAAGTTTAAATAAGGCT" +
                "AGTCCGTTATCAACTTGAAAAAGTGGCACCGAGTCGGTGCTTTTT" +
                "TTGGTGCGTTTTTATGCTTGTAGTATTGTATAATGTTTTT";


// Create Sequence for gRNA_b coding sequence
gRNA_b_elements = "AAGGTCGGGCAGGAAGAGGGCCTATTTCCCATGATTCCTTCATAT" +
                "TTGCATATACGATACAAGGCTGTTAGAGAGATAATTAGAATTAAT" +
                "TTGACTGTAAACACAAAGATATTAGTACAAAATACGTGACGTAGA" +
                "AAGTAATAATTTCTTGGGTAGTTTGCAGTTTTAAAATTATGTTTT" +
                "AAAATGGACTATCATATGCTTACCGTAACTTGAAAGTATTTCGAT" +
                "TTCTTGGCTTTATATATCTTGTGGAAAGGACGAAACACCGTACCT" +
                "CATCAGGAACATGTGTTTAAGAGCTATGCTGGAAACAGCAGAAAT" +
                "AGCAAGTTTAAATAAGGCTAGTCCGTTATCAACTTGAAAAAGTGG" +
                "CACCGAGTCGGTGCTTTTTTT";


// Create Sequence for mKate
mKate_seq_elements = "TCTAAGGGCGAAGAGCTGATTAAGGAGAACATGCACATGAAGCTG" +
                "TACATGGAGGGCACCGTGAACAACCACCACTTCAAGTGCACATCC" +
                "GAGGGCGAAGGCAAGCCCTACGAGGGCACCCAGACCATGAGAATC" +
                "AAGGTGGTCGAGGGCGGCCCTCTCCCCTTCGCCTTCGACATCCTG" +
                "GCTACCAGCTTCATGTACGGCAGCAAAACCTTCATCAACCACACC" +
                "CAGGGCATCCCCGACTTCTTTAAGCAGTCCTTCCCTGAGGTAAGT" +
                "GGTCCTACCTCATCAGGAACATGTGTTTTAGAGCTAGAAATAGCA" +
                "AGTTAAAATAAGGCTAGTCCGTTATCAACTTGAAAAAGTGGCACC" +
                "GAGTCGGTGCTACTAACTCTCGAGTCTTCTTTTTTTTTTTCACAG" +
                "GGCTTCACATGGGAGAGAGTCACCACATACGAAGACGGGGGCGTG" +
                "CTGACCGCTACCCAGGACACCAGCCTCCAGGACGGCTGCCTCATC" +
                "TACAACGTCAAGATCAGAGGGGTGAACTTCCCATCCAACGGCCCT" +
                "GTGATGCAGAAGAAAACACTCGGCTGGGAGGCCTCCACCGAGATG" +
                "CTGTACCCCGCTGACGGCGGCCTGGAAGGCAGAAGCGACATGGCC" +
                "CTGAAGCTCGTGGGCGGGGGCCACCTGATCTGCAACTTGAAGACC" +
                "ACATACAGATCCAAGAAACCCGCTAAGAACCTCAAGATGCCCGGC" +
                "GTCTACTATGTGGACAGAAGACTGGAAAGAATCAAGGAGGCCGAC" +
                "AAAGAGACCTACGTCGAGCAGCACGAGGTGGCTGTGGCCAGATAC" +
                "TGCG";


// Create Sequence for CRP_b promoter
CRP_b_seq_elements =  "GCTCCGAATTTCTCGACAGATCTCATGTGATTACGCCAAGCTACG" +
                "GGCGGAGTACTGTCCTCCGAGCGGAGTACTGTCCTCCGAGCGGAG" +
                "TACTGTCCTCCGAGCGGAGTACTGTCCTCCGAGCGGAGTTCTGTC" +
                "CTCCGAGCGGAGACTCTAGATACCTCATCAGGAACATGTTGGAAT" +
                "TCTAGGCGTGTACGGTGGGAGGCCTATATAAGCAGAGCTCGTTTA" +
                "GTGAACCGTCAGATCGCCTCGAGTACCTCATCAGGAACATGTTGG" +
                "ATCCAATTCGACC";
```

Next, we specify **ComponentDefinitions** for all **FunctionalComponents** in the CRPb Characterization Circuit. The code snippet below first creates a **ComponentDefinition** of DNA type for the `CRP_b` promoter (lines 1-7). Note that by using compliant URIs, the sequence can be looked up using its *displayId*, i.e. `CRP_b_seq` that was added previously, and since no version is provided, it is referenced by its *persistentIdentity* URI (line 7). It is simply its URI without the version when using compliant URIs. The purpose of a persistent identity is to allow an object to refer to the latest version of another object using this URI. The latest version of an object is determined using *semantic versioning* conventions (c.f., http://semver.org/). Then, we create two **ComponentDefinition** objects, one for the EYFP *coding sequence* (CDS), and one for the EYFP gene (lines 8-17). We use a **SequenceConstraint** object (lines 18-23)

Table 1: **ComponentDefinition** objects

component definition	type	role	sequence	sequence constraint
pConst	DNA	PROMOTER	n/a	n/a
cas9m_BFP_cds	DNA	CDS	n/a	n/a
cas9m_BFP_gene	DNA	PROMOTER	n/a	cas9m_BFP_gene_constraint
cas9m_BFP	PROTEIN	n/a	n/a	n/a
CRa_U6	DNA	PROMOTER	CRa_U6_seq	n/a
gRNA_b_nc	DNA	CDS	gRNA_b_seq	n/a
gRNA_b_terminator	DNA	TERMINATOR	n/a	n/a
gRNA_b_gene	DNA	PROMOTER	n/a	gRNA_b_gene_constraint1 gRNA_b_gene_constraint2
gRNA_b	RNA	SGRNA	n/a	n/a
cas9m_BFP_gRNA_b	COMPLEX	n/a	n/a	n/a
mKate_cds	DNA	CDS	mKate_seq	n/a
mKate_gene	DNA	PROMOTER	n/a	mKate_gene_constraint
mKate	PROTEIN	n/a	n/a	n/a
Gal4VP16_cds	DNA	CDS	n/a	n/a
Gal4VP16_gene	DNA	PROMOTER	n/a	GAL4VP16_gene_constraint
Gal4VP16	PROTEIN	n/a	n/a	n/a
CRP_b	DNA	PROMOTER	CRP_b_seq	n/a
EYFP_cds	DNA	CDS	n/a	n/a
EYFP_gene	DNA	PROMOTER	n/a	EYFP_gene_constraint
EYFP	PROTEIN	n/a	n/a	n/a

Table 2: **SequenceConstraint** objects

displayId	restriction type	subject	object
cas9m_BFP_gene_constraint	PRECEDES	pConst	cas9m_BFP_cds
gRNA_b_gene_constraint1	PRECEDES	CRa_U6	gRNA_b_nc
gRNA_b_gene_constraint2	PRECEDES	gRNA_b_nc	gRNA_b_terminator
mKate_gene_constraint	PRECEDES	pConst	mKate_cds
GAL4VP16_gene_constraint	PRECEDES	pConst	Gal4VP16_cds
EYFP_gene_constraint	PRECEDES	CRP_b	EYFP_cds

to indicate that the `CRP_b` promoter precedes the `EYFP_cds`, because the sequence for the CDS has not been provided and thus cannot be given an exact **Range**.

Other **ComponentDefinition** objects can be created using the same set of method calls. As an exercise, the reader is encouraged to specify them according to Table 1 and 2. You can use the same `version` for all of the **ComponentDefinition** objects. Entries "type" column in the table are defined as constants in the **ComonentDefinition** class. Roles are SO terms defined as URI constants in the **SequenceOntology** class. URIs for these terms are described in Table 3 of the Specification (Data Model 2.0) document.

```
1
2 CRP_b = ComponentDefinition(ns, "CRP_b", v, BIOPAX_DNA)
3 EYFP_cds = ComponentDefinition(ns, "EYFP_cds", v, BIOPAX_DNA)
4 EYFP_gene = ComponentDefinition(ns, "EYFP_gene", v, BIOPAX_DNA)
5 EYFP_gene_constraint  = ComponentDefinition(ns, "EYFP_gene_constraint ", v,
    BIOPAX_DNA)
```

We are now ready to create all remaining **FunctionalComponent** objects. We first create a **ModuleDefinition** object for the CRPb Characterization Circuit as shown below. Then we create a **FunctionalComponent** `EYFP` that has a **PRIVATE** access type and refers to the `EYFP` **ComponentDefinition** object. The private access type means that this **FunctionalComponent** must not be referred to by remote **MapsTo** objects. Its direction property is `NONE` , indicating that it serves neither input or output with regards to the `CPRb_circuit`. Other **FunctionalComponent** objects, namely `cas9m_BFP`, `cas9m_BFP_gene`, `gRNA_b`, `gRNA_b_gene`, `mKate`, `mKate_gene`, `Gal4VP16`,

Table 3: **Interaction** objects

interaction	type	participant	role
mKate_production	GENETIC_PRODUCTION	mKate_gene mKate	PROMOTER PRODUCT
Gal4VP16_production	GENETIC_PRODUCTION	Gal4VP16_gene Gal4VP16	PROMOTER PRODUCT
cas9m_BFP_production	GENETIC_PRODUCTION	cas9m_BFP_gene cas9m_BFP	PROMOTER PRODUCT
gRNA_b_production	GENETIC_PRODUCTION	gRNA_b_gene gRNA_b	PROMOTER PRODUCT
EYFP_Activation	STIMULATION	EYFP_gene Gal4VP16	PROMOTER STIMULATOR
mKate_deg	DEGRADATION	mKate	REACTANT
GAL4VP16_deg	DEGRADATION	GAL4VP16	REACTANT
cas9m_BFP_deg	DEGRADATION	cas9m_BFP	REACTANT
gRNA_b_BFP_deg	DEGRADATION	gRNA_b_BFP	REACTANT
EYFP_BFP_deg	DEGRADATION	EYFP_BFP	REACTANT
cas9m_BFP_gRNA_b_BFP_deg	DEGRADATION	cas9m_BFP_gRNA_b_BFP	REACTANT

`Gal4VP16_gene`, `EYFP_gene`, and `cas9m_BFP_gRNA_b`, can be created similarly with the same access type, version, and direction type.

```
1
2  CRPb_circuit = ModuleDefinition(ns, "CRPb_circuit", v)
3
4  cas9 = FunctionalComponent(ns, "cas9", v, "sys-bio.org/ComponentDefinition/Cas9
       /1.0.0",
5                             SBOL_ACCESS_PUBLIC, SBOL_DIRECTION_IN)
6  guide_rna = FunctionalComponent(ns, "guide_rna", v, GuideRNA.identity.get(),
7                             SBOL_ACCESS_PUBLIC, SBOL_DIRECTION_IN)
8  target_protein = FunctionalComponent(ns, "target_protein", v, TargetProtein.
       identity.get(),
9                             SBOL_ACCESS_PUBLIC, SBOL_DIRECTION_OUT)
10 cas9_mbfp = FunctionalComponent(ns, "cas9_mbfp", v, Cas9mBFP.identity.get(),
11                            SBOL_ACCESS_PUBLIC, SBOL_DIRECTION_OUT);
12 guide_rna_b = FunctionalComponent(ns, "guide_rna_b", v, GuideRNAb.identity.get(),
13                            SBOL_ACCESS_PUBLIC, SBOL_DIRECTION_OUT);
14 eyfp = FunctionalComponent(ns, "eyfp", v, EYFP.identity.get(),
15                            SBOL_ACCESS_PUBLIC, SBOL_DIRECTION_OUT);
```

Next, we need to specify all interactions for the CRPb Characterization Circuit. Following the same procedure for creating **Interactions** before, we can create those specified in Table 3.

Now, the CRISPR-based Repression Template **Module** is instantiated and connected to the CRPb Characterization Circuit using **MapsTo** objects. The code below shows these mappings. For example, a **MapsTo** object is used to indicate that the `target_gene` in the template should be refined to be the `EYFP_gene` specified in the CRPb circuit (lines 30-35).

```
1
2  crispr_template = Module(ns, "crispr_template", v, "sys-bio.org/ModuleDefinition/
       CRISPRTemplate/1.0.0")
3  CRISPR_Template.modules.add(crispr_template);
4
5  cas9_input = MapsTo(ns, "cas9_input", v, cas9.identity.get(), cas9_mbfp.identity.
       get(),
6                      SBOL_REFINEMENT_USE_REMOTE)
7  guide_rna_input = MapsTo(ns, "guide_rna_input", v, guide_rna.identity.get(),
       guide_rna_b.identity.get(),
8                           SBOL_REFINEMENT_USE_REMOTE)
9  eyfp_output = MapsTo(ns, "eyfp_output", v, target_protein.identity.get(), eyfp.
       identity.get(),
10                      SBOL_REFINEMENT_USE_REMOTE)
```

At this point, we have completed the CRISPR circuit model.

References

[1] S. Kiani, J. Beal, M. Ebrahimkhani, J. Huh, R. Hall, Z. Xie, Y. Li, and R. Weiss, "Crispr transcriptional repression devices and layered circuits in mammalian cells," *Nature Methods*, vol. 11, no. 7, pp. 723–726, 2014.

[2] M. Courtot, N. Juty, C. Knüpfer, D. Waltemath, A. Zhukova, A. Dräger, M. Dumontier, A. Finney, M. Golebiewski, J. Hastings, S. Hoops, S. Keating, D. Kell, S. Kerrien, J. Lawson, A. Lister, J. Lu, R. Machne, P. Mendes, M. Pocock, N. Rodriguez, A. Villeger, D. Wilkinson, S. Wimalaratne, C. Laibe, M. Hucka, and N. Le Novère, "Controlled vocabularies and semantics in systems biology," *Molecular Systems Biology*, vol. 7, 2011.