Supplementary material

March 11, 2009

1 The translation to parts

Recall from the main paper that the target semantic objects of the translation we wish to define on programs are pairs (Δ, Θ) of *device templates* $\Delta \subsetneq U^*$, i.e. sets of lists over variables and names, and sets $\Theta \subsetneq CS$ of context-sensitive substitutions. The target semantic objects of *modules* are partial functions f of the form $f(\widetilde{A}) = (\Delta, \Theta)$ mapping actual parameters to the target semantic object of the module. Module definitions are then recorded in environments which are partial finite functions Γ of the form $\Gamma(p) = f$. We denote by $\Gamma\{p' \mapsto f'\}$ the update of Γ with a new binding to p', defined in the standard way.

The substitution of formal parameters $\widetilde{u}$ for actual parameters $\widetilde{A}$ in program P is written $P[\widetilde{A}/\widetilde{u}]$ and is defined inductively on programs along standard lines, except that the evident multiset interpretation is assumed so that nested multisets are flattened to a single multiset following the substitution. For example, the substitution $[\{s2a, s2b\}/s2]$ applied to the complex species $\{s1, s2, s2\}$ results in $\{s1, s2a, s2b, s2a, s2b\}$ rather than $\{s1, \{s2a, s2b\}, \{s2a, s2b\}\}$. In the corresponding concrete syntax, the substitution [`s2a-s2b/s2`] applied to the complex species `s1-s2-s2` is written `s1-s2a-s2b-s2a-s2b`.

We assume a denotational function of the form $[\![K]\!]_\theta = b$ for evaluating numerical constraints K, relative to a given substitution θ, to a boolean value $b \in$ **true**/**false**. The full denotational semantics of GEC is then given by a partial function of the form $[\![P]\!]_\Gamma = (\Delta, \Theta)$, which maps a program P to a set Δ of device templates and a set Θ of substitutions. It is defined inductively on programs as follows.

1. $[\![u : t(Q^t)]\!]_\Gamma \triangleq (\{(u)\}, \Theta)$ where
   $$\begin{aligned}\Theta \triangleq \{(\theta_i, \rho_i, \sigma_i, \mathrm{FS}(Q_i) \setminus \sigma_i) \mid\ & u\theta_i : t(Q_i) \in \mathcal{K}_b, Q^t\theta_i \subseteq Q_i,\\ & \mathrm{Dom}(\theta_i) = \mathrm{FV}(u : t(Q^t)),\\ & \rho_i = \mathrm{Dom}_\mathrm{S}(\theta_i), \sigma_i = \mathrm{FS}(Q^t\theta_i)\}.\end{aligned}$$

2. $[\![\mathbf{0}]\!]_\Gamma \triangleq (\emptyset, (\emptyset, \emptyset, \emptyset, \emptyset))$.

3. $[\![p(\widetilde{u})\ \{P_1\}\ ;\ P_2]\!]_\Gamma \triangleq [\![P_2]\!]_{\Gamma\{p \mapsto f\}}$ where
   $f(\widetilde{A}) \triangleq [\![P_1\{\widetilde{A}/\widetilde{u}\}]\!]_\Gamma$.

4. $[\![p(\widetilde{A})]\!]_\Gamma \triangleq f(\widetilde{A})$ where $f \triangleq \Gamma(p)$.

5. $[\![P \mid C]\!]_\Gamma \triangleq (\Delta, \Theta_1 \oslash \Theta_2)$ where
   $(\Delta, \Theta_1) \triangleq [\![P]\!]_\Gamma$ and $\Theta_2 = [\![C]\!]$.

6. $[\![P_1 \parallel P_2]\!]_\Gamma \triangleq (\Delta_1 \cup \Delta_2, \Theta_1 \oslash \Theta_2)$ where
   $(\Delta_1, \Theta_1) \triangleq [\![P_1]\!]_\Gamma$ and $(\Delta_2, \Theta_2) = [\![P_2]\!]_\Gamma$.

7. $[\![P_1 \,;\, P_2]\!]_\Gamma \triangleq (\{\delta_{1_i}\delta_{2_j}\}_{I\times J}, \Theta_1 \oslash \Theta_2)$ where
   $(\{\delta_{1_i}\}_I, \Theta_1) \triangleq [\![P_1]\!]_\Gamma$ and $(\{\delta_{2_j}\}_J, \Theta_2) = [\![P_2]\!]_\Gamma$.

8. $[\![c[P]]\!]_\Gamma \triangleq (\Delta, \{(\theta, \emptyset, \emptyset, \emptyset) \mid (\theta, \rho, \sigma, \tau) \in \Theta\})$ where
   $(\Delta, \Theta) \triangleq [\![P]\!]_\Gamma$.

9. $[\![\text{new } x.P]\!]_\Gamma \triangleq [\![P[x'/x]]\!]_\Gamma$ for some fresh x'.

10. $[\![R]\!] \triangleq \{(\theta_i, \text{Dom}_\text{S}(\theta_i), \text{FS}(R\theta_i), \emptyset) \mid$
    $R\theta_i \in \mathcal{K}_r, \text{Dom}(\theta_i) = \text{FV}(R)\}$.

11. $[\![T]\!] \triangleq \{(\theta_i, \text{Dom}_\text{S}(\theta_i), \text{FS}(T\theta_i), \emptyset) \mid$
    $T^{\downarrow}\theta_i \in \mathcal{K}_r, \text{Dom}(\theta_i) = \text{FV}(T)\}$.

12. $[\![K]\!] \triangleq \{(\theta_i, \text{Dom}_\text{S}(\theta_i), \text{FS}(T\theta_i), \emptyset) \mid$
    $[\![K]\!]_{\theta_i} = \textbf{true}, \text{Dom}(\theta_i) = \text{FV}(K)\}$.

13. $[\![C_1 \mid C_2]\!] \triangleq (\Theta_1 \oslash \Theta_2)$ where
    $\Theta_1 \triangleq [\![C_1]\!]$ and $\Theta_2 = [\![C_2]\!]$.

2 The translation to reactions

The main paper outlines how specific part types can be translated to reactions, and how the rates of these reactions can be determined by examining the associated rate information in the parts database. While this process is simple in principle, it is complicated by modularity and the necessity to give a compositional definition. We illustrate this with the following small example:

```
prom<con(RT)>; rbs<rate(R)>; pcr<codes(p, RD)>
```

Each of the parts is translated to a corresponding set of reactions, as shown in Table 1. The actual species used in the reactions will often depend on the context in which the parts are placed. Thus, the reactions can take some species as *inputs* from neighbouring parts, and produce other species as *outputs* for these parts. As a result, the reactions generated from each part are associated with corresponding input and output species, as shown in the table. The parts are composed from left to right, and the species names in the corresponding reactions are resolved during the composition.

1. Evaluating the promoter gives rise to reactions `g ->{RT} m` and `m ->{rdm}`, where `g` and `m` are unique species names for the gene and the resulting mRNA, respectively. The rate `rdm` used for mRNA degradation is assumed to be defined globally and may be adjusted manually for individual cases if necessary. The mRNA can be used by neighbouring parts in the sequence.

Table 1: Translation from part sequences to reactions for a simple GEC program. The table also shows the inputs and outputs associated to the reactions. A globally defined mRNA degradation rate `rdm` is assumed.

#	Part Sequence	Input	Output	Reactions

1	`prom<con(RT)>`		`m`	`g ->{RT} g + m`
				`m ->{rdm}`

2	`rbs<rate(R)>`	`m', p'`		`m' ->{R} m' + p'`
3	`pcr<codes(p, RD)>`		`p`	`p ->{RD}`

4	`prom<con(RT)>;`
	`rbs<rate(R)>`
		`p'`		`g ->{RT} g + m`
				`m ->{R} m + p'`

5	`prom<con(RT)>;`
	`rbs<rate(R)>;`
	`pcr<codes(p, RD)>`
				`g ->{RT} g + m`
				`m ->{R} m + p`
				`p ->{RD}`

2. Evaluating the ribosome binding site gives rise to a reaction of the form `m' ->{RT} p'` where `m'` and `p'` are species names for the mRNA and the resulting protein, respectively. However, since neither `m'` nor `p'` are known until the ribosome binding site is placed between two parts, the reaction is effectively a *function* of `m'` and `p'`. The mRNA `m'` is obtained from a preceding promoter region, while the protein `p'` is obtained from a subsequent protein coding region.

3. Evaluating the protein coding region immediately gives rise to a degradation reaction. The evaluation also provides the species name `p` for the protein, which can be used by neighbouring parts.

4. The left-most sequential composition can now be evaluated by using the mRNA `m` obtained from the promoter as an input to the ribosome binding site. The composition gives rise to two reactions for the gene and mRNA, respectively. However, the protein `p'` produced by the gene is still not known, and constitutes an input to the reactions.

5. The right-most sequential composition can now be evaluated by using the protein `p` obtained from the protein coding region as an input to the sequence of promoter and ribosome binding site. This gives rise to three reactions for the gene, mRNA and protein, respectively. Now all three species names have been resolved by the composition of the parts and their corresponding reactions.

The translation is complicated further in the presence of compartments and transport, since degradation reactions may need to be placed in multiple compartments that do not necessarily express the given protein. To address this, the translation function returns separately an LBS program and a set of deduced degradation reactions. After translation, these degradation reactions can be placed in the relevant compartments and composed with the LBS program.

Simulation of GEC models is achieved through a translation to models in a subset of LBS. For our purpose, we define an LBS model L to consist of reactions and transport reactions in parallel, taking place inside some hierarchy of compartments. Formally:

$$L ::= R \;\vdots\; T \;\vdots\; \mathbf{0} \;\vdots\; L_1|L_2 \;\vdots\; c[L]$$

Given a set $\mathcal{L}$ of LBS models, we let (par $\mathcal{L}$) denote their parallel composition; this operator is commutative, so the order is insignificant. With the above motivation in mind, the translation function takes the following form:

$$[\![P]\!]_\Gamma = (L, D, M, Pr, F, G, H)$$

where

- L is an LBS program.
- D is a set of degradation reactions.
- $M \subset U$ is a set of mRNA names.
- $Pr \subset U$ is a set of protein names.
- F is a function of the form $f(m,p) = R$ mapping pairs $(m,p) \in U \times U$ of mRNA and protein names to a reaction.
- G is a function of the form $g(m) = R$ mapping an mRNA species name $m \in U$ to a reaction.
- H is a function of the form $h(p) = R$ mapping a protein name $p \in U$ to a reaction.

The translation is defined inductively on GEC programs as follows, where we again assume a global mRNA degradation rate rdm.

1. $[\![u : \text{prom}(Q)]\!]_\Gamma \triangleq (\text{par}\{\text{reacs}(q) \mid q \in Q\} \mid m \rightarrow^{rdm}, \{m\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ where

   - $\text{reacs}(\text{con}(rt)) \triangleq g \rightarrow^{rt} g + m$.
   - $\text{reacs}(\text{pos}(S, rb, rub, rtb)) \triangleq g + S \rightarrow^{rb} g\text{-}S \mid g\text{-}S \rightarrow^{rub} g + S \mid g\text{-}S \rightarrow^{rtb} g\text{-}S + m$.
   - $\text{reacs}(\text{neg}(S, rb, rub, rtb)) \triangleq g + S \rightarrow^{rb} g\text{-}S \mid g\text{-}S \rightarrow^{rub} g + S \mid g\text{-}S \rightarrow^{rtb} g\text{-}S + m$.

   with g and m fresh.

2. $[\![u : \text{rbs}(\{\text{rate}(r)\})]\!]_\Gamma \triangleq (\mathbf{0}, \emptyset, \emptyset, \emptyset, \{f\}, \emptyset, \emptyset)$ where $f(m,p) \triangleq m \rightarrow^{r} p$.

3. $[\![u : \text{pcr}(\{\text{codes}(p, r)\})]\!]_\Gamma \triangleq (\mathbf{0}, \{p \rightarrow^{r}\}, \emptyset, \{p\}, \emptyset, \emptyset, \emptyset)$.

4. $[\![u : \text{ter}]\!]_\Gamma \triangleq (\mathbf{0}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$.

5. $[\![\mathbf{0}]\!]_\Gamma \triangleq (\mathbf{0}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$.

6. $[\![p(\widetilde{u})\ \{P_1\}\ ;\ P_2]\!]_\Gamma \triangleq [\![P_2]\!]_{\Gamma\{p\mapsto f\}}$ where
$f(\widetilde{A}) \triangleq [\![P_1\{\widetilde{A}/\widetilde{u}\}]\!]$.

7. $[\![p(\widetilde{A})]\!]_\Gamma \triangleq f(\widetilde{A})$ where $f \triangleq \Gamma(p)$.

8. $[\![P \mid C]\!]_\Gamma \triangleq (L_1 \mid L_2, D, M, Pr, F, G, H)$ where
$(L_1, D, M, Pr, F, G, H) \triangleq [\![P]\!]_\Gamma$ and
$L_2 \triangleq [\![C]\!]$.

9. $[\![P_1 \parallel P_2]\!]_\Gamma \triangleq (L_1 \mid L_2, D_1 \cup D_2, M_1 \cup M_2, Pr_1 \cup Pr_2, F_1 \cup F_2, G_1 \cup G_2, H_1 \cup H_2)$ where
$(L_1, D_1, M_1, Pr_1, F_1, G_1, H_1) \triangleq [\![P_1]\!]_\Gamma$ and
$(L_2, D_2, M_2, Pr_2, F_2, G_2, H_2) \triangleq [\![P_2]\!]_\Gamma$.

10. $[\![P_1\ ;\ P_2]\!]_\Gamma \triangleq (L_1 \mid L_2 \mid L, D_1 \cup D_2, M, Pr, F_1' \cup F_2', G, H)$ where
$(L_1, D_1, M_1, Pr_1, F_1, G_1, H_1) \triangleq [\![P_1]\!]_\Gamma$,
$(L_2, D_2, M_2, Pr_2, F_2, G_2, H_2) \triangleq [\![P_2]\!]_\Gamma$,
$L \triangleq \text{par}\{g(m) \mid g \in G_2, m \in M_1\} \cup \text{par}\{h(p) \mid h \in H_1, p \in Pr_2\}$,
$$M \triangleq \begin{cases} M_1 & \text{if } M_2 = \emptyset \\ M_2 & \text{otherwise} \end{cases}$$
$$Pr \triangleq \begin{cases} Pr_2 & \text{if } Pr_1 = \emptyset \\ Pr_1 & \text{otherwise} \end{cases}$$
$$(F_1', H_1') \triangleq \begin{cases} (F_1, H_1) & \text{if } Pr_2 = \emptyset \\ (\emptyset, \emptyset) & \text{otherwise} \end{cases}$$
$$(F_2', G_2') \triangleq \begin{cases} (F_2, G_2) & \text{if } M_1 = \emptyset \\ (\emptyset, \emptyset) & \text{otherwise} \end{cases}$$
$G \triangleq \{g \mid g(m) \triangleq f(m,p), f \in F_1, p \in Pr_2\} \cup G_1 \cup G_2'$,
$H \triangleq \{h \mid h(p) \triangleq f(m,p), f \in F_2, m \in M_1\} \cup H_2 \cup H_1'$.

11. $[\![c[P]]\!]_\Gamma \triangleq (c[L], D, M, Pr, F, G, H)$ where
$(L, D, M, Pr, F, G, H) \triangleq [\![P_1]\!]_\Gamma$.

12. $[\![\text{new } x.P]\!]_\Gamma \triangleq [\![P[x'/x]]\!]_\Gamma$ for some fresh x'.

13. $[\![R]\!] \triangleq R$.

14. $[\![T]\!] \triangleq T$.

15. $[\![K]\!] \triangleq \mathbf{0}$.

16. $[\![C_1 \mid C_2]\!] \triangleq [\![C_1]\!] \mid [\![C_2]\!]$.

Note that the resulting LBS program may generally contain variables, and that each substitution arising from the translation to parts can be applied to the LBS program in order to obtain an LBS program for each device.

3 The compilation process by example

This section illustrates the compilation process for the repressilator and predator-prey models through examples.

3.1 The repressilator

Compilation of a GEC program yields three structures: A device template, an LBS program, and a set of substitutions. We consider each in turn for the basic repressilator program shown in the main paper.

```
module tl(o) { rbs; pcr<prot(o)>; ter      };
module gateNeg(i, o) { prom<neg(i)>; tl(o) };
gateNeg(C, A); gateNeg(A, B); gateNeg(B, C)
```

The device template is computed as defined by the translation to parts. Conceptually, however, one can think of the device template as being computed by in-lining module invocations, introducing fresh variables where part identifiers have been omitted, and disregarding everything but the part identifiers/variables and their sequential/parallel compositions. The in-lining and introduction of variables yields the following "intermediate" program for the repressilator:

```
_X75:prom<neg(C)>; _X81:rbs; _X86:pcr<codes(A)>; _X90:ter;
_X101:prom<neg(A)>; _X107:rbs; _X112:pcr<codes(B)>; _X116:ter;
_X49:prom<neg(B)>; _X55:rbs; _X60:pcr<codes(C)>; _X64:ter
```

Disregarding everything but the part variables then results in the following device template, consisting of a single list with 12 parts (four for each gate):

```
[_X101,_X107,_X112,_X116,_X49,_X55,_X60,_X64,_X75,_X81,_X86,_X90]
```

If the GEC program had included parallel compositions as in the predator-prey example, the device template would essentially consists of a list for each parallel program.

The second structure resulting from compilation is an LBS program representing a general set of reactions. It is computed from the translation defined formally in the previous section:

```
initpop g51 1 |
mrna52 ->{0.001}    |
g51 ->{_X47} g51 + mrna52  |
g51 + C ->{_X44} g51-C  |
g51-C ->{_X45} g51 + C  |
g51-C ->{_X46} g51-C + mrna52  |
mrna52 ->{_X54} mrna52 + A  |

initpop g77 1 |
mrna78 ->{0.001}    |
g77 ->{_X73} g77 + mrna78  |
g77 + A ->{_X70} g77-A  |
g77-A ->{_X71} g77 + A  |
g77-A ->{_X72} g77-A + mrna78  |
mrna78 ->{_X80} mrna78 + B  |

initpop g103 1 |
mrna104 ->{0.001}   |
g103 ->{_X99} g103 + mrna104  |
g103 + B ->{_X96} g103-B  |
g103-B ->{_X97} g103 + B  |
g103-B ->{_X98} g103-B + mrna104  |
mrna104 ->{_X106} mrna104 + C      |

A ->{_X57}   |
B ->{_X83}   |
C ->{_X109}
```

The `initpop` expression featuring in the LBS program introduces initial populations of 1 for the three genes (this aspect is omitted from the formal definition of the translation to reactions). Recall that e.g. the property `neg(A)` is a derived form and is here replaced automatically by `neg(A, _X70, _X71, _X72)` before computing reactions; hence the use of these variables in the LBS program. Note also that reactions for constitutive expression of genes are included in the LBS program even though the `con` property of promoters is omitted in the GEC program; `con` properties are automatically inserted by the compiler if absent.

The third and final structure resulting from the compilation is a set of substitutions that can be applied to the device template and the LBS program to obtain a final set of devices. In practice, these substitutions are obtained by generating and solving a Prolog goal according to the principles set forth in the formal definition of the translation to parts. The generated Prolog goal for the repressilator example is shown in the following.

```
_X1=[],
prom(_X49,con([_X47])),prom(_X49,neg([C],[_X44],[_X45],[_X46])),
exclusiveNames(prom,
_X49,[[C]],_X50),
rbs(_X55,rate([_X54])),exclusiveNames(rbs, _X55,[],_X56),
pcr(_X60,codes([A],[_X57])),exclusiveNames(pcr, _X60,[[A]],_X61),
ter(_X64,_),exclusiveNames(ter, _X64,[],_X65),
noDuplicates([A]),
namesDisjoint([[A]], _X61, [], _X65),union(_X61,_X65,_X66),
namesDisjoint([], _X56, [[A]], _X66),union(_X56,_X66,_X67),
noDuplicates([A,C]),
namesDisjoint([[C]], _X50, [[A]], _X67),union(_X50,_X67,_X68),

prom(_X75,con([_X73])),prom(_X75,neg([A],[_X70],[_X71],[_X72])),
exclusiveNames(prom, _X75,[[A]],_X76),
rbs(_X81,rate([_X80])),
exclusiveNames(rbs, _X81,[],_X82),
pcr(_X86,codes([B],[_X83])),
exclusiveNames(pcr, _X86,[[B]],_X87),ter(_X90,_),
exclusiveNames(ter, _X90,[],_X91),
noDuplicates([B]),
namesDisjoint([[B]], _X87, [], _X91),union(_X87,_X91,_X92),
namesDisjoint([], _X82, [[B]], _X92),union(_X82,_X92,_X93),
noDuplicates([A,B]),
namesDisjoint([[A]], _X76, [[B]], _X93),union(_X76,_X93,_X94),

prom(_X101,con([_X99])),prom(_X101,neg([B],[_X96],[_X97],[_X98])),
exclusiveNames(prom, _X101,[[B]],_X102),
rbs(_X107,rate([_X106])),exclusiveNames(rbs, _X107,[],_X108),
pcr(_X112,codes([C],[_X109])),exclusiveNames(pcr, _X112,[[C]],_X113),
ter(_X116,_),exclusiveNames(ter, _X116,[],_X117),
noDuplicates([C]),
namesDisjoint([[C]], _X113, [], _X117),union(_X113,_X117,_X118),
namesDisjoint([], _X108, [[C]], _X118),union(_X108,_X118,_X119),
noDuplicates([B,C]),
namesDisjoint([[B]], _X102, [[C]], _X119),union(_X102,_X119,_X120),
noDuplicates([A,B,C]),
namesDisjoint([[A],[B]], _X94, [[B],[C]],_X120),union(_X94,_X120,_X122),
namesDisjoint([[A],[C]], _X68, [[A],[B],[C]],_X122),union(_X68,_X122,_X123),
namesDisjoint([], _X1, [[A],[B],[C]], _X123),union(_X1,_X123,_X124).
```

In outline, the Prolog database contains predicates `prom(ID, P)`, `rbs(ID, P)`, `pcr(ID, P)` and `ter(ID, P)` for representing each property `P` of a part with name `ID`. If a part has no properties, as is the case for the single `ter` part, it is represented by a single clause with a distinguished "don't care" property. The `union` predicate is standard, and the remaining three predicates used in the above code are defined informally as follows:

- The `exclusiveNames(T, ID, Names, ExNames)` predicate holds true if `ExNames` is a list of exclusive names for the part identified by the type `T` and name `ID`, where `Names` are the names mentioned explicitly in the properties of the part. This is used for implementing the first case of the semantics defined in the main paper.
- The `namesDisjoint(N1, EN1, N2, EN2)` predicate is used to enforce cross-talk avoidance in parallel and sequential compositions. It holds true if `N1`, representing the names occurring on the left hand side of a composition operator, is disjoint from `EN2`, representing the exclusive names of the program occurring on the right hand side of a composition operator, and visa-verse for `EN1` and `N2`.
- The `noDuplicates(L)` predicate is used to enforce injectivity over species names of the resulting substitutions. It holds true if there are no duplicates in the list `L`.

Executing the Prolog goal yields 24 solutions. One of these is given by the following pairs:

```
(_X101,r0040), (_X107,b0034), (_X112,c0080), (_X116,b0015),
(_X49,i0500),  (_X55,b0034),  (_X60,c0051),  (_X64,b0015),
(_X75,r0051),  (_X81,b0034),  (_X86,c0040),  (_X90,b0015),

(A,clR),(B,tetR),(C,araC),
(_X106,0.1), (_X109,0.001),  (_X44,1),
(_X45,1e-6), (_X46,0.00005),  (_X47,0.1),
(_X54,0.1),  (_X57,0.001),   (_X70,1),
(_X71,0.5),  (_X72,0.00005),  (_X73,0.12),
(_X80,0.1),  (_X83,0.001),   (_X96,1),
(_X97,0.5),  (_X98,0.00005),  (_X99,0.09)
```

Applying this solution to the device template yields a concrete device:

```
[r0040, b0034, c0080, b0015, i0500, b0034, c0051, b0015,
 r0051, b0034, c0040, b0015]
```

and a concrete LBS program:

```
rate RMRNADeg = 0.001;

initpop g51 1 |
mrna52 ->{RMRNADeg}   |
g51 ->{0.1} g51 + mrna52  |
g51 + araC ->{1} g51-araC  |
g51-araC ->{1e-6} g51 + araC  |
g51-araC ->{0.00005} g51-araC + mrna52  |
mrna52 ->{0.1} mrna52 + clR  |

initpop g77 1 |
mrna78 ->{RMRNADeg}   |
g77 ->{0.12} g77 + mrna78  |
g77 + clR ->{1} g77-clR  |
g77-clR ->{0.5} g77 + clR  |
g77-clR ->{0.00005} g77-clR + mrna78  |
mrna78 ->{0.1} mrna78 + tetR  |

initpop g103 1 |
mrna104 ->{RMRNADeg}   |
g103 ->{0.09} g103 + mrna104  |
g103 + tetR ->{1} g103-tetR  |
g103-tetR ->{0.5} g103 + tetR  |
g103-tetR ->{0.00005} g103-tetR + mrna104  |
mrna104 ->{0.1} mrna104 + araC      |

araC ->{0.001}   |
clR ->{0.001}   |
tetR ->{0.001}
```

4 The predator-prey system

As for the repressilator, the compilation of the predator-prey program results in a set of device templates, an LBS program and a set of substitutions obtained by executing a Prolog goal. Here we list only the generated Prolog goal together with a specific instance of the LBS program.

```
_X1=[],
prom(r0051,con([_X44])), exclusiveNames(prom, r0051,[],_X46),
rbs(_X51,rate([_X50])),exclusiveNames(rbs, _X51,[],_X52),
pcr(_X56,codes([Q2b],[_X53])),exclusiveNames(pcr, _X56,[[Q2b]],_X57),
rbs(_X60,rate([_X59])),exclusiveNames(rbs, _X60,[],_X61),
pcr(_X65,codes([Q1a],[_X62])),exclusiveNames(pcr, _X65,[[Q1a]],_X66),
ter(_X69,_),exclusiveNames(ter, _X69,[],_X70),
noDuplicates([Q1a]),
namesDisjoint([[Q1a]], _X66, [], _X70),union(_X66,_X70,_X71),
namesDisjoint([], _X61, [[Q1a]], _X71),union(_X61,_X71,_X72),
noDuplicates([Q1a,Q2b]),
namesDisjoint([[Q2b]], _X57, [[Q1a]], _X72),union(_X57,_X72,_X73),
namesDisjoint([], _X52, [[Q2b],[Q1a]], _X73),union(_X52,_X73,_X74),
namesDisjoint([], _X46, [[Q2b],[Q1a]], _X74),union(_X46,_X74,_X75),
reac([[Q1a]],[],[[H1]], [_X78]),
noDuplicates([]),
namesDisjoint([], _X1, [], _X1),union(_X1,_X1,_X79),
noDuplicates([H1,Q1a]),
namesDisjoint([[Q1a],[H1]], _X1, [], _X79),union(_X1,_X79,_X80),
noDuplicates([H1,Q1a,Q2b]),
namesDisjoint([[Q2b],[Q1a]], _X75, [[Q1a],[H1]], _X80),union(_X75,_X80,_X81),
reac([],[[Q2b],[H2]],[[Q2b,H2]], [_X82]),
reac([],[[Q2b,H2]],[[Q2b],[H2]], [_X83]),
noDuplicates([H2,Q2b]),
namesDisjoint([[Q2b],[H2],[Q2b,H2]], _X1, [[Q2b,H2],[Q2b],[H2]], _X1),
union(_X1,_X1,_X84),
prom(_X90,con([_X88])),prom(_X90,pos([Q2b,H2],[_X85],[_X86],[_X87])),
exclusiveNames(prom, _X90,[[Q2b,H2]],_X91),
rbs(_X96,rate([_X95])),exclusiveNames(rbs, _X96,[],_X97),
pcr(_X101,codes([A],[_X98])),exclusiveNames(pcr, _X101,[[A]],_X102),
ter(_X105,_),exclusiveNames(ter, _X105,[],_X106),
noDuplicates([A]),
namesDisjoint([[A]], _X102, [], _X106),union(_X102,_X106,_X107),
namesDisjoint([], _X97, [[A]], _X107),union(_X97,_X107,_X108),
noDuplicates([A,H2,Q2b]),
namesDisjoint([[Q2b,H2]], _X91, [[A]], _X108),union(_X91,_X108,_X109),
namesDisjoint([[Q2b],[H2],[Q2b,H2]], _X84, [[Q2b,H2],[A]], _X109),
union(_X84,_X109,_X111),
prom(r0051,con([_X112])),exclusiveNames(prom, r0051,[],_X114),
rbs(_X119,rate([_X118])),exclusiveNames(rbs, _X119,[],_X120),
pcr(_X124,codes([ccdB],[_X121])),exclusiveNames(pcr, _X124,[[ccdB]],_X125),
ter(_X128,_),exclusiveNames(ter, _X128,[],_X129),
noDuplicates([ccdB]),
namesDisjoint([[ccdB]], _X125, [], _X129),union(_X125,_X129,_X130),
namesDisjoint([], _X120, [[ccdB]], _X130),union(_X120,_X130,_X131),
namesDisjoint([], _X114, [[ccdB]], _X131),union(_X114,_X131,_X132),
reac([],[[A],[ccdB]],[[A]], [_X134]),
noDuplicates([A,ccdB]),
namesDisjoint([[A],[ccdB],[A]], _X1, [], _X1),union(_X1,_X1,_X135),
namesDisjoint([[ccdB]], _X132, [[A],[ccdB],[A]], _X135),union(_X132,_X135,_X136),
noDuplicates([A,H2,Q2b,ccdB]),
namesDisjoint([[Q2b],[H2],[Q2b,H2],[A]], _X111, [[ccdB],[A]], _X136),
union(_X111,_X136,_X137),
noDuplicates([A,H1,H2,Q1a,Q2b,ccdB]),
namesDisjoint([[Q2b],[Q1a],[H1]], _X81, [[Q2b],[H2],[Q2b,H2],[A],[ccdB]], _X137),
union(_X81,_X137,_X138),_X139=[],
prom(_X145,con([_X143])),prom(_X145,pos([H1,Q1b],[_X140],[_X141],[_X142])),
exclusiveNames(prom, _X145,[[H1,Q1b]],_X146),
rbs(_X151,rate([_X150])),exclusiveNames(rbs, _X151,[],_X152),
pcr(_X156,codes([ccdB],[_X153])),exclusiveNames(pcr, _X156,[[ccdB]],_X157),
ter(_X160,_),exclusiveNames(ter, _X160,[],_X161),
namesDisjoint([[ccdB]], _X157, [], _X161),union(_X157,_X161,_X162),
namesDisjoint([], _X152, [[ccdB]], _X162),union(_X152,_X162,_X163),
noDuplicates([H1,Q1b,ccdB]),
namesDisjoint([[H1,Q1b]], _X146, [[ccdB]], _X163),union(_X146,_X163,_X164),
reac([],[[H1],[Q1b]],[[H1,Q1b]], [_X166]),
reac([],[[H1,Q1b]],[[H1],[Q1b]], [_X167]),
noDuplicates([H1,Q1b]),
namesDisjoint([[H1],[Q1b],[H1,Q1b]], _X1, [[H1,Q1b],[H1],[Q1b]], _X1),
union(_X1,_X1,_X168),
namesDisjoint([[H1,Q1b],[ccdB]], _X164, [[H1],[Q1b],[H1,Q1b]], _X168),
union(_X164,_X168,_X169),
reac([[Q2a]],[],[[H2]], [_X170]),
prom(r0051,con([_X171])),exclusiveNames(prom, r0051,[],_X173),
rbs(_X178,rate([_X177])),exclusiveNames(rbs, _X178,[],_X179),
pcr(_X183,codes([Q2a],[_X180])),exclusiveNames(pcr, _X183,[[Q2a]],_X184),
rbs(_X187,rate([_X186])),exclusiveNames(rbs, _X187,[],_X188),
pcr(_X192,codes([Q1b],[_X189])),exclusiveNames(pcr, _X192,[[Q1b]],_X193),
ter(_X196,_),exclusiveNames(ter, _X196,[],_X197),
noDuplicates([Q1b]),
namesDisjoint([[Q1b]], _X193, [], _X197),union(_X193,_X197,_X198),
namesDisjoint([], _X188, [[Q1b]], _X198),union(_X188,_X198,_X199),
noDuplicates([Q1b,Q2a]),
namesDisjoint([[Q2a]], _X184, [[Q1b]], _X199),union(_X184,_X199,_X200),
namesDisjoint([], _X179, [[Q2a],[Q1b]], _X200),union(_X179,_X200,_X201),
namesDisjoint([], _X173, [[Q2a],[Q1b]], _X201),union(_X173,_X201,_X202),
namesDisjoint([], _X1, [[Q2a],[Q1b]], _X202),union(_X1,_X202,_X205),
namesDisjoint([], _X1, [[Q2a],[Q1b]], _X205),union(_X1,_X205,_X206),
namesDisjoint([], _X1, [[Q2a],[Q1b]], _X206),union(_X1,_X206,_X207),
noDuplicates([H2,Q1b,Q2a]),
namesDisjoint([[Q2a],[H2]], _X1, [[Q2a],[Q1b]], _X207),union(_X1,_X207,_X208),
noDuplicates([H1,H2,Q1b,Q2a,ccdB]),
namesDisjoint([[H1,Q1b],[ccdB],[H1],[Q1b]], _X169, [[Q2a],[H2],[Q1b]], _X208),
union(_X169,_X208,_X209),_X210=[],
namesDisjoint([], _X139, [], _X210),union(_X139,_X210,_X211),
transport(compartment([H1]), [H1],[_X212]),
transport([H1], compartment([H1]),[_X213]),
transport(compartment([H2]), [H2],[_X214]),
transport([H2], compartment([H2]),[_X215]),
union(_X1,_X1,_X216),_X217=[],union(_X1,_X1,_X218),_X219=[],
namesDisjoint([], _X217, [], _X219),union(_X217,_X219,_X220),
noDuplicates([H2]),
namesDisjoint([[H2],[H2]], _X1, [], _X220),union(_X1,_X220,_X221),
namesDisjoint([[H2],[H2]], _X1, [[H2]], _X221),union(_X1,_X221,_X222),
noDuplicates([H1,H2]),
namesDisjoint([[H1],[H1]], _X1, [[H2]], _X222),union(_X1,_X222,_X223),
namesDisjoint([[H1],[H1]], _X1, [[H1],[H2]], _X223),union(_X1,_X223,_X224),
namesDisjoint([], _X211, [[H1],[H2]], _X224),union(_X211,_X224,_X225),
namesDisjoint([], _X1, [[H1],[H2]], _X225),union(_X1,_X225,_X226)])
```

In addition to the predicates used in the Prolog goal generated for the repressilator program, the above goal also uses predicates for reactions and for transport with the evident interpretation. Invoking the goal in Prolog results in four substitutions. The resulting LBS program with one of these substitutions applied is shown below.

```
rate RMRNADeg = 0.001;

c1 [
  initpop g252 1 |
  mrna253 ->{RMRNADeg} |
  g252 ->{0.12} g252 + mrna253 |
  mrna253 ->{0.1} mrna253 + luxR |
  mrna253 ->{0.1} mrna253 + lasI |
  lasI ~  ->{1} m3OC12HSL |
  m3OC12HSL ->{10} |
  m3OC6HSL ->{10} |
  luxR + m3OC6HSL ->{0.5} luxR-m3OC6HSL |
  luxR-m3OC6HSL ->{1} luxR + m3OC6HSL |
  initpop g297 1 |
  mrna298 ->{RMRNADeg} |
  g297 ->{1e-6} g297 + mrna298 |
  g297 + luxR-m3OC6HSL ->{1} g297-luxR-m3OC6HSL |
  g297-luxR-m3OC6HSL ->{0.8} g297 + luxR-m3OC6HSL |
  g297-luxR-m3OC6HSL ->{0.1} g297-luxR-m3OC6HSL + mrna298 |
  mrna298 ->{0.1} mrna298 + ccdA2 |
  initpop g320 1 |
  mrna321 ->{RMRNADeg} |
  g320 ->{0.12} g320 + mrna321 |
  mrna321 ->{0.1} mrna321 + ccdB |
  ccdA2 ~ ccdB ->{0.00001} |
  ccdB + lasI ->{10} ccdB |
  ccdB + luxR ->{10} ccdB
] |

c2 [
  initpop g353 1 |
  mrna354 ->{RMRNADeg} |
  g353 ->{1e-6} g353 + mrna354 |
  g353 + m3OC12HSL-lasR ->{1} g353-m3OC12HSL-lasR |
  g353-m3OC12HSL-lasR ->{0.8} g353 + m3OC12HSL-lasR |
  g353-m3OC12HSL-lasR ->{0.1} g353-m3OC12HSL-lasR + mrna354 |
  mrna354 ->{0.1} mrna354 + ccdB |
  m3OC12HSL + lasR ->{0.5} m3OC12HSL-lasR |
  m3OC12HSL-lasR ->{1} m3OC12HSL + lasR |
  luxI ~  ->{1} m3OC6HSL |
  m3OC6HSL ->{10} |
  m3OC12HSL ->{10} |
  ccdB + luxI ->{10} ccdB |
  ccdB + lasR ->{10} ccdB |
  initpop g380 1 |
  mrna381 ->{RMRNADeg} |
  g380 ->{0.12} g380 + mrna381 |
  mrna381 ->{0.1} mrna381 + luxI |
  mrna381 ->{0.1} mrna381 + lasR
] |

c1[m3OC12HSL] ->{0.5} m3OC12HSL |
m3OC12HSL->{0.5} c2[m3OC12HSL] |
c2[m3OC6HSL] ->{0.5} m3OC6HSL |
m3OC6HSL->{0.5} c1[m3OC6HSL] |

c1 [
  m3OC12HSL ->{1} |
  m3OC6HSL ->{1}
] |

c2 [
  m3OC12HSL ->{1} |
  m3OC6HSL ->{1}
] |

c1 [
  ccdA2 ->{10.0}   |
  ccdB ->{0.005} |
  lasI ->{0.001} |
  lasR ->{0.001} |
  luxI ->{0.001} |
  luxR ->{0.001}
] |

c2 [
  ccdA2 ->{10.0}   |
  ccdB ->{0.005} |
  lasI ->{0.001} |
  lasR ->{0.001} |
  luxI ->{0.001} |
  luxR ->{0.001}
]
```

Table 2: A type system for a subset of GEC.

TBrick	$\dfrac{}{(t : u(Q^t)) : \{(t)\}}$
TNil	$\dfrac{}{\mathbf{0} : \emptyset}$
TCons	$\dfrac{P : \tau}{P \mid C : \tau}$
TPar	$\dfrac{P : \tau \quad P' : \tau'}{P \parallel P' : \tau \cup \tau'}$
TSeq	$\dfrac{P : \{\widetilde{t_i}\}_I \quad P' : \{\widetilde{t'_j}\}_J \quad \{\widetilde{t_i}\widetilde{t'_j}\}_{I \times J} \subsetneq \mathcal{GCL}^{\mathrm{s}}}{P; P' : \{\widetilde{t_i}\widetilde{t'_j}\}_{I \times J}}$
TComp	$\dfrac{P : \tau}{c[P] : \tau}$
TNew	$\dfrac{P : \tau}{\text{new } x(P) : \tau}$

5 Towards a type system for GEC

A type system defines which syntactically well-formed programs are also semantically meaningful. We build on the work of the GenoCad tool which employs a context-free grammar that generates biologically meaningful strings of part identifiers. For the purpose of our type system, we assume a modified GenoCad language, called $\mathcal{GCL}$, with strings over part *types* rather than explicit part *names*. This can be accomplished by pruning the grammar for GenoCad appropriately. We observe however that "incomplete" devices, such as a promoter followed by a ribosome binding site, are not allowed in $\mathcal{GCL}$. But we do want to allow incomplete devices in GEC since these can be used to naturally compose complete GenoCad devices as in the definition of gate modules in the Results section. We therefore let a *program type* τ be a set of substrings $\widetilde{t}$ of strings in $\mathcal{GCL}$, where each $\widetilde{t}$ represents a single list of parts in a device. More formally, we let $\mathcal{GCL}^{\mathrm{s}} \triangleq \{s \mid \exists s' \in \mathcal{GCL} \text{ s.t. } s \text{ is a substring of } s'\}$ and we let $\widetilde{t} \in \tau \subsetneq \mathcal{GCL}^{\mathrm{s}}$. Membership of the substring closure $\mathcal{GCL}^{\mathrm{s}}$ can be decided by e.g. the algorithm presented in [1].

A first step towards a type system for GEC without modules is then shown in Table 2. The addition of modules to the type system requires type environments in order to keep track of the types of defined modules, but this is a standard and easy extension. A more complete type system would additionally ensure that e.g. complex species are not used where a part identifier or rate constant is expected, and that there are no inconsistencies in the use of compartment hierarchies.

6 Proofs

In the following proofs we relax our definition of contexts to be any term with zero or more holes (and not exactly one hole, as in the paper). We say that $\Theta = (\{(\theta_i, \rho_i, \sigma_i, \tau_i)\})$ is N_S-injective iff $\mathrm{Dom}_\mathrm{S}(\theta_i) = \rho_i$; for then $\theta_i \downarrow \mathrm{Dom}_\mathrm{S}(\theta_i)$ is injective per definition of context-sensitive substitutions. Here, and in the proofs to follow, we implicitly assume that indices such as i are universally quantified over an appropriate domain that will be apparent from the context.

Lemma 1. *If Θ_1 and Θ_2 are N_S-injective, then also $\Theta_1 \circledcirc \Theta_2$ is N_S-injective.*

Proof. let $\Theta_1 = \{(\theta_i, \rho_i, \sigma_i, \tau_i)\}$ and $\Theta_2 = \{(\theta'_j, \rho'_j, \sigma'_j, \tau'_j)\}$. Take any $\theta_i \cup \theta'_j$ in $\Theta_1 \circledcirc \Theta_2$. Per assumption, $\mathrm{Dom}_\mathrm{S}(\theta_i) = \rho_i$ and $\mathrm{Dom}_\mathrm{S}(\theta'_j) = \rho'_j$. Hence $\mathrm{Dom}_\mathrm{S}(\theta_i \cup \theta'_j) = \rho_i \cup \rho'_j$, so $\Theta_1 \circledcirc \Theta_2$ is N_S-injective. □

Lemma 2. *Let C be a constraint. Then $[\![C]\!]$ is N_S-injective.*

Proof. By induction on C. For the three base cases (R, T and K), $\rho_i = \mathrm{Dom}_\mathrm{S}(\theta_i)$ per definition. For the inductive steps, we invoke the IH to get that $[\![C_1]\!]$ and $[\![C_2]\!]$ are N_S-injective. It then follows from Lemma 1 that also $[\![C_1]\!] \circledcirc [\![C_2]\!]$ is N_S-injective.

□

Lemma 3. *For any compartment-free program P and environment Γ for which $\Gamma(p)(\widetilde{A})$ is N_S-injective for all $p \in Dom(\Gamma) \cap FP(P)$ and all matching $\widetilde{A}$, also $[\![P]\!]_\Gamma$ is N_S-injective.*

Proof. By induction on P. Selected cases:

- $P = u : t(Q^t)$. Then $\rho = \mathrm{Dom}_\mathrm{S}(\theta)$ per definition.
- $P = \mathbf{0}$. Holds vacuously since $\mathrm{Dom}_\mathrm{S}(\theta) = \rho = \emptyset$.
- $P = p(\widetilde{u})\ \{P_1\}\ ;\ P_2$. Let $f(\widetilde{A}) \triangleq [\![P_1\{\widetilde{A}/\widetilde{u}\}]\!]_\Gamma$. By the IH, $f(\widetilde{A})$ is N_S-injective. Per assumption $\Gamma(p)(\widetilde{A})$ is N_S-injective for all $p \in \mathrm{Dom}(\Gamma) \cap \mathrm{FP}(P)$. Therefore also $\Gamma'(p)(\widetilde{A})$ is N_S-injective for all $p \in \mathrm{Dom}(\Gamma') \cap \mathrm{FP}(P)$ where $\Gamma' \triangleq \Gamma\{p \mapsto f\}$. The IH then applies to P_2 and Γ'.
- $P = p(\widetilde{A})$. Follows from the assumption that $\Gamma(p)(\widetilde{A})$ is N_S-injective.
- $P = P_1 \mid C$. The IH gives that $[\![P_1]\!]_\Gamma$ is N_S-injective and Lemma 2 gives that $[\![C]\!]$ is N_S-injective. It follows from 2 that also $[\![P_1]\!]_\Gamma \circledcirc [\![C]\!]$ is N_S-injective.
- $P = \mathrm{new}\ x.P_1$. The IH applies to $[\![P_1[x'/x]]\!]_\Gamma$.

□

Let P_0 be a program and let $\Theta = (\{(\theta_i, \rho_i, \sigma_i, \tau_i)\})$. We say that Θ is N_S-P_0-injective iff $\theta_i \downarrow (\mathrm{Dom}_\mathrm{S}(\theta_i) \cap \mathrm{FV}(P_0))$ is injective and $\mathrm{FV}(P_0) \subseteq \mathrm{Dom}_\mathrm{S}(\theta_i)$.

Proposition 1 (Piece-wise injectivity)**.** *Let P_0 be any compartment-free program with $FP(P_0) = \emptyset$, let $\mathcal{C}(\cdot)$ be any context and let Γ be any environment. If $\mathcal{C}(\cdot)$ has a hole, or if there is a $p \in Dom(\Gamma) \cap FP(\mathcal{C}(\cdot))$ s.t $\Gamma(p)(\widetilde{A})$ is N_S-P_0-injective for all matching $\widetilde{A}$, then also $[\![\mathcal{C}(P)]\!]_\Gamma$ is N_S-P_0-injective.*

Proof. By induction on $\mathcal{C}(\cdot)$. Selected cases:

- $\mathcal{C}(\cdot) = \cdot$. Then $[\![\mathcal{C}(P)]\!]_\Gamma = [\![P]\!]_\Gamma$ is N_S-injective by Lemma 3, and N_S-injectivity implies N_S-P_0-injectivity.

- $\mathcal{C}(\cdot) = \mathcal{C}_1(\cdot) \parallel \mathcal{C}_2(\cdot)$. If the precondition holds for P then it must hold for at least one side, say for $\mathcal{C}_1(\cdot)$ without loss of generality. Then $[\![\mathcal{C}_1(P_0)]\!]_\Gamma$ is N_S-P_0-injective by the IH. The union of a function injective on some interval with any other function, when defined, is also injective on this interval.

- $\mathcal{C}(\cdot) = p(\widetilde{u})\ \{\mathcal{C}_1(\cdot)\}\ ;\ \mathcal{C}_2(\cdot)$. If the precondition holds for P there are two cases to consider:

  1. The precondition holds for $\mathcal{C}_1(\cdot)$. Then $f(\widetilde{A}) \ \triangleq\ [\![\mathcal{C}_1(P_0)\{\widetilde{A}/\widetilde{u}\}]\!]_\Gamma = [\![\mathcal{C}_1\{\widetilde{A}/\widetilde{u}\}(P_0)]\!]_\Gamma$ per definition and per assumption that context instantiations are capture free, i.e. no free variables of P_0 become bound by formal parameters. By the IH, $f(\widetilde{A})$ N_S-P-injective. Apply the IH to $[\![\mathcal{C}_2(P_0)]\!]_{\Gamma\{p\mapsto f\}}$ which now satisfies the precondition and hence is N_S-P_0-injective.
  2. The precondition holds for $\mathcal{C}_2(\cdot)$. We can then apply the IH directly to $[\![\mathcal{C}_2(P_0)]\!]_\Gamma$.

- $\mathcal{C}(\cdot) = p(\widetilde{A})$. There is no hole in $p(\widetilde{A})$, so if the precondition holds, there is a $p' \in \mathrm{Dom}(\Gamma) \cap \mathrm{FP}(\mathcal{C}(\cdot))$ s.t $\Gamma(p')(\widetilde{A})$ is N_S-P_0-injective for all matching $\widetilde{A}$. Since $\mathrm{FP}(\mathcal{C}(\cdot)) = \{p\}$, we must have that $p' = p$.

- $\mathcal{C}(\cdot) = c[\mathcal{C}'(\cdot)]$. If the precondition holds for $\mathcal{C}(\cdot)$ then it must also hold for $\mathcal{C}'(\cdot)$. Per definition $[\![c[\mathcal{C}'(P_0)]]\!]_\Gamma$ has the same substitutions as $[\![\mathcal{C}'(P_0)]\!]_\Gamma$, and the latter is N_S-P_0-injective by the IH.

- $\mathcal{C}(\cdot) = \text{new } x.\mathcal{C}'(\cdot)$. Again we rely on context instantiations being capture-free so that $x \notin \mathrm{FV}(P_0)$. Hence $\mathcal{C}'(P_0)[x'/x] = \mathcal{C}'[x'/x](P_0)$ and the IH applies.

□

Proposition 1 in the paper is an immediate corollary of Proposition 1 above.

Let $P_0 = u : t(Q^t)$ be a basic program. We say that a program P is P_0-sound in environment Γ iff for $[\![P]\!]_\Gamma = \{(\theta_i, \rho_i, \sigma_i, \tau_i)\}$ it holds that $u\theta_i : t(Q) \in \mathcal{K}_b$ for some Q and $\mathrm{FS}(Q) \setminus \mathrm{FS}(Q^t\theta_i) \subseteq \tau_i$.

Lemma 4. *Let $P_0 = u : t(Q^t)$ be a basic program and let Θ, Θ' be two substitutions with Θ P_0-sound. Then $\Theta \circledcirc \Theta'$ is also P_0-sound.*

Proof. Let $\Theta = \{(\theta_i, \rho_i, \sigma_i, \tau_i)\}$, $\Theta' = \{(\theta'_j, \rho'_j, \sigma'_j, \tau'_j)\}$ and take any $(\theta_i \cup \theta'_j, \rho_i \cup \rho'_j, \sigma_i \cup \sigma'_j, \tau_i \cup \tau'_j) \in \Theta \circledcirc \Theta'$. Per assumption that Θ is P_0-sound, there is a Q s.t. $u\theta_i : t(Q) \in \mathcal{K}_b$, $Q^t\theta_i \subseteq Q$ and $\mathrm{FS}(Q) \setminus \mathrm{FS}(Q^t\theta_i) \subseteq \tau_i$. Note that $\mathrm{FV}(Q^t) \subseteq \mathrm{Dom}(\theta_i)$ and $\mathrm{FV}(u) \subseteq \mathrm{Dom}(\theta_i)$ because $\mathcal{K}_b$ consists of ground terms, so $Q^t\theta_i = Q^t(\theta_i \cup \theta'_j)$ and $u\theta_i = u(\theta_i \cup \theta'_j)$. Therefore also $u(\theta_i \cup \theta'_j) : t(Q) \in \mathcal{K}_b$, $Q^t(\theta_i \cup \theta'_j) \subseteq Q$ and $\mathrm{FS}(Q) \setminus \mathrm{FS}(Q^t(\theta_i \cup \theta'_j)) \subseteq \tau_i \subseteq (\tau_i \cup \tau'_j)$.

□

Proposition 2 (Non-interference)**.** *Let $P_0 = u : t(Q^t)$ be a basic program, let $\mathcal{C}(\cdot)$ be a compartment-free context and let Γ be an environment. If $\mathcal{C}(\cdot)$ has a hole, or if there is a $p \in Dom(\Gamma) \cap FP(\mathcal{C}(\cdot))$ s.t $\Gamma(p)(\widetilde{A})$ is P_0-sound for all matching $\widetilde{A}$, then also $[\![\mathcal{C}(P_0)]\!]_\Gamma$ is P_0-sound.*

Proof. By induction on $\mathcal{C}(\cdot)$. Selected cases (the cases for module definition, module invocation and new variables are shown as in the proof of Proposition 1):

- $\mathcal{C}(\cdot) = \cdot$. Then $[\![\mathcal{C}(P_0)]\!]_\Gamma = [\![u : t(Q^t)]\!]_\Gamma$ and the result follows directly from the definition of the denotation function.

- $\mathcal{C}(\cdot) = \mathcal{C}_1(\cdot) \parallel \mathcal{C}_2(\cdot)$. If the precondition holds for P then it must hold for for either side, say for $\mathcal{C}_1(\cdot)$ without loss of generality. Then $[\![\mathcal{C}_1(P_0)]\!]_\Gamma$ is P_0-sound by the IH. By Lemma 4 also $[\![\mathcal{C}_1(P_0)]\!]_\Gamma \otimes [\![\mathcal{C}_2(P_0)]\!]_\Gamma = [\![P]\!]_\Gamma$ is P_0-sound.

$\square$

Proposition 2 in the paper is an immediate corollary of Proposition 2 above.

References

[1] J. Rekers and W. Koorn. Substring parsing for arbitrary context-free grammars. *SIGPLAN Not.*, 26(5):59–66, 1991.