A Language for Biochemical Systems: Design and Formal Specification

Michael Pedersen and Gordon D. Plotkin

LFCS, School of Informatics, University of Edinburgh

Abstract. This paper introduces a *Language for Biochemical Systems* (LBS) which combines rule-based approaches to modelling with modularity. It is based on the *Calculus of Biochemical Systems* (CBS) which affords modular descriptions of metabolic, signalling and regulatory networks in terms of reactions between modified complexes, occurring concurrently inside a hierarchy of compartments and with possible cross-compartment interactions and transport. Additional features of LBS, targeted towards practical and large-scale applications, include species expressions for manipulating large complexes in a concise manner, parameterised modules with a notion of subtyping for writing reusable modules, and nondeterminism for handling combinatorial explosion. These features are demonstrated through examples. A formal specification of LBS is then given through an abstract syntax and a general semantics which is parametric on a structure pertaining to the specific choice of target semantical objects. Examples of such structures for the specific cases of Petri nets, coloured Petri nets, ODEs and continuous time Markov chains are also given.

Keywords: Large-scale, parametrised modules, subtyping, combinatorial explosion, nondeterminism, Petri nets, coloured Petri nets, ordinary differential equations, continuous time Markov chains.

1 Introduction

Systems biology. Systems biology is a rapidly growing field which seeks a refined and quantitative understanding of organisms, particularly studying how molecular species such as metabolites, proteins and genes interact in cells to form the complex emerging behaviour that living systems exhibit. Such an understanding is for example important for the development of new drugs and to predict the impact of these on an organism. Mathematical modelling plays a key rôle in pursuit of this by facilitating the generation of new knowledge through the cycle of simulation, experimental validation, and model refinement.

Formalisms for systems biology. As our biological knowledge-base increases through rapid improvements of e.g. high-throughput sequencing methods, the models under study also increase in size and complexity. New methods are therefore needed to support the structured development of large models, and also to

C. Priami et al. (Eds.): Trans. on Comput. Syst. Biol. XII, LNBI 5945, pp. 77–145, 2010.

complement simulations with other kinds of analysis. Hence an abundance of formalisms inspired by computer science have been applied to biological modelling over the past decade. These include Petri nets [23] and coloured Petri nets [18]; process calculi such as the stochastic π-calculus [32, 29], the continuous π-calculus [20], Beta binders [33,14], BlenX [12], PEPA [4] and BioPEPA [8]; rule-based languages such as κ [11,10], BioNetGen [13] and BIOCHAM [6]; state-based formalisms such as Statecharts [15]; and more specialised languages such as Bioambients [34], the Brane calculi [5] and P-systems [24] aimed specifically at describing biological compartments and membranes.

Contributions. Some of the above mentioned languages, in particular those from the process calculus family, excel in their support for modularity by allowing large systems to be described in terms of their components. These languages may be difficult for non-specialists, including some biologists, to use and to understand. Other languages, for example from the rule-based family, are more intuitive to use but only allow flat, non-modular descriptions. The contribution of this paper is a *Language for Biochemical Systems* (LBS) which aims at bridging the above gap: it is based on a notion of reaction rules while also being modular.

LBS builds on the *Calculus of Biochemical Systems* (CBS) [31]. Just as CBS, LBS affords modular descriptions of metabolic, signalling and regulatory networks in terms of reactions between modified complexes, occurring concurrently inside a hierarchy of compartments and with possible cross-compartment interactions and transport. It has a compositional semantics, translating programs into semantical objects such as Petri nets, coloured Petri nets, ordinary differential equations (ODEs) and continuous time Markov chains (CTMCs). Petri nets allow a range of established analysis techniques to be used in the biological setting [16], while ODEs and CTMCs enable respectively deterministic and stochastic simulations to be carried out. The compositional semantics of LBS can be exploited in analysis, as previously demonstrated for the case of Petri net flows [25], potentially improving analysis efficiency and enabling the reuse of analysis results.

LBS, unlike CBS, includes support for practical, large-scale applications through three main features, namely: *species expressions*, *parameterised modules* and *nondeterminism*. Species expressions provide a concise way of constructing large complexes and modifying these incrementally, a common scenario in signal transduction pathways. Parameterised modules allow common motifs such as phosphorylation/dephosphorylation cycles or entire MAPK cascades to be modelled once and reused in different contexts. Modules may be parameterised on compartments, rates, and species; species are typed by their component atomic species names and their modification site types; and the typing system includes subtyping and a notion of parametric type. Modules may furthermore output species, thus providing a natural mechanism for connecting related modules. Nondeterminism provides a mechanism for specifying that any one of a given set of species can take place in a reaction, thus going some way towards compact descriptions of combinatorially complex systems.

LBS further includes species and compartment definitions which involve scope and new name generation, and complex species which may span several compartments. LBS also allows arbitrary modification site types for representing e.g. spatial location (real number pairs) or DNA sequences (strings). A suitable choice of modification site type can furthermore capture connectivity in complex species which we anticipate can lead to a translation of LBS to κ and BioNetGen. Finally, LBS allows both mass-action rates and general rate expressions in reactions, and also provides a means of model variation so that a single LBS program may translate to multiple related semantical objects which differ in e.g. their initial conditions.

A compiler from LBS to the *Systems Biology Markup Language* (SBML) [17] has been implemented, allowing existing SBML-compliant tools for simulation and analysis to be exploited; modular compilations to e.g. ODEs are left for future work. The compiler has been validated on a model of the yeast pheromone pathway [19] for which simulation results coincide with published results.

There are two other recently introduced languages which combine the reaction-based approach with modularity, namely Little b [22] and Antimony [35]. Little b is based on Lisp and thus boasts the full power of modularity found in this general-purpose functional programming language. Antimony follows an object-oriented approach rather than the more functional approach of LBS; it also has dedicated features for the composition of genes aimed at applications in synthetic biology. To our knowledge, neither of these languages has a direct counterpart of the LBS species expressions, nondeterminism and subtyping. Furthermore, they do not have a formally defined semantics (barring the standard semantics of Lisp which is not particularly well suited for the biological domain), let alone a compositional one, and are hence not directly amenable to modular analysis.

Paper outline. We start in Section 2 with an informal overview of CBS through basic examples of gene expression and MAPK cascades. In Section 3 we introduce LBS by further examples and demonstrate how specific limitations of CBS can be overcome. We then turn to a formal presentation, starting with an abstract syntax of LBS in Section 4. A general semantics, parameterised on a structure pertaining to specific semantical objects, is given in Section 5. Examples of such structures for translating to Petri nets, coloured Petri nets, ODEs and CTMCs are given in Section 6. Section 7 concludes with a discussion of future directions.

The present version of LBS is a wholesale redesign of a previous version [27], merging patterns and species into one syntactic category with consequent extensive changes to the typing system, and with a number of extensions such as species nondeterminism and model variation.

2 The Calculus of Biochemical Systems

2.1 Located Parallel Reactions

As a first example we consider a basic model of eukaryotic gene expression as illustrated by the informal pictorial diagram in Figure 2.1. A corresponding

Listing 2.1. A CBS program for gene expression.

```
c[
  n[gene + rnap -> gene + rnap + mrna] |
  n[mrna] -> mrna |
  mrna -> prot
]
```

CBS program is shown in Listing 2.1. The program consists of three reactions composed in *parallel* using the operator |, and taking place inside a cytosol *compartment* c. The first reaction models transcription. It is located inside a nested nucleus compartment n and produces mRNA from a gene and an RNA polymerase. The second reaction models transport of mRNA out of the nucleus into the enclosing cytosol compartment, and the third reaction models translation of mRNA into protein. Compartments can thus be used at the level of individual species and at the level of entire programs. In this example we could have omitted the cytosol compartment in which case a default top level compartment would be assumed. Reactants and products can be labelled with stoichiometry, but when omitted, as in this example, a default of 1 is assumed.

As an illustration of the underlying formal semantics, the Petri nets arising from each of the three individual (but located) reactions are shown in Figure 2.2a. Reactions are represented by *transitions* (rectangles), and located species are represented by *places* (circles). Including species location in place names allows for a semantical distinction of e.g. nuclear and cytosolic mRNA.

When considering the reactions together in parallel, the standard interpretation is for the reactions to share and compete for species which have syntactically identical located names. In this example, the first and second reactions hence share the species n[mrna] and the second and third reactions share the species c[mrna]. A Petri net representation of the parallel composition based on this interpretation is shown in Figure 2.2b. This illustrates how located reactions, and

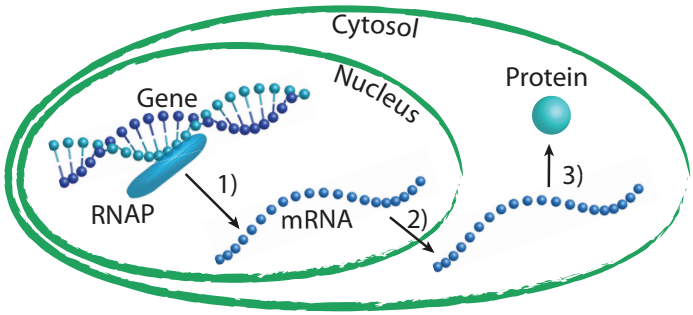
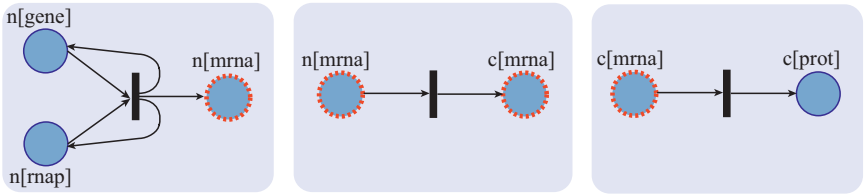
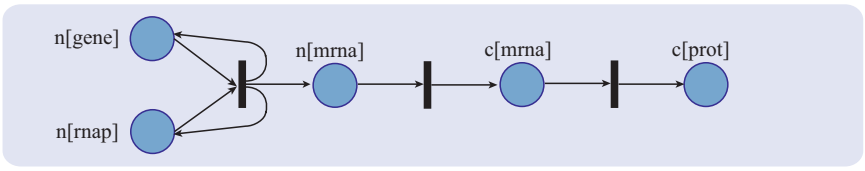


Fig. 2.1. An informal pictorial diagram of eukaryotic gene expression taking place in three steps: 1) transcription, 2) transport and 3) translation



(a) Three Petri nets resulting from individual (but located) reactions. Shared places are highlighted.



(b) The composition obtained by merging the shared places of the three Petri nets.

Fig. 2.2. Petri net models resulting from the CBS program for gene expression

more generally modules, are composed in CBS and hints at how a compositional semantics in terms of Petri nets can be defined; a formal definition is given in Section 6. Semantically we observe that compartments distribute over parallel compositions and reactant/product sums, and that the parallel composition and sum operators are commutative.

2.2 Modification Sites and Complexes

The next example is based on a scaffolded MAPK cascade from the yeast pheromone pathway [19] and features complex species with modification sites. An informal graphical representation is shown in Figure 2.3 and the corresponding CBS program is shown in Listing 2.2.

The first five reactions in lines $1-12$ model the formation of the scaffold complex and correspond to the left part of Figure 2.3. The last three reactions in lines $14-21$ model the actual MAPK cascade, with each reaction phosphorylating a single atomic species in a complex reactant, and correspond to the right part of Figure 2.3. The scaffold is formed by the atomic species Ste5, Ste20 and Gbg, and the species Fus3, Ste7 and Ste11 serve the MAPK, MAPK2 and MAPK3 rôles, respectively. All species except Ste20 and Gbg have a single modification site, p, which can be either phosphorylated or unphosphorylated, indicated by the assignment of boolean values **tt** and **ff**. For example, Fus3{p=**ff**} represents Fus3 in its unphosphorylated state. Complexes are formed by composing modified primitive species using the *complex formation operator*, -.

Semantically, a complex of modified atomic species can be represented by a Petri net place named by the *multiset* of modified atomic species. Hence the complex formation operator is commutative. More generally, modification sites

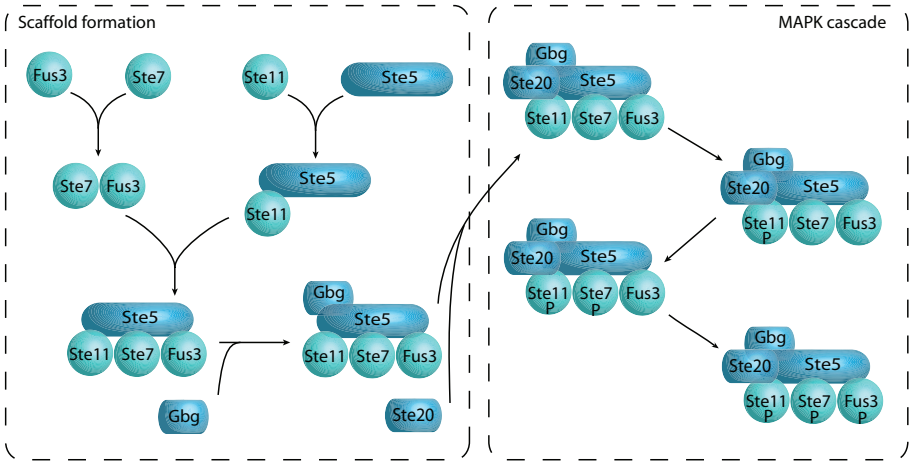


Fig. 2.3. Scaffold formation and a scaffolded MAPK cascade, adapted from [19]

Listing 2.2. A CBS program for a scaffolded MAPK cascade in yeast.

```
1  Ste5{p=ff} + Ste11{p=ff} -> Ste5{p=ff}-Ste11{p=ff} |
2
3  Ste7{p=ff} + Fus3{p=ff} -> Ste7{p=ff}-Fus3{p=ff} |
4
5  Ste5{p=ff}-Ste11{p=ff} + Ste7{p=ff}-Fus3{p=ff} ->
6      Ste5{p=ff}-Ste11{p=ff}-Ste7{p=ff}-Fus3{p=ff} |
7
8  Ste5{p=ff}-Ste11{p=ff}-Ste7{p=ff}-Fus3{p=ff} + Gbg ->
9      Ste5{p=ff}-Ste11{p=ff}-Ste7{p=ff}-Fus3{p=ff}-Gbg |
10
11 Ste5{p=ff}-Ste11{p=ff}-Ste7{p=ff}-Fus3{p=ff}-Gbg + Ste20 ->
12     Ste5{p=ff}-Ste11{p=ff}-Ste7{p=ff}-Fus3{p=ff}-Gbg-Ste20 |
13
14 Ste5{p=ff}-Ste11{p=ff}-Ste7{p=ff}-Fus3{p=ff}-Gbg-Ste20 ->
15     Ste5{p=ff}-Ste11{p=tt}-Ste7{p=ff}-Fus3{p=ff}-Gbg-Ste20 |
16
17 Ste5{p=ff}-Ste11{p=tt}-Ste7{p=ff}-Fus3{p=ff}-Gbg-Ste20 ->
18     Ste5{p=ff}-Ste11{p=tt}-Ste7{p=tt}-Fus3{p=ff}-Gbg-Ste20 |
19
20 Ste5{p=ff}-Ste11{p=tt}-Ste7{p=tt}-Fus3{p=ff}-Gbg-Ste20 ->
21     Ste5{p=ff}-Ste11{p=tt}-Ste7{p=tt}-Fus3{p=tt}-Gbg-Ste20
```

may have arbitrary boolean expressions assigned which may include variables for "matching" multiple physical species, thereby ameliorating the combinatorial explosion problem at the level of species modifications. Semantically, a reaction with such variables can be viewed as generating one concrete reaction for each possible assignment of variables. It is however also possible to handle modification state and variables directly through a semantics in terms of *coloured Petri nets* as is demonstrated in Section 6.

One immediately notices a problem with the CBS program in Listing 2.2: the program is difficult to read due to a high level of *redundancy*. As is common in signalling pathways, some reactions change just a single state of modification in a large complex, yet unaffected parts of the complexes are listed repeatedly. We can improve the situation slightly by introducing an abbreviated notation where an omitted modification site is implicitly assumed to be false, and a modification site with no assignment is assumed to be true, so that for example Ste11{p}−Ste5 is understood as Ste11{p=**tt**}−Ste5{p=**ff**}. We assume this convention in the following. But the underlying problem of redundancy remains and is addressed with dedicated language constructs in LBS.

2.3 Modules

The scaffolded MAPK cascade program is simple in that each reaction represents an autophosphorylation involving only a single reactant. We now shift the focus and consider a larger, unscaffolded cascade model in which the MAPK and MAPK2 proteins each have two phosphorylation sites, and in which each phosphorylation step involves three reactions: binding of kinase and ligand, phosphorylation of bound ligand, and dissociation of phosphorylated ligand and kinase. We furthermore include the corresponding dephosphorylation steps. We choose for this example to adapt a previously published Ras/Raf/MEK/ERK cascade [9]. An informal graphical representation is shown in Figure 2.4, and the corresponding modular CBS program is shown in Listing 2.3.

The new language feature in this program is that of *module definitions* for representing the relevant phosphorylation/dephosphorylation cycles, and the main body of the program in line 47 then consists of five parallel *module invocations*. Such a modular approach simplifies the presentation and should be contrasted with other rule-based approaches using e.g. BIOCHAM where the program would consist of one long, unstructured list of reactions. But as with a previous program, we observe a high degree of redundancy. All five modules have the same structure, consisting of two sets of three reactions for binding, modification and unbinding.

A shorter version of the program could in principle be obtained through appropriate derived forms for enzymatic reactions. But it appears unlikely that a small, fixed set of derived forms can cater for all the variants that a modeller may encounter. We may for example wish to consider variants of the above model in which all the binding reactions are reversible, or in which binding and phosphorylation are combined into a single reaction. Hence we seek to address the

Listing 2.3. A modular CBS program for the Raf/Ras/MEK/ERK MAPK cascade.

```
module rafCycle {
  Ras + Raf -> Ras-Raf |
  Ras-Raf -> Ras-Raf{m} |
  Ras-Raf{m} -> Ras + Raf{m} |
  PP2A1 + Raf{m} -> PP2A1-Raf{m} |
  PP2A1-Raf{m} -> PP2A1-Raf |
  PP2A1-Raf -> PP2A1 + Raf
};

module mekCycle1 {
  Raf{m} + MEK -> Raf{m}-MEK |
  Raf{m}-MEK -> Raf{m}-MEK{S218} |
  Raf{m}-MEK -> Raf{m} + MEK{S218} |
  PP2A2 + MEK{S218} -> PP2A2-MEK{S218} |
  PP2A2-MEK{S218} -> PP2A2-MEK |
  PP2A2-MEK -> PP2A2 + MEK
};

module mekCycle2 {
  Raf{m} + MEK{S218} -> Raf{m}-MEK{S218} |
  Raf{m}-MEK{S218} -> Raf{m}-MEK{S218, S222} |
  Raf{m}-MEK{S218, S222} -> Raf{m} + MEK{S218, S222} |
  PP2A2 + MEK{S218, S222} -> PP2A2-MEK{S218, S222} |
  PP2A2-MEK{S218, S222} -> PP2A2-MEK{S218} |
  PP2A2-MEK{S218} -> PP2A2 + MEK{S218}
};

module erkCycle1 {
  MEK{S218,S222} + ERK -> MEK{S218,S222}-ERK |
  MEK{S218,S222}-ERK -> MEK{S218,S222}-ERK{T185} |
  MEK{S218,S222}-ERK{T185} -> MEK{S218,S222} + ERK{T185} |
  MKP3 + ERK{T185} -> MKP3-ERK{T185} |
  MKP3-ERK{T185} -> MKP3-ERK |
  MKP3-ERK -> MKP3 + ERK
};

module erkCycle2 {
  MEK{S218,S222} + ERK{T185} -> MEK{S218,S222}-ERK{T185} |
  MEK{S218,S222}-ERK{T185}->MEK{S218,S222}-ERK{T185,Y187} |
  MEK{S218,S222}-ERK{T185, Y187} ->
    MEK{S218,S222} + ERK{T185, Y187} |
  MKP3 + ERK{T185,Y187} -> MKP3-ERK{T185,Y187} |
  MKP3-ERK{T185,Y187} -> MKP3-ERK{T185} |
  MKP3-ERK{T185} -> MKP3 + ERK{T185}
};

rafCycle | mekCycle1 | mekCycle2 | erkCycle1 | erkCycle2
```

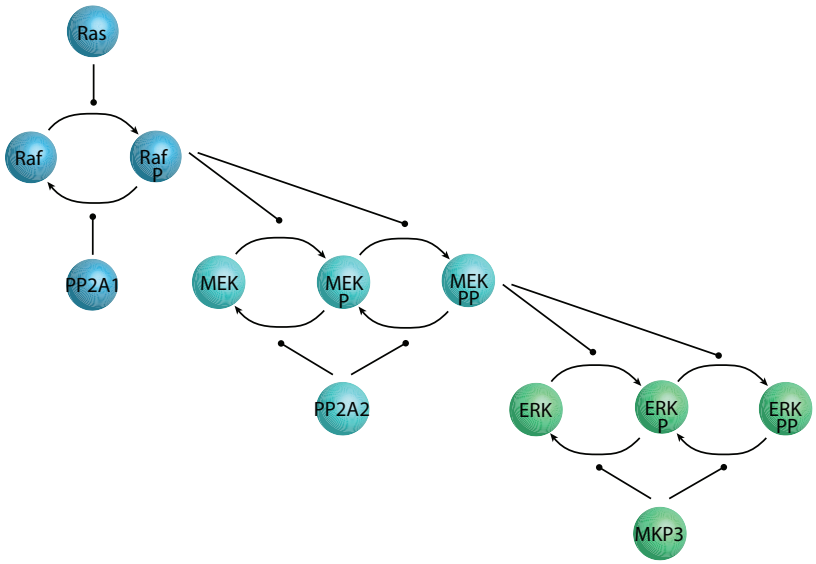


Fig. 2.4. A Ras/Raf/MEK/ERK MAPK cascade represented by five phosphorylation/dephosphorylation cycles. Each phosphorylation and dephosphorylation step covers three underlying reactions for binding, phosphorylation/dephosphorylation, and unbinding.

problem of reusability through language-level support for parameterised modules in LBS.

3 The Language for Biochemical Systems

3.1 New Species and Compartment Definitions

CBS has a static semantics which catches typos by requiring that only species names in a given set are used in programs. In LBS we include both *new species definitions* and *new compartment definitions* directly in the language. New species definitions include a list of modification sites and their type if any, and new compartment definitions include a specification of the parent, if any, and an optional volume. The volume is used when compartments are referred to in rate expressions. The semantics of LBS requires that species are only used with their defined modification types and that compartments are only used inside their defined parents. New species and compartment definitions are demonstrated by the program in Listing 3.1 which is identical to the corresponding CBS program in Listing 2.1 except for the added definitions in the first three lines.

Species *identifiers* such as mrna are here assigned to new species. The text **new**{} is formally a species *expression* which evaluates to a species *value* with no modification sites and with a globally unique name that is used in e.g. the underlying Petri net semantics. Hence the name of the species identifier, mrna, does not in itself hold any identity of a species, and we may bind the same

Listing 3.1. An LBS program for gene expression.

```
1 spec gene = new{}, rnap = new{}, mrna = new{}, prot = new{};
2 comp c new comp; comp n = new comp inside c;
3
4 c[
5   n[gene + rnap -> gene + rnap + mrna] |
6   n [mrna] -> mrna |
7   mrna -> prot
8 ]
```

identifier to an entirely different species in another part of the program. This allows different modules, possibly developed by different people, to use the same species identifier for mRNA molecules which are semantically and biologically different, and subsequently combine the modules into a single program without unintended cross-talk. On the other hand, when species *are* intended to be shared between modules, the species should be defined globally or made parameters of modules as we demonstrate next.

3.2 Parameterised Modules

We extend the basic gene expression program to express *two* proteins, prot1 and prot2, from two different genes, gene1 and gene2. We do so by abstracting the gene expression process into a *parameterised module* and invoking the module twice with the relevant parameters. The result is shown in Listing 3.2.

RNAP is defined globally in line 1, meaning that it will be shared between all instances of the module defined in lines $3-8$. This is biologically meaningful since the same RNAP species is used for transcription independently of the gene in question. The module is parameterised on the nucleus compartment, the gene and the target protein. The body is similar to before, except that the new mRNA species is defined locally. This means that each instance of the module uses semantically distinct mRNA, which again is biologically meaningful. Lines $10-13$ define the genes and proteins to be expressed together with the relevant compartments, and line 15 is a parallel composition of two module invocations inside the cytosol compartment. We could choose to define the nucleus compartment globally in this particular case, but instead give it as a common parameter in both module invocations in order to illustrate how this can be done in the more general case.

3.3 Species Expressions

New species, species identifiers and complexes are technically considered species expressions. Species identifiers can be bound to any species expressions, not just the new atomic ones, allowing large complexes to be defined once and used

Listing 3.2. A modular LBS program for gene expression instantiated with two genes and two target proteins.

```
1  spec rnap = new{};
2
3  module m(comp nuc; spec gene, prot) {
4    spec mrna = new{};
5    nuc[gene + rnap -> gene + rnap + mrna] |
6    nuc[mrna] -> mrna |
7    mrna -> prot
8  };
9
10 spec gene1 = new{}, prot1 = new{};
11 spec gene2 = new{}, prot2 = new{};
12 comp c = new comp;
13 comp n = new comp inside c;
14
15 c[ m(n, gene1, prot1) | m(n, gene2, prot2) ]
```

repeatedly. Species expressions also include a construct for *updating* the modification state of atomic species inside a complex. We illustrate this in Listing 3.3 which gives a more concise version of the CBS scaffolded MAPK cascade.

The first two lines consist of new species definitions as before, but now some of the species are defined with a modification site called p of boolean type. Lines 4-8 represent scaffold formation, but now the intermediate complexes are bound to identifiers using the **as** keyword. This *in-line* approach to binding is an abbreviation for binding a species expression to an identifier, then using the identifier in subsequent reactions, so e.g. the program:

```
1  Ste5 + Ste11 -> Ste5−Ste11 as a; ...
```

is an abbreviation for the program:

```
1  spec a = Ste5−Ste11;
2  Ste5 + Ste11 -> Ste5−Ste11 | ...
```

Reactions which have in-line definitions are composed in *sequence*, using the ; operator rather than the parallel one, as the order of such reactions matters since identifiers defined in one reaction can only be used in the following ones.

The species bound to e in line 8 is the full scaffold complex in its unphosphorylated form. Lines 10-12 represent the actual MAPK cascade. In line 10, the complex bound to e becomes the same complex, but updated by changing the phosphorylation state of site p in Ste11 to true. The result is then bound to a new identifier, f. The last two lines follow a similar pattern. When updates are made on atomic species we use an abbreviation and write e.g. Fus3{p} instead of Fus3<Fus3{p}>, although this is not relevant in the above example.

Listing 3.3. An LBS program for a scaffolded MAPK cascade in yeast.

```
spec Fus3=new{p:bool}, Ste7=new{p:bool}, Ste11=new{p:bool};
spec Ste5=new{p:bool}, Ste20=new{}, Gbg = new{};

Ste5 + Ste11 -> Ste5-Ste11 as a;
Ste7 + Fus3  -> Ste7-Fus3  as b;
a + b        -> a-b        as c;
c + Gbg      -> c-Gbg      as d;
d + Ste20    -> d-Ste20    as e;

e -> e<Ste11{p}> as f;
f -> f<Ste7{p}>  as g;
g -> g<Fus3{p}>
```

The reactions in the above LBS program avoid the redundancy which impairs the reactions in the corresponding CBS program in Listing 2.2. This improves readability. It also facilitates the process of program revision since adding e.g. a new phosphorylated site to the definition of Ste5 only involves a subsequent change to the first reaction. Contrast this to the corresponding CBS program in which the same revision requires two changes in each of the eight reactions.

There are two further, perhaps less commonly used, species expression operators which enable complex species to be taken apart. Assuming the definition of g given above, the *selection* expression g.Ste7 results in the atomic species from g identified by Ste7; the *removal* expression g\Ste7 results in the complex species g without Ste7. Hence the reaction g -> g.Ste7 + g\Ste7 represents dissociation of Ste7 from g and could in this case be written explicitly, if more laboriously, as follows:

```
Fus3-Ste7{p}-Ste11{p}-Ste5-Ste20-Gbg ->
  Ste7{p} + Fus3-Ste11{p}-Ste5-Ste20-Gbg
```

However, the species selection and removal operators are needed language constructs, not just notational conveniences. A species identifier used as the target of the selection and removal operators could be a formal parameter, and as a consequence of subtyping (introduced in the next subsection) its complete make-up in terms of atomic species may not generally be known.

If the complex bound to g is a homo-multimer and has several copies of e.g. Ste7, then selection and removal operate on all copies. This convention also applies to the update operator: if the target of an update contains multiple copies of the given species name, all copies are updated accordingly. It is however possible to make distinctions between different copies of the same species in homo-multimers, and we give an example of this when presenting the formal semantics of LBS.

Listing 3.4. A modular LBS program for the Ras/Raf/MEK/ERK signalling cascade.

```
1  module ph( spec k, s:{m} ) {
2        k + s -> k-s |
3        k-s -> k-s{m} |
4        k-s{m} -> k + s{m}
5  };
6
7  module dph( spec p, s:{m} ) {
8        p + s{m} -> p-s{m} |
9        p-s{m} -> p-s |
10       p-s  -> p + s
11 };
12
13 module cycle( spec k, p, s:{m} ) {
14      ph( k, s:{m} )  |  dph( p, s:{m} )
15 };
16
17 spec Ras = new{};
18 spec Raf = new{m:bool};
19 spec MEK = new{S218:bool, S222:bool};
20 spec ERK = new{T185:bool, Y187:bool};
21 spec PP2A1 = new{}, PP2A2 = new{}, MKP3 = new{};
22
23 cycle(Ras, PP2A1, Raf:{m}) |
24 cycle(Raf{m}, PP2A2, MEK:{S218}) |
25 cycle(Raf{m}, PP2A2, MEK{S218}:{S222}) |
26 cycle(MEK{S218,S222}, MKP3, ERK:{T185}) |
27 cycle(MEK{S218,S222}, MKP3, ERK{T185}:{Y187})
```

3.4 Parametric Type and Subtyping

We now return to the unscaffolded MAPK cascade and the issue of reusability. We noted that all the cycle modules used by the CBS program in Listing 2.3 have the same structure, each with two sets of reactions for representing, respectively, phosphorylation and dephosphorylation. The LBS program in listing 3.4 shows how a general, parameterised cycle module can be defined, which in turn relies on two general modules for phosphorylation and dephosphorylation.

The phosphorylation module named ph in lines 1-5 contains three reactions: binding of a kinase k and substrate s, phosphorylation of s in the bound state, and unbinding after phosphorylation. The two species are formal parameters, but in contrast to earlier examples, the formal parameter s has an *annotation* specifying that it must have a modification site m. The dephosphorylation module named dph in lines 7-11 follows a similar structure and is parameterised on a phosphatase p rather than a kinase. The cycle module in lines 13-15 is parameterised on a kinase, a phosphatase and a substrate and invokes the phosphorylation and dephosphorylation modules in parallel. The invocations provide

annotations for matching up the modification sites in the actual parameters with the corresponding modification sites in the formal parameters, which in this case is trivial since there is only the single choice, m. Note that there is scope for further abstraction since the phosphorylation and dephosphorylation modules are very similar. In fact they could be abstracted into a single module, but we refrain from doing so for the sake of clarity.

Lines 17-21 define the new species participating in the program and the remaining lines invoke modules for the appropriate cycles. Let us consider the invocation in line 24 in more detail. The first actual parameter, Raf{m}, provides Raf in its phosphorylated state as the kinase, and the second parameter provides PP2A2 as the phosphatase. The third parameter, MEK:{S218}, provides unphosphorylated MEK as the substrate and the annotation {S218} specifies the target site for phosphorylation. This raises two important points. Firstly, the names of modification sites in the actual and formal annotations differ, resulting in a notion of *parametric type*. The underlying semantics maintains a mapping from formal to actual modification site names when evaluating the body of a module. Secondly, there are two possible choices of modification sites to be phosphorylated in the actual parameter, namely S218 and S222. The annotation picks out the former, and the latter then plays no rôle from the perspective of the module. This results in a notion of *subtyping*: any actual parameter will do, as long as it contains at least the sites specified in the annotation and with types that match the corresponding formals. This corresponds to record subtyping in classical programming languages [30]. The module invocation in line 25 is similar but picks out the second site, S222, for phosphorylation, and also specifies that MEK is already phosphorylated on site S218.

In general, parameters may be complexes rather than atomic species. Suppose for example that MEK is in a complex with some other species a in the actual parameter in line 25. This can be written as follows:

```
1 ...
2 cycle(Raf{m}, PP2A2, MEK{S218}-a:MEK{S222}) |
3 ...
```

This results in an additional layer of subtyping: any actual parameter will do, as long as it contains at least the atomic species in the annotation. The annotation is here extended in order to specify that it is the atomic species MEK rather than a that should be mapped to the substrate. In fact the annotations used in Listing 3.4 are abbreviations of this more general form, so e.g. MEK{S218}:{S222} is an abbreviation of MEK{S218}:MEK{S222}. Similarly, the annotated formal parameter s:{m} in the cycle module abbreviates s:s{m}, and formal annotations may in general contain multiple atomic species.

We end the discussion of parameterised modules with an abstraction of the entire MAPK cascade into a module which is itself reusable. This, together with a module invocation, is shown in Listing 3.5.

Listing 3.5. A general, modular LBS program for the Ras/Raf/MEK/ERK signalling cascade. The species and module definitions from Listing 3.4 are omitted.

```
...
module mapk(
  spec k4, k3:{m}, k2:{m1, m2}, k1:{m1, m2},
       p3, p2, p1) {

  cycle(k4, p3, k3:{m}) |
  cycle(k3{m}, p2, k2:{m1}) |
  cycle(k3{m}, p2, k2{m1}:{m2}) |
  cycle(k2{m1,m2}, p1, k1:{m1}) |
  cycle(k2{m1,m2}, p1, k1{m1}:{m2})
};

mapk(Ras, Raf:{m}, MEK:{S218,S222}, ERK:{T185,Y187},
     PP2A1, PP2A2, MKP3)
```

3.5 Nondeterminism

Nondeterminism for contextual combinatorial explosion. In the previous examples we assumed that species only participate in reactions when they are atomic or when they are in the context of a specific complex as in the scaffolded MAPK cascade. In reality however, atomic species are likely to react in the context of many possible complexes. In the unscaffolded MAPK cascade, Raf may for example continue to function as a kinase for MEK when it is bound to its own kinases and/or phosphatases and when MEK is bound to its own phosphatase. This gives rise to a kind of combinatorial explosion which we call *contextual* and which is difficult to model in CBS under the basic semantics in terms of Petri nets, ODEs and CTMCs. The reason is that species in reactions are interpreted at face value with an "empty context" (except for modification site variables which allow combinatorial explosion at the level of modifications to be handled). In contrast, reaction rules in rule-based languages such as κ and BioNetGen are interpreted "in any context", and can be executed or analysed without generating the full set of empty context reactions which may in some cases be infinite. Languages such as κ and BioNetGen therefore handle combinatorial systems very well.

This could be exploited by giving a translation of LBS into κ or BioNetGen based on a suitable choice of modification site types, but we do not pursue this direction here. Instead we seek a middle ground in which all possible species contexts continue to be specified in reactions, but in a syntactically compact manner through the notion of *nondeterministic* species. Listing 3.6 shows an example of phosphorylation using nondeterministic versions of Raf and MEK.

The **or** operator expresses that either of its operands can take place in reactions where the expression is used. The distinguished species **SNil** is a neutral element under the complex formation operator, i.e. the axiom a−**SNil** = a holds

Listing 3.6. Phosphorylation using nondeterministic species

```
1 spec NRaf = Raf{m}−(SNil or Ras or PP2A1);
2 spec NMEK = MEK−(SNil or PP2A2);
3
4 ph(NRaf, NMEK:MEK{S218})
```

for any species a. The distributivity axiom a−(b **or** c) = a−b **or** a−c also holds for all species a, b and c. Hence line 1 in the above program expands to a choice of three species, namely Raf{m} in isolation or in complex with Ras or PP2A1, and line 2 expands to a choice of two species, namely MEK in isolation or in complex with PP2A2.

A reaction with nondeterministic species semantically gives rise to a number of parallel reactions, one for each possible choice of species. For example, the first reaction k + s −> k−s in the ph module now gives rise to a parallel composition of 6 reactions:

```
1 Raf{m} + MEK −> Raf{m}−MEK |
2 Raf{m}−Ras + MEK −> Raf{m}−Ras−MEK |
3 Raf{m}−PP2A1 + MEK −> Raf{m}−PP2A1−MEK |
4
5 Raf{m} + MEK−PP2A2 −> Raf{m}−MEK−PP2A2 |
6 Raf{m}−Ras + MEK−PP2A2 −> Raf{m}−Ras−MEK−PP2A2 |
7 Raf{m}−PP2A1 + MEK−PP2A2 −> Raf{m}−PP2A1−MEK−PP2A2 |
```

The two other reactions in the ph module have similar expansions, and the ph module invocation hence results in a total of 18 reactions.

Nondeterminism for species variant combinatorial explosion. The above example demonstrates how nondeterminism can be used to drastically reduce the size of programs in which the combinatorial explosion is contextual. Nondeterminism can also be used to handle combinatorial explosions arising from variants of individual proteins which largely react in the same way. For example, Raf has three variants RafA, RafB and RafC; MEK has two variants MEK1 and MEK2; and ERK has two variants ERK1 and ERK2 [28]. Rule-based languages such as κ and BioNetGen do *not* per se have any dedicated means of handling this source of nondeterminism, but a recent extension of κ provides some level of syntactical support [10]. Indeed this κ extension, together with our need to handle contextual combinatorial explosion, are the two motivating factors for the introduction of nondeterminism into LBS. Listing 3.7 shows how the MAPK cascade module can be used with species variants. In order to be of interest, one would expect that *some* reactions distinguish between the variants, but we omit this aspect in the present example.

Each individual member of the nondeterministic species in lines 16-18 is given a separate annotation at time of definition rather than at time of module invocation. The reason is that the members do not have any common atomic species, and in general they may not have common modification sites either,

Listing 3.7. The MAPK cascade module instantiated with nondeterministic species

```
1  ...
2  spec Ras  = new{};
3
4  spec RafA = new{m:bool};
5  spec RafB = new{m:bool};
6  spec RafC = new{m:bool};
7
8  spec MEK1 = new{S218:bool, S222:bool};
9  spec MEK2 = new{S218:bool, S222:bool};
10
11 spec ERK1 = new{T185:bool, Y187:bool};
12 spec ERK2 = new{T185:bool, Y187:bool};
13
14 spec PP2A1 = new{}, PP2A2 = new{}, MKP3 = new{};
15
16 spec NRaf = RafA:{m} or RafB:{m} or RafC:{m};
17 spec NMEK = MEK1:{S218, S222} or MEK2:{S218, S222};
18 spec NERK = ERK1:{T185, Y187} or ERK2:{T185, Y187};
19
20 mapk(Ras, NRaf, NMEK, NERK, PP2A1, PP2A2, MKP3)
```

so it is necessary to identify the atomic species and sites to be mapped from the corresponding formals on an individual basis; recall here that e.g. RafA:{m} and RafB:{m} abbreviate respectively RafA:RafA{m} and RafB:RafB{m}, so the annotations do indeed differ between different members of the nondeterministic species. For that reason also semantically, annotations are associated with species rather than with module invocations, and the mappings between formals and actuals are maintained locally within individual species rather than globally. Invocation of the mapk module results in 102 reactions as opposed to the 30 reactions in the original program.

The mechanism of nondeterministic selection. An important point about Listing 3.7 is that nondeterministic species are expanded *at the level of reactions* and not at the level of modules. This means that the mapk module invocation is *not* equivalent to a parallel composition of module invocations for each choice of species. Such an interpretation would result in 360 reactions, but $360 - 102 = 258$ of these would be duplicates, effectively adding up the rates of duplicated reactions, which is certainly not what we intend. In this respect LBS has a *call-by-name* semantics. On the other hand, species identifiers are resolved at time of module invocation where actual species parameters are formally evaluated to sets of species values, so in this respect LBS has a *call-by-value* semantics.

An additional subtlety arises in reactions where the same species occurs multiple times as a reactant or product. Consider for example the following:

```
spec a1 = new{}, a2 = new{}, b = new{};
spec a = a1 or a2;
a + a + b -> a–a–b
```

There are two reasonable, but very different, possibilities for expansion of the reaction. The first requires that the same choice for a is made within the scope of the reaction:

```
a1 + a1 + b -> a1–a1–b |
a2 + a2 + b -> a2–a2–b
```

The second possibility allows different copies of the same identifier to take different values, but with the correspondence between the occurrences on the reactant and product sides being preserved:

```
a1 + a1 + b -> a1–a1–b |
a1 + a2 + b -> a2–a2–b |
a2 + a1 + b -> a2–a1–b |
a2 + a2 + b -> a2–a2–b
```

The correct expansion depends on the specific application. Although the first may seem most appropriate in the general case, the second is for example useful for modelling the combinatorial dimerisation of different variants of ErbB receptors [10] (note that two of the resulting reactions are equivalent, effectively duplicating their rates). In order to cater for these different possibilities, LBS has two reaction arrows. The basic reaction arrow, −>, which has been used in the examples so far, results in the first expansion, and we call this a *selection arrow*. The double-headed reaction arrow, −>>, results in the second expansion, and we call this an *L-R equality-preserving arrow*.

In order to give a uniform semantical treatment of nondeterminism, and to enable other expansions than the two described above, LBS has a dedicated **force** operator for forcing nondeterministic choice. For example, the program:

```
spec a = force a1 or a2;
P
```

results in a parallel composition of P with a binding of a to a1 in one parallel component and P with a binding to a2 in the other parallel component. Reactions using either of the two arrows are then derived forms expressed in terms of the force operator and a third *deterministic* reaction arrow, =>, which requires that reactants and products do not contain nondeterministic species. For example, the program:

```
a + a + b -> a–a–b
```

abbreviates the program:

```
spec a = force(a);
spec b = force(b);
a + a + b => a–a–b
```

and the program:

```
a + a + b –>> a–a–b
```

abbreviates the program:

```
spec a1 = force(a);
spec a2 = force(a);
spec b1 = force(b);
a1 + a2 + b1 => a1–a2–b1
```

Nondeterministic species expressions which are not bound to identifiers are not allowed in reactions with any of the three arrows. This is because the implicit forcing is done on identifiers rather than on general species expressions, which allows the identity between different occurrences of e.g. the species a to be preserved after nondeterministic selection.

Limitations. In the examples we have assumed that reaction rates are independent of nondeterministic choices. This assumption appears to be in line with published models of e.g. the EGFR pathway [7], although this may be due to limited knowledge rather than biological reality. We note however that it *is* possible to choose different rates for different members of nondeterministic species based on their state of modification by using conditionals in rate expressions. Even with no distinguishable biological state of modification, one can "cheat" and add a site which serves the sole purpose of identifying members of nondeterministic species. But in the extreme case where all possible combinations of nondeterministic members in reactions give rise to different rates, this approach merely shifts the problem of combinatorial explosion from the reaction level to the reaction rate level. This is a potential practical limitation of nondeterminism.

A related limitation is the inability to constrain the choice of one nondeterministic species based on the choice of another; currently two different nondeterministic species in a reaction gives rise to reactions with *all* possible combinations of choices. This can however be addressed using a similar approach to that described above, where reactions with any incompatible choices are given rates of 0.

Finally, we note that even though nondeterminism can be used to write combinatorial programs in a syntactically compact manner, the resulting semantical objects may be too large for analysis or simulation to be feasible.

3.6 Model Variation

Given an LBS program it is sometimes of interest to vary it in a number of ways and examine the resulting effect on behaviour. In support of a structured approach to variations, LBS has a *variation operator*, $||$, which semantically

Listing 3.8. An extension of the MAPK program in Listing 3.7 (not repeated here) with variations for generating one semantical object for each possible initial condition.

```
1  ... |
2  (init RafA 500 || PNil) |
3  (init RafB 500 || PNil) |
4  (init RafC 500 || PNil) |
5
6  (init MEK1 500 || PNil) |
7  (init MEK2 500 || PNil) |
8
9  (init ERK1 500 || PNil) |
10 (init ERK2 500 || PNil)
```

gives the union of its operands, i.e. programs evaluate to *sets* of semantical objects. Listing 3.8 shows how the variation operator can be used to generate a semantical object for each of the given possible initial conditions of species variants in the MAPK cascade program.

Initial condition statements such as **init** RafA 500 are first-class programs and specify a given initial population or concentration for a species. If no initial conditions are specified in a program, the 0 initial population or concentration is assumed for all participating species. The distinguished program **PNil** is a neutral element under parallel composition, i.e. the axiom P | **PNil** = P holds for all programs P; also the distributivity axiom P1 | (P2 || P3) = (P1 | P2) || (P1 | P3) holds for all programs P1, P2 and P3. Hence the parallel composition shown above is a power set construction and expands to a variation composition of all $2^7 = 128$ possible combinations of initial conditions in parallel with the program represented by dots in line 1, in this case the previously defined MAPK cascade.

3.7 Output Species Parameters

Manipulations of large complexes are often spread across multiple modules. Sometimes there is a natural input-output relationship between these modules where a species which is constructed in one module may be the starting point for further manipulation in another. This applies for example to the scaffolded MAPK cascade program in Listing 3.3 which can benefit from a decomposition into two modules, one for scaffold formation, and one for the actual MAPK cascade. The fully formed scaffold in its unphosphorylated state can be considered as an output of the first module and as an input to the second. Although one could simply pass this connecting species as a common parameter to both modules, this would involve the entire scaffold to be written out at the time of module invocation, thus repeating the definitions already given during scaffold formation. In order to avoid this, we introduce the notion of *output species*, and a modular version of the yeast MAPK cascade using this idea is shown in Listing 3.9.

The first two lines define four new species while the remaining two species used in the program are defined locally in the formation module. The formation module

Listing 3.9. A modular LBS program for scaffold formation and a scaffolded MAPK cascade in yeast.

```
1  spec Fus3 = new{p:bool}, Ste7 = new{p:bool};
2  spec Ste11 = new{p:bool}, Ste5 = new {p:bool};
3
4  module formation(specout e : Fus3-Ste7-Ste11-Ste5) {
5    spec Ste20 = new{}, Gbg = new{};
6    Ste5 + Ste11 -> Ste5-Ste11 as a;
7    Ste7 + Fus3  -> Ste7-Fus3  as b;
8    a + b        -> a-b as c;
9    c + Gbg      -> c-Gbg as d;
10   d + Ste20    -> d-Ste20 as e
11 };
12
13 module mapk(spec a : k1{m}-k2{m}-k3{m}; specout d : a) {
14   a -> a<k3{m=tt}> as b;
15   b -> b<k2{m=tt}> as c;
16   c -> c<k1{m=tt}> as d
17 };
18
19 formation(spec link1);
20 mapk(link1 : Fus3{p}-Ste7{p}-Ste11{p}, spec link2);
21 ...
```

has a single formal parameter which specifies that the species e defined in the module body is given as an output, and the associated annotation specifies that the output contains the species Fus3, Ste7, Ste11 and Ste5. In fact the output also contains Ste20 and Gbg, but these are not exposed, which gives rise to a notion of subtyping similar to that of standard species parameters.

The mapk module has a parameter a and also an output species parameter d which is defined in the module body and is specified to contain at least the species of a. In line 19 the formation module is invoked and results in a binding of the identifier link1 to the output scaffold species. In line 20 this is passed on as a parameter to the mapk module which in turn results in a binding of the identifier link2 to the phosphorylated scaffold. In the full model of the yeast pheromone pathway, Ste5 dissociates from the scaffold link2 resulting from the MAPK cascade. We can deduce by inspection of the program that the complex bound to link2 does indeed contain the species Ste5, since link1 contains Ste5 and link2 contains at least the same species as link1. The full program contains five modules which can be connected naturally using this approach [27].

4 The Abstract Syntax of LBS

The previous section gave an informal introduction to the *concrete* syntax and main features of LBS. This section formally defines the *abstract* syntax of LBS which forms the basis of the general and concrete semantics given in the next

two sections. The formal definition of the concrete syntax and its mapping into the abstract syntax are omitted, since both can be deduced without surprises from the examples and from the abstract syntax.

In order to achieve our aim of generality, the abstract syntax is parameterised on a set of *modification site types* ρ and *modification site expressions* e_{m}. We divide the language into four main syntactic categories, for compartments, species, programs and definitions, and consider each in turn. But first we introduce the notation used in this section and throughout the paper.

4.1 Notation

We let $\mathbb{R}$ denote the set of *real numbers* and $\mathbb{N}$ denote the set of *natural numbers*. We write $\underline{x}$ for *lists*, $\underline{x}.i$ for the *ith element* (starting from 1) of a list, $|\underline{x}|$ for the *length* of a list and ε for the *empty* list. When a list should be thought of as representing a set, we write $\underline{\underline{x}}$ instead of $\underline{x}$. The set of *indices* of a list $\underline{x}$ is $\{i \mid 1 \leq i \leq |x|\}$. The *sublist* of $\underline{x}$ consisting of the elements at some subset I of the indices of $\underline{x}$ is written $\underline{x}.I$. The *Cartesian product* $\underline{x} \times_\circ \underline{y}$, where $\circ$ is a pairing operator on elements of the respective lists, is the list of length $|\underline{x}| \cdot |\underline{y}|$ s.t. $(\underline{x} \times_\circ \underline{y}).((i-1)|\underline{x}| + j) \stackrel{\Delta}{\simeq} \underline{x}.i \circ \underline{y}.j$. The *concatenation* of lists $\underline{x}$ and $\underline{y}$ is written $\underline{x}\,\underline{y}$, and the *prefix* and *postfix* of an element a to a list $\underline{x}$ are written $a\underline{x}$ and $\underline{x}a$, respectively.

We write $\{x_i\}_{i \in I}$ for a finite *indexed set* and omit I and/or i and write $\{x_i\}_I$, $\{x_i\}$ or $\{x\}$ when they are understood from the context. The set of *multisets* of a set S is denoted by $MS(S)$ and is defined as the set of total functions from S to the natural numbers, i.e. $MS(S) \stackrel{\Delta}{\simeq} S \to \mathbb{N}$. We adopt the usual multiset notation and write e.g. $x + 2 \cdot y$ for the multiset containing the element x and two copies of y, and we write $MS(\underline{x})$ for the multiset representation of a list $\underline{x}$. We also use the standard notation $\prod_{i \in I} X_i$ for dependent sets.

We write $x \stackrel{\Delta}{\simeq} y$ for definitions where x equals y if y is defined, and where x is undefined otherwise. When a notion of well-typedness applies to y, we furthermore write $x \stackrel{\Delta}{\simeq}_t y$ for definitions where x equals y if y is defined and well-typed, and where x is undefined otherwise.

Partial finite *functions* f are denoted by finite indexed sets of pairs $\{x_i \mapsto y_i\}$ where $f(x_i) = y_i$. The *empty* function is correspondingly denoted by $\emptyset$. The *domain of definition* and *image* of a function f are denoted by $dom(f)$ and $im(f)$, respectively. For functions f and g we define the *update* of f by g, written $f\langle g \rangle$, as follows:

$$f\langle g \rangle(x) \stackrel{\Delta}{\simeq} \begin{cases} f(x) & \text{if } x \in dom(f) \setminus dom(g) \\ g(x) & \text{if } x \in dom(g) \end{cases}$$

If g consists of a single binding $x \mapsto y$ we write $f\langle x \mapsto y \rangle$ instead of $f\langle \{x \mapsto y\} \rangle$. We specify the type of a partial function f by writing $f(x) = y$ where x and y are given variables ranging over two sets; these sets are then understood to form the domain and image of f.

When an element of a list or an indexed set is referred to without explicit quantification in a semantical definition, the index is assumed to be universally quantified over a set which is understood from the context. Under such circumstances we often omit the index and write e.g. x instead of $\underline{x}.i$. If $\circ$ is an operation on the elements of lists $\underline{x}$ and $\underline{y}$ both of length n, we write $\underline{x \circ y}$ for the list of length n in which the ith element is $\underline{x}.i \circ \underline{y}.i$.

4.2 Compartments

Compartment expressions. The abstract syntax for basic compartment expressions is shown in Table 4.1, where id_{c} ranges over the set of *compartment identifiers* and $r \in \mathbb{R}$. New compartments are created using the new compartment expression which explicitly records a parent compartment and a volume. In cases where a compartment is used at the top level, the world compartment can be specified as a parent, hence allowing compartment hierarchies to be terminated. A compartment is generally used in multiple contexts by binding it to an identifier at time of creation. The nil compartment functions as a neutral element for the composition of compartment lists. It is paired with a parent compartment, which is necessary for type-checking of compartment hierarchies. Nil compartments can for example be used to decrease the depth of a module hierarchy when passed as parameters to modules.

Table 4.1. The abstract syntax for compartment expressions

$e_{\mathrm{c}} ::=$	Compartment expression
$\mid$ **new comp vol** r **inside** e_{c}	New compartment
$\mid \top$	World compartment
$\mid id_{\mathrm{c}}$	Compartment identifier
$\mid \mathbf{1}_{\mathrm{c}}$ **in** e_{c}	Nil compartment

Although the world compartment figures as a general compartment in the abstract syntax, it is only intended for use as a parent of new compartments and of the nil compartment. It is not intended for use in e.g. reactions, and its proper usage is enforced in the semantics for programs. One could enforce this intended usage syntactically by introducing separate production rules for top level and nested new compartment and nil compartment expressions. However, whether or not a compartment features at the global top level of a program is not generally known at time of definition: take for example compartment definitions inside a module, where the parent compartment may be a formal parameter.

Derived compartment expressions. The volume in new compartment expressions may be omitted, in which case a default volume of 1.0 is assumed. The parent compartment in new compartment and nil compartment expressions may also be omitted, in which case the world parent compartment is assumed.

4.3 Species

Modification site expressions. Recall that the abstract syntax for species expressions is parameterised on a set of modification site types ρ and a set of modification site expressions e_{m}. Since boolean expressions are of widespread practical use as demonstrated in the examples, we assume that the set of modification types contains the boolean type **bool**, and that the set of modification site expressions contains the boolean expressions e_{b} generated by the grammar in Table 4.2 where x ranges over the set of *variables*. The boolean expressions contain the usual **tt**/**ff** base values and a minimal set of connectives from which the full set of boolean connectives can be defined as derived forms in the usual manner. Variables are used to create species expressions which can match multiple concrete species. We assume that the set of variables is closed by prefixing of underscore-terminated binary strings, i.e. that b_x is a variable for all $b \in \{0,1\}^*$; this is needed to confine variables to their appropriate namespace in the semantics. The type annotation of variables is likewise used for technical convenience in the semantics.

Table 4.2. The abstract syntax for boolean expressions

e_{b} ::=	Boolean expression
$\mid$ **tt**	True
$\mid$ **ff**	False
$\mid$ x : **bool**	Typed variable
$\mid$ e_{b} **or** e'_{b} $\mid$ **not** e_{b}	Boolean operators

Species expressions. The abstract syntax for species expressions is shown in Table 4.3, where n_{s} ranges over the set of *species names*, n_{m} ranges over the set of *modification site names* and id_{s} ranges over the set of *species identifiers*. Species names identify atomic species independently of any modification sites, while species identifiers refer to possibly complex species including both the names and modification states of atomic species in the complex. We assume for technical reasons that both the set of species identifiers and the set of binary strings is contained in the set of species names. We assume furthermore, as for variables, that the set of species identifiers is closed by prefixing of underscore-terminated binary strings, i.e. that b_id_{s} is also a species identifier for all $b \in \{0,1\}^*$.

The grammar distinguishes between species expressions e_{s} and extended species expressions $e_{\mathrm{s}+}$ which add the new atomic species expression. This is because new species expressions only make sense in the context of definitions, where the resulting new species value can be bound to an identifier. Species bound to an identifier can then be used in multiple contexts and given an initial population through the construct given in the abstract syntax for programs. Technically, separating out the new species expression alleviates the need to consider fresh names in the semantics for the remaining expressions and for certain cases of programs; this significantly simplifies the presentation.

Table 4.3. The abstract syntax for species expressions

$e_{s+} ::= e_s$	Extended species expression
$\mid$ **new** n_s, σ	New atomic species
$e_s ::=$	Species expression
$\mid$ $id_c[e_s]$	Located species
$\mid$ $e_s - e'_s$	Composite species
$\mid$ $e_s.\underline{id_c}[n_s]$	Species selection
$\mid$ $e_s \backslash \underline{id_c}[n_s]$	Species removal
$\mid$ $e_s\langle \underline{id_c}[n_s, \alpha] \rangle$	Species update
$\mid$ e_s **or** e'_s	Species choice
$\mid$ $e_s : \xi$	Species annotation
$\mid$ id_s	Species identifier
$\mid$ $\mathbf{0}_s$	Nil species
$\xi ::= \underline{\underline{id_c}[n_s, n_m]}$	Annotation
$\sigma ::= \{n_m \mapsto \rho\}$	Modification type
$\alpha ::= \{n_m \mapsto e_m\}$	Modification assignment

New atomic species are created by specifying a name and a type consisting of a partial finite function from modification site names to modification site types. The modification sites are assigned default expressions appropriate for the corresponding type, e.g. **ff** in the case of the **bool** type. In contrast to new compartment expressions, a new species expression explicitly includes a species name. Often this name is the same as the identifier to which the new species expression is assigned, which is reflected in a derived form of definitions. Although semantically the underlying unique species name will be freshly generated, the specified name is used to identify specific atomic species in subsequent species selection, removal and update expressions. Species names rather than general species expressions are used here for two reasons. First, the update expression updates a specific atomic species in a complex. Second, atomic species names are local to a species, meaning that the same atomic species name in two different species may map to different underlying fresh species names. This is used to cater for nondeterminism in the context of parametric types in species parameters of modules. Similar considerations of nondeterminism apply to compartments, which are only used in species expressions indirectly through compartment identifiers rather than through general compartment expressions.

The species annotations necessary to match the names and sites of actual parameters to those of formal parameters are handled in the abstract syntax for species expressions rather than in the abstract syntax for module invocation in programs. This too is because of nondeterminism where separate annotations

may be required for each member of a nondeterministic species expression as demonstrated previously in Listing 3.7.

Derived species expressions. Two derived forms of species expressions allow updates and annotations of atomic species without having to repeat atomic species names. Specifically, the expressions:

$$id_{\mathrm{s}}\{\alpha\} \quad \text{and} \quad id_{\mathrm{s}} : \underline{n_{\mathrm{m}}}$$

abbreviate respectively the expressions:

$$id_{\mathrm{s}}\langle\varepsilon[id_{\mathrm{s}}, \alpha]\rangle \quad \text{and} \quad id_{\mathrm{s}} : \underline{\varepsilon[id_{\mathrm{s}}, \underline{n_{\mathrm{m}}}]}$$

4.4 Programs

Basic programs. The abstract syntax for programs is shown in Table 4.4, where $n \in \mathbb{N}$, id_{p} ranges over the set of *program identifiers* and id_{a} ranges over the set of *algebraic rate function identifiers*. Definitions, ranged over by D, are treated in the next subsection. Module invocations include actual parameters for compartments, species, rates and output species, and as already pointed out, the annotations of actual species parameters necessary to match the formal parameters are handled in the abstract syntax for species expressions.

Reaction rate expressions can either be mass-action rates, given inside braces, or general algebraic rate expressions, given inside square brackets. Algebraic rate expressions include rate constants, compartments and species, where the latter two are interpreted as respectively a volume and a population. Algebraic rate expressions also include a number of basic functions and arithmetic operators which feature regularly in the biological literature; these are inspired by similar features found in BioPEPA. Custom rate functions which are parameterised on compartments, species and algebraic rate expressions can be defined and invoked repeatedly. These also allow the definition of common rate functions for e.g. Michaelis-Menten or Hill kinetics. Conditionals enable different rates to be chosen depending on the state of modification of reactants as recorded by match variables. This mechanism also allows a distinction between different members of a nondeterministic species to be made assuming a convention where an additional site is added in which a value identifies each individual nondeterministic member. Note that mass-action rates are represented by algebraic expressions in the abstract syntax because this allows for a uniform treatment of defined constants and conditionals. Semantically however, mass-action algebraic rate expressions are required to evaluate to constants.

Only the simplest possible reaction is included in the abstract syntax for programs. Species expressions are assumed to be deterministic, requiring any nondeterministic selection to be carried out in advance through the use of the force operator; there are no in-line species definitions; and there are no reversible or enzymatic reactions.

Table 4.4. The abstract syntax for basic programs

$P ::=$	Program
$\mid \underline{n \cdot e_{\mathrm{s}}} \Rightarrow^{e_{\mathrm{r}}} \underline{n' \cdot e'_{\mathrm{s}}}$ **if** e_{b}	Reaction
$\mid \mathbf{0}_{\mathrm{p}}$	Nil program
$\mid P \mid P'$	Parallel composition
$\mid P \mid\mid P'$	Variation composition
$\mid id_{\mathrm{c}}[P]$	Located program
$\mid D \ ; \ P$	Definition
$\mid id_{\mathrm{p}}(\underline{e_{\mathrm{c}}}; \underline{e_{\mathrm{s}+}}; \underline{e_{\mathrm{a}}}; \mathbf{out}\ \underline{id_{\mathrm{s}}}) \ ; \ P$	Module invocation
$\mid id_{\mathrm{s}} =$ **force** $e_{\mathrm{s}} \ ; \ P$	Nondeterministic selection
$\mid$ **init** $e_{\mathrm{s}} = r$	Initial population
$e_{\mathrm{r}} ::=$	Rate expression
$\mid \{e_{\mathrm{a}}\}$	Mass-action rate
$\mid [e_{\mathrm{a}}]$	Algebraic rate
$e_{\mathrm{a}} ::=$	Algebraic rate expression
$\mid r$	Constant
$\mid id_{\mathrm{c}}$	Volume
$\mid e_{\mathrm{s}}$	Population
$\mid$ **if** e_{b} **then** e_{a} **else** e'_{a}	Conditional
$\mid id_{\mathrm{a}}(\underline{e_{\mathrm{c}}}; \underline{e_{\mathrm{s}}}; \underline{e_{\mathrm{a}}})$	Function invocation
$\mid exp(e_{\mathrm{a}}) \mid log(e_{\mathrm{a}}) \mid sin(e_{\mathrm{a}}) \mid cos(e_{\mathrm{a}})$	Standard functions
$\mid e_{\mathrm{a}} + e'_{\mathrm{a}} \mid e_{\mathrm{a}} - e'_{\mathrm{a}}$	Arithmetic operators
$\mid e_{\mathrm{a}} \times e'_{\mathrm{a}} \mid e_{\mathrm{a}} \ / \ e'_{\mathrm{a}} \mid e_{\mathrm{a}} \hat{} e'_{\mathrm{a}}$	

Derived programs. More complicated reactions are generated by the abstract syntax for derived programs in Table 4.5, all of which can be defined in terms of basic programs. The dots in the grammar indicate extension of the grammar for basic programs. Derived programs include the two additional reaction arrows which cater for nondeterministic species and which implicitly force nondeterministic selection in two different manners, as exemplified in section 3.5. Enzymatic reactions are given by a list of enzymes to the left of the tilde symbol. All types of reactions can be reversible with any combination of mass action and general algebraic rate expressions for each of the two directions. Finally, species expressions in derived reactions may contain in-line definitions which go into scope in the sequential program following the reaction.

Further derived forms arise by omitting the enzyme or boolean expression parts of reactions. The absence of an enzyme part is understood as an enzyme part with an empty list of species, and the absence of a boolean expression part

Table 4.5. The abstract syntax for derived programs

$P ::= \ldots$	Derived program
$\mid \underline{e_{\mathrm{s}}''} \sim \underline{n \cdot e_{\mathrm{s}}}\ A^{e_{\mathrm{r}}}\ \underline{n' \cdot e_{\mathrm{s}}'}$ **if** $e_{\mathrm{b}}\,;\, P$	General reaction
$\mid \underline{e_{\mathrm{s}}''} \sim \underline{n \cdot e_{\mathrm{s}}}\ A_2^{e_{\mathrm{r}},e_{\mathrm{r}}'}\ \underline{n' \cdot e_{\mathrm{s}}'} \sim \underline{e_{\mathrm{s}}'''}$ **if** $e_{\mathrm{b}}, e_{\mathrm{b}}'\,;\, P$	General reversible reaction
$A ::=$	Reaction arrows
$\mid \Rightarrow$	Deterministic arrow
$\mid \rightarrow$	Selection arrow
$\mid \twoheadrightarrow$	L-R equality-preserving arrow
$A_2 ::= \Leftrightarrow \mid \leftrightarrow \mid \twoheadleftarrow\!\!\!\!\twoheadrightarrow$	Reversible reaction arrows
$e_{\mathrm{s}} ::= \ldots$	Derived species expressions
$\mid e_{\mathrm{s}}$ **as** id_{s}	Inline definition

is understood as a boolean expression part with the expression **tt**. Stoichiometry in reactions can be omitted, in which case stoichiometry 1 is assumed. Finally, the sequential programs following reactions and module invocations can be omitted when there are no in-line species definitions or output species parameters, respectively. In these cases the nil sequential program is assumed.

4.5 Definitions

Basic definitions. The abstract syntax for definitions is shown in Table 4.6 and should be self-explanatory. Formal species parameters have annotations ξ as defined in the abstract syntax for species expressions. Together with the corresponding annotation of actual species parameters, this is sufficient to construct a mapping that allows use of the species inside the module body.

Derived definitions. There is one important derived form concerning new species definitions. Recall from the abstract syntax for species that a new species

Table 4.6. The abstract syntax for definitions

$D ::=$	Definition
$\mid id_{\mathrm{s}} = e_{\mathrm{s}+}$	Species
$\mid id_{\mathrm{c}} = e_{\mathrm{c}}$	Compartment
$\mid id_{\mathrm{a}}(\underline{id_{\mathrm{c}}}; \underline{id_{\mathrm{s}} : \xi}; \underline{id_{\mathrm{a}}}) = e_{\mathrm{a}}$	Function
$\mid id_{\mathrm{p}}(\underline{id_{\mathrm{c}}}; \underline{id_{\mathrm{s}} : \xi}; \underline{id_{\mathrm{a}}};$ **out** $\underline{id_{\mathrm{s}}' : e_{\mathrm{s}}}) = P$	Module

expression includes a species name. But in most cases this name will be identical to the identifier that the new species expression is bound to. The name can then be omitted, i.e. the expression:

$$id_{\mathrm{s}} = \mathbf{new}\ \sigma$$

abbreviates the expression:

$$id_{\mathrm{s}} = \mathbf{new}\ id_{\mathrm{s}}, \sigma$$

This is the reason that the set of species names is assumed to contain the set of species identifiers.

5 The General Semantics

This section defines a denotational framework for compositionally assigning semantical objects such as Petri nets to LBS programs. Our aim is to abstract away from the specific kind of semantical object under consideration.

Assumptions. We achieve our aim of abstraction by assuming a given structure $(S, |_S, \mathbf{0}_S, R_S, I_S)$ consisting of:

- A set S of *semantical objects* ranged over by $\mathcal{O}$.
- A partial binary *composition function* $|_S$ on semantical objects.
- A distinguished *nil* semantical object $\mathbf{0}_S \in S$.
- A partial *reaction assignment function* of the form $R_S(R, b) = \mathcal{O}$ assigning a semantical object to a given reaction R, named b, in a normal form, defined below (b is used to name e.g. Petri net transitions).
- A partial *initial condition assignment function* of the form $I_S(v_{\mathrm{gns}}, r) = \mathcal{O}$ assigning a semantical object to an initial population or concentration r of species v_{gns} in a ground normal form, defined below.

The last implies that semantical objects have a representation of initial conditions, e.g. an initial marking in the case of a Petri net. Specific examples are given in Section 6.

Recall that the abstract syntax is parameterised on modification site types ρ and modification site expressions e_{m}. We assume the following relations and functions pertaining to these:

- A *typing relation* of the form $e_{\mathrm{m}} : \rho$ giving types to modification site expressions. This is used for determining well-typedness of species expressions.
- A *default expression function* of the form $\mathit{default}(\rho) = e_{\mathrm{m}}$ giving default expressions to types. This is used for assigning expressions to unassigned sites in species expressions.
- A *variable function* of the form $FV(e_{\mathrm{m}}) = \{x_i : \rho_i\}$ giving the set of (typed) variables in a modification site expression. This is used for determining well-typedness of reactions and for computing semantical objects of reactions in some of the concrete semantics.

- An *expression denotation function* of the form $[\![e_{\mathrm{m}}]\!]_{\mathrm{m}}\Gamma_{\mathrm{x}} = v_{\mathrm{m}}$ for evaluating a modification site expression to a value v_{m} in a given set $[\![\rho]\!]_{\mathrm{t}}$ where $e_{\mathrm{m}} : \rho$, given a *variable environment* of the form $\Gamma_{\mathrm{x}}(x : \rho) = v_{\mathrm{m}}$ assigning values $v_{\mathrm{m}} \in [\![\rho]\!]_{\mathrm{t}}$ to typed variables. This is used for computing semantical objects of reactions in some of the concrete semantics.
- An *update function* of the form $e_{\mathrm{m}}\langle e'_{\mathrm{m}}\rangle = e''_{\mathrm{m}}$ for updating one modification site expression with another. This is used in the semantics of species update expressions. While this operation is trivial for e.g. boolean expressions in which the original expression is simply disregarded, we anticipate the addition of other types for which the situation is more subtle.
- A *seal function* of the form $\mathit{seal}(e_{\mathrm{m}}, b) = e'_{\mathrm{m}}$ for confining names in modification site expressions to a *namespace* given by a binary string $b \in \{0,1\}^*$. The namespace is used to avoid capture of e.g. variables in actual species parameters when used inside the body of a module.

In the case where only the boolean modification site type is given, and where the set of modification site expressions is hence the set of boolean expressions, the above functions can be defined as follows:

- $e_{\mathrm{m}} : \mathbf{bool}$ for all e_{m}.
- $\mathit{default}(\rho) \stackrel{\Delta}{\simeq} \mathbf{ff}$.
- $\mathit{FV}(e_{\mathrm{m}})$ is defined inductively as follows:
  - $\mathit{FV}(\mathbf{tt}) \stackrel{\Delta}{\simeq} \emptyset$
  - $\mathit{FV}(\mathbf{ff}) \stackrel{\Delta}{\simeq} \emptyset$
  - $\mathit{FV}(x : \mathbf{bool}) \stackrel{\Delta}{\simeq} \{x : \mathbf{bool}\}$
  - $\mathit{FV}(e_{\mathrm{b}}\ \mathbf{or}\ e'_{\mathrm{b}}) \stackrel{\Delta}{\simeq} \mathit{FV}(e_{\mathrm{b}}) \cup \mathit{FV}(e'_{\mathrm{b}})$
  - $\mathit{FV}(\mathbf{not}\ e_{\mathrm{b}}) \stackrel{\Delta}{\simeq} \mathit{FV}(e_{\mathrm{b}})$
- $[\![e_{\mathrm{b}}]\!]_{\mathrm{m}}\Gamma_{\mathrm{x}} = [\![e_{\mathrm{b}}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}}$ is defined inductively as follows:
  - $[\![\mathbf{tt}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} \mathbf{tt}$
  - $[\![\mathbf{ff}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} \mathbf{ff}$
  - $[\![x : \mathbf{bool}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{x}}(x)$
  - $[\![e_{\mathrm{b}}\ \mathbf{or}\ e'_{\mathrm{b}}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} \begin{cases} \mathbf{tt} & \text{if } [\![e_{\mathrm{b}}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} = \mathbf{tt} \text{ or } [\![e_{\mathrm{b}}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} = \mathbf{tt} \\ \mathbf{ff} & \text{otherwise} \end{cases}$
  - $[\![\mathbf{not}\ e_{\mathrm{b}}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} \begin{cases} \mathbf{tt} & \text{if } [\![e_{\mathrm{b}}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} = \mathbf{ff} \\ \mathbf{ff} & \text{otherwise} \end{cases}$
- $e_{\mathrm{m}}\langle e'_{\mathrm{m}}\rangle = e'_{\mathrm{m}}$
- $\mathit{seal}(e_{\mathrm{m}}, b)$ is defined inductively as follows:
  - $\mathit{seal}(\mathbf{tt}, b) \stackrel{\Delta}{\simeq} \emptyset$

- $seal(\mathbf{ff}, b) \stackrel{\Delta}{\simeq} \emptyset$
- $seal(x : \mathbf{bool}, b) \stackrel{\Delta}{\simeq} b\_x : \mathbf{bool}$
- $seal(e_{\mathrm{b}}\ \mathbf{or}\ e'_{\mathrm{b}}, b) \stackrel{\Delta}{\simeq} seal(e_{\mathrm{b}}, b)\ \mathbf{or}\ seal(e'_{\mathrm{b}}, b)$
- $seal(\mathbf{not}\ e_{\mathrm{b}}, b) \stackrel{\Delta}{\simeq} \mathbf{not}\ seal(e_{\mathrm{b}}, b)$

Overview. As for the abstract syntax, the semantics is presented in four subsections each treating one of the four syntactic categories in detail. An overview of the denotation functions and associated symbols is given in Tables 5.1 and 5.2. The environments are partial finite functions from appropriate sets of identifiers to appropriate sets of values. For the rate function and module environments these values are themselves functions mapping actual parameters to some other appropriate values. The binary string b is a parameter of some of the denotation functions which pass it on to the *seal* and R_S functions. Freshness of b is ensured by appropriate extensions as denotation functions are computed. This follows the approach of CBS, except that in CBS, fresh names are computed bottom-up, whereas we compute them top-down in order to avoid some unpleasant technicalities.

Table 5.1. Denotation functions

Function signature	**Denotation of**
$[\![e_{\mathrm{c}}]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b = v_{\mathrm{c}}$	Compartment expressions
$[\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}} = \underset{\sim}{v_{\mathrm{s}}}$	Species expressions
$[\![e_{\mathrm{s}+}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, b = \underset{\sim}{v_{\mathrm{s}}}$	Extended species expressions
$[\![e_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \underline{v_{\mathrm{c}}} = v_{\mathrm{a}}$	Algebraic rate expressions
$[\![e_{\mathrm{m}}]\!]_{\mathrm{m}}\Gamma_{\mathrm{x}} = v_{\mathrm{m}}$	Modification site expressions
$[\![P]\!]_{\mathrm{p}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b, \underline{v_{\mathrm{c}}} = \{(\mathcal{O}_i, \Gamma_{\mathrm{so}\,i})\}$	Programs
$[\![D]\!]_{\mathrm{d}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b = \Gamma'_{\mathrm{c}}, \Gamma'_{\mathrm{s}}, \Gamma'_{\mathrm{a}}, \Gamma'_{\mathrm{m}}, \Gamma_{\mathrm{so}}$	Definitions

Table 5.2. Symbols in the denotation function signatures

Symbol	**Description**
v_{c}	Compartment value
v_{s}	Species value
v_{a}	Algebraic rate value
v_{m}	Modification site value
$\mathcal{O}$	Semantical object
b	Binary string
Γ_{c}	Compartment environment
Γ_{s}	Species environment
Γ_{a}	Algebraic rate function environment
Γ_{x}	Variable environment
Γ_{m}	Module environment
Γ_{so}	Output species environment

5.1 Compartments

Compartment values. We let n_{c} range over a given set of *compartment names* which is assumed to include the set of binary strings and contain the nil compartment name $\mathbf{1}_{\mathrm{c}}$. In contrast to compartment identifiers, which are language constructs used for binding compartment values, compartment names are used to uniquely and globally identify a compartment. Compartment values are of the following form:

$v_{\mathrm{c}} ::=$	Compartment value
$\mid (n_{\mathrm{c}}, r, v_{\mathrm{c}})$	Nested compartment
$\mid \top$	World compartment

We let V_{c} denote the set of all compartment values generated by this grammar. Parent compartments v_{c} are recorded as values rather than names, since names do not generally identify a value uniquely. Specifically, $\mathbf{1}_{\mathrm{c}}$ may occur in compartment values with different parents. Compartment volumes r represent the volume of a compartment in the "biological sense" that the volume of a child compartment does not count towards the volume of its enclosing parent.

The denotation function. A *compartment environment* is a partial finite function of the form $\Gamma_{\mathrm{c}}(id_{\mathrm{c}}) = v_{\mathrm{c}}$ mapping compartment identifiers to compartment values. The denotation function for compartment expressions is of the form:

$$[\![e_{\mathrm{c}}]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b = v_{\mathrm{c}}$$

and is defined inductively as follows:

- $[\![\textbf{new comp vol } r \textbf{ inside } e_{\mathrm{c}}]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b \stackrel{\Delta}{\simeq} (b, r, v_{\mathrm{c}})$ where
  - $v_{\mathrm{c}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{c}}]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, 0b$
- $[\![\top]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b \stackrel{\Delta}{\simeq} \top$
- $[\![\mathbf{1}_{\mathrm{c}} \textbf{ inside } e_{\mathrm{c}}]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b \stackrel{\Delta}{\simeq} (\mathbf{1}_{\mathrm{c}}, 0.0, v_{\mathrm{c}})$ where
  - $v_{\mathrm{c}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{c}}]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b$
- $[\![id_{\mathrm{c}}]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}(id_{\mathrm{c}})$

The denotation function is partial since it is not defined for identifiers which do not have bindings in the given environment. New compartments are named by the binary string argument to the denotation function. The denotation of the parent compartment is computed recursively, but with a 0 prefixed to the

fresh name, hence resulting in a new fresh name. Nil compartment values are arbitrarily given the volume 0.0.

Well-typedness of compartment value lists. Compartments generally occur in the context of lists of other compartments, and we are only interested in such lists which respect the hierarchy captured in compartment values. Formally, we say that a *list* $\underline{(n_c, r, v_c)}$ *is well-typed* if $\underline{v_c}.i = \underline{(n_c, r, v_c)}.(i-1)$ for $i \in 2 \ldots |\underline{v_c}|$. Any other lists, including those which contain the world compartment in any other position than possibly the first, are ill-typed.

Compartment lists in turn occur in the context of sets of other compartment lists, and we are only interested in such sets where all compartment lists agree on parent compartments. To formalise this, we define a function of the form $\mathit{parent}(\underline{v_c}) = \{v_{c_i}'\}$ which gives the set of legal parent compartments of a compartment list:

$$\mathit{parent}(\underline{v_c}) \stackrel{\Delta}{\simeq} \begin{cases} V_c & \text{if } \underline{v_c} = \varepsilon \\ \{v_c'\} & \text{if } |\underline{v_c}| > 0 \text{ and } \underline{v_c}.1 = (n_c, r, v_c') \\ \emptyset & \text{if } |\underline{v_c}| > 0 \text{ and } \underline{v_c}.1 = \top \end{cases}$$

In words, the empty list of compartments can be put inside any compartment; a non-empty list of compartments can only be put inside the compartment specified by the first element of the list unless this is the world compartment, in which case it can be put nowhere. Formally, we then say that a *set* $\{\underline{v_{c_i}}\}$ *is well-typed* if all $\underline{v_{c_i}}$ are well-typed and either $\mathit{parent}(\underline{v_{c_i}}) = \emptyset$ for all i or $\bigcap_i \mathit{parent}(\underline{v_{c_i}}) \neq \emptyset$.

The forest structure of well-typed sets of compartment value lists. The motivation for defining well-typedness of sets of compartment value lists is that only physically meaningful compartment hierarchies should be allowed in programs. By this we mean that sets of compartment value lists should form a forest structure, here a directed acyclic graph in which each node has at most one parent.

Observe first that one can obtain a directed graph from a well-typed set of compartment value lists in which nodes are compartment values and edges are determined by left-to-right neighbourhood in lists. Formally, given a well-typed set $\{\underline{v_{c_i}}\}$ we define $G\{\underline{v_{c_i}}\} \stackrel{\Delta}{\simeq} (V, E)$ where

$$\begin{aligned} V &\stackrel{\Delta}{\simeq} \{\underline{v_{c_i}}.j\} \\ E &\stackrel{\Delta}{\simeq} \{(\underline{v_{c_i}}.j, \underline{v_{c_i}}.(j+1))\} \end{aligned}$$

(Recall from our notational convention that the above definitions give indexed sets where i is an index into the set of compartment value lists and j is an index into list positions). We now show that these graphs are indeed forests:

Proposition 5.1. $G\{\underline{v_{c_i}}\}$ *is a forest.*

Proof. By induction in $|\{\underline{v_{\mathrm{c}_i}}\}|$. In the following we additionally use a and b to range over compartment values.

- **Basis** ($\{\underline{v_{\mathrm{c}_i}}\} = \emptyset$). Holds vacuously.
- **Step** ($\{\underline{v_{\mathrm{c}_i}}\} \cup \{\underline{v'_{\mathrm{c}}}\}$).
  Acyclic: by the induction hypothesis, $G\{\underline{v_{\mathrm{c}_i}}\}$ is acyclic. Also $G\{\underline{v'_{\mathrm{c}}}\}$ is acyclic, for otherwise $\underline{v'_{\mathrm{c}}}$ would take the form $\underline{v_{\mathrm{c}}}'_1\, a\, \underline{v_{\mathrm{c}}}'_2\, a\, \underline{v_{\mathrm{c}}}'_3$, and it follows from well-typedness that the compartment value a must include itself as an ancestor; this is impossible since compartment values are finite. Suppose towards a contradiction that there is a cycle in $G(\{\underline{v_{\mathrm{c}_i}}\} \cup \{\underline{v'_{\mathrm{c}}}\})$. This can then only arise from a branch in $G\{\underline{v_{\mathrm{c}_i}}\}$ of the form $\underline{v_{\mathrm{c}}}_1\, a\, \underline{v_{\mathrm{c}}}_2\, b\, \underline{v_{\mathrm{c}}}_3$ and $\underline{v'_{\mathrm{c}}}$ of the form $\underline{v_{\mathrm{c}}}'_1\, b\, \underline{v_{\mathrm{c}}}'_2\, a\, \underline{v_{\mathrm{c}}}'_3$, both of which are well-typed. This means that the compartment value a must include b as an ancestor, and b in turn must include a as an ancestor. Hence a must include itself as an ancestor. But this is impossible since compartment values are finite.
  Max one parent: by the induction hypothesis, each node in $G\{\underline{v_{\mathrm{c}_i}}\}$ has at most one parent. Also each node in $G\{\underline{v'_{\mathrm{c}}}\}$ has at most one parent, for otherwise the graph would contain a cycle. Suppose towards a contradiction that there is some node a in $G(\{\underline{v_{\mathrm{c}_i}}\} \cup \{\underline{v'_{\mathrm{c}}}\})$ with two parents. This can only arise from a branch in $G\{\underline{v_{\mathrm{c}_i}}\}$ of the form $\underline{v_{\mathrm{c}}}_1\, b\, a\, \underline{v_{\mathrm{c}}}_2$ and $\underline{v'_{\mathrm{c}}}$ of the form $\underline{v_{\mathrm{c}}}'_1\, c\, a\, \underline{v_{\mathrm{c}}}'_2$ with $b \neq c$. But this is impossible since both lists are well-typed and a can contain only a single parent.

Normal form of compartment value lists. Parent compartments in compartment values are necessary for type-checking in the general semantics, and volumes are necessary in algebraic rate expressions. But from the view of any concrete semantics, we are interested in a normal form of compartment lists in which only compartment names are retained and in which the nil compartments are removed. This *normal form function* is of the form $nf(\underline{v_{\mathrm{c}}}) = \underline{n_{\mathrm{c}}}$ and is defined as $nf(\underline{n_{\mathrm{c}}}, r, \underline{v_{\mathrm{c}}}) \stackrel{\Delta}{\simeq} \underline{n_{\mathrm{c}}}.I$ where $I \stackrel{\Delta}{\simeq} \{i \mid \underline{n_{\mathrm{c}}}.i \neq \mathbf{1}_{\mathrm{c}}\}$ if $\underline{v_{\mathrm{c}}}$ is a normal form compartment value list and is undefined otherwise. The graph arising from the normal form of a set of compartment value lists is also a forest if all non-nil compartment values have distinct compartment names, i.e. if the same non-nil compartment name does not occur with different parents or with different volumes. This is always the case for graphs in which compartment values arise from compartment expressions.

5.2 Species

Species values. Recall from the abstract syntax for species expressions that ξ is an annotation used to provide a match between actual and formal species parameters. Recall also that ρ ranges over modification site types, that n_{m} ranges over modification site names, and that n_{s} ranges over species names. *Species values* are generated by the following grammar where we use $Q \subset_{\mathrm{fin}} \mathbb{N}$ to range over sets of list indices:

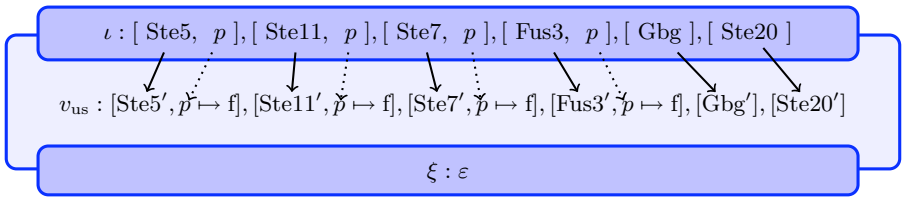
$$
\begin{array}{rll}
v_{\mathrm{s}} & ::= v_{\mathrm{us}}^{\iota:\xi} & \textsc{Species value} \\
v_{\mathrm{us}} & ::= \underline{\underline{v_{\mathrm{c}}}}[n_{\mathrm{s}}, \alpha_\sigma] & \textsc{Unboxed species value} \\
\alpha_\sigma & ::= \{n_{\mathrm{m}} \mapsto (\rho, e_{\mathrm{m}})\} & \textsc{Typed assignment} \\
\iota & ::= \{\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}] \mapsto (Q, \iota_{\mathrm{m}})\} & \textsc{Species interface} \\
\iota_{\mathrm{m}} & ::= \{n_{\mathrm{m}} \mapsto n'_{\mathrm{m}}\} & \textsc{Modification site interface}
\end{array}
$$

An *unboxed species value* represents a possibly complex species by a list of located atomic species, each of which is represented by a name and a *typed assignment* mapping modification site names to pairs of modification types and expressions. *Species values* add annotations and interfaces. Annotations are as in the abstract syntax for species expressions: they are used for selecting the located atomic species and modification sites in an actual species parameter which should be mapped from the corresponding atomic species and modification sites in a formal parameter. *Interfaces* capture this mapping from formals to actuals and can hence be viewed as a product of module invocation. The need for a local mapping from formals to actuals arises from our having nondeterministic species, where different members of the set of values denoted by a nondeterministic actual species parameter may require different mappings from formals to actuals as demonstrated in Listing 3.7.

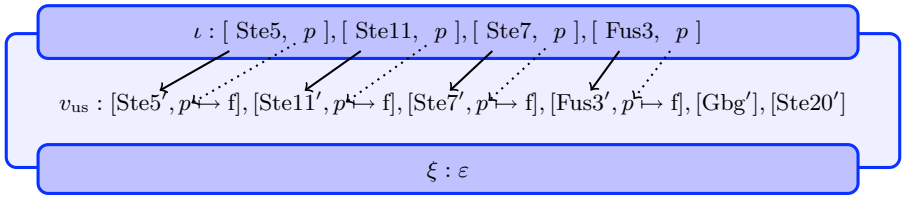
Interfaces map formal located names to pairs consisting of a set of position indices in the associated unboxed species values and a *modification site interface*. The sets of indices are used to cater for the general case of homo-multimers in which there are multiple instances of some atomic species. Modification site interfaces map formal modification site names to actual modification site names in the unboxed species value. Interfaces may expose only a subset of species indices in an unboxed species value, and for each exposed set of species indices, the associated modification site interface may expose only a subset of the modification sites recorded in the unboxed species value. Hence interfaces give rise to a notion of subtyping. For species values which have not been subjected to module invocations, the interface exposes all atomic species and all modification sites. Interfaces also give rise to a notion of parametric type since they provide means of renaming atomic species and modification site names.

Examples of species values. Examples of some species values arising from Listing 3.9 are shown informally in Figure 5.1 where we let f be the pair $(\mathbf{bool}, \mathbf{ff})$ and t be the pair $(\mathbf{bool}, \mathbf{tt})$. Interfaces are depicted in the top part of each figure, with solid lines representing the mapping from atomic species names to indices, and dotted lines representing the embedded mapping between modification site names. Note that none of these examples are homo-multimers, so interfaces map to singleton sets of indices. Unboxed species values are depicted in the center part of each figure, and annotations are depicted at the bottom.

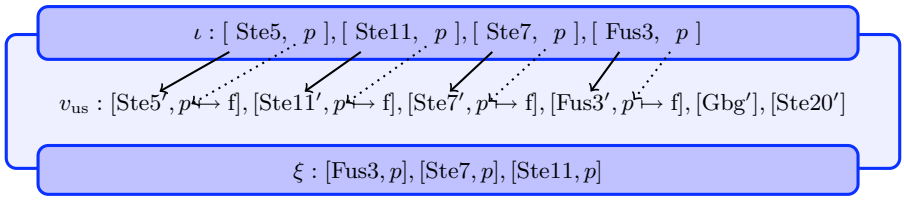
Figure 5.1a shows a complex species value before it has been subjected to any module invocation, hence all primitive species and their modification sites are exposed by the interface. The species names in the unboxed value are primed, indicating that these are fresh. Figure 5.1b shows the species value after it has



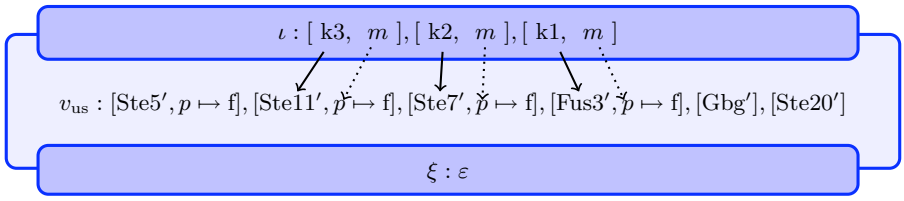
(a) The species value bound to e in the formation module, line 10.



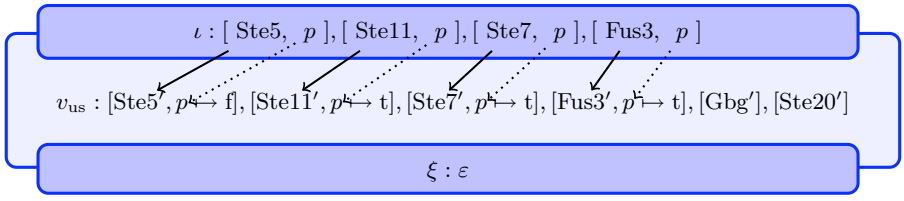
(b) The species value bound to the output species identifier link1 after invocation of the formation module in line 19.



(c) The species value resulting from evaluating the first actual parameter of the mapk module invocation in line 20.



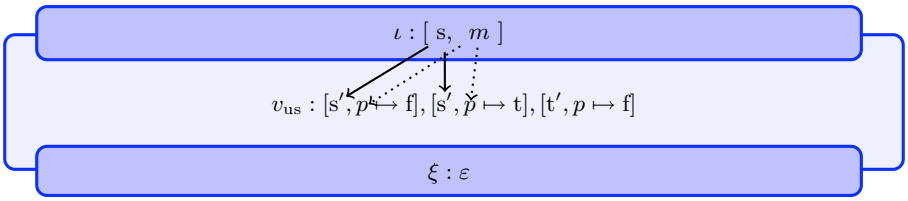
(d) The species value bound to a in the body of the mapk cascade module in line 14.



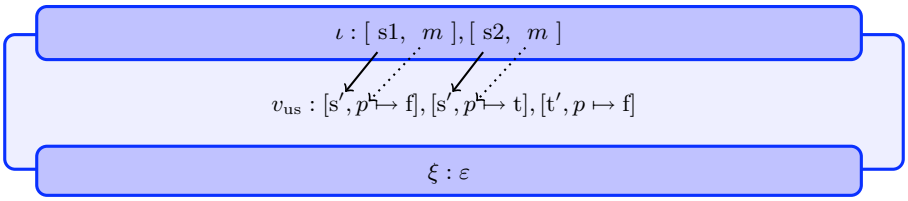
(e) The species value bound to the identifier link2, line 20, after invocation of the mapk module.

Fig. 5.1. Examples of species values from Listing 3.9 represented in an informal graphical notation

been output from the formation module where Gbg and Ste20 have been removed from the interface. In Figure 5.1c the annotation of the actual species parameter of the mapk module has been recorded in the species value. Figure 5.1d shows the species value after the interface has been updated based on the annotation in Figure 5.1c and the corresponding formal annotation, $\xi' : [\mathrm{k1}, m], [\mathrm{k2}, m], [\mathrm{k3}, m]$; together these provide a mapping from e.g. k1 to Fus3, which is traced through the interface in Figure 5.1c down to the fourth index of the unboxed species value. The annotation has now served its purpose and is discarded. Finally, figure 5.1d shows the species value where three atomic species have been phosphorylated, and following output from the mapk module, the interface of this species value has been restored to the interface of the original input species value in Figure 5.1d.



(a) A homomultimer species value with a single atomic species in its interface.



(b) A homomultimer species value with two atomic species in its interface, mapping to different occurrences of the same underlying fresh atomic species.

Fig. 5.2. Examples of homomultimer species values

A smaller example which illustrates how homomultimers can be represented is shown in Figure 5.2a; here the same atomic species name, s, maps to two occurrences of the same underlying fresh species name, s′. An interface may however also map different located names to indices with the same located fresh species names as shown in Figure 5.2b. This allows multiple instances of the same atomic species within a homo-multimer to be distinguished, a capability which a previous version of the language lacked.

Well-typedness of species values. A number of well-typedness conditions apply to species values. For an unboxed species value $\underline{\underline{v_c}}[n_s, \alpha_\sigma]$ we require that the lists of compartment values are well-typed and hence form a forest structure. We also require that assignments respect their associated type. These conditions can be phrased formally as follows:

1. $\{\underline{\underline{v_{\mathrm{c}}}}.i\}$ is a well-typed set of compartment value lists.
2. $\forall(\rho, e_{\mathrm{m}}) \in im(\alpha_\sigma).\ e_{\mathrm{m}} : \rho$

For a species value $v_{\mathrm{us}}^{\iota:\xi}$ we furthermore require that the interface maps to 1) non-empty and 2) disjoint sets of indices; that 3) all indices in a set exist in the unboxed species value and 4) contain species with identical located fresh names and modification site names; that 5) the modification site interfaces map to sites which exist in the assignments at the corresponding indices; that 6) the annotation only mentions located species and sites which exist in the interface. These conditions can be summarised formally as follows, where $\underline{\underline{v_{\mathrm{c}}}}[n_{\mathrm{s}}, \alpha_\sigma] = v_{\mathrm{us}}$, $\underline{\underline{id_{\mathrm{c}}}}[n_{\mathrm{s}}, \underline{n_{\mathrm{m}}}] = \xi$ and, for $\alpha_\sigma = \{n_{\mathrm{m}} \mapsto (\rho, e_{\mathrm{m}})\}$, we let $\sigma(\alpha_\sigma) \stackrel{\Delta}{\simeq} \{n_{\mathrm{m}} \mapsto \rho\}$.

1. $\forall(Q, \iota_{\mathrm{m}}) \in im(\iota).\ |Q| > 0$
2. $\forall l, l' \in dom(\iota).\ l \neq l' \Rightarrow ind(\iota(l)) \cap ind(\iota(l')) = \emptyset$ where $ind(Q, \iota_{\mathrm{m}}) \stackrel{\Delta}{\simeq} Q$
3. $\forall(Q, \iota_{\mathrm{m}}) \in im(\iota).\ Q \subseteq_{\mathrm{fin}} \{1, \ldots, |v_{\mathrm{us}}|\}$
4. $\forall(Q, \iota_{\mathrm{m}}) \in im(\iota).\ \forall q, q' \in Q.\ (\underline{\underline{v_{\mathrm{c}}}}[n_{\mathrm{s}}].q = \underline{\underline{v_{\mathrm{c}}}}[n_{\mathrm{s}}].q') \wedge (\sigma(\underline{\alpha_\sigma}.q) = (\sigma(\underline{\alpha_\sigma}.q'))$
5. $\forall(Q, \iota_{\mathrm{m}}) \in im(\iota).\ \forall q \in Q.\ im(\iota_{\mathrm{m}}) \subseteq_{\mathrm{fin}} dom(\underline{\alpha_\sigma}.q)$
6. $\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}] \in dom(\iota) \wedge \forall(Q, \iota_{\mathrm{m}}).\ (Q, \iota_{\mathrm{m}}) = \iota(\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}]) \Rightarrow \{\underline{n_{\mathrm{m}}}.i\} \subseteq_{\mathrm{fin}} dom(\iota_{\mathrm{m}})$

The denotation function. We now turn to the semantics for species expressions. A *species environment* is a partial finite function of the form $\Gamma_{\mathrm{s}}(id_{\mathrm{s}}) = \underset{\sim}{v_{\mathrm{s}}}$ mapping species identifiers to lists of species values. The denotation function for species expressions is of the form:

$$[\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}} = \underset{\sim}{v_{\mathrm{s}}}$$

and is parametric on compartment and species environments. The denotation of a species is a *list* of species values. More than one species value may arise because of nondeterminism, and we use lists rather than sets to cater for output species in module parameters, as will be apparent in the semantics for programs; however, for most purposes we may think of these lists as sets, hence the wavy underline notation.

The definition of the denotation function for species expressions is given in the following. In order to simplify notation, we write Γ instead of $\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$ for cases where the environments are not used. Let us also reiterate the subtle notational convention that given e.g. a list $\underline{v_{\mathrm{us}}}$ we write v_{us} for $\underline{v_{\mathrm{us}}}.i$, and that i is implicitly assumed to be universally quantified over the indices of $\underline{v_{\mathrm{us}}}$ in definitions; see for example the last 3 lines of the first case below.

– $[\![id_{\mathrm{c}}[e_{\mathrm{s}}]]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}} \stackrel{\Delta}{\simeq} \underset{\sim}{v_{\mathrm{us}}^{\iota:\xi}}$ where
  - $v_{\mathrm{c}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}(id_{\mathrm{c}})$
  - $\underset{\sim}{v_{\mathrm{us}1}^{\iota_1:\xi_1}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$

- $v_{\mathrm{us}} \stackrel{\Delta}{\simeq}_t \underline{v_{\mathrm{c}} \underline{v_{\mathrm{c}_1}}[n_{\mathrm{s}}, \alpha_\sigma]}$ where $\underline{\underline{v_{\mathrm{c}_1}}[n_{\mathrm{s}}, \alpha_\sigma]} \stackrel{\Delta}{\simeq} v_{\mathrm{us}1}$
- $\iota \stackrel{\Delta}{\simeq} \{id_{\mathrm{c}} \underline{id_{\mathrm{c}_1}}[n_{\mathrm{s}}] \mapsto \iota_1(\underline{id_{\mathrm{c}_1}}[n_{\mathrm{s}}]) \mid \underline{id_{\mathrm{c}_1}}[n_{\mathrm{s}}] \in dom(\iota_1)\}$
- $\xi \stackrel{\Delta}{\simeq} \underline{id_{\mathrm{c}} \underline{id_{\mathrm{c}_1}}[n_{\mathrm{s}}, \underline{n_{\mathrm{m}}}]}$ where $\underline{\underline{id_{\mathrm{c}_1}}[n_{\mathrm{s}}, \underline{n_{\mathrm{m}}}]} \stackrel{\Delta}{\simeq} \xi_1$

– $[\![e_{\mathrm{s}1} - e_{\mathrm{s}2}]\!]_{\mathrm{s}} \Gamma \stackrel{\Delta}{\simeq}_t \underset{\sim}{v_{\mathrm{s}1}} \times_\circ \underset{\sim}{v_{\mathrm{s}2}}$ where

- $\underset{\sim}{v_{\mathrm{s}1}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}1}]\!]_{\mathrm{s}} \Gamma$
- $\underset{\sim}{v_{\mathrm{s}2}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}2}]\!]_{\mathrm{s}} \Gamma$
- ${v_{\mathrm{us}1}}^{\iota_1:\xi_1} \circ {v_{\mathrm{us}2}}^{\iota_2:\xi_2} \stackrel{\Delta}{\simeq} v_{\mathrm{us}}^{\iota:\xi}$ where
  - $v_{\mathrm{us}} \stackrel{\Delta}{\simeq} v_{\mathrm{us}1} v_{\mathrm{us}2}$
  - $\iota(l) \stackrel{\Delta}{\simeq} \begin{cases} \iota_1(l) & \text{if } l \in dom(\iota_1) \setminus dom(\iota_2) \\ (A(Q_2), \iota_{\mathrm{m}2}) & \text{if } l \in dom(\iota_2) \setminus dom(\iota_1) \wedge \\ & (Q_2, \iota_{\mathrm{m}2}) = \iota_2(l) \\ (Q_1 \cup A(Q_2), \iota_{\mathrm{m}}) & \text{if } l \in dom(\iota_1) \cap dom(\iota_2) \wedge \\ & (Q_1, \iota_{\mathrm{m}1}) = \iota_1(l) \wedge (Q_2, \iota_{\mathrm{m}2}) = \iota_2(l) \wedge \\ & \iota_{\mathrm{m}} = \iota_{\mathrm{m}1} = \iota_{\mathrm{m}2} \end{cases}$
    where $A(Q) = \{q + |v_{\mathrm{us}1}| \mid q \in Q\}$
  - $\underline{\xi} \stackrel{\Delta}{\simeq} \underline{\xi_1 \xi_2}$

– $[\![e_{\mathrm{s}}.\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}]]\!]_{\mathrm{s}} \Gamma \stackrel{\Delta}{\simeq} \underset{\sim}{v_{\mathrm{us}}^{\iota:\xi}}$ where

- $\underset{\sim}{{v_{\mathrm{us}1}}^{\iota_1:\xi_1}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}} \Gamma$
- $(Q, \iota_{\mathrm{m}}) \stackrel{\Delta}{\simeq} \iota_1(\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}])$
- $v_{\mathrm{us}} \stackrel{\Delta}{\simeq} v_{\mathrm{us}1}.Q$
- $\iota \stackrel{\Delta}{\simeq} \{\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}] \mapsto (\{1 \ldots |Q|\}, \iota_{\mathrm{m}})\}$
- $\xi \stackrel{\Delta}{\simeq} \xi_1.\{q \mid \exists \underline{n_{\mathrm{m}}}.\ \xi_1.q = (\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}, \underline{n_{\mathrm{m}}}])\}$

– $[\![e_{\mathrm{s}} \backslash \underline{id_{\mathrm{c}}}[n_{\mathrm{s}}]]\!]_{\mathrm{s}} \Gamma \stackrel{\Delta}{\simeq} \underset{\sim}{v_{\mathrm{us}}^{\iota:\xi}}$ where

- $\underset{\sim}{{v_{\mathrm{us}1}}^{\iota_1:\xi_1}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}} \Gamma$
- $(Q, \iota_{\mathrm{m}}) \stackrel{\Delta}{\simeq} \iota_1(\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}])$
- $v_{\mathrm{us}} \stackrel{\Delta}{\simeq} v_{\mathrm{us}1}.(\{1 \ldots |v_{\mathrm{us}1}|\} \setminus Q)$
- $\iota \stackrel{\Delta}{\simeq} \{l \mapsto (A(Q'), \iota_{\mathrm{m}}) \mid l \in dom(\iota_1) \setminus \{\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}]\} \wedge (Q', \iota_{\mathrm{m}}) = \iota_1(l)\}$
  where
  - $A(Q') \stackrel{\Delta}{\simeq} \{q' - |\{q \in Q \mid q \leq q'\}| \mid q' \in Q'\}$
- $\xi \stackrel{\Delta}{\simeq} \xi_1.\{q \mid \neg\exists \underline{n_{\mathrm{m}}}.\ \xi_1.q = (\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}, \underline{n_{\mathrm{m}}}])\}$

– $[\![e_{\mathrm{s}}\langle \underline{id_{\mathrm{c}}}[n_{\mathrm{s}}, \alpha]\rangle]\!]_{\mathrm{s}}\Gamma \stackrel{\Delta}{\simeq}_t \underbrace{\underline{\underline{v_{\mathrm{c}}}}[n'_{\mathrm{s}}, {\alpha_\sigma}'']^{\iota:\xi}}$ where

- $\underbrace{\underline{\underline{v_{\mathrm{c}}}}[n'_{\mathrm{s}}, {\alpha_\sigma}']^{\iota:\xi}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma$
- $(Q, \iota_{\mathrm{m}}) \stackrel{\Delta}{\simeq} \iota(\underline{id_{\mathrm{c}}}[n_{\mathrm{s}}])$
- $\underline{{\alpha_\sigma}''}.q \stackrel{\Delta}{\simeq} \begin{cases} \underline{{\alpha_\sigma}'}.q\langle \alpha \circ ({\iota_{\mathrm{m}}}^{-1})\rangle & \text{if } q \in Q \\ \underline{{\alpha_\sigma}'}.q & \text{otherwise} \end{cases}$

  where
  - ${\alpha_\sigma}'\langle \alpha'\rangle(n_{\mathrm{m}}) \stackrel{\Delta}{\simeq} \begin{cases} {\alpha_\sigma}'(n_{\mathrm{m}})\langle \alpha'(n_{\mathrm{m}})\rangle & \text{if } n_{\mathrm{m}} \in dom({\alpha_\sigma}') \cap dom(\alpha') \\ {\alpha_\sigma}'(n_{\mathrm{m}}) & \text{if } n_{\mathrm{m}} \in dom({\alpha_\sigma}') \setminus dom(\alpha') \end{cases}$

    for all ${\alpha_\sigma}'$, α' and n_{m}.

– $[\![e_{\mathrm{s}1}\ \mathbf{or}\ e_{\mathrm{s}2}]\!]_{\mathrm{s}}\Gamma \stackrel{\Delta}{\simeq} \underbrace{v_{\mathrm{s}1}}\ \underbrace{v_{\mathrm{s}2}}$ where

- $\underbrace{v_{\mathrm{s}1}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}1}]\!]_{\mathrm{s}}\Gamma$
- $\underbrace{v_{\mathrm{s}2}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}2}]\!]_{\mathrm{s}}\Gamma$

– $[\![e_{\mathrm{s}} : \xi]\!]_{\mathrm{s}}\Gamma \stackrel{\Delta}{\simeq}_t \underbrace{v_{\mathrm{us}}^{\iota:\xi}}$ where

- $\underbrace{v_{\mathrm{us}}^{\iota:\xi_1}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma$

– $[\![id_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{s}}(id_{\mathrm{s}})$

– $[\![\mathbf{0}_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma \stackrel{\Delta}{\simeq} \underbrace{\varepsilon}$

The denotation function is partial because some species expressions do not result in lists of well-typed species values or in environments which are functions, or because some operations are undefined for some of the intermediate objects which arise. Given suitable environments, we say that a species expression is *well-typed* if its denotation is defined.

Explanation of the denotation function. In the case of located species, the compartment value assigned to the compartment identifier is looked up in the compartment environment and the species value denoting the nested species expression is obtained recursively. The denotation of the located species expression is obtained from this species value by adding the compartment value to the left of every located atomic species in the unboxed species value, by adding the compartment identifier to the left of each list of compartment identifiers in the domain of the interface, and by likewise adding the compartment identifier to each list of compartments in the annotation. Note that the interface records the compartment identifier rather than the compartment value. The expression is well-typed when the compartment identifier is defined in the given environment and when the resulting sets of compartment value lists are well-typed.

The denotation of a composite species expression is given by a Cartesian product of the denotations of the two operands. The corresponding pairing operation on species values concatenates the two unboxed species values and composes the interfaces in a manner that reflects this concatenation. The composed interface essentially maps located names to the union of indices given by the individual interfaces, with the twist that indices from the second interface are increased by the length of the first unboxed species value. This adjustment of indices is handled by an auxiliary function A. Annotations are composed simply by list concatenation. The resulting species value is well-typed when the two components agree on atomic species names and modification site interfaces for any common members of their interface.

In the case of species selection, the resulting unboxed species value is obtained by selecting the indices determined by the interface of the target species on the given located name. The resulting interface maps the given located name to a set of consecutive indices together with the original modification site interface. The resulting annotation is obtained by selecting for those entries which contain the given located name. The expression is well-typed when the located name is in the domain of the interface, which is necessary for the list selection operations to be well-defined. The case of species removal follows a similar idea, although special care is needed for the appropriate adjustment of indices using an auxiliary function A.

In the case of species update, the interface and annotation of the denotation of the operand are preserved, and an updated unboxed species is obtained by updating the assignments at the indices with the given located name. The modification site names in the given update expression are first renamed by function composition with the inverse of the modification site interface, and the result is given as an argument to an update operation on typed assignments which is defined as an auxiliary function. Here, the assignment to modification sites which are not mentioned in the update are preserved. For sites which are both in the original assignment and in the update, the expression update function (which is a parameter of the general semantics) is used; the update function is assumed extended to pairs of modification site types and expressions. The expression is well-typed if the located species name is in the domain of the target species interface, if the domain of the update is in the domain of the relevant modification site interface, and if the update respects the relevant species types.

The remaining cases are simpler. For nondeterministic species expressions, the species value lists obtained from the denotations of the operands are simply concatenated. The species annotation expression replaces the annotation in the denotation of the nested species expression with a new annotation. For this to be well-typed, the new annotation must mention only located names and sites which exist in the domain of the interface of the operand, thus ensuring that condition 6 of well-typedness for species values is satisfied. In the case of species identifier expressions, the corresponding value is simply looked up in the species environment which must be defined for the given identifier to be well-typed. Finally, the nil species evaluates to a singleton list containing just the empty species value.

Extended species expressions. The denotation function for extended species expressions is of the form:

$$[\![e_{\mathrm{s}+}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, b = \underset{\sim}{v_{\mathrm{s}}}$$

It is parametric on compartment and species environments, and also on a binary string b used to create fresh names for new species. Here is the definition:

- $[\![e_{\mathrm{s}}]\!]_{\mathrm{s}+}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, b \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$
- $[\![\mathbf{new}\ n_{\mathrm{s}}, \sigma]\!]_{\mathrm{s}+}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, b \stackrel{\Delta}{\simeq} \underset{\sim}{\varepsilon[b, \alpha_\sigma]^{\iota:\varepsilon}}$ where
  - $\alpha_\sigma \stackrel{\Delta}{\simeq} \{n_{\mathrm{m}} \mapsto (\rho, \mathit{default}(\rho)) \mid n_{\mathrm{m}} \in \mathit{dom}(\sigma) \wedge \rho = \sigma(n_{\mathrm{m}})\}$
  - $\iota \stackrel{\Delta}{\simeq} \{\varepsilon[n_{\mathrm{s}}] \mapsto (\{1\}, \{n_{\mathrm{m}i} \mapsto n_{\mathrm{m}i}\})$ where $\{n_{\mathrm{m}i} \mapsto \rho_i\} \stackrel{\Delta}{\simeq} \sigma$

A new species expression evaluates to a singleton unboxed species with a fresh species name given by the binary string parameter to the denotation function, together with a typed assignment which extends the given type with default modification expressions. The interface simply maps the given species name in the empty list of compartment identifiers to the first and only index of the unboxed species value, together with the identity interface on modification site names.

Normal form species values. Interfaces, annotations and parent definitions in compartment values are needed for determining well-typedness and for making module invocation work, as detailed in the next subsection, but they are not needed for normal form reactions. All that is needed here is a *normal form*:

$$v_{\mathrm{ns}} ::= \underline{\underline{n_{\mathrm{c}}}[n_{\mathrm{s}}, \alpha_\sigma]}$$

of species values. The *normal form function* then takes the form $\mathit{nf}(v_{\mathrm{s}}, \underline{v_{\mathrm{c}}}) = v_{\mathrm{ns}}$ where $\underline{v_{\mathrm{c}}}$ is a list of compartment values which is required to go all the way from the world compartment down to the compartment enclosing the species value. The function is used in the semantics for programs and is applied to species values in a located reaction. In the following definition we use the normal form function for compartment value lists defined in the previous subsection:

$$\mathit{nf}(\underline{\underline{v'_{\mathrm{c}}}[n_{\mathrm{s}}, \alpha_\sigma]}^{\iota,\xi}, \underline{v_{\mathrm{c}}}) \stackrel{\Delta}{\simeq} \underline{\mathit{nf}(\underline{v_{\mathrm{c}}}\ \underline{v'_{\mathrm{c}}})[n_{\mathrm{s}}, \alpha_\sigma]}$$

The function is defined only when $\underline{v_{\mathrm{c}}}\ \underline{v'_{\mathrm{c}}}$ is a well-typed list of compartment values. Although retaining the full list of compartment names is unnecessary for some concrete semantics such as Petri nets, this information may be relevant in other cases. For example, it allows the compartment forest structure of programs to be obtained through the general semantical framework by defining an appropriate concrete semantics for representing and composing forests.

Ground normal form species values. We introduce one further *ground normal form* of species values:

$$v_{\mathrm{gns}} ::= \underline{\underline{n_{\mathrm{c}}}}[n_{\mathrm{s}}, \{n_{\mathrm{m}} \mapsto (\rho, v_{\mathrm{m}})\}]$$

Here modification sites map to pairs of modification types and values, rather than to pairs of modification types and expressions with variables. Ground normal form species values are used in the semantics for programs in the case of initial conditions; indeed, as described above, the general semantics is parameterised on a function of the form $I_S(v_{\mathrm{gns}}, r) = \mathcal{O}$ for assigning semantical objects to initial populations or concentrations of ground normal form species values. Recall also that the general semantics is parameterised on a function of the form $[\![e_{\mathrm{m}}]\!]_{\mathrm{m}} \Gamma_{\mathrm{x}} = v_{\mathrm{m}}$ which assigns values to modification site expressions given a variable environment. This function can be extended in an evident manner to a function from normal form species values to ground normal form species values; this is needed when defining concrete semantics.

Further functions on species values. We define two functions on species values which are required for the semantics of module invocations. For the first function, the intuition is that one can update an interface ι_1 given the associated annotation ξ_1 together with a second matching annotation ξ_2 obtained from a corresponding formal species parameter. The updated interface is then used to access the species within the body of the module. An example of this is shown in the transition from the species value in Figure 5.1c to that in Figure 5.1d. Formally, the function takes the form $close(v_{\mathrm{s}1}, \xi_2) = v_{\mathrm{s}2}$ and is defined as follows:

$$close(v_{\mathrm{us}}^{\iota_1 : \xi_1}, \xi_2) \stackrel{\Delta}{\simeq} v_{\mathrm{us}}^{\iota_2 : \varepsilon}$$

where, for $\underline{\underline{id_{\mathrm{c}}}}[n_{\mathrm{s}}, \underline{n_{\mathrm{m}}}]_1 \stackrel{\Delta}{\simeq} \xi_1$ and $\underline{\underline{id_{\mathrm{c}}}}[n_{\mathrm{s}}, \underline{n_{\mathrm{m}}}]_2 \stackrel{\Delta}{\simeq} \xi_2$,

$$\iota_2 \stackrel{\Delta}{\simeq} \{\underline{\underline{id_{\mathrm{c}}}}[n_{\mathrm{s}}]_2.i \mapsto (Q, \{\underline{\underline{n_{\mathrm{m}2}}}.i.j \mapsto \iota_{\mathrm{m}}(\underline{\underline{n_{\mathrm{m}1}}}.i.j)\}) \mid (Q, \iota_{\mathrm{m}}) = \iota_1(\underline{\underline{id_{\mathrm{c}}}}[n_{\mathrm{s}}]_1.i)\}$$

The function is only defined if the lists ξ and ξ' have the same length, and if all of the embedded lists $\underline{\underline{n_{\mathrm{m}1}}}.i$ and $\underline{\underline{n_{\mathrm{m}2}}}.i$ also have the same length.

The second function enables one species value to take on the interface and annotation of another species value. This is needed in the semantics for output species in programs, and an example is shown in the transition of the species value in Figure 5.1a to that in Figure 5.1b, and from the species value in Figure 5.1d to that in Figure 5.1e. Formally, we first need two supporting functions. The first gives the located name of an unboxed species value at a given index, and the second counts the number of previous occurrences of the located species name at a given index:

$$l^q(v_{\mathrm{us}}) \stackrel{\Delta}{\simeq} \underline{\underline{id_{\mathrm{c}}}}[n_{\mathrm{s}}] \text{ where } \exists \alpha_\sigma.\ \underline{\underline{id_{\mathrm{c}}}}[n_{\mathrm{s}}, \alpha_\sigma] = v_{\mathrm{us}}.q$$

$$c^q(v_{\mathrm{us}}) \stackrel{\Delta}{\simeq} |\{q' \mid l^q(v_{\mathrm{us}}) = l^{q'}(v_{\mathrm{us}}) \wedge q' < q\}|$$

The function of interest then takes the form $adapt(v_{\mathrm{s}1}, v_{\mathrm{s}2}) = v_{\mathrm{s}3}$ and is defined as follows:

$$adapt({v_{\mathrm{us}1}}^{\iota_1:\xi_1}, {v_{\mathrm{us}2}}^{\iota_2:\xi_2}) \stackrel{\Delta}{\simeq}_t {v_{\mathrm{us}1}}^{\iota_3:\xi_2}$$

where, for all $l \in dom(\iota_2)$:

$$\begin{aligned}(Q_2, \iota_{\mathrm{m}2}) &\stackrel{\Delta}{\simeq} \iota_2(l)\\ Q_3 &\stackrel{\Delta}{\simeq} \{q \mid \exists q_2 \in Q_2.\ l^q(v_{\mathrm{us}1}) = l^{q_2}(v_{\mathrm{us}2}) \wedge c^q(v_{\mathrm{us}1}) = c^{q_2}(v_{\mathrm{us}2})\}\\ \iota_3(l) &\stackrel{\Delta}{\simeq} (Q_3, \iota_{\mathrm{m}2})\end{aligned}$$

Here the new interface ι_3 maps a located species name l to the indices in $v_{\mathrm{us}1}$ which have the same located fresh name l' as the indices in $v_{\mathrm{us}2}$ mapped from l by ι_2. If there are multiple choices of such indices, the index which results in the same number of previous occurrences of the fresh name l' in the two unboxed species values is chosen. The function is defined only when the resulting species value is well-typed, which is the case whenever the resulting sets of indices are non-empty and modification site interfaces map to sites which exist in the resulting unboxed species value.

Species value design choices. We end the treatment of species values with some remarks about possible alternative representations. First note that we choose to include modification types in species values. However, since new species values are always created with fresh names and there are no expressions which allow modification sites to change type, species values with identical names also have identical modification types. Hence it would also be possible to maintain modification types separately from species values as indeed was done in a previous version of the language [27], with the benefit of reduced redundancy. But this approach would have the downside of cluttering the presentation of the semantics with an additional environment needing to be maintained.

Alternative representations of species interfaces are also possible. We choose for example to include compartment identifiers in the renaming of interfaces, which allows compartments to differ between different members of a nondeterministic species in the same way that atomic species may differ. But interfaces could instead provide local mappings for only species and modification site names, and require compartments to be evaluated externally. This would ensure that two species with the same location in their interface are indeed in the same location. Such guarantees cannot be made when location is included in the interface.

Another design choice involves the relatively relaxed conditions on interfaces. For example, an interface may map a located name to indices in which the compartment structure is completely different, and different located names may map to indices with the same species names. The latter allows the elements of a homomultimer to be distinguished within the same species as demonstrated by the earlier example in Figure 5.2b. This is the reason why unboxed species values are

lists rather than multisets. We also anticipate that the ordering of atomic species within a complex may be significant for future concrete semantics, although the Petri net, ODE and CTMC concrete semantics disregard the ordering.

Annotations are maintained explicitly in species values. This incurs some overhead in the semantics for species expressions since all operations must take annotations into account. Alternatively, the interface could be represented by lists and the annotation could be captured by an appropriate ordering and restriction of the interface. This would give a more compact semantics at the cost of reduced transparency. It is however impossible for another reason pertaining to output species: these can adopt the interface of actual species values at time of module invocation using the *adapt* function, so interfaces of actual species parameters must be preserved.

5.3 Programs

Normal form reactions. Recall that the general semantics is parameterised on a structure $(S, |_S, \mathbf{0}_S, R_S, I_S)$ and that R_S is a function of the form $R_S(R, b) = \mathcal{O}$ assigning a semantical object to a reaction R, named b, in a suitable normal form. More precisely, R takes the form:

$$\underline{n \cdot v_{\mathrm{ns}}} \Rightarrow^{v_{\mathrm{r}}} \underline{n' \cdot v'_{\mathrm{ns}}} \textbf{ if } e_{\mathrm{b}}$$

where v_{ns} and v'_{ns} are normal form species values as defined in the semantics for species and v_{r} is a *rate value*, i.e. a rate expression in which species expressions have been evaluated to their normal forms, compartment expressions have been replaced by their resulting volumes, and rate function invocations have been evaluated. Rate values and algebraic rate values are generated by the grammar in Table 5.3.

The denotation function for algebraic rate expressions is of the form:

$$[\![e_{\mathrm{a}}]\!]_{\mathrm{a}} \Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \underline{v_{\mathrm{c}}} = v_{\mathrm{a}}$$

Here Γ_{a} is an *algebraic rate function environment* of the form $\Gamma_{\mathrm{a}}(id_{\mathrm{a}}) = f$ where f in turn is a function of the form $f(\underline{v_{\mathrm{c}}}, \underline{\underset{\sim}{v_{\mathrm{s}}}}, \underline{v_{\mathrm{a}}}, \underline{v'_{\mathrm{c}}}) = v_{\mathrm{a}}$ mapping actual parameters, together with a list $\underline{v'_{\mathrm{c}}}$ of parent compartment values at time of invocation, to algebraic rate values. The denotation function is defined below, but with some standard cases for functions and arithmetic operators omitted. We adopt a convention here and throughout where any parameters of a denotational function that are not explicitly used by a given case are represented by Γ; in the second case below, for example, Γ hence represents the parameters Γ_{s}, Γ_{a} and $\underline{v_{\mathrm{c}}}$.

- $[\![r]\!]_{\mathrm{a}} \Gamma \stackrel{\Delta}{\simeq} r$
- $[\![id_{\mathrm{c}}]\!]_{\mathrm{a}} \Gamma, \Gamma_{\mathrm{c}} \stackrel{\Delta}{\simeq} r$ where
  - $(n_{\mathrm{c}}, r, v_{\mathrm{c}}) \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}(id_{\mathrm{c}})$

Table 5.3. The abstract syntax for rate values

$v_{\mathrm{r}} ::=$	Rate value
$\mid \{v_{\mathrm{a}}\}$	Mass-action rate
$\mid [v_{\mathrm{a}}]$	Algebraic rate
$v_{\mathrm{a}} ::=$	Algebraic rate value
$\mid r$	Constant
$\mid v_{\mathrm{ns}}$	Population
$\mid$ **if** e_{b} **then** v_{a} **else** v'_{a}	Conditional
$\mid exp(v_{\mathrm{a}}) \mid log(v_{\mathrm{a}}) \mid sin(v_{\mathrm{a}}) \mid cos(v_{\mathrm{a}})$	Functions
$\mid v_{\mathrm{a}} + v'_{\mathrm{a}} \mid v_{\mathrm{a}}$ - v'_{a}	Arithmetic operators
$\mid v_{\mathrm{a}} \times v'_{\mathrm{a}} \mid v_{\mathrm{a}} / v'_{\mathrm{a}} \mid v_{\mathrm{a}} \hat{} v'_{\mathrm{a}}$	

– $[\![e_{\mathrm{s}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \underline{v_{\mathrm{c}}} \stackrel{\Delta}{\simeq} nf(\underset{\sim}{v_{\mathrm{s}}}.1, \underline{v_{\mathrm{c}}})$ where
  - $\underset{\sim}{v_{\mathrm{s}}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$

  if $|\underset{\sim}{v_{\mathrm{s}}}| = 1$

– $[\![id_{\mathrm{a}}(\underline{id_{\mathrm{c}}}; \underline{e_{\mathrm{s}}}; \underline{e_{\mathrm{a}}})]\!]_{\mathrm{a}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \underline{v'_{\mathrm{c}}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{a}}(id_{\mathrm{a}})(\underline{v_{\mathrm{c}}}, \underline{\underset{\sim}{v_{\mathrm{s}}}}, \underline{v_{\mathrm{a}}}, \underline{v'_{\mathrm{c}}})$ where
  - $v_{\mathrm{c}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}(id_{\mathrm{c}})$
  - $\underset{\sim}{v_{\mathrm{s}}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$
  - $v_{\mathrm{a}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \underline{v'_{\mathrm{c}}}$

  if $|\underset{\sim}{v_{\mathrm{s}}}| = 1$

– $[\![\textbf{if } e_{\mathrm{b}} \textbf{ then } e_{\mathrm{a}} \textbf{ else } e'_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma \stackrel{\Delta}{\simeq} \textbf{if } e_{\mathrm{b}} \textbf{ then } [\![e_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma \textbf{ else } [\![e'_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma$

– $[\![exp(e_{\mathrm{a}})]\!]_{\mathrm{a}}\Gamma \stackrel{\Delta}{\simeq} exp([\![e_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma)$

– $[\![e_{\mathrm{a}} + e'_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma \stackrel{\Delta}{\simeq} [\![e_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma + [\![e'_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma$

The case of compartments is only defined for non-world compartment expressions, reflecting our convention that the world compartment should only be used as a parent in definitions of new compartments. The case of species is only defined when the species expression does not contain nondeterminism because any nondeterministic choice is forced in the appropriate derived forms of reactions. In the case of algebraic rate function invocation, the semantic function is looked up in the algebraic rate function environment and applied to the actual parameters after these have been evaluated.

The remaining cases simply evaluate components recursively. Note in particular that conditionals are preserved in rate values, since a full evaluation requires

an assignment to variables. As for normal form species expressions, this assignment is left as a concern for the concrete semantics because certain semantical objects, such as coloured Petri nets, have their own distinct way of handing variables.

The denotation function for basic programs. The denotation function for basic programs is of the form:

$$[\![P]\!]_{\mathrm{p}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b, \underline{v_{\mathrm{c}}} = \{(\mathcal{O}_i, \Gamma_{\mathrm{so}i})\}$$

Here Γ_{m} is a *module environment* of the form $\Gamma_{\mathrm{m}}(id_{\mathrm{p}}) = g$ where g in turn is a function of the form $g(\underline{v_{\mathrm{c}}}, \underset{\sim}{v_{\mathrm{s}}}, \underline{v_{\mathrm{a}}}, \underline{id_{\mathrm{s}}}, b, \underline{v_{\mathrm{c}}}) = \{(\mathcal{O}_i, \Gamma_{\mathrm{so}i})\}$ mapping actual parameters to a set of pairs of semantical objects and *output species environments* $\Gamma_{\mathrm{so}i}$ which take the same form as species environments. Note that we obtain a *set* of semantical objects and output species environments in order to account for variation composition. The output species environments allows the formal output species, defined inside a module, to become available in the program following module invocation where they are bound to the corresponding actual output species identifiers. Note also that g is parameterised on a fresh name b and a list $\underline{v_{\mathrm{c}}}$ of parent compartments. The latter is because parent compartments for a module are determined dynamically rather than statically.

The denotation function is defined below and relies on the function

$$\delta_n^m \stackrel{\Delta}{\simeq} \{0\}^{n-1}\,1\,\{0\}^{m-n}$$

for constructing a binary string of length m with zeros everywhere except in the nth position which holds a one.

- $[\![\underline{n \cdot e_{\mathrm{s}}} \Rightarrow^{e_{\mathrm{r}}} \underline{n' \cdot e'_{\mathrm{s}}} \textbf{ if } e_{\mathrm{b}}]\!]_{\mathrm{p}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b, \underline{v_{\mathrm{c}}} \stackrel{\Delta}{\simeq} \{(\mathcal{O}, \emptyset)\}$ where
  - $v_{\mathrm{r}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{r}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \underline{v_{\mathrm{c}}}$
  - $\underset{\sim}{v_{\mathrm{s}}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$
  - $\underset{\sim}{v'_{\mathrm{s}}} \stackrel{\Delta}{\simeq} [\![e'_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$
  - $v_{\mathrm{ns}} \stackrel{\Delta}{\simeq} nf(\underset{\sim}{v_{\mathrm{s}}}.1, \underline{v_{\mathrm{c}}})$
  - $v'_{\mathrm{ns}} \stackrel{\Delta}{\simeq} nf(\underset{\sim}{v'_{\mathrm{s}}}.1, \underline{v_{\mathrm{c}}})$
  - $\mathcal{O} \stackrel{\Delta}{\simeq} R_S(\underline{n \cdot v_{\mathrm{ns}}} \Rightarrow^{v_{\mathrm{r}}} \underline{n' \cdot v'_{\mathrm{ns}}} \textbf{ if } e_{\mathrm{b}}, b)$

  if $|\underset{\sim}{v_{\mathrm{s}}}| = |\underset{\sim}{v'_{\mathrm{s}}}| = 1$
- $[\![\mathbf{0}_{\mathrm{p}}]\!]_{\mathrm{p}}\Gamma \stackrel{\Delta}{\simeq} \{(\mathbf{0}_S, \emptyset)\}$
- $[\![P \mid P']\!]_{\mathrm{p}}\Gamma, b \stackrel{\Delta}{\simeq} \{(\mathcal{O}_i|_S\mathcal{O}'_j, \Gamma_{\mathrm{so}i}\langle\Gamma'_{\mathrm{so}j}\rangle)\}$ where
  - $\{(\mathcal{O}_i, \Gamma_{\mathrm{so}i})\} \stackrel{\Delta}{\simeq} [\![P]\!]_{\mathrm{p}}\Gamma, 0b$
  - $\{(\mathcal{O}'_j, \Gamma'_{\mathrm{so}j})\} \stackrel{\Delta}{\simeq} [\![P']\!]_{\mathrm{p}}\Gamma, 1b$

– $[\![P \mid\mid P']\!]_{\mathrm{p}}\Gamma, b \stackrel{\Delta}{\simeq} \{(\mathcal{O}_i, \Gamma_{\mathrm{so}i})\} \cup \{ (\mathcal{O}'_j, \Gamma'_{\mathrm{so}j})\}$ where
  - $\{(\mathcal{O}_i, \Gamma_{\mathrm{so}i})\} \stackrel{\Delta}{\simeq} [\![P]\!]_{\mathrm{p}}\Gamma, 0b$
  - $\{(\mathcal{O}'_j, \Gamma'_{\mathrm{so}j})\} \stackrel{\Delta}{\simeq} [\![P']\!]_{\mathrm{p}}\Gamma, 1b$

– $[\![id_{\mathrm{c}}[P]]\!]_{\mathrm{p}}\Gamma, \Gamma_{\mathrm{c}}, \underline{v_{\mathrm{c}}} \stackrel{\Delta}{\simeq} [\![P]\!]_{\mathrm{p}}\Gamma, \Gamma_{\mathrm{c}}, v'_{\mathrm{c}}\underline{v_{\mathrm{c}}}$ where
  - $v'_{\mathrm{c}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}(id_{\mathrm{c}})$

– $[\![id_{\mathrm{p}}(\underline{e_{\mathrm{c}}}; \underline{e_{\mathrm{s}+}}; \underline{e_{\mathrm{a}}}; \mathbf{out}\ \underline{id_{\mathrm{s}}}); P]\!]_{\mathrm{p}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b, \underline{v_{\mathrm{c}}} \stackrel{\Delta}{\simeq} \{(\mathcal{O}_i |_S \mathcal{O}'_{j_i}, \Gamma_{\mathrm{so}i}\langle\Gamma'_{\mathrm{so}j_i}\rangle)\}$
  where
  - $m \stackrel{\Delta}{\simeq} |\underline{e_{\mathrm{c}}}| + |\underline{e_{\mathrm{s}+}}| + 2$
  - $b_i \stackrel{\Delta}{\simeq} \delta_i^m$
  - $b'_j \stackrel{\Delta}{\simeq} \delta^m_{|\underline{e_{\mathrm{s}+}}|+j}$
  - $b^1 \stackrel{\Delta}{\simeq} \delta^m_{|\underline{e_{\mathrm{s}+}}|+|\underline{e_{\mathrm{c}}}|+1}$
  - $b^2 \stackrel{\Delta}{\simeq} \delta^m_{|\underline{e_{\mathrm{s}+}}|+|\underline{e_{\mathrm{c}}}|+2}$
  - $\underset{\sim}{\underline{v_{\mathrm{s}}}}.i \stackrel{\Delta}{\simeq} [\![\underline{e_{\mathrm{s}+}}.i]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, b_i b,$
  - $\underline{v_{\mathrm{c}}}.j \stackrel{\Delta}{\simeq} [\![\underline{e_{\mathrm{c}}}.j]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b'_j b$
  - $\underline{v_{\mathrm{a}}}.k \stackrel{\Delta}{\simeq} [\![\underline{e_{\mathrm{a}}}.k]\!]_{\mathrm{a}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \underline{v_{\mathrm{c}}}$
  - $\{(\mathcal{O}_i, \Gamma_{\mathrm{so}i})\} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{m}}(id_{\mathrm{p}})(\underline{v_{\mathrm{c}}}, \underset{\sim}{\underline{v_{\mathrm{s}}}}, \underline{v_{\mathrm{a}}}, \underline{id_{\mathrm{s}}}, \underline{v_{\mathrm{c}}}, b^1 b)$
  - $\{\{(\mathcal{O}'_j, \Gamma'_{\mathrm{so}j})\}_i\} \stackrel{\Delta}{\simeq} [\![P]\!]_{\mathrm{p}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}\langle\Gamma_{\mathrm{so}i}\rangle, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b^2 b$

– $[\![D\ ;\ P]\!]_{\mathrm{p}}\Gamma, b, \underline{v_{\mathrm{c}}} \stackrel{\Delta}{\simeq} \{(\mathcal{O}_i, \Gamma'_{\mathrm{so}}\langle\Gamma_{\mathrm{so}i}\rangle)\}$ where
  - $\Gamma', \Gamma'_{\mathrm{so}} \stackrel{\Delta}{\simeq} [\![D]\!]_{\mathrm{d}}\Gamma, 0b$
  - $\{(\mathcal{O}_i, \Gamma_{\mathrm{so}i})\} \stackrel{\Delta}{\simeq} [\![P]\!]_{\mathrm{p}}\Gamma', 1b, \underline{v_{\mathrm{c}}}$

– $[\![id_{\mathrm{s}} = \mathbf{force}(e_{\mathrm{s}})\ ;\ P]\!]_{\mathrm{p}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b, \underline{v_{\mathrm{c}}} \stackrel{\Delta}{\simeq} \{(\underline{\mathcal{O}_{j_1}}.1 \mid_S \ldots \mid_S \underline{\mathcal{O}_{j_m}}.m, \emptyset)\}$
  where
  - $\underset{\sim}{v_{\mathrm{s}}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$
  - $\underline{\{(\mathcal{O}_j, \Gamma_{\mathrm{so}j})\}}.i \stackrel{\Delta}{\simeq} [\![P]\!]_{\mathrm{p}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}\langle id_{\mathrm{s}} \mapsto \underset{\sim}{v_{\mathrm{s}}}.i\rangle, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, \delta_i^{|\underset{\sim}{v_{\mathrm{s}}}|} b, \underline{v_{\mathrm{c}}}$

– $[\![\mathbf{init}\ e_{\mathrm{s}} = r]\!]_{\mathrm{p}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b, \underline{v_{\mathrm{c}}} \stackrel{\Delta}{\simeq} \{(\mathcal{O}, \emptyset)\}$ where
  - $\underset{\sim}{v_{\mathrm{s}}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}$
  - $v_{\mathrm{ns}} \stackrel{\Delta}{\simeq} nf(\underset{\sim}{v_{\mathrm{s}}}.1, \underline{v_{\mathrm{c}}})$
  - $\mathcal{O} \stackrel{\Delta}{\simeq} I_S(v_{\mathrm{ns}}, r)$

  if $|\underset{\sim}{v_{\mathrm{s}}}| = 1$

We furthermore define $[\![P]\!]_{\mathrm{p}} = [\![P]\!]_{\mathrm{p}}\emptyset, \emptyset, \emptyset, \emptyset, \epsilon, \top$ for programs which constitute a complete model, i.e. which have no free identifiers.

Explanation of the denotation function. The case of reactions relies on the given concrete semantic function for assigning a semantical object to the reaction evaluated to its normal form. This normal form reaction is in turn obtained by evaluating the species expressions to their normal forms, which involves completing the compartment hierarchy in species values, and by evaluating rate expressions to rate values. The latter assumes the denotation function for algebraic rate expressions to be extended to rate expressions in an evident manner. There is one explicit condition for well-typedness, namely that species expressions must be deterministic, i.e. evaluate to singleton lists of species values. There is also the implicit condition that the concrete semantic function must be defined for the computed normal form reaction, which may e.g. fail if non-mass-action rates are used in a CTMC semantics.

The denotation of the nil program is simply the singleton set with the nil semantical object and the empty output species environment.

The denotation of a parallel composition is the pairwise composition of all semantical objects in the denotations of the operands, together with the pairwise update of output species environments from the first component with those of the second. The fresh name prefixes are extended appropriately. This case is well-typed when the composition operation, which is a parameter of the general semantic function, is defined.

The denotation of a variation composition is similar to that of parallel composition but results in a union of semantical objects rather than a Cartesian product.

In the case of located programs, the compartment identifier is looked up in the compartment environment and appended to the list of compartment values used to compute the denotation of the nested program. The denotation is defined when the compartment identifier is in the given compartment environment and when the resulting list of compartment values is well-typed.

The case of module invocation evaluates the actual parameters and passes the resulting values as parameters to the function denoting the module as given by the module environment. This function takes two additional parameters, namely the parent compartments at time of invocation and a fresh name string. From the function we obtain a set of semantical objects together with output species environments with bindings for the actual output species parameters. The sequential program is then evaluated in the species environment updated with the appropriate bindings for the output species. The result is the set of all pairwise compositions of semantical objects from the module and from the sequential program, together with the pairwise update of output species environments from the module with those from the sequential program. Hence the sequential program is treated as a parallel program with respect to semantical objects.

Special care must be taken to ensure the proper extension of fresh name strings for evaluating compartment expressions, species expressions, the module body and the sequential program. The crucial characteristic of these strings is

that none is a postfix of another, ensuring that there is no way of extending one string to match another. So far we have achieved this in the semantics of binary operators by prefixing respectively a 0 and a 1 to the fresh name string. But here we are faced with lists of expressions to be evaluated. We then achieve the desired property by letting all prefixes be of length $|\underline{e_{\mathrm{s}}}| + |\underline{e_{\mathrm{s}+}}| + 2$, where the plus two term accounts for the module body and for the sequential program. For the ith compartment expression we choose a prefix in which the ith symbol is 1 and the remaining symbols are 0s, and a similar construction is used for the remaining prefixes. The denotation function for module invocation is defined when the module identifier is in the given environment and the associated function is defined for the given arguments.

The case of definitions relies on the denotation function for definitions to obtain an updated collection of environments in which the sequential program following the definition is evaluated.

The case of nondeterministic selection evaluates the species expression, and for each resulting species value, it evaluates the sequential program. As for module invocation, special care is needed to ensure that the fresh name strings are extended appropriately. The resulting set of semantical objects consists of all possible compositions of semantical objects associated with each species value, and is hence effectively a Cartesian product. The output species environments resulting from repeated evaluation of the sequential program are disregarded, since there does not appear to be any meaningful way to reconcile them. They all have the same domain, but generally differ in their images, since each is a result of evaluating the same sequential program with different bindings for the forced species.

Finally, the case of initial population or concentration definitions evaluates the given species expression, obtains the corresponding normal form based on the current parent compartments, and uses the concrete semantic function to obtain a semantical object.

Derived programs. Next we define the denotation of derived forms in terms of basic programs. We start by considering in-line species definitions which intuitively give rise to a sequence of standard species definitions followed by the reaction in which the in-line definitions have been removed and by the given sequential program; an example of this is given in Section 3.3. The formal presentation relies on an auxiliary *definition extraction function* of the form $[\![e_{\mathrm{s}}]\!]_{\mathrm{ds}} = e'_{\mathrm{s}}, \underline{D}$ where e_{s} is a derived species expression as defined in the abstract syntax for derived programs, e'_{s} is a basic species expression and $\underline{D}$ is a list of extracted species definitions. Selected cases of the definition are shown below; the remaining cases are similar:

- $[\![id_{\mathrm{c}}[e_{\mathrm{s}}]]\!]_{\mathrm{ds}} \stackrel{\Delta}{\simeq} id_{\mathrm{c}}[e'_{\mathrm{s}}], \underline{D}$ where
  - $e'_{\mathrm{s}}, \underline{D} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{ds}}$

- $[\![e_{\mathrm{s}} - e'_{\mathrm{s}}]\!]_{\mathrm{ds}} \stackrel{\Delta}{\simeq} e''_{\mathrm{s}} - e'''_{\mathrm{s}}, \underline{D}\,\underline{D'}$ where
  - $e''_{\mathrm{s}}, \underline{D} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{ds}}$
  - $e'''_{\mathrm{s}}, \underline{D'} \stackrel{\Delta}{\simeq} [\![e'_{\mathrm{s}}]\!]_{\mathrm{ds}}$
- $[\![id_{\mathrm{s}}]\!]_{\mathrm{ds}} \stackrel{\Delta}{\simeq} id_{\mathrm{s}}$
- $[\![e_{\mathrm{s}}\ \mathbf{as}\ id_{\mathrm{s}}]\!]_{\mathrm{ds}} \stackrel{\Delta}{\simeq} e'_{\mathrm{s}}, (id_{\mathrm{s}} = e'_{\mathrm{s}})\underline{D}$ where
  - $e'_{\mathrm{s}}, \underline{D} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}}]\!]_{\mathrm{ds}}$

The definition of L-R-equality preserving arrows, outlined informally in Section 3.5, relies on an auxiliary *linearisation function* for renaming identifiers in species expressions in a linear manner. Informally, the renaming is such that all species identifiers in the reactants become distinct, all species identifiers in the products become distinct, and the ith occurrences of a given identifier in the original reactants and products are given the same name.

The linearisation function is of the form $lin(e_{\mathrm{s}}, M) \stackrel{\Delta}{\simeq} e'_{\mathrm{s}}, M'$ where M and M' are multisets of species identifiers. Two key cases of the definition are given below where, for $i \in \mathbb{N}$, $bs(i)$ is the binary string representation of i:

- $lin(id_{\mathrm{s}}, M) \stackrel{\Delta}{\simeq} bs(M(id_{\mathrm{s}}))\_id_{\mathrm{s}}, M + id_{\mathrm{s}}$
- $lin(e_{\mathrm{s}} - e'_{\mathrm{s}}, M) \stackrel{\Delta}{\simeq} e''_{\mathrm{s}} - e'''_{\mathrm{s}}, M''$ where
  - $e''_{\mathrm{s}}, M' \stackrel{\Delta}{\simeq} lin(e_{\mathrm{s}}, M)$
  - $e'''_{\mathrm{s}}, M'' \stackrel{\Delta}{\simeq} lin(e'_{\mathrm{s}}, M')$

The base case prefixes an identifier with the binary string representation of the number of the identifier's previous occurrences, and adds the identifier to the multiset. The case of complex formation evaluates the first expression in the given multiset, resulting in a new multiset in which the second expression is evaluated. The remaining cases which are not shown here simply evaluate components recursively in the original multisets.

We extend the linearisation function to the form $lin(\underline{e_{\mathrm{s}}}, M) = \underline{e'_{\mathrm{s}}}, M'$ in order to rename species identifiers in reactant and product lists:

- $lin(\varepsilon, M) \stackrel{\Delta}{\simeq} \varepsilon, M$
- $lin(e_{\mathrm{s}}\underline{e'_{\mathrm{s}}}, M) \stackrel{\Delta}{\simeq} e''_{\mathrm{s}}\underline{e'''_{\mathrm{s}}}, M''$ where
  - $e''_{\mathrm{s}}, M' \stackrel{\Delta}{\simeq} lin(e_{\mathrm{s}}, M)$
  - $\underline{e'''_{\mathrm{s}}}, M'' \stackrel{\Delta}{\simeq} lin(\underline{e'_{\mathrm{s}}}, M')$

We also extend it to the form $lina(e_{\mathrm{a}}, M) = \underline{e'_{\mathrm{a}}}, M'$ for algebraic rate expressions with selected cases defined as follows:

- $lina(r, M) \stackrel{\Delta}{\simeq} r, M$

– $\mathit{lina}(e_{\mathrm{s}}, M) \stackrel{\Delta}{\simeq} \mathit{lin}(e_{\mathrm{s}}, M)$

– $\mathit{lina}(\mathbf{if}\ e_{\mathrm{b}}\ \mathbf{then}\ e_{\mathrm{a}}\ \mathbf{else}\ e'_{\mathrm{a}}) \stackrel{\Delta}{\simeq} \mathbf{if}\ e_{\mathrm{b}}\ \mathbf{then}\ e''_{\mathrm{a}}\ \mathbf{else}\ e'''_{\mathrm{a}}, M''$ where
- $e''_{\mathrm{a}}, M' \stackrel{\Delta}{\simeq} \mathit{lina}(e_{\mathrm{a}}, M)$
- $e'''_{\mathrm{a}}, M'' \stackrel{\Delta}{\simeq} \mathit{lina}(e'_{\mathrm{a}}, M')$

– $\mathit{lina}(e_{\mathrm{a}} + e'_{\mathrm{a}}, M) \stackrel{\Delta}{\simeq} e''_{\mathrm{a}} + e'''_{\mathrm{a}}, M''$ where
- $e''_{\mathrm{a}}, M' \stackrel{\Delta}{\simeq} \mathit{lina}(e_{\mathrm{a}}, M)$
- $e'''_{\mathrm{a}}, M'' \stackrel{\Delta}{\simeq} \mathit{lina}(e'_{\mathrm{a}}, M)$

The derived forms are then defined by a denotation function of the form:

$$[\![P]\!]_{\mathrm{dp}} = P'$$

where P is a derived form program and P' is a basic program. In the following, we assume a function of the form $\fatsemi \underline{D}; P = P'$ which, given a list $\underline{D}$ of definitions and a program P, gives a program P' in which the definitions in $\underline{D}$ have been composed sequentially following the order of $\underline{D}$ and have scope P. We assume a function of the form $\mathit{order}(\mathcal{D}) = \underline{D}$ which orders a set $\mathcal{D}$ of definitions in some arbitrary but definite order. The function FS gives the set of species identifiers in a species expression and is defined along standard lines.

– $[\![\underline{e''_{\mathrm{s}}} \sim \underline{n \cdot e_{\mathrm{s}}}\ \ A_2^{e_{\mathrm{r}}, e'_{\mathrm{r}}}\ \ \underline{n' \cdot e'_{\mathrm{s}}} \sim \underline{e'''_{\mathrm{s}}}\ \mathbf{if}\ e_{\mathrm{b}}, e'_{\mathrm{b}}\ ; P]\!]_{\mathrm{dp}} \stackrel{\Delta}{\simeq}$
$[\![\underline{e''_{\mathrm{s}}} \sim \underline{n \cdot e_{\mathrm{s}}}\ \ A(A_2)^{e_{\mathrm{r}}}\ \ \underline{n' \cdot e'_{\mathrm{s}}}\ \mathbf{if}\ e_{\mathrm{b}}\ ; \mathbf{0}_{\mathrm{p}}]\!]_{\mathrm{dp}} \ \mid$
$[\![\underline{e'''_{\mathrm{s}}} \sim \underline{n' \cdot e'_{\mathrm{s}}}\ \ A(A_2)^{e'_{\mathrm{r}}}\ \ \underline{n \cdot e_{\mathrm{s}}}\ \mathbf{if}\ e'_{\mathrm{b}}\ ; P]\!]_{\mathrm{dp}}$

where
- $A(\Leftrightarrow) \stackrel{\Delta}{\simeq} \Rightarrow$
- $A(\leftrightarrow) \stackrel{\Delta}{\simeq} \rightarrow$
- $A(\twoheadleftarrow\!\!\twoheadrightarrow) \stackrel{\Delta}{\simeq} \twoheadrightarrow$

– $[\![\underline{e''_{\mathrm{s}}} \sim \underline{n \cdot e_{\mathrm{s}}}\ \ A^{e_{\mathrm{r}}}\ \ \underline{n' \cdot e'_{\mathrm{s}}}\ \mathbf{if}\ \ e_{\mathrm{b}}\ ; P]\!]_{\mathrm{dp}} \stackrel{\Delta}{\simeq}$
$[\![\underline{1 \cdot e''_{\mathrm{s}}}\ \underline{n \cdot e_{\mathrm{s}}}\ \ A^{e_{\mathrm{r}}}\ \ \underline{1 \cdot e''_{\mathrm{s}}}\ \underline{n' \cdot e'_{\mathrm{s}}}\ \mathbf{if}\ e_{\mathrm{b}}\ ; P]\!]_{\mathrm{dp}}$

– $[\![\underline{n \cdot e_{\mathrm{s}}}\ \ A^{e_{\mathrm{r}}}\ \ \underline{n' \cdot e'_{\mathrm{s}}}\ \mathbf{if}\ \ e_{\mathrm{b}}\ ; P]\!]_{\mathrm{dp}} \stackrel{\Delta}{\simeq} \fatsemi \underline{D}\ \underline{D'}; ([\![\underline{n \cdot e''_{\mathrm{s}}}\ \ A^{e_{\mathrm{r}}}\ \ \underline{n' \cdot e'''_{\mathrm{s}}}\ \mathbf{if}\ e_{\mathrm{b}}]\!]_{\mathrm{dp}} \mid P)$

where
- $\underline{e''_{\mathrm{s}}}, \underline{D} = [\![\underline{e_{\mathrm{s}}}]\!]_{\mathrm{ds}}$
- $\underline{e'''_{\mathrm{s}}}, \underline{D'} = [\![\underline{e'_{\mathrm{s}}}]\!]_{\mathrm{ds}}$

– $[\![\underline{n \cdot e_{\mathrm{s}}}\ \ \rightarrow^{e_{\mathrm{r}}}\ \ \underline{n' \cdot e'_{\mathrm{s}}}\ \mathbf{if}\ \ e_{\mathrm{b}}]\!]_{\mathrm{dp}} \stackrel{\Delta}{\simeq}$
$\fatsemi \mathit{order}\{id_{\mathrm{s}} = \ \mathbf{force}\ id_{\mathrm{s}} \mid id_{\mathrm{s}} \in FS(\underline{e_{\mathrm{s}}}, \underline{e'_{\mathrm{s}}})\}\ ;\ \underline{n \cdot e_{\mathrm{s}}}\ \ \Rightarrow^{e_{\mathrm{r}}}\ \ \underline{n' \cdot e'_{\mathrm{s}}}\ \mathbf{if}\ e_{\mathrm{b}}$

– $[\![\underline{n \cdot e_{\mathrm{s}}}\ \ \twoheadrightarrow^{e_{\mathrm{r}}}\ \ \underline{n' \cdot e'_{\mathrm{s}}}\ \mathbf{if}\ \ e_{\mathrm{b}}]\!]_{\mathrm{dp}} \stackrel{\Delta}{\simeq}$
$\fatsemi \mathit{order}\{bs(i)\_id_{\mathrm{s}} = \ \mathbf{force}\ id_{\mathrm{s}} \mid id_{\mathrm{s}} \in dom(M) \wedge i \in M(id_{\mathrm{s}})\}\ ;$
$\underline{n \cdot e''_{\mathrm{s}}}\ \ \Rightarrow^{e'_{\mathrm{r}}}\ \ \underline{n' \cdot e'''_{\mathrm{s}}}\ \mathbf{if}\ e_{\mathrm{b}}$

where

- $e'_{\mathrm{r}}, M' \stackrel{\Delta}{\simeq} lin(e_{\mathrm{r}}, \emptyset)$
- $\underline{e''_{\mathrm{s}}}, M'' \stackrel{\Delta}{\simeq} lin(\underline{e_{\mathrm{s}}}, \emptyset)$
- $\underline{e'''_{\mathrm{s}}}, M''' \stackrel{\Delta}{\simeq} lin(\underline{e'_{\mathrm{s}}}, \emptyset)$
- $M(id_{\mathrm{s}}) \stackrel{\Delta}{\simeq} \{1 \ldots max(M'(id_{\mathrm{s}}), M''(id_{\mathrm{s}}), M'''(id_{\mathrm{s}}))\}$

The first case defines a reversible reaction as the parallel compositions of the two reactions, one for each direction. The second case defines an enzymatic reaction as a non-enzymatic reaction in which the enzymes are included in both reactants and products and hence do not get consumed. Other choices, following e.g. Michaelis-Menten kinetics, could also be made here.

The third case defines a reaction with in-line definitions as a reaction where definitions have been extracted and put in scope of both the reaction and the following program which is composed in parallel. Although conceptually simple, we note that there are some artificial cases which may not expand to the expected result, specifically when the same identifier is defined multiple times in a reaction as in e.g. a−b **as** a + a−b **as** a => a−b; here all occurrences of a will be bound to the same expression, namely that resulting from the second in-line definition.

The last two cases define nondeterministic reaction arrows in terms of the force operator and the deterministic reaction arrow. Note that nondeterministic species must be bound to identifiers in reactions in order to preserve the relationship between identical nondeterministic species in reactants and products. Reactions with explicit nondeterminism are therefore ill-typed. Note also that for reactions with the L-R equality-preserving arrow, a given identifier should generally have the same number of occurrences in the reactants and products to obtain meaningful results, although this condition is not explicitly enforced. In particular, reactions such as 2 s −>> s−s and s + s −>> s−s are *not* equivalent according to the above definition of derived forms.

The order of evaluation of derived forms is significant. Specifically, in-line species definitions are expanded before nondeterministic selection. This ensures that e.g. the program s + t −> s−t **as** a; P expands correctly, i.e. to:

```
1 spec a = s−t;
2 (s = force s;
3  t = force t;
4  s + t => s−t
5 ) | P
```

rather than to:

```
1 spec a = s−t;
2 s = force s;
3 t = force t;
4 (s + t => s−t | P)
```

5.4 Definitions

The denotation function. The denotation function for definitions updates the environments with bindings for a given definition. It takes the following form:

$$[\![D]\!]_{\mathrm{d}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b = \Gamma'_{\mathrm{c}}, \Gamma'_{\mathrm{s}}, \Gamma'_{\mathrm{a}}, \Gamma'_{\mathrm{m}}, \Gamma_{\mathrm{so}}$$

The output species environment is *created* by the denotation function and is always empty except for the case of species definitions, where it captures the binding for the defined species. This is in contrast to the other environments which are *updated* by the denotation function. Here is the definition:

- $[\![id_{\mathrm{s}} = e_{\mathrm{s}+}]\!]_{\mathrm{d}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}\langle id_{\mathrm{s}} \mapsto \underset{\sim}{v_{\mathrm{s}}}\rangle, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, \{id_{\mathrm{s}} \mapsto \underset{\sim}{v_{\mathrm{s}}}\}$ where
  - $\underset{\sim}{v_{\mathrm{s}}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{s}+}]\!]_{\mathrm{s}+}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, b$
- $[\![id_{\mathrm{c}} = e_{\mathrm{c}}]\!]_{\mathrm{d}}\Gamma, \Gamma_{\mathrm{c}} \stackrel{\Delta}{\simeq} \Gamma, \Gamma_{\mathrm{c}}\langle id_{\mathrm{c}} \mapsto v_{\mathrm{c}}\rangle, \emptyset$ where
  - $v_{\mathrm{c}} \stackrel{\Delta}{\simeq} [\![e_{\mathrm{c}}]\!]_{\mathrm{c}}\Gamma_{\mathrm{c}}, b$
- $[\![id_{\mathrm{a}}(\underline{id_{\mathrm{c}}}; \underline{id_{\mathrm{s}} : \xi}; \underline{id_{\mathrm{a}}}) = e_{\mathrm{a}}]\!]_{\mathrm{d}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}\langle id_{\mathrm{a}} \mapsto f\rangle, \Gamma_{\mathrm{m}}, \emptyset$
  where
  - $f(\underline{v_{\mathrm{c}}}, \underline{\underset{\sim}{v_{\mathrm{s}}}}, \underline{v_{\mathrm{a}}}, \underline{v_{\mathrm{c}}}') \stackrel{\Delta}{\simeq} [\![e_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma'_{\mathrm{c}}, \Gamma'_{\mathrm{s}}, \Gamma'_{\mathrm{a}}, \underline{v_{\mathrm{c}}}'$
  - $\Gamma'_{\mathrm{c}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}\langle\{id_{\mathrm{c}} \mapsto v_{\mathrm{c}}\}\rangle$
  - $\Gamma'_{\mathrm{s}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{s}}\langle\{id_{\mathrm{s}} \mapsto \mathit{close}(\underset{\sim}{v_{\mathrm{s}}}, \xi)\}\rangle$
  - $\Gamma'_{\mathrm{a}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{a}}\langle\{id_{\mathrm{a}} \mapsto v_{\mathrm{a}}\}\rangle$
- $[\![id_{\mathrm{p}}(\underline{id_{\mathrm{c}}}; \underline{id_{\mathrm{s}} : \xi}; \underline{id_{\mathrm{a}}};\ \mathbf{out}\ \underline{id'_{\mathrm{s}} : e'_{\mathrm{s}}}) = P]\!]_{\mathrm{d}}\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b \stackrel{\Delta}{\simeq}$
  $\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{s}}, \Gamma_{\mathrm{a}}, \Gamma_{\mathrm{m}}\langle id_{\mathrm{p}} \mapsto g\rangle, \emptyset$
  where
  - $g(\underline{v_{\mathrm{c}}}, \underline{\underset{\sim}{v_{\mathrm{s}}}}, \underline{v_{\mathrm{a}}}, \underline{id''_{\mathrm{s}}}, b', \underline{v_{\mathrm{c}}}') \stackrel{\Delta}{\simeq} \{(\mathcal{O}_i, \Gamma_{\mathrm{so}i})\}$
  - $\Gamma'_{\mathrm{c}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{c}}\langle\{id_{\mathrm{c}} \mapsto v_{\mathrm{c}}\}\rangle$
  - $\Gamma'_{\mathrm{s}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{s}}\langle\{id_{\mathrm{s}} \mapsto \mathit{seal}(\mathit{close}(\underset{\sim}{v_{\mathrm{s}}}, \xi), b)\}\rangle$
  - $\Gamma''_{\mathrm{s}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{s}}\langle\{id_{\mathrm{s}} \mapsto \underset{\sim}{v_{\mathrm{s}}}\}\rangle$
  - $\Gamma'_{\mathrm{a}} \stackrel{\Delta}{\simeq} \Gamma_{\mathrm{a}}\langle\{id_{\mathrm{a}} \mapsto \mathit{seal}(v_{\mathrm{a}}, b)\}\rangle$
  - $\{(\mathcal{O}_i, \Gamma'_{\mathrm{so}i})\} \stackrel{\Delta}{\simeq} [\![P]\!]_{\mathrm{p}}\Gamma'_{\mathrm{c}}, \Gamma'_{\mathrm{s}}, \Gamma'_{\mathrm{a}}, \Gamma_{\mathrm{m}}, b', \underline{v_{\mathrm{c}}}'$
  - $\Gamma_{\mathrm{so}i} \stackrel{\Delta}{\simeq} \{id''_{\mathrm{s}} \mapsto \mathit{adapt}(\underset{\sim}{v'_{\mathrm{s}}}, \underset{\sim\sim}{v''_{\mathrm{s}}})\}$ where
    - $\underset{\sim}{v'_{\mathrm{s}}} \stackrel{\Delta}{\simeq} \Gamma'_{\mathrm{so}i}(id'_{\mathrm{s}})$
    - $\underset{\sim\sim}{v''_{\mathrm{s}}}, \emptyset = [\![e'_{\mathrm{s}}]\!]_{\mathrm{s}}\Gamma'_{\mathrm{c}}, \Gamma''_{\mathrm{s}}$

Explanation of the denotation function. The cases for species and compartment definitions are straightforward since they rely on the respective denotation functions. The case of rate function definitions updates the rate function environment with a new binding to a function f from actual parameters and parent compartments to an algebraic rate value. This algebraic rate value is computed in the environments at time of definition updated with bindings for the actual parameters, and with the parent compartments at time of invocation. The interfaces of the actual species parameters are updated based on the annotations of the corresponding formal parameters using the *close* function defined in Subsection 5.2 which here is assumed extended to lists of species values. The function f is only defined when the number of actual and formal parameters match, and when the species interface updates result in well-typed species values.

The case of module definitions updates the module environment with a new binding to a function from actual parameters, a fresh name string and parent compartments, to a set of semantical objects and species output environments. The semantical objects are computed in the environments at time of definition updated with bindings for actual parameters, and with the fresh name string and parent compartments at time of invocation. As for algebraic rate expressions, the interfaces of actual species values are updated. But an additional step is taken to confine species values to a namespace given by the fresh name string at time of definition, ensuring that e.g. variables in actual parameters are not captured inside the module. This is done using the *seal* function on modification site expressions, which is given as a parameter of the general semantics; we assume this to be extended appropriately to lists of species values and also to algebraic rate values. Finally, the resulting output species environment is given by a mapping from actual output species identifiers to the values of the corresponding formal output species identifiers as recorded in the output species environment of the body, but with interfaces updated using the *adapt* function defined in Subsection 5.2. The function is assumed extended to pairs of species value lists of the same length. Hence the updates are carried out in a pair-wise manner by matching up corresponding positions in the lists constituting the nondeterministic species values; this is the reason for having nondeterministic species represented by lists rather than sets.

6 Some Concrete Semantics

Practical applications of LBS require specific choices of concrete semantics to be made, and any questions of language expressiveness must also be addressed in the context of a specific concrete semantics. This section therefore gives four examples of concrete semantics, namely: basic Petri nets; coloured Petri nets; ordinary differential equations; and continuous time Markov chains. These follow the ideas in [31], but are adapted to adhere to the general semantics of LBS.

6.1 Preliminaries

The general semantics preserves variables in species modification sites because variables can be exploited by some concrete semantics. But for other concrete

semantics this is not the case, and we can instead parameterise the general semantic function on a structure $(S, |_S, \mathbf{0}_S, G_S, I_S)$ which is the same as before, except that G_S is a function assigning semantical objects to named *ground normal form reactions*. These are normal form reactions in which expressions have been appropriately evaluated based on a variable environment: species values have been evaluated to ground normal form species values as defined previously; rate values have been evaluated to obtain *ground rate values* defined below; and reaction conditionals are omitted because reactions with conditionals which evaluate to **ff** are simply discarded. We therefore start by defining the general assignment R_S of semantical objects to named normal form reactions in terms of an assignment G_S to named ground normal form reactions, allowing a concrete semantics to be defined in terms of either of these.

Ground normal form reactions. *Ground algebraic rate values* differ from algebraic rate values in that species values are replaced by ground normal form species values and conditionals are not included. Ground rate values contain ground algebraic rate values rather than algebraic rate values, and for the mass-action case, these must be constants. The formal definition is given by the grammar in Table 6.1.

Table 6.1. The abstract syntax for ground rate values

$v_{\mathrm{gr}} ::=$	Ground rate value
$\mid \{r\}$	Mass-action rate constant
$\mid [v_{\mathrm{ga}}]$	Ground algebraic rate value
$v_{\mathrm{ga}} ::=$	Ground algebraic rate value
$\mid r$	Constant
$\mid v_{\mathrm{gns}}$	Population
$\mid \mathit{exp}(v_{\mathrm{ga}}) \mid \mathit{log}(v_{\mathrm{ga}}) \mid \mathit{sin}(v_{\mathrm{ga}}) \mid \mathit{cos}(v_{\mathrm{ga}})$	Functions
$\mid v_{\mathrm{ga}} + v'_{\mathrm{ga}} \mid v_{\mathrm{ga}} - v'_{\mathrm{ga}}$	Arithmetic operators
$\mid v_{\mathrm{ga}} \times v'_{\mathrm{ga}} \mid v_{\mathrm{ga}} / v'_{\mathrm{ga}} \mid v_{\mathrm{ga}} \hat{} v'_{\mathrm{ga}}$	

A denotation function of the form $[\![v_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} = v_{\mathrm{ga}}$ assigning ground algebraic rate values to algebraic rate values, given a variable environment, is defined below. Only selected cases for functions and arithmetic operators are shown since the remaining cases are similar.

- $[\![r]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} r$
- $[\![v_{\mathrm{ns}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} [\![v_{\mathrm{ns}}]\!]_{\mathrm{m}}\Gamma_{\mathrm{x}}$

– $[\![\textbf{if } e_{\mathrm{b}} \textbf{ then } v_{\mathrm{a}} \textbf{ else } v'_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} \begin{cases} [\![v_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} \text{ if } [\![e_{\mathrm{b}}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} = \textbf{tt} \\ [\![v'_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} \text{ otherwise} \end{cases}$

– $[\![exp(v_{\mathrm{a}})]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} exp([\![v_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}})$

– $[\![v_{\mathrm{a}} + v'_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} \stackrel{\Delta}{\simeq} [\![v_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}} + [\![v'_{\mathrm{a}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}}$

In the case of normal form species values we assume the denotation function for modification site expressions extended in an evident manner.

Ground normal form reactions are then of the form:

$$G ::= \underline{n \cdot v_{\mathrm{gns}}} \Rightarrow^{v_{\mathrm{gr}}} \underline{n' \cdot v'_{\mathrm{gns}}}$$

The general semantics in terms of ground normal form reactions. The idea in the following construction is to obtain a ground normal form reaction for each possible variable environment associated with a normal form reaction, then get the semantical object of each ground normal form reaction, and finally apply the parallel composition operator to these objects. We therefore start by defining a function of the form $R_S(R, b, \Gamma_{\mathrm{x}}) = \mathcal{O}$ assigning a semantical object $\mathcal{O}$ to a normal form reaction R, named b, given a variable environment Γ_{x}:

$$R_S(\underline{n \cdot v_{\mathrm{ns}}} \Rightarrow^{v_{\mathrm{r}}} \underline{n' \cdot v'_{\mathrm{ns}}} \textbf{ if } e_{\mathrm{b}}, b, \Gamma_{\mathrm{x}}) \stackrel{\Delta}{\simeq} \begin{cases} G_S(\underline{n \cdot [\![v_{\mathrm{ns}}]\!]_{\mathrm{m}}\Gamma_{\mathrm{x}}} \Rightarrow^{[\![v_{\mathrm{r}}]\!]_{\mathrm{a}}\Gamma_{\mathrm{x}}} \underline{n' \cdot [\![v'_{\mathrm{ns}}]\!]_{\mathrm{m}}\Gamma_{\mathrm{x}}}, b) & \text{if } [\![e_{\mathrm{b}}]\!]_{\mathrm{b}}\Gamma_{\mathrm{x}} = \textbf{tt} \\ \mathbf{0}_S & \text{otherwise} \end{cases}$$

If the conditional evaluates to **ff**, the reaction is assigned the nil object, and otherwise the assignment relies on the function G_S for assigning a semantical object to the ground normal form of the reaction. Again we assume the denotation function on modification site expressions to be extended to normal form species values in an evident manner. We also assume the denotation function for ground algebraic rate values to be extended to ground rate values in an evident manner; note that this function is only defined when ground algebraic rate values which are used as mass-action rates evaluate to constants.

The set of all variable environments associated with a normal form reaction is defined as follows, using the standard notation for dependent sets:

$$Val(R) \stackrel{\Delta}{\simeq} \prod_{(x:\rho)\in FV(R)} [\![\rho]\!]_{\mathrm{t}}$$

We here assume the variable function FV on modification site expressions to be extended to reactions in an evident manner. Observe that variable environments are restricted to only assign values of given types to variables, and that for finite types, we get a finite set of variable environments.

In order to construct appropriate binary strings for naming reactions, we assume an arbitrary but fixed total ordering $\leq$ on variable environments Γ_{x}.

In practise this can for example be obtained from a lexicographical ordering on variables together with a suitable ordering on values. We assume an operator $\textcircled{s}$ which gives the parallel composition in some definite order of its operands. Recall also that the function δ_i^m gives a binary string of length m with 0s everywhere except for the ith entry. The assignment R_S can then be defined in terms of G_S as follows:

$$R_S(R,b) \stackrel{\Delta}{\simeq} \\ \textcircled{s}\{R_S(R,\delta_i^m b,\Gamma_{\mathrm{x}}) \mid \Gamma_{\mathrm{x}} \in \mathit{Val}(R) \wedge i = |\{\Gamma'_{\mathrm{x}} \in \mathit{Val}(R) \mid \Gamma'_{\mathrm{x}} \leq \Gamma_{\mathrm{x}}\}| \wedge m = |\mathit{Val}(R)|\}$$

6.2 A Petri Net Semantics

Petri nets. We already encountered a graphical representation of a Petri net in Figure 2.2. *Places*, depicted as circles, represent species, and *transitions*, depicted as rectangles, represent reactions. In the figure we considered only atomic species with no modification sites, but in the more general case, a separate place is used to represent each modification state of a complex species. *Flow functions*, depicted as arcs between places and transitions, are used to identify the reactants and products of a reaction. In the general case, arcs are labelled with integers representing stoichiometry. Finally, a *marking* defines the state of a Petri net by the number of *tokens*, representing individual molecules, contained in each place. For our purposes, tokens are multiset versions of ground normal form species values.

The formal definition of our Petri nets is given below, where V_{gns} is the set of all ground normal form species values v_{gns}.

Definition 1. *An LBS*-Petri net *P is a tuple $(S, T, F^{in}, F^{out}, M^0)$ where*

- $S \subset_{fin} \{\mathit{MS}(v_{gns}) \mid v_{gns} \in V_{gns}\}$ *is the set of* places.
- $T \subset_{fin} \{0,1\}^*$ *is the set of* transitions.
- $F^{in}, F^{out} : T \times S \to \mathbb{N}$ *are the* flow-in *and* flow-out *functions, respectively.*
- $M^0 \in \mathit{MS}(S)$ *is the* initial marking

Recall in the above definition that $\mathit{MS}(\underline{x})$ gives the multiset representation of a list $\underline{x}$. The set of places hence contains multiset-representations of normal form species values, reflecting that the ordering of atomic species within normal form species values is insignificant, i.e. that the complex formation operator is commutative. Transitions are binary strings since these are used to name reactions in the general semantics. We use the notation S_P to refer to the places S of Petri net P, and similarly for the other Petri net elements. The set of all Petri nets is denoted by $\mathcal{P}$.

The qualitative semantics of Petri nets. The qualitative semantics determines how the marking of a Petri net changes over discrete time. Informally, this can be illustrated by playing the *token game*: a transition can fire whenever

all its input places contain at least the number of tokens specified by the corresponding arc weights; when a transition fires, the number of tokens specified by arc weights are consumed from the input places and added to the output places.

Formally, the set of all *markings* of a Petri net is the set of multisets of places:

$$\mathcal{M}(P) \stackrel{\Delta}{\simeq} MS(S_P)$$

The behaviour of a Petri net is defined in terms of a transition relation which captures all possible moves in the token game.

Definition 2. *Let P be a Petri net, let $X \in MS(T_P)$ and let $M, N \in \mathcal{M}(P)$. Then define $M \xrightarrow{X} N$ iff*

1. $M \geq \sum_{t \in X} F_P^{in}(t)$
2. $N = M + \sum_{t \in X} F_P^{out}(t) - F_P^{in}(t)$

Note that a flow function applied to only one argument, a transition, is interpreted as a function on places, here a marking. The arithmetic operations and relations are understood to be extended to markings in the expected way, e.g. $M \geq M'$ iff $M(s) \geq M'(s)$ for all s. Condition 1 hence states that the marking M must have sufficient tokens for transitions in X to fire, and condition 2 states that N is the marking resulting from firing the transitions from X in marking M.

The concrete Petri net semantics of LBS

Definition 3. *The concrete semantics for LBS in terms of Petri nets is given by the tuple $(\mathcal{P}, |_{\mathcal{P}}, \mathbf{0}_{\mathcal{P}}, G_{\mathcal{P}}, I_{\mathcal{P}})$ where*

- $P_1 \,|_{\mathcal{P}}\, P_2 \stackrel{\Delta}{\simeq} P$ *where*
  - $S_P \stackrel{\Delta}{\simeq} S_{P_1} \cup S_{P_2}$
  - $T_P \stackrel{\Delta}{\simeq} T_{P_1} \cup T_{P_2}$
  - $F_P^{io}(t,s) \stackrel{\Delta}{\simeq} \begin{cases} F_{P_1}^{io}(t,s) & \text{if } t \in T_{P_1} \wedge s \in S_{P_1} \\ F_{P_2}^{io}(t,s) & \text{if } t \in T_{P_2} \wedge s \in S_{P_2} \\ 0 & \text{otherwise} \end{cases}$ *for* $io \in \{in, out\}$
  - $M_P^0 \stackrel{\Delta}{\simeq} M_{P_1}^0 + M_{P_2}^0$

  if $T_{P_1} \cap T_{P_2} = \emptyset$
- $\mathbf{0}_P \stackrel{\Delta}{\simeq} (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
- $G_{\mathcal{P}}(\underline{n \cdot v_{gns}} \Rightarrow^{v_{gr}} \underline{n' \cdot v'_{gns}}, t) \stackrel{\Delta}{\simeq} P$ *where*
  - $S_P \stackrel{\Delta}{\simeq} \{MS(\underline{v_{gns}}.i)\} \cup \{ MS(\underline{v'_{gns}}.j)\}$
  - $T_P \stackrel{\Delta}{\simeq} \{t\}$
  - $F_P^{in}(t,s) \stackrel{\Delta}{\simeq} \sum_{MS(\underline{v_{gns}}.i)=s} \underline{n}.i$

- $F_P^{out}(t,s) \stackrel{\Delta}{\simeq} \sum_{MS(\underline{v'_{gns}}.j)=s} \underline{n'}.j$
- $M_P^0 \stackrel{\Delta}{\simeq} \emptyset$

– $I_P(v_{gns}, n) \stackrel{\Delta}{\simeq} (\{MS(v_{gns})\}, \emptyset, \emptyset, \emptyset, \{MS(v_{gns}) \mapsto n\})$

The function I_P is only defined for natural-numbered initial populations, not for real-numbered initial concentrations, because markings in Petri nets are discrete. The parallel composition operator is only defined for Petri nets with disjoint sets of transitions. The transition sets of two Petri nets resulting from the general semantics are however always disjoint because reactions have fresh names. This is in contrast to CBS where a bottom-up approach is taken: the semantics for parallel composition renames transitions before composition.

6.3 A Coloured Petri Net Semantics

Coloured Petri nets (CPNs) allow a single place to represent a species in any of its possible states of modification. CPNs hence allow for a compact description of models and can potentially lead to more efficient simulation and analysis, and in contrast to standard Petri nets, they are capable of representing species with infinite modification site types such as strings.

Coloured Petri nets. Places in CPNs are assigned *types* (or *colour*), and tokens are structured values of the type assigned to the place in which they reside. In our case, the type of a place is given by a multiset of located atomic species names and their modification site types, hence representing a complex species independently of its state of modification. As for standard Petri nets, tokens are multiset versions of ground normal form species values. But in contrast to standard Petri nets, arcs are equipped with multiset representations of normal form species values which are not necessarily ground. This enables a transition to operate selectively on species in a given state of modification, or indeed to ignore the state of certain sites. *Boolean guards* with variables allow transitions to assert further control over tokens.

We give a definition of coloured Petri nets which is tailored to our needs and which avoids some details of the standard definition [18]. For example, the standard definition distinguishes between place names and place types, but for our purposes a place is identified uniquely by its type. Our definition can however be recast in standard terms, as would be necessary for exploiting existing CPN tools.

Formally, we define a *species type* τ as follows:

$$\tau ::= \sum_i \underline{n_{\mathrm{c}i}}[n_{\mathrm{s}i}, \sigma_i]$$

and we let *Types* be the set of all species types. We define a function of the form $type(v_{\mathrm{ns}}) = \tau$ giving the type of a normal form species value:

$$type(\underline{\underline{n_{\mathrm{c}}}[n_{\mathrm{s}}, \alpha_\sigma]}) \stackrel{\Delta}{\simeq} \sum_i \underline{n_{\mathrm{c}i}}[n_{\mathrm{s}i}, \sigma_i]$$

where σ_i is α_i in which each pair of the image has been projected to the type component. We assume a similar definition for a function of the form $type(v_{\mathrm{gns}}) = \tau$ for ground normal form species values.

The formal definition of our coloured Petri nets is given below, where $E_{\mathbf{bool}}$ is set of boolean expressions e_{b}.

Definition 4. *An LBS*-coloured Petri net C *is a tuple* $(S, T, F^{in}, F^{out}, B, M^0)$ *where*

- $S \subset_{fin}$ *Types is a finite set of* places.
- $T \subset_{fin} \{0,1\}^*$ *is a finite set of* transitions.
- $F^{in}, F^{out} : \prod_{(t,\tau)\in T\times S} MS(\{MS(v_{ns}) \mid type(v_{ns}) = \tau\})$ *are the* flow-in *and* flow-out *functions, respectively.*
- $B : T \to E_{\mathbf{bool}}$ *is the* transition guard *function.*
- $M^0 : \prod_{\tau\in S} MS(\{MS(v_{gns}) \mid type(v_{gns}) = \tau\})$ *is the* initial marking.

As for basic Petri nets, we use the notation S_C to refer to the places S of a coloured Petri net C, and similarly for the other elements. The set of all coloured Petri nets is denoted by $\mathcal{C}$.

The qualitative semantics of coloured Petri nets. The set of all *markings* $\mathcal{M}(C)$ of a coloured Petri net C is defined as follows:

$$\mathcal{M}(C) \stackrel{\Delta}{\simeq} \prod_{\tau\in S_C} MS(\{MS(v_{\mathrm{gns}}) \mid type(v_{\mathrm{gns}}) = \tau\})$$

We furthermore let

$$VE_X \stackrel{\Delta}{\simeq} \{\Gamma_{\mathrm{x}} \mid dom(\Gamma_{\mathrm{x}}) = X\}$$

be the set of variable environments with domain X, and we let

$$FV(t, C) \stackrel{\Delta}{\simeq} FV(F_C^{in}(t)) \cup FV(F_C^{out}(t)) \cup FV(B_C(t))$$

be the set of typed variables associated with a transition t in CPN C; here FV is assumed extended in an evident manner. The behaviour of a coloured Petri net is defined in terms of a transition relation as follows.

Definition 5. *Let C be a coloured Petri net, let $X \in MS(\prod_{t\in T_C} VE_{FV(t,C)})$ and let $M, N \in \mathcal{M}(C)$. Then define $M \xrightarrow{X} N$ iff*

1. $M \geq \sum_{(t,\Gamma_x)\in X} [\![F_C^{in}(t)]\!]_m \Gamma_x$
2. $N = M + \sum_{(t,\Gamma_x)\in X} [\![F_C^{out}(t)]\!]_m \Gamma_x - [\![F_C^{in}(t)]\!]_m \Gamma_x$
3. $\bigwedge_{(t,\Gamma_x)\in X} [\![B_C(t)]\!]_m \Gamma_x = \boldsymbol{tt}$

Recall that the modification site denotation function is a parameter of the species semantics, and in the above definition we assume this function to be extended from modification site expressions to normal form species values and to markings in an evident manner. A flow function applied to a transition is here interpreted as a marking, i.e. a mapping from places to multisets, and the multiset operations are assumed to be appropriately extended. Conditions 1 and 2 then correspond to conditions 1 and 2 in the qualitative semantics of standard Petri nets. Condition 3 states that the guards of all fired transitions must evaluate to **tt**.

The concrete coloured Petri net semantics of LBS

Definition 6. *The concrete semantics for LBS in terms of coloured Petri nets is given by the tuple* $(\mathcal{C}, |_{\mathcal{C}}, \mathbf{0}_{\mathcal{C}}, R_{\mathcal{C}}, I_{\mathcal{C}})$ *where*

- $C_1 \mid_{\mathcal{C}} C_2 \stackrel{\Delta}{\simeq} C$ *where*
  - $S_C \stackrel{\Delta}{\simeq} S_{C_1} \cup S_{C_2}$
  - $T_C \stackrel{\Delta}{\simeq} T_{C_1} \cup T_{C_2}$
  - $F_C^{io}(t,\tau) \stackrel{\Delta}{\simeq} \begin{cases} F_{C_1}^{io}(t,\tau) & \text{if } t \in T_{C_1} \wedge \tau \in S_{C_1} \\ F_{C_2}^{io}(t,\tau) & \text{if } t \in T_{C_2} \wedge \tau \in S_{C_2} \\ \emptyset & \text{otherwise} \end{cases}$ *for* $io \in \{in, out\}$
  - $B_C(t) \stackrel{\Delta}{\simeq} \begin{cases} B_{C_1}(t) & \text{if } t \in T_{C_1} \\ B_{C_2}(t) & \text{if } t \in T_{C_2} \end{cases}$
  - $M_C^0 \stackrel{\Delta}{\simeq} M_{C_1}^0 + M_{C_2}^0$

  if $T_{C_1} \cap T_{C_2} = \emptyset$
- $\mathbf{0}_C \stackrel{\Delta}{\simeq} (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
- $R_{\mathcal{C}}(\underline{n \cdot v_{ns}} \Rightarrow^{v_r} \underline{n' \cdot v'_{ns}} \textbf{ if } e_b, t) \stackrel{\Delta}{\simeq} C$ *where*
  - $S_C \stackrel{\Delta}{\simeq} \{type(\underline{v_{ns}}.i)\} \cup \{ type(\underline{v'_{ns}}.j)\}$
  - $T_C \stackrel{\Delta}{\simeq} \{t\}$
  - $F_C^{in}(t,\tau) \stackrel{\Delta}{\simeq} \sum_{type(\underline{v_{ns}}.i)=\tau} \underline{n}.i \cdot MS(\underline{v_{ns}}.i)$
  - $F_C^{out}(t,\tau) \stackrel{\Delta}{\simeq} \sum_{type(\underline{v'_{ns}}.j)=\tau} \underline{n'}.j \cdot MS(\underline{v'_{ns}}.j)$
  - $B_C(t) \stackrel{\Delta}{\simeq} e_b$
  - $M_C^0 \stackrel{\Delta}{\simeq} \emptyset$
- $I_C(v_{ns}, n) \stackrel{\Delta}{\simeq} (\{type(v_{ns})\}, \emptyset, \emptyset, \emptyset, \emptyset, \{MS(v_{ns}) \mapsto n\})$

The definition is similar to that for standard Petri nets, but differs in the inclusion of guards and in the definition of the reaction and initial condition functions where species types and normal form values are used rather than ground values.

6.4 An ODE Semantics

The Petri net semantics presented above are *qualitative* in that they do not take reaction rates into account. In this section we give a *quantitative* semantics in terms of *ordinary differential equations* (ODEs). ODEs are *continuous* since they define system dynamics in terms of species concentrations. They are also *deterministic* since they, given initial conditions, uniquely determine the state of a system at any point of time in terms of species concentrations.

Ordinary differential equations. A set of ODEs specifies how the concentration $[s_i]$ of a species s_i changes over time and is traditionally written in the following notation:

$$\begin{aligned} \mathrm{d}[s_1] &= p_1 \\ &\vdots \\ \mathrm{d}[s_n] &= p_n \end{aligned}$$

where the p_i are real polynomials over $[s_i]$. The initial conditions of a set of ODEs are specified by the concentration of each species at time 0.

Formally, let $(\mathit{Pol}(X), +, \cdot)$ be the ring of real polynomials over variables in the set X. We then define the structure of ODEs with initial conditions as follows:

Definition 7. *A structure D of LBS-*ODEs with initial conditions *is given by a tuple (X, P, I) where*

- $X \subset_{\mathit{fin}} \{\mathit{MS}(v_{gns}) \mid v_{gns} \in V_{gns}\}$ *is the set of* variables.
- $P \colon X \to \mathit{Pol}(X)$ *is the assignment of* polynomials *to variables.*
- $I \colon X \to \mathbb{R}$ *is the* initial condition.

The set of all structures of ODEs with initial conditions is denoted by $\mathcal{D}$, and we denote e.g. X in D by X_D. Although non-linear ODEs cannot generally be solved in closed form, numerical integration methods are available and described in standard text books [1].

The ordinary differential equation semantics of LBS. Given two total functions $f_1 : X_1 \to Y$ and $f_2 : X_2 \to Y$ with a binary operator $+$ on the elements of Y, we define $f_1 + f_2 : X_1 \cup X_2 \to Y$ as follows:

$$(f_1 + f_2)(x) \stackrel{\Delta}{\simeq} \begin{cases} f_1(x) & \text{if } x \in X_1 \setminus X_2 \\ f_2(x) & \text{if } x \in X_2 \setminus X_1 \\ f_1(x) + f_2(x) & \text{if } x \in X_1 \cap X_2 \end{cases}$$

The semantics of LBS in terms of ODEs is defined below.

Definition 8. *The concrete semantics for LBS in terms of ODEs is given by the tuple $(\mathcal{D}, |_{\mathcal{D}}, \mathbf{0}_{\mathcal{D}}, G_{\mathcal{D}}, I_{\mathcal{D}})$ where*

- $D_1 \mid_{\mathcal{D}} D_2 \stackrel{\Delta}{\simeq} D$ *where*
  - $X_D \stackrel{\Delta}{\simeq} X_{D_1} \cup X_{D_2}$
  - $P_D \stackrel{\Delta}{\simeq} P_{D_1} + P_{D_2}$
  - $I_D \stackrel{\Delta}{\simeq} I_{D_1} + I_{D_2}$
- $\mathbf{0}_{\mathcal{D}} \stackrel{\Delta}{\simeq} (\emptyset, \emptyset, \emptyset)$

- $G_{\mathcal{D}}(\underline{n \cdot v_{gns}} \Rightarrow^{v_{gr}} \underline{n' \cdot v'_{gns}}, b) \stackrel{\Delta}{\simeq} D$ *where*
  - $X_D \stackrel{\Delta}{\simeq} \{MS(\underline{v_{gns}}.i)\} \cup \{ MS(\underline{v'_{gns}}.j)\}$
  - $P_D(s) \stackrel{\Delta}{\simeq} \begin{cases} (N(s) - M(s)) \cdot r \cdot \prod_i (MS(\underline{v_{gns}}.i))^{\underline{n}.i} & \text{if } v_{gr} = \{r\} \\ v_{ga} & \text{if } v_{gr} = [v_{ga}] \end{cases}$

    where
    - $M(s) \stackrel{\Delta}{\simeq} \sum_{MS(\underline{v_{gns}}.i)=s} \underline{n}.i$
    - $N(s) \stackrel{\Delta}{\simeq} \sum_{MS(\underline{v'_{gns}}.j)=s} \underline{n'}.j$
  - $I_D(s) = 0$.

- $I_V(v_{gns}, r) \stackrel{\Delta}{\simeq} (\{s\}, \{s \mapsto 0\}, \{s \mapsto r\})$ *where*
  - $s \stackrel{\Delta}{\simeq} MS(v_{gns})$

In the case of reactions, rate expressions are constructed from mass-action rate constants in the standard way [36]. Note that the assignment to reactions is only defined when mass-action rates are constants.

6.5 A CTMC Semantics

We now give another quantitative semantics in terms of *continuous time Markov chains* (CTMCs). In contrast to ODEs, CTMCs are *discrete* since they describe the system state in terms of species populations rather than concentrations, and they give rise to *stochastic* behaviour.

Continuous time Markov chains. The state of a CTMC corresponds to a marking of a Petri net and is hence given by a multiset of ground normal form species values in their multiset form. State transitions are described directly in terms of a transition rate matrix. Here is the formal definition:

Definition 9. *An LBS*-continuous time Markov chain with initial state V *is a tuple* (X, Q, I) *where*

1. $X \subset_{fin} MS(\{MS(v_{gns}) \mid v_{gns} \in V_{gns}\})$ *is the set of* states.
2. $Q : X^2 \to \mathbb{R}$ *is the* transition rate matrix *satisfying*
   (a) $Q(M, N) \geq 0$ *for all* $M, N \in X$ *with* $M \neq N$.
   (b) $Q(M, M) = -\sum_{M \neq N} Q(M, N)$.
3. $I \in X$ *is the* initial state.

The set of all CTMCs with initial state is denoted by $\mathcal{V}$, and we denote e.g. X in V by X_V. We refer to the literature [36] for further details on CTMCs and their associated simulation methods.

The continuous time Markov chain semantics of LBS

Definition 10. *The concrete semantics for LBS in terms of CTMCs is given by the tuple* $(\mathcal{V}, |_\mathcal{V}, \mathbf{0}_\mathcal{V}, G_\mathcal{V}, I_\mathcal{V})$ *where*

- $V_1 \mid_\mathcal{V} V_2 \stackrel{\Delta}{\simeq} V$ *where*
  - $X_V \stackrel{\Delta}{\simeq} \{M + N \mid M \in X_{V_1} \wedge N \in X_{V_2}\}$
  - $Q_V \stackrel{\Delta}{\simeq} Q_{V_1} + Q_{V_2}$
  - $I_V \stackrel{\Delta}{\simeq} I_{V_1} + I_{V_2}$
- $\mathbf{0}_\mathcal{V} \stackrel{\Delta}{\simeq} (\emptyset, \emptyset, \emptyset)$
- $G_\mathcal{V}(\underline{n \cdot v_{gns}} \Rightarrow^{\{r\}} \underline{n' \cdot v'_{gns}}, t) \stackrel{\Delta}{\simeq} V$ *where*
  - $X_V \stackrel{\Delta}{\simeq} MS(\{MS(\underline{v_{gns}}.i)\} \cup \{ MS(\underline{v'_{gns}}.j)\})$
  - $Q_V(M', N') \stackrel{\Delta}{\simeq} \begin{cases} r\binom{M'}{M} & \text{if } (M,N) \preceq (M',N') \wedge M \neq N \\ -r\binom{M'}{M} & \text{if } M' = N' \wedge M \neq N \wedge M' \geq M \\ 0 & \text{otherwise} \end{cases}$

    where
    - $M(s) \stackrel{\Delta}{\simeq} \sum_{MS(\underline{v_{gns}}.i)=s} \underline{n}.i$
    - $N(s) \stackrel{\Delta}{\simeq} \sum_{MS(\underline{v'_{gns}}.j)=s} \underline{n'}.j$
    - $\binom{M'}{M} \stackrel{\Delta}{\simeq} \prod_{s \in dom(M')} \binom{M'(s)}{M(s)}$
    - $(M,N) \preceq (M',N')$ *iff* $\exists L.\ M' = M + L \wedge N' = N + L$
  - $I_V(s) = 0$
- $I_V(v_{gns}, n) \stackrel{\Delta}{\simeq} V$ *where*
  - $X_V \stackrel{\Delta}{\simeq} MS(\{MS(v_{gns})\})$
  - $Q_V(M,N) \stackrel{\Delta}{\simeq} 0$
  - $I_V(\{MS(v_{gns})\}) = n$

In the case of reactions, $\binom{x}{y}$ is the binomial coefficient and state transition rates are constructed from mass-action constants in the standard way [36]. Note that the reaction assignment is only defined when mass-action rates are used and, as in the ODE semantics, mass-action rates must be constants.

7 Future Directions

Combinatorial explosion. Whether the explicit modelling of "empty context" reactions (i.e. where species are listed in fully specified complexes) is desirable or not depends on the particular system under study. Systems which exhibit low

or moderate levels of combinatorial complexity are amenable to modelling in LBS and can benefit from its simplicity and the large body of tools and techniques available for the analysis and simulation of Petri nets, ODEs and CTMCs. Although nondeterminism provides means of handling moderately combinatorial systems in a compact manner, one would like a more refined approach to nondeterminism in order to control which members of nondeterministic species interact in reactions.

Systems which are characterised by high levels of combinatorial complexity may be better modelled in e.g. rule-based languages such as κ and BioNetGen. As mentioned in the introduction, there is however scope for supporting κ and BioNetGen within the general framework of LBS through an appropriate choice of concrete semantics and modification site types. This may furthermore open possibilities for exploiting modularity in the κ analysis methods.

Graphical representations. Although LBS has been designed with ease of use in mind, it is still a textual language that may not be easily accessible to some biologists. Graphical representations of LBS models can ameliorate this problem. Specifically, tools for visualising LBS programs and, conversely, for generating LBS programs from visual diagrams, would be useful. These tools might follow the *Systems Biology Graphical Notation* (SBGN) [21].

A static semantics. The denotation functions impose certain constraints on their arguments. The resulting notion of well-typedness is a dynamical one: the denotation of a module is a function, and whether or not this function is defined for a given set of actual parameters is determined by applying the function to these parameters. This approach falls short in two respects. Firstly, the function may not be defined for any inputs at all. In this case it is the module definition that should be reported as ill-typed, rather than the module invocation. Secondly, well-typedness of a module invocation should be determined based on the actual parameters and an appropriate interface of the module, rather than by attempting to translate the module implementation under a given set of actual parameters. It may be that these limitations can be addressed through a dedicated type system along the lines of a previous version of LBS [27].

Synthetic biology. Whether natural biological system are amenable to extensive modular decomposition remains an open question that must be addressed through further modelling exercises. In the setting of *synthetic* biology, however, systems are *designed* rather than *modelled*, so it should be possible to exploit modularity fully there. Languages with dedicated features for synthetic biology are starting to emerge, including the previously mentioned Antimony language as well as GenoCad [2, 3] and the language for *Genetic Engineering of Cells* (GEC) [26]. GEC has a notion of parallel and located reactions which serve as constraints for deducing appropriate genetic parts for constructing genes, and it

also has a basic notion of modularity. It may then be of interest to add some of the more advanced features of LBS for e.g. structured species and subtyping into an extension of GEC, or conversely to add the support of GEC for genetic design into an extension of LBS.

Acknowledgements. The authors would like to thank Vincent Danos, Stuart Moodie and Nicolas Oury for useful discussions. This work was supported by Microsoft Research through its European PhD Scholarship Programme and by a Royal Society-Wolfson Award. The second author is grateful for the support from The Centre for Systems Biology at Edinburgh, a Centre for Integrative Systems Biology funded by BBSRC and EPSRC, reference BB/D019621/1. Part of the research was carried out at the Microsoft Silicon Valley Research Center.

References

1. Blanchard, P., Devaney, R.L., Hall, G.R.: Differential Equations. Brooks/Cole (2002)
2. Cai, Y., Hartnett, B., Gustafsson, C., Peccoud, J.: A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts. Bioinformatics 23(20), 2760–2767 (2007)
3. Cai, Y., Lux, M.W., Adam, L., Peccoud, J.: Modeling structure-function relationships in synthetic DNA sequences using attribute grammars. PLoS Comput. Biol. 5(10), e1000529 (2009)
4. Calder, M., Gilmore, S., Hillston, J.: Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. Trans. on Comput. Syst. Biol. VII 4230, 1–23 (2006)
5. Cardelli, L.: Brane calculi. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 257–280. Springer, Heidelberg (2005)
6. Chabrier-Rivier, N., Fages, F., Soliman, S.: The biochemical abstract machine BIOCHAM. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 172–191. Springer, Heidelberg (2005)
7. Chen, W.W., Schoeberl, B., Jasper, P.J., Niepel, M., Nielsen, U.B., Lauffenburger, D.A., Sorger, P.K.: Input-output behavior of ErbB signaling pathways as revealed by a mass action model trained against dynamic data. Mol. Syst. Biol. 5(239) (2009)
8. Ciocchetta, F., Hillston, J.: Bio-PEPA: An extension of the process algebra PEPA for biochemical networks. Electron. Notes Theor. Comput. Sci. 194(3), 103–117 (2008)
9. Danos, V.: Agile modelling of cellular signalling. In: Computation in Modern Science and Engineering, vol. 2, Part A 963, pp. 611–614 (2007)
10. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling and model perturbation. TCSB 5750(11), 116–137 (2009)
11. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling of cellular signalling. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 17–41. Springer, Heidelberg (2007)

12. Dematté, L., Priami, C., Romanel, A.: Modelling and simulation of biological processes in BlenX. SIGMETRICS Performance Evaluation Review 35(4), 32–39 (2008)
13. Faeder, J.R., Blinov, M.L., Hlavacek, W.S.: Graphical rule-based representation of signal-transduction networks. In: Liebrock, L.M. (ed.) Proc. 2005 ACM Symp. Appl. Computing, pp. 133–140. ACM Press, New York (2005)
14. Guerriero, M.L., Heath, J.K., Priami, C.: An automated translation from a narrative language for biological modelling into process algebra. In: Calder, M., Gilmore, S. (eds.) CMSB 2007. LNCS (LNBI), vol. 4695, pp. 136–151. Springer, Heidelberg (2007)
15. Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comput. Program. 8(3), 231–274 (1987)
16. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets for systems and synthetic biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)
17. Hucka, M., et al.: The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19(4), 524–531 (2003)
18. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, vol. 1. Springer, Heidelberg (1992)
19. Kofahl, B., Klipp, E.: Modelling the dynamics of the yeast pheromone pathway. Yeast 21(10), 831–850 (2004)
20. Kwiatkowski, M., Stark, I.: The continuous π-calculus: a process algebra for biochemical modelling. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 103–122. Springer, Heidelberg (2008)
21. Le Novère, N., et al.: The systems biology graphical notation. Nature Biotechnology 27, 735–741 (2009)
22. Mallavarapu, A., Thomson, M., Ullian, B., Gunawardena, J.: Programming with models: modularity and abstraction provide powerful capabilities for systems biology. J. R. Soc. Interface (2008)
23. Murata, T.: Petri nets: properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
24. Paun, G., Rozenberg, G.: A guide to membrane computing. Theor. Comput. Sci. 287(1), 73–100 (2002)
25. Pedersen, M.: Compositional definitions of minimal flows in Petri nets. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 288–307. Springer, Heidelberg (2008)
26. Pedersen, M., Phillips, A.: Towards programming languages for genetic engineering of living cells. J. R. Soc. Interface special issue (2009)
27. Pedersen, M., Plotkin, G.: A Language for Biochemical Systems. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 63–82. Springer, Heidelberg (2008)
28. Peyssonnaux, C., Eychène, A.: The Raf/MEK/ERK pathway: new concepts of activation. Biol. Cell. 93(1-2), 53–62 (2001)
29. Phillips, A., Cardelli, L., Castagna, G.: A graphical representation for biological processes in the stochastic pi-calculus. In: Priami, C., Ingólfsdóttir, A., Mishra, B., Riis Nielson, H. (eds.) Transactions on Computational Systems Biology VII. LNCS (LNBI), vol. 4230, pp. 123–152. Springer, Heidelberg (2006)

30. Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)
31. Plotkin, G.: A calculus of biochemical systems (in preparation)
32. Priami, C.: Stochastic pi-calculus. The Computer Journal 38(7), 578–589 (1995)
33. Priami, C., Quaglia, P.: Beta binders for biological interactions. In: Danos, V., Schachter, V. (eds.) CMSB 2004. LNCS (LNBI), vol. 3082, pp. 20–33. Springer, Heidelberg (2005)
34. Regev, A., Paninab, E.M., Silverman, W., Cardelli, L., Shapiro, E.: BioAmbients: an abstraction for biological compartments. Theor. Comput. Sci. 325(1), 141–167 (2004)
35. Smith, L.P., Bergmann, F.T., Chandran, D., Sauro, H.M.: Antimony: a modular model definition language. Bioinformatics 25(18), 2452–2454 (2009)
36. Wilkinson, D.J.: Stochastic Modelling for Systems Biology. Chapman & Hall/CRC, Boca Raton (2006)