Preliminary Proceedings of the 11th International Workshop on Developments in Computational Models (DCM 2015)

http://dcm-workshop.org.uk/2015

A satellite event of ICTAC 2015

28 October 2015

Universidad Javeriana

Cali – Colombia

Edited by César A. Muñoz
Jorge A. Pérez
Preface

This volume contains the papers presented at DCM 2015: 11th International Workshop on Developments in Computational Models held on October 28, 2015 in Cali, Colombia. DCM 2015 was organized as a one-day satellite event of the 12th International Colloquium on Theoretical Aspects of Computing (ICTAC 2015).

Several new models of computation have emerged in the last few years, and many developments of traditional computational models have been proposed with the aim of taking into account the new demands of computer systems users and the new capabilities of computation engines. A new computational model, or a new feature in a traditional one, usually is reflected in a new family of programming languages, and new paradigms of software development.

The aim of the DCM workshop series is to bring together researchers who are currently developing new computational models or new features for traditional computational models, in order to foster their interaction, to provide a forum for presenting new ideas and work in progress, and to enable newcomers to learn about current activities in this area. Topics of interest include all abstract models of computation and their applications to the development of programming languages and systems. This includes (but is not limited to):

- Functional calculi: lambda-calculus, rho-calculus, term and graph rewriting;
- quantum computation, including implementations and formal methods in quantum protocols;
- probabilistic computation and verification in modelling situations;
- chemical, biological and bio-inspired computation, including spatial models, self-assembly, growth models;
- models of concurrency, including the treatment of mobility, trust, and security;
- infinitary models of computation;
- information-theoretic ideas in computing.

There were 6 submissions. Each submission was reviewed by at least 3 program committee members. The committee decided to accept 4 papers. The program also includes 3 invited talks by Mauricio Ayala-Rincón (Universidade de Brasília) on “Formalising confluence”, Gilles Dowek (INRIA) on “Discrete version of special and general relativistic trajectories”, and Pawel Sobocinski (University of Southampton) on “Compositional model checking of concurrent systems, with Petri nets”.

The organizers are grateful to the members of the programme committee and external reviewers for their careful work in the reviewing process. The organizers would also like to thank the general chair and local organizers of ICTAC 2015 for their constant support.

October 28, 2015
Cali, Colombia

César A. Muñoz
Jorge A. Pérez
# Program Committee

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mario Benevides</td>
<td>Universidade Federal do Rio de Janeiro, Brazil</td>
</tr>
<tr>
<td>Luís Caires</td>
<td>Universidade Nova de Lisboa, Portugal</td>
</tr>
<tr>
<td>Ugo Dal Lago</td>
<td>Università di Bologna, Italy</td>
</tr>
<tr>
<td>Nachum Dershowitz</td>
<td>Tel Aviv University, Israel</td>
</tr>
<tr>
<td>Jérôme Feret</td>
<td>INRIA &amp; Ecole normale supérieure, France</td>
</tr>
<tr>
<td>Marcelo Frias</td>
<td>Universidad de Buenos Aires, Argentina</td>
</tr>
<tr>
<td>Russ Harmer</td>
<td>CNRS &amp; ENS Lyon, France</td>
</tr>
<tr>
<td>Ivan Lanese</td>
<td>University of Bologna &amp; INRIA, France</td>
</tr>
<tr>
<td>Radu Mardare</td>
<td>Aalborg University, Denmark</td>
</tr>
<tr>
<td>Elvira Mayordomo</td>
<td>Universidad de Zaragoza, Spain</td>
</tr>
<tr>
<td>César Muñoz (Co-Chair)</td>
<td>NASA, US</td>
</tr>
<tr>
<td>Jorge A. Pérez (Co-Chair)</td>
<td>University of Groningen, The Netherlands</td>
</tr>
<tr>
<td>Andrés Sicard-Ramírez</td>
<td>EAFIT University, Colombia</td>
</tr>
<tr>
<td>Alexandra Silva</td>
<td>University College London, UK</td>
</tr>
<tr>
<td>Daniele Varacca</td>
<td>Université Paris Est Créteil, France</td>
</tr>
</tbody>
</table>

# Additional Reviewers

- Xue, Bingtian
Table of Contents

Formalising Confluence................................................................. 1
  Mauricio Ayala-Rincon

Discrete version of special and general relativistic trajectories............... 2
  Gilles Dowek

Compositional model checking of concurrent systems, with Petri nets........ 3
  Pawel Sobocinski

A New Encoding for Efficient Generation, Ranking and Unranking of Semi-Chemical Trees 4
  Mahdi Amani and Abbas Nowzari-Dalini

A knowledge representation meta-model for rule-based modelling of signalling networks... 16
  Adrien Basso-Blandin, Walter Fontana and Russ Harmer

Finiteness and Computation in Toposes........................................... 27
  Edward Hermann Haeusler

Mean Field Semantics for a Process Calculus for Spatially-Explicit Ecological Models..... 39
  Mauricio Toro-Bermudez, Anna Philippou, Sair Arboleda, Maria Puerta and Carlos Vélez
Formalising Confluence
(Invited Talk)

Mauricio Ayala-Rincón
Universidade de Brasília, Brazil

Abstract

Confluence is a critical property of computational systems which is related with determinism and non ambiguity and thus with other relevant computational attributes of functional specifications and rewriting system as termination and completion. Several criteria have been explored that guarantee confluence and their formalisations provide further interesting information. This talk will discuss topics related with the formalisation of confluence properties in the prototype verification system PVS.

Syntactic criteria such as avoiding overlapping of rules as well as linearity of rules have been used as a discipline of functional programming which avoids ambiguity. In the context of term rewriting systems, well-known results such as Newman’s Lemma [7], Rosen’s Confluence of Orthogonal term rewriting systems [9] as well as the famous Knut-Bendix(-Huet) Critical Pair Theorem [6, 5] are of great theoretical and practical relevance. The first one, guarantees confluence of Noetherian and locally confluent abstract reduction systems; the second one, assures confluence of orthogonal term rewriting systems, that are systems which avoid ambiguities generated by overlapping of their rules and whose rules do not allow repetitions of variables in their left-hand side (i.e., left-linear); and, the third one provides local confluence of term rewriting systems whose critical pairs are joinable.

Formalisations of these confluence criteria provide valuable and precise data about the theory of rewriting (cf. [4], [3], [8]). Several aspects that arise from these formalisations are of great relevance for the formal discussion about how these properties should be adequately ported to different computational contexts such as the nominal approach of rewriting (cf. [1] [2]).

References

Discrete version of special and general relativistic trajectories (Invited Talk)

Gilles Dowek*

INRIA, France

Abstract

If we assume information has a bounded density and a bounded velocity, physical phenomena should have a description in terms of cellular automata. In this talk, I will describe several attempts to describe the trajectories of bodies in Newtonian mechanics, special relativity, and general relativity with cellular automata. Interestingly, the local nature of relativity make it more amenable to a description in terms of cellular automata than Newtonian mechanics.

*Joint work with Pablo Arrighi
Compositional model checking of concurrent systems, with Petri nets
(Invited Talk)

Pawel Sobocinski*

University of Southampton, United Kingdom

Abstract

Compositionality and process equivalence are both standard concepts of process algebra. Compositionality means that the behaviour of a compound system relies only on the behaviour of its components, i.e. there is no emergent behaviour. Process equivalence means that the explicit statespace of a system takes a back seat to its interaction patterns: the information that an environment can obtain through interaction. Petri nets are a classical, yet widely used and understood, model of concurrency. Nevertheless, they have often been described as a non-compositional model, and tools tend to deal with monolithic, globally-specified models. In this talk I will introduce Petri Nets with Boundaries (PNB), which is a compositional, graphical algebra of elementary net systems, an important class of Petri nets. I will show that compositionality and process equivalence are a powerful combination that can be harnessed to improve the performance of checking reachability and coverability in several common examples where Petri nets model realistic concurrent systems.

*Joint work with Julian Rathke and Owen Stephens
A New Encoding for Efficient Generation, Ranking and Unranking of Semi-Chemical Trees

Mahdi Amani1 and Abbas Nowzari-Dalini2

1 Dipartimento di Informatica, Università di Pisa, Pisa, Italy. m_amani@di.unipi.it
2 School of Mathematics, Statistics, and Computer Science Colleague of Science, University of Tehran, Tehran, Iran. nowzari@ut.ac.ir

Abstract

Unlabeled ordered trees whose nodes have maximum degree of ∆ represent a generalization of chemical trees. A chemical tree is an unlabeled tree in which no node has degree greater than 4. We generalize chemical trees by allowing up to ∆ children for each node instead of 4. We introduce a new encoding over an alphabet of size 4 for representing unlabeled ordered trees with maximum degree of ∆. We use this encoding for generating these trees in A-order with $O(1)$ average time and $O(n)$ worst case time complexities. Due to the given encoding, both ranking and unranking algorithms are also designed taking $O(n)$ and $O(n \log n)$ time complexities (with a precomputation of size and time $O(n^2)$).

1 Introduction

A labeled tree is a tree in which each node is given a unique label. A rooted tree is a tree in which one of the nodes is distinguished from the others as the root. An ordered tree or plane tree is a rooted tree for which an ordering is specified for the children of each node. Studying combinatorial properties of restricted trees and outerplanar graphs (e.g. ordered trees with bounded degrees) can be used for many purposes including virtual exploration of chemical universe, reconstruction of molecular structures from their signatures, and the inference of structures of chemical compounds [1, 5, 7, 8, 13, 12, 17]. In this paper we study the generation of unlabeled ordered trees whose nodes have maximum degree of ∆ which is a generalization of chemical trees, for the sake of simplicity, we denote this class of trees by $T^\Delta$, we also denote $T^\Delta_n$ trees with $n$ nodes by $T^\Delta_n$.

Chemical trees are the most similar trees to $T^\Delta$ trees. Chemical trees are the graph representations of alkanes [7], or more precisely, the carbon atom skeleton of the molecules of alkanes [13, 7, 6, 4, 3, 11]. The alkane molecular family is partitioned into classes of homologous molecules, that is molecules with the same numbers of carbonium and hydrogen atoms; the $n^{th}$ class is characterized by the formula $C_nH_{2n+2}$, $n = 1, 2, ..., 1$. An alkane molecule is usually represented by indicating the carbonium atoms and their (primary) links, omitting to represent hydrogen atoms [1], therefore all the nodes would have the same label; carbon (i.e., the tree is unlabeled). A chemical tree is defined as a tree in which no node has degree greater than 4 [13, 7, 6, 4, 3, 11], chemical trees are also considered to be unlabeled [6, 4, 3, 11]. Therefore, $T^\Delta$ tree can be considered as a generalization of chemical trees to unlabeled ordered trees whose nodes have maximum degree of ∆ instead of 4.

Generation, ranking and unranking of trees are very basic problems in computer science and discrete mathematics [16]. The generation problem of combinatorial structures is to construct all the possible structures of a particular kind in a certain order [10]. Any generation algorithm
imposes an ordering on the set of trees. Classical orderings on trees are \(A\)-order and \(B\)-order which are defined as follows [14, 15, 16].

**Definition 1.** Let \(T\) and \(T'\) be two trees in \(T^\Delta\) and \(k = \max\{\deg(T), \deg(T')\}\), we say that \(T\) is less than \(T'\) in \(A\)-order \((T \prec_A T')\), iff

- \(|T| < |T'|\), or
- \(|T| = |T'|\) and for some \(1 \leq i \leq k\), \(T_j =_A T'_j\) for all \(j = 1, 2, \ldots, i - 1\) and \(T_i \prec_A T'_i\).

Where \(|T|\) is the number of nodes in the tree \(T\).

**Definition 2.** Let \(T\) and \(T'\) be two trees in \(T^\Delta\) and \(k = \max\{\deg(T), \deg(T')\}\), we say that \(T\) is less than \(T'\) in \(B\)-order \((T \prec_B T')\), iff

- \(\deg(T) < \deg(T')\), or
- \(\deg(T) = \deg(T')\) and for some \(1 \leq i \leq k\), \(T_j =_B T'_j\) for all \(j = 1, 2, \ldots, i - 1\) and \(T_i \prec_B T'_i\).

Where \(\deg(T)\) is the degree of the root of \(T\).

For a given set of trees, with respect to the given ordering, the next function gives the successor tree of a given tree \(T\) and the generation algorithm generates them all. The position of tree \(T\) in given set of trees is called rank, the rank function determines the rank of \(T\); the inverse operation of ranking is unranking, for a position \(r\), the unrank function gives the tree \(T\) corresponding to this position [14, 15, 16].

Related to the generation of semi-chemical trees, Hendrickson and Parks in [9] investigated the enumeration and the generation of carbon skeletons which can have cycles and are not necessarily trees. The most related work to our paper is an algorithm for the generation of certain classes of trees such as chemical trees in [2] with no ranking or unranking algorithm and redundant generations of trees. In [17], a generation algorithm with constant average delay time but with no ranking or unranking algorithms was given for all unrooted trees of \(n\) nodes and a diameter at least \(d\) such that the degree of each vertex with distance \(k\) from the center of the tree is at most by a capacity function. Up to now, to our knowledge, apparently, no efficient generation, ranking or unranking algorithms are known for either ‘chemical trees’ or ‘ordered trees with bounded degrees’.

In most of the tree generation algorithms, a tree is represented by an integer or alphabet sequence called codeword, hence all possible sequences of this representation are generated. This operation is called tree encoding. In section 2, we present a new encoding for \(T^\Delta_n\) trees. Our generation algorithm, which is given in Section 3, produces the sequences corresponding to \(T^\Delta_n\) trees in \(A\)-order, and in section 4, we study the corresponding ranking and unranking algorithms.

### 2 The encoding schema

The main point in generating trees is to choose a suitable encoding to represent them, and generate their corresponding codewords instead. Regarding the properties of \(T^\Delta_n\), we present our new encoding. For any \(T^\Delta_n\) tree \(T\), the encoding over 4 letters \(\{s, \ell, m, r\}\) is defined as follows. The root of \(T\) is labeled by \(s\), and for any internal node, if it has only one child, that child is labeled by \(s\), otherwise the leftmost child is labeled by \(\ell\), and the rightmost child is labeled by \(r\), and the children between the leftmost and the rightmost children (if exist) are all labeled by \(m\). Nodes are labeled in the same way for any internal node in each level recursively,
Figure 1: An example of a tree $T \in T_{\Delta}^n$ (for $\Delta \geq 4$). Its codeword is “$s\ell s\ell m\ell s\ell m\ell r$”.

and by a pre-order traversal of $T$ we obtain the codeword. This labeling is illustrated in Figure 1. Note that the 4-letters alphabet codeword corresponding to the first and last $T_{\Delta}^n$ trees in A-order are respectively “$s\ell m\ell^{-2} r\ell m\ell^{-2} r \ldots \ell m^{(n \mod \Delta)^{-2} r}$” and “$s^n$” which are shown in the Figure 2-a and Figure 2-b. Now, we prove the validity of this encoding for $T_{\Delta}^n$ trees (one-to-one correspondence).

**Definition 3.** Suppose that \( \{s, \ell, m, r\}^* \) is the set of all sequences with alphabet of \( s, m, \ell, r \) and let \( A \) be a proper subset of \( \{s, \ell, m, r\}^* \), then we call the set \( A \) a CodeSet$^\Delta$ iff \( A \) satisfies the following properties:

1. \( \epsilon \in A \) (\( \epsilon \) is a string of length 0),
2. \( \forall x \in A : sx \in A \),
3. \( \forall x_1, x_2, \ldots, x_i \in A \), and \( 2 \leq i \leq \Delta : \ell x_1 m x_2 m x_3 \ldots m x_{i-1} r x_i \in A \).

Now we show that a valid codeword is obtained by the concatenation of the character $s$ and each element of CodeSet$^\Delta$.

**Theorem 1.** Let \( A \) be the “CodeSet$^\Delta$” and \( \delta \) be a string such that \( \delta \in A \) and \( C \) be a codeword obtained by the concatenation of the character $s$ and $\delta$ (we show it by $s\delta$). There is a one-to-one correspondence between \( C \) and a unique $T^\Delta$ tree.
Proof. It can be proved by induction on the length of $C$. Initially for a codeword of length equal to 1, the proof is trivial. Assume that any codeword obtained in the above manner with length less than $n$ encodes a unique $T^\Delta$ tree. For a given codeword with length $n$, because of that concatenation of $s$ and $\delta$, we have:

1. $C = sx$, such that $x \in A$, or
2. $C = s\ell x_1 m x_2 \ldots m x_{j-1} r x_j$, such that $x_i \in A, \forall 1 \leq i \leq j \leq \Delta$.

For the first case by induction hypothesis, $x$ is a valid codeword of a $T^\Delta$ tree $T$; therefore $sx$ is another codeword corresponding to a $T^\Delta$ tree by adding a new root to the top of $T$. This tree is shown in Figure 3-a. For the second case, by induction hypothesis and that concatenation of $s$ and $\delta$, each $sx_i$ for $1 \leq i \leq j$ is a valid codeword for a $T^\Delta$ tree, therefore with replacement of ‘$s$ with $\ell$ in $sx_1$’ and ‘$s$ with $m$ in $sx_i$ for $2 \leq i \leq j-1$’ and finally ‘$s$ with $r$ in $sx_j$’ we can produce $\ell x_1, m x_2, \ldots, m x_{j-1}, r x_j$ codewords. Now they all are subtrees of a $T^\Delta$ tree whose codeword is $C = s\ell x_1 m x_2 \ldots m x_{j-1} r x_j$ (add a new root and connect it to each one of them). This tree is shown in Figure 3-b.

In a similar manner, by another induction we can prove that each $T^\Delta$ tree has a unique code in the form $C = s\ell x_1 m x_2 \ldots m x_{j-1} r x_j$. Therefore one-to-one correspondence between codewords and $T^\Delta$ trees holds.

For a $T^\Delta_n$ tree, this encoding needs only 4 alphabet letters and has a length of $n$. This encoding is simple and powerful, so it can be used for many other applications besides the generation algorithm.

3 The generation algorithm

For generating the successor of a given codeword $C$ corresponding to a $T^\Delta_n$ tree $T$, the codeword $C$ is scanned from right to left, corresponding to a reverse pre-order traversal of $T$. First we describe how this algorithm works directly on $T$, then we present the algorithm for generating the successor of $C$. For generating the successor of a given $T^\Delta_n$ tree $T$ we traverse the tree in reverse pre-order as follows.

1. Let $v$ be the last node of $T$ in pre-order traversal.
2. If $v$ doesn’t have any brothers, then
   - repeat $\{v = \text{parent of } v.\}$
     until $v$ has at least one brother or $v$ be the root of tree $T$.
   - If $v = \text{root}$, then the tree is the last tree in A-order and there is no successor.
3. If $v$ has at least one brother (obviously it has to be a left brother), delete one node from the subtree of $v$ and insert this node into its left brother’s subtree, then rebuild both subtrees (each one as a first tree with corresponding nodes in A-order).

The pseudo code of this algorithm for codewords corresponding to $T_n^\Delta$ trees is presented in Figure 4. In this algorithm, $C$ is a global array of characters holding the codeword (the algorithm generates the successor sequence of this codeword), $n$ shows the size of the codeword (the number of nodes of the tree corresponded to $C$), $STsize$ is a variable contains the size of the subtree rooted by node corresponded to $C[i]$ and $STNum$ holds the number of consecutive visited $s$ characters. This algorithm also calls two functions $updateChildren(i, ChNum)$ presented in Figure 5, and $updateBrothers(i, ChNum)$ presented in Figure 6. The procedure $updateChildren(i, ChNum)$ regenerates the codeword corresponding to the children of an updated node and the procedure $updateBrothers(i, ChNum)$ also regenerates the codeword corresponding to the brothers of a node with regard to the maximum degree $\Delta$ for each node. In these algorithms, $C$ is a global array of characters holding the codeword, $i$ is the position of the current node in the array $C$, $ChNum$ is the number of children/brothers of $C[i]$ to regenerate the corresponding codeword and $NChild$ is a global array which $NChild[i]$ holds the number of left brothers of node corresponding to $C[i]$ plus one.

**Theorem 2.** The algorithm Next presented in Figure 4 has a worst case time complexity of $O(n)$ and an average time complexity of $O(1)$ (without considering the input or the output time).

*Proof.* The worst case time complexity of this algorithm is $O(n)$ because the sequence is scanned just once. For computing the average time, it should be noted that during the scanning process, every time we visit the characters $s$, $m$ or $\ell$, the algorithm will terminate, so we define $S_n^\Delta$ as the number of codewords of $T_n^\Delta$ trees whose the last character $s$, $m$ or $\ell$ has distance $i$ from the end, and $S_n^{\Delta}$ as the total number of $T_n^\Delta$ trees. Obviously we have:

$$S_n^{\Delta} = \sum_{i=1}^{n} S_i^{\Delta}. \quad (1)$$

We define $H_n$ as the average time of generating all codewords of $T_n^\Delta$ trees,

$$H_n \leq \frac{k}{S_n^{\Delta}} \sum_{i=1}^{n} i S_i^{\Delta},$$

$$\leq \frac{k}{S_n^{\Delta}} \sum_{j=1}^{n} \sum_{i=j}^{n} S_i^{\Delta}.$$

Where $k$ is a constant value. On the other hand, consider that for $S_j^{n+1,\Delta}$ we have two cases, in the first case, the last character $s$, $m$ or $\ell$ is a leaf and in the second one, it is not. Therefore $S_j^{n+1,\Delta}$ is greater than or equal to just the first case, and in that case by removing the node corresponding to the ‘last character $s$, $m$ or $\ell$ of the codeword’, the remaining tree will have a corresponding codeword belongs to exactly one of $S_k^{n,\Delta}$ cases, for $j \leq k \leq n$. By substituting $k$ and $i$ we have:

$$S_j^{n+1,\Delta} \geq \sum_{i=j}^{n} S_i^{n,\Delta}.$$

Therefore for $H_n$ we have:

$$H_n \leq \frac{k}{S_n^{\Delta}} \sum_{j=1}^{n} S_j^{n+1,\Delta},$$

then by using Equation (1),

$$H_n \leq k S_n^{n+1,\Delta}/S_n^{\Delta}.$$

Finally from [16] we know that the total number of ordered trees is growing same as Catalan number, while $T_n^\Delta$ is a subset of ordered trees can not grow faster than that, this guarantees that for large enough values of $n$, $S_n^{n+1,\Delta}/S_n^{\Delta} = O(1)$. Therefore, $H_n \leq kO(1) = O(1)$. 

\qed
Function AOrder-Next(n : integer);
var i, Current, STsize, SNum: integer; finished, RDeleted: boolean;
begin
Current := n; STSize := 0; RDeleted := false; finished := false;
while ( (C[Current] = 's') & (Current $\geq$ 1) ) do
STSize := STSize + 1; Current := Current - 1;
if (Current = 0) then return (no successor);
while (not finished) do
begin
STSize := STSize + 1;
switch C[Current] of
case 'r':
i := Current - 1; SNum := 0;
while (C[i] = 's') do
SNum := SNum + 1; i := i - 1;
end;
if ( (C[i] = 'm') or (C[i] = 'l') ) then begin
if (STSize = 1) then RDeleted := true;
if (STSize > 1) then begin
STSize := STSize - 1; updateBrothers(Current + 1, STSize);
Current := i; STSize := SNum + 1;
end;
end;
case 'm':
if (RDeleted = true) then C[Current] := 'r';
updateChildren(Current + 1, STSize - 1); finished := true;
case 'l':
if (RDeleted = true) then C[Current] := 's';
updateChildren(Current + 1, STSize - 1); finished := true;
end;
end:
Figure 4: Algorithm for generating the successor codeword for $T_{\Delta}^n$ trees in A-order.

4 Ranking and Unranking algorithms

Ranking and unranking algorithms usually use exact formula (if exists any) or a precomputed
table of number of a subclass of given trees with some specified properties to achieve efficient
time complexities; these precomputations will be done only once and stored in a table for further use. Let $S_{\Delta}^n$ be the number of $T_{\Delta}^n$ trees, $S_{m,\Delta}^n$ be the number of $T_{\Delta}^n$ trees whose first subtree has exactly $m$ nodes and its root has maximum degree of $d$, and $D_{m,d}^{\Delta}$ be the number of $T_{\Delta}^n$ trees whose first subtree has at most $m$ nodes and its root has maximum degree of $d$.

Theorem 3.

• $D_{m,d}^{\Delta} = \sum_{i=1}^{m} S_{i,d}^{\Delta}$. 

procedure updateChildren( i, ChNum : integer);
begin
while (ChNum > 0) do begin
  if ChNum = 1 then begin
    C[i] := 's'; NChild[i] := 1; i ++; ChNum --;
  end;
  if ChNum > 1 then begin
    C[i] := 'l'; NChild[i] := NChild[i - 1] + 1; i ++; ChNum --;
    while ( (NChild[i] < (∆ - 1)) & (ChNum > 1) ) do begin
      C[i] := 'm'; NChild[i] := NChild[i - 1] + 1; i ++; ChNum --;
    end;
    C[i] := 'r'; NChild[i] := NChild[i - 1] + 1; i ++; ChNum --;
  end
end;
end;

Figure 5: Algorithm for updating the children.

Procedure updateBrothers( i, ChNum : integer);
begin
  if ChNum = 1 then begin
    C[i] := 'r'; NChild[i] := NChild[i - 1]; ChNum --;
  end
  if ChNum > 1 then begin
    C[i] := 'm'; ChNum --; i ++;
    while ( (NChild[i] < (∆ - 1)) & (ChNum > 1) ) do begin
      C[i] := 'm'; NChild[i] := NChild[i - 1] + 1; i ++; ChNum --;
    end;
    C[i] := 'r'; NChild[i] := NChild[i - 1] + 1; i ++; ChNum --; updateChildren(i, ChNum);
  end;
end;

Figure 6: Algorithm for updating the neighbors.

- $S_{\Delta}^{n} = \sum_{i=1}^{n-1} S_{\Delta}^{n,i}$.

Proof. The proof is trivial.

Theorem 4.

$$S_{m,d}^{\Delta} = S_{m,1}^{\Delta+1} \times \sum_{i=1}^{n-m-1} (S_{i,d-1}^{n-m,\Delta}).$$

Proof. Let $T$ be a $T_{\Delta}^{n}$ tree whose first subtree has exactly $m$ nodes and its root has maximum degree of $d$; by the definition and as shown in the Figure 7 the number of the possible cases for the first subtree is $S_{m,1}^{\Delta+1}$ and the number of cases for the other parts of the tree is: $\sum_{i=1}^{n-m-1} (S_{i,d-1}^{n-m,\Delta})$. So:

$$S_{m,d}^{\Delta} = S_{m,1}^{\Delta+1} \times \sum_{i=1}^{n-m-1} (S_{i,d-1}^{n-m,\Delta}).$$
Figure 7: $T_n^\Delta$ tree whose first subtree has exactly $m$ nodes and its root has a maximum degree of $d$.

Now, let $T$ be a $T_n^\Delta$ tree whose subtrees are defined by $T_1, T_2, \ldots, T_k$ and for $1 \leq i \leq k \leq \Delta$ : $|T_i| = n_i$ and $\sum_{i=1}^{k} n_i = n - 1$. For computing the rank of $T$, we have to enumerate the number of trees generated before $T$. Let $\text{Rank}(T, n)$ be the rank of $T$. The number of $T_n^\Delta$ trees whose first subtree is smaller than $T_1$ is equal to:

$$\sum_{i=1}^{n_1-1} S_{i, \Delta}^{n_1} + (\text{Rank}(T_1, n_1) - 1) \times \sum_{i=1}^{n-n_1} S_{i, \Delta-1}^{n-n_1},$$

and the number of $T_n^\Delta$ trees whose first subtree is equal to $T_1$ but the second subtree is smaller than $T_2$ is equal to:

$$\sum_{i=1}^{n_2-1} S_{i, \Delta-1}^{n-n_1} + (\text{Rank}(T_2, n_2) - 1) \times \sum_{i=1}^{n-n_1-n_2} S_{i, \Delta-2}^{n-n_1-n_2}.$$

Similarly, the number of $T_n^\Delta$ trees whose first $(j-1)$ subtrees are equal to $T_1, T_2, \ldots, T_{j-1}$ and the $j^{th}$ subtree is smaller than $T_j$ is equal to:

$$\sum_{i=1}^{n_j-1} S_{i, \Delta-j+1}^{n-(\sum_{\ell=1}^{j-1} n_\ell)} + (\text{Rank}(T_j, n_j) - 1) \times \sum_{i=1}^{n-n_1-n_2-\sum_{\ell=1}^{j-1} n_\ell} S_{i, \Delta-\sum_{\ell=1}^{j-1} n_\ell}^{n-n_1-n_2-\sum_{\ell=1}^{j-1} n_\ell}.$$

Therefore, regarding enumerations explained above, for given tree $T \in T_n^\Delta$ whose subtrees are defined by $T_1, T_2, \ldots, T_k$, we can write:

$$\text{Rank}(T, 1) = 1,$$

$$\text{Rank}(T, n) = 1 + \sum_{j=1}^{k} \left( \sum_{i=1}^{n_j-1} S_{i, \Delta-j+1}^{n-(\sum_{\ell=1}^{j-1} n_\ell)} \right) +$$

+ $$(\text{Rank}(T_j, n_j) - 1) \sum_{i=1}^{n-n_1-n_2-\sum_{\ell=1}^{j-1} n_\ell} S_{i, \Delta-\sum_{\ell=1}^{j-1} n_\ell}^{n-n_1-n_2-\sum_{\ell=1}^{j-1} n_\ell}.$$
Function Rank( Beg: integer; var Fin: integer) :
Var R, Point, PointFin, j, Nodes, n: integer;
begin
n := N[Beg];
if (n = 1) then begin
Fin := Beg; return(1) end;
else begin
Point := Beg + 1; R := 0; Nodes := 0; j := 1;
while (Nodes < n) do begin
R := R + D[n – Nodes, N[Point] – 1, ∆ – j + 1] +
(Rank(Point, PointFin) – 1) ×
D[(n – Nodes – N[Point]), (n – Nodes – N[Point]), ∆ – j];
Nodes := Nodes + N[Point]; j := j + 1;
Point := PointFin + 1;
end;
Fin := Point – 1;
return( R + 1); end;
end;

Figure 8: Ranking algorithm for $T_n^\Delta$ trees.

Hence, from Theorem 3, by using $D_{m,d}^{n,\Delta} = \sum_{i=1}^{m} S_{i,d}^{n,\Delta}$, we have:

$$
\text{Rank}(T, 1) = 1,
$$

$$
\text{Rank}(T, n) = 1 + \sum_{j=1}^{k} (D^{(n-\sum_{i=1}^{j-1} n_i,\Delta)}_{(n-\sum_{i=1}^{j-1} n_i,\Delta-j+1)} +
(Rank(T, n_j) – 1) D^{(n-\sum_{i=1}^{j-1} n_i,\Delta)}_{(n-\sum_{i=1}^{j-1} n_i,\Delta-j)}).
$$

To achieve the most efficient time for ranking and unranking algorithms, we need to pre-compute $D_{m,d}^{n,\Delta}$ and store it for further use. Assuming $\Delta$ is constant, to store $D_{m,d}^{n,\Delta}$ values, a 3-dimensional table denoted by $D[n,m,d]$ is enough, this table will have a size of $O(n \times n \times \Delta) = O(n^2)$ and can be computed using Theorems 3 and 4 with time complexity of $O(n \times n \times \Delta) = O(n^2)$.

To compute the rank of a codeword stored in array $C$, we also need an auxiliary array $N[i]$ which keeps the number of nodes in the subtree whose root is labeled by $C[i]$ and corresponds to $n_i$ in the above formula. This array can be computed by a pre-order traversal or a depth first search (DFS) algorithm just before we call the ranking algorithm.

The pseudo code for ranking algorithm is given in Figure 8. In this algorithm, $Beg$ is the variable that shows the positions of the first character in the array $C$ whose rank is being computed ($Beg$ is initially set to 1), and $Fin$ is the variable that returns the position of the last character of $C$.

**Theorem 5.** The ranking algorithm has the time complexity of $O(n)$.

**Proof.** Let $T$ be a $T_n^\Delta$ tree whose subtrees are defined by $T_1$, $T_2$, . . . , $T_k$ and for $1 \leq i \leq k \leq \Delta$: $|T_i| = n_i$ and $\sum_{i=1}^{k} n_i = n – 1$, and let $T(n)$ be the time complexity of ranking algorithm,
then we can write:

\[ T(n) = T(n_1) + T(n_2) + \ldots + T(n_k) + \alpha_k, \]

where \( \alpha \) is a constant and \( \alpha_k \) is the time complexity of the non-recursive parts of the algorithm. By using induction, we prove that if \( \beta \) is a value greater than \( \alpha \) then \( T(n) \leq \beta n \). We have \( T(1) \leq \beta \). We assume \( T(m) \leq \beta (m-1) \) for each \( m < n \), therefore:

\[
\begin{align*}
T(n) & \leq \beta(n_1 - 1) + \beta(n_2 - 1) + \ldots + \beta(n_k - 1) + \alpha_k, \\
T(n) & \leq \beta(n_1 + \ldots + n_k - k) + \alpha_k, \\
T(n) & \leq \beta n - \beta k + \alpha k, \\
T(n) & \leq \beta n.
\end{align*}
\]

So the induction is complete and we have \( T(n) \leq \beta n = O(n) \). \( \square \)

Before giving the description of the unranking algorithm we need to define two new operators.

- If \( a \) and \( b \) are integer numbers then \( a \text{ div}^+ b \) is defined as follows:
  - If \( b \nmid a \) then \( a \text{ div}^+ b \) is equal to \( (a \text{ div } b) \).
  - If \( b \mid a \) then \( a \text{ div}^+ b \) is equal to \( (a \text{ div } b) - 1 \).

- If \( a \) and \( b \) are integer numbers then \( a \text{ mod}^+ b \) is defined as follows:
  - If \( b \nmid a \) then \( a \text{ mod}^+ b \) is equal to \( (a \text{ mod } b) \).
  - If \( b \mid a \) then \( a \text{ mod}^+ b \) is equal to \( b \).

After designing the ranking algorithm, the unranking algorithm is just the reverse manner of ranking algorithm, this algorithm is given in Figure 9 with \( O(n \log n) \) time complexity, for lack of space the proof is omitted. For unranking algorithm, we need the values of \( S^n, \Delta \), these values can be stored in an array of size \( n \), denoted by \( S[n] \) (we assume \( \Delta \) is constant). In this algorithm, the rank \( R \) is the input, \( \text{ Beg } \) is the variable to show the position of the first character in the global array \( C \) and initially is set to 1. The generated codeword will be stored in array \( C \). The variable \( n \) is the number of nodes and \( \text{ Root } \) stores the character corresponding to the node we consider for the unranking procedure. For the next character we have two possibilities. If the root is \( r \) or \( s \) then the next character, if exists, will be \( \ell \) or \( s \) (based on the number of root’s children). If the root is \( m \) or \( \ell \), we have again two possible cases: if all the nodes of the current tree are not produced then the next character is \( m \) otherwise the next character will be \( r \).

5 Conclusion

We have studied the problem of generation, ranking and unranking of ordered trees of size \( n \) and maximum degree \( \Delta \); we presented an efficient algorithm for generating these trees in A-order with an encoding over 4 letters and size \( n \). Also two efficient ranking and unranking algorithms were designed for this encoding. The generation algorithm has \( O(n) \) time complexity in worst case and \( O(1) \) in average case. The ranking and unranking algorithms have \( O(n) \) and \( O(n \log n) \) time complexity, respectively. The presented ranking and unranking algorithms use a precomputed table of size \( O(n^2) \) (assuming \( \Delta \) is constant). For the future works, generating this class of trees in B-order and minimal change ordering and an explicit enumeration, are major unsolved problems.
Function UnRank (R, Beg, n: integer; Root: char);
var Point, i, t, ChildNum: integer;
begin
  if (n = 0) or (R = 0) then return(Beg - 1)
  else begin
    if (n = 1) then begin
      C[Beg] := Root; return(Beg);
    end;
    else begin
      C[Beg] := Root; Point := Beg + 1;
      Root := 'r'; ChildNum := 0;
      while (n > 0) do begin
        ChildNum := ChildNum + 1;
        find the smallest i that D[n, i, \Delta - ChildNum + 1] \geq R;
        R := R - D[n, i - 1, \Delta - ChildNum + 1];
        if (n - i) = 1 then
          if (ChildNum = 1) then Root := 's';
          else Root := 't';
        t := S[n];
        Point := UnRank(\(\text{div}^+(R, t)\)) + 1, Point, i, Root + 1;
        R := \text{mod}^+(R, t);
        n := n - i; Root := 'm';
      end;
      return(Point - 1);
    end;
  end;
end

Figure 9: The pseudo code of unranking algorithm.

References


A knowledge representation meta-model for rule-based modelling of signalling networks

Adrien Basso-Blandin\textsuperscript{1}, Walter Fontana\textsuperscript{2}, and Russ Harmer\textsuperscript{3}

\textsuperscript{1} LIP, ENS Lyon, Lyon, France
\texttt{adrien.basso-blandin@ens-lyon.fr}

\textsuperscript{2} Harvard Medical School, Boston, USA
\texttt{walter@hms.harvard.edu}

\textsuperscript{3} CNRS & LIP, ENS Lyon, Lyon, France
\texttt{russell.harmer@ens-lyon.fr}

Abstract

The study of cellular signalling pathways and their deregulation in disease states, such as cancer, is a large and extremely complex task. Indeed, these systems involve many parts and processes but are studied piecewise and their literatures and data are consequently fragmented, distributed and sometimes—at least apparently—inconsistent. This makes it extremely difficult to build significant explanatory models with the result that effects in these systems that are brought about by many interacting factors are poorly understood.

The rule-based approach to modelling has shown some promise for the representation of the highly combinatorial systems typically found in signalling where many of the proteins are composed of multiple binding domains, capable of simultaneous interactions, and/or peptide motifs controlled by post-translational modifications. However, the rule-based approach requires highly detailed information about the precise conditions for each and every interaction which is rarely available from any one single source. Rather, these conditions must be painstakingly inferred and curated, by hand, from information contained in many papers—each of which contains only part of the story.

In this paper, we introduce a graph-based meta-model, attuned to the representation of cellular signalling networks, which aims to ease this massive cognitive burden on the rule-based curation process. This meta-model is a generalization of that used by Kappa and BNGL which allows for the flexible representation of knowledge at various levels of granularity. In particular, it allows us to deal with information which has either too little, or too much, detail with respect to the strict rule-based meta-model. Our approach provides a basis for the gradual aggregation of fragmented biological knowledge extracted from the literature into an instance of the meta-model from which we can define an automated translation into executable Kappa programs.

1 Introduction

We propose a knowledge representation (KR) meta-model to enable the study of the dynamics of cellular signalling networks and, in particular, the consequences of mutations on dynamics. Our aim is therefore not to construct a Description Logic-based terminology (or any other ontology of this general kind) of static concepts to perform inference of the kind “ERK is an enzyme that phosphorylates so it is a kinase”; nor is it to build a representation of dynamics to support inference about the time-evolution of systems. Rather, we seek to represent each individual protein-protein interaction (PPI) that constitutes a signalling network as a formal rule that expresses the known, empirically necessary conditions for that PPI to occur.
These rules resemble those of Kappa or BNGL [11] but need not respect the stringent meta-model imposed by those formalisms wherein all bonds must occur between explicitly specified sites and all other relevant factors—such as protein conformation, post-translational modifications (PTMs) or, more generally, the presence or absence of key residues—are opaquely encoded into monolithic states attached to sites. In a second step, these rules can be automatically assembled into \textit{bona fide} rule-based models that can be simulated, subjected to static analysis and whose causal structure can be examined in detail.

The need for such a two-stage approach to the rule-based modelling of signalling network-sarises because the pertinent information is dispersed across the literature in such a way that any given paper typically contains fragments, or \textit{nuggets}, of partial mechanistic knowledge about multiple PPIs; and nuggets appear, for any given PPI, in many papers. Such a situation inevitably lends itself to a curation process focussed not directly on the PPIs themselves—as must any manual curation of rules—but rather on extracting nuggets, identifying which PPIs they refer to and incrementally aggregating them into more detailed nuggets.

A further novelty of our approach lies in our notion of agent which we take to represent not a single gene product but a \textit{neighbourhood in sequence space}. This design decision is central to our approach and enables our KR to represent interactions that depend on certain key residues in such a way that the effects of both loss- and gain-of-function mutations can be automatically determined in the process of assembly to Kappa rules. This resolves a dangling question from earlier work on ‘meta Kappa’ [5, 10] which was only able to represent loss-, but not gain-, of-function mutations.

It is instructive to compare our approach with that embodied by production rule-based expert systems, such as MYCIN [1] and its descendants, which saw a gradual drift away from considering each rule as an independent element [as we do] towards a view where the KR should be “designed and maintained as a coherent whole” [2]. Such a viewpoint may be entirely appropriate in a domain where human experts can be reasonably expected to agree on most points; but, in a domain characterized by a large, dispersed and fragmented body of knowledge that no single human expert can hope to master, expert opinion can not reasonably be expected to converge towards a consensus.

As such, we advocate an approach where the KR does not seek to reproduce and augment human expert consensus but rather positions itself as a \textit{tool for discovery} which, starting from purely objective nuggets of knowledge, enables and aids the human expert to investigate—and hopefully resolve—areas of apparent, or real, incoherence by comparing the dynamic consequences of various collections of independently-conceived rules. In particular, our system does \textit{not} seek to figure out the ‘correct’ necessary conditions for a nugget and even less seeks to impose a pre-conceived structure on the Kappa model implied by the contents of the KR: there is no specification of what the system ‘should be’: indeed, the basic philosophy of our approach can be summarized as: taking as input partial empirical knowledge of a system; and producing as output the various consequences of this knowledge, \textit{e.g.} the necessary conditions for PPIs, the causal structure, or pathways, that the system contains, \textit{etc}. In a sense, from a computer science perspective, the workflow may seem backward: from partial knowledge of the behaviour of the system, we seek to \textit{determine} its specification. Our approach is thus intrinsically oriented towards \textit{systems}, not synthetic, biology which shares precisely this aim.

\textbf{Overview of the paper.} In section 2, we describe our graphical formalism, define our meta-model and formalize the notions of nuggets and their aggregation. In section 3, we illustrate these ideas with some simple examples. We conclude in section 4 with some remarks about our prototype implementation and directions for future work.
2 The graphical formalism

In this section, we first introduce a notion of (simple) graph with an additional forest structure on its nodes. This provides a more flexible and general starting point than in previous works [4,6,7] on ‘site graphs’ for rule-based modelling where the permitted kinds of nodes and edges were hard-wired. We then introduce a particular graph—our meta-model—which we use to type graphs so as to reintroduce the previously hard-wired constraints in a transparent fashion.

2.1 Site graphs

A site graph $G$ is defined by

- a finite set of nodes $N$ with a specified subset of states $S$;
- a forest structure $\mathcal{F}$, i.e. a single-valued binary relation whose transitive closure is irreflexive, on $N$ such that all states are leaves and no states are roots;
- a simple directed edge structure $\mathcal{E}$, i.e. a binary relation, on $N$;
- a function assigning, to each state $s \in S$, a set $V_s$ of possible values.

The forest structure allows us to formalize the notion that a node may ‘belong to’ another, *e.g.* a node representing a binding site of some protein belongs to the node representing the protein in question. A state represents some (fixed or variable) attribute of a node, *e.g.* the identity of an amino acid at a certain sequence location or the presence or absence of a post-translational modification like phosphorylation; we sometimes refer to states of the latter kind as flags. The requirement that states be leaves (and not roots) enforces the idea that a state belongs to the node for which it acts as an attribute.

The ‘site graph’ terminology comes from a line of work on the definition of the rule-based modelling language Kappa in terms of graph rewriting [4,6,7] where various more constrained variants of this style of definition have been used. The principal novelty of the present definition is to avoid hard-wiring the various kinds of nodes that can exist—agents, sites, &c.—and their hierarchical structure. Instead, it proposes a homogeneous space of nodes with a forest structure that can capture arbitrary hierarchies.

2.2 Homomorphisms

A homomorphism $h : G \rightarrow G'$ of site graphs is a function $h : N \rightarrow N'$ such that states are preserved, *i.e.* $h(S) \subseteq S'$, the forest and its roots are preserved, edges are preserved and values are preserved, *i.e.* $V_s \subseteq V'_{h(s)}$ for all states $s$.

Site graphs and homomorphisms form a category SGrph: a homomorphism is a mono if, and only if, its underlying node function is injective. The category SGrph has all pull-backs, all push-outs and all pull-back complements over monos. As such, it possesses all the structure required to support general sesqui-push-out rewriting [3]. The sub-category of monos has all multi-sums [9].

Given a fixed site graph $G$, the slice category over $G$, written SGrph$/G$, can be usefully thought of as the category of graphs typed by $G$. Standard categorical reasoning establishes that the slice category construction preserves all the above categorical structure. In particular, the hierarchical structure of $G$ and its edges constrain the objects of SGrph$/G$ so that we can reintroduce, in a flexible manner, the kinds of conditions and constraints that were hard-wired in previously proposed definitions of site graphs simply by selecting an appropriate base object $G$ over which to take the slice category.
In section 2.3, we make such a choice for our knowledge representation by defining a specific graph $M$ which types, in the slice category sense, all possible models that we wish to consider, i.e. $M$ is our meta-model. In section 2.4, we introduce *nuggets* precisely as the class of site graphs that exist in $\text{SGrph}/M$. However, the general procedure of defining a desired class of site graphs by a choice of base object $G$ could be applied in many other situations; it provides a unifying framework for discussing a broad class of related, but distinct, graphical formalisms.

### 2.3 The meta-model

We now introduce the concrete site graph $M$ that serves as our meta-model: it defines the various kinds of nodes that can exist, specifies which of them are states (and specifies the values they can take), defines the hierarchical structure on nodes and, finally, constrains the way that edges can be placed.

It is defined by the following graph where the forest structure is represented implicitly. The nodes labelled ‘agent’, ‘BND’ and ‘MOD’ are the roots; nodes with a ‘jagged’ outline are states.

![Graph Diagram](image)

We complete the definition by specifying the value sets of the states: $\text{agent\_flag}$, $\text{reg\_flag}$, $\text{res\_flag}$ and $\text{bnd}$ are assigned the set $\{0, 1\}$ of Booleans; $\text{loc}$, which identifies the position of a residue in a sequence, is assigned the set $\mathbb{Z}^+$ of positive integers; $\text{aa}$, which identifies amino acids, is assigned the standard 20-element set of one-letter amino acid codes, i.e. all letters of the alphabet except B, J, O, U, X and Z; and $\text{bnd\_rc}$, $\text{brk\_rc}$ and $\text{mod\_rc}$, which specify the rate constants of actions, are assigned the set $\mathbb{R}^+$ of positive reals.

Note that the names of nodes, such as ‘agent’ and ‘BND’, are not part of the formalism; they are just a convenient labelling for the purposes of discussion.

A typing $h : G \to M$ formally designates each of the nodes of $G$ as being either an agent, a region, a residue; or as one of the fixed attributes (aa, loc, etc.); or as a flag; or as a binding action (BND), unbinding action (BRK), modification action (MOD) or as a s(source) for a BND or MOD or as a t(target) for a BRK or MOD. The mapping of $G$ into $M$ further implies that the forest structure of $G$ must respect the restrictions imposed by $M$, i.e. that regions and residues belong to agents; that attributes and flags must be leaves; that sources and targets belong to their respective actions; and that unbinding actions belong to their corresponding binding actions. Finally, the edge structure of $G$ must also respect the restrictions imposed by $M$; this means that only agents and regions can engage in binding actions; and that only flags can be targeted by modification actions.
This meta-model provides the foundation for a rigorous ontology for the kinds of information that are pertinent to rule-based descriptions of signalling networks, *i.e.* proteins as agents, domains and other binding sites as regions, key amino acid locations as (key) residues, &c. However, the present framework remains purely formal and does not have any means to enforce correct semantic usage of this ontology; as such, we plan to augment our framework with a system of appropriate annotations in order to be able to carry out semantic checking and reasoning. We will return to this point later.

### 2.4 Nuggets

Let us now motivate the particular choice $\mathcal{M}$ of meta-model made in the previous section by considering a typical ‘nugget’ of knowledge in molecular biology: “EGFR binds the SH2 domain of Grb2 provided that EGFR is phosphorylated and residue 90 of Grb2 is a serine”. This would naturally be represented as the following site graph

![Site graph](image)

where the agent ‘EGFR’ has a state ‘phos’ (with value 1, meaning true) and the agent ‘Grb2’ has a region ‘SH2’ and a residue (with unimportant name) located at position 90 of the sequence and which is required to be a serine (the value S). Note that no region has been specified on EGFR; the meta-model explicitly allows for this by the fact that a BND action can have either an agent or a region as its source.

Formally speaking, a *nugget* is a connected graph $G$ typed by $\mathcal{M}$ in such a way that each node has at most one copy of each of its possible attributes, where all attributes and flags have a uniquely specified value and exactly one action node does not have a specified value for its bnd flag. This unique action is the *principal* action of the nugget; any other (necessarily BND) actions represent the required presence or absence of their corresponding bonds in order for the principal action to be possible. We further ask that any BND has exactly two sources, any BRK has exactly two targets and any MOD has at most one source and one target (and at least one of the two). Other than asking for $G$ to be connected, all of these constraints are motivated by domain-specific considerations and, in due course, will be handled via semantic checking.

Our representation language is thus a generalization of that typically used in rule-based modelling. The principal differences are that (i) the *action* of a rule is represented explicitly as a node in the graph; (ii) binding actions can act directly on agents rather than necessarily via sites/regions; and (iii) static attributes and dynamic flags are represented as values associated to persistent nodes rather than as mutually exclusive sets of nodes. We draw a distinction between flags and attributes in order to make an explicit difference between things that can be modified ‘in the system’, *e.g.* the phosphorylation status of some residue, and things that can only be modified ‘out of the system’, *e.g.* the identity of an amino acid that can only be modified by an external mutation event.
The latter points (ii) and (iii) are important in order to be able to represent biological knowledge as faithfully as possible: knowledge is often stated in a piecemeal and incomplete fashion but this should not prevent us from being able to formalize it. For example, when the site at one end of a bond is unknown, this can now be represented as is with no need—as there would have been in standard rule-based modelling—to create a ‘fictitious’ site. A second example could be the use of attributes to provide a transparent representation of detailed structural information, e.g. about key residues of a protein or cases where a binding interaction depends on multiple PTMs that can otherwise only be opaquely encoded. We will return to point (i) in section 2.6 where we will introduce the notion of nugget aggregation which depends critically on the explicit representation of actions as nodes.

2.5 Models

A site graph \( m : M \to \mathcal{M} \) is a pre-model of a collection \( C \) of nuggets \( n_i : N_i \to \mathcal{M} \) iff, for all \( i \), the arrow \( n_i \) factors through \( m \). In words, \( M \) is a graph, itself typed by the meta-model, which types all of the nuggets; indeed, \( M \) can be thought of as a summary statement of the collection \( C \) of nuggets. We refer to the pair \((C, M)\) as a model. Note that a given collection of nuggets may be assigned many different pre-models. The import of any particular choice is that it identifies which nodes in one nugget correspond to those in another: in the above example, we have a node labelled as ‘EGFR’; but that label does not exist in the formalism so, if we have a second nugget which also speaks of the same agent ‘EGFR’, we need a way to say that these two nodes are the same. The pre-model gives us precisely this possibility: the two agents are mapped to the same node of \( M \), i.e. nodes of \( M \) provide labels/names for the nodes of nuggets. This means that two different pre-models, \( M_1 \) and \( M_2 \), for the same collection \( C \) of nuggets can have completely different meanings; in particular, \( C \) and \( M \) provide two (uninteresting) extremes with interesting cases lying in between. In general, the two components of a model evolve together as we add more and more information; we discuss this briefly in section 2.6.

Let us note here that the necessity of a pre-model partially arises in order to enforce minimal semantic coherence in our formal framework. If we had semantic annotations that uniquely identify agents, we could potentially use them—instead of a pre-model—to solve the above cross-nugget identification problem. However, we have chosen to take a different approach so as to provide a more flexible notion of agent: in a general site graph—and, in particular, in a pre-model—the \( aa \) attribute of a residue may be assigned a set of one-letter codes in order to express the fact that an agent represents, in general, a neighbourhood in sequence space rather than a unique sequence. This flexibility affords us the possibility of organizing knowledge about minor variants of a protein using a single agent; this (rather prosaically) pre-empts the need to define, and name, lots of tedious variants but, more to the point, matches everyday practice in biology where, for example, (wild-type) ‘Ras\(_{WT}\)’ and (mutant) ‘Ras\(_{G12V}\)’ are both thought of as being ‘Ras’—they just differ in one or two small, although possibly very significant, ways.

As such, the notion of model is our first step towards a full semantic layer for our knowledge representation scheme: all semantic annotations will be made at the level of pre-models, not individual nuggets, so as to minimize the amount of needed annotation and, more importantly, to ease the maintenance of semantic coherence across the entire current collection of nuggets. It should be noted that this approach to grounding differs from more traditional approaches, such as that used by BioPAX [8], which insist upon each formal entity corresponding to a unique physical entity. Indeed, our approach is, by design, particularly attuned to the needs of representing signalling networks and, as such, is less constrained than BioPAX which, as a framework of far broader applicability, has to bear a far stronger semantic burden.
2.6 Aggregation

Suppose that, at some point in time, we have a model \((C, M)\) and that we now obtain a new nugget \(N'\). We always have the possibility simply to add \(N'\), yielding a new collection \(C'\) of nuggets; this might necessitate updating the underlying pre-model \(M\) to \(M'\) in the event that \(N'\) contains entirely novel nodes or edges.

If there was already a nugget in \(C\) that has the same action as that of \(N\), this would result in \((C', M')\) having two distinct actions involving the same agents, a situation that may or may not be desirable: sometimes, for example, two proteins can indeed bind each other in two distinct ways; however, it could also be the case that \(N'\) has actually brought some new information about a single binding interaction that we would rather use to update the pre-existing nugget \(N\). Such an update amounts to the assumption that the nuggets \(N\) and \(N'\) represent the same interaction mechanism; but this does not necessarily mean that the two nuggets refer to exactly the same agents since mechanisms can be shared across families of proteins. We will return to this point in section 3 after briefly describing the formal process of aggregation.

If we wish to update the nugget \(N\) with the information contained in \(N'\), we need to specify two things: the new information brought by \(N'\) and any deprecated information in \(N\) that should now be removed. The former is specified by the choice of a co-span of monos \(h_+ : N \rightarrow N_+ \leftarrow N' : h'_+\) from the multi-sum of \(N\) and \(N'\); while the latter is specified by a mono \(h_- : N_- \rightarrow N\). In most cases, there is a canonical choice of co-span given by the intuitive unification of \(N\) and \(N'\) but, in cases where there are non-trivial automorphisms of \(N\) and \(N'\), the unification process may be non-deterministic and a choice becomes necessary (or we allow multiple conflicting versions).

The pull-back complement of \(h_-\) and \(h_+\) defines a graph \(N_\pm\) containing precisely the new information from \(N'\) with all deprecated information from \(N\) removed. Formally, this is exactly a step of graph rewriting taking \(N\) to \(N_\pm\). In the event that nothing is to be removed from \(N\), i.e., \(h_-\) is the identity on \(N\), this rewriting step degenerates to being simply the refinement of \(N\) to \(N_+\) as specified by \(h_+\). This step of rewriting is also propagated to the pre-model \(M\), resulting in a new model \((C', M')\) where, unlike in the case of adding \(N'\), \(C'\) clearly has the same number of nuggets as \(C\).

3 Examples of aggregation

Nugget update  Consider updating the example nugget \(N\) of section 2.4 with the information contained in \(N'\):

\[
\text{grb2\_phos:1
BNDEGFR Grb2\_aa:Y
loc:1092}
\]

i.e. “EGFR binds Grb2 provided EGFR is phosphorylated on Y1092”.
If we choose not to deprecate anything from the original nugget, we obtain:

![Diagram of EGFR binding to Grb2 through BND]

If instead we were to specify that the `phos` flag from N is to be removed, we would obtain:

![Diagram of EGFR binding to Grb2 through BND]

**Nugget aggregation** Unification can also be partial: if we further update with

![Diagram of Shc binding to Grb2 through BND]

_i.e._ “tyrosine-phosphorylated Shc binds the SH2 domain of Grb2”
we obtain:

Note how the contextual conditions on Grb2 that occur in the original nugget are propagated, by the very process of aggregation, to its newly added interaction with Shc. This is a typical example of the use of our framework as a tool for discovery at the level of necessary conditions for PPIs which renders completely transparent the ‘by similarity’ style of reasoning which is ubiquitous in molecular biology.

Note that EGFR and Shc both target the same source of the (unique) BND node: this nugget has a disjunctive interpretation, giving rise to two distinct Kappa rules, and should be read (modulo the contextual conditions) as “EGFR or Shc binds the SH2 domain of Grb2”. The importance of this is that, in the translation to Kappa, agents are assigned one site for each BND action in which they participate: before aggregation, Grb2 would have been given two sites, one to bind EGFR and the other for Shc, with no conflict between the two generated Kappa rules; after aggregation, it would receive only one site, giving rise to an intrinsic conflict between the two generated rules.

4 Conclusion

We have presented a meta-model for aggregation of the kind of knowledge required to build rule-based models of cellular signalling networks. The framework provides a generalization of the usual strict rule-based meta-model and, in particular, represents the actions of rules explicitly as nodes. This enables the key notion of aggregation which serves as our source of biologically-plausible inference. The presentation is entirely mathematical, being framed in terms of a graph rewriting formalism, but can be considered as a specification for an actual system. We are currently building a prototype implementation of the framework, including the automatic translation of a collection of nuggets into bona fide Kappa, and will report on this in more detail in a full-length version of this paper.
Let us nonetheless discuss one important aspect of the translation: each BND node generates its own sites; so an agent with two incident BND actions will be translated into a Kappa agent (let’s call it A) with two sites. This is a consequence of our general philosophy that, in the absence of information to the contrary, we should draw the most general conclusions possible: were we to generate but a single site for that Kappa agent A, we would be unwarrantedly hard-wiring the constraint that its binding partners must compete in order to bind. On the other hand, in the event that we subsequently learn that both BND actions depend on some common residue of A, this would imply an intrinsic conflict between the two actions and would result in the automatic translation generating a single binding site, thus enforcing competitive binding to A. (This is a more general manifestation of the above example of nugget aggregation where the fusing of two actions gave rise to an analogous conflict.) In this way, we achieve a pragmatically (and cognitively) convenient separation of concerns whereby knowledge is integrated and aggregated as we learn it; while its eventual consequences for a future Kappa model are determined, at model generation time, by the automatic translation procedure.

After the automatic translation into Kappa, each concrete agent that is generated must have a unique value for each of its aa attributes; these manifest as unmodifiable states in the resulting Kappa. As such, a nugget that tests such an aa attribute will be translated into a Kappa rule (or rules) that will only ever apply to those concrete agents that have the appropriate state. As such, a rule that tests for a wild-type value of such an attribute would only apply to a concrete agent that is not mutated at that residue; conversely, a rule that tests for a non-wild-type value would not apply to a concrete agent unless it had undergone an appropriate mutation. This therefore enables a transparent account of both loss- and gain-of-function mutations that sidesteps the various difficulties (concerning gain-of-function mutations) that were encountered by the original ‘meta Kappa’ project [5,10]

Finally, the framework presented here remains entirely formal and, although we obviously have in mind an interpretation in terms of signalling networks, it does not actually embody any domain-specific knowledge. In particular, although we have used suggestive names for nodes in our examples, these have no actual significance and could be arbitrarily renamed without affecting the content of the knowledge representation. We plan to address issue this by introducing semantic annotations for nodes that would allow us to express domain-specific properties such as “this region is an SH2 domain” or “this residue must be serine or threonine”.

This grounding process opens up the possibility of performing a second level of semantic checking on nuggets that have already been verified to be syntactically well-formed. Such checks could be purely routine, e.g. “binding actions have exactly two participants”, “only a kinase can phosphorylate a protein” or “a serine/threonine kinase cannot phosphorylate a tyrosine residue”. However, we principally envisage semantic annotations as a means to automate certain default aggregation decisions, e.g. “an SH2 domain can bind only one phospho-tyrosine ligand at a time”, with a particular focus on the numerous domain-domain and domain-ligand PPIs that occur in signalling networks which, by their very nature, embody highly generic binding mechanisms constrained by relatively simple—but tedious and error-prone to write by-hand—conflict conditions.

Acknowledgement. This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the U. S. Army Research Office under grant number W911NF-14-1-0367. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.
References


Finiteness and Computation in Toposes

Edward Hermann Haeusler∗
Departamento de Informática, PUC-Rio, Rio de Janeiro, Brazil
hermann@inf.puc-rio.br

Abstract

Some notions in mathematics can be considered relative. Relative is a term used to denote when the variation in the position of an observer implies variation in properties or measures on the observed object. We know, from Skolem theorem, that there are some first-order models where R is countable and some where it is not. This fact depends on the position of the observer, i.e., it depends on whether he/she is inside the model or not. In this article, we assume that computation is based on finiteness rather than natural numbers and discuss Turing machines computable morphisms defined on top of Dedekind and Kuratowski finiteness. The models are provided by toposes where the Axiom of Choice (AC) does not hold, since Tarksi proved that inside Set Theory with AC all these finiteness notions are equivalent. The toposes do not have natural numbers object (NNO) either, since in a topos with a NNO these finiteness notions are equivalent to Peano finiteness. The main contribution of this article is to show that although from inside every topos, with the properties previously stated, the computation model is standard, from outside some toposes, unexpected properties on the computation arise, e.g., infinitely long programs, finite computations containing infinitely long ones, infinitely branching computations.

1 Introduction

Investigations on effectiveness usually follows two non-exclusive approaches, the model-theoretical and the proof-theoretical one. The model-theoretical approach provides a “model” $X$, such that, any (partial) function $F$ from $A$ into $A$ is $X$-effective if and only if there is an instance $X_F$ of $X$ that represents $F$. The meaning of the “instance $X_F$ that represents $F$” is provided informally by stating that for all input $i_X$ submitted to $X_F$, produces an output $o_X$, if and only if, $F(i) = o$, for some fixed representation for the input and output data, e.g. numerals or strings simply. The meaning of “submitting” and of “producing” is also at least informally defined when introducing the “model” $X$ main concepts. $X$-effectiveness has to as close to our intuition on effectiveness as possible. The proof-theoretical provides a logical theory $T$, such that, for any partial function $F$ from $A$ to $A$, $F$ is said to be $T$-effective if and only if there is a term $t_F$, the codification of $F$, and a proof that $t_F \in Terms(T)$, such that for every $a \in A$, $F(a) = b$, iff, $t_F(a) = b \in T$. $T$ is presented by a set of axioms and inference rules for deriving propositions on membership of $T$ and identity between elements of $T$. The main judgment in $T$ are either of the form $t_F \in Terms(T)$ or $[F \equiv G] \in T$.

Typical of a model-theoretic approach are Turing machines, while Gödel’s partial recursive functions follows the proof-theoretical. Of course, the approaches are not purely model-theoretical or proof-theoretical. One can consider lambda-calculus as a purely proof-theoretical example, by ignoring the underlying evaluation model that provides the intentional identity relationship. On the other hand, we can consider the lambda calculus as a model-theoretical approach if we focus on the lambda-terms evaluation model. Roger’s theorem stating an abstract axiomatization for the proof-theoretical approach on effectiveness provides stronger evidence (read [11] or [8] ch.4) for Turing-Church thesis: “A (partial)

∗The author thanks CAPES and the Petrobras project led by prof. Eduardo Sani Laber at PUC-Rio, for the academic and financial support.

1In classical theory of recursive functions $A$ is always the set $\mathbb{N}$ of natural numbers.

2Stronger than evidences provided by some concrete models, as those raised since Turing’s work.
function from \( \mathbb{N} \) to \( \mathbb{N} \) is effective, if and only if, it is Turing computable”. Thus, Roger’s theorem can be seen as a meta-theoretical prof-theory based approach for effectiveness.

This work is based on the observation that finiteness is basic for effectiveness definition. The Turing machine, for example, relies the restriction on the tape content, set of symbols and states on finite sets. “Programs”, or whatever is used to represent the effective functions, are finite, they “run” in finite time, they use a finite amount of “data”, if they are non-deterministic the corresponding non-deterministic range is always finite, etc. Although natural numbers may appear as central in effectiveness studies, some computational models do not need any explicit mention to them, namely Turing machines. Finiteness seems to be more basic than natural numbers.

In \( \text{ZFC} \), Peano’s definition of finite sets uses the set \( \mathbb{N} \) of the natural numbers. Nevertheless, definitions of finite in \( \text{ZFC} \), such as those due to Dedekind or Kuratowski, do not depend on the existence of any infinite set, such as \( \mathbb{N} \). To the best of our knowledge, we can say that almost all model-theoretical approaches for effectiveness are carried out in \( \text{ZFC} \). In 1924, Tarski proved that the many existing and well-known definitions of finite set are equivalent. Tarksi mentioned Peano-finiteness, Dedekind-finiteness and some inductive definitions due to M.M. Russell, Sierpinski and Kuratowski. A first fact to be noted is that he had to use the Axiom of Choice in his proof. Another fact is that, due to the duality between finite and infinite, when defining one of these concepts the respective dual is obtained by means of negations. The use of negation adds a logical dimension to this discussion. Thus, besides the Axiom of Choice, the fact that we are inside Intuitionistic or Classical framework has interesting consequences on the relationship among these mathematical definitions of finiteness.

Outside the realm of \( \text{ZFC} \), finiteness is a relative notion. We use the term relative meaning that the variation in the position of an observer implies variation in properties or measures on the observed object. For example, Skolem’s theorem provides this relativity effect. We have some models where \( \mathbb{R} \) is countable, some where it is not. This fact depends on the position of the observer. In this specific case it may depend on whether he/she is inside the model or not. Outside \( \text{ZFC} \), for example, a Dedekind-finite object \( O \) can be intuitively infinite, that is, it can be expressed as \( O_1 \cup O_2 \) with both, \( O_1 \) and \( O_2 \) not Dedekind finite objects, i.e., Dedekind-infinite ones.

Based on the relativity of finiteness, we want to start a discussion on what is its real role in Theory of Computation. The methodology is to use a finiteness property as a parameter in Turing-machines definition. For example, by choosing the Dedekind definition of finite we have Turing-Dedekind machines and hence, Turing-Dedekind computable functions. By Tarksi’s analysis, this Turing-Dedekind machines cannot be formalized inside \( \text{ZFC} \), without proving the usual/classical Turing-machines. Category Theory (\( \text{CT} \)) is not completely dissociated from set theory as an alternative theory for the foundation of mathematics\(^4\). It provides, nevertheless, an alternative ontology\(^5\) for mathematics. In \( \text{CT} \), classes of objects and morphisms form a category. Morphisms are typed by domain and co-domain. For example, \( A \) and \( B \) are objects in a category \( C \) and \( f : A \rightarrow B \) is a morphism in \( C \) having domain \( A \) and co-domain \( B \). There is a typed composition “\( o \)” operation between morphisms that has a monoidal flavor. However, the meta-theory \( \text{CT} \), apart from the parcel of \( \text{ZFC} \) that it uses, does not provide meaning to propositions of the form \( A = B \) in \( C \). The meta-theory \( \text{CT} \) only provides meaning to assertions of the type \( f = g \), whenever \( f \) and \( g \) are morphisms. The whole class of all sets and all functions between these sets is the archetypal category. Inside this category (known by \( \text{SETS} \)) only identity of functions have semantics. About the objects of \( \text{SETS} \), i.e., the sets themselves, it cannot be stated that any two sets are equal or not equal. The most that can be said is that they are isomorphic or not\(^6\). This changing of perspective is quite interesting since it provides more ways to compare models of certain concepts

---

\(^3\)Zermelo Fraenkel set theory with the axiom of Choice.

\(^4\)The definition of category mention a set/class of objects and a set/class of functions.

\(^5\)Terminology, in philosophical sense.

\(^6\)A is isomorphic to \( B \), iff, there are \( f : A \rightarrow B \) and \( g : B \rightarrow A \), such that, \( f \circ g = \text{Id}_B \) and \( g \circ f = \text{Id}_A \).
We will find out that, at least hypothetically, some Turing-Dedekind machines when observed outside the model, i.e., from the ZFC perspective, have infinite set of states and/or infinitely long transition-tables. However, they are finite when observed from inside the model. We could show many other non-standard finite computational models. Because of Tarski’s result and the fact that Zermelo-Fraenkel set theory is usually classical, our discussion will be carried in the language CT. The use of Local set theory and toposes is justified by the fact that the internal logic of a topos is able to easily express set-theoretically based concepts as membership, sub-objects, functions, power-objects. This helps to translate the set-theoretically inspired notions of as Dedekind-finiteness and Kuratowski-finiteness to the Local language. Other categorical approaches to finiteness, as lfp-categories (presented [10] and [1] in an English comprehensive presentation) lacks an internal logic able to define in a first glance Turing-computable morphisms. The main contribution of this article is to provide a definition of Turing-computable morphism inside any topos, that is parametric on a finiteness notion and does not depend on NNO. From this definition we show some examples of toposes, without NNO and not satisfying the Axiom of Choice, where from outside the them, many unexpected computational properties may hold: 1- Infinitely long Turing machine codes, 2- Infinitely long computations contained in finitely long ones and 3- Infinitely branching computations. In Section 2 we briefly explain Local set theory and toposes, in Section 3 we show a Dedekind-finite automaton having an infinite sub-automaton, and how this can be extended to and example of a Dedekind-finite automaton with arbitrarily many different, and non-isomorphic, infinite sub-automata. We also show, in this topos of automata that there exists an object $\mathcal{A}$, an automaton, having only one element $1 \rightarrow \mathcal{A}$, but three non-empty proper sub-objects. In Section 4 we define, from the literature, using the language of Local Set Theory, the finiteness notions of Dedekind-finite, Kuratowski-finite and Peano-finite. In Section 5 we define Turing machines, their computations and the Turing-computable morphisms induced by them, by means of Local Set Theory formulas in an arbitrary topos. In Section 6 we remark the existence of some topos where from the outside the above mentioned non-standard computational properties of Turing-machine computations hold. In this section, we also remark our finiteness assumption, taken instead of the number-theoretical notion usual to Classical Recursion theory, on the light of some recent articles on that discuss the nature of Turing-Church thesis.

## 2 Local set theory and toposes

One of the useful aspects of topos theory [5, 7] from a logical point of view concerns investigating the internal logic of toposes, namely categories with some special properties, by means of localized language, called local set theory (LST) [2]. This has been accomplished by taking any topos as a model of a theory in the language of LST, which is basically a higher-order typed language. The interpretation of such a theory in the particular topos provide us with a convenient way of treating the objects of the topos as set-likes entities and the morphisms between them as function-like relations between them.

With the purpose of fixing terminology and provide some (useful) intuition, we write down some definitions. We remind the reader that the definitions in category theory work up to isomorphism.

**Definition 1 (Topos).** A topos is a category $\mathcal{T}$ having: (1) Terminal Object; (2) Pull-Backs; (3) Exponential Objects; (4) Sub-object Classifier.

**Definition 2 (Sub-object Classifier).** A sub-object classifier, in a category $\mathcal{T}$, is an object $\Omega$, together with a morphism $\top : 1 \rightarrow \Omega$, such that, for every monomorphism $f : B \rightarrow A$, there is a unique morphism...
\( \chi_f : A \to \Omega \), such that, the following diagram is a Pull-Back:

\[
\begin{array}{ccc}
A & \xrightarrow{f} & B \\
\downarrow & & \downarrow \chi_f \\
1 & \xrightarrow{\top} & \Omega
\end{array}
\]

The morphism \( \top \) plays the rule of the truth-value “true”. Monomorphisms provide a way of defining sub-objects inside a category. Inside a topos, many set-theoretical notions can be categorified, that is, translated to the CT language in a way that preserves its original meaning in SETS, the category of sets and functions between them. For example, the notion of element of set is in a one-to-one correspondence with functions from a singleton to this set. Each \( a \in A \) is associated with the function \( f : \{\star \} \to A \), such that, \( f(\star) = a \). The fact that this correspondence can be seen as a bijection between functions from a singleton to \( A \) and elements of \( A \) allow us to categorify the set-theoretical notion of elements in any category with a terminal object. Terminal object are the categorical counterpart of singletons. As a matter of notation, we denote by (up to isomorphism) \( \{a\} \in A \) the fact that \( \{a\} : 1 \to A \) is a morphism in a particular category. The categorification of the empty set is the initial object, since there is one and only one function (in SETS) from \( \emptyset \) to any other set. As we could expect, categorified notions does not preserve all properties they have in SETS, as the two following examples illustrates.

**Example 1.** Initial objects in any category not having a zero object\(^7\) cannot have elements\(^8\). Initial objects and empty objects in a category might be the same class. However, in the functor category \( \text{SETS}^{\rightarrow} \), there are objects that are empty and are not initial. An initial object in this category is isomorphic to the only function from \( \emptyset \) into \( \emptyset \). On the other hand, functions from \( \emptyset \) into \( A \), for any set \( A \), cannot have elements either. The formula \( \neg \exists x (x \in X) \) is true if \( X \) is assigned to \( \emptyset \to A \) in \( \text{SETS}^{\rightarrow} \).

**Example 2.** Let \( f, g : A \to B \), two functions in \( \text{SETS} \), such that \( f \neq g \). Consider now the objects \( ! : \emptyset \to A \) and \( \text{Id}_B : B \to B \) in \( \text{SETS}^{\rightarrow} \) and the commutative diagram below, showing that the \( (!, f) \) and \( (!, g) \) are morphisms from \( ! : \emptyset \to A \) in \( \text{Id}_B : B \to B \) in \( \text{SETS}^{\rightarrow} \).

\[
\begin{array}{ccc}
A & \xrightarrow{f} & B \\
\downarrow & \nearrow \chi_f & \downarrow \chi_f \\
\emptyset & \xrightarrow{\text{Id}_B} & B
\end{array}
\]

*Because there is no element in \( ! : \emptyset \to A \), inside \( \text{SETS}^{\rightarrow} \) it is not possible to falsify the formula \( \forall x \in X (F(x) = G(x)) \to F = G \), if \( F, G \) are assigned to \( (!, f) \) and \( (!, g) \), respectively, and \( X \) to \( ! : \emptyset \to A \). Internally \( F \) and \( G \) are equal, but they are not externally equal.*

Using sub-object classifiers it is possible to locally define equality, the membership relation, existential and the universal quantifier. For any topos they form the semantics of Local Set Theory. The reader can check that the Definition 3 corresponds to the identity in \( \text{SETS} \). The other definitions are omitted because of lack of space. Being a morphism from \( A \times A \) into \( \Omega, =_A \) is a predicate. Thus, \( \text{LST} \) has a propositional meaning for \( \in_{A}, =_A, \forall_A \) and \( \exists_A \). They are typed (localized) counterparts of \( \in, =, \forall \) and \( \exists \). This is briefly explained in the following subsection.

---

\(^7\)A zero object is an object that is initial and terminal.

\(^8\)Any morphism from \( 1 \) to \( 0 \) would make them isomorphic.
2.1 The internal language of a topos

Definition 3 (Local Identity). Consider an object $A$ in a topos $T$. Let $\delta : A \to A \times A$ be the diagonal morphism, defined as $(\text{Id}_A, \text{Id}_A)$ from the usual categorical cartesian product definition. The sub-object classifier pullback below defines local equality $=_A$, in $A$. $=_A$ is the characteristic morphism of $\delta$.

\[
\begin{array}{c}
A \\ \delta \\
\downarrow \\
1 \\
\end{array} 
\quad 
\begin{array}{c}
A \times A \\
= \\
\Omega \\
\end{array}
\]

Using the internal logic of the topos, provided by the sub-object classifier, it is possible to define the membership relationship $\in_A$, localized in any object $A$ of the topos. This definition strongly relies on the fact that $\text{Hom}(A, \Omega)$ represents the collection of sub-objects of $A^A$. In fact $\text{Hom}(A, \Omega)$ is a Heyting algebra.

Definition 4 (Local Membership). Consider a topos $T$ and an object $A$ in it. Let $\text{ev}_A : A \times \Omega^A \to \Omega$ be the evaluation morphism provided by the exponential object $\Omega^A$. The following instance of the sub-object classifier diagram defines $\in_A$.

\[
\begin{array}{c}
\in_A \\
\downarrow \\
1 \\
\end{array} 
\quad 
\begin{array}{c}
A \times \Omega^A \\
\text{ev}_A \\
\Omega \\
\end{array}
\]

So, the local membership to $A$ is the characteristic morphism of $\text{ev}_A$.

There is a very useful and strong way of building toposes. This is provided by one the fundamental theorem of topos. Its proof can be found in [5, 7, 2]. A category is locally small whenever $\text{Hom}(A, B)$ is a set, for any $A$ and $B$ in the category.

Theorem 1. Let $C$ be a locally small category. $\text{SETS}^C$ is a topos.

The category $\text{SETS}^C$, when $C$ is a pre-order, is naturally interpreted as sets varying according $C$. The pre-order works as a temporal structure over each set evolves. When $C$ is more than a pre-order category, sometimes it is possible to see a kind of topology on any object $A$ induced by the morphism with co-domain $A$. In this case, we have a temporal structure induced by this topology. Anyway, in some cases, $\text{SETS}^C$ is naturally equivalent to a category of dynamic systems. Since discrete dynamical systems can be seen as a semantics for computing process, the use of the above functorial category in providing examples for non-standard model of computing is justified.

2.2 The logical language related to toposes

Any topos is (naturally isomorphic to) a model of some local set theory ($\text{LST}$). In $\text{LST}$, the notion of type replaces “set”. In the language of $\text{LST}$ each term (including those representing sets) has an associated type. The terminology “local” in $\text{LST}$ provide us with a scope (locality) to any “set-theoretical” operations, such as union, intersection, identity (see Definition 3), membership (see Definition 4), etc. These “set-theoretical” operations are only defined for terms of the same type (i.e., locally). Apart from that, the language is very similar to set-theory language, based on the primitive symbols $=, \in$ and the

---

9By the sub-object classifier axiom 2 each morphism from $A$ to $\Omega$ corresponds to a sub-object of $A$ in a bijective way.
The language of \( LST \) is defined in the sequel. This presentation follows [2] (see pp. 91ff), where the details on how to interpret a local language in an arbitrary topos are provided.

**Definition 5 (Local language).** A local language \( \mathbf{L} \) is defined by:

- **Symbols** the unit symbol \( 1 \), the truth-value type symbol \( \Omega \), a collection of ground type symbols \( A, B, C, \ldots \), and a collection of function symbols \( f, g, h, \ldots \);
- **Types** the set of types of \( \mathbf{L} \) is the least set \( T \) containing \( 1, \Omega \), all ground type symbols \( A, B, C, \ldots \) and closed under the following operations:
  - For \( A \in T \), the type of power object \( PA \) is also in \( T \);\(^{11}\)
  - For \( A_1, \ldots, A_n \in T \), the product type \( A_1 \times \ldots \times A_n \) is also in \( T \) (for \( n = 0 \), the product type is \( 1 \));
  - For \( A, B \in T \), the exponential type \( A \to B \) is also in \( T \).
- **Signatures** Each function symbol \( f \) is associated to a signature \( A \to B \), where \( A \) and \( B \) are types. We use \( f : A \to B \) to denote this;
- **Variables** For each type \( A \) there is a countable set of variables \( V_A \);
- **Terms** For each type \( A \), there is a set \( T_A \) of terms of type \( A \), defined as follows:
  - \( \star \in T_1 \);
  - \( V_A \subseteq T_A \);
  - For \( f : A \to B \) and \( \tau \in T_A \), we have that \( f(\tau) \in T_B \);
  - For \( \tau_1, \ldots, \tau_n \in T_{A_1}, \ldots, T_{A_n} \), we have that \( (\tau_1, \ldots, \tau_n) \in T_{A_1 \times \ldots \times A_n} \). In the case \( n = 0 \), this term is \( \star \in T_1 \);
  - For \( \tau \in T_{A_1 \times \ldots \times A_n} \), we have that \( \pi_i(\tau) \in T_{A_i} \) (for \( i = 1, \ldots, n \));
  - For \( \varphi \in T_\Omega \) and \( x \in V_A \), we have that \( \{ x \mid \varphi \} \in T_{PA} \);
  - For terms \( \sigma \) and \( \tau \) of type \( A \), we have that \( \sigma = \tau \) is a term in \( T_\Omega \);
  - For terms \( \sigma \) and \( \tau \) of types \( A \) and \( PA \), respectively, we have that \( \sigma \in \tau \) is a term in \( T_\Omega \).

Terms of type \( \Omega \) are called formulae. We use superscripts to indicate the type of a variable, and, in order to not have overloaded superscripts we allow the omission of some of these superscripts in some terms, since this omission does not interfere in their unique typing. Free and bound occurrences of variables are defined in the usual way. Logical operators can be define as abbreviations, (see [2], p. 70). For example, \( \top \) is defined as \( \star = \star \); given formule \( \varphi, \psi \) we have that \( \varphi \land \psi \) is defined as \( (\varphi, \psi) = (\top, \top) \), and \( \varphi \Rightarrow \psi \) is defined as \( \varphi \land \psi = \varphi \). Quantiﬁers are also (locally) deﬁned: given a variable \( x \) of the appropriate type \( A \), \( \forall x^A : \varphi \) is an abbreviation of \( \{ x \mid \varphi \} = \{ x \mid \top \} \). The falsum \( (\top) \) is deﬁned as \( \forall \omega^\Omega : \omega \). Consider a formule \( \varphi \), with no occurrence of the variable \( \omega \) of type \( \Omega \), \( \exists x^A : \varphi \) is deﬁned as \( \forall \omega^\Omega : (\forall x^A : (\varphi \Rightarrow \omega) \Rightarrow \omega) \). Some terms in a local language represent set-like objects in the Topos.

**Definition 6 (Set-terms).** A set-term is any term of power type \( PA \) for some type \( A \).

Set-theoretical-like deﬁnitions are listed in [2] (see pp. 83ff). For example: \( X \subseteq Y \) is deﬁned as \( \forall x^X : (x \in X \Rightarrow x \in Y) \); \( X \cap Y \) is deﬁned as \( \{ x \mid x \in X \land x \in Y \} \); \( X \cup Y \) is deﬁned as \( \{ x \mid x \in X \lor x \in Y \} \), where \( X \) and \( Y \) are of type \( PA; A \) is deﬁned as \( \{ x \mid \top \} \), of type \( PA \), with \( x \) a variable of type \( A \). Thus, for every type symbol \( A \), there is a corresponding set-term \( A \). The term \( \emptyset_A \) is deﬁned as \( \{ x \mid \bot \} \), of type \( PA \), with \( x \) a variable of type \( A \); \( PA \) is deﬁned as \( \{ x \mid x \subseteq A \} \), of type \( PPA \), with \( x \) a variable of type

\(^{10}\)The operation \( [ ] \) when applied to a predicate \( \phi(x) \) of type \( \Omega^A \) provides a sub-object \( \{ x \mid \phi(x) \} \) of \( X \).

\(^{11}\)A power object \( PA \) internalizes the notion of the “collection” of all sub-objects of \( A \).
A more general version of defining set-like objects from set-like objects is provided by the term \( \{ \tau \mid \phi \} \), which is defined as \( \{ \tau \mid \phi \} \). A set of formulas is consistent, iff, it does not derive \( \bot \).

When the Topos determines the local language, each morphism \( f : A \rightarrow B \) in \( L \) corresponds to the set-term \( \{(x, f(x)) \mid x \in A\} \) of type \( B^A \). The type of \( B^A \) is \( PP(A \times B) \). Besides that, to each function symbol \( f : A \rightarrow B \) in \( L \) corresponds the set-term \( \{(i^A, x^B) \mid i \in I \land x \in X_i\} \). In [2] it is shown a deductive system, in sequent-style, to derive (draw conclusion) formulas from set of formulas. It is shown how any consistent\(^1\) set of formulas in \( LST \) gives raise to a syntactical topos (in the style of a Herbrand term model), and how from syntactical models one can derive a completeness theorem for \( LST \) logical consequence.

A notion of validity of a formula in a topos is defined in the expected way. Consider a mapping \( M \) from a Local Language \( L \) into a topos \( T \), such that, the types \( T \) are mapped into the objects of \( T \), \( \Omega \) is mapped into a sub-object classifier of \( T \) (\( \Omega_T \)), \( 1 \) is mapped into a terminal object of \( T \), products are mapped into products, function symbols are adequately mapped into morphisms, and variables of type \( X \) into morphisms from \( M(1) \) into \( M(X) \). This mapping is recursively extended to a mapping \( \hat{M} \) from terms into objects and morphisms of \( T \). Thus, for each set-like term \( t \), of type \( PA \), \( \hat{M}(t) \) is mapped to a corresponding sub-object of \( M(A) \). Terms of type \( \Omega \) are mapped to morphism from \( M(1) \) to \( M(\Omega) \), that are truth-values inside the topos. \( M \) can be seen as denotational interpretation of \( L \) into \( T \). Thus, having denotations for \( t_1 \) and \( t_2 \) of type \( A \), \( M(t_1) \) and \( M(t_2) \) sub-objects of \( M(A) \), the denotation of the formula \( t_1 = t_2 \) can be considered as true, if and only if, the monomorphism related to \( M(t_1) \) is equal to the one related to \( M(t_2) \), since Category Theory provides equality between morphism, this is unproblematic at first sight. However, the (local) equality can be defined inside a topos. As it was shown in Definition 3, the denotation of \( t_1 = t_2 \) is itself a general element of (a morphism from \( M(1) \) to \( M(\Omega) \)), and hence a morphism from \( 1 \) into \( \Omega_T \), a truth value itself. Since the equality defined in Definition 3 is a local notion, it is hardly the case that the truth value \( M(t_1 = t_2) \) is the same of \( M(t_1) = M(t_2) \). Example 2 confirm this and Example 1 provide a typical case of a categorification of a set-theoretical concept that does not have the same truth value on every topos. The concept of monomorphism, used here and not defined yet, is of a different kind. Monomorphism are categorification of injective functions: \( f : A \rightarrow B \) is a monomorphism (mono), iff, for every pair of morphisms \( h, g : C \rightarrow A \), such that \( f \circ h = f \circ g \), then \( h = g \). In a topos monomorphisms and injections coincide. In a general category this is not the case. In a topos every morphism that is injective and surjective is an isomorphism. Of course, this is the case in \( SETS \), since \( SETS \) is a topos. It is clear that there are properties expressed in \( LST \) that hold in some topos and does not in \( SETS \).

3 A Motivating Example

M-\( SETS \) form an universe that can be identified with the class of automata and morphisms between them. Let \( M \) be a monoid and \( A \) be a set. \( A : A \times A \rightarrow A \) is a M-Set, if, \( A(a, m \ast n) = A(a, n \ast m) \). Let \( A : A \times M \rightarrow A \) and \( B : B \times M \rightarrow B \) be M-\( SETS \). A function \( F : A \rightarrow B \), such that, \( F(A(x, m)) = B(F(x), m) \) is a morphism in \( S e t s^M \). We denote it by \( F : A \rightarrow B \). An M-\( SET \) (monoid actions) can be seen a a family \( (\mathcal{A}_{\tau})_{\tau \in \mathcal{M}} \) of functions from \( A \) into \( A \). Note that we use \( A \) for both, the family of functions and their domain and co-domain sets. T functions must satisfy: \( A_m(x) = A_m(x_0) \). \( F : A \rightarrow B \) is morphism, iff, \( F(A_m(x)) = B_m(F(x)) \). M-\( SETS \) and M-\( SETS \) morphisms form a category that is a topos. Let \( \mathcal{A} \) be the finite automaton at the right side of Figure 1(a). It can be uniquely defined by the actions \( m_a, m_b \) and

\(^{12}\)\( \exists y^A : \varphi \) is an abbreviation to \( (\exists y^A : \varphi) \wedge (\forall x^A \exists y^A : (\varphi \land \varphi(y/x) \Rightarrow x = y)) \), where \( \varphi(y/x) \) denotes the replacing of \( x \) by \( y \) in \( \varphi \), usual conditions on replacing free variables applies.

\(^{13}\)A set of formulas is consistent, iff, it does not derive \( \bot \).
\( m_c \) on the set \( \{q_1, q_2, q_3, q_4\} \), such that: 
\( A_a(q_1) = q_2, A_a(q_2) = q_4, A_a(q_3) = q_4, A_a(q_4) = q_4, A_b(q_1) = q_2, A_b(q_2) = q_3, A_b(q_3) = \ldots \) 

an isomorphism by
\[
\text{Iso}(f) \iff \exists h ((f \circ h = \text{Id}_B) \land (h \circ f = \text{Id}_A).
\]

We express that \( f \) is a monomorphism by
\[
\text{Mono}(f) \iff \exists h ((f \circ h = \text{Id}_B) \land (h \circ f = \text{Id}_A).
\]

Figure 1: Monomorphisms: Left (a) from one-state automaton \( B \) into \( A \). Right, (b) two-state \( B \) into \( A \)

What are the automata \( B \) that can be related to \( A \) by monomorphisms. Consider \( B \) with only one state \( X \). A mono \( F : B \to A \) is a function \( f : \{X\} \to \{q_1, q_2, q_3, q_4\} \) that is equivariant. \( f(X) = q_4 \) is the only possibility, for 
\( f(B_r(X)) = A_r(f(X)) \), see Figure 1(a). In Figure 1(b), \( B \) has two states \( X \) and \( Y \). A mono \( F : B \to A \) is a function \( f : \{X, Y\} \to \{q_1, q_2, q_3, q_4\} \) equivariant and injective. With \( f(X) = q_4 \), there is no way to have 
\( f(B_r(Y)) = A_r(f(Y)) \), for any choice, such that \( f(Y) = q_3, q_1, q_2 \).

There are only two monomorphisms \( F_1, F_2 : B \to A \) that are equivariant from an automaton with 3 states. They correspond to the functions \( f_1 \) and \( f_2 \): 
\( f_1(X) = q_4 \) and \( f_2(X) = q_4 \) and \( f_1(Y) = q_3 \) with \( f_2(Y) = q_2 \), and, \( f_1(Z) = q_2 \) with \( f_2(Z) = q_3 \). See figure above.

The \( \{a, b, c\}^* - \text{Set} \) \( A \) has only one element \( 1 \to A \), but three non-empty proper sub-objects.

In ZFC this statement cannot be true. Any set with only one element cannot have a power-set with three elements. This is yet another manifestation of the relativity of the concept of finiteness. Let us show an example on Dedekind notion of finiteness. In LST, we express that \( f \) is an isomorphism by
\[
\text{Iso}(f) \iff \exists h ((f \circ h = \text{Id}_B) \land (h \circ f = \text{Id}_A).
\]

We express that \( f \) is a monomorphism by
\[
\text{Mono}(f) \iff
\]
∀hAC ∀gAC ((f ◦ h = BC f ◦ g) → h =AC g). In a Topos T, A is D-finite in T, iff, ∀f ∈ A^A(Mono(f) ⇒ Iso(f)) holds in T. Here only as matter of completeness.

Example 3. Let M be the free monoid generated by \{m_i/i \in \mathbb{N}\}, A=\{a_n/n \in \mathbb{N}\}, B=\{b_n/n \in \mathbb{N}\}, and C = A \cup B. The automaton C : M \times C → C is the following.

We show that any injective morphism F : C → C is the identity IdC (see Example 1). Thus, C is D-finite, no matter the set-theoretical cardinality of A and B. If B is an infinite set then there is G : B ↦→ B, an injective function that is not bijective. Thus, B(m_n, b_k) = G(b_k) is not D-finite\(^{14}\). That is, there is a D-finite (C) object that has a an D-infinite sub-object (B). This can be extended for any n ∈ \mathbb{N}, such that, C = \mathcal{A} \cup \mathcal{A}_1 \cup \mathcal{A}_2 \cup \ldots \cup \mathcal{A}_n, C is D-finite and B_i, i = 1, \ldots, n, are D-infinite, Example 4.

4 Finiteness in Local Set Theories

The main discussion in this article, is the role played by finiteness in computability. We are sure that this notion is essential when defining any computation model, even it is used implicitly. From the literature on topos theory, we are aware of the fact that the well-known notions of finite, namely, Dedekind-finite, Kuratowski-finite, and Peano-finite, for example are not equivalent. This in fact means that some of these notions correspond to non-finite (infinite) extensions, externally, in some toposes. Besides that, such notions have counter-intuitive properties as we will mention in the sequel.

As a starting point, we write down, in LST, sometimes with the help of diagrams these finiteness notions.

The first item of the above list, Dedekind finiteness or D-finiteness as defined in Section 3, is quite interesting. There we shown D-finite objects that are externally infinite. Besides that, a result due to Johnstone [9] shows that any sub-object classifier is D-finite. However, there are many toposes that have infinitely many truth-values. Sheaves over a topological space S have \text{Opens}(S) as the "set" of truth-values, for example. One can argue that D-finiteness is obtained by negating D-infinity, and hence, it is strongly dependent on whether the topos is classical or not. Kuratowski finiteness, in contrast, is based on a positive aspect of finiteness. Intuitively, an object is Kuratowski finite, iff, we can provide an inductive proof that it is finite. This induction is based on the facts that the empty object and the singletons are finite and any binary union of finite objects is finite too.

Definition 7 (Kuratowski-finite). In a Topos T, A is Kuratowski-finite (K-finite), iff, the following holds:

∀z ∈ \Omega^2 ([0 → A] ∈ z ∧ ∀a ∈ A[|a| ∈ z] ∧ ∀y ∈ \Omega^2 ∃ y' ∈ \Omega^4((y ∈ z ∧ y' ∈ z) ⇒ (y \cup y') ∈ z) ⇒ [id_A] ∈ z)

Considering K(A) as the sub-object of \Omega^4 formed by the K-finite subobjects of A, we have (cf. [7]):

a) 0 \in 1 are K-finite; b) If f : A → B is an epimorphism and A is K-finite then B is K-finite too; c) B, C ∈ K(A), if and only if, B \cup C ∈ K(A); d) If B and C are K-finite then B + C and B \times C are K-finite; e) If A is K-finite and B ∈ \Omega^4 and B is complemented then B is K-finite too; f) X is K-finite, if and only if, K(X) is K-finite.

In both finiteness notions above it is possible to have a non-finite sub-object of a finite object. It happens in any topos that does not have always complemented sub-objects. This is the case in all non-classical toposes. Example of these toposes are used in the computational definitions in the sequel and in examples is Section 3. The third notion of finiteness is reported here only as matter of completeness.

\(^{14}\)Remember that a M-Set is an action M \times B ↦→ B.
It involves the existence of a NNO in the topos. We are not considering that NNO is an essential starting point for defining a computational model. In a topos having NNO, any Peano-finite object is a K-finite and a D-finite object. A natural number object is a rather categorical definition of natural numbers in a category. It states that there is an object \( N \), with two morphisms \( 0 : 1 \to N \) and \( s : N \to N \), such that, for any other object \( A \) and morphisms \( g : 1 \to A \) and \( h : A \to A \), there is one and only one morphism \( f : N \to A \), such that the primitive recursive equations \( f(0) = g(0) \) and \( f(s(n)) = h(f(n)) \) hold categorically. NNO allows us to define any primitive recursive function inside its category. Thus, in a topos having a NNO, the relation less-than is defined, by primitive recursion, stating that \( \langle \) as a sub-object of \( N \times N \). We define \([p] \approx \{0,\ldots,p-1\}\), as a sub-object of \( N \) using \( \Omega \) axiom:

\[
\begin{array}{c|c}
[p] & 1 \\
\hline
\langle & \to N \times N \xrightarrow{\pi_1} N \xrightarrow{\text{suc}} N
\end{array}
\]

**Definition 8** (Cardinal-finite). In a Topos \( T \) with NNO, \( A \) is c-finite, or Peano-finite, if and only if, \( A \) is isomorphic to \([p]\) for some \( p : 1 \to N \).

## 5 Turing Machines in Local Language

In this section we detail our finiteness-parametric Turing-computable functions inside toposes that may not have natural numbers objects. The potential cases of internal standard Turing machines representing the external non-standard are: (1) Infinitely long programs; (2) Infinitely many branching, and; (3) Infinitely long traces inside finitely long ones; as remarked in Section 6.

In this subsection we express Turing Machines by means of a local language. We consider that finiteness is essential for the definition of any computational model. Instead of fixing a specific finiteness notion, we use it as a parameter. So, consider \( \text{fin}(X) \) a predicate that defines a finiteness notion. In the usual (set-theoretical) definition of a Turing Machine we have \( \langle Q, \Sigma, \{q_0\}, \{q_f\}, \delta \rangle \), and, \( \delta \subseteq 2^{\Sigma \times Q \times \{\rightarrow,\leftarrow\} \times \Sigma \times Q} \). \( Q \) and \( \Sigma \) has to be finite and non-empty sets of states and symbols, and \( \delta \), the transition function, is finite as a consequence of the finiteness of \( Q \) and \( \Sigma \), and the fact that the power set of a finite set is finite. Besides that, the elapsed time of any meaningful computation has to be finite. The behavior of the Turing Machine is described by the definition of \( \delta \). We proceed in defining the semantics of a Turing Machine, in a topos without NNO. It is important to remember that in a Topos with NNO, the finiteness notions collapse to Peano-finiteness.

**Definition 9** (Turing Machine in LST). Let \( \Sigma \) and \( Q \) be types and \( X, Y, q_0^Q \) and \( q_f^Q \) variables, then \( T \), a variable of type \( Q \times \Sigma \times Q \times Q \times Q) \to P(Q \times \Sigma \times 2) \) is a Turing Machine, iff \( \pi_1(T) = X \land \pi_2(T) = Y \land (\pi_3(T) \in X) \land (\pi_4(T) \in X) \land \text{fin}(X) \land \text{fin}(Y) \). We denote this predicate as \( TM(T) \).

The type \( 2 \), in the above definition, is used to denote \( \text{Right} \) and \( \text{Left} \), the directions of a possible moving in a one-step transition of the Turing Machine.

Note that we do not have the type of the Turing Machines in this formalization. We have chosen this way, in order to not have to fix a type for states and alphabets. Whenever we refer to a Turing Machine \( T \), it is a variable \( x \) of the type \( Q \times \Sigma \times Q \times Q \times Q) \to P(Q \times \Sigma \times 2) \), for some types \( Q \) and \( \Sigma \), such that \( TM(x) \). In the sequel, we show, in an informal way, the definition of the behavior of a Turing Machine \( T = \langle X, Y, q_0, \delta \rangle \), with adequate typing corresponding to \( Q \times \Sigma \times Q \times Q \times Q) \to P(Q \times \Sigma \times 2) \).

Consider a type \( \text{Pos} \), a variable \( Z^\text{Pos} \) and a variable \( X^\Sigma \). \( H^\Sigma^\Sigma^\Sigma \) is a tape, if it satisfies \( \text{Tape}(H^\Sigma^\Sigma^\Sigma) \iff \text{fin}(|p^{\text{Pos}} \mid H(p) \neq \text{""}) \land \text{fin}(|\text{Pos})^{\Sigma\Sigma\Sigma} \)\(^{\Sigma\Sigma\Sigma} \). In a Turing machine tape, the non-blank cells must be only blank in the cell.
finitely many. Besides that, there is no limit in the tape for storing symbols.

**Definition 10** (Closuse). The behavior of $T$ is $\hat{\delta}$ (a variable of type defined in the sequel). It is specified by means of the following variables (implicitly locally quantified) and predicates. At the end we have the morphism computed by $T$.

- $H : \text{Pos} \rightarrow \Sigma$, such that $\text{Tape}(H)$
- $f_\delta : \mathcal{P}(Q \times \text{Pos} \times T_{\text{fin}}) \rightarrow \mathcal{P}(Q \times \text{Pos}_{\text{fin}} \times Q)$, such that $f_\delta(S_1) = S_2 \iff S_2 = \{s \mid \exists x_1 \in S_1(\pi_3(T)(\pi_1(x_1),\pi_2(x_1),\pi_3(x_1))) = s \land \text{Tape}(\pi_3(x_1))\}$, where $\pi_3(T)(X,Z,H)$ is an usual translation for the local typed language of the one-step of a Turing Machine, possibly non-deterministic, $T$, in state $X$, on position $Z$, and tape $H$.
- $\text{Config}(X) = \{S \subseteq X : f_\delta(S) \subseteq S\}$
- $i : \text{St}(X) \hookrightarrow \mathcal{P}(X)$ has left adjoint $O : \mathcal{P}(X) \rightarrow \text{St}(X)$, $O(Z) = \cap\{S \in \text{St}(X) : Z \subseteq S\}$
- $O \circ i$ is a closure operator, thus we have $\hat{\delta} = O \circ i$
- $f = \langle a^{\Sigma}, y^{\Sigma} \rangle / (q, z^{\text{pos}}, y^f) \in \hat{\delta} \circ \langle q_0^{\hat{\delta}}, o^{\text{pos}}, x^f \rangle$ and final$(q)$. Here, to obtain $\Sigma^*$, the type of strings, a similar closure definition and application has to be done.

This definition uses a technique from [3] to denote Orbits in arbitrary toposes, without NNO. It is important to note that if the topos has a NNO, then the closure defined above is just Kleene closure.

Having defined the concept of a morphism being Turing Computable in a Topos, we have that: In $\text{Sets}_{\text{fin}}^G$, $G$ a particular free group, with $\text{fin}(A) = \text{Dedekind}(A)$. $\text{Sets}_{\text{fin}}^G \models \exists W(\text{fin}(W) \land \neg\text{fin}(\mathcal{P}(W)))$

Hence in $\text{Sets}_{\text{fin}}^G$, a T.M. with states in $W$ is a non-standard computational model possibly with a D-infinitely long program, or a D-infinitely branching non-deterministic behavior.

In $\text{Sets}_{\text{fin}}^{0\rightarrow 1}$ with $\text{fin}(A) = \text{Kuratowski}(A)$. $\text{Sets}_{\text{fin}}^{0\rightarrow 1} \models \exists W(\text{fin}(W) \land \exists V(V \leq W \land \neg\text{fin}(V)))$

Thus, a T.M. with states in $W$ can be a non-standard computational model able to K-finitely compute on K-infinitely long transitions.

We provided here arguments in favor of a kind of non-standard finiteness phenomena inside $M$-Sets. They justify in details what is observed in the conclusion of our work (Section 6).

**Fact 1.** Let $M$ be the free monoid generated by $\{m_i / i \in \mathbb{N}\}$ $A = \{a_n / n \in \mathbb{N}\}$, $B = \{b_n / n \in \mathbb{N}\}$, and $C = A \cup B$ Let $C : M \times C \rightarrow C$ be the action of $M$ on $C$, such that,

$$C(m_n, x) = \begin{cases} a_k & \text{if } x = a_k \text{ and } k \neq n \\ b_k & \text{if } x = a_k \text{ and } k = n \\ b_k & \text{if } x = b_k \end{cases}$$

Any injective morphism $\mathcal{F} : C \rightarrow C$ is the identity $\text{Id}_C$. This is justified by observing that, if $\mathcal{F}$ was not the identity, then there would exist $n$, such that, $f(a_n) \neq a_n$. Thus, $f(a_n) = x \neq a_n \Rightarrow C(m_n, x) = x$, for $a_n$ is the only element of $A \cup B$ changed by the action $C$. Then, $f(b_n) = f(C(m_n, a_n)) = C(m_n, f(a_n)) = C(m_n, x) = x = f(a_n)$. This cannot be possible.

Let $G : B \rightarrow B$ be an injective function that is not bijective. The action $\mathcal{B}(m_n, b_k) = G(b_k)$ is not Dedekind finite. Thus, $\mathcal{B}$ is a Dedekind infinite object inside a Dedekind finite object in $M$-Sets.

**Example 4.** It is also possible to have arbitrarily many disjoint infinite sub-objects of $C$. Let $C : M \times (A \cup B \cup D) \rightarrow (A \cup B \cup D)$ be the action of $M$ on $C$, such that,$$
C(m_n, x) = \begin{cases} a_k & \text{if } x = a_k \text{ and } k \neq n \\ b_k & \text{if } x = a_k \text{ and } k = n \\ b_k & \text{if } x = b_k \text{ and } k \neq n \\ c_k & \text{if } x = b_k \text{ and } k = n \\ c_k & \text{if } x = c_k \end{cases}$$

\[\text{Note that we confuse the set with the action.}\]
Any injective morphism $F : C \to C$ is the identity $\text{Id}_C$. The same argument used in fact 1 is used to prove that if $f(a_n) = x \neq a_n \Rightarrow C(m_n, x) = x$, then $f(b_n) = f(C(m_n, a_n)) = C(m_n, f(a_n)) = C(m_n, x) = x = f(a_n)$. That is not possible. On the other hand if each of $B$ and $D$ and $A$ are infinite sets then there are $G : B \to B$ and $H : D \to D$ injective functions that are not bijective. The actions $B(m_n, b_k) = G(b_k)$ and $D(m_n, c_k) = H(c_k)$ prove that both, $B$ and $D$ are not Dedekind finite.

In [4], a categorial presentation of recursiveness is provided using $\text{CT}$. It axiomatizes categories able to define primitive recursive morphisms in a completely abstract way. Using the internal language of the category it is possible to precisely define any primitive recursive function. This work is very interesting, since, it joins in a quite harmonious way a model-theoretic definition with a proof-theoretic one. The identity present in the meta-theory provides meaning for a theory of equality between intentionally distinct ways of defining the primitive recursive functions. Besides that no mention on a concrete numerical system of even richer definition of natural number is needed, but primitive recursiveness.

6 Conclusion

In some mathematical universes, e.g. M-Sets, finite objects may not share essential properties with their $\text{Sets}$ counterparts. Finiteness is relative. The first example in Section 3 was carried out using a rather intuitive notion of finiteness. Given a property $F(S)$ that provides a finiteness definition for $S$, in $\text{LST}$, and a Topos $T$, we can express Turing machines (TM) and their semantics. In Definition 10 some effort is need to define TM-semantics without natural numbers. $f : A \to B$ in $T$ is TM-computable, iff, there is a TM that has $f$ as semantics. What is interesting in this general definition of TM computability is that, without appealing to infinitely long computations and programs, as well non-deterministic branching, for the TMs are internally finite, programs and branching can be externally infinite. We intend to see how this is related to computation on infinite-time Turing machines ([6]), but at the present stage of the research and the lack of space we can not provide any definite result. Finally, we would like to comment on some recent analysis on the status of Turing-Church thesis, by Sieg, Kripke, Dershowitz and Gurevich, for example. Due to lack of space, such remarks cannot be made. Nevertheless, we can say at least that most of these writings agree on the point that Church, and Turing, admitted that finiteness is a rather basic principle in computation, being this human based or machine based.

References

Mean-Field Semantics for a Process Calculus for Spatially-Explicit Ecological Models

Mauricio Toro¹, Anna Philippou², Sair Arboleda³, María Puerta¹, and Carlos M. Vélez S.¹

¹ Universidad Eafit, Medellin, Colombia
mtorobe@eafit.edu.co, mpuerta@eafit.edu.co, cmvelez@eafit.edu.co

² University of Cyprus, Nicosia, Cyprus
annap@cs.ucy.ac.cy

³ Universidad de Antioquia, Medellin, Colombia
sairorieta@yahoo.es

Abstract

We define a mean-field semantics for S-PALPS, a process calculus for spatially-explicit, individual-based modeling of ecological systems. The new semantics of S-PALPS allows an interpretation of the average behavior of a system as a set of recurrence equations. Recurrence equations are a useful approximation when dealing with a large number of individuals, as it is the case in epidemiological studies. As a case study, we compute a set of recurrence equations capturing the dynamics of an individual-based model of the transmission of dengue in Bello (Antioquia), Colombia.

1 Introduction

The collective evolution of a group of individuals is of importance in many fields; for instance, in system biology, ecology and epidemiology. When modeling such systems, we want to know the emergent behavior of the whole population given a description of the low-level interactions of the individuals in the system. As an example, in eco-epidemiology the focus is on the number of individuals infected in a certain population and how a small number of individuals infected may lead to an epidemic.

Eco-epidemiology can be seen as a particular case of population ecology. The main aim of population ecology is to gain a better understanding of population dynamics and make predictions about how populations will evolve and how they will respond to specific management schemes. In epidemiology, such management schemes can be a cure to a disease, mechanisms to prevent a disease such as vaccines, or mechanisms to prevent the vector (species infected with a disease) to spread a disease. To achieve these goals, scientists construct models of ecosystems and management schemes (e.g., [30]).

Various formalisms have been proposed in the literature for the individual-based modeling of biological and ecological systems. Examples of such formalisms include the calculus of looping sequences [7] and its spatial extension [6], cellular automata [14, 11], Petri nets [16], synchronous automata [13], P systems [9, 8, 23] and process calculi (or process algebras) [25, 17].

In our work, we are interested in the application of process calculi for studying the population dynamics of ecological systems. Process calculi are formal frameworks to model and reason about concurrent systems and provide constructs to express sequential and parallel composition of processes as well as different means of communication between processes. In contrast to the traditional approach to modeling ecological systems using ordinary differential equations which describe a system in terms of changes in the population as a whole, process calculi are suited towards the so-called “individual-based” modeling of populations. Process calculi enable one to describe the evolution of each individual of the population as a process and, subsequently, to compose a set of individuals (as well as their environment) into a complete ecological system. Process calculi include features such as time [24, 20], probability [26] and stochastic behavior [15]. Furthermore, following a model construction, one can use model-checking
tools for automatically analyzing properties of models (e.g., [18, 28]) as opposed to just simulating trajectories, as it is typically carried out in most ecological studies.

In a previous work, we presented PALPS (Process Algebra with Locations for Population Systems), a process calculus developed for modeling and reasoning about spatially-explicit individual-based systems [22]. In PALPS, individuals are modeled using discrete time and probabilistic behavior, and space is modeled as a graph of discrete locations. We associated PALPS with a translation to the probabilistic model checker PRISM [3] to perform more advanced analysis of ecological models. Our experiments with PALPS [22, 21] delivered promising results via the use of statistical model checking provided by PRISM. However, it also revealed a limitation: the interleaving nature of the parallel composition operator led to a high level of nondeterminism and thus a very quick explosion of the state space. Moreover, this interleaving nature comes in contrast to the usual approach of modeling adopted by ecologists where it is considered that execution evolves in phases during which individuals of a population engage simultaneously in some process, such as birth, dispersal and reproduction.

To alleviate the problem of the interleaving nature of parallel composition and the high degree of nondeterminism in PALPS, we proposed a new semantics of PALPS, which captures more faithfully the synchronous evolution of populations and removes as much unnecessary nondeterminism as possible. Our proposal consisted of a synchronous extension of PALPS, named synchronous PALPS (S-PALPS) [28]. The semantics of S-PALPS implements the concept of maximum parallelism: at any given time all individuals that may execute an action will do so simultaneously [28]. Furthermore, we proposed a new translation of S-PALPS to PRISM which implemented this synchronous semantics as well as other features that removed the restrictions existing in the original framework. This led to a significant improvement regarding the size of S-PALPS models that can be analyzed via translation to PRISM in the range of hundreds of individuals. However, the approach cannot support the modeling of epidemiological systems where components can number in millions.

To deal with this challenge, in this paper we present a mean-field semantics to represent the average behavior of S-PALPS systems for populations of potentially millions of individuals. Mean-field semantics gives a deterministic approximation of the average behavior of a system, given low-level specifications at the individual level in terms of discrete time and continuous space mean-field equations. In this work we propose an algorithm of polynomial-time complexity for producing the mean-field equations given an S-PALPS model of a system. The algorithm avoids computing the complete state-space of a system and its complexity is independent of the size of the populations. Rather, given the stochastic nature of the systems in question, the accuracy of the method relies on the fact that the numbers of each agent in the system are sufficiently large. We illustrate the application of our semantics for the construction of mean-field equations of an S-PALPS model of the transmission of dengue in Bello, Colombia.

Mean-field semantics have been proposed for a number of process calculi including PEPA [15, 29] and WSCCS [26, 18]. The former line of work differs from our semantics since the underlying model is continuous time. Instead, our work is closely related to that of [18] for WSCCS. Our semantics extends that of [18] in two ways. First, we extend the semantics to deal with locations, since S-PALPS includes an explicit notion of discrete space not present in WSCCS. Second, the nature of our calculus and, specifically, the presence of an explicit probabilistic operator as opposed to weights and the absence of nondeterminism at the level of individuals, yields a simpler semantics as well as the lifting of some of the restrictions imposed in [18]. As related work, we also mention the mean-field semantics proposed for reactive networks in [12] which, however, is not directly related with our aim of providing this analysis capability to the spatially-explicit process calculus S-PALPS.

The remainder of the paper is as follows. In Section 2 we present the syntax and the semantics of S-PALPS. In Section 3 we present a mean-field semantics for S-PALPS. We then apply our techniques to study the population dynamics of dengue in Section 4. Section 5 concludes the paper and presents future work. The full exposition of the mean-field semantics of S-PALPS and the case study is in [27].
2 Synchronous PALPS

In S-PALPS, we consider a system as a set of individuals operating in space, each belonging to a certain species and inhabiting a location. Individuals who reside at the same location may communicate with each other upon channels (e.g., for preying) or they may migrate to a new location. S-PALPS models probabilistic events with the aid of a probabilistic operator.

The syntax of S-PALPS is based on the following basic entities: (1) \( S \) is a set of species ranged over by \( s, s' \). (2) \( \text{Loc} \) is a set of locations ranged over by \( \ell, \ell' \). The habitat is then implemented via a relation \( \text{Nb} \), where \( (\ell, \ell') \in \text{Nb} \) exactly when \( \ell \) and \( \ell' \) are neighbors. (3) \( \text{Ch} \) is a set of channels ranged over by lower-case strings. The syntax of S-PALPS is given at two levels, the individual level ranged over by \( P \) and the system level ranged over by \( S \) which are defined as follows:

\[
P : = 0 \mid \eta.P \mid \sum_{i \in I} p_i : P_i \mid \gamma?(P_1, P_2) \mid P_1 | P_2 \mid C
\]

\[
S : = 0 \mid P:(s, \ell, q) \mid S_1 \parallel S_2 \mid S \setminus L
\]

where \( L \subseteq \text{Ch} \), \( I \) is an index set, \( p_i \in (0, 1] \) with \( \sum_{i \in I} p_i = 1 \), \( C \) ranges over a set of process constants \( \mathcal{C} \), each with an associated definition of the form \( C \overset{\text{def}}{=} P \), and the actions that a process can perform are

\[
\eta : = a \mid \overline{a} \mid \text{go}\ell \mid \sqrt{} \quad \gamma : = a \mid \overline{a}
\]

Beginning with the individual level, \( P \) can be one of the following:

- Process \( 0 \) represents the inactive individual, that is, an individual who has ceased to exist.
- Process \( \eta.P \) describes the action-prefixed process which executes action \( \eta \) before proceeding as \( P \). An activity \( \eta \) can be an input action on a channel \( a \), written simply as \( a \); an output action on a channel \( a \), written as \( \overline{a} \); a movement action to location \( \ell \), \( \text{go}\ell \); or the tick action \( \sqrt{} \) that measures a discrete-time tick on a global clock action, \( \sqrt{} \). Actions of the form \( a, \overline{a} \), \( a \in \text{Ch} \), are used to model activities performed by an individual; for instance, preying and reproduction.
- Process \( \sum_{i \in I} p_i : P_i \) represents the probabilistic choice between processes \( P_i, i \in I \). The process randomly selects an index \( i \in I \) with probability \( p_i \), and then evolves to process \( P_i \). We write \( p_1 : P_1 \parallel p_2 : P_2 \) for the binary form of this operator.
- Process \( \gamma?(P_1, P_2) \) depends on the availability of a communication on a certain channel as described by \( \gamma \). If a communication is available according to \( \gamma \) then the communication is executed and the flow of control proceeds according to \( P_1 \). If not, the process proceeds as \( P_2 \). This operator is a deterministic operator as, in any scenario, the process \( \gamma?(P_1, P_2) \) proceeds as either \( P_1 \) or \( P_2 \) but not both, depending on the availability of the complementary action of \( \gamma \) in the environment in which the process is running.

Moving on to the population level, population systems are built by composing in parallel sets of located individuals. A set of \( q \) individuals of species \( s \) located at location \( \ell \) is defined as \( P:(s, \ell, q) \). In a composition \( S_1 \parallel S_2 \) the components may proceed while synchronizing on their actions following a set of restrictions. These restrictions enforce that probabilistic transitions take precedence over the execution of other actions and that time proceeds synchronously in all components of a system. That is, for \( S_1 \parallel S_2 \) to execute a \( \sqrt{} \) action, both \( S_1 \) and \( S_2 \) must be willing to execute \( \sqrt{} \). Action \( \sqrt{} \) measures a tick on a global clock. These time steps are abstract in the sense that they do not necessarily have a defined length and, in practice, \( \sqrt{} \) is used to separate the rounds of an individual’s behavior. In the case of multi-species systems these actions must be carefully placed in order to synchronize species with possibly different time scales.

System \( S \setminus L \) models the restriction of channels in \( L \) within \( S \). This construct is important to define valid systems: We define a valid system to be any process of the form \( S \setminus L \), where, for all of \( S \)’s
subprocesses of the form \(a?(P, Q)\) and \(\pi?(P, Q)\) we have that \(a \in L\). Hereafter, we consider only processes that are valid systems.

**Example 1.** Let us consider a species \(s\) where individuals cycle through a dispersal phase followed by a reproduction phase. Further, suppose that the habitat is a ring of size \(m\) where the neighbors of location \(\ell\) are \(\ell \pm 1\). In S-PALPS, we may model \(s\) by \(P_0\), where

\[
P_0 \overset{\text{def}}{=} \sum_{\ell \in \text{Nh}(\text{myloc})} \frac{1}{2} : g_0 \ell, \sqrt{.} P_1 \\
P_1 \overset{\text{def}}{=} p: \sqrt{.} (P_0 | P_0) \oplus (1 - p): \sqrt{.} (P_0 | P_0 | P_0)
\]

According to the definition, during the dispersal phase, an individual moves to a neighboring location which is chosen probabilistically among the neighboring locations of the current location (myloc) of the individual. Subsequently, the flow of control proceeds according to \(P_1\) which models the probabilistic production of one offspring (case of \(P_0 | P_0\)) or two offspring (case of \(P_0 | P_0 | P_0\)). A system that contains two individuals at a location \(\ell\) and one at location \(\ell'\) can be modeled as

\[
\text{System} \overset{\text{def}}{=} P_0; (s, \ell, 2) | P_0; (s, \ell', 1).
\]

The semantics of S-PALPS is defined operationally via two transition relations, the nondeterministic transition relation and the probabilistic transition relation yielding transition systems that can be easily translated into Markov decision processes. The main features of the semantics is that probabilistic transitions take precedence over all other actions and that all processes must synchronize on timed actions. Finally, at any given time, all individuals that may execute an action will do so simultaneously. A full account of the semantics can be found in [28].

# 3 Mean-field semantics for S-PALPS

Using the operational semantics of S-PALPS, we can study the transient dynamics of a system: the time series evolution of the model. This is necessary for the simulation of models that can be obtained by translating S-PALPS to PRISM. Using PRISM it is also possible to use model checking or approximated model checking (i.e., statistical model checking). Although this approach can be effective for S-PALPS models with fairly large state spaces (consisting of populations in the range of a few hundreds of individuals), the size of a state space is exponential in the number of components and locations, and, in epidemiological systems, components can number in millions. To address this challenge, in this section we develop a mean-field semantics for S-PALPS which can be used for reasoning about systems with very large populations.

To compute the mean-field semantics, we proceed in 3 steps: (1) compute the initial-state matrix, (2) compute the state-transition table and (3) compute the mean-field equations. In particular, we begin by assuming an S-PALPS model of the form:

\[
\text{System} = \langle \Pi_{1 \leq j \leq m} P_1; (s_1, \ell_j, q_{1,j}) \rangle \cdots \langle \Pi_{1 \leq j \leq m} P_n; (s_n, \ell_j, q_{n,j}) \rangle \mid L
\]

where \(P_1, \ldots, P_n\) is the set of all processes the populations may evolve into, \(s_1, \ldots, s_n \in S\), \(\ell_1, \ldots, \ell_m \in \text{Loc}\) is the set of all locations in the system, and the \(q_{i,j} \geq 0\) are the sizes of the population of individuals at state \(P_i\) at location \(\ell_j\) where, if a location-state pair \((P_i, \ell_j)\) for some species \(s\) is not present in the initial configuration then \(\text{System}\) includes the component \(P_i; (s, \ell_j, 0)\).

Restrictions for our method are the following: As usual, the numbers of the agents \(q_i\) must be sufficiently large and process constants must be guarded. That is, we do not allow definitions of the form \(C \overset{\text{def}}{=} P|C\), since these yield infinite-sized systems.
1. Initial-state matrix. This matrix, Init, captures the initial configuration of the system under study by noting the number of individuals of each type and at each location. It is a matrix of size $n \times m$, where $n$ is the number of all accessible process-states and $m$ the number of distinct locations in the system (see the definition of System above) and it is obtained directly from the definition of System. Specifically, $\text{Init}[i, j] = q$ where $q$ is the number of individuals of state $P_i$ at location $\ell_j$.

2. State-transition table. This 3-dimensional table, STT, shows how processes evolve from one state to another and their location change. Each entry in the state-transition table is an expression that captures the average evolution, after the execution of a single action of a process $P$ at a location $\ell$ to some process $Q$ at a location $\ell'$. This is expressed as a function of the size of the population of process $P$. Formally, matrix $\text{STT}$ is of size $n \times n \times m$. This matrix captures the evolution after one action step and not necessarily after a time unit, since actions under study may include actions such as $\text{go} \ell$, communication actions and probabilistic actions.

In order to capture the evolution in a manner compatible with the original S-PALPS semantics we employ the following notions: timed($S$) captures whether $S$ may engage in a timed action (all its active components may execute $\sqrt{\cdot}$); prob($S$) captures whether $S$ may engage in a probabilistic actions (at least one of its components may execute a probabilistic action). These notions are essential to capture that probabilistic actions take precedence over all other actions and that $\sqrt{\cdot}$ actions may take place only if all components of the system are willing to synchronize on a timed step.

In particular, given a system $S$, to construct the values of the transition matrix $\text{STT}$ capturing the evolution of its components $P_i:(s_i, \ell_i, q_i)$ we use a function $\llbracket P \rrbracket : \mathcal{P} \mathcal{P} \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P} \mathcal{E} \mathcal{X} \mathcal{P} : \mathcal{P}$, where $\mathcal{P} \mathcal{S}$ is the set of all processes of the form $P:(s_i, \ell_i, q_i)$ and $\mathcal{E} \mathcal{X} \mathcal{P}$ is an expression capturing the evolution in question. We proceed to define this function. We begin with the evolution according to probabilistic transitions and timed actions, where we have:

\[
\llbracket P:(s, \ell, q), a \rrbracket = \begin{cases} \langle \rangle, & \text{if } \text{prob}(S) \\ \llbracket P_i : P_i:(s_i, \ell_i, q_i) | i \in I \rangle, & \text{if } P = \sum_{i \in I} p_i : P_i \end{cases}
\]
\[
\llbracket P:(s, \ell, q), \text{prob} \rrbracket = \langle p_i : P_i : P_i:(s, \ell, q) | i \in I \rangle, \text{ if } P = \sum_{i \in I} p_i : P_i
\]
\[
\llbracket P:(s, \ell, q), \sqrt{\cdot} \rrbracket = \langle P_i : P_i:(s, \ell, q) \rangle, \text{ if } P = \sqrt{P'} \text{ and } \text{timed}(S)
\]
\[
\llbracket P:(s, \ell, q), \sqrt{\cdot} \rrbracket = \langle \rangle, \text{ if } P = \sqrt{P'} \text{ and } \neg\text{timed}(S)
\]

Thus, no communication on channel $a$ may take place if a process occurs within a system satisfying prob($S$). Similarly, a $\sqrt{\cdot}$ action may not take place if the process does not occur within a system satisfying timed($S$). On the other hand, probabilistic transitions may take place freely and so do $\sqrt{\cdot}$ actions within timed systems. Note that in the above, we write $P_i$ for the number of agents $P$ at step $t$.

Moving on to the execution of a movement action, we define:

\[
\llbracket P:(s, \ell, q), \tau_{\text{go}\ell} \rrbracket = \langle P_i : P_i:(s, \ell', q) \rangle, \text{ if } P = \text{go } \ell'. P'
\]

This leaves us with the execution of channel-based actions where we distinguish the following cases:

- If $P = \eta.P'$, where $\eta \in \{a, \bar{a}\}$ and $a \notin L$, that is $a$ does not belong to the set of restricted channels, then we have

\[
\llbracket P:(s, \ell, q), \eta \rrbracket = \langle P_i : P_i:(s, \ell, q) \rangle, \text{ if } P = \eta.P'
\]

- If $P = \eta.P'$ where $\eta \in \{a, \bar{a}\}$ and $a \in L$, then the number of agents evolving to $P'$ depends on the number of agents co-located with $P$ and available to execute action $\eta$ and the complementary action $\bar{\eta}$. Let us write $X_t$ for the number of co-located agents able to execute $\eta$ and $Y_t$ for the number of
co-located agents able to execute the complementary action \( \eta \). If \( Y_t \geq q + X_t \) then all agents of type \( P \) will proceed to state \( P' \). If not, then the mean change in agent \( P \) is expressed as

\[
\frac{\sum_{k=1}^{q} k \binom{q}{k} \left( \frac{X_t}{Y_t - k} \right)}{\sum_{k=1}^{q} \binom{q}{k} \left( \frac{X_t}{Y_t - k} \right)}
\]

This term can be simplified using Vandermonde’s Convolution and standard theory regarding the binomial coefficient to \( \frac{X_t}{Y_t} \). Thus, we have:

\[
\left[ \eta, P : (s, \ell, q), \eta \right] = \langle \min(q, \frac{q \cdot Y_t}{X_t}) : P' : (s, \ell, q) \rangle
\]

- Finally, we have to consider the evolution of a process. In such processes we know that \( \gamma \in \{a, \overline{a}\} \) where \( a \in L \). Thus, the evolution is similar to the previous case. The point in which this case differs is when there is not a sufficient number of collaborating agents to provide the complementary \( \overline{\eta} \) actions. In such a case a number of instances of the process will evolve to \( P_2 \) thus, giving:

\[
\left[ \gamma ? (P_1, P_2) : (s, \ell, q), \overline{\eta} \right] = \langle \min(q, \frac{q \cdot Y_t}{X_t}) : P_1 : (s, \ell, q), (q - \min(q, \frac{q \cdot Y_t}{X_t})) : P_2 : (s, \ell, q) \rangle
\]

3. Mean-field equations (MFES). Using the state-transition table and the initial-state matrix, we can derive a set of recurrence equations that represent the mean-field semantics of a system. The system of recurrence equations contains one variable for each different process and at each location in a system. A variable \( P_i(t) @ \ell_j \) represents the average number of individuals of process \( P_j(s, \ell_j, m') \), at time \( t \) and location \( \ell_j \) and a variable \( P(t-1) @ \ell_j \) represents the number of individuals of process \( P_i \) at location \( \ell_j \) during time \( t - 1 \). Formally, \( P_i(t) @ \ell_j \) is defined by

\[
P_i(t) @ \ell_j = \left\{ \begin{array}{ll}
\text{Init}[i, j] & t = 0 \\
\sum_{1 \leq k \leq n} STT[k][i][\ell_j] & \text{otherwise}
\end{array} \right.
\]

In the first case, for \( t = 0 \), the value is obtained from the initial-state matrix. The second case, for \( t > 0 \), the value is obtained from state-transition from all the processes in the system. According to the state-transition table, processes that do not derive into \( P' : (s, \ell, m') \) are said to produce 0 individuals of process \( P' : (s, \ell, m') \) in the next time unit.

Algorithm 1 is the pseudocode to construct the state-transition table \( STT \). For simplicity, we assume that the entries of the state-transition table are symbolic expressions over the number of individuals of each process.

**Theorem 1.** The complexity of Algorithm 1 is \( O(n^2 \cdot m) \) where \( n \) is the number of different process states existing in the system and \( m \) the number of different locations in the system.

**Proof.** The algorithm consists of a loop that fills in the positions of the 3-dimensional matrix \( STT \). Assuming that by some preprocessing we have the set of processes executing the actions \( \eta \) and \( \overline{\eta} \) of interest, each value of the matrix can be computed at constant time, yielding the result.

**Theorem 2.** Given \( S \)-PALPS, the system of recurrence equations with variables \( P_1(t) @ \ell_1 \), \( \ldots \), \( P_n(t) @ \ell_1 \), \( \ldots \), \( P_n(t) @ \ell_m \) represents the average behavior of the system at any discrete-time unit \( t \).
Algorithm 1 Algorithm to compute the state-transition table (STT) of a S-PALPS model

1: procedure COMPUTE_STT(System) 
2: \( STT = \) matrix of dimension \( n \times n \times m \) initialized with 0 
3: \quad for each \( P_1, \ell_j \) in System do 
4: \quad \quad if \( P = \sum_{j} P_j \) then \( STT[i][j][\ell_j] = p_j \cdot P_1 @ (t - 1), \forall j \in J \) 
5: \quad \quad else if \( P = \sqrt{P_1} \) and timed(System) then \( STT[i][j][\ell_j] = P_1 @ (t - 1) \) 
6: \quad \quad else if \( P = go \) and \( \eta \) then \( STT[i][j][\ell_j] = P_1 @ (t - 1) \) 
7: \quad \quad else if \( \eta = \alpha \) or \( \eta = \pi \) and \( P = \gamma \) then \( STT[i][j][\ell_j] = P_1 @ (t - 1) \) 
8: \quad \quad end if 
9: \quad end for 
10: end procedure 

Proof. The proof will be presented in the post-proceedings version of this paper. 

Example 2. In what follows we present the mean-field semantics of Example 1. In this example we have a total of 7 process states: \( R_1 = P_0, R_2 = go \) (myloc + 1). \( R_3 = go \) (myloc - 1). \( R_4 = \sqrt{P_1}, R_5 = P_1, R_6 = \sqrt{P_0 | P_0}, R_7 = \sqrt{P_0 | P_0 | P_0} \). Furthermore, let us assume that the habitat is a ring of size 4.

1. In the initial-state, there are 2 individuals of process \( P_0 \) at location 1 and 1 individual of process \( P_0 \) at location 2. This is represented in the following \( 4 \times 7 \) initial-state matrix:

\[
\begin{bmatrix}
2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0
\end{bmatrix}
\]

2. Using the methodology defined above, the transition matrix for the example is given below. Note that in fact this is a 3-dimensional matrix, the third dimension being the location dimension. To capture this in two dimensions we write \( @ \ell' \) whenever the resulting individuals have moved to another location, \( \ell' \) being this location.
Let us write $P(t)_{i,\ell}$ for the average number of individuals of process $R$, at time $t$ and location $\ell$. This can be computed as follows: $R_0(t)_{i,\ell}$ is equal to 2 if $i, t, \ell = 1, 0, 1$, equal to 1 if $i, t, \ell = 1, 0, 2$, and 0, if $t = 0$ and $i \neq 1$ or $\ell \notin \{1, 2\}$, whereas if $t > 0$ we have:

$$
R_1(t)_{\ell} = 2 \cdot R_0(t-1)_{\ell} + 3 \cdot R_0(t-1)_{\ell}
$$

$$
R_2(t)_{\ell} = \frac{1}{2} \cdot R_1(t-1)_{\ell}
$$

$$
R_3(t)_{\ell} = \frac{1}{2} \cdot R_1(t-1)_{\ell}
$$

$$
R_4(t)_{\ell} = R_2(t-1)_{\ell}(\ell + 1) + R_3(t-1)_{\ell}(\ell - 1)
$$

$$
R_5(t)_{\ell} = R_4(t-1)_{\ell}
$$

$$
R_6(t)_{\ell} = p \cdot R_5(t-1)_{\ell}
$$

$$
R_7(t)_{\ell} = (1 - p) \cdot R_5(t-1)_{\ell}
$$

By manipulating the equations and restricting attention to how the system evolves between √ actions, we obtain:

$$
R_1(t)_{\ell} = 2 \cdot p \cdot R_5(t-2)_{\ell} + 3 \cdot (1 - p) \cdot R_5(t-2)_{\ell}
$$

$$
R_5(t)_{\ell} = \frac{1}{2} \cdot R_1(t-3)_{\ell}(\ell + 1) + \frac{1}{2} \cdot R_1(t-3)_{\ell}(\ell - 1)
$$

4 Case study: Population dynamics of dengue in Bello, Colombia

An interesting case study where system components can number in millions is the eco-epidemiology of dengue. Dengue is a disease caused by a virus transmitted to humans by the bite of the *Aedes aegypti* mosquito. To date, there is no available treatment nor specific vaccine for this disease. Dengue is a serious public health problem in Colombia. During the last 10 years, there were around 600,000 cases, from which 9% correspond to aggravated forms of the disease [2]. Unfortunately, current programs to prevent and control dengue in Colombia are insufficient [1]. In the Valley of Aburrá (Department of Antioquia), the city of Bello is one of the most affected by dengue. In Bello, dengue is endemic; the rate oscillated from 11.1 to 427 cases by 100,000 inhabitants, during the years 2002-2009.

Given the endemic status of dengue in Bello, it is important to analyze the factors involved in the epidemiology of the dengue disease. Previous results have shown the influence of environmental variables in the distributions of cases of the disease [4]. Ongoing work carried out by three of the authors, Arboleda, Puerta and Vélez, aims to analyze the macro and micro climatic and population factors to determine cases of dengue in Bello\(^1\). There is a disadvantage with such models. Such models are population models based on differential equations; it means, that they analyze the average behavior of the populations, but it is not possible to know how low-level specifications at the individual level will affect the population behavior. Another problem that these models suffer from is that the magnitude orders among the size of the populations is very different. As an example, during the last epidemic, there were around 500,000 inhabitants in Bello, 1,000,000 *Aedes aegypti* mosquitoes, 800 infected mosquitoes and 50 infected persons. We believe that the difference on the magnitudes may lead to inaccuracies on the simulations.

4.1 Spatially-explicit model of dengue in Bello

The model we present is an individual-based version of the model presented in [30]. To establish the initial conditions for the model defined with respect to the human population, we adopted a total population size

---

\(^1\)Research carried out within a project founded by Colombian research agency Colciencias: “Design and Computational Implementation of a Mathematical Model for the Prediction of Occurrences of Dengue”
of 403,235, as recorded for the urban area of Bello (Antioquia) in 2010 by the Colombian administrative
department of statistics 2. The size of the susceptible human population at the beginning of the last
registered epidemic was estimated based on the risk map developed by Arboleda et al. [5], in which the
probability of infection was reported to be 0.3 in 2008 and 2009, with a standard deviation of 0.096;
thus, it was determined that the size of the susceptible human population should be between 244,402 and
321,734. The initial condition considered for the infectious human population was the number of cases
reported at the beginning of the epidemic. There were six reported cases in Bello during the first week of
the epidemic (week 51 in 2009). Because of under-reporting concerns, which can affect up to 75% of the
total number of cases [10], we assumed that the initial number of infectious human individuals should lie
in the range of 6 to 24. In what follows we explain how we model the mosquitoes and the humans.

Mosquitoes. The aquatic phases of the mosquito’s life cycle are described briefly. The egg, larva and
pupa states are represented by \( E, L \) and \( P \), respectively. Parameters \( \sigma_e, \sigma_l, \sigma_p \) are the probabilities to
change from egg to larva, from larva to pupa and from pupa to adult. Co-action \( infect \) models when a
mosquito infects a human. We only consider female mosquitoes in the model. When a mosquito infects a
human, it may reproduce and produce 3 offspring; otherwise, it will die in the next time unit. Mosquitoes
can migrate from one district to another.

Humans. The dynamics of dengue transmission in human population is described by susceptible
(\( s \)), exposed (\( e \)), infectious (\( i \)), recovered (\( r \)) and dead (\( d \)) individuals. Parameter \( \mu_h \) represents the
probability of an infected human to die from the disease. Action \( infect \) represents the action of being
infected by a mosquito. When a human being is infected by a mosquito, it will remain in the exposed
state for 3 time units; afterwards, it will become infectious. For simplicity, we do not consider how
humans may infect mosquitoes when the mosquitoes bite on infected humans; this is left as future work.

\[
\begin{align*}
E & \overset{\text{def}}{=} \sigma_e \sqrt{L} \oplus (1 - \sigma_e) \sqrt{W_0} & s & \overset{\text{def}}{=} \overline{infect}(\sqrt{e}, \sqrt{s}) \\
L & \overset{\text{def}}{=} \sigma_l \sqrt{P} \oplus (1 - \sigma_l) \sqrt{W_0} & e & \overset{\text{def}}{=} \sqrt{e_1} \\
P & \overset{\text{def}}{=} \sigma_p \sqrt{W} \oplus (1 - \sigma_p) \sqrt{W_0} & e_1 & \overset{\text{def}}{=} \sqrt{e_2} \\
W & \overset{\text{def}}{=} \overline{infect}(W_3, \sqrt{W_4}) & e_2 & \overset{\text{def}}{=} \sqrt{i} \\
W_1 & \overset{\text{def}}{=} \sum_{\ell \in \mathbb{N}(\text{myloc})} \frac{1}{4} : go \ell . W_4 & i & \overset{\text{def}}{=} \mu_h \cdot \sqrt{d} \oplus (1 - \mu_h) \cdot \sqrt{r} \\
W_2 & \overset{\text{def}}{=} \overline{infect}(W_3, \sqrt{W_5}) & r & \overset{\text{def}}{=} \sqrt{r} \\
W_3 & \overset{\text{def}}{=} \sqrt{W(E|E|E)} & d & \overset{\text{def}}{=} \sqrt{d} \\
W_4 & \overset{\text{def}}{=} \sqrt{W_2} \\
W_5 & \overset{\text{def}}{=} 0
\end{align*}
\]

Control of dengue. In the literature, there are 4 main mechanisms to control dengue [30]. In what
follows, we briefly explain each mechanism and how they can be included in our model.

1. Mechanical control. Removing eggs, larvae and pupas from the nests. This can be modeled by
varying the parameters \( \sigma_e, \sigma_l \) and \( \sigma_p \), respectively.
2. Biological control. Using parasites or viruses to make mosquitoes infertile. To model this, it will
be necessary to model female and male mosquitoes, their reproduction patterns and the spread of
parasites or viruses on the mosquito’s population.

\[\text{http://www.dane.gov.co} \]
3. **Fumigation.** Using insecticides to kill larvae, pupas and adult mosquitoes. This can be modeled by varying the parameters $\sigma_l$ and $\sigma_p$, and including a new probability modeling the action of adults to die from fumigation.

4. **Vaccination.** Giving a vaccine to the human population. This can be modeled by varying the initial number of susceptible humans to the disease. Although there is currently no vaccine for dengue, there are several ongoing works in this direction.

**Mean-field semantics.** In this case study, we have a total of 17 process states: $E, L, P, W, W_1 ... W_6, s, e, e_1, e_2, i, r, d$, but only $s, e, i, r, W, W_2$ are of interest. Furthermore, there are 11 districts in Bello, Colombia, thus we have 11 locations. Let us write $P(t)@\ell$ for the average number of individuals of process $P$, at time $t$ and location $\ell$. In what follows, we describe $s(t)@\ell$, $e(t)@\ell ... W_2(t)@\ell$, for $t > 0$, which represents the mean-field equations for the behavior of the mosquitoes and humans, respectively. The estimation of the initial populations in each district is left as future work. Note that, for each process of interest, we have 11 equations, one for each location that represents each district in Bello. A detailed explanation on how we computed the mean-field semantics of the case study are presented in [27].

\[
W(t}@\ell = \sigma_p \cdot \sigma_L \cdot \sigma_E \cdot 3 \cdot \left( \min \left( W(t-4}@\ell, \frac{W(t-4}@\ell \cdot s(t-4}@\ell}{W(t-4}@\ell + W_2(t-4}@\ell} \right)
+ \min \left( W_2(t-4}@\ell, \frac{W_2(t-4}@\ell \cdot s(t-4}@\ell}{W(t-4}@\ell + W_2(t-4}@\ell} \right) \right)
\]

\[
W_2(t}@\ell = \frac{1}{4} \cdot \left( W_1(t-3}@\ell' 
- \min \left( W(t-3}@\ell', \frac{W(t-3}@\ell' \cdot s(t-3}@\ell'}{W(t-3}@\ell' + W_2(t-3}@\ell'} \right) \right)
\]

\[
s(t}@\ell = \min(s(t-1}@\ell, W(t-1}@\ell + W_2(t-1}@\ell
\]

\[
e(t}@\ell = e(t-1}@\ell - \min(s(t-1}@\ell, W(t-1}@\ell + W_2(t-1}@\ell
\]

\[
i(t}@\ell = e(t-3}@\ell
\]

\[
r(t}@\ell = r(t-1}@\ell + (1 - \mu_h) \cdot i(t-1}@\ell
\]

where $\ell' \in \text{Nb}(\ell)$

5 **Conclusions**

In this paper we presented a mean-field semantics for $S\text{-PALPS}$. Up to our knowledge, $S\text{-PALPS}$ is the first spatially-explicit probabilistic process calculus to be extended with mean-field semantics. Using this semantics we can analyze deterministically the average behavior of a spatially-explicit ecological model even for large populations. The advantages of this new semantics is that it allows us to translate from an individual-based model to the underlying population dynamics and that it is possible to do this efficiently without computing the complete state space of the model. In particular, we showed how this is applicable for epidemiological models by our case study on the transmission of dengue.

As future work, we want to further study the spatial distribution of dengue on the lines of Otero et al. [19]. In fact, there is demographic and epidemiological information about the reported cases in each district of Bello (Antioquia), Colombia. It is of vital importance for public health to determine which
districts have more risk by determining the migration patterns of mosquitoes from district to district. This is of importance to define vaccination and fumigation schemes to prevent epidemics.

References


Author Index

Amani, Mahdi 4
Arboleda, Sair 39
Ayala-Rincon, Mauricio 1
Basso-Blandin, Adrien 16
Dowek, Gilles 2
Fontana, Walter 16
Harmer, Russ 16
Hermann Haeusler, Edward 27
Nowzari-Dalini, Abbas 4
Philippou, Anna 39
Puerta, Maria 39
Sobocinski, Pawel 3
Toro-Bermudez, Mauricio 39
Vélez, Carlos 39