EPiC Series in Computing

Volume 45, 2017, Pages 96–106

SCSS 2017. The 8th International Symposium on Symbolic Computation in Software Science 2017

Formal Development of Distributed Enumeration Algorithms By Refinement-Based Techniques

Maha Boussabbeh[1,2], Mohamed Tounsi[2],
Mohamed Mosbah[1], and Ahmed Hadj Kacem[2]

[1] LaBRI Laboratory, CNRS, Bordeaux INP, University of Bordeaux, France
{maha.bousabbah,mohamed.mosbah}@u-bordeaux.fr
[2] ReDCAD Laboratory, University of Sfax, Tunisia
{mohamed.tounsi,ahmed.hadjkacem}@fsegs.rnu.tn

Abstract

The enumeration problem addresses a collection of important algorithmic issues related to distributed computations. Among existing solutions, we are interested on the seminal algorithm of *Mazurkiewicz*, based on local computations. Our paper contributes to the design of a *correct-by-construction* enumeration algorithm. The main idea relies upon the development of the problem following a top/down approach that can be supported by an incremental process controlled by the refinement of models. Event-B modelling language is supporting our methodology. Our main objective is to provide a verified component for distributed enumeration in order to be used and extended for solving other problems of distributed algorithms.

1 Introduction

1.1 Overview

Distributed algorithms [14] are designed to run on interconnected autonomous computing entities for achieving a common task: each entity executes asynchronously the same code and interacts locally with its immediate neighbours. Local computations [10], [4], [6], based on graph rewriting, facilitate the design of these algorithms. A distributed system is modelled with respect to the local computations model, by a simple, connected and undirected graph where nodes denote processors and edges denote communication links. Local computations provide an abstract model to reason about problems independently of the network topology.

The naming problem is a fundamental aspect in distributed computing, since it produces a final configuration where all entities have unique identities. Such a configuration is motivated by several computations and applications which require the assumption that all entities should be unambiguously identified. Therefore it constitutes basic initial steps of many other distributed algorithms. Enumeration problem is a variant of the naming problem. It aims to give each entity a unique number between 1 and the size of the graph.

Several solutions for the enumeration problem have been published, among them we consider the seminal algorithm of Mazurkiewicz [11]. Mazurkiewicz's algorithm, based on local

computations, enumerates nodes in an anonymous minimal-covering graph. It is applied on the asynchronous computation model where nodes exchange their states (labels) and accordingly to those, they do a computation step and update local states. In one computation step, labels of nodes are modified on a sub-graph consisting of a node and its neighbours, according to rules depending on this sub-graph only.

1.2 Related Work

Mazurkiewicz's algorithm has been studied to give pertinent solutions to a large family of problems. Among these problems, the distributed election constitutes a building block of many other algorithms. It is clearly related to the enumeration problem. A distributed algorithm solves the election problem if in the final configuration, exactly one node is marked as *elected* and all other nodes are marked as *non-elected*. Given an enumeration, we can obtain an election algorithm by considering the node with the highest/lowest identifier as *elected*. Based on the enumeration algorithm, authors in [7] introduced the notion of *quasi-covering*, i.e., which captures the existence of large enough area of one graph that looks locally like another graph, and characterised families of graphs which admit an election algorithm in the message passing model. Authors in [9] presented a new algorithm, based on Mazurkiewicz's one for the self-stabilizing enumeration. Enumeration problem has been also studied for snapshot problems and detection of stable properties in anonymous networks. Most known snapshot algorithms assume that vertices of network have unique identifier and/or there is exactly one initiator. Based on the enumeration process, authors in [8] explained what stable properties of a distributed system can be computed anonymously with local snapshots with multiple initiators.

Formal methods provide a real help for expressing correctness with respect to safety properties in the design of distributed computing. Nevertheless, specifications of a complex system, become very large and difficult to understand. The *correct-by-construction* [12] approach provides an incremental proof-based process to construct and prove algorithms. Its main idea relies upon a formal development following a top/down approach. This technique can be supported by a progressive and incremental process controlled by the refinement of models. A growing activity on *correct-by-construction* developments has started and is addressing many kinds of case studies and problems in different fields: formalizing snapshot algorithms [3], investigating formal patterns for proof-based development [15], analysing protocols [13], etc.

1.3 Contribution

In this paper, we are interested on the study of Mazurkiewicz's algorithm. We explore the *correct-by-construction* process to formalize enumeration problem in distributed systems. The main objective is to solve the problem using refinement techniques and provide a solution which can be reused for conceiving correct algorithms based on a correct enumeration. We start with an abstract initial specification of the problem and we enrich it gradually by a progressive and incremental refinement. Several refinement steps allow to capture the complete and desired behaviour of the algorithm. The refinement of models is the key element allowing preservation of properties between the different levels of abstraction. The Event-B [1] modelling language is supporting this methodology proposal suggesting proof-based guidelines using the RODIN plateform [2].

1.4 Organization Of The Paper

The paper is organized as follows: Section 2 recalls basic concepts of the local computation models and Event-B method. Section 3 gives a description of Mazurkiewicz's algorithm. Section 4 introduces the refinement process of the algorithm where we describe different abstraction levels. Section 5 details formal specifications of each level. Finally, a short conclusion and ongoing work round the paper up.

2 Preliminaries

2.1 Local Computations Model

In this section, we illustrate, in an intuitive way, the notions of local computations, and particularly those of graph relabelling systems by showing how some algorithms on networks of processors may be encoded within this framework. As usual, such a network is represented by a graph whose vertices stand for processors and edges for (bidirectional) links between processors. At every time, each vertex and each edge is in some particular state and this state will be encoded by a vertex or an edge label. According to its own state and to the states of its neighbours, each vertex may decide to realize an elementary computation step. After this step, the states of this vertex, its neighbours and the corresponding edges may be changed according to some specific computation rules. The graph relabelling systems meet the following requirements:

- they do not change the underlying graph but only the labelling of their components (edges and/or vertices), the final labelling being the result,
- they are local, that is, each rewriting changes only a connected subgraph of a fixed size in the underlying graph. Given G a graph, a subgrph contains a subset of the vertices and edges in G,
- they are locally generated, that is, the applicability condition of the rewriting depends only on the local context of the relabelled subgraph.

2.2 The Modelling Framework

The Event-B [1] modelling language defines mathematical structures as contexts and formal model of the system as machines. The context is defined by abstract sets, constants, and axioms which describe constant properties. An Event-B machine describes a reactive system by a set of invariant properties and a finite list of events modifying state variables. An invariant is defined as a predicate that holds in all reachable states. An event is decomposed into a guard that specifies under which circumstances it might occur and some generalized substitutions called actions that define the state transition associated with the event. A machine M may see a context C, this means that all carrier sets and constants defined in C can be used in M. A context C' can extend a context C, this means that all properties defined in C' are added to C.

Event-B uses a top-down refinement-based approach. The refinement [5] of a specification allows to enrich it in a *step-by-step* fashion. It provides a way to strengthen invariants and add details to a model. It is used to transform an abstract model into a more concrete version by modifying the state description. This is done by extending the list of state variables, by refining

each abstract event into a corresponding concrete version, and by adding new events. In Fig. 1, the diagram illustrates the refinement-based relationship among events and models.

We suppose that an abstract model AM with variables x and invariant $I(x)$ is refined by a concrete model CM with variables y. The abstract state variables, x, and the concrete ones, y, are linked together by means of the so-called *gluing invariant* $J(x, y)$. A number of proof obligations ensure that (1) each abstract event of AM is correctly refined by its corresponding concrete version of CM, (2) each new event of CM refines *skip*, which is intending to model *hidden* actions over variables appearing in the refinement model CM. More formally, if $BA\_ae(x, x')$ and $BA\_ce(y, y')$ are respectively the abstract and concrete before-after predicates of events, we say that ce in CM refines ae in AM or that ce simulates ae, if one proves the following statement corresponding to proof obligation: $I(x) \wedge J(x, y) \wedge BA\_ce(y, y') \Rightarrow \exists x' \cdot (BA\_ae(x, x') \wedge J(x', y'))$. To summarize, refinement guarantees that the set of traces of the abstract model AM contains (modulo stuttering) the traces of the concrete model CM.

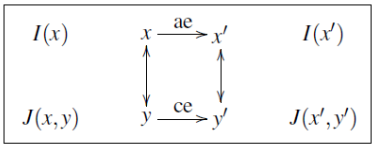

Figure 1: Refinement-Based Relationship

3 A Distributed Enumeration Algorithm

Distributed enumeration algorithms are considered to solve distributed naming problems. The aim of such algorithms is to attain a final configuration where all processes have unique identities. In this section we present a description of an enumeration algorithm given by Mazurkiewicz [11].

Given G an undirected graph, i.e., no distinction between two nodes associated with an edge; $N(G)$ stands for the nodes of G with a cardinality denoted by $|N(G)|$. During the execution of the algorithm, each node x attempts to get its own unique identity which is a number between 1 and $|N(G)|$. Initially every node is defined by a 0-identity: it does not get a number yet. Once a node x has chosen a number $n(x)$, it sends it to each neighbour y. When y receives a message from x, it stores the number $n(x)$. A *Local view* can be constructed by each node x of the graph, containing all numbers of its neighbours. The set of *Local views* of neighbours numbers stands for the *MailBox* of x. Each node broadcasts its number and its *MailBox*. If a node x discovers the existence of another node y with the same number (identity), then it should decide if it changes its identity: it compares its *Local view* with the *Local view* of y. If the *Local view* of y is *weaker*, then y picks another number and broadcasts it with its *Local view*. At the end of the execution, we can construct a graph H whose vertices are defined by all G node identities. If G is a minimal covering of H, then the algorithm terminates correctly and every node has a unique number.

A total order should be defined on *Local views* of nodes: let $N1$ and $N2$ be respectively the set of *Local view* of x and y. Let E be the set of elements of the *symmetric difference* of $N1$ and $N2$. $E = (N1 \setminus N2) \cup (N2 \setminus N1)$. Formally, $N1$ is *weaker* than $N2$ if the maximum of elements of E belongs to $N2$.

During the execution, the label of each node x is a tuple $(n(x), Lv(x), Mb(x))$ where: $n(x)$ is the current number of x computed by the algorithm; $Lv(x)$ is the *Local view* of x. It is a set of

identities of nodes connected to x. $Mb(x)$ is the *MailBox* of x. It is a set of neighbour identities and their *Local views*.
Based on local computations [10], Mazurkiewicz [11] defines some computation rules applied on nodes in order to change their numbers, when needed, and diffuse their own information. Given x a node of the graph, the *ball* around x, denoted by $B(x)$ is the set of nodes covering x and its neighbours. Mazurkiewicz distinguishes two rules applied on $B(x)$ as follows:
The *Renaming Rule* describes the instruction that a node has to follow when the *Mailbox* of elements of $B(x)$ is not modified; and it has not a number yet, or when it discovers the existence of another *Local view* N' greater than its own. In such cases, this node chooses another number, updates its *Local view* by removing elements which corresponds to its old number and adding the chosen one. Then, every node of $B(x)$ should update their *Mailbox*.

Renaming Rule:

Pre-conditions:

$\forall y \in B(x), Mb(y) = Mb(x)$

$(n(x) = 0) \vee (n(x) > 0 \wedge \exists (n(x), N') \in M(x) | N(x) < N')$

Post-conditions:

$n'(x) = 1 + max\{n | (n, N) \in M(x)\}$

$\forall y \in B(x) \setminus \{x\}, N'(y) = (N(y) \setminus \{n(x)\}) \cup \{n'(x)\}$

$\forall y \in B(x), M'(y) = M(y) \cup \{n'(w), N'(w)) | w \in B(x)\}$

If the *Mailbox* has been modified, then nodes of $B(x)$ should diffuse their informations and update their *Mailbox*.

Diffusion Rule:

Pre-conditions:

$\exists y \in B(x), Mb(y) \neq Mb(x)$

Post-conditions:

$\forall y \in B(x), M'(y) = \bigcup_{w \in B(x)} M(w)$

4 Incremental Development

In this section we present the complete refinement process (Figure. 2) which starts from GLOBAL-ENUM and NETWORK and progressively leads to LOCAL-ENUM-PROCESS. NETWORK is an event-B context specifying the application field of the enumeration algorithm. The network is represented by a graph whose vertices stand for processors and edges for (bidirectional) links between processors. The network is supposed to be fixed (edges and nodes are not modified or created or deleted). Moreover, it should be noted that Mazurkiewicz's algorithm terminates correctly on graphs that are simple, connected, undirected and minimal for the covering relations. The GLOBAL-ENUM model is an abstract event-B machine specifying the result of the algorithm in one shot (Figure. 3) without specifying the algorithmic process: each node picks a unique identity. The GLOBAL-PROGRESS-ENUM model is an event-B machine refining GLOBAL-ENUM. It introduces in a high level abstraction the basic idea for computing the result in a progressive way. In this level, we take a global view on the graph and we fix a number called $maxId$. This number is initialized to 0 and reaches gradually the cardinality of the graph $|N(G)|$. The progressive aspect comes from the fact that all numbers that are less than $maxId$ should be allocated. By this way, we ensure the *surjective* property of the enumeration. The LOCAL-PROGRESS-ENUM model is an event-B machine refining the previous level. It takes a local view on the graph and explains how the the progressive computation is working on *balls* of the graph. This level ensures the *injective* property of the enumeration restricted to the *balls* of the graph. The LOCAL-RENAM-PROCESS model is an event-B machine refining

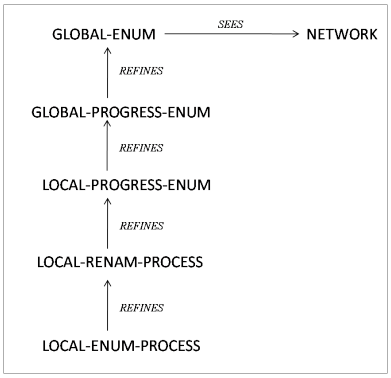

Figure 2: A Refinement Process

the previous one. It introduces an abstract definition of the *Renaming rule* and explains how nodes can construct their *Local views* from their neighbours. The LOCAL-ENUM-PROCESS model refines the LOCAL-RENAM-PROCESS and ensures the correctness of the algorithm. It computes through an event-B machine a concrete definition of the *Renaming* and the *Diffusion* rules of the algorithm. *MailBox* of nodes are constructed in this level.

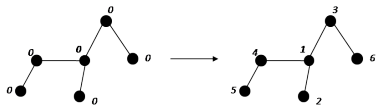

Figure 3: Enumeration of nodes in one shot

5 Formal Development

5.1 Network Specification

In this section we introduce formal specifications of a network on which the distributed enumeration algorithm can be executed. An undirected graph means that there is no distinction between two nodes associated with an edge (see $axm4$). A graph g is simple, if it has at most one edge between any two nodes (see $axm2$ and $axm3$) and no edge starts and ends at the same node (see $axm5$); it is obviously expressed by the choice of the relation representation by a set. The domain restriction $ND \lhd id$ is a subset of the relation id that contains all of the pairs whose first element is in ND; id is the identity relation that maps every element to itself. A graph (directed or not) is connected, if for each pair of nodes, there exists a path joining these two nodes (see $axm6$). A connected graph g over a set of finite nodes ND (see $axm1$) can be presented as follows:

$$
\begin{array}{|l|}
\hline
axm1 : finite(ND) \\
axm2 : g \subseteq ND \times ND \\
axm3 : dom(g) = ND \\
axm4 : g = g^{-1} \\
axm5 : ND \lhd id \cap g = \varnothing \\
axm6 : \forall s \cdot s \subseteq ND \wedge s \neq \varnothing \wedge g[s] \subseteq s \Rightarrow ND \subseteq s \\
\hline
\end{array}
$$

It has been proved that enumeration algorithms cannot be running on *ambiguous* graphs [11]. The notion of ambiguity is formulated equivalently using coverings of simple graphs.

Mazurkiewicz's algorithm terminates correctly on graphs that are minimal for the covering relations: they can cover only themselves. Formally, a graph g is a covering of another graph H, if there is a *surjective homomorphism phi* from g to H (*axm7: line 2*) which is locally bijective (*axm7: lines 3, 4 and 5*). g is minimal if *phi* is an *isomorphism*: *phi* is a bijection (*axm7: line 6*). Therefore, we add the following axiom to ensure that if g is a covering of H, then g and H are *isomorphic* via the homomorphism *phi*

$$\begin{array}{l} axm7 : \forall H, phi, N, v \cdot (v \in ND \land N \subseteq \mathbb{N} \land H \subseteq N \times N \land \\ \quad (phi \in ND \nrightarrow N \land \\ \quad \forall x, y \cdot (x \mapsto y \in g \Rightarrow phi(x) \mapsto phi(y) \in H)) \land \\ \quad phi[g[\{v\}]] = H[phi[\{v\}]] \land \\ \quad (\forall y, z \cdot \{y, z\} \subseteq g[\{v\}] \cup \{v\} \land y \neq z \Rightarrow phi(z) \neq phi(y)) \land \\ \quad card(g[\{v\}]) = card(H[phi[\{v\}]])) \\ \quad \Rightarrow phi \in ND \rightarrowtail N \end{array}$$

5.2 The Initial Model

The first level of the model expresses only the goal of the distributed algorithm and does not describe how the solution is computed. This level can be defined by an abstract machine modelling the result in one shot. It computes through one event (*OneShot*) a configuration where all network nodes have unique identities. Formally, this enumeration can be specified as a bijection ($\rightarrowtail$) from the set of nodes $N(G)$ to $\{1, 2, ..., |N(G)|\}$. If ND is an abstract set specifying $N(G)$, $card(ND)$ stands for $|N(G)|$. *enumeration* is an abstract variable introduced for computing the final configuration of the network. It is a function computed on ND which, starting with a uniform identity of nodes, say 0 (see the $INITIALIZATION$ event), eventually terminates with a bijection being an enumeration of nodes (see the *OneShot* event).

$$\begin{array}{l} \textbf{EVENT} \quad INITIALISATION \\ \quad act1 : enumeration :\in ND \rightarrow \{0\} \\ \textbf{EVENT} \quad OneShot \\ \quad act1 : enumeration :\in ND \rightarrowtail 1 \mathinner{..} card(ND) \end{array}$$

5.3 The First Refinement

This level remains in a high level abstraction. It refines the previous one and computes the result in a progressive way. New variables are introduced: *enum* is a function computed on ND, being the enumeration under definition; and $maxId$ is an integer, initialized to 0, encoding the largest value of the current nodes identities. Let n be a node, we assume that $enum(n)$ does not exceed $maxId$. These variables are specified as follows:

$$inv1 : maxId \in 0 \mathinner{..} card(ND) \quad \land \quad enum \in ND \rightarrow 0 \mathinner{..} maxId$$

The idea is to define nodes identities in a progressive way ensuring that all values being in $[0..maxId]$ or in $[1..maxId]$ are currently allocated. For the sake of simplicity, we distinguish two cases:

- the 0-identity still exists in the current configuration. Formally, we ensure that $enum \rhd \{0\} \neq \varnothing$. Note that $enum \rhd \{0\}$ is a *range restriction* that contains all of the pairs that are in *enum* and whose second element is equal to 0. In such a case, we ensure the following invariant ($inv2$): *enum* is a *total surjection* ($\twoheadrightarrow$) from ND to $0..maxId$. It is *surjective*; means that for every element of $0..maxId$ there exists an element in ND that is mapped to it; and it is total means that its domain contains all elements of ND.

$$inv2 : enum \rhd \{0\} \neq \varnothing \Rightarrow enum \in ND \twoheadrightarrow 0 \mathinner{..} maxId$$

- the 0-identity does not exist in the current configuration: $enum \rhd \{0\} = \varnothing$. In such a case, we ensure the following invariant ($inv3$): *enum* is a *total surjection* from ND to $1..maxId$.

$$inv3 : enum \rhd \{0\} = \varnothing \Rightarrow enum \in ND \twoheadrightarrow 1 \mathinner{..} maxId$$

Consequently, we can ensure the following properties:

- the minimum value of the current node identities is either a 0-identity or a 1-identity ($th1$).
- if there exists an identity that is mapped to two different nodes equally, then the current configuration of node identities does not contain the value of $card(ND)$ yet. More precisely, the value of $maxId$ is strictly less than $card(ND)$ ($th2$).

$$\begin{array}{|l} th1 : min(ran(enum)) \in \{0,1\} \\ th2 : \forall x,y \cdot \{x,y\} \subseteq ND \wedge x \neq y \wedge enum(x) = enum(y) \Rightarrow maxId < card(ND) \end{array}$$

A new event $enumProgress$ is introduced to change node identities and update $maxId$ during enumeration. If the current configuration contains a node, say x, that is still in its initial state (0-identity) ($grd2$), or if all nodes changed their identities ($enum \rhd \{0\} = \varnothing$) but the current configuration assigns a same identity to two different nodes: x and another one, say y, $y \in ND \setminus \{x\}$ ($grd2$); then x is mapped to an another identity, say n ($act1$). The new value n is selected in such a way that the range of $enum$ increases towards covering all values being in $[1..card(ND)]$. To this end, we ensure that $n \in enum(x) + 1 \,..\, maxId + 1$ ($grd4$). In such a case, $maxId$ should not obviously exceed $card(ND)$ ($grd3$). If the new value n exceed the largest value of the current node identities, $maxId$ sets to the value of n. In another words, we ensure that the new value of $maxId$ takes the largest identity $max(\{maxId, n\})$ ($act2$).

$$\begin{array}{|l} \textbf{EVENT} \quad enumProgress \\ \textbf{any} \quad x, n \\ \textbf{where} \\ \quad grd1 : x \in ND \\ \quad grd2 : (enum(x) = 0) \vee (enum \rhd \{0\} = \varnothing \wedge (\exists y \cdot y \in ND \setminus \{x\} \wedge enum(x) = enum(y))) \\ \quad grd3 : maxId < card(ND) \\ \quad grd4 : n \in enum(x) + 1 \,..\, maxId + 1 \\ \textbf{then} \\ \quad act1 : enum(x) := n \\ \quad act2 : maxId := max(\{maxId, n\}) \end{array}$$

5.4 The Second Refinement

For any node x of the graph g, we take a local view on the *ball* around x, denoted $B(x)$, and we prove an *injection* from elements of $B(x)$ to their identities, excepting nodes that are still mapped to 0: two distinct nodes of $B(x)$ are always mapped to distinct identities. For instance, as presented in Fig. 4, $B(x)$, covers the nodes, x, z, y and t. The *ball* around y, denoted $B(y)$ covers y, x, k and w. In such a case, different inequalities should be ensured between node identities. For example: $enum(x) \neq enum(y)$; $enum(x) \neq enum(k)$; $enum(y) \neq enum(z)$; $enum(w) \neq enum(x)$, etc. We can say that every node should not be mapped to an identity assigned to one of its neighbours being in a distance ≤ 2. More formally, we model these

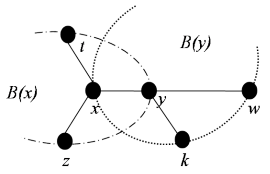

Figure 4: Balls in the graph

requirements with the following invariants and theorems: given x a node of the graph g, $g[x]$ is a subset of ND containing all of neighbours of x. Fore every connected nodes of the graph g, say x and y, such that $enum(x) = enum(y)$, we have to ensure that these nodes did not change their identities yet. They still have the 0-identity ($inv4$). If there is another node z, connected to x, such that $z \neq y$ and $enum(z) = enum(y)$, then $enum(z) = 0$ ($inv5$).

$inv4 : \forall x, y \cdot x \mapsto y \in g \wedge enum(x) = enum(y) \Rightarrow enum(x) = 0$
$inv5 : \forall x, y, z \cdot z \in g[\{x\}] \wedge y \in g[\{x\}] \wedge z \neq y \wedge enum(z) = enum(y) \Rightarrow enum(z) = 0$

Consequently, we can prove that for every other node w connected to y, such that $w \neq x$ and $enum(w) = enum(x)$, then $enum(w) = 0$ ($th3$). Therefore, we prove that for every node x, the function *enum* computed on the *ball* around x, excepting nodes with 0-identity, is *injective* ($th4$). The *domain restriction* of *enum* on $B(x)$ is specified by $(\{x\} \cup g[\{x\}]) \lhd enum$. The *range restriction* on non 0-identities is specified by $((\{x\} \cup g[\{x\}]) \lhd enum) \rhd \{0\}$. The theorem $th4$ ensures that the function *enum* with its *domain/range restriction* belongs to a *partial injection* ($\rightarrowtail$), from ND to $1 \,..\, maxId$. Note that the *partial* property of this function stems from the fact that the *ball* around x is a subset of ND.

$th3 : \forall x, y, w \cdot x \in g[\{y\}] \wedge w \in g[\{y\}] \wedge w \neq x \wedge enum(x) = enum(w) \Rightarrow enum(w) = 0$
$th4 : \forall x \cdot x \in ND \Rightarrow ((\{x\} \cup g[\{x\}]) \lhd enum) \rhd \{0\} \in ND \rightarrowtail 1 \,..\, maxId$

No new variables are introduced in this model. We refine the *enumProgress* event of the previous machine by two events: *enum*0 and *enum*. The first one models the case where a node does not get an identity yet ($grd2$). Guards are strengthened by $grd5$ to ensure that x will take a different value comparing to its *neighbourhood* identities. For any node x of the graph g, $g[x]$ is a subset of ND containing all of neighbours of x; $g[g[x]]$ is the *neighbourhood* of x. It is a subset of ND containing elements of $g[x]$ and their neighbours.

EVENT *enum*0
refines *enumProgress*
any x, n
where $grd1 : x \in ND$
$grd2 : enum(x) = 0$
$grd3 : maxId < card(ND)$
$grd4 : n \in enum(x) + 1 \,..\, maxId + 1$
$grd5 : n \notin enum[g[\{x\}]] \cup enum[g[g[\{x\}]]]$
then
$act1 : enum(x) := n$
$act2 : maxId := max(\{maxId, n\})$

The second event *enum* models the case where two connected nodes get the same identity ($grd2$). Similarly to the previous event, guards are strengthened by $grd5$ to ensure that x will take a different value comparing to its *neighbourhood* identities.

EVENT *enum*
refines *enumProgress*
any x, n
where $grd1 : x \in ND$
$grd2 : enum(x) \neq 0 \wedge (\exists y \cdot y \in g[\{x\}] \wedge enum(x) = enum(y))$
$grd3 : maxId < card(ND)$
$grd4 : n \in enum(x) + 1 \,..\, maxId + 1$
$grd5 : n \notin enum[g[\{x\}]] \cup enum[g[g[\{x\}]]]$
then
$act1 : enum(x) := n$
$act2 : maxId := max(\{maxId, n\})$

5.5 The Third and Fourth Refinements

The third level describes the instructions that every node of $B(x)$ has to follow when x modifies its identity. Events of the previous level, *enum*0 and *enum*, are refined respectively by two other events *enum*0′ and *enum*′. Each refined event models the updating action of the *Lcoal view* of $B(x)$ elements. When x changes its number from $enum(x)$ to n, then every node y, connected to x must update its *Local view* from $Lv(y)$ to $Lv'(y)$ by removing $enum(x)$ and adding n to $Lv(y)$. From all information it can be gathered from neighbour identities and their *Local views*, every node x of the graph can construct its *MailBox*, denoted by $Mb(x)$. The last level introduces the instructions that a node has to follow to construct its *MailBox* and make it up to date at every computation step. Events of the previous level, *enum*0′ and *enum*′, are refined respectively by

*renaming*0 and *remaining*. The first event models the first case of the *Renaming rule*, i.e., when a node has a 0-identity.

The *remaining* event refines $enum'$ and models the second case of the *Renaming rule*, i.e., a node x discovers the existence of another node y mapped to the same identity as $n(x)$. In such a case, x should compare its *Local view* $Lv(x)$ with the *Local view* stored in its *Mailbox* ($N0$), and decide if it picks another number or not. A new event $Diffusion$ is introduced to compute the $Diffusion$ rule. Finally, we can strengthen the guards of the *Oneshot* event in order to ensure requirements that yields to a final enumeration: if the $B(x)$ elements have the same *Mailbox*; and if for every node x of the graph $enum(x) \neq 0$; and if the identity of x is already stored in its *Mailbox* and mapped to another *Local view*, *weaker* than $n(x)$; then all events are blocked excepting the *Oneshot* that should yield to a bijection. In order to ensure the correctness of the bijection result, we verify that for every graph say H constructed from g node identities, such that *enum* is a *surjective homomorphism* from g to H, then g and H are *isomorphic* via *enum*. A detailed formal development of our approach is available[1].

6 Conclusion and Future Work

Throughout this paper, we investigated Mazurkiewicz's algorithm, based on local computations, and we proposed an incremental reasoning for managing the complexity of the development and preserving properties between the different abstraction levels. These levels are related by refinement, so that the properties and the behaviour of any model are kept in all its subsequent refinements. This technique supports the step-by-step development and provides an easy way to carry on the correctness of the proofs and to validate the integration of requirements. During development, a set of so-called proof obligations is generated by the Rodin platform. Some of these proofs are discharged automatically by Rodin provers and some need to be discharged interactively.

The computed result constitutes basic initial steps of many other distributed algorithms. Therefore, it can be used, by composition with other algorithms, to give solutions of distributed problems. Combining verified components is one way to overcome the complexity of proving a distributed algorithm. Therefore, numerous future works can be done on this area. As a part of our other efforts, we investigate snapshot problems with multiple initiator using the studied enumeration algorithm, by referring to new theories published in [8]. Finally, we think that investigating enumeration problems with complex distributed systems like mobile networks would be an interesting case study.

References

[1] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering.* Cambridge University Press, 2010.

[2] Jean-Raymond Abrial, Michael J. Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in event-b. *STTT*, 12(6):447–466, 2010.

[3] Manamiary Bruno Andriamiarina, Dominique Méry, and Neeraj Kumar Singh. Revisiting snapshot algorithms by refinement-based techniques. *Comput. Sci. Inf. Syst.*, 11(1):251–270, 2014.

[1] http://visidia.labri.fr/html/formal-development.html

[4] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 82–93, 1980.

[5] Ralph Johan Back. A calculus of refinements for program derivations. *Acta Inf.*, 25(6):593–624, 1988.

[6] Paolo Boldi and Sebastiano Vigna. An effective characterization of computability in anonymous networks. In *Distributed Computing, 15th International Conference, DISC 2001, Lisbon, Portugal, October 3-5, 2001, Proceedings*, pages 33–47, 2001.

[7] Jérémie Chalopin, Emmanuel Godard, and Yves Métivier. Election in partially anonymous networks with arbitrary knowledge in message passing systems. *Distributed Computing*, 25(4):297–311, 2012.

[8] Jérémie Chalopin, Yves Métivier, and Thomas Morsellino. On snapshots and stable properties detection in anonymous fully distributed systems (extended abstract). In *Structural Information and Communication Complexity - 19th International Colloquium, SIROCCO 2012, Reykjavik, Iceland, June 30-July 2, 2012, Revised Selected Papers*, pages 207–218, 2012.

[9] Brahim Hamid and Mohamed Mosbah. A local enumeration protocol in spite of corrupted data. *JCP*, 1(7):9–20, 2006.

[10] Igor Litovsky, Yves Métivier, and Eric. Sopena. Handbook of graph grammars and computing by graph transformation. chapter Graph Relabelling Systems and Distributed Algorithms, pages 1–56. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.

[11] Antoni W. Mazurkiewicz. Distributed enumeration. *Inf. Process. Lett.*, 61(5):233–239, 1997.

[12] Dominique Méry. Refinement-based guidelines for algorithmic systems. *Int. J. Software and Informatics*, 3(2-3):197–239, 2009.

[13] Dominique Méry and Neeraj Kumar Singh. Analysis of DSR protocol in event-b. In *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings*, pages 401–415, 2011.

[14] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

[15] Mohamed Tounsi, Mohamed Mosbah, and Dominique Méry. Proving distributed algorithms by combining refinement and local computations. *ECEASST*, 35, 2010.