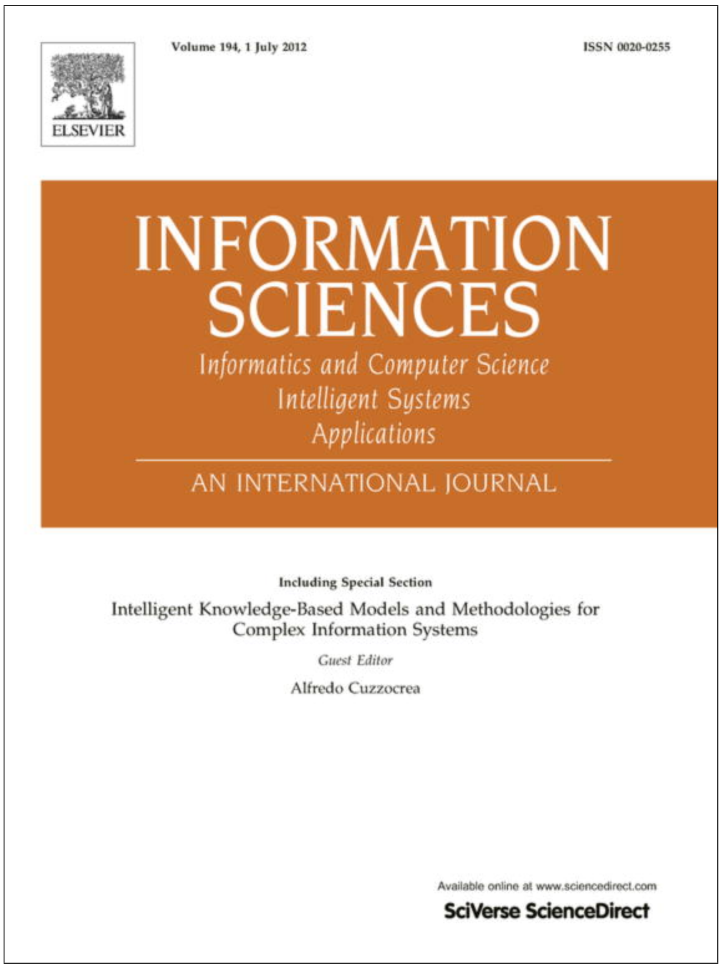

Information Sciences 194 (2012) 120–137

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Investigations on Stochastic Information Control Nets

Clarence A. Ellis[a], Kwanghoon Kim[b,*], Aubrey Rembert[c], Jacques Wainer[d]

[a] Collaboration Technology Research Group, Department of Computer Science, University of Colorado at Boulder, Campus Box 0430, Boulder, CO 80309-0430, USA
[b] Collaboration Technology Research Lab., Department of Computer Science, Kyonggi University, Suwon-si Kyonggi-do 443-760, South Korea
[c] IBM TJ Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA
[d] Institute of Computing, University of Campinas, Campinas, SP, Brazil

A R T I C L E I N F O

Article history:
Available online 28 July 2011

Keywords:
Stochastic workflow analysis
Quantitative organizational process analysis
Probabilistic languages
Information Control Nets
Knowledge mining
Fidelity

A B S T R A C T

Information Control Nets have been well used as a model for knowledge mining, discovery, and delivery to increase organizational intelligence. In this document, we extend the notions of classic Information Control Nets [15] to define new concepts of **Stochastic Information Control Nets**. We introduce a simple and useful AND-probability semantic and show how this probabilistic mathematical model can be used to generate probabilistic languages. The notion of a probabilistic language is introduced as a normalizer for comparisons of organizational knowledge repositories to organizational models. We discuss model-log conformance and present a definition of **fidelity** of a model. We show how to manipulate the residual error factor of this model. We describe a set of recursive functions and algorithms for generation of probabilistic languages from stochastic ICNs. We prove an important aspect of our generation algorithms: they generate probabilistic languages that are normalized. Since ICN models with loops generate infinitely many execution sequences, we present new notions of **most probable sequence generation**, and ε-**equivalent approximation languages**. These definitions can be applied to many aspects of organizational modeling including the process, the informational, and the resource perspectives. The model that we introduce here can be used to augment and expand on analyses that have been useful and insightful within varied enterprise information systems modeling and organizational analysis applications.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

The design, analysis, development, and successful evolution of today's complex organizations require intelligent knowledge-based models and methodologies that capture and integrate the many aspects of a multi-dimensional dynamically changing organization. A 2009 Gartner paper on the future of business intelligence states "Because of lack of information, processes, and tools, more than 35% of the top 5000 global companies regularly fail to make insightful decisions about significant changes in their business and markets" [19]. We believe that this Gartner statement supports the relevance and importance of the model and research work described in this document. Models must be intelligent, and based upon a flexible, time proven framework. We are concerned with models that support the concepts of knowledge mining [16].

The Information Control Net (ICN) is an open-ended meta-model conceived over 30 years ago as a family of models for organizational understanding, description, analyses, and implementation [14,15]. Many enterprise information systems have

* Corresponding author.
E-mail addresses: skip@cs.colorado.edu (C.A. Ellis), kwang@kgu.ac.kr (K. Kim), aubrey.rembert@gmail.com (A. Rembert), wainer@ic.campinas.br (J. Wainer).
URL: http://www.ccsrc.kgu.ac.kr (K. Kim).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.07.031

been modeled using ICNs. The steady evolution of the ICN metamodel over the years has made available many augmentations (e.g. colored ICNs [27]) and many perspectives (e.g. actor-oriented perspective [20]). The ICN consists of perspectives that include the process, the informational, and the resource perspectives. It allows business systems analysis that transcends the single perspective limits in a powerful and uniform manner [10]. In this document, we introduce Stochastic Information Control Nets (s-ICNs) as a new probabilistic extension of ICNs. This allows us to do both qualitative and quantitative information systems analyses, as illustrated later in this document. The main thrust of this document is concerned with presentation and characterization of Stochastic Information Control Nets and their probabilistic languages. The document presents theorems, functions, and algorithms which generate normalized probabilistic languages from s-ICNs. For the cases where these languages are infinite, we present theorems and algorithms to generate and evaluate approximation languages, and to generate maximally probable strings.

Organizations have multiple models – some may be explicit and formal; others are implicit and informal [1]. Some models focus on data processing, some on process modeling, some on people and social networks, etc. Members of the ICN family of models have been used for modeling in numerous of these domains. In this document, we focus upon the process flow perspective. Organizations have multiple historical information repositories [17]. For example, some may be transaction logs, communication records, audit trails, etc. In this document, we are especially interested in the logs (audit trail) of organizational process events. By noting the relative frequency of various event types in such a log, we can assign occurrence probabilities to these event types. By assigning probabilities to the AND/OR forks in our ICN process model, we can calculate the probability of every execution sequence of the model. Then we can compare these two probability distributions. This calculation and comparison can be complex and non-obvious when the model has loops (infinite sequences), and the log is quite large with anomalous entries [6]. There are cases where the quantities are incomparable. If this comparison is successful, then we can define metrics such as the *fidelity* of a model with respect to a log to quantitatively gauge our conceptual grasp of the organization.

In this document, we present concepts and formal definitions of the s-ICN model, and of probabilistic languages. We also present theorems, algorithms, and proofs for language generation, analysis, and log comparison. Previous model-log comparisons in the literature have not included probabilistic considerations as we introduce. This document introduces new concepts of *fidelity*, *ε-equivalent approximation languages*, and *residual error factors*.

In Section 2, we present the ICN Meta-Model. Section 3 presents a summary of related work. Section 4 introduces our Stochastic Information Control Nets with s-ICN formal definitions in Section 4.2. Section 4.3 defines the concept of a probabilistic language (*p-Language*) and model-log comparison. We introduce the new concept of model fidelity. In Section 5, we explain our algorithms to generate *p-Languages* from s-ICNs. Section 6 is concerned with approximation and manipulation of *p-Languages*. In Section 7, we present summary and conclusions followed by acknowledgements and bibliography.

2. The ICN Meta-Model

Different organizations have different goals, different structures, and different organizational styles. Therefore, different organizations typically need different methodologies, different workflow products, and different models to express different business perspectives. The concept of a meta-model provides a coherent, uniform notation and a set of conceptual tools for creating various models appropriate to various organizations. The Information Control Net Meta-Model represents a family of models that have been designed and used to model various aspects of organizations and business process management.

The ICN modeler chooses certain objects of interest and structures from an organizational framework, from an organizational schema, and from an organizational net. The organizational framework is used to specify various classes of organizational objects (e.g. goals, constraints, resources, tasks). The organizational schema is used to specify the set of mappings over the classes of organizational objects (e.g. who does what, which tasks precede which). The organizational net is used to specify the dynamic behavior of an organization [14].

As an example, the data flow perspective is formed when we impose relationships between three dimensions: tasks, data items, and repositories. The data dependence mapping is one of the most interesting in this perspective. It reveals the data dependencies of tasks. For example, the review university admissions application task is data dependent on the repository that stores all of the university admissions applications, and the actual data within that repository.

Another example of interest is the task assignment perspective formed by defining a set of relationships between three dimensions: employees, roles, and tasks. Depending on the size and nature of an organization, the dimensions involved in the definition of this perspective can vary. For a small organization, with say two people and a relatively simple process, it is quite adequate and convenient to relate participants directly to the tasks they perform. However, in organizations with large complex organizational structures and processes, this type of relationship is not practical; it is more appropriate to relate tasks to roles, then relate those roles to participants. Therefore, through one level on indirection, tasks are related to participants.

The probabilistic analyses that are a focus of this document can be applied to all of the above perspectives, and others [27], however we will limit our further considerations to the process flow perspective. Process models of organizations are common and useful within complex enterprises. They are important for organizational understanding, analysis, and evaluation. They are also instrumental for design, deployment, and enhancement/evolution of workflow management systems. Streamlining, critical path analysis and workload analysis are all examples of useful analyses that can be performed using this perspective [15,17,22].

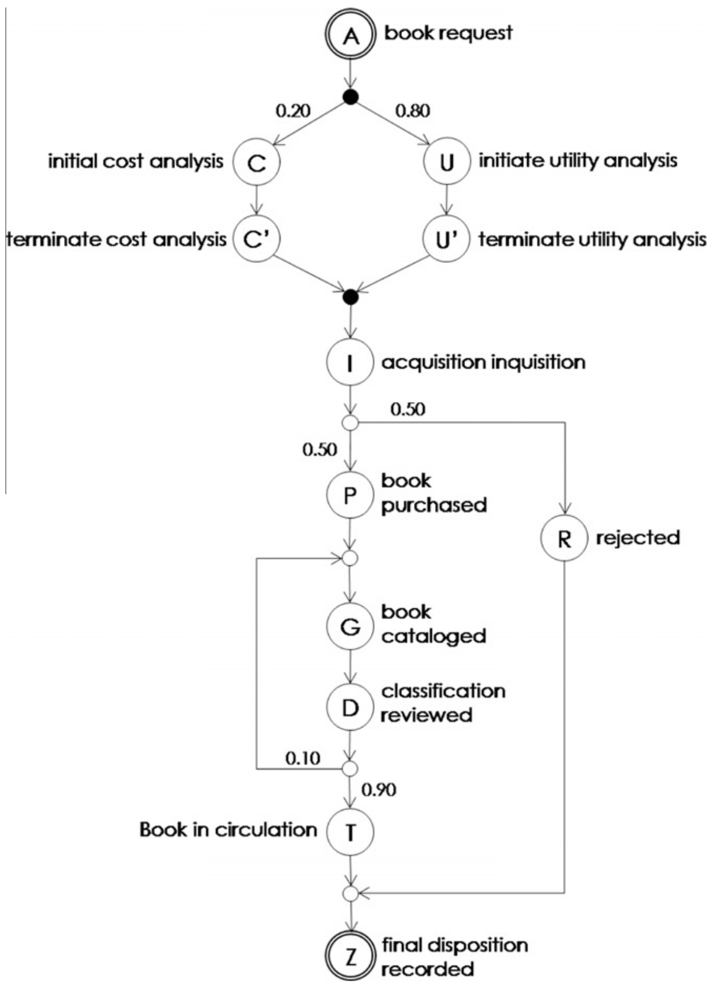
A book request
0.20
0.80
initial cost analysis C
U initiate utility analysis
terminate cost analysis C'
U' terminate utility analysis
I acquisition inquisition
0.50
0.50
P book purchased
R rejected
G book cataloged
D classification reviewed
0.10
0.90
Book in circulation T
Z final disposition recorded

Fig. 1. Library book acquisition process.

In the application domains of business process management and workflow analysis, the potential implications of this research are enormous. This research opens the door for more quantitative analyses, and therefore more in-depth understanding of the workings (and failures [36]) of organizations. In the target organization referenced in this document, it was possible for a large organization (the University of Colorado library) to compare and contrast their formal procedures (represented by s-ICNs) with the actual daily workings of their organization (represented by workflow logs). In some procedures, this yielded a validation of well developed smooth teamwork; in other procedure analyses, there were surprises showing that the actual work was quite different from management's specification and expectation.

The classic definition of "basic ICN" within the process flow perspective [15] defines an AND/OR graph in which action nodes, denoted by large circles, can have one incoming arc and one outgoing arc. All arcs are directed edges from one node to another, and represent precedence. There are two exceptions: the entry arc has no from-node, and the exit arc has no to-node. In the graphic, these two nodes are represented by double concentric circles. Parallelism and decision making are modeled by AND/OR split and join control nodes. AND-logic nodes are denoted by small solid filled circles, and OR-logic nodes are small hollow circles. In this document, we extend the basic ICN definition by attaching probabilities to the arcs emanating from the split nodes. See Fig. 1 for a simple stochastic ICN example diagram. This definition of an admissible basic ICN includes restrictions that an ICN is single entry, single exit, and has properly nested AND/OR logic. Basic ICNs allow duplicate labels on action nodes and properly nested loops. A simple looping mechanism is implemented by a special type of OR split node (loop termination) that has one of its two outgoing arcs pointing back to another special OR join node (loop initiation) that has two incoming arcs and one outgoing arc. This is illustrated in Fig. 1 by the arc labeled 0.10 that allows arbitrarily many repetitions of the (catalog (G), review (D)) sequence to occur.

3. Related work

Our work is concerned with Information Control Nets, with probabilistic organizational models and languages, with model-log comparison, and with generation and approximation of infinite languages. There is a large literature [40] covering these topics. We summarize and describe the relevance and similarity of some of that literature here.

Probabilistic languages. There is a rich literature concerned with stochastic grammars and the *p-Languages* that they recognize within computer science [37,38], mathematics [21], and linguistics [25]. Our work has an intersection with work on Markov models and Bayesian networks. Also there is highly related work on probabilistic automata which are a generalization of Markov chains. The literature of probabilistic automata mathematically discusses many techniques and findings about, e.g. which *p-Languages* are generated by which types of automata [18]. This literature elucidates equivalences between stochastic grammars and probabilistic automata. It is interesting that there are non-obvious cases in the literature showing normalized stochastic grammars that generate *p-Languages* that are not normalized [21]. Like ICNs, these representations are good for capturing sequence and OR behavior, but are different from the ICN model because they do not have any explicit notation to represent the parallelism of AND forks and joins.

Enterprise analysis models. ICNs, as defined in [14], are an organizational information systems analysis metamodel. There are numerous interesting, thoughtfully conceived organizational analysis metamodels and frameworks [7,8]. Multiple published articles compare various organizational models [5,13] and metamodels [24,39]; we do not repeat this comparison here. Some of these organizational models are partially probabilistic in nature [22]. These models are used in domains such as organizational efficiency analysis. Our probabilistic work concerned with s-ICNs can be viewed as different, but complementary to these other works.

Probabilistic workflow models. Within the area of organizational analysis, numerous workflow models, especially ones used in simulations [22], have a feature that allows the placement of probabilities, or probability distributions on the outgoing arcs of OR nodes. For example the IBM system, FlowMark, allowed the simulator to draw a model, and place probabilities on the exclusive OR nodes. Also the ProM system at TU/e has plug-ins for many types of analyses. None of these systems have adopted our semantic to allow meaningful placement of probabilities on AND-fork nodes to extend to probabilistic languages.

Petri net models. There are a number of AND/OR type graph models that have been presented in the workflow modeling literature [31]. One popular workflow model is the Petri net [23]. Petri nets were defined as an abstraction such that execution of transitions is instantaneous, and token time on places is finite, but unspecified. There is an interesting literature concerned with stochastic Petri nets [4,26]. These nets allow probability distributions specifying amount of time to be associated with transitions and/or places. This is different from the s-ICN which associates probabilities with the branches of OR-forks and AND-forks; our model is honed for organizational analysis and comparisons, and always generates normalized *p-Languages*. The work of Varacca and Nielsen on Probabilistic Petri nets [34] is notably different from most of the stochastic Petri net literature; it is concerned with keeping track of probabilistic behavior, but does not employ timed nets. V + N define the notion of probabilistic word as a probability distribution over all strings of the same length, and then define a probabilistic language as a set of probabilistic words. Their model is concocted for different application purposes, and does not attack the important issue of normalization of probabilistic languages – this is more complex within their model.

Model-log comparison. In the area of process mining, there have been publications that are concerned with the comparison of workflow models to event logs [2,9,12,35], and comparison of models to other models [3,28]. There have also been definitions of complexity, conformance, fitness, consistency, and completeness of models [14,23]. The work on conformance specifically compares a model to a log, and asks how well they conform [2,29,30,38]. Their concepts of overfitting and underfitting are directly related to our concepts of completeness and correctness. Our model and algorithm are useful for the calculation of various probabilistic conformance metrics and to calculate fidelity. We note that none of these previous definitions have used probabilities on branches of OR-forks and AND-forks in the way that this document proposes to help quantify their analyses.

4. Stochastic Information Control Nets

4.1. s-ICN conceptual initiation

As mentioned above, there are a number of AND/OR graph models in the process modeling literature [31]. If AND/OR models are enhanced to be probabilistic, then even more types of analyses can be performed. For example, a model may show decision making action. Adding probability of selection to each possible decision choice creates an enhanced model that can be used to analyze congestion.

We create stochastic ICNs (abbreviated s-ICNs) by extending the classic ICN definitions by adding probabilities to arcs. Each arc in the ICN graph has a probability associated with it. Given any node, the sum of the probabilities of arcs emanating from it must be one. Thus, for example, since action nodes are restricted to having only one outgoing arc (out-degree = 1), the probability on the arc out of any action must be one. For convenience, arcs having a probability of one associated with them are unlabeled in our diagrams.

Fig. 2 shows the primitives that can be utilized to construct an s-ICN. Note that OR-fork and AND-fork have probabilities p_1 through p_n attached to outgoing arcs (with $\sum_{i=1}^{n} p_i = 1$ as their normalization criterion), but OR-join and AND-join do not have probabilities on their incoming arcs because the probabilities of arcs are strictly determined by their from-nodes, not their to-nodes. The outgoing arcs of OR-join and AND-join nodes have probabilities one associated with them. Since these graphs must be properly nested, each fork node can have any finite number of out-arcs, but it must be matched by the same number of in-arcs on the corresponding join node. The full formal definition of the s-ICN is presented in the next section.

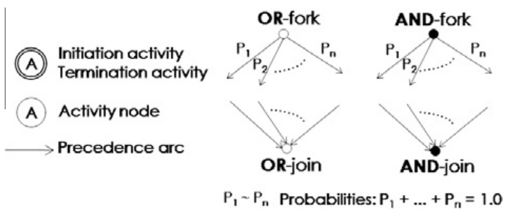

Fig. 2. s-ICN primitives.

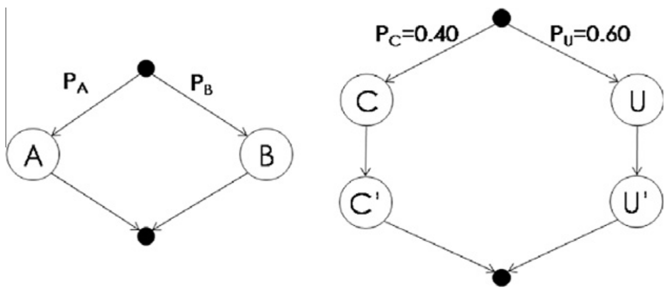

Fig. 3. Simple s-ICNs with AND-fork nodes.

The University of Colorado (CU) library, like many libraries, has been investigating automation alternatives. In 2008/2009, a process modeling and analysis study of the CU library was performed by students and faculty within the Collaboration Technology Research Group at CU [32] to assist in this investigation. Stochastic ICNs were used and proved to be highly useful. Fig. 1 shows an example s-ICN illustrating a highly simplified process (book acquisitions) within the library.

The process begins with a request for the purchase of a book by a Professor or Department (action circle labeled A). Notice that after this, parallel action (AND split) occurs because the cost analysis task (labeled C and C′ for its beginning action and its end action respectively) can proceed in parallel with the utility analysis task (labeled U (begin) and U′ (end)). The AND join shows that acquisition inquisition, which is a decision making action to decide to purchase or not, can only happen after both action U′ and action C′ are finished. After action I, either the book is purchased (P) or a letter of rejection is sent to the requestor (R). In either case, the action labeled Z is the final action of the process. Also note the back loop arc from after D to before G. It means that the actions G followed by D can be repeatedly executed. This loop is possibly executed a large number of times, but the probability of this is quite small since the back arc is associated with probability 0.10.

Semantically, the OR split node represents an *exclusive or* function, so the probability attached to an OR arc represents the probability of selecting that arc as the execution path (and thus NOT selecting any other competing arc). Semantically, the probability attached to an AND arc represents the probability of selecting the action at the tail of that arc for execution first (followed by execution of actions on other arcs and this arc in arbitrary order). As a simplifying abbreviation, arcs out of a fork node are allowed to be unlabeled, which means that the alternatives in that case are equiprobable. In all unspecified cases, alternatives are assumed to be equiprobable.

The attachment of probabilities to AND arcs is less straight forward than OR, has a novel semantic here, and has never been published in the workflow modeling literature. Consider the simple s-ICN fragment shown in Fig. 3 which depicts the concurrent execution of two actions, labeled A and B. The model shows that either action A or action B can begin first, but both must occur. Since ICNs are defined such that no two actions are executed at exactly the same time (like Petri net transitions), there are only two possible strings (execution sequences), AB and BA. Note that this definition does not severely restrict modeling possibilities; if a long running action (e.g. utility analysis), needs to be modeled, it can be done via two instantaneous actions labeled initiate utility analysis followed by terminate utility analysis. This is exactly what appears in our model of Fig. 1. The probabilities on the two arcs emanating from an AND node each denote the likelihood that arc is selected first. Since one or the other must be selected first, the probabilities on the outgoing edges of an AND fork node must logically also sum to one ($p_A + p_B = 1$ in Fig. 3).

A long running task is frequently modeled with ICNs as two actions – a start of task action, and an end of task action. As another example, Fig. 3 shows an ICN of the AND combination of the two sequences CC′ and UU′. There are six possible execution sequences: three starting with C and three starting with U. The total probability of the first three sequences must add to 0.40 and be equiprobable; thus the probability of any one of the three strings is $0.40 \times \frac{1}{3} = 0.133+$. Similar logic holds for the other branch. Thus we can calculate $0.60 \times \frac{1}{3} = 0.20$ as the probability attached to the string UCC′U′. For the general AND case, we are interested in counting the number of admissible interleavings and their probabilities. For example we do not want to count the permutations such as C′UCU′ because it is not admissible for C′ to precede C according to Fig. 4. We later present a recursive function to compute the number of admissible interleavings in the general case. Note that this definition also does not severely restrict modeling possibilities; if we are concerned with attaching disparate probabilities to

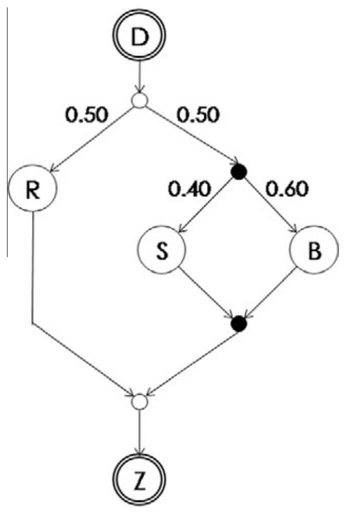

Fig. 4. Library decision making process.

various strings, then we can always draw a more complex model to do this. In general, there is frequently a tradeoff between complexity and fidelity that should be carefully evaluated [23].

4.2. s-ICN formal definition

Stochastic Information Control Nets can be formally defined as follows using the standard terminology of graph theory and probability theory. An s-ICN is a directed graph where every node (or vertice) has a unique ID number, a (not necessarily unique) label, and a type; every arc (or edge) is defined as an ordered pair $<n_1,n_2>$ of nodes, and has a probability (in the $(0,1]$ range) associated with it denoted $pr < n_1, n_2>$. Directed arcs connect nodes, and semantically denote precedence. There are seven node types defined: activity, OR-fork, OR-join, AND-fork, AND-join, LOOP-fork, and LOOP-join.

- Each activity node, n_a, has in-degree, d^{in}, of one, and out-degree, d^{out}, of one. The exception is the one unique initial node, n_I such that $d^{in}(n_I) = 0$; and the one unique final node, n_F, such that $d^{out}(n_F) = 0$. Semantically, activity nodes denote activities, events or actions; and have at most a single predecessor node, n_{pred}, and at most a single successor node, n_{next}. Stochastically, $pr<n_a,n_{next}> = 1.0$.
- Each OR-fork node, n_{OF}, has in-degree, $d^{in}(n_{OF}) = 1$, and out-degree, $d^{out}(n_{OF}) = m$ where m is an integer, $m > 1$. Semantically, OR-forks denote choice or conditional branch control nodes. They have a single predecessor node, and more than one successor nodes, n_1, $n_2,\ldots,n_m$, of which only one can be chosen to execute next during any execution sequence (exclusive OR semantics.) Stochastically, $pr<n_{OF},n_{next}> > 0.0$ for all out-arcs, and the node is normalized if $\sum_1^m pr < n_{OF}, n_{next} >= 1.0$.
- Each OR-join node, n_{OJ}, has out-degree, $d^{out}(n_{OJ}) = 1$, and in-degree, $d^{in}(n_{OJ}) = m$ where m is an integer, $m > 1$. Semantically, OR-joins denote control flow alternatives coming together. They have a single successor node, and more than one predecessor nodes, $n_1,n_2,\ldots,n_m$. The completion of any one predecessor can trigger the execution of n_{OJ} during any execution sequence (exclusive OR semantics.) Stochastically, $pr<n_{OJ},n_{next}> = 1.0$.
- Each AND-fork node, n_{AF}, has in-degree, $d^{in}(n_{AF}) = 1$, and out-degree, $d^{out}(n_{AF}) = m$ where m is an integer, $m > 1$. Semantically, AND-forks denote concurrent execution of all successor nodes. They have a single predecessor node, and more than one successor nodes, $n_1,n_2,\ldots,n_m$, of which only one can be chosen to begin execution first during any execution sequence (AND semantics.) The probabilities associated with the out-arcs denote the probability of each individual successor being the first to begin execution. Stochastically, $pr<n_{AF},n_{next}> > 0.0$ for all out-arcs, and the node is normalized if $\sum_1^m pr < n_{AF}, n_{next} >= 1.0$.
- Each AND-join node, n_{AJ}, has out-degree, $d^{out}(n_{AJ}) = 1$, and in-degree, $d^{in}(n_{AJ}) = m$ where m is an integer, $m > 1$. Semantically, AND-joins denote control flow alternatives synchronizing together. They have a single successor node, and more than one predecessor nodes, $n_1,n_2,\ldots,n_m$. The completion of all predecessors triggers the execution of n_{AJ} during any execution sequence (AND semantics.) Stochastically, $pr<n_{AJ},n_{next}> = 1.0$.
- Each LOOP-fork node, n_{LF}, has in-degree, $d^{in}(n_{LF}) = 1$, and out-degree, $d^{out}(n_{LF}) = 2$. Semantically, LOOP-forks denote the final step of a repeated block of nodes. They represent choice or conditional branch control nodes in which there is a choice between cycling back to repeat again the block via transferring to the corresponding LOOP-join node (with probability $pr<n_{LF},n_{LJ}>$) or exiting the loop and continuing with the next node (with probability $pr<n_{LF},n_{next}>$). LOOP-fork nodes have a single predecessor node, and exactly two successor nodes. The probability $pr<n_{LF},n_{LJ}>$ must be within the range $(0,1]$. Stochastically, $pr<n_{LF},n_{LJ}> + pr<n_{LF},n_{next}> = 1.0$.

- Each LOOP-join node, n_{LJ}, has out-degree, $d^{out}(n_{LJ}) = 1$, and in-degree, $d^{in}(n_{LJ}) = 2$. Semantically, LOOP-joins, which occur before their corresponding LOOP-forks denote the beginning of a repeated block of nodes. They have a single successor node, and exactly two predecessor nodes. The completion of any one predecessor can trigger the execution of n_{LJ} during any execution sequence (exclusive OR semantics.) Stochastically, $pr\langle n_{LJ}, n_{next}\rangle = 1.0$.

If the sum of probabilities on all out-arcs equals one for all nodes of an s-ICN, then the s-ICN is normalized. In this document, when we say ICN, we always imply that it is normalized unless explicitly stated otherwise. All admissible s-ICNs must be properly nested, and non-empty. Informally, properly nested means that all control constructs must be entirely contained within other control constructs, and may not interleave nor have unstructured "GO TO" arcs. This is defined formally in the follows sentences using the concepts of block initiation nodes and block termination nodes. The control node types of OR-split, AND-split, and LOOP-join are categorized as block initiation nodes. The control node types of OR-join, AND-join, and LOOP-split are categorized as block termination nodes. An s-ICN is properly nested iff there is a pairing of control nodes such that each OR-split is paired with an OR-join, each AND-split is paired with an AND-join, and each LOOP-split is paired with a LOOP-join. Furthermore, each block initiation node, n_i, within an s-ICN must be paired with its block termination node, n_t, in such a way that:

(1) $n_i \ll n_t$,
(2) $d^{out}(n_i) + d^{in}(n_i) = d^{out}(n_t) + d^{in}(n_t)$,
(3) $(n_i \ll n_i^*) \Rightarrow (n_t^* \ll n_t)$ where (n_i, n_t) and (n_i^*, n_t^*) are matched pairings,
(4) Every execution sequence (path) that reaches n_i must eventually reach n_t.

(Note that the notation $a \ll b$ means that node a precedes node b in all simple paths to b from the initial node of the graph.)

A block initiation node is non-empty if there is at least one activity node on each path to its matching block termination node. An s-ICN is non-empty if it contains at least one activity node, and all block initiation nodes are non-empty.

4.3. Probabilistic languages and model-log comparison

Information Control Nets can be compared and contrasted with other models which are generators (e.g. phrase structure grammars, Petri nets) or acceptors (e.g. automata) of sets of strings or languages [21,25]. An ICN is a generalization of a phrase structure grammar. A Stochastic Information Control Net generates (or specifies) a set of strings and each string has a probability associated with it. In the context of workflow management system models, a string can be interpreted as an execution sequence. This set of strings with probabilities attached can be viewed as a probabilistic language.

A probabilistic language (*p-Language*) has been defined in the literature as a set of strings, s_i, defined over an alphabet, Λ, in which each string, s_i, has a probability, $p(s_i)$, associated with it [18]. Each element of a *p-Language* is called a Probabilistic String (*p-String*), denoted $[s_i; p(s_i)]$. A *p-Language* is normalized if and only if summation $p(s_i)$ over all strings in the language is one. In our model domain, the alphabet consists of action labels in an ICN process model, and strings represent possible execution sequences of a workflow process depicted by the model. In this document, we use capitol letters to denote action labels, and s or s_i to represent strings. There are interesting results in the probabilistic languages literature that some normalized stochastic grammars generate probabilistic languages that are not normalized [21]; further discussion of this topic is beyond the scope of this document. An important result established in this document is that each and every probabilistic language generated by a normalized s-ICN is normalized.

Workflow management systems generate workflow log files. Workflow models can be compared to workflow log files to discover anomalies, exceptions, task evolution, security leaks, etc. [2,6,11,33]. The workflow models sometimes represent the management view of processes. The log file may represent how the procedural work is actually done. It can be quite enlightening to discover how an organization really works, as compared to management's view of how the organization ought to work. All of the above investigations can be facilitated via model-log comparison. See, for example, the Rozinat work on conformance checking [29,30].

Model-log comparison is usually computed via deterministic comparison of log entries to model execution sequences. This document introduces new definitions, functions, and algorithms to allow probabilistic model-log comparisons via an s-ICN model. There are numerous insights, anomalies, and organizational lessons that can be gleaned from this type of probabilistic analysis. Suppose there exist two complex, concurrent, long-running tasks denoted by action sequences $A_1, A_2, \ldots, A_n$ and $B_1, B_2, \ldots, B_m$. Many relevant questions can be posed as "what is the likelihood that A_i occurs before B_j?" The expected probabilistic behavior of A_i and B_j can be modeled by an s-ICN, and the actual frequency of occurrence can be computed using the log, to allow comparisons. For example, in our library study it was found that in 80% of the cases, utility analysis was initiated before the more complicated cost analysis was initiated. However, in almost all of these cases, cost analysis was completed long before utility analysis. Further investigation of this uncovered an excellent automation opportunity to drastically speed up cost analysis.

Fig. 4 shows a simple s-ICN of another library decision making process. Note that all execution sequences in this example begin with the decision making action (D) and end with action Z. One execution sequence is rejection (R); this sequence

Table 1
Log of customer sequences.

CUSTOMER 1:	DBSZ
CUSTOMER 2:	DRZ
CUSTOMER 3:	DSBZ
CUSTOMER 4:	DRZ

consists of the three actions DRZ. If the decision is positive, then the second arc of the OR node is traversed and there are two possible execution sequences: DSBZ or DBSZ. Associated with any basic ICN model is a language defined as the set of all possible execution sequences. In the example of Fig. 4, there are three possible execution sequences, so the language generated by this model is DRZ, DSBZ, DBSZ.

Let us compare this model to a log. The log shown in Table 1 below shows a history of actions for four customers: The first customer was accepted, and action B happened before action S (DBSZ), the second customer was rejected (DRZ), the third customer was accepted, and action S happened before action B (DSBZ), and the fourth was rejected (DRZ).

From this log and Fig. 4, we can do deterministic model-log comparison for model *correctness* and *completeness*. It is an easy matter to validate that all execution sequences of the model are in the log (correctness of the model), and that all customer entries in the log can be generated as execution sequences of the model (completeness of the model). This is a tiny toy example for exposition purposes. Note that it is not so easy to do this comparison by inspection if the model and the log are realistically large (a log typically has a very huge number of entries, some of which may be inaccurate).

Notice that, in Fig. 4, the OR-split (hollow small circle) has a probability of 0.5 attached to each of its two out-arcs. This means that about 50% of the customer cases go to the left (rejection), and 50% go to the right (shipping and billing). We can utilize the probabilities attached to the OR-fork arcs (0.5 and 0.5) and the AND-fork arcs (0.4 and 0.6) of the example to do probabilistic model-log comparison to compute **fidelity**.

By multiplying probabilities shown in Fig. 4 for each execution sequence in the model, we get the following probabilistic language for the model:

$$\{[\mathrm{DRZ}; 0.5], [\mathrm{DSBZ}; 0.2], [\mathrm{DBSZ}; 0.3]\}$$

It is also possible to derive a *p-Language* from the log by labeling each entry of the log by its percentage of appearances. From the log in Table 1, we can see that DRZ appeared in two out of four entries (50%), and DSBZ and DBSZ each appeared in 25% of the entries (one out of four). Thus the log generates the following probabilistic language:

$$\{[\mathrm{DRZ}; 0.5], [\mathrm{DSBZ}; 0.25], [\mathrm{DBSZ}; 0.25]\}$$

Note that although both probabilistic languages have the same strings, they do not have the same probabilities, so the model does not have as much fidelity as might be hoped. The fidelity of a model with respect to a log can be mathematically computed by calculating the distance between the *p-Language* generated by the model and the *p-Language* derived from the log. There are numerous metrics that can be used as distance functions between probabilistic languages including chi squared, earth mover distance, etc. The closer the distance is to zero, the better is the fidelity. If we change the probabilities on the AND-fork arcs in Fig. 4 to 0.5 and 0.5, then we obtain a complete fidelity model (zero distance).

To perform this comparison, we assume that name equivalencing of actions in the model and in the log has been completed (e.g.: all actions such as "book purchased" have the same label (P) in the log as in the model). Also, creating the *p-Language* corresponding to a log is hard work, but it is a straight-forward normalization process as described above. Also, creating the *p-Language* corresponding to an s-ICN is hard work that can get quite complex if there is a lot of looping and nesting of split/ join nodes within other split/ join nodes. In Section 5, we present the algorithms to generate a *p-Language* from an s-ICN, and prove that the *p-Language* obtained is always normalized.

5. Normalized *p-Language* generation

5.1. s-ICNs without loops

In this section, we present our algorithm to generate a *p-Language* from an s-ICN in the case of non-looping s-ICNs. In this algorithm, when we encounter fork and join nodes, we must combine strings, and attach probabilities to them. In this section we first present the linear notation for s-ICNs. We use $(s_i \cdot s_j)$ and $(f_i \times f_j)$. The dot notation denotes concatenation of strings, and the $\times$ simply denotes multiplication.

Since all s-ICNs are properly nested, we can rewrite the graphical representation in a linear functional notation. Each activity will simply be written as its label. Each control node pair can be written as its type (CAT, OR, AND, or LOOP (where CAT denotes concatenation)). Thus, the sequential execution of n activities, $a_1, a_2, \ldots, a_n$, would be represented by the concatenation function, CAT $(a_1, a_2, \ldots, a_n)$, and the concurrent equiprobable AND execution of shipping and billing would be written as $AND_{[0.5,0.5]}(\mathrm{S}, \mathrm{B})$.

An entire graph is thereby linearized as activities and control functions nested within control functions of these activities. Functions AND, OR, and LOOP are parameterized by their outgoing arc probabilities, and this is specified in the subscript vector that follows the function type. All functions can have an arbitrary but finite number of arguments depending upon their number of arcs, except the LOOP function which is restricted to having one argument. As an example, the graph shown in Fig. 1 is linearized as follows:

$$\mathsf{CAT}(\mathsf{A}, \mathsf{AND}_{[0.20,0.80]}(\mathsf{CAT}(\mathsf{C}, \mathsf{C}'), \mathsf{CAT}(\mathsf{U}, \mathsf{U}')), \mathsf{I}, \mathsf{OR}_{[0.5,0.5]}(\mathsf{CAT}(\mathsf{P}, \mathsf{LOOP}_{[0.10,0.90]}(\mathsf{G}, \mathsf{D}), \mathsf{T}), \mathsf{R}), \mathsf{Z})$$

We define a recursive procedure `plangFinite` that, given an s-ICN without loops (we will abbreviate it to ICN), produces the probabilistic language generated by ICN. We provide a proof that in all cases, the generated *p-Language* is normalized. Note that the procedure is recursive, and therefore the proof is by induction. The input to this procedure is the linear representation of the s-ICN which we process from outermost function to inner functions in standard recursive fashion. The steps, 2 through 4 of the `plang` procedure, use the *p-Language* of their arguments to generate their output *p-Language*.

PROCEDURE `plangFinite`:

(1) if ICN is a single activity A, then `plangFinite`(ICN) = {[A; 1]},
(2) if ICN = $CAT(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then $\texttt{plangFinite}(ICN) = combine\{[s_1 \cdot s_2 \cdots s_{m-1} \cdot s_m; \prod_{i=1}^{m} p(s_i)] | [s_i; p(s_i)] \in \texttt{plangFinite}(ICN_i)\}$,
(3) if ICN $= OR_{[p_1,p_2,\ldots,p_m]}(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then $\texttt{plangFinite}(ICN) = combine\left(\bigcup_{i=1}^{m} \texttt{plangFinite}(ICN_i) \times p_i\right)$, where we define $\texttt{plangFinite}(ICN_i) \times p = \{[s_i; p(s_i) \times p] | [s_i; p(s_i)] \in plangFinite(ICN_i)\}$,
(4) if ICN $= AND_{[p_1,p_2,\ldots,p_m]}(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then $\texttt{plangFinite}(ICN) = combine\left(\bigcup_{i=1}^{m} \{[s_i; \frac{p_i}{\Gamma_i}] | s_i \in I_i\}\right)$ where I_i = {all admissible interleavings starting with the *i*th head symbol} and Γ_i = {the number of strings starting with the *i*th head symbol}.

Since duplicate labels are allowed, the function *combine* is a clean-up function which coalesces *p-Strings* with identical label sequences. If $[s; p_1]$ and $[s; p_2]$ are both generated by a primitive function, then they are combined into one by adding their probabilities to yield $[s; p_1 + p_2]$. For an example, see Fig. 7.

Here is our general formula, used above for calculation of the number of admissible interleavings possible when joining a set of strings. This enables us to calculate the probability of each string. In general, an AND fork can have an arbitrarily large, but finite, number of arcs emanating from it. The tail of each arc can be captured as a set of strings and their probabilities (a *p-Language*). For each choice of a single string from each arc tail, we will calculate the number of admissible interleavings. We will denote the number of arcs as *m*. Also, a string s_i can have an arbitrarily large, but finite, number of symbols (activities) in it; we will denote the number of symbols in string s_i as its length, ln_i. Let the number of admissible interleavings of *m* strings of length $ln_1, ln_2, \ldots, ln_m$ be $\Gamma(ln_1, ln_2, \ldots, ln_m)$. We can calculate the number of strings recursively using the observation that all strings must start with one of the *m* head symbols, and the number of strings starting with the *k*th head symbol is $\Gamma_k = \Gamma(ln_1, ln_2, \ldots, ln_k - 1, \ldots, ln_m)$. Summing these we get;

$$\Gamma(ln_1, ln_2, \ldots, ln_m) = \sum_{i=1}^{m} (\Gamma_i(ln_1, ln_2, \ldots, ln_k - 1, \ldots, ln_m))$$

The base cases for this recursion are:

$$\Gamma(ln_1, 0) = 1; \Gamma(ln_1, 1) = ln_1 + 1$$
$$\Gamma(ln_1, ln_2, \ldots, ln_{m-1}, 0) = \Gamma(ln_1, ln_2, \ldots, ln_{m-1})$$
$$\Gamma(ln_1, ln_2, \ldots, ln_{m-1}, 1) = \left(\sum_{i=1}^{m-1} ln_i + 1\right) \times \Gamma(ln_1, ln_2, \ldots, ln_{m-1})$$

Theorem 1. *The probabilistic language generated by procedure* `plangFinite` *for a normalized s-ICN without loops is always normalized.*

As an aside, note that several proofs in this document are based upon the fundamental rule of probability that the probability of (A or B) equals the probability of A plus the probability of B, given that A and B are independent events. Likewise, the probability of (A and B) equals the probability of A multiplied by the probability of B, given that A and B are independent events.

Proof. If the s-ICN is a single activity A, then `plangFinite`(ICN) = {[A; 1]}. In this case, `plangFinite`(ICN) is normalized because there is only one string in the language, and its probability is one.

- The function CAT maintains normalization because if $ICN_1, ICN_2, \ldots, ICN_{n-1}$, and ICN_n all have normalized *p-Languages*, then `plangFinite` $(CAT(ICN_1, ICN_2, \ldots, ICN_{n-1}, ICN_n))$ is normalized. This is true because as we concatenate strings, we multiply their probabilities. Thus, the total is a product of sums: $\sum_1 \times \sum_2 \times \cdots \times \sum_{n-1} \times \sum_n$. Given that each ICN_i is normalized, this means that each $\sum_i = 1$. Therefore, the product of sums is $1 \times 1 \times \cdots \times 1 \times 1 = 1$.
- The function OR maintains normalization because each subset of elements *k* of the union has sum $\sum p(s_k) = 1$. Thus, $\sum p(s_k) \times p_i = p_i$, and we are given that $p_1 + p_2 + \cdots + p_n = 1$.

- The function AND maintains normalization because we have identified equivalence classes of strings that all have the same probability, p_i/Γ_i. Since there are Γ_i of these strings, they sum to p_i, and we are given that $p_1 + p_2 + \cdots + p_n = 1$.

Finally, note that procedure `plangFinite` always terminates because it has a finite input expression containing a finite number of function invocations.

Since all compound finite ICN *p-Languages* are built from primitives via CAT, OR, and AND functions, and these primitives maintain normalization, all *p-Languages* generated via `plangFinite` from admissible s-ICNs without loops are normalized. □

We end this section with an example of an s-ICN and the derivation of some of the elements of its *p-Language*, see Fig. 5.

Traversal of inner AND and OR components:

1. Traversal of (U AND V) yields two *p-Strings*: $[\mathrm{UV}; \frac{1}{3}]$ and $[\mathrm{VU}; \frac{2}{3}]$
2. Traversal of (A OR B) yields two *p-Strings*: $[\mathrm{A}; \frac{1}{5}]$ and $[\mathrm{B}; \frac{4}{5}]$
3. Traversal of (X AND YW) yields three strings: $[\mathrm{XYW}; \frac{1}{6}]$, $[\mathrm{YXW}; \frac{5}{6} \times \frac{1}{2}]$ and $[\mathrm{YWX}; \frac{5}{6} \times \frac{1}{2}]$

Next, we concatenate these components. The first component is S which is the prefix of all strings of the language; this is always followed by either UV or VU. Note that we must generate a string for each interleaving of A with all admissible permutations of X, Y, and W; there are 12 of these. The same is true for interleavings of B; there are 12 of these. Thus total number of strings in the language is 48. Also we calculate probabilities for each string. For the string SVUYBXWZ, as an example, we get the product probability of $\frac{2}{3} \times \frac{3}{4} \times \frac{4}{5} \times \frac{5}{6} \times \frac{1}{6}$. The first $\frac{2}{3}$ is for SVU; next the $\frac{3}{4}$ is for choosing the AND path before the OR path; the $\frac{4}{5}$ is for choosing B, not A; and the remainder is for choosing Y followed by one of the 6 sub-paths of X and W, interleaving in B. Finally, Z is appended to all strings of the language with probability `1.0`.

5.2. s-ICNs with Loops

In this section, we expand our `plangFinite` algorithm to cover loops. This case requires manipulation of infinite languages.

PROCEDURE `plang`:

This is an expansion of the procedure `plangFinite` to generate *p-Languages* for s-ICNs with loops. We add the fifth case of LOOP constructs to the `plangFinite` procedure.

(1) through (4) are copied from procedure `plangFinite` but we now allow them to accept infinite sets of strings,
(5) if ICN = $LOOP_{[p,1-p]}(ICN_1)$, then

$$\mathtt{plang}(ICN) = combine\left(\bigcup_{k=1}^{\infty}(\mathtt{plang}(CAT(ICN_1)^k) \times (p^{k-1} \times (1-p)))\right)$$

The notation CAT $(ICN)^m$ denotes the concatenation of ICN with itself m times i.e. CAT(ICN,ICN,...,ICN). Also, note that the variable p in (5) above denotes the loop-back probability, $pr\langle n_{LS}, n_{LJ}\rangle$. Note also that this procedure is nonterminating.

Theorem 2. *The probabilistic language generated by procedure* `plang` *from a normalized s-ICN is always normalized.*

Proof. If the s-ICN is a single activity A, then `plang`(ICN) = {[A; 1]}. In this case, `plang`(ICN) is normalized because there is only one string in the language, and its probability is one.

- CAT, OR, and AND functions maintain normalization. If one or more of the ICN_i parameters of CAT, OR, or AND are infinite, then the output language is also infinite. Nevertheless, the output language L is still normalized because the infinite sums, $\sum_{k=1}^{\infty}(p_i^{k-1} \times (1-p_i)) = 1$.
- The $LOOP_{[p,1-p]}(ICN_1)$ function maintains normalization because if the strings of the *p-Language* generated by ICN_1 sum to one, then the infinite sum $\sum_{k=1}^{\infty}(p^{k-1} \times (1-p))$ equals to one since $0 < p < 1$.

Since all s-ICN *p-Languages* are generated from primitives via CAT, OR, AND and LOOP functions, and these primitives maintain normalization, all *p-Languages* derived from admissible s-ICNs via `plang` are normalized. □

We end this subsection with an example of an s-ICN with a loop and the derivation of some of the elements of its *p-Language*. See Fig. 6.

A looping s-ICN is illustrated in Fig. 6; it is a fragment of the ICN for order processing shown in Fig. 1. Its *p-Language* contains the *p-Strings*:

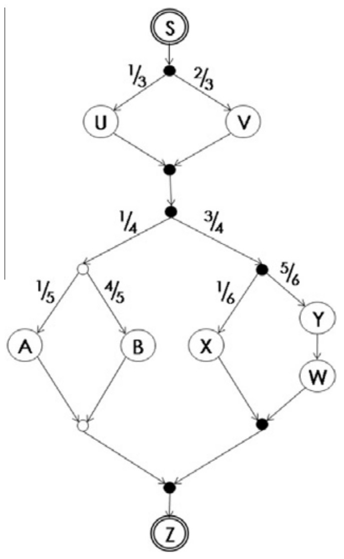

Fig. 5. Example of s-ICN with AND/OR.

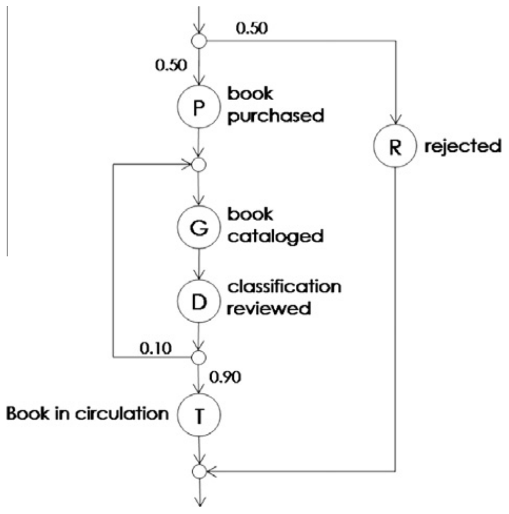

Fig. 6. Book purchase decision fragment.

$$\begin{aligned}
&\{\\
&\quad [\mathrm{R}; \tfrac{1}{2}], \quad [\mathrm{PGDT}; \tfrac{1}{2} \times \tfrac{9}{10}],\\
&\quad [\mathrm{PGDGDT}; \tfrac{1}{2} \times \tfrac{1}{10} \times \tfrac{9}{10}],\\
&\quad [\mathrm{PGDGDGDT}; \tfrac{1}{2} \times \tfrac{1}{10} \times \tfrac{1}{10} \times \tfrac{9}{10}],\\
&\quad \ldots\\
&\}
\end{aligned}$$

Note that the probability of strings gets smaller as the length of strings gets longer. Although there are an infinite number of strings in this language, the sum of all string probabilities still adds to one. The second and third and succeeding strings involve a loop. With probability 0.10, a loop-back occurs. In reality in this example, the frequency of repetition is low, because the probability attached to the loop-back arc is small. This is frequently the case, since loop-backs frequently are exception handling mechanisms.

From the diagrams and examples that we have seen, one might hypothesize that the maximal string derived from any s-ICN could be discovered by looking at the ICN without repeated paths. After all, when comparing a *p-String* $[s; p(s)]$ to a longer *p-String*, $[s'; p(s')]$ that took the same path, but added one or more loop-backs, $p(s) > p(s')$ always. However, this hypothesis is not always true, as shown by the s-ICN in Fig. 7.

An important attribute of ICNs is the allowance for duplicate labels on action nodes. Indeed, concise and understandable modeling is greatly enhanced by this, and our unfold algorithm requires it. Fig. 7 is a simple example of an ICN with duplicate

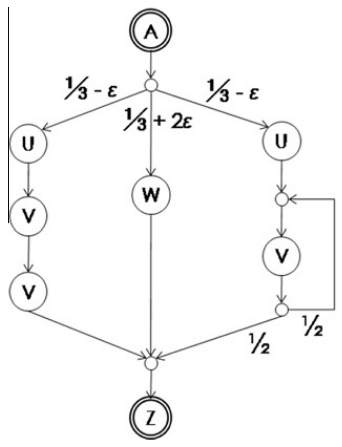

Fig. 7. Duplicate activity labels.

labels. The string AWZ appears to be the maximal string if no loops are retraced. However, by traversing the loop body twice, the string AUVVZ is produced, and when added to a previous string AUVVZ along a different path, the sum may be greater than the probability associated with AWZ.

Notice that the OR-fork node in Fig. 7 has a variable, ε, as part of the probability expression on its three out-arcs. If the value of this variable is chosen to be a small number $(\varepsilon < \frac{1}{39})$, then the string AUVVZ has a higher probability than the string AWZ. We can see this by solving the inequality:

$$pr(\mathsf{AUVVZ}\ along - two - branches) > pr(\mathsf{AWZ})$$

This is equivalent to:

$$pr(\mathsf{AUVVZ}\ along - branch - 1) + pr(\mathsf{AUVVZ}\ along - branch - 3) > pr(\mathsf{AWZ}\ along - branch - 2)$$

This is equivalent to:

$$\left(\frac{1}{3} - \varepsilon\right) + \left(\left(\frac{1}{3} - \varepsilon\right) \times \frac{1}{2} \times \frac{1}{2}\right) > \left(\frac{1}{3} + 2\varepsilon\right)$$

By solving this inequality for ε, we find that the string AUVVZ is maximal whenever $\varepsilon < \frac{1}{39}$.

6. Approximation and manipulation of *p-Languages*

In this section, we make definitions, theorems, and proofs concerning s-ICNs with loops. Note that procedure `plang` may never terminate because the LOOP function requires it generate an infinite *p-Language*. This leads us to approximation considerations. We present an algorithm for generating the maximal string. We also present an algorithm for finding the total probability of an arbitrary given string. Also, we prove that given a *p-Language L*, and given any small number ε, it is always possible to effectively generate an approximation language such that its distance from *L* (i.e. its residual error factor) is less than ε.

As an aside, note that several proofs in this section on approximation languages are based upon the fundamental formulas on sums of geometric series:

For $r \neq 1$, the sum of the first n terms of a geometric series is:

$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} = \sum_{k=0}^{n-1} ar^k = a\frac{1 - r^n}{1 - r}$$

where a is the first term of the series, and r is the common ratio.

As n goes to infinity, the absolute value of r must be less than one for the series to converge. The sum then becomes

$$s = \sum_{k=0}^{\infty} ar^k = \frac{a}{1 - r} = a + ar + ar^2 + ar^3 + ar^4 \cdots$$

6.1. Probabilistic unfolding

Next we define the concept of probabilistic loop unfolding in which each LOOP construct is replaced by a k-branching OR construct. Thus, instead of an ICN allowing infinitely many repetitions of its body, we manipulate an approximation ICN that allows all loop bodies to be iterated one OR two OR$\cdots$OR k times. We show that as k grows large, the finite approximation language gets arbitrarily close to the approximated infinite language.

We define a recursive procedure $unfold_n$ that, given an s-ICN (we will abbreviate it to ICN), produces an approximation s-ICN that executes all loops a finite number, n, of times. The input to this procedure is the linear representation of the s-ICN which we process from outermost function to inner functions in standard recursive fashion.

PROCEDURE $unfold_n$:

(1) if ICN is a single activity A, then $unfold_n$(ICN) = A
(2) if ICN = CAT($ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m$), then $unfold_n$(ICN) = CAT($unfold_n(ICN_1), unfold_n(ICN_2), \ldots, unfold_n(ICN_{m-1}), unfold_n(ICN_m)$)

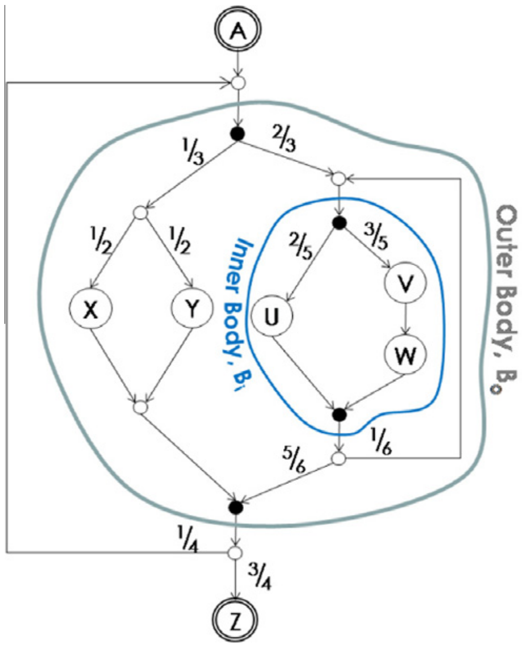

Fig. 8. s-ICN with nested loops, nested ANDs.

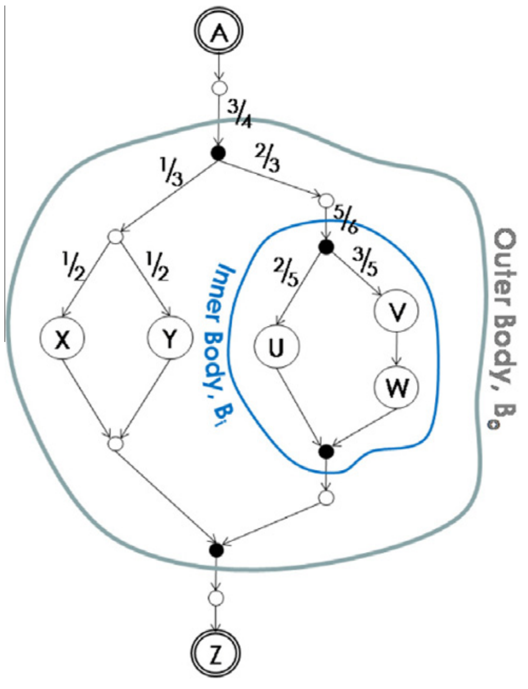

Fig. 9. $Unfold_1$ of s-ICN.

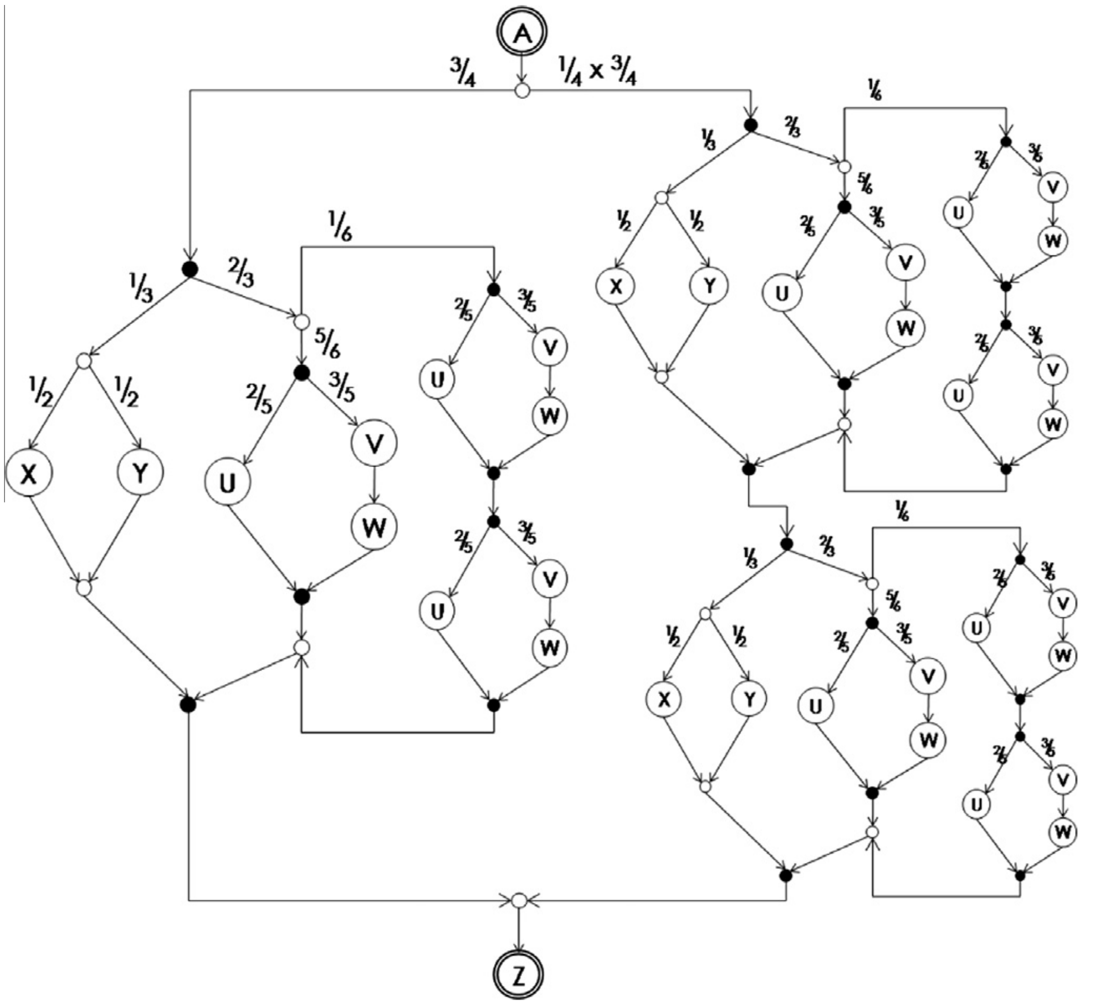

Fig. 10. *Unfold*$_2$ of s-ICN.

(3) if ICN $= OR_{[p_1,p_2,\ldots,p_m]}(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then $unfold_n(\mathrm{ICN}) = OR_{[p_1,p_2,\ldots,p_m]}(unfold_n(ICN_1), unfold_n(ICN_2), \ldots, unfold_n(ICN_{m-1}), unfold_n(ICN_m))$
(4) if ICN $= AND_{[p_1,p_2,\ldots,p_m]}(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then $unfold_n(ICN) = AND_{[p_1,p_2,\ldots,p_m]}(unfold_n(ICN_1), unfold_n(ICN_2), \ldots, unfold_n(ICN_{m-1}), unfold_n(ICN_m))$
(5) if ICN = LOOP(ICN_1), then $unfold_n(ICN) = OR_{[1-p,p(1-p),\ldots,p^{k-2}(1-p),p^{k-1}(1-p)]}(\mathrm{CAT}(ICN_1)^1, \mathrm{CAT}(ICN_1)^2, \ldots, \mathrm{CAT}(ICN_1)^{n-1}, CAT(ICN_1)^n)$

$\underline{L_n}$ is the name we give to the *p-Language* generated by *unfold*$_n$. Note that the generated approximation s-ICNs and their *p-Languages* are unnormalized.

As an unfolding example, we present a more complex s-ICN and some of its unfoldings in Figs. 8–11. The diagram of Fig. 8 is extracted from a previous study of the workflow processes within the University of Colorado library system done in preparation for some automation transitions [32]. The example has nested loops intertwined with nested AND functions. Notice that when there are loops within AND functions, there are infinitely many AND equivalence classes of strings.

Nevertheless, our theorems and algorithms all apply here. The shortest possible string has length 6 (e.g. AXUVWZ). This string also is in the equivalence class with maximal probability. Interestingly, the AND branch to choose V before U has higher probability, but the strings generated by selecting this branch first have lower probability. This is because there are more combinations, so we divide by a larger Γ.

Recall that a k unfolding, a finite approximation to an infinite ICN, is generated by replacing each loop construct by an OR construct with out-degree k. When $k = 1$, we have OR constructs with out-degree one. This is equivalent to simply omitting all back-loops from the original ICN (see Fig. 9). The approximations *unfold*$_2$(*ICN*) and *unfold*$_3$(*ICN*) are presented in Figs. 10 and 11 respectively.

Lemma 3. *For any integer $n > 0$, the set of all p-Strings of* `plang`*(ICN) of length $\leqslant n$ can be effectively generated for any s-ICN.*

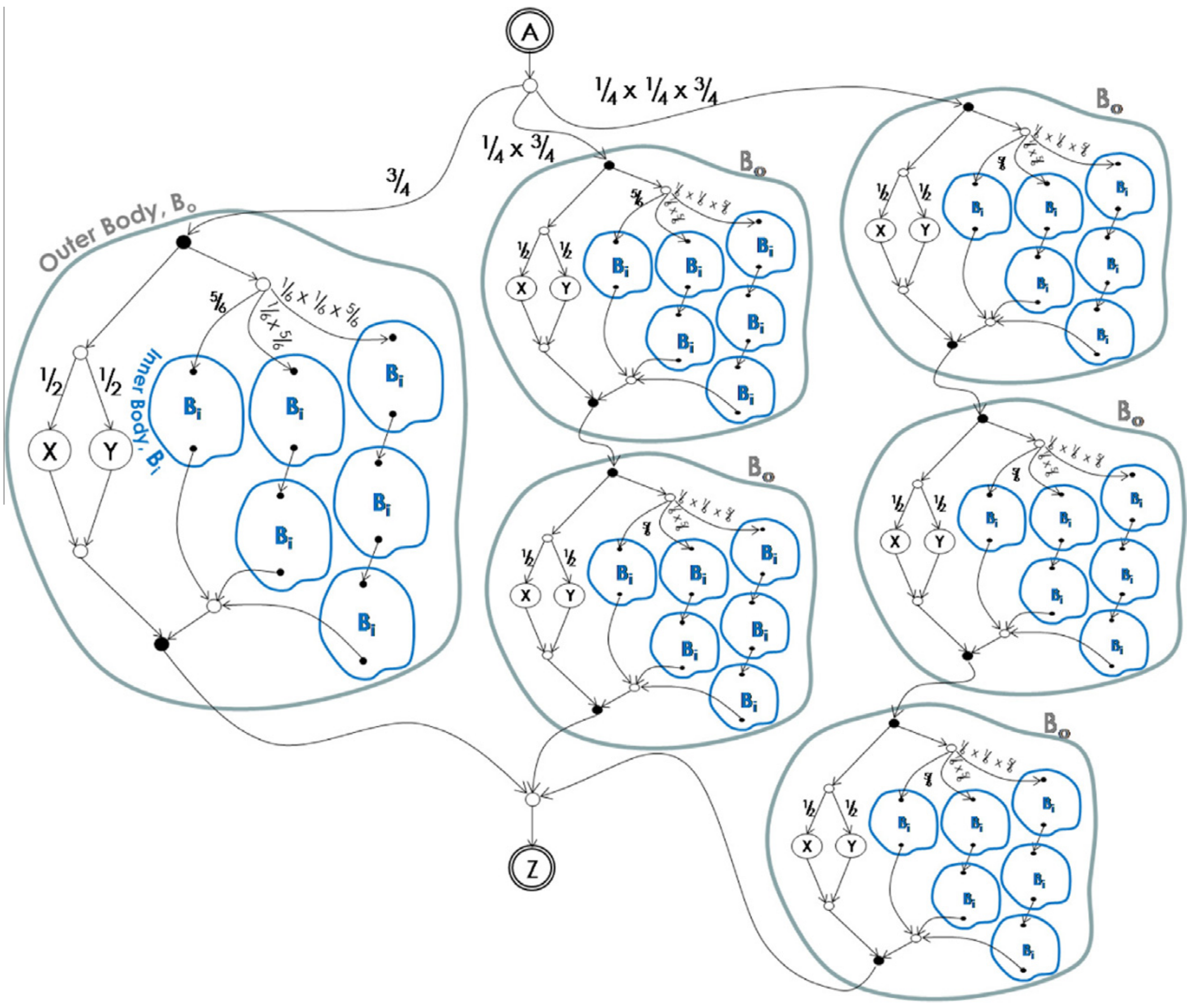

Fig. 11. *Unfold*$_3$ of s-ICN.

Algorithm 3

(1) *generate* all *p-Strings* of *unfold*$_n$(ICN);
(2) *discard* all *p-Strings* $[s; p(s)]$ such that $|s| > n$;
(3) the resultant *p-Language* is all and only the *p-Strings* of `plang`(ICN) with $|s| \leqslant n$;

Proof

(1) By construction, all *p-Strings* remaining have string length less than or equal to n
(2) Since we use the algorithm unfold, all generated *p-Strings* are in `plang`(ICN)
(3) *unfold*$_{n+1}$ has all the execution sequences of *unfold*$_n$ plus new ones using the new OR paths that concatenate loop bodies $n + 1$ times. Since our ICNs are non-empty, these new execution sequences have length at least $n + 1$. Thus there are no new execution sequences of length n or less, so we have effectively generated all *p-Strings* of length $< n$. □

6.2. Approximation languages

Definition 1. Given an infinite *p-Language*, L, generated by a normalized s-ICN with loops, and given a finite approximation language, $\underline{L}$ containing a subset of L, we define $|L - \underline{L}|$ as the sum of the probabilities of all strings that are in L, but not in $\underline{L}$.

This quantity is called the *residual error factor*. If the language $\underline{L}$ is within a very small increment of the infinite *p-Language L*, then $\underline{L}$ is called an ε-equivalent approximation language.

Theorem 4. *Given a normalized looping s-ICN with p-Language L, for any $\varepsilon > 0$, there exists N, such that $(n > N)$ implies using the procedure unfold$_n$, it is possible to effectively generate a p-Language, $\underline{L_n}$, such that $|L - \underline{L_n}| < \varepsilon$*

Algorithm 4

‡ Given a desired combined residual error factor ε for any normalized looping s-ICN with infinite *p-Language*, *L*, this algorithm effectively generates a *p-Language* $\underline{L_n}$(ICN) such that $|L - \underline{L_n}| < \varepsilon$.

(1) if ICN = $LOOP_{[p,1-p]}(ICN_1)$ and the loop is primitive (no loops inside of it), then *N* is directly computable as $N > \left(\frac{\log \varepsilon}{\log p}\right) + 1$.

(2) if ICN = $CAT(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then given ε as the combined error of unfolding, we select $\varepsilon_i = \frac{\varepsilon}{n}$. From each ε_i, we recursively compute N_i, and set $N = max(N_i)$.

(3) if ICN $= OR_{[p_1,p_2,\ldots,p_n]}(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then given ε as the combined error of unfold, we select $\varepsilon_i = \varepsilon$. From each ε_i, we recursively compute N_i, and set $N = max(N_i)$.

(4) if ICN $= AND_{[p_1,p_2,\ldots,p_n]}(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then given ε as the combined error of unfold, we select $\varepsilon_i = \frac{\varepsilon}{n}$. From each ε_i, we recursively compute N_i, and set $N = max(N_i)$.

(5) if ICN = $LOOP_{[p,1-p]}(ICN_1)$ and the loop is non-primitive (loops inside of it), then we select $\varepsilon_1 = \varepsilon$, recursively compute N_1, and set $N = N_1$.

Proof. The residual error factor ε is concerned with the difference in probability between an infinite set *L*, and a finite approximation, $\underline{L_N}$. We can select any small value (>0) for ε. For our computation of *N* from ε, the recursion terminates whenever we encounter a primitive loop (one which has no loops inside of it). In this case, the *p-Language* of the body of the loop is a finite set of *p-Strings*, so ICN_1 has no residual error. *N* is directly computable by solving the inequality $\varepsilon < |L - \underline{L_n}| = \sum_{k=N}^{\infty}(p^{k-1} \times (1-p))$.

(1) if ICN = $LOOP_{[p,1-p]}(ICN_1)$ and the loop is primitive, then the infinite language *L* generated by $LOOP_{[p,1-p]}(ICN_1)$ has total probability of $\sum_{k=1}^{\infty}(p^{k-1} \times (1-p))$; the strings generated by *unfold$_N$* have a total probability of $\sum_{k=1}^{N-1}(p^{k-1} \times (1-p))$. Subtracting these quantities, the residual error factor, $\varepsilon < |L - \underline{L_N}| = \sum_{k=N}^{\infty}(p^{k-1} \times (1-p))$. Since $0 < p < 1$, this is a convergent geometric series with limit of 0 as *N* approaches infinity, we can solve this inequality to get $N > \left(\frac{\log \varepsilon}{\log p}\right) + 1$.

(2) if ICN = $CAT(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, then the residual error of unfold for ICN_i, ε_i, can be selected as $\frac{\varepsilon}{m}$ which is smaller than necessary. Note that it is always OK to choose ε_i too small; it simply results in a closer approximation to *L*. The next step is to recursively compute each N_i from each ε_i. Finally we set *N* equal to the maximum N_i. Note that it is always OK to set *N* too big since the *unfold$_{N+k}$* encompasses all of the *p-Strings* of *unfold$_N$*. In the statement of our theorem, we specifically state $n > N$, rather than $n = N$, for this reason.

(3) if ICN $= OR_{[p_1,p_2,\ldots,p_n]}(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, Note that for the OR function, we take union of subsets of strings, and weighted addition of probabilities. Thus the residual error of unfold for ICN_i, ε_i, can be selected as ε for all $i = 1$ through *m*, so that $\sum(\varepsilon_i \times p_i) = \varepsilon \times \sum p_i = \varepsilon$. The next step is to recursively compute each N_i from each ε_i. Finally we set *N* equal to the maximum N_i because this number of unfoldings is needed for at least one of the component ICNs.

(4) if ICN $= AND_{[p_1,p_2,\ldots,p_n]}(ICN_1, ICN_2, \ldots, ICN_{m-1}, ICN_m)$, Note that for the AND function, we look at the set of all interleavings of the argument ICNs. We divide the resultant set of strings into equivalence classes such that all strings in one class are permutations of each other. The probability of each string is divided by the number of strings in the class, so we can substitute one surrogate for all members of the class. Thus the residual error of unfold for ICN_i, ε_i, can be selected as $\frac{\varepsilon}{m}$ for all $i = 1$ through *m*. As in CAT above, we have simplified and chosen ε_i overly small. The next step is to recursively compute each N_i from each ε_i. Finally we set *N* equal to the maximum N_i because this number of unfoldings is needed for at least one of the component ICNs.

(5) if ICN = $LOOP_{[p,1-p]}(ICN_1)$ and the loop is non-primitive (loops inside of it), then we select $\varepsilon_1 = \varepsilon$, recursively compute N_1, and set $N = N_1$. This is the easy case.

Given (1) through (5), we can, for any given ε, effectively generate a *p-Language*, $\underline{L_n}$, such that $|L - \underline{L_n}| < \varepsilon$. □

Theorem 5. *Given any s-ICN with loops and a positive integer, k, there is an algorithm to generate the k most probable p-Strings of s-ICN.*

Algorithm 5

```
‡ This algorithm generates the ordered set $\underline{MP_N}$. It uses the set of "probable strings" called pSet.
V ← 1; pSet←{}; qSet←{};
while (|pSet| < k) do
   (pSet ← unfold_V(ICN); V ← V + 1);
find total probability of each string in pSet (using Lemma 3);
sort strings of pSet in probability order (from most to least probable);
discard from pSet all strings except k most probable;
‡ The above yields a first approximation ordered p-Language, $\underline{MP_0} = \{[s_1; p(s_1)], [s_2; p(s_2)], \ldots, [s_k; p(s_k)]\}$.
choose $\varepsilon = p(s_k)$, and use Theorem 4 to calculate N, such that $|L - \underline{MP_n}| < \varepsilon$;
qSet ← unfold_N(ICN);
forAll ([s; ps] ∈ qSet) do
   (find total probability, p(s), of s (using Lemma 3); if p(s) > p(s_k) then (pSet ← pSet ∪ {[s; p(s)]}))
sort strings of pSet in probability order (from most to least probable);
discard from pSet all strings except k most probable;
‡ The above yields the desired final ordered p-Language, $\underline{MP_N} = \{[s_1; p(s_1)], [s_2; p(s_2)], \ldots, [s_k; p(s_k)]\}$.
```

Proof

(1) The algorithm terminates because each step terminates. The while loop finds the first k strings of L; it ends because there are more than k strings in L.
(2) The operations of generating, unfolding, sorting, and discarding in the first half of the algorithm yield a first approximation answer where the lowest probability element in the set is $[s_k; p(s_k)]$.
(3) By choosing $\varepsilon = p(s_k)$, we guarantee that the residual error $\sum p_j \cdot (1 - p_j)$ is less than $p(s_k)$. Therefore the probability of any string not in *unfold*$_N$(ICN) is too small: $p(s_{n+i}) < \sum p(s_{n+i}) < p(s_k)$.
(4) It only remains to check if other strings in *unfold*$_N$(ICN) have probability > $p(s_k)$. We do this using the **forAll** statement. We add these strings to pSet, and sort and discard. The remaining ordered set, $\underline{MP_N} = \{[\underline{s_1}; p(\underline{s_1})], [\underline{s_2}; p(\underline{s_2})], \ldots, [\underline{s_k}; p(\underline{s_k})]\}$ contains the k most probable elements in descending order. □

Corollary 6. *Given any s-ICN, there is an algorithm to generate the most likely string of s-ICN and its probability.*

Proof. This follows immediately by applying Algorithm 5 and Theorem 5 with $k = 1$. □

7. Summary and conclusions

This document has aimed to deliver a solid mathematical presentation of a new and novel set of concepts. More motivational and applicative information on knowledge mining and stochastic ICNs are available elsewhere [16]. In this document, we extended the notions of classic Information Control Nets [14,15] to define new concepts of *Stochastic Information Control Nets*. We introduced a simple and useful AND-probability semantic and showed how this probabilistic mathematical model can be used to generate probabilistic languages. The notion of a probabilistic language was introduced as a normalizer for comparisons of organizational information repositories to organizational models. We discussed model-log conformance and presented a definition of fidelity of a model. We also presented new notions of *most probable sequence generation* and *ε-equivalent approximation languages*.

We stress that in this document, we presented a general model/meta-model, and then we specialized our explanation to the process flow perspective within workflow management systems. Within this perspective, it is useful and significant do knowledge mining by quantitatively comparing workflow logs with workflow models. Logs typically record time stamped events, not tasks; our s-ICN model captures instantaneous actions, not long running tasks. Thus there is a good basis here for comparison of events (in the log) with actions (in the model.).

Acknowledgement

The authors thank our colleagues, and our institutions for sponsoring this research. Particularly, appreciation is extended to the University of Colorado Collaboration Technology Research Group students Jasmine Jackson and William Hamlette; and to Professor Deborah Hollis, head of Serials Processing and Special Collections at the University of Colorado library. Additionally, this work was partially supported by the Ministry of Knowledge Economy (MKE) of the Republic of Korea, 10035448: Development of Global Logistics Information Synchronization Technology for QoS.

References

[1] W.P.M. Aalst, M. Song, Mining social networks: uncovering interaction patterns in business processes, Business Process Management (2004) 244–260.
[2] W.P.M. Aalst, Business alignment: using process mining as a tool for delta analysis and conformance testing, Requirements Engineering Journal 10 (2005) 198–211.
[3] W.P.M. Aalst, A.K. Alves de Medeiros, A.J.M.M. Weijters, Process equivalence: comparing two process models based on observed behavior, Lecture Notes in Computer Science (2006).
[4] F. Bause, P.S. Kritzinger, Stochastic Petri Nets, second ed., Vieweg, 2002.
[5] K. Kim, A layered workflow knowledge Grid/P2P architecture and its models for future generation workflow systems, Future Generation Computer Systems 23 (2007) 304–316.
[6] J. Halpin et al, Enterprise, business-process and information systems modeling, Lecture Notes in Business Information Processing 29 (2009) 149–161.
[7] C. Braun, R. Winter, A comprehensive enterprise architecture metamodel and its implementation using a metamodeling platform, in: Proc. Int. Works. Enterprise Modeling and Information Systems Architectures, Bonn, Germany, 2005.
[8] Caetano et al., A Role-based enterprise architecture framework, in: Proc. Int. Conf. SAC'09, 2009.
[9] J. Cardoso, W.M.P. Aalst. Path mining and process mining for workflow management systems, in: J. Wang (Ed.), Encyclopedia of Data Warehousing and Mining III, 2009, pp. 1489–1496.
[10] J.E. Cook, A.L. Wolf, Software process validation: quantitatively measuring the correspondence of a process to a model, ACM Transactions on Software Engineering and Methodology 8 (1999) 147–176.
[11] J. Desel, G. Juhas, R. Lorenz, C. Neumair, Modelling and validation with VipTool, in: W. Aalst, A. Hofstede, M. Weske (Eds.), BPM and Workflow Handbook, 2003.
[12] D. Ferreira, D. Gillblad, Discovering process models from unlabelled event logs, in: Proc. Int. Conf. Business Process Management, BPM'09, Ulm, Germany, 2009.
[13] M. Dumas, W.M.P. Aalst, A.H.M. Hofstede, Process Aware Information Systems, John Wiley and Sons, New Jersey, 2005.
[14] C.A. Ellis, Formal and informal models of office activity, in: Proc. IFIP Int. Comp. Congress, Paris, 1983.
[15] C.A. Ellis, Information control nets: a mathematical model of office information flow, in: Proc. 9th Annual ACM Conf. Simulation, Measurement, and Modeling of Computer Systems, 1979.
[16] C.A. Ellis, Knowledge Mining and Stochastic Information Control Nets, Invited talk, Kyonggi University Colloquium Presentation, 2010.
[17] L. Fischer, The 2007 BPM and Workflow Handbook, Future Strategies, Inc., 2007.
[18] K.S. Fu, Stochastic Languages, Stochastic Automata, and Pattern Recognition, Defense Technical Information Center Report #AD0719801, 1990.
[19] The Gartner Group, Gartner Reveals Five Business Intelligence Predictions for 2009 and Beyond, <http://www.gartner.com/it/page.jsp?id=856>.
[20] K. Kim, S. Paik, C.A. Ellis, An actor-oriented workflow model, in: Proc. Int. Conf. CODAS'99, Wollonggong, Australia, 1999, pp. 150–164.
[21] A. Kornai, Probabilistic grammars and languages, in: Proc. 10th Int. Works. Mathematics of Language, 2007.
[22] M. Laguna, J. Marklund, Business Process Modeling, Simulation, and Design, Pearson Prentice Hall, New Jersey, 2005.
[23] K.B. Lassen, W.M.P. van der Aalst, Complexity metrics for workflow nets, Information and Software Technology 51 (2009) 610–626.
[24] Y. Lei, M.P. Singh, A comparison of workflow metamodels, in: Proc. Int. Works. Behavioral Modeling and Design Transformations, ER'97, Los Angeles, USA, 1997.
[25] M.O. Rabin, Probabilistic automata, Information and Control 6 (1963) 230–245.
[26] C. Ramchandani, Analysis of Asynchronous Concurrent Aystems by Timed Petri Nets, Ph.D. Thesis Dissertation, Massachusetts Institute of Technology, 1974.
[27] A.J. Rembert, Automatic Discovery of Workflow Models, Ph.D. Thesis Dissertation, University of Colorado at Boulder, 2008.
[28] A. Rembert, C.A. Ellis, K. Kim, J. Wainer, Beyond workflow mining'', in: Proc. Int. Conf. Business Process Management, BPM'06, Vienna, Austria, 2006.
[29] A. Rozinat, W.M.P. Aalst, Conformance checking of processes based on monitoring real behavior, Information Systems 33 (2008).
[30] A. Rozinat, I.S.M. de Jong, C.W. Gunther, W.M.P. Aalst, Conformance analysis of ASML's test process, in: Proc. 2nd Int. Works. Governance, Risk and Compliance, GRCIS'09, CEUR-WS.org, 2009, pp. 1–15.
[31] S. Perumal, A. Mahanti, Applying graph search techniques for workflow verification, in: Proc. 40th Hawaii Int. Conf. System Sciences, 2007.
[32] University of Colorado Library Renovation. <http://www.ucblibraries.colorado.edu/learningcommons/index.htm>.
[33] I. Vanderfeesten, J. Cardoso, J. Mendling, H. Reijers, W.P.M. Aalst, Quality Metrics for Business Process Models, BPM and Workflow Handbook, 2007.
[34] D. Varacca, M. Nielsen, Probabilistic petri nets and Mazurkiewicz equivalence, in: Proc. Int. Conf. Decision and Control 5, 2001, pp. 4104–4109.
[35] L. Wen, J. Wang, W.M.P. Aalst, B. Huang, J. Sun, A novel approach for process mining based on event types, Journal of Intelligent Information Systems 32 (2009) 163–190.
[36] The Forrester Group, Cultural Resistance Main Cause of BPM Failure, <http://www.computerweekly.com/Articles/2005/03/14/208863/Cultural-resistance-main-cause-of-BPM-project-failure.htm>, 2005.
[37] E.M. Ortega, Stochastic comparisons for rooted butterfly networks and tree networks, with random environments, Information Sciences 181 (2011) 2247–2259.
[38] J. Oh, N. Cho, H. Kim, Y. Min, S. Kang, Dynamic execution planning for reliable collaborative business processes, Information Sciences 181 (2011) 351–361.
[39] V.T. Nunes, F.M. Santoro, M.R.S. Borges, A context-based model for knowledge management embodied in work processes, Information Sciences 179 (2009) 2538–2554.
[40] J. Yong, W. Shen, Y. Yang, Special issue on computer-supported cooperative work: techniques and applications, Information Sciences 179 (2009) 2513–2514.