Using the Internal Logic of a Topos to Model Search Spaces for Problems

FERNANDO NÁUFEL DO AMARAL, *Dept. of Science and Technology, PURO, UFF, Rio das Ostras, RJ, Brazil*
fnaufel@ic.uff.br

EDWARD HERMANN HAEUSLER, *Dept. of Informatics, PUC-Rio, Rio de Janeiro, RJ, Brazil*
hermann@inf.puc-rio.br

Abstract

We present a structural model for (meta)heuristic search strategies for solving computational problems. The model is defined through the use of topos-theoretical tools and techniques, which provide an appropriate internal logic (with the language of local set theory) where objects of interest can be represented.

Keywords: topos theory, internal logic, local set theory, search, heuristics, metaheuristics

1 Introduction

One of the most interesting developments of topos theory [7, 8] from a logical point of view has been the investigation of the internal logics of topoi by means of local set theory (henceforth LST) [1]. This is accomplished by viewing any topos as a model of a theory in the language of LST, which is basically a higher-order language or type theory. The interpretation of such a theory in the chosen topos allows one to conveniently treat the objects of the topos as "sets" and the morphisms of the topos as "functions".

Higher-order logic and type theory have been very important in foundational studies in computer science. Recent work has suggested practical applications of topos theory to computing (see, e.g., [12] for a study involving databases). Our contribution focuses on an application of topos theory and LST to the field of (meta)heuristic search strategies [6], which attempt to solve a given computational problem by making use of "rules of thumb", approximations, guesses and stochastic processes, sometimes inspired by physical (e.g. simulated annealing) and biological (e.g. genetic algorithms) phenomena. Obviously, (meta)heuristic strategies cannot guarantee correct or optimal solutions all of the time; their usefulness comes from the fact that they are often able to produce good quality solutions when applied to intractable problems (i.e., those which cannot be solved by exact algorithms in acceptable time).

The main goal of the research reported in this paper is the development of a logic-based formal model for (meta)heuristics general enough to encompass the many (sometimes informal) definitions found in the literature. Among such definitions, we may quote, for example: "Heuristics are criteria, methods, or principles for deciding

which among several alternative courses of action promises to be the most effective in order to achieve some goal" [11]. According to [16], a metaheuristic is "an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions."

In a more abstract, unified view, both heuristics and metaheuristics are techniques for solving a given problem by first defining some kind of "space" inhabited by candidate answers (which are related by some structure imposed by the definition of such a space) and then defining a strategy for moving in the defined space in search of an appropriate answer. These two aspects — the search space and the search strategy — are the basis for our formal definition of (meta)heuristics.

A general enough formal model for problems and (meta)heuristics can be of great use in the process of choosing, comparing and combining different strategies for solving computational problems of any kind. A *logic-based* model, such as the one presented here, could allow us to represent (meta)heuristics at any desired level of abstraction. The model could provide means for transforming higher-level descriptions of (meta)heuristics into more concrete, detailed representations, much in the same way a software framework is refined and instantiated to generate executable code. Moreover, our logic-based model could allow us to develop a deductive calculus to reason, again at any desired level of abstraction, about (meta)heuristics and their characteristics, as well as the relationships among them.

In this attempt to reconcile abstraction and concreteness, the tools and techniques of category theory show their usefulness. More precisely, by using topos theory [7, 8], we will define a universe — a topos — that comes equipped with a logical language and theory of its own. It is in this logic, by using local set theory (LST), that our logic-based formal model for (meta)heuristics will be developed. Topoi are, in a strong sense, the formal counterparts of sheaves, a geometrical concept that also arises in logical form. Thus, in a single theoretical approach we are able to put together two of the main aspects of (meta)heuristics: the geometry of search spaces and the logic of search strategies. Implementing a (meta)heuristic will then be a matter of coding search spaces as functors (data structures) and search strategies as natural transformations (algorithms).

The main advantage of using LST to construct a formal model of (meta)heuristics is the fact that, once we define an appropriate topos relating problems and search spaces, we will immediately have at our disposal a logical theory whose model is the defined topos. This eliminates questions of expressiveness, consistency and completeness which would fall upon us if we were to construct our logical theory otherwise. In other words, one of the greatest benefits of LST is that the model (i.e., the defined topos) comes equipped with a logical language and theory of its own.

2 Defining the topos

We assume the reader is familiar with basic categorical and topos-theoretical notions such as objects, morphisms, (contravariant) functors, natural transformations etc. Comprehensive references on the subject are [7, 8].

2.1 Problems and reductions

We define a category with computational problems as objects and problem reductions as morphisms. We base our definitions on the general theory of problems presented in [14].

DEFINITION 2.1 (Problems)
A problem is a triple $P = \langle D, R, p\rangle$, with D and R countable, nonempty sets, and $p \subseteq D \times R$ a relation. Elements $d \in D$ are called *data* or *instances*; elements $r \in R$ are called *results* or *answers*; the relation p is called the *problem condition*; $(d, r) \in p$ means that r is a correct answer for instance d.

Problems are related to each other through the notion of *reduction*:

DEFINITION 2.2 (Reductions)
Given $P = \langle D, R, p\rangle$ and $P' = \langle D', R', p'\rangle$, a reduction $P \xrightarrow{(\tau,\sigma)} P'$ consists of a pair (τ, σ) of functions computable in polynomial time, with $\tau : D \to D'$ and $\sigma : R' \to R$ such that correct answers are preserved; more precisely, for every $d \in D$ and every $r' \in R'$, we have that $(\tau(d), r') \in p' \;\Rightarrow\; (d, \sigma(r')) \in p$.

There may be many reductions from a problem P to a problem P'. In this article, however, we assume a set of problems has been defined such that between any two problems in the set there is at most one reduction.[1] It can be easily checked that problems and reductions form a category. This corresponds to the following definition:

DEFINITION 2.3 (The category **Prob** of problems)
Prob is a designated thin, skeletal category[2] having as objects a set of problems and having as morphisms a set of reductions between these problems.

It should be noted that **Prob** is a *small* category (one whose collection of objects is a set). This will turn out to be important in the proof of Theorem 2.8 below.

2.2 Forests and forest assignments

Part of the definition of a (meta)heuristic strategy to solve a given problem is the construction of the search space. In state space search, this means defining states as well as actions corresponding to arcs (or transitions) between the states; in local search, this means defining neighborhoods; in population-based search, this means defining populations and operators to transform them.

We will consider a search space for a problem P to be a forest whose nodes are associated with certain information about P. A convenient category-theoretical definition of forests and forest homomorphisms is the following:

DEFINITION 2.4 (Forests, forest homomorphisms)
A forest S is a functor $S : \omega^{op} \to \mathbf{Set}$, where ω^{op} is the category $0 \leftarrow 1 \leftarrow 2 \leftarrow \cdots$. A homomorphism h from a forest S to a forest S' is a natural transformation $h : S \stackrel{\cdot}{\to} S'$.

To see how a functor $S : \omega^{op} \to \mathbf{Set}$ defines a forest, one views $S(0)$ as the set of root vertices of the forest, $S(1)$ as the set of vertices on the second level of the

[1] This constraint will be justified in Sect. 2.4.

[2] A *thin* category is one where, given any two objects a and b, there is at most one morphism from a to b. A *skeletal* category is one where object isomorphism coincides with object equality.

forest, and so on. The functor S will map the morphism $0 \leftarrow 1$ to the function $parent_1 : S(1) \rightarrow S(0)$, the morphism $1 \leftarrow 2$ to the function $parent_2 : S(2) \rightarrow S(1)$ and so on.

DEFINITION 2.5 (The category **Forest**)
The category **Forest** of forests and forest homomorphisms is the functor category $\mathbf{Set}^{\omega^{op}}$.

We assign forests to problems by means of a contravariant functor from **Prob** to the category **Forest** of forests:

DEFINITION 2.6 (Forest Assignment)
A Forest Assignment (FA) is a functor $F : \mathbf{Prob}^{op} \rightarrow \mathbf{Forest}$.

The fact that F is a contravariant functor is indicated by the superscript "op" in $\mathbf{Prob}^{op}$. This means that a reduction $P \xrightarrow{(\tau,\sigma)} P'$ is mapped to a forest homomorphism $FP' \xrightarrow{F(\tau,\sigma)} FP$, in the opposite direction. This corresponds to the intuition that, as P reduces to P' (and answers to P' can be transformed into answers to P), a traversal of the search space of P' can be transformed into a traversal of the search space of P.

As each FA is a functor, the collection of all FA's can be structured as a functor category:

DEFINITION 2.7 (The category **FA** of Forest Assignments)
FA is the functor category $\mathbf{Forest}^{\mathbf{Prob}^{op}}$.

An object F of **FA** assigns a forest FP to each problem P in **Prob**. The assigned forest FP has unlabeled nodes and edges. Ultimately, we will want to label the nodes with information concerning the answers of P. For now, however, we may appreciate the fact that we already have a topos:

THEOREM 2.8
The category **FA** is a topos.

PROOF. **FA** is the functor category $(\mathbf{Set}^{\omega^{op}})^{\mathbf{Prob}^{op}}$, which, in turn, is isomorphic to the functor category $\mathbf{Set}^{\omega^{op} \times \mathbf{Prob}^{op}}$, a category of the form $\mathbf{Set}^{\mathbf{C}}$, which is a presheaf topos [7, 8] as long as $\mathbf{C}$ is a small category, which is the case here. ∎

2.3 Answer forests and answer forest assignments

Given a problem $P = \langle D, R, p \rangle$ and an FA F, we want the forest FP to be labeled with information concerning the answers of P. More precisely, we want to label each node n of FP with a set of answers $\lambda(n) \subseteq R$. This is justified by the following:

- In population-based metaheuristics, each node in the search space corresponds to a population, i.e., a set of answers;
- In local search metaheuristics and transformation heuristics, each node corresponds to a single answer r, which can be seen as the singleton $\{r\}$;
- In constructive heuristics, each node corresponds to a "partial" answer, and moving from one node to another corresponds to adding "elements" to the partial answer

in order to build a complete answer. One example would be to try to solve the Traveling Salesman Problem (TSP — see [11], e.g.) by picking one initial city and subsequently adding one city at a time, constructing a complete tour in an incremental fashion. In [9], e.g., it is shown that a partial answer of this kind can be seen as (or rather represented by) a set of answers, a point of view which provides an interesting connection between generate-and-test methods of AI and split-and-prune methods of Operations Research. In the example of the TSP, a partial tour t can be represented by the set of all complete tours having t as a prefix.

So we want FAs to work as follows: to each problem $P = \langle D, R, p\rangle$ in **Prob**, the FA F will assign a forest FP whose nodes are labeled by sets of answers in R. Furthermore, given a reduction $P \xrightarrow{(\tau,\sigma)} P'$, the FA F will assign a labeled forest homomorphism $FP' \xrightarrow{F(\tau,\sigma)} FP$ with the added constraint that a node n' of FP' labeled by a set A' of answers must be mapped to a node $F(\tau,\sigma)(n')$ of FP labeled by the set $\sigma(A') = \{\sigma(r') \mid r' \in A'\}$.

This arrangement corresponds to the intuition that finding the answer r' in the search space for P' implies finding the answer $\sigma(r')$ in the search space for P, which reduces to P'. Formally, this can be achieved by defining an *answer forest assignment* (AFA) to be a special kind of functor from $\mathbf{Prob}^{op}$ to the category **LForest** of labeled forests (whose morphisms preserve the node labels as desired). However, a more elegant alternative presents itself: we can label the forests assigned to problems by means of a categorical construction involving a specific object of the category **FA**.

Definition 2.9 (The L functor)
$L : \mathbf{Prob}^{op} \to \mathbf{Forest}$ is the object of **FA** that maps each problem $P = \langle D, R, p\rangle$ to the forest LP with infinitely many levels, whose set of nodes at level 0 is $\mathcal{P}(R)$ (the powerset of R), and whose set of nodes at level i, for $i > 0$, is $\mathcal{P}(R)^{i+1}$, the set of all $(i+1)$-tuples whose components are sets of answers in R. In this forest LP, the $parent_i$ function $(i > 0)$ mapping each node at the ith level to its parent is simply the projection function in the first i components of a $(i+1)$-tuple $\pi_i : \mathcal{P}(R)^{i+1} \to \mathcal{P}(R)^i$. As for morphisms, L maps a reduction $P \xrightarrow{(\tau,\sigma)} P'$ to the forest homomorphism $LP' \xrightarrow{L(\tau,\sigma)} LP$ such that a node $(A'_1, \ldots, A'_i)$ at the i-th level of LP' is mapped to the node $(\sigma(A'_1), \ldots, \sigma(A'_i))$ at the i-th level of LP.

It can be seen that, given a forest S and a problem $P = \langle D, R, p\rangle$, a forest homomorphism $S \xrightarrow{\lambda} LP$ will label the nodes of S with sets of answers in R. A node n at the i-th level of S with $\lambda(n) = (A_1, \ldots, A_i)$ is considered to be labeled by the set A_i. The other components $A_1, \ldots, A_{i-i}$ store the labels of the ancestor nodes of n from the root down to the parent of n.

Furthermore, given an FA F, if the labeling must occur consistently for all search forests assigned by F, then a natural transformation $\lambda : F \stackrel{\cdot}{\to} L$ will do. The main consequence of these considerations is the fact that, if we want our FAs to assign *labeled* forests to problems, it suffices to take pairs $\langle F, \lambda\rangle$, with F an FA and λ a natural transformation from F to L, as shown in Fig. 1.

It turns out that pairs of the form $\langle F, \lambda\rangle$ as above form a category themselves, called the *slice category* $\mathbf{FA} \downarrow L$. What's more, by the Fundamental Theorem of

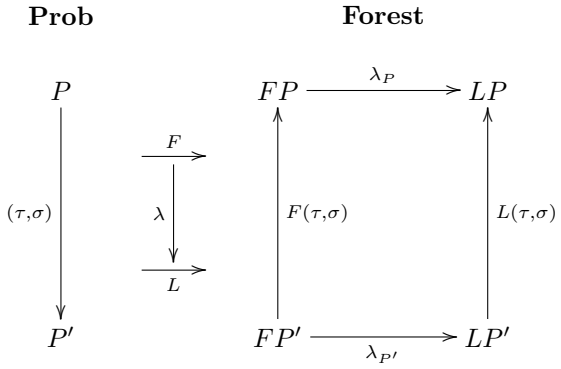

FIG. 1. Labeling search forests via a natural transformation $\lambda : F \stackrel{\cdot}{\rightarrow} L$

Topoi [7, 8], a slice category of any topos is again a topos. So we find ourselves with two topoi at our disposal:

1. The topos **FA**, whose objects are FAs (functors assigning *unlabeled* forests to problems); and
2. The topos **FA** $\downarrow L$, whose objects are pairs of the form $\langle F, \lambda \rangle$, with F an object of **FA** and λ a natural transformation from F to L; i.e., a collection of forest homomorphisms giving a labeling of the forests assigned by F. We call this second, sliced, topos **AFA** for (Answer Forest Assignments).

Both **FA** and **AFA** are topoi with internal logics suitable for specifying (meta)heuristics. For the development of our model, we choose to use the internal logic of **FA** instead of that of **AFA** for the following reasons: (1) every object of the slice topos **AFA** can be represented in the internal logic of **FA** by a term of the form $\langle F, \lambda \rangle$; (2) every morphism of the slice topos **AFA** can be represented in the internal logic of **FA** by a function term of the form $\alpha : \langle F, \lambda \rangle \rightarrow \langle F', \lambda' \rangle$ satisfying some simple constraints; and (3) special objects of **FA** (such as the terminal object and the subobject classifier) are easier to describe and to manipulate than the corresponding objects of **AFA**, because in this latter topos, all the information about the structure and the labeling of the forests is contained in the objects, whereas in **FA** the structure of the forests is given by the objects (the functors F) and the labeling is given by the morphisms (the natural transformations $\lambda : F \stackrel{\cdot}{\rightarrow} L$). In fact, it is generally the case [1, pp. 131ff] that the internal logic of a slice topos $\mathbf{C} \downarrow A$ can be represented in the internal logic of the unsliced topos **C**.

2.4 A remark: the number of reductions between problems

Between two problems P and P' there may exist many reductions (possibly infinitely many). This fact creates a bit of a difficulty when we attempt to assign forest homomorphisms to reductions: as shown in Fig. 2, all four forests FP, FP', LP and LP' are fixed by F and L; yet these four forests must allow for as many homomorphisms as there are reductions from P to P'. Depending on the amount of such reductions,

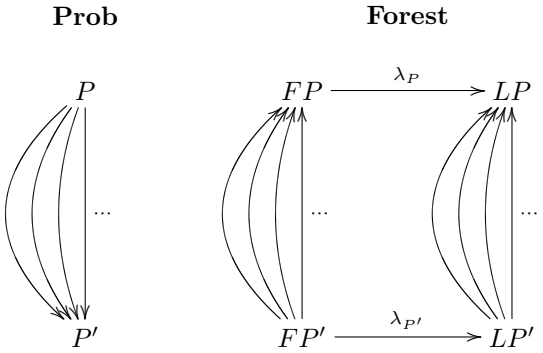

FIG. 2. Several reductions between two problems

it may occur that the forest FP must contain one node labeled by A for each set A of answers to P. This is a rather unrealistic situation: in practice, when defining a search space, one does not want to account for all possible reductions between the problem under consideration and those that are reducible to it. It is in order to avoid this inconvenience that we require that **Prob** be a thin and skeletal category, so that there may exist at most one reduction between any two problems.

3 The internal logic of FA

Our defined topos **FA** is a presheaf topos (i.e., one of the form $\mathbf{Set}^{\mathbf{C}}$). The logical structure of such topoi (e.g., terminal objects, subobject classifiers, etc.) has been known for quite a long time. The calculations yielding the corresponding objects of **FA** are merely special cases of the general calculations that can be found in, e.g., [5] or [7]. In this section, we describe some interesting aspects of the logical structure of **FA** only in order to appreciate the meaning of those general constructions in the context of problems, reductions and search spaces.

3.1 Terminal object

The terminal object of **FA** is the functor $1 : \mathbf{Prob}^{op} \to \mathbf{Forest}$ mapping each problem P to the forest $1P$ consisting of a single linear tree with infinitely many vertices, depicted as $\bullet - \bullet - \bullet - \bullet - \cdots$, with the root on the left. The functor 1 maps each reduction $P \xrightarrow{(\tau,\sigma)} P'$ to the identity homomorphism from $1P$ to itself.

An interesting use of the terminal object is in the definition of *node* in the internal logic of **FA**. To simplify, assume **Prob** consists of a single problem P. Then, given a forest assignment F, a node v of the forest FP is identified with the finite path from the root to the node. In other words, a node v is a *nonempty, finite, linear subobject* of the forest. Nonemptiness corresponds to v being different from the initial object $\varnothing$ (the empty forest); finiteness corresponds to the unique morphism $!_v : v \to 1$ not being epic; linearity corresponds to $!_v$ being monic.

3.2 Subobject classifier

First, some new notation must be developed. Let **Prob** be a category of problems as described in Def. 2.3. Following [7], we define a set S of problems in **Prob** to be *hereditary* if for each problem $P \in S$, all problems in **Prob** reducible to P are also in S. Analogously, a set S of natural numbers is hereditary if for each natural $n \in S$, all naturals n' with $n' \leq n$ are also in S.

We also write $[P)$ for the set $\{P' \mid P' \to P \text{ in } \mathbf{Prob}\}$ of all problems reducible to P, and we write $[P)^+$ for the set $\{S \subseteq [P) \mid S \text{ hereditary }\}$ of all hereditary subsets of $[P)$.

For simplicity, suppose the set of problems in **Prob** is countable.[3] Then, for any P, the set $[P)$ is countable and can be totally ordered, so that any subset of $[P)$ can be represented by a (possibly infinite) tuple $(t_1, t_2, \ldots)$, where each component t_i is either "$-$" or "$+$". The symbol "$-$" as the ith component means that problem P_i is not in the subset in question; the symbol "$+$" as the ith component means that problem P_i is in the subset in question.

An element of $[P)^+$ (i.e., a hereditary subset of $[P)$) can then be represented by a (possibly infinite) tuple satisfying the following condition: for all t_i, t_j, $(P_i \to P_j \text{ in } \mathbf{Prob} \implies (t_j = \text{“}+\text{”} \implies t_i = \text{“}+\text{”}))$.

The object of truth values of **FA** is a functor $\Omega : \mathbf{Prob}^{op} \to \mathbf{Forest}$. As will be seen below, each node at level k of the forest $\Omega(P)$ assigned to P corresponds to a set S of pairs of the form (k', P'), with $k' \leq k$ and $P' \to P$ in **Prob**, such that

$$(k', P') \in S \implies \forall n \leq k' : \forall Q \text{ with } Q \to P' \text{ in } \mathbf{Prob} : (n, Q) \in S$$

i.e., S is hereditary in both components of the pairs.

A set S of this kind (and the node v at level k of $\Omega(P)$ that S represents) will also be denoted by a (possibly infinite) tuple $(t_1, t_2, \ldots)$, but this time each component t_i is either the symbol "$-$" or a natural number k' less than or equal to k. The symbol "$-$" as t_i means that there does not exist in S any pair with P_i as the second component. The value k' as t_i means that S contains all pairs (n, P_i) with $n \leq k'$. The additional constraint that states that the tuple represents a hereditary set in both components is: for all t_i, t_j, $(P_i \to P_j \text{ in } \mathbf{Prob} \implies (t_j \leq t_i))$.

We can now describe the subobject classifier of **FA**. The object of truth values Ω maps each problem P to a forest $\Omega(P)$ having $[P)^+$ as the set of vertices at level 0. I.e., there is a tree in $\Omega(P)$ for each hereditary set of problems reducible to P in **Prob**. Given a node v at level k of the forest $\Omega(P)$, we denote v by a tuple $(t_1, t_2, \ldots)$, with $t_i = \text{“}-\text{”}$ or t_i a natural number less than or equal to k for each i. Children of v in $\Omega(P)$ will be of the form $v' = (t'_1, t'_2, \ldots)$, where for each i, it holds that

$$\begin{cases} t'_i & = t_i & \text{if } t_i < k \text{ or } t_i = \text{“}-\text{”} \\ t'_i & \in \{k, k+1\} & \text{if } t_i = k \end{cases}$$

Given $P \xrightarrow{(\tau,\sigma)} P'$ in **Prob**, we have that $[P) \subseteq [P')$. Ω maps (τ, σ) to the forest homomorphism $\Omega(\tau, \sigma)$ that maps each node $v' = (t'_1, t'_2, \ldots)$ of forest $\Omega(P')$ to the node of $\Omega(P)$ represented by the projection of v' over the components t'_i corresponding

[3] Because D and R are countable in Def. 2.1, the set of problems in **Prob** has at most the cardinality of the continuum.

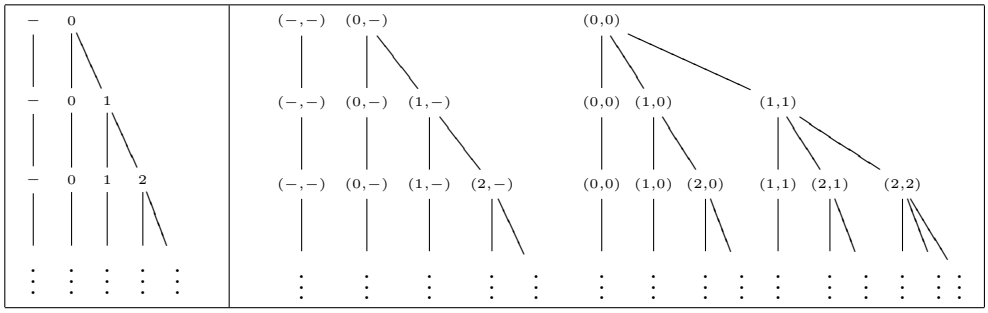

FIG. 3. Fragments of forests $\Omega(P_1)$ (left) and $\Omega(P_2)$ (right).

to problems $P_i \in [P)$. That is, $\Omega(\tau, \sigma)(v') = (t_{i1}, t_{i2}, t_{i3}, \ldots)$ with $\{P_{i1}, P_{i2}, P_{i3}, \ldots\} = [P) \cap [P')$ following the same ordering of $[P')$.

The truth value $\top : 1 \to \Omega$ is the natural transformation such that $\top_P$ is the homomorphism mapping forest $1P$ to the infinite branch $(0, 0, 0, \ldots) - (1, 1, 1, \ldots) - (2, 2, 2, \ldots) - \cdots$ of forest $\Omega(P)$.

The truth value $\bot : 1 \to \Omega$ is the natural transformation such that $\bot_P$ is the homomorphism mapping forest $1P$ to the infinite branch $(-, -, -, \ldots) - (-, -, -, \ldots) - (-, -, -, \ldots) - \cdots$ of forest $\Omega(P)$.

It can be shown that Ω has infinitely many truth values, each one of them corresponding to an infinite branch of each forest $\Omega(P)$.

As a very simple example, suppose **Prob** consists of two problems P_1 and P_2, with $P_1 \to P_2$. The first three levels of the forests $\Omega(P_1)$ and $\Omega(P_2)$ are shown in Fig. 3.

3.3 Natural numbers object

Any presheaf topos has a natural numbers object (NNO). In the case of **FA**, it is the functor $N : \mathbf{Prob}^{op} \to \mathbf{Forest}$ mapping each problem P to the forest consisting of (countably) infinitely many infinite linear trees. More precisely, the set of nodes at each level is the set $\mathbb{N}$ of natural numbers, and the parent function at each level is the identity function on $\mathbb{N}$.

The NNO is useful in providing recursive definitions, in the internal logic of **FA**, of morphisms and objects of interest, as described in [1]. It also allows for the definition of many sets and structures that are found in everyday mathematics. For example, there is an object Q representing the rational numbers; elements of this object will be used below to express discrete probabilities in the definition of stochastic search strategies.

4 Local set theory

Any topos can be seen as a model of some local set theory (LST). In LST, the notion of set is replaced by that of *type*. In the language of LST each term (including those representing sets) has an associated type. The "local" in LST means that some

common set-theoretical operations, such as union, intersection etc., are only defined for terms of the same type (i.e., locally). Apart from that, the language is very similar to that of set theory, with primitive symbols $=$, $\in$ and $\{\,|\,\}$. The language of LST is defined below. In [1, pp. 91ff], it is shown how a local language can be interpreted in an arbitrary topos.

Definition 4.1 (Local language)
A local language $\mathcal{L}$ is determined by the following components:

- *Symbols*: the unit symbol $\mathbf{1}$, the truth-value type symbol $\mathbf{\Omega}$, a collection of ground type symbols $\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$, and a collection of function symbols $\mathbf{f}, \mathbf{g}, \mathbf{h}, \ldots$;
- *Types*: the set of types of $\mathcal{L}$ is the least set $\mathcal{T}$ containing $\mathbf{1}, \mathbf{\Omega}$, all ground type symbols $\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots$ and closed under the following operations:
  - For $\mathbf{A} \in \mathcal{T}$, the power type $\mathbf{PA}$ is also in $\mathcal{T}$;
  - For $\mathbf{A_1}, \ldots, \mathbf{A_n} \in \mathcal{T}$, the product type $\mathbf{A_1} \times \cdots \times \mathbf{A_n}$ is also in $\mathcal{T}$ (for $n = 0$, the product type is $\mathbf{1}$).
- *Signatures*: Each function symbol $\mathbf{f}$ is associated to a signature $\mathbf{A} \to \mathbf{B}$, where $\mathbf{A}$ and $\mathbf{B}$ are types. This is denoted by $\mathbf{f} : \mathbf{A} \to \mathbf{B}$;
- *Variables*: For each type $\mathbf{A}$ there is a countable set of variables $V_{\mathbf{A}}$;
- *Terms*: For each type $\mathbf{A}$, there is a set $T_{\mathbf{A}}$ of terms of type $\mathbf{A}$, defined as follows:
  - $\star \in T_{\mathbf{1}}$;
  - $V_{\mathbf{A}} \subseteq T_{\mathbf{A}}$;
  - For $\mathbf{f} : \mathbf{A} \to \mathbf{B}$ and $\tau \in T_{\mathbf{A}}$, we have that $\mathbf{f}(\tau) \in T_{\mathbf{B}}$;
  - For $\tau_i \in T_{\mathbf{A}_i} (i = 1, \ldots, n)$, we have that $(\tau_1, \ldots, \tau_n) \in T_{\mathbf{A_1} \times \cdots \times \mathbf{A_n}}$. For $n = 0$, this term is $\star$;
  - For $\tau \in T_{\mathbf{A_1} \times \cdots \times \mathbf{A_n}}$, we have that $\pi_i(\tau) \in T_{\mathbf{A}_i}$ (with $i = 1, \ldots, n$);
  - For $\varphi \in T_{\mathbf{\Omega}}$ and $x \in V_{\mathbf{A}}$, we have that $\{x \mid \varphi\} \in T_{\mathbf{PA}}$;
  - For terms σ and τ of type $\mathbf{A}$, we have that $\sigma = \tau$ is a term in $T_{\mathbf{\Omega}}$;
  - For terms σ and τ of types $\mathbf{A}$ and $\mathbf{PA}$, respectively, we have that $\sigma \in \tau$ is a term in $T_{\mathbf{\Omega}}$.

Terms of type $\mathbf{\Omega}$ are called formulae. Free and bound occurrences of variables are defined in the usual fashion. Logical operators are defined as abbreviations, as shown in [1, p. 70]. For example, $\top$ is defined as $\star = \star$; given formulae φ, ψ, we have that $\varphi \wedge \psi$ is defined as $(\varphi, \psi) = (\top, \top)$, and $\varphi \Rightarrow \psi$ is defined as $(\varphi \wedge \psi) = \varphi$. For an example involving a quantifier: given a variable x of the appropriate type, $\forall x : \varphi$ is defined as an abbreviation of $\{x \mid \varphi\} = \{x \mid \top\}$.

Some terms in a local language will represent sets:

Definition 4.2 (Set-terms)
A set-term is any term of power type $\mathbf{PA}$ for some type $\mathbf{A}$.

Set-theoretical definitions are listed in [1, pp. 83ff]. Some of them are:

- $X \subseteq Y$ is defined as $\forall x : (x \in X \Rightarrow x \in Y)$;
- $X \cap Y$ is defined as $\{x \mid x \in X \wedge x \in Y\}$, of the same type as X and Y;
- $X \cup Y$ is defined as $\{x \mid x \in X \vee x \in Y\}$, of the same type as X and Y;
- A is defined as $\{x \mid \top\}$, of type $\mathbf{PA}$, with x a variable of type $\mathbf{A}$. In other words, for every type symbol $\mathbf{A}$, there is a corresponding set-term A;

- $\varnothing_{\mathbf{A}}$, or simply $\varnothing$, is defined as $\{x \mid \bot\}$, of type $\mathbf{PA}$, with x a variable of type $\mathbf{A}$;
- PX is defined as $\{x \mid x \subseteq X\}$, of type $\mathbf{PPX}$, with x a variable of type $\mathbf{PX}$;
- $\{\tau \mid \varphi\}$ is defined as $\{x \mid \exists x_1 : \cdots : \exists x_n : (x = \tau \wedge \varphi)\}$, with x a variable of the same type as the term τ;
- $X \times Y$ is defined as $\{(x, y) \mid x \in X \wedge y \in Y\}$, of type $\mathbf{P}(\mathbf{X} \times \mathbf{Y})$. Note that X and Y may be of different types;
- Y^X is defined as $\{z \mid z \subseteq X \times Y \wedge \forall x \in X : \exists! y \in Y : (x, y) \in z\}$, of type $\mathbf{PP}(\mathbf{X} \times \mathbf{Y})$. [1, pp. 85ff] shows that to each function symbol $\mathbf{f} : \mathbf{A} \to \mathbf{B}$ of $\mathcal{L}$, there corresponds the set-term $\{(a, \mathbf{f}(a)) \mid a \in A\}$, of type B^A. When the language is interpreted in a topos, this means that each morphism $A \xrightarrow{f} B$ is associated to a set-term f of type B^A, allowing us to represent morphisms as functions in classical set theory: as sets of ordered pairs;
- $\coprod_{i \in I} X_i$ is defined as $\{(i, x) \mid i \in I \wedge x \in X_i\}$ of type $\mathbf{P}(\mathbf{B} \times \mathbf{A})$, with I of type $\mathbf{B}$ and with X_i a term of type $\mathbf{PA}$ which may or may not contain free occurrences of variable i.

In [1, pp. 91ff], it is explained in detail how a local language can be interpreted in an arbitrary topos.

5 Stochastic Search Strategies in LST

Given a forest F, we will represent the behavior of a stochastic search strategy by a sequence $(S_i)_{i \in \mathbb{N}}$ where each S_i is a set of triples of the form $\langle T, v, q \rangle$, with T a subforest[4] of F corresponding to the forest formed by all nodes visited up to step i, with v the node last visited, and with q a probability. The sequence $(S_i)_{i \in \mathbb{N}}$ is determined by the set S_0 and by a function *step* from the set of all such sets of triples to itself such that $step(S_i) = S_{i+1}$ for all $i > 0$. The sequence $(S_i)_{i \in \mathbb{N}}$ must be such that the set S_0 contains only triples where T is a single-node subforest of F and where v is that single node. This means that S_0 is actually a collection of possible initial nodes of the search, each node accompanied by a probability. Moreover, for each triple $\langle T', v', q' \rangle$ in S_{i+1} there must exist some triple $\langle T, v, q \rangle$ in S_i such that either $T = T'$, meaning that no new node is visited at step $i+1$, or T' is an extension of T by exactly the one node v'. Finally, obvious conditions on the values of the probabilities must also hold.

In order to describe search strategies in the logic of $\mathbf{FA}$, we must consider pairs of the form $\langle F, \lambda \rangle$ and $\langle T, \theta \rangle$ instead of forests F and subforests T. Furthermore, given $\langle F, \lambda \rangle$ and $\langle F', \lambda' \rangle$, we have that a morphism $\langle F, \lambda \rangle \xrightarrow{\alpha} \langle F', \lambda' \rangle$ can only be considered if α preserves the labels of F. This is expressed by the predicate[5]

$$\mathbf{isLabelPreserving}(\alpha, F, \lambda, F', \lambda') \iff \forall x \in F : \lambda(x) = \lambda'(\alpha(x))$$

Then, $\langle H, \kappa \rangle$ is a subobject of $\langle F, \lambda \rangle$ iff the following predicate is satisfied (where

[4] Here and in what follows, by a "subforest of F" we mean a forest that can be embedded in the forest F by a forest homomorphism. The roots of the subforest must be at the same level as the roots of the forest.

[5] As seen in Sect. 4, the language of LST is typed; here, in order to unclutter the notation, we omit typing information about all terms whose type can be apprehended from the context.

isMonic(α) is satisfied iff α is a monomorphism — see [1]):

$$\begin{array}{l}\mathbf{isSubObject}(H,\kappa,F,\lambda) \iff \\ \quad \exists \alpha \in F^H : (\mathbf{isMonic}(\alpha) \ \wedge \ \mathbf{isLabelPreserving}(\alpha,H,\kappa,F,\lambda))\end{array}$$

From now on, in order to make the formulae more readable, the labeling morphisms (e.g., λ, κ, etc.) and the condition that a morphism α must be label-preserving will be omitted, except where they must be explicitly mentioned. So, labeled forests and subforests will simply be denoted by F, H, T, etc.

The set of all subforests of a given labeled forest F will be denoted PF.

As was mentioned in Sect. 3.1, a node H of a labeled forest F is identified with the finite path from the root to the node. In other words, a node H is a nonempty, finite, linear subobject of the forest. Nonemptiness corresponds to H being different from the initial object $\varnothing$; finiteness corresponds to the unique morphism 1_H from H to 1 not being epic (where 1 is the terminal object of **AFA**: a linear tree with infinitely many nodes); linearity corresponds to 1_H being monic:

$$\begin{array}{l}\mathbf{isNode}(H,\kappa,F,\lambda) \iff \\ \quad \mathbf{isSubObject}(H,\kappa,F,\lambda) \ \wedge \ H \neq \varnothing_F \ \wedge \ \mathbf{isMonic}(1_H) \ \wedge \ \neg\mathbf{isEpic}(1_H)\end{array}$$

Recall that L is the codomain of the morphisms responsible for the labeling of nodes. Then the set of all nodes of a labeled forest $\langle F,\lambda\rangle$ is represented by the following set-term, which we will abbreviate by $nodes(F,\lambda)$:

$$nodes(F,\lambda) = \left\{ \ \langle H,\kappa\rangle \ \in \coprod_{H\in PF} L^H \ \middle| \ \mathbf{isNode}(H,\kappa,F,\lambda) \ \right\}$$

Again, we will omit labeling morphisms and write $nodes(F)$. Clearly, we have that $nodes(F) \subseteq PF$.

A root of a labeled forest F is an element of $nodes(F)$ that is minimal with respect to the partial order "is a subobject of":

$$\begin{array}{l}\mathbf{isRoot}(H,F) \iff \\ \quad H \in nodes(F) \wedge \\ \quad \forall H' \in nodes(F) : (\mathbf{isSubObject}(H',H) \ \Rightarrow \ H' = H)\end{array}$$

The fact that a labeled forest F extends another forest F' by exactly one node is represented by the following predicate:

$$\begin{array}{l}\mathbf{extendsByOne}(F,F') \iff \\ \quad \mathbf{isSubObject}(F',F) \ \wedge \ \exists! H \in nodes(F) : H \notin nodes(F')\end{array}$$

And the following predicate states that the node given by H is the one that was added to F' to yield F:

$$\begin{array}{l}\mathbf{wasAdded}(H,F,F') \iff \\ \quad \mathbf{extendsByOne}(F,F') \ \wedge \ H \in nodes(F) \ \wedge \ H \notin nodes(F')\end{array}$$

In our definition of stochastic search strategies, we want to consider only finite, nonempty sets of triples of the form $\langle T,H,q\rangle$ such that the sum of all probabilities q equals 1. To this end, let C be the set

$$C = \left(\coprod_{T\in PF} nodes(T) \ \times \ \{q \in Q \mid 0 < q \leq 1\} \right)$$

where Q is the object of rational numbers. The elements of C are triples of the form $\langle T, H, q\rangle$ with q a probability. Now define S to be the set

$$S = \left\{ X \in PC \mid X \neq \varnothing,\ X \text{ finite},\ \textstyle\sum_{(x,y,q)\in X} q\ = 1 \right\}$$

The functor $R : \mathbf{Prob}^{op} \to \mathbf{Forest}$ maps each problem $P = \langle D, R, p\rangle$ to a specific forest representing the set R of answers of P. In specifying a search strategy, we must define how answers are to be returned; to this end, define the set A as

$$A = \left\{ X \in P(\ R \times \{q \in Q \mid 0 < q \leq 1\}\ )\ \mid\ X \text{ finite},\ \textstyle\sum_{(x,q)\in X} q\ = 1 \right\}$$

Then a morphism $answer : N \to A$ from the natural number object N determines the answers returned by the strategy if the search terminates at the nth iteration. Note that each answer r is accompanied by a probability value q.

Now we may give a definition of a stochastic search strategy in LST:

Definition 5.1 (Stochastic search strategy)
A stochastic search strategy over a forest F is represented by a term $\langle init, step, answer\rangle$ with *init* a term of type $\mathbf{S}$, *step* a term of type $\mathbf{S^S}$ and *answer* a term of type $\mathbf{A^N}$ satisfying the following predicate:

$$\begin{array}{l}
\mathbf{isStochasticSearchStrategy}(init, step, answer, F) \iff \\
\quad \forall\langle T, H, q\rangle \in init : (T = H\ \wedge\ \mathbf{isRoot}(H, F)) \wedge \\
\quad \forall X, Y \in S : (step(X) = Y\ \Rightarrow \\
\qquad \forall\langle T, H, q\rangle \in Y : \exists\langle T', H', q'\rangle \in X : \\
\qquad\quad (T = T'\ \vee\ (\mathbf{extendsByOne}(T, T')\ \wedge\ \mathbf{wasAdded}(H, T, T'))) \\
\quad)
\end{array}$$

6 Examples

To illustrate the capabilities of the model, we will specify some search strategies in the language of LST. First, however, we define two additional useful terms.

Many heuristics are search strategies where the nodes of the search space are evaluated according to some heuristic function. In our language, a heuristic function to evaluate the nodes of a given labeled forest F can be defined as a term h of type $\mathbf{N}^{nodes(F)}$, where N is the natural number object. In other words, the "grade" a node receives upon evaluation is a natural number. We assume that the lower the grade, the better the evaluation.

For every stochastic search strategy $\langle init, step, answer\rangle$ we may define by simple recursion (see [1]) a term *stage* of type $\mathbf{S^N}$, where N is the natural number object. The idea is that, given any natural number n, the term $stage(n)$ will be equivalent to $step(step(\cdots step(init)\cdots))$, with n applications of *step*.

6.1 Greedy Search

In a simple greedy search strategy, the node visited at each stage after the first is the best child of the node visited in the previous stage, as long as the best child has a better (or equal) evaluation than the node visited in the previous stage. Nothing is assumed about the initial node.

$$
\begin{array}{ll}
 & \textbf{isGreedy}(\mathit{init}, \mathit{step}, \mathit{answer}, F, h) \iff \\
1 & \quad \textbf{isStochasticSearchStrategy}(\mathit{init}, \mathit{step}, \mathit{answer}, F) \wedge \\
2 & \quad \forall T, T', H, H' : (\\
3 & \qquad \mathit{step}(T, H) = (T', H') \iff (\\
4 & \qquad\quad H' = H \ \wedge \ \forall K \in \mathit{children}(H, F) : h(K) > h(H) \\
5 & \qquad\quad \vee \\
6 & \qquad\quad h(H') \leq h(H) \ \wedge \ \forall H'' \in \mathit{children}(H, F) : h(H') \leq h(H'') \\
7 & \qquad) \\
8 & \quad)
\end{array}
$$

FIG. 4. Specification of greedy search in LST

As simple greedy search is deterministic rather than stochastic, for each n the set $\mathit{stage}(n)$ is a singleton. To simplify the formulae, for all n we consider $\mathit{stage}(n)$ to be a pair of the form $\langle T, H \rangle$, with T representing the labeled forest (a tree, actually) formed by the nodes visited so far and H the last node visited.

A search strategy $\langle \mathit{init}, \mathit{step}, \mathit{answer} \rangle$ is considered greedy if it satisfies the predicate in Fig. 4 for some interpretation of h (the heuristic function). There, $\mathit{children}(H)$ is a term denoting the set of children[6] of node H in forest F. Line 4 specifies that the search terminates (i.e., the current node is revisited indefinitely) when the current node H has no better children. Line 6 says that otherwise the next node to be visited is the best child of the current node.

6.2 Simulated annealing

Simulated annealing [6] is a metaheuristic strategy frequently used to solve combinatory optimization problems. At each stage, a child of the current node is chosen at random; the chosen child may be visited with a probability that depends both on the difference between the grade of the current node and the grade of the chosen child (ΔE) and on the value of a parameter T, which, by analogy with a physical process, is called "temperature". At each stage n, the value of T is given by a function $\mathit{sched}(n)$. The idea is to define this function in such a way that it will occasionally happen that a child that is worse than its parent is visited, so as to escape local optima.

A stochastic search strategy $\langle \mathit{init}, \mathit{step}, \mathit{answer} \rangle$ is an example of simulated annealing over a forest F, using heuristic function h and cooling schedule sched, precisely when it satisfies the predicate in Fig. 5.

Line 2 states that sched is a nonincreasing function (sn represents the successor of n); line 5 establishes the termination criterion: when $\mathit{sched}(n)$ becomes 0, the strategy makes no more progress; line 6 describes one iteration, using the terms defined in Fig. 6:[7]

In *pNoAdvance* (a term denoting the probability that no child of the current node is visited) and *pVisit* (a term denoting the probability that a given child of the current node is visited), e represents a rational approximation of the real constant e.

---

[6] A node H' is a child of a node H iff $\textbf{extendsByOne}(H', H)$ is satisfied.

[7] Not all set-theoretical terms and operations used here (e.g. definition by cases) have had their definitions included in this paper. See [3] for details.

$$
\begin{array}{l}
\textbf{isSimAnn}(\mathit{init}, \mathit{step}, \mathit{answer}, F, h, \mathit{sched}) \iff \\
\quad \textbf{isStochasticSearchStrategy}(\mathit{init}, \mathit{step}, \mathit{answer}, F) \wedge \\
\quad \forall n : \mathit{sched}(sn) \leq \mathit{sched}(n) \wedge \\
\quad \forall X, Y, n : (\\
\qquad X = \mathit{stage}(n) \ \wedge \ Y = \mathit{stage}(sn) \ \Rightarrow \ (\\
\qquad\quad (\mathit{sched}(n) = 0 \iff X = Y) \wedge \\
\qquad\quad (\mathit{sched}(n) > 0 \iff Y = \mathit{gatherTriples}(\mathit{expandSet}(X))) \\
\qquad) \\
\quad)
\end{array}
$$

FIG. 5. Specification of simulated annealing in LST

$$
\begin{array}{l}
\mathit{expandSet}(X) = \{ \ \mathit{expandTriple}(\langle T, H, q\rangle) \mid \langle T, H, q\rangle \in X \ \} \\
\mathit{expandTriple}(\langle T, H, q\rangle) = \{ \ \langle T, H, q \cdot \mathit{pNoAdvance}(H, \mathit{sched}(n))\rangle \ \} \cup \\
\qquad \bigcup_{H' \in \mathit{children}(H)} \{ \ \langle T', H', q \cdot \mathit{pVisit}(H', H, \mathit{sched}(n))\rangle \mid \textbf{wasAdded}(H', T, T') \ \} \\
\mathit{gatherTriples}(Z) = \{ \ \langle T, H, s\rangle \mid \exists z \in Z : \langle T, H, q\rangle \in z \ \wedge \ s = \sum_{\substack{z \in Z \\ \langle T, H, q\rangle \in Z}} q \ \} \\
\mathit{pNoAdvance}(H, t) = \dfrac{|\mathit{worseChildren}(H)| - \sum_{H' \in \mathit{worseChildren}(H)} e^{(h(H) - h(H'))/t}}{|\mathit{children}(H)|} \\
\mathit{worseChildren}(H) = \{ \ H' \in \mathit{children}(H) \mid h(H') > h(H) \ \} \\
\mathit{pVisit}(H', H, t) = \begin{cases} e^{(h(H) - h(H'))/t} \ / \ \mathit{children}(H) & \text{if } h(H') > h(H) \\ 1 \ / \ \mathit{children}(H) & \text{if } h(H') \leq h(H) \end{cases}
\end{array}
$$

FIG. 6. Terms used in the specification of simulated annealing

7 Verifying Properties of Search Strategies

By using the sound and complete sequent calculus for Local Set Theory defined in [1], one can prove properties of the search strategies specified in our model. We offer below some brief comments on examples of provable formulae involving greedy search and simulated annealing. The properties are quite simple and intuitive, and are included here only for illustrative purposes. The proofs themselves are not presented, and although a bit lengthy, can be easily constructed.

The provable formula that says that simple greedy search is deterministic asserts that each set $\mathit{stage}(n)$ is a singleton:

$$
\begin{array}{l}
\forall \mathit{init}, \mathit{step}, \mathit{answer}, F, h : \textbf{isGreedy}(\mathit{init}, \mathit{step}, \mathit{answer}, F, h) \ \Rightarrow \\
\quad \forall n : \exists! \langle T, H\rangle : \langle T, H, 1\rangle \in \mathit{stage}(n)
\end{array}
$$

Furthermore, simple greedy search never backtracks. Equivalently, at each iteration, the subforest consisting of all nodes visited by the search is actually a linear tree. This is expressed by the provable formula (recall that a node is identified with

a path from the root to the node; i.e., a finite linear tree)

$$\forall init, step, answer, F, h : \mathbf{isGreedy}(init, step, answer, F, h) \Rightarrow \\ \forall n : \forall \langle T, H, q \rangle \in stage(n) : \mathbf{isNode}(T)$$

Actually, an analogous formula for simulated annealing is also provable.

Although not shown in the paper, one can use the logic to specify properties of the instance d of the problem $P = \langle D, R, p \rangle$ being solved and of the search forest F assigned to it. Once this is done, one can compare the performances of simple greedy search and simulated annealing in solving the problem in question. For example, the following formula states that (for the given problem P and in the given search forest F, using a single heuristic function h) the simple greedy search strategy $\langle init, step, answer \rangle$ (with associated function $stage$) fails in finding a correct answer, whereas the simulated annealing strategy $\langle init', step', answer' \rangle$ (with associated function $stage'$) has a better than 50% chance of succeeding:

$$\begin{array}{l} \forall n : ((stage(sn) = stage(n) \Rightarrow \forall r : (\langle r, 1 \rangle \in answer(n) \Rightarrow \neg correct(r))) \\ \quad \wedge (stage'(sn) = stage'(n) \Rightarrow \\ \qquad \exists r \exists q : (\langle r, q \rangle \in answer'(n) \wedge correct'(r) \wedge q \geq 1/2)) \\ ) \end{array}$$

Alternatively, the model of the problem P and the search forest F could be defined outside our logical language, and verification of the above formula be conducted through a model-checking procedure.

The implementation of theorem-proving and model-checking techniques for local set theory is work in progress. As soon as this is completed, we will be able to present examples of the verification of more complex, real-life properties of search spaces and search strategies.

8 Conclusion

8.1 Related work

Models, classifications and taxonomies of heuristic techniques have appeared in the literature since the 70's. An example of a well-founded formal approach is the work presented in [9], which seeks to define a grand unifying model for heuristic search, branch-and-bound algorithms and dynamic programming.

More recently, there has been great interest in formal models for search strategies motivated by the need to define and implement software frameworks and object-oriented algorithms for optimization problems, as discussed in [4] and [15]. The models defined in this kind of work are usually in the form of class or component libraries in object-oriented languages or in the form of new modeling languages (or search-oriented extensions of existing languages) like that in [13], possibly coupled with automatic problem solvers.

Our use of category- and topos-theoretical tools and techniques to build frameworks for problems and (meta)heuristics is, to the best of our knowledge, unprecedented. In [2], different categories of problems and reductions ("questions and answers") are examined, and — somewhat surprisingly — found to be present in subjects as diverse as linear logic and cardinal characteristics of the continuum. Our category **Prob** of

problems is a minor variation of one of these categories of questions and answers. A deeper study of **Prob** and related categories is beyond the scope of this article, but can be found in [3].

In [10], categories of optimization problems and reductions are defined, and approximation algorithms are examined from a categorical perspective, but no topos-theoretical considerations are made, and the approach is not logic-based.

The Formal Methods community has long been using higher-order logics (HOL) as a framework in which to specify software systems and to reason about their properties. As LST is a higher-order language, the framework presented in this article could be faithfully represented in HOL (with a suitable definition of the type of rational numbers), with no mention of categories or topoi. However, if we had chosen the HOL approach, we would soon be confronted with questions about the existence of models of our specifications and about the nature of these models. By taking the **FA** topos as our starting point, we are formally defining our models and then eliciting the very theory of these models. It seems to us that (1) using HOL specifications and their models and (2) using the theory of our topos in the language of LST are different ways of achieving the same goal, with (2) apparently showing the advantage of having a precisely defined model as a starting point. Furthermore, from a foundational point of view our approach is preferable inasmuch as it does not allow the use of reasoning external to the models we are interested in.

8.2 Further work

We believe our most important achievements so far have been the successful use of topos-theoretical tools to define our model and the generation of examples to provide evidence to the effect that the model is comprehensive enough to represent the techniques used by practitioners. In fact, [3] discusses further examples of specifications of (meta)heuristics, including the well-known paradigm of Genetic Algorithms.

In our view, (meta)heuristics are built from search spaces and search strategies. The former are represented by functors, and the latter are represented by natural transformations. In a strong sense, we may say that this view provides a complete model. Furthermore, this model is made formal by means of adequate manipulations on the respective local set theories, yielding means of proving properties, besides correctness, of any conceivable strategy. This goes beyond modeling languages like the one presented in [13] (which usually consist of formalisms whose main purpose is the expression of algorithms) in the sense that our formal model also supports property-checking.

We envisage some applications of the present work: (1) Logical specifications can be used to check the correctness of more concrete representations of (meta)heuristics, possibly even in the form of code in a programming language or in a modeling language like that presented in [13]. To this end, the study and implementation of theorem-proving and/or model-checking techniques for local set theory would certainly lead to interesting and useful results; (2) the language of our model can be used to define a high-level software framework, which would be refined through the use of techniques of code transformation to generate modules in some programming language; these generated modules would then be combined with a fixed library of supporting modules, written in the programming language, to generate complete im-

plementations of (meta)heuristics, profiting from the code reuse advantages offered by software frameworks.

References

[1] J. L. Bell. *Toposes and Local Set Theories, an Introduction*. Oxford U. Press, 1988.

[2] Andreas Blass. Questions and answers: a category arising in linear logic, complexity theory, and set theory. In *Proceedings of the Workshop on Advances in Linear Logic*, pages 61–81. Cambridge University Press, 1995.

[3] Fernando Náufel do Amaral. *Teoria de Modelos para Heurísticas Baseada em* Topoi *(in Portuguese)*. PhD thesis, PUC-Rio, Brazil, 2004. Available at `http://www2.dbd.puc-rio.br/pergamum/biblioteca/php/mostrateses.php?open=1&arqtese=0016027_04_Indice.html`.

[4] A. Fink, S. Voss, and D. Woodruff. Metaheuristic class libraries. In Glover and Kochenberger [6].

[5] Peter Freyd. Aspects of topoi. *Bulletin of the Australian Mathematical Society*, 7:1–76 and 467–480, 1972.

[6] F. Glover and G.A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer, 2002.

[7] R. Goldblatt. *Topoi – the Categorial Analysis of Logic*. North Holland, 1979.

[8] P. T. Johnstone. *Sketches of an Elephant: a Topos Theory Compendium*. Oxford U. Press, 2002.

[9] V. Kumar. A general heuristic bottom-up procedure for searching AND/OR graphs. *Information Sciences*, 56(1–3):39–57, 1991.

[10] L. A. S. Leal, P. B. Menezes, D. M. Claudio, and L. V. Toscani. Optimization problems categories. In R. Moreno-Diaz and A. Quesada-Arencibia, editors, *EUROCAST 2001 – Extended Abstracts*, pages 93–96, 2001.

[11] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley, 1985.

[12] K.-D. Schewe and J. M. Turull Torres. A theory of local set queries. Technical Report 2003/03, Dept. of Information Systems, Massey University, New Zealand, 2003.

[13] P. Van Hentenryck and L. Michel. The modeling language OPL: A short overview. In Voss and Woodruff [15].

[14] P. A. S. Veloso and S. R. M. Veloso. Problem decomposition and reduction: Applicability, soundness, completeness. In R. Trappl, editor, *Progress in Cybernetics and Systems Research*, volume 8, pages 199–203. Hemisphere Publ. Co., 1981.

[15] S. Voss and D. L. Woodruff, editors. *Optimization Software Class Libraries*. Kluwer, 2002.

[16] Stefan Voss. Meta-heuristics: The state of the art. In *Proceedings of the Workshop on Local Search for Planning and Scheduling*, pages 1–23. Springer-Verlag, 2001.

Received September 15, 2006