

Chapter 3

Constructing taxonomies

In this chapter, we provide a brief introduction to the construction of taxonomies. The McGraw-Hill Dictionary of scientific and technical terms provides the following (somewhat biology oriented) definition of a *taxonomy*:

A study aimed at producing a hierarchical system of classification of organisms which best reflects the totality of similarities and differences. [Park89, p. 1892].

In a manner analogous to a biological taxonomy, we intend to classify algorithms according to their essential details. This classification, which is frequently presented in the form of a (*directed acyclic*) *taxonomy graph*, will allow us to compare algorithms and determine easily what they have in common and where they differ. In the following paragraphs, we detail the structure and construction of a taxonomy.

Given a particular problem area (for example, keyword pattern matching), the algorithms will be derived from a common starting point. The starting point is usually a naïve algorithm whose correctness is shown easily. Each of the algorithms appears as a vertex in the taxonomy graph, and the first algorithm is placed at the root of the taxonomy graph. The derivation proceeds by adding either *problem* or *algorithm* details. A problem detail is a correctness preserving restriction of the problem. Such a detail may enable us to make a change in the algorithm, usually to improve performance. The more specific problem may permit some transformation which is not possible in the algorithm solving the general problem. An algorithm detail, on the other hand, is a correctness-preserving transformation of the algorithm itself. These algorithm details may be added to restrict nondeterminacy, or to make a change of representation; either of these changes to an algorithm, gives a new algorithm meeting the same specification. In the taxonomies presented in Chapters 4 and 6, the particular details are explicitly defined and given mnemonic names. In the remaining taxonomy (Chapter 7), the details are only introduced implicitly.

Both types of details are chosen so as to improve the performance of an algorithm, or to arrive at one of the well-known algorithms appearing in the literature. The addition of a detail to algorithm A (arriving at algorithm B) is represented by adding an edge from A to B (the vertices representing algorithms A and B , respectively) to the taxonomy graph. The edge is labeled with the name of the detail. The use of correctness preserving transformations, and the correctness of the algorithm at the root of the graph, means that

the correctness argument for any given algorithm is encoded in the root path leading to that particular algorithm.

It should be noted that, while the taxonomy is presented in a top-down manner, the taxonomy construction process proceeds initially in a bottom-up fashion. Each of the algorithms found in the literature is rewritten in a common notation and examined for any essential components or encoding tricks. The common encoding tricks, algorithm skeletons, or algorithm strategies can be made into details. The details making up the various algorithms can then be factored, so that some of them are presented together in the taxonomy graph — highlighting what some of the algorithms have in common.

A few notes on the taxonomy graph are in order. In some cases, it may have been possible to derive an algorithm through the application of some of the details in a different order¹. The particular order chosen (for a given taxonomy) is very much a matter of taste. A number of different orders were tried, the resulting taxonomy graphs were compared, and the most elegant one was chosen. Frequently, the taxonomy graph is a tree. When the graph is not a tree, there may be two or more root paths leading to algorithm *A*. This means that there are at least two ways of deriving *A* from the naïve algorithm appearing at the root. It is also possible that not all of the algorithms solving a particular problem can be derived from a common starting point. In this case, we construct two or more separate taxonomy graphs, each with its own root.

This type of taxonomy development and program derivation has been used in the past. A notable one is Broy's sorting algorithm taxonomy [Broy83]. In Broy's taxonomy, algorithm and problem details are also added, starting with a naïve solution; the taxonomy arrives at all of the well-known sorting algorithms. A similar taxonomy (which predates Broy's) is by Darlington [Darl78]; this taxonomy also considers sorting algorithms. Our particular incarnation of the method of developing a taxonomy was developed in the dissertation of Jonkers [Jonk82], where it was used to give a taxonomy of garbage collection algorithms. Jonkers' method was then applied successfully to attribute evaluation algorithms by Marcelis in [Marc90]. A recent taxonomy (not using Jonkers' method) by Hume and Sunday [HS91] gives variations on the Boyer-Moore pattern matching algorithms; the taxonomy concentrates on many of the practical issues, and provides data on the running time of the variations and their respective precomputations.

Two primary aims of the taxonomies are clarity and correctness of presentation. We abandon low levels of abstraction, such as indexing within strings. Instead, we adopt a more abstract (but equivalent) presentation. Because of this, all of the abstract algorithms derived in this dissertation will be presented in a slightly extended version of Dijkstra's *guarded command language* [Dijk76]. The reasons for choosing the guarded command language are:

- Correctness arguments are more easily presented in the guarded commands than in programming languages such as Pascal or C.

¹Only some of the details may be rearranged, since the correctness of some details may depend upon the earlier application of some detail.

- In order to present algorithms that closely represent their original imperative presentations (in journals or conference proceedings), we do not make use of other formalisms and paradigms, such as functional or relational programming.

We will frequently present algorithms without full annotations (invariants, preconditions, and postconditions) since most of the algorithm skeletons are relatively simple, and the annotations do not add much to the taxonomic classification. Annotations will be used when they help to introduce a problem or an algorithm detail.

This part is structured as follows:

- Chapter 4 presents a taxonomy of keyword pattern matching algorithms. The taxonomy concentrates on those algorithms that perform pattern matching of a finite set of keywords, and those algorithms that do not use precomputation of the input string.
- Chapter 5 gives a derivation of a new regular expression pattern matching algorithm. The existence (and derivation) of the algorithm answers an open question first posed by A.V. Aho in 1980. The algorithm, which is a generalization of the Boyer-Moore keyword pattern matching algorithm, displays good performance in practice.
- Chapter 6 presents a taxonomy of algorithms which construct a finite automaton from a regular expression. All of the well-known algorithms (including some very recently developed ones) are included.
- Chapter 7 presents a taxonomy of deterministic finite automata minimization algorithms. All of the well-known algorithms, and a pair of new ones, are included.