A GENERAL THEORY OF BOUNDARY-BASED QUALITATIVE REPRESENTATION OF TWO-DIMENSIONAL SHAPE

Richard Charles Meathrel

Submitted by Richard Charles Meathrel to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Computer Science in the Faculty of Science.

September 2001

I certify that all material in this thesis which is not my own work has been identified and that no material is included for which a degree has previously been conferred upon me.

Richard Charles Meathrel

Abstract

Shape is an important spatial attribute that features in much of our everyday reasoning and intelligent action. Consequently, representations of shape are important in the domain of Artificial Intelligence (AI). For the kind of commonsense reasoning that AI is interested in, it is necessary to represent shape in qualitative terms, using higher-level symbolic representations in preference to (or in addition to) low-level numerical data. Schemes for representing shape are most commonly categorised as either region-based or boundary-based. In this thesis, we focus our attention on the boundary-based approach to the qualitative representation of two-dimensional shape.

Existing boundary-based schemes exhibit a number of common features. They all use high-level descriptors which are ultimately analysable in terms of qualitative curvature variation. The central claim of this thesis is that such an analysis provides an adequate theoretical underpinning for the boundary-based approach, leading to a theory which provides a unifying account of, and generalises, existing boundary-based schemes.

Our analysis results in an unbounded hierarchy of *atomic tokens*, each of which corresponds to a particular qualitative composition of tangent bearing and zero or more of its derivatives. A shape has a token-string description at each level of the hierarchy and each level provides a conceptual granularity that is finer than the previous level and coarser than the next. Strings of atomic tokens give rise to *complex tokens* capable of representing localised curve features of greater abstraction. A *local-feature scheme* is defined as a set of complex tokens that supports token-string description. Given a local-feature scheme, a *token-ordering graph* can be constructed that visually encodes its token-ordering constraints. The atomic and complex tokens, local-feature schemes, and token-ordering graphs constitute a general theory of the boundary-based approach, which we call *qualitative boundary theory* (QBT). We apply QBT by showing that each of the existing boundary-based schemes is definable as a local-feature scheme.

Acknowledgements

First, and most importantly, I would like to thank Antony Galton for supervising my research and providing encouragement and support when it was needed. I greatly appreciate the guidance and time that Antony has given me over the last four years. Secondly, I would like to thank the other three members of my thesis committee, whose comments and advice I always found to be considered and useful. They are Derek Partridge, Jamie Henderson, and Helen Gaylard.

During my years at Exeter I have had the pleasure of meeting and getting to know many new people, for this I am grateful. In the four years that I have been undertaking my doctoral research, there are four people who I would like to thank by name, for their presence, friendship, and advice. They are Robert Bishop, Michael Grivas, Gareth Penny, and Julia Wallace.

- Chapter 3 includes material presented at the *16th International Joint Conference on Artificial Intelligence (IJCAI'99)* in Stockholm, Sweden (Galton & Meathrel 1999).

- An initial version of QBT was presented at the *14th European Conference on Artificial Intelligence (ECAI2000)* in Berlin, Germany (Meathrel & Galton 2000).[1]

- Chapter 4 includes material presented at the *17th International Joint Conference on Artificial Intelligence (IJCAI'01)* in Seattle, USA (Meathrel & Galton 2001).

[1]The acronym QBT was not used in the paper. The "primitive curve tokens" (PCTs) referred to in the paper have evolved into the "complex tokens" described in Chapter 5 of this thesis. The paper also contains the first two levels of the atomic hierarchy (cf. Chapter 4), the notion of a "local-feature scheme" (cf. Chapter 6), and a simple method for constructing token-ordering graphs (cf. Chapter 7).

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this chapter, we discuss the importance of shape for AI reasoning tasks and the complexity involved in the general representation of shape. We choose to focus our attention on the qualitative representation of two-dimensional shape. We propose the development of a theory, based on tangent bearing and its successive derivatives, that underpins the boundary-based approach to the qualitative representation of two-dimensional shape.

1.1 Background

1.1.1 The nature of shape

Shape is a ubiquitous spatial attribute in our everyday lives. All the objects that we perceive, by sight or touch, have a shape. We use the word "shape" to describe a number of different phenomena. We may be describing the configuration of the space that an object occupies, we may be talking about the formation of a flock of birds in the sky, or we may be interested in the spatial distribution of components that make up some artefact. In each of these cases, the common denominator is the notion of shape. Most of the time, when we are awake and alert, we are processing shapes. There is a constant need to recognise objects in our environment so that we may manipulate them and, when navigating, avoid them. Without the ability to perceive shape in any way our lives would be very restricted.

Shape is of importance to us as it is involved in much of the processing that characterises intelligent behaviour, such as recognising objects, distinguishing between objects ("I want the round one"), describing objects, assessing the possible behaviour and uses of objects, and manipulating objects.

Shape is a complicated spatial attribute, because of its almost infinite variation and richness. The spatial attribute *position*, for example, is easier to specify. To locate an object in space we can specify its coordinates in some system relative to some origin. Shape is more complex than this, and, compared with position, is analogous to dealing with an infinite number of points, each of which could be considered an almost insignificant part of a substantial whole. Shape carries with it a great deal of information, and our capabilities with regard to shape, although taken very much for granted, are impressive. Perhaps most impressive of all is the capability we have for recognising objects that we have seen before and classifying objects that we have never before encountered. We perform this task, principally, by perceiving an object's shape and comparing it with the shapes of objects that are held in memory from previous experience. In doing this, there is a need to take account of the object's orientation in space (since we may be viewing it from a novel angle) and also the possibility of occlusion: some parts of the object may not be visible to us (hidden by virtue of orientation) or there may be some other object(s) obscuring our view. We must also bear in mind that the complexity of object shapes necessitates the construction and storage of shape descriptions that characterise, rather than exactly replicate, the shapes that we perceive. This raises an interesting question: how do we internally describe shape, so that our descriptions contain the information and structure required to allow us to efficiently recognise objects and perform related tasks of manipulation and reasoning? Do we have just one representational scheme, or a number of separate schemes, each tailored to a particular processing requirement?

1.1.2 Representing shape

Given that the processing of shape is necessary for a wide range of tasks that demonstrate intelligence, the representation of shape is an important concern for AI. Shape plays a crucial role in the recognition of objects and environmental features, as we have discussed. For this kind of task, different levels of processing are required, given the need to take sensory data as input (in the form of retinal images) and extract higher-level information. First it is necessary to process the data at a low level, to infer such information as contour segments and surfaces. This level of processing is the subject matter of the field of low-level computer vision (see, for example, (Ballard & Brown 1982, Boyle & Thomas 1988)). Higher-level visual processing involves models that have been derived from the primitive information obtained at earlier stages. Two such higher-level models include the stick-figure representation developed by Marr & Nishihara (1978) and the componential approach of Biederman (1987).

In addition to the field of computer vision, there are other areas of AI research where formalisms for representing shape are required. One of these, and the one that will be of most interest to us in this thesis, is the area of research known as qualitative spatial reasoning (QSR), a relatively new field that has evolved from earlier work on qualitative physics (QP). The aim of QP being to develop useful models of the physical world and physical systems that exploit commonsense knowledge (see (Forbus 1990) for a survey of the work carried out in QP). As suggested by its title, QSR is concerned with spatial representations that can be described as qualitative in nature, rather than quantitative. This is so that representations are supportive of tasks involving high-level reasoning. Formalisms for a number of spatial attributes have been developed, e.g., topology (Randell et al. 1992), position (Hernández et al. 1995), distance (Hernández et al. 1995), and orientation (Latecki & Röhrig 1993). The qualitative representation of shape, however, has not been extensively studied. When we refer to a *qualitative* representation, we mean one that makes explicit certain qualities that are important for particular tasks of reasoning, without including specific metric information that is substantially less relevant to high-level commonsense

reasoning. A predominantly quantitative representation, on the other hand, is one including detailed metric information that may be difficult to process in high-level terms. It is not always easy or desirable to classify a representation as being *either* qualitative or quantitative, there may be occasions when it is useful to augment non-numeric information with metric information. The main point to make is that a representation designed for a particular kind of task must make explicit the information required by the task. The fact that we are interested in commonsense reasoning means that the information needs to be of a relatively high-level nature. The distinction between qualitative and quantitative representations, then, is most usefully considered as one of degree. In general, we can think of a representation as having a qualitative aspect if it includes non-numeric information, and a quantitative aspect if it includes numeric information. For more information about QSR, and the qualitative representation of spatial knowledge, see (Cohn 1997, Freksa & Röhrig 1993, Hernández 1994).

When we develop a representational scheme for describing shape, the decisions we make, with regard to the qualities of shape that we want to make explicit, depend on the processing that the scheme is to be involved in. It isn't feasible, perhaps, to develop a scheme for describing shape that is so generic that it can be used to adequately model shape of any kind for any given kind of task. A more realistic approach is to have a number of separate schemes, each encompassing a particular scope of shapes, and each amenable to a particular class of tasks. Moreover, because of the inherent complexity of shape, any attempt to formulate a scheme that is capable of describing shapes of all kinds would benefit from an analysis of more manageable schemes, i.e., as with most development processes, an earlier prototype can "inform" the design of later ones. For this reason, even if a truly generic scheme is a possibility, there is much value in developing specific schemes with restricted scopes, since such schemes have the potential for highlighting those aspects of shape description that are problematic. In short, specific schemes may inform the development of more general ones.

The most straightforward and useful way of classifying shape representations is to

make the distinction between those that deal with shape in three dimensions and those that deal with shape in two dimensions. In this thesis we choose to focus our attention on the description of two-dimensional shape. We do this for the reasons already given, namely, that the complexity of representing many or all kinds of shape using a single scheme is too great, thereby justifying the development of schemes with more modest scopes. By concentrating on two-dimensional shape, we are still dealing with shapes that have an important role to play in both object recognition and commonsense reasoning. A number of application areas for schemes that qualitatively represent two-dimensional shape are listed in Section 2.1.1.

1.2 Thesis statement

Schemes for representing two-dimensional shape are commonly classified as being either boundary-based or region-based. Boundary-based schemes typically describe the type and position of localised features round the bounding curve of a region. In contrast, region-based schemes base their descriptions on shape interior. In this thesis, we focus our attention on the boundary-based approach to qualitatively representing shape. Existing boundary-based schemes, which describe shapes using strings of symbols, differ as to the number and kind of curve features they represent. Each symbol in a string is a label identifying a particular kind of localised curve feature present on the bounding curve of the shape described by the string. Some examples of localised features include straight-line segments, curvilinear segments, inflection points, and points of tangent discontinuity (i.e., angles and cusps). The kinds of tasks that a particular scheme is suitable for is dependent on the symbols (or "primitives") provided by the scheme. Although schemes differ as to the sets of primitives they provide, those schemes that are instances of the boundary-based approach exhibit a number of common features. Principally, they all use high-level descriptors which are ultimately analysable in terms of qualitative curvature variation. This leads us to the central claim of this thesis:

An analysis of qualitative curvature variation, based on tangent bearing, provides an adequate theoretical underpinning for the boundary-based approach to the qualitative representation of two-dimensional shape, leading to a theory which (i) provides a unifying account of, and (ii) generalises, existing qualitative boundary-based schemes.

In this thesis, an analysis of curvature variation is undertaken, in which tangent bearing and its successive derivatives (of which curvature is the first), are represented qualitatively by values taken from one of two quantity spaces. The analysis yields an unbounded hierarchy of basic shape descriptors, referred to as "atomic tokens", from which a general theory of the boundary-based approach, which we call *qualitative boundary theory* (QBT), is developed. We show that a set of tokens taken from any level of the hierarchy can be regarded as a set of fundamental building-blocks, from which the higher-level primitives of existing boundary-based schemes can be specified. In doing so, we are able to precisely specify the semantics of a primitive, and highlight potential ambiguities relating to the primitives of existing schemes. The atomic tokens, together with rules for combining them into *complex tokens* capable of representing curve features of greater abstraction, define the space of possible higher-level primitives. A complex token is specified by a (possibly infinite) set of atomic-token-string triples, each consisting of an *identity* string, together with leading and trailing *context* strings. The set of primitives of a scheme can be specified as a set of complex tokens. A *local-feature scheme* is defined to be a set of complex tokens which represent features that do not *overlap* round the bounding curve of a shape, thus ensuring every description is a string containing only shape descriptors.[1] We show that each of the qualitative boundary-based schemes in the literature can be defined as a local-feature scheme, by specifying the primitives of each of them as complex tokens and verifying that the features of the scheme do not overlap. A set of tokens (atomic or complex) can be seen as providing a formal language for representing shape. Strings in a language represent closed and/or open curves and are subject to *token-ordering* and *closure*

[1] I.e., no additional symbols are required in the string to encode the relative position of features along the bounding curve.

constraints. We show how the token-ordering constraints for a language (set of tokens) can be visually encoded as a *token-ordering graph*, with nodes representing tokens and directed edges the ordering constraints between tokens. Algorithms are provided for systematically constructing such graphs for sets of atomic and complex tokens. We demonstrate how token-ordering graphs, which we construct for each of the existing boundary-based schemes, can aid in the evaluation and comparison of local-feature schemes.

In summary, QBT consists of three parts: (i) an analysis of qualitative curvature variation yielding an unbounded hierarchy of basic shape descriptors (the atomic tokens), (ii) support for higher-level curve description, including a formalism for specifying complex tokens, and (iii) a theory of token-ordering graphs.

1.3 Structure of the thesis

The remainder of the thesis is structured as follows:

- In Chapter 2 we review existing approaches to the representation of two-dimensional shape, including schemes which differ with respect to the amount of qualitative and quantitative information used for description. We highlight the characteristics of qualitative representations that make them suitable for commonsense spatial reasoning, focussing our attention on boundary-based schemes.

- In Chapter 3 we describe a boundary-based scheme, referred to as "QOT", which represents outlines using a set of seven *qualitative curvature types*. QOT is shown to have greater scope than the boundary-based schemes covered in Chapter 2, but, in some respects, less discriminatory power. These differences highlight the common tradeoff between scope and detail.

- Qualitative boundary theory (QBT) is presented in Chapters 4 to 7. In Chapter 4, the unbounded hierarchy of basic shape descriptors is derived from an analysis of qualitative curvature variation. The correspondence between an open/closed curve

and its token-string description is explained and token-ordering graphs are provided for the first three levels of the hierarchy. Chapter 5 describes the means by which complex tokens are specified in terms of atomic tokens, enabling higher-level curve features to be represented. In Chapter 6, the problem of aggregating complex tokens into sets, such that shapes are described using strings consisting solely of tokens, is addressed, resulting in the definition of a local-feature scheme. In Chapter 7, the construction of a token-ordering graph for a set of complex tokens is discussed. A procedure is then provided for constructing token-ordering graphs for certain restricted kinds of local-feature schemes.

- Chapter 8 provides an analysis of the boundary-based schemes described in Chapters 2 and 3, defining each one as a local-feature scheme and presenting complex-token specifications for the primitives of each scheme. Token-ordering graphs are constructed for each scheme and used to aid the evaluation and comparison of the schemes.

- QBT is evaluated in Chapter 9.

- Finally, in Chapter 10, we conclude by reflecting on the work carried out and highlighting the contributions made by it. We end the chapter by discussing the most promising directions for future research.

Also contained in this thesis are three appendices. Appendix A provides a regular grammar that generates valid string descriptions for the QOT scheme presented in Chapter 3. Appendix B contains a detailed discussion of the construction of token-ordering graphs for sets of atomic tokens, culminating in a construction algorithm. Appendix C provides a list of complex tokens that correspond to the boundary-based primitives of Rosin (1993).

Chapter 2

Existing schemes

In this chapter, we begin by listing the different kinds of applications that require qualitative representations of shape and looking at some important aspects of qualitative schemes. We go on to consider the region-based and boundary-based approaches to representing shape, looking at, and comparing, a number of schemes from the literature.

2.1 Overview of shape representation

In the previous chapter, we highlighted the significance of shape for AI tasks and the importance of shape in general. Representations of shape are required for tasks that involve a degree of reasoning, but they are also required for other kinds of tasks, e.g., non-AI tasks such as those related to the field of computer graphics that involve the *generation* of shapes. As has been already mentioned, it is not feasible to develop a single representational scheme for shape that satisfies the needs of each and every task involving shape. Instead, it is necessary to develop separate schemes that are designed according to particular processing requirements. The information that a scheme makes explicit is therefore dependent on the task(s) it is designed to be used for. Application areas that are not associated with AI usually require representations of shape that are predominantly quantitative in nature, i.e., shapes are described using numerical, rather than non-numerical,

information. Application areas that require quantitative representations of shape include: computer-aided design (CAD), where precise models of artefacts are constructed, and accurate distance and size measurements are important; computer graphics and animation, where bitmapped and vector-based graphics are generated for aesthetic or instructive purposes; the modelling of entity growth and development, as exemplified by tools based on "L-systems" (Rozenberg & Salomaa 1980, Prusinkiewicz & Lindenmayer 1990); and creative architectural design (Chase 1989, Piazzalunga & Fitzhorn 1998, Stiny 1980).

Rather than concerning ourselves with application areas that relate to "non-AI" tasks (such as those just listed), we choose, in this thesis, to focus our attention on representations of shape that include qualitative information that is supportive of commonsense reasoning. Such representations are most appropriate for applications within AI that are concerned with the identification and classification of objects, and other tasks that involve reasoning about shape. Qualitative schemes yield shape descriptions that make explicit those (relatively high-level) qualities of a shape that are considered most important, within a particular context. We may be interested, for example, in how many concavities a shape has, whether it has any lines of symmetry, whether it is rectilinear or curvilinear, whether it has any holes, etc. Information of this kind is usually only *implicit* in numerical descriptions, which sacrifice high-level information for low-level detail. Although it may be possible to infer the kind of high-level information we are interested in from numerical descriptions, extraction of the information is likely to be computationally expensive. Looking at the bigger picture and, in particular, the origin of qualitative descriptions, it is of course likely that, in many circumstances, some process is carried out to obtain a high-level description from a lower-level one. The field of computer vision serves as a good example here, since low-level raster images are commonly the input, and these need to be converted into a higher-level representation before object identification is feasible. It is instructive to note that qualitative representations are to be considered different from quantitative ones (rather than "better"), and are preferable only in particular situations. In this thesis, we are primarily concerned with schemes that contain qualitative information, because we are

interested in supporting applications within the domain of AI.

2.1.1 Application areas for qualitative schemes

Here we list a selection of application areas that require representations of shape that are predominantly qualitative, to give an idea of the kinds of systems that could benefit from the general theory that we develop in Chapters 4 to 7.

- *Object identification and classification*

  Although low-level vision, which tackles the extraction of primitive image features such as edge contours, is not of direct concern to us, *high-level* visual processing includes tasks of object identification and classification that require qualitative descriptions. There exist recognition-oriented representations for shape that use primitives based on the idea of a *generalised cylinder* (see Ballard & Brown 1982), most notably the 3-D model representation of Marr & Nishihara (1978) and the *geon*-based representation of Biederman (1987, 1988). These schemes aim to construct qualitative 3-D models of objects from images, and are best suited to the identification of objects characterised by well-defined part structures. Another recognition-oriented approach, which focuses on part segmentation but doesn't construct 3-D models, i.e., stays within the framework of two dimensions, describes shapes using *contour codons* (Hoffman & Richards 1984, Rosin 1993).

- *Qualitative kinematics (QK)*

  QK is a domain concerned with qualitative reasoning about the motion of objects. The movement of an object may or may not be constrained by other objects; whether motion is possible, and what kinds of motion are possible, is in part a function of the configuration of the objects and their shapes. For a survey of work in this area see (Weld & de Kleer 1990). Related work includes Shoham's (1985) analysis of the concept of *freedom*, and the feature algebra of Karinthi & Nau (1989).

- *Sketch/diagram understanding*

  In order to compute the meaning of a drawing, it is necessary to formulate high-level descriptions of the individual elements that the drawing comprises. A line drawing could originate from an automated system, or it may have been produced by hand. It is reasonable to assume that computer-generated drawings are usually easier to process, because their content is likely to be consistently produced, according to fixed and accessible rules. Perhaps a hybrid system, consisting of a suitable representation of shape, together with an Optical Character Recognition (OCR) subsystem, would be useful here. Freeman (1974) has surveyed the computer processing of line drawings. A more recent survey, of diagrammatic representation and reasoning, is provided by Kulpa (1994).

- *Geographical Information Systems (GIS)*

  Within the domain of GIS, shape is clearly important, because much geographical data relates to the shape of landmasses and other features on the Earth's surface. In the context of GIS, modelling shape is of interest when predicting the movement, formation, and deformation of landmasses and landmass features. Two qualitative shape representations of particular relevance here are those of Cohn (1995) and Leyton (1988), where shape concavities are given the most emphasis.

- *Evaluating spatial expressions*

  Some conception of the shapes of objects is necessary in order to evaluate certain spatial expressions (Herskovits 1997). Consider the following example of preposition applicability. It makes no sense to say that "the flower is in the table". The combination of "the flower", "the table", and the preposition "in" would not normally be accepted as a pragmatically correct sentence. We can, however, say that "the flower is *on* the table". By substituting "flower" with "nail", either preposition may be used. This example suggests that a consideration of the shapes of the objects in a discourse may help determine which utterances containing prepositions are allowable.

2.1.2 Aspects of qualitative schemes

Marr & Nishihara (1978) provide, along with a discussion of their 3-D model representation, three criteria for assessing the suitability of a shape representation for object recognition. As a starting point, it will be useful to recall these, because they are relevant to other kinds of tasks as well. The first criterion is *accessibility*, which relates to the computability of a description. Clearly, and for object recognition in particular (since there may be a real-time requirement), it is important that shape descriptions can be computed from source data within a reasonably short space of time. As well as considering the efficiency of computation, descriptions must actually *be* computable; as Marr and Nisihara point out: "There are fundamental limitations inherent in the information available in an image – for example its resolution – and the requirements of a representation have to fall within the limits of what is possible" (p. 270). The second criterion is *scope and uniqueness*. The scope of a representation is given by the class of shapes that the representation is designed to describe. Marr and Nishihara's 3-D model representation has as its scope stick-figure objects, e.g., animals, because those are the kinds of entities that can be meaningfully approximated by arrangements of simple cylindrical primitives. The uniqueness of a representation relates to the issue of viewpoint invariance. For recognition, it is important that there exists a canonical description of a shape that is independent of the viewing position, otherwise "...at some point in the recognition process, the difficult problem of deciding whether two descriptions describe the same shape would have to be addressed" (p. 272). The third and final criterion is *stability and sensitivity*. A representation is considered stable if similar shapes, i.e., shapes that differ only in minor respects, have similar descriptions. To complement stability, the notion of sensitivity refers to the extent to which a representation is able to handle fine detail. The criterion of stability and sensitivity is related to the notion of granularity.

Freksa & Röhrig (1993) present a general discussion of spatial knowledge *dimensions*, as a preface to a comparison of a number of spatial reasoning research projects. Although some of the dimensions are not applicable here (because they are interested in the domain

of space, rather than the representation of shape specifically), two of them are worth mentioning. The first, *granularity*, refers to "...the resolution of spatial features or decision criteria and to the transition between different levels of resolution". The other dimension of interest concerns how a system handles lack of knowledge. More specifically, Freksa & Röhrig identify two senses in which the information available to a system may be considered "incomplete": (i) certain details may be inaccessible and not possible to extract (as is the case with occluded objects in computer vision, for example), or (ii) the information may not be of sufficient precision. We include the granularity dimension, and the first interpretation of incompleteness, in our list of important aspects of a qualitative shape scheme.

We are now in a position to list those aspects of a qualitative shape representation that we consider to be most relevant. We will revisit the list later on in the chapter, when we are presenting and comparing various schemes from the literature. From this point onwards, we are solely interested in qualitative schemes for describing shape in two dimensions, although whenever a scheme applies also to three dimensions we will make this clear. We also make the assumption that all of the schemes we are interested in yield descriptions that are invariant with respect to translation, rotation, and uniform scaling transformations. Any exceptions to this assumption will be highlighted. Our assumption corresponds, in a loose sense, to Marr & Nishihara's *uniqueness* criterion. Here, then, is our list of important aspects of qualitative shape representations that deal with the description of two-dimensional shape:

- *Computability*

  Are descriptions actually obtainable in practice and can they be obtained efficiently? Of importance with regard to computability are the sources from which descriptions are obtained. As well as considering the "overall" computability of a scheme, it may be the case that certain primitives of the scheme are easily computable, whereas others are more difficult (or very difficult) to compute. In this thesis we are interested in the sources that are capable of providing descriptions, rather than any difficulties

that may be associated with a description's construction by a source, e.g., a low-level computer vision process may be one of a chain of processes that derives a description, but we are not interested in the details of such low-level processing.

- *Scope*

  What range of shapes are adequately described by the scheme? Are there shapes that cannot adequately be described because they are characterised by features not modelled by any of the available primitives?

- *Primitives*

  What are the basic building-blocks from which descriptions are composed? How many primitives are available, and are they domain-specific or of a more general kind? How computable is each primitive? As well as geometric building-blocks, the notion of a "primitive" encompasses the basic operations that may be available for constructing descriptions and specifying the relationships between the building-blocks that comprise a description.

- *Discriminatory power*

  Because we are dealing with predominantly qualitative schemes, it is not the case that each description corresponds to only one shape; rather, each description corresponds to (approximates) an equivalence class of shapes. We can think of a scheme as partitioning a "shape-space" into a set of such equivalence classes. The discriminatory power of a scheme is given by the partition defined by it. It is therefore the nature of partitions that allows us to compare the discriminatory power of differing schemes. The precise partition defined by a scheme is dependent on its scope and primitives.

- *Granularity*

  Is the representation able to describe the same shape at more than one level of detail? And, if so, how many levels are supported? Is the notion of granularity built into

the system, i.e., are operations provided for moving between the different levels of a description and for relating descriptive elements from different levels?

- *Incomplete information*

  This aspect relates to the extent to which a scheme is able to cope with missing information. If a shape is occluded in some way, does the resultant description differ significantly from that obtained if the shape were wholly visible? Is the scheme capable of encoding, in the description, the fact that information is missing?

2.2 Region-based schemes

Schemes that we may classify as region-based place greater emphasis on the *interior* of a shape, as opposed to the bounding contour of a shape. In this section, we concentrate on region-based schemes and, in the next section, we switch our attention to boundary-based schemes. We start this section by briefly considering the symmetry-based approach to describing shape, we then go on to describe the more qualitative approach of Cohn, which is based on the connection primitive of the Region Connection Calculus (RCC) together with a convexity primitive.

2.2.1 Symmetry-based representations

The most popular representation based on symmetry is the Symmetric Axis Transform (SAT)[1], as described by (Blum & Nagel 1978). The symmetry axis of a shape is given by the locus of the centres of the maximal discs that fit within the shape. A maximal disc is a circle which touches the boundary at at least two points, exists within the inner region defined by the boundary, and is not wholly contained by any other circle within the region. The SAT is defined by the symmetry axis together with a radius function which maps each point along the axis to the radius value of the maximal disc associated with that point. Two example shapes, and their symmetry axes, are given in Figure 2.1.

[1] Also known as the Medial Axis Transform (MAT).

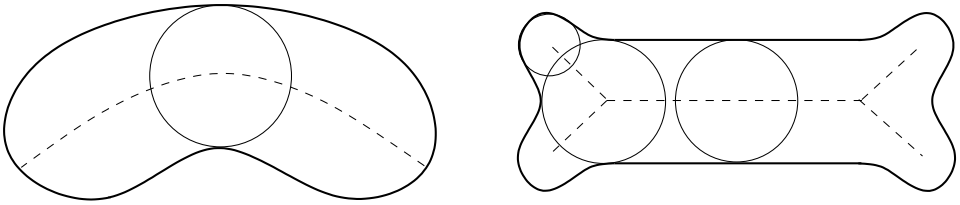


Figure 2.1: Two shapes and their symmetry axes

The values of the radius function are normalised, with the effect that SAT descriptions are independent of an external coordinate system and therefore invariant with respect to translation, rotation, and uniform scaling transformations. Given a description under SAT, the symmetry axis of a shape may be segmented into parts, each of which is referred to as a *simplified segment*. Each point along an axis is either a *normal* point, a *branch* point, or an *end* point. An axis is segmented at its branch points, and each simplified segment consists wholly of an interval of normal points. In Figure 2.1, then, the shape on the left has just one simplified segment, while the shape on the right has five. Blum & Nagel (1978) show how each simplified segment may be qualitatively characterised, by associating with each segment a sequence of elementary descriptors, that reflect the changes in the curvature of the axis segment and associated boundary segments. The maximal discs associated with each point of the simplified segment's axis define the boundary segments that correspond to the part of the symmetry axis given by the simplified segment. There are two boundary segments, a "left" one and a "right" one. Changes in the curvature of the boundary segments relate to changes in the radii of the maximal discs. The elementary descriptors include "worm", "opening wedge", "closing wedge", "cup", and "flare".

The SAT representation is more suited to "natural" shapes than rectilinear ones. It was introduced and explored by Blum, in the context of biology, where the shapes of interest are reasonably well described by the SAT. Blum & Nagel (1978) state that "[t]he SAT is not the simplest description for rectilinear figures" (p. 169). A rectangle, for example, has an unintuitive symmetry axis that is segmented into five simplified segments, as shown in

Figure 2.2(a).

A common criticism of the SAT (see (Pizer et al. 1987)) is that very small changes in the shape of a boundary can result in *significant* changes in the symmetry axis, to the extent that extra branch points may be introduced. As branch points are the principal means by which a shape is segmented into parts, the SATs sensitivity to noise and small detail is of concern, and recognised as problematic. There have been attempts to overcome the sensitivity problem by adopting a multi-resolution approach to SAT description; see (Pizer et al. 1987) and, more recently, (Chung & Ohnishi 1997). Algorithms for computing the SAT of shapes represented by n-sided polygons have been devised, e.g., (Lee 1982).

An alternative symmetry-based analysis is given by Smoothed Local Symmetries (SLS), as described in (Brady 1983, Brady & Asada 1984). Brady re-defines the notion of local symmetry and introduces a subsumption rule, by which an axis is removed if its "producing region" is wholly subsumed by that of another axis. The SLS analysis gives preferable results for rectilinear shapes by suppressing the "mini-axes" that appear in SAT descriptions, as shown in Figure 2.2(b).

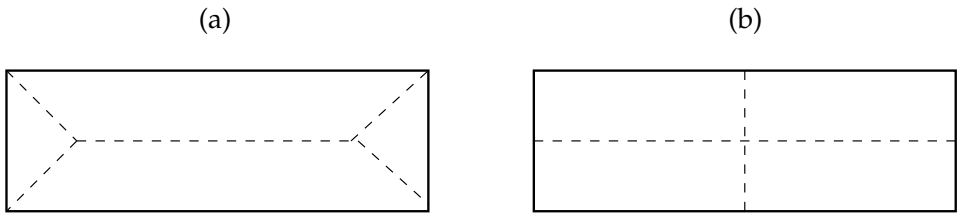


Figure 2.2: The symmetry axes of a rectangle under (a) SAT and (b) SLS

2.2.2 Cohn's RCC-inspired approach

An alternative region-based approach to describing shape, based on the notions of connectivity and convexity, has been explored by Cohn (1995). The logical theory of space known as the Region Connection Calculus (RCC), see (Randell et al. 1992), provides eight jointly exhaustive pairwise disjoint (JEPD) qualitative base relations that may hold between two

regions. The relations are listed in Table 2.1.

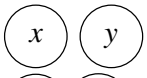
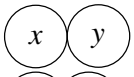
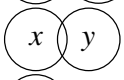
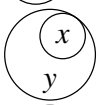
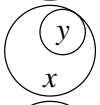
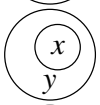
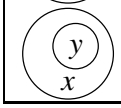
	Relation	*Meaning*
	DC(x, y)	x is disconnected from y
	EC(x, y)	x is externally connected with y
	PO(x, y)	x partially overlaps y
	EQ(x, y)	x is identical to y
	TPP(x, y)	x is a tangential proper part of y
	TPPi(x, y)	x has y as a tangential proper part
	NTPP(x, y)	x is a non-tangential proper part of y
	NTPPi(x, y)	x has y as a non-tangential proper part

Table 2.1: The RCC base relations

The two relations of immediate relevance to Cohn's shape representation are DC and EC. In addition to these two base relations, a primitive function, $Conv(x)$, is introduced, which returns the convex hull of the region x. The description of a shape is given by the number of concavities that a shape has, together with the RCC relationships that hold between each pair of concavities. All *convex* shapes, therefore, have the same description under the representation.

The basic idea of the representation will be described with reference to the shape given in Figure 2.3. The first stage is to determine the concavities of the shape, which correspond to the maximal connected parts of the shape's *geometric inside*. The geometric inside of a shape is given by the remainder of the convex hull of the shape minus the region occupied by the shape itself. The shape in Figure 2.3 has four concavities (or "insides"), labelled c_1, c_2, c_3, and c_4. Each pair of concavities is either disconnected (DC) or externally connected

(EC). With reference to the Figure, we can see that c_1 and c_2 are externally connected, and that all other pairs are disconnected. The basic description of the shape then, using the DC and EC base relations, is as follows:

$$\mathsf{EC}(c_1, c_2) \wedge \mathsf{DC}(c_1, c_3) \wedge \mathsf{DC}(c_1, c_4) \wedge \mathsf{DC}(c_2, c_3) \wedge \mathsf{DC}(c_2, c_4) \wedge \mathsf{DC}(c_3, c_4)$$

As illustrated in the figure, the technique of specifying concavity relationships can be re-applied to the concavities themselves. In this way, shapes with the same number and configuration of concavities can be distinguished, so long as there is a difference in the shape of their concavities that can be expressed. The representation therefore supports hierarchical shape description.

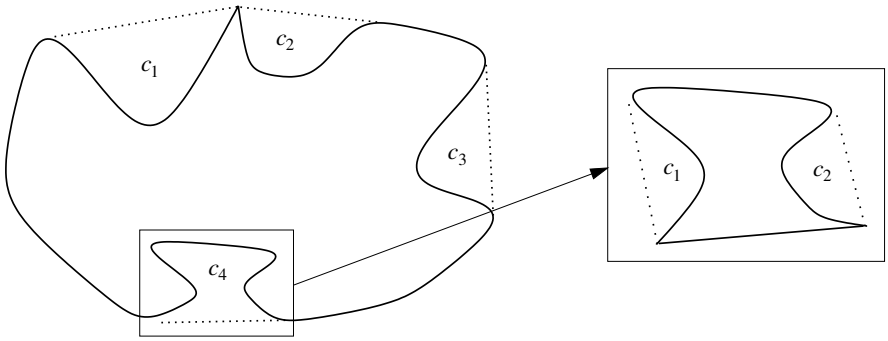


Figure 2.3: Describing a shape by considering its concavities

As described so far, the representation is unable to distinguish between two shapes that have the same number of concavities and differ only with respect to the *order* of the concavity relationships. To overcome this, Cohn shows how a shape can be partitioned in such a way as to detect differences in concavity ordering. To supplement the partitioning technique, the predicate $\mathsf{Adjacent}(c, c')$ is introduced, which holds *iff* the concavities c and c' are adjacent to one another round the perimeter of a shape. With just DC, EC, and Adjacent, it is not possible to distinguish between the three shapes shown in Figure 2.4, because each shape has two concavities that are disconnected and adjacent to one

another[2], i.e., each shape is described by $\mathsf{DC}(c_1, c_2) \wedge \mathsf{Adjacent}(c_1, c_2)$. In order that the shapes *can* be distinguished, three extra predicates are introduced. First, SameSide is defined, which allows the first shape in Figure 2.4 to be distinguished from the other two shapes. $\mathsf{SameSide}(c, c', x)$ holds if the concavities c and c' occur on the "same side" of the shape x. For the first shape, we have $\neg\mathsf{SameSide}(c_1, c_2, x)$. Second, in order to distinguish between the last two shapes, the predicate $\mathsf{SsColinear}(c, c', x)$ is defined, which holds if a straight line can be drawn which ECs all of the "arms" that adjoin the concavities c and c'. Also, the predicate $\mathsf{SsNotColinear}(c, c', x)$ is defined, which holds if $\mathsf{SameSide}(c, c', x)$ holds but $\mathsf{SsColinear}(c, c', x)$ does not.

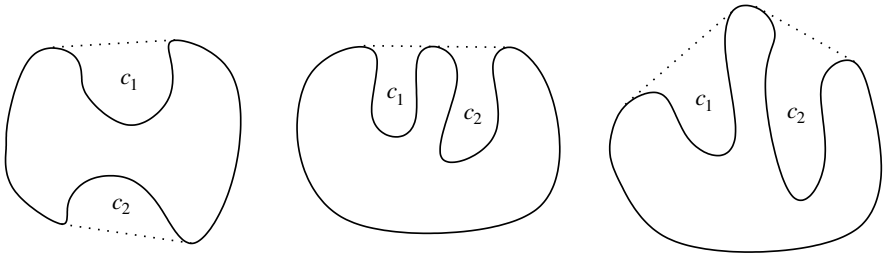


Figure 2.4: Three shapes that Cohn's representation can distinguish between.

The three shapes can now be distinguished:

Leftmost shape: $\mathsf{DC}(c_1, c_2) \wedge \mathsf{Adjacent}(c_1, c_2) \wedge \neg\mathsf{SameSide}(c_1, c_2, x)$

Middle shape: $\mathsf{DC}(c_1, c_2) \wedge \mathsf{Adjacent}(c_1, c_2) \wedge \mathsf{SsColinear}(c_1, c_2, x)$

Rightmost shape: $\mathsf{DC}(c_1, c_2) \wedge \mathsf{Adjacent}(c_1, c_2) \wedge \mathsf{SsNotColinear}(c_1, c_2, x)$

We are now in a position to summarise the constituents of a shape description. A description consists of one or more granularity levels, where the number of levels is dependent on the extent to which the technique is re-applied to the concavities of a shape. The shapes at each level are described by strings of conjuncts representing the relationships that hold between their concavities. For each pair of concavities, $\alpha \wedge \beta \wedge \gamma$ holds, where $\alpha \in \{\mathsf{DC}, \mathsf{EC}\}$, $\beta \in \{\mathsf{Adjacent}, \neg\mathsf{Adjacent}\}$, and $\gamma \in \{\mathsf{SsColinear}, \mathsf{SsNotColinear}, \neg\mathsf{SameSide}\}$,

[2]Note that only shapes with more than three concavities can have two concavities that are not adjacent.

with the constraint that if $\alpha = \mathsf{EC}$ then $\beta = \mathsf{Adjacent}$.[3] Between two concavities, then, one of eleven mutually exclusive relationships must hold. The *full* description of the shape given in Figure 2.3, i.e., including re-application of the technique to c_4 (giving the second level of the hierarchy), is as follows:

Description at first level

$\langle c_1, c_2 \rangle$:	$\mathsf{EC} \wedge \mathsf{Adjacent} \wedge \mathsf{SsNotColinear}$
$\langle c_1, c_3 \rangle$:	$\mathsf{DC} \wedge \neg\mathsf{Adjacent} \wedge \mathsf{SsNotColinear}$
$\langle c_1, c_4 \rangle$:	$\mathsf{DC} \wedge \mathsf{Adjacent} \wedge \neg\mathsf{SameSide}$
$\langle c_2, c_3 \rangle$:	$\mathsf{DC} \wedge \mathsf{Adjacent} \wedge \mathsf{SsNotColinear}$
$\langle c_2, c_4 \rangle$:	$\mathsf{DC} \wedge \neg\mathsf{Adjacent} \wedge \neg\mathsf{SameSide}$
$\langle c_3, c_4 \rangle$:	$\mathsf{DC} \wedge \mathsf{Adjacent} \wedge \mathsf{SsNotColinear}$

Description at second level

$\langle c_1, c_2 \rangle$:	$\mathsf{DC} \wedge \mathsf{Adjacent} \wedge \neg\mathsf{SameSide}$

2.3 Boundary-based schemes

In this section, we focus our attention on schemes that derive their descriptions by traversing the boundary contour of a shape. There are other schemes that may be categorised as "boundary-based" which do not derive descriptions by curve traversal, e.g., Fourier transforms (see Ballard & Brown 1982), the sinusoidal transform (Pratt 1999) and the scale-space approaches (e.g., Witkin 1983, Mokhtarian & Mackworth 1986, Mokhtarian et al. 1996). However, these schemes are predominantly quantitative and therefore only of noteworthy interest to us here. A common characteristic of the schemes we look at here is that shapes are described by strings of symbols, such that each symbol corresponds to a particular localised curve feature located on the bounding curve of the shape. The schemes differ principally with respect to the primitives (symbols) they use, reflecting the salience attributed to particular curve features by each of the schemes.

[3] I.e., concavities that are externally connected must be adjacent.

2.3.1 Contour codons

Hoffman & Richards (1982) introduced a set of four *contour codons* for describing smooth planar curves for recognition purposes. Each contour codon is a segment of curve that is bounded by curvature minima and which contains either zero, one, or two points of zero curvature. The four codon types are labelled 0, 1^+, 1^-, and 2, reflecting the number of zero-curvature points present within each type of segment. There are two codon types that contain a single point of zero curvature (1^+ and 1^-), they are distinguished by the place within the segment at which the point occurs: for the 1^+ codon, the point of zero curvature occurs *after* the point of maximum curvature, whereas, for the 1^- codon, the point of zero curvature occurs *before* the point of maximum curvature. In a more recent paper, (Richards & Hoffman 1987), the 0 codon type is subdivided into two codon types (0^+ and 0^-), increasing the original set of contour codons to five.[4] Figure 2.5 shows the five codons. The dots shown on the curve segments signify points of zero curvature and the slashes minima of curvature. The arrows indicate the direction of curve traversal (from the "tail" of the codon to the "head"). The *figure* is taken to be the region to the left of the direction of traversal and *ground* the region to the right. A curve is traversed in a particular direction, and described by a string of codon labels, according to the variation in curvature encountered along the curve. A curve is segmented into parts according to the presence of negative curvature minima: "[n]atural parts…lie between concave cusps" (Richards & Hoffman 1987, p. 700). An explanation of why negative curvature minima are considered important for determining parts is given in (Hoffman & Richards 1984). Further work on part salience has been carried out by Hoffman & Singh (1997).

Hoffman and Richards' choice of primitives is motivated by their theory of the human visual perception of figures, which attempts to explain how we segment a figure into parts for subsequent recognition tasks. Hoffman and Richards postulate that figures are segmented into parts at negative curvature minima, and this is reflected by the contour

[4] For completeness, they also consider a sixth codon type for straight-line segments, labelled ∞, since a straight line can be thought of as a curve segment containing infinitely many points of zero curvature.

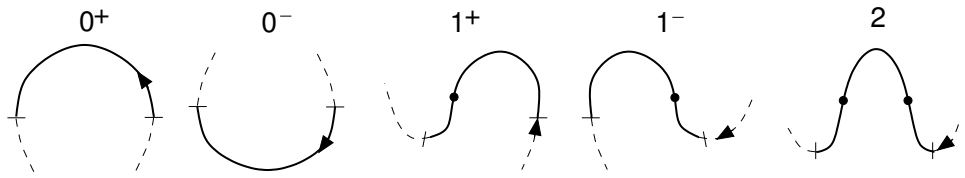


Figure 2.5: Hoffman and Richards' contour codons

codons, which are curve segments bounded by curvature minima rather than curvature maxima or points of zero curvature. For any given curve, there are two directions in which it can be traversed (two "curve orientations"). A reversal of the traversal direction (equivalent to a figure-ground reversal) yields a significantly different codon description, i.e., it is not the case that a reversal of direction simply equates to a reversal of the string description. As a consequence of this difference, part segmentation leads to a different set of parts for each description, and hence for each figure-ground interpretation. The effect of figure-ground reversal is illustrated in Figure 2.6. The curve orientation shown on the left yields the description $1^+\,2\,1^-\,1^+\,0^-\,1^-$, a reversal of traversal direction (the alternative orientation, shown on the right) yields the somewhat different description $0^-\,1^-\,1^+\,0^-\,2\,2$. The dashed lines indicate the likely part segmentation in each case, given the relative positions of the negative curvature minima. We see that a change in orientation results in a markedly different set of parts.

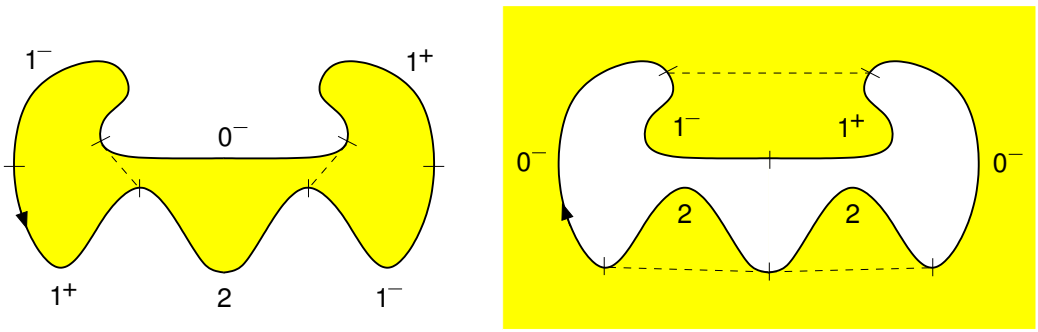


Figure 2.6: A change in curve orientation yields a different codon description

String constraints

Legitimate strings of codons are subject to certain constraints, since not all pairs of codons may smoothly join together. Table 2.2 encodes the constraints, as determined by Richards & Hoffman (1987, Table 1): a tick in (row x, column y) indicates that the head of codon x may smoothly join to the tail of codon y, i.e., y may follow x in a string description.

	0^+	0^-	1^+	1^-	2
0^+	√		√		
0^-		√		√	√
1^+		√		√	√
1^-	√		√		
2		√		√	√

Table 2.2: The legal smooth joins between pairs of codons

Both open and closed curves can be described by codon strings. However, Richards and Hoffman (1987) point out that: "[b]ecause most objects have closed bounding contours, matching to closed codon sequences is of greater interest for shape recognition than representing open strings" (p. 702). In general, the number of legitimate open and closed codon strings of length N is much less than the number of possible combinations of codons without the constraints. For strings of length six, for example, the number of possible combinations is $5^6 = 15,625$, while there are actually just 610 legitimate open strings and 33 closed ones. Within the appendices of (Richards & Hoffman 1987), equations are provided for determining the number of open and closed codon strings of length N, denoted, respectively, by $\Sigma_o(N)$ and $\Sigma_c(N)$:

$$\begin{aligned} \Sigma_o(N) &= 3\Sigma_o(N-1) - \Sigma_o(N-2) && (N > 2) \\ \Sigma_c(N) &= 2^{N-1} + 1 && (N \geq 2) \end{aligned}$$

String transformations

Two transforms are provided for converting codon string descriptions, both of which operate on binary representations of codon strings. Each codon is assigned a two-digit binary number, as follows: $0^+, 0^- \mapsto 00$, $1^+ \mapsto 01$, $1^- \mapsto 10$, and $2 \mapsto 11$. Note that, although 0^+ and 0^- map to the same binary number, they can be distinguished in a string so long as the string contains at least one codon *not* of type 0. Given these mappings, the description of the curve on the left of Figure 2.6 is represented as 011110010010 (starting at the bottomleft codon).

The *mirror transform* converts the description of a curve into the description of its mirror image, and consists simply of reading the binary representation backwards. For the curve just cited we get 010010011110 which, when decoded, gives $1^+\,0^-\,1^-\,1^+\,2\,1^-$. This description is a rotational shift of, and therefore equivalent to, the original description, which is to be expected since the curve is symmetrical about a central vertical axis. The *figure-ground transform* converts the description of a curve under one orientation to the description of the curve under the other, alternative, orientation. The transform consists of rotating the binary representation by one bit to the right and reading the result backwards. The description of the curve on the left of Figure 2.6, 011110010010, when rotated by one bit, gives 001111001001. Reading the rotated binary string in reverse, we get 100100111100, which decodes to $1^-\,1^+\,0^-\,2\,2\,0^-$. This result accords with the annotated curve shown on the right of Figure 2.6, i.e., the figure-ground reversal of the original curve.

Rosin's extended set of codons

Hoffman and Richards' original set of codons has been extended by Rosin (1993), in the context of multi-scale representation and matching of curves. The five original codons are supplemented by a large number of additional codons, in order to facilitate the description of curves that contain straight-line segments and tangent discontinuities. For completeness, extra codons are also introduced to explicitly represent the ends of open curves.

Rosin's extended set of codons supports the description of a much wider range of curves than Hoffman and Richards' codons. However, with respect to discriminatory power and completeness, there are still two noteworthy deficiencies:

- Circular-arc segments are not distinguishable from other kinds of arcs. Rosin's motivation for incorporating straight-line segments, but not circular-arc segments, is principally that "[u]nder perspective transformation straight lines remain straight lines whereas circular arcs become ellipses. That is, straight lines are an important feature since they are stable over most viewpoints" (Rosin 1993, p. 290). Another reason he gives for not including circular arcs is that, if they were to be included, the extra codons required would make the representation "unnecessarily unwieldy".
- Although tangent discontinuities *are* incorporated in the extended representation, the method by which they are included (the joining of semi-cusp codons) means that cusps cannot be distinguished from angles.

As well as extending the set of available codons, thereby providing a greater scope, Rosin's scheme is geared towards describing a curve at a number of "natural" scales, in the form of a codon-tree. The use of a tree structure for hierarchical description resembles the strip-tree representation of (Ballard 1981). The purpose of a multi-scale analysis is to overcome the problems of noise and minor occlusion that are encountered when only one level of resolution is available. An important objective of the work presented in (Rosin 1993) is "to make codons a practical representation for applications in computer vision" (p. 287).

2.3.2 Extremum primitives

An interesting and somewhat different approach to shape description is given by the process-based descriptions and inference rules of Leyton (1988). Leyton takes the view that the shapes of certain objects can be meaningfully described in terms of the deformational processes, both internal and external, acting on the objects. As well as describing

the shape of an object at some snapshot in time, the theory provides grammar rules which are capable of accounting for the change in the shape of an object *over* time. Because of the nature of the theory, it is restricted in applicability to "natural forms", such as clouds, tumours, and embryos, rather than man-made objects, which tend to have straight edges and sharp corners.

Curvature extrema and processes

According to Leyton's theory of *symmetry-curvature duality*, each curvature extremum has an associated unique differential symmetry axis (under each of the symmetry analyses SAT, SLS, and "PISA"[5]) that terminates at the extremum. Along with the symmetry-curvature duality theorem, an *interaction principle* states that "[t]he symmetry axes of a perceptual organisation are interpreted as the principal directions along which processes are most likely to have acted." (p. 218). From these assertions, an important connection between curvature extrema and processes can be made. Specifically, each extremum of a shape implies a process that is acting along the unique symmetry axis associated with, and terminating at, the extremum. Figure 2.7 shows a smooth shape containing a number of positive and negative curvature extrema. The dashed lines indicate the process activity inferred by Leyton's theory.

There are four kinds of curvature extremum: positive maxima and minima, and negative maxima and minima. Within Leyton's theory, curves are described by strings composed of the following primitives: M^+ (positive maximum), m^+ (positive minimum), M^- (negative maximum), m^- (negative minimum), and 0 (representing a point of zero curvature). Each of the four kinds of curvature extremum is given a *semantic interpretation*, as listed in Table 2.3.

The representation of a curve in terms of these symbols is illustrated by a *process-diagram*, which is an annotation of the curve showing the curvature extremum points,

[5]Process-Inferring Symmetry Analysis (PISA) is defined by Leyton and shown by him to be the most appropriate analysis for his process-based approach.

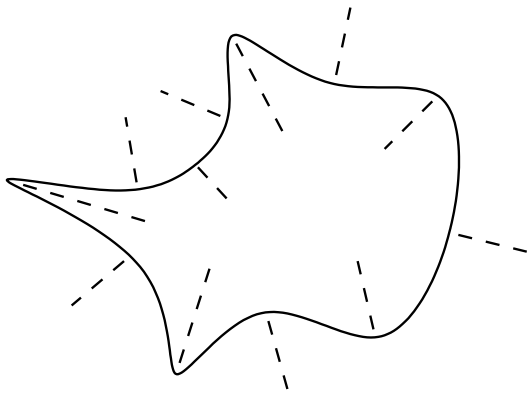


Figure 2.7: A shape and its process-inferring symmetry axes

Extremum type	*Semantic interpretation*
M^+	Protrusion
m^+	Squashing
M^-	Internal resistance
m^-	Indentation

Table 2.3: Leyton's semantic interpretations of curvature extrema

zero-curvature points, and the process traces implied by the curvature extrema. The convention we will adopt here is that curvature extrema are shown as small filled circles, zero-curvature points as small unfilled circles, and process traces as directed lines. The process-diagram for the shape shown in Figure 2.7 is given in Figure 2.8. Starting at the leftmost M^+, the string description of the shape is as follows:

$$M^+\,0\,m^-\,M^-\,m^-\,0\,M^+\,0\,m^-\,0\,M^+\,m^+\,M^+\,0\,m^-\,0\,M^+\,0\,m^-\,0$$

Inferring process history

Leyton derives a set of grammar rules that can be used to generate a process history which reflects how the shape of an object changes between two instants in time. The rules are

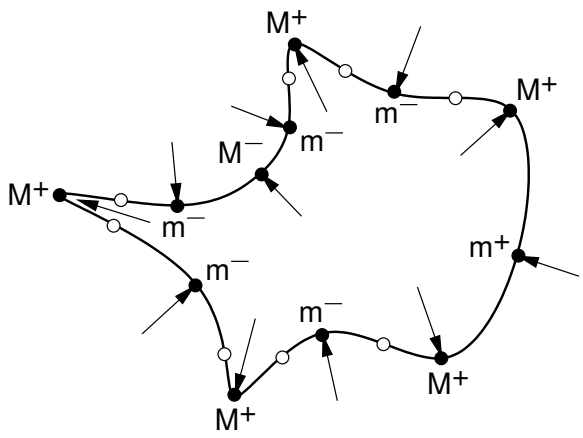


Figure 2.8: The process-diagram for the shape given in Figure 2.7

obtained by considering the possibilities that exist for the continuation and bifurcation of the four kinds of process. The two rules for continuation, and the four rules for bifurcation, are listed in Table 2.4, together with illustrative examples. Given the shape of an object at two developmental stages, the rules can be used to infer the process history between the two stages, by transforming the description of the earlier shape into the description of the later one. Of consideration is the order in which the rules can be applied, which it is not possible to ascertain given just the initial string description. Some processes, for example, may be more pronounced than others because they may have been acting on the object for a greater length of time. Leyton relates the time a process has been acting on an object to the likely amount of deformation. If process P starts acting on an object at an earlier time than process P', then the assumption made is that the effect on the shape of the object caused by P is greater than that caused by P':

> "**Size-is-Time Heuristic**. Size corresponds to time. That is, later processes have had a shorter time to develop than earlier ones and are thus smaller." (p. 232).

By considering different levels of *blurring*, the relative amounts of time that different processes have been acting can be established, e.g., if a shape is blurred a great deal, then

we should expect that only the very earliest processes will be detectable. In (Leyton 1989), the Size-is-Time heuristic, and the issue of process ordering, is examined in greater detail. Note that the idea of blurring a shape in order to obtain a coarser description is characteristic of the multi-resolution approach, where noise and fine detail is seen to impinge on higher-level model matching.

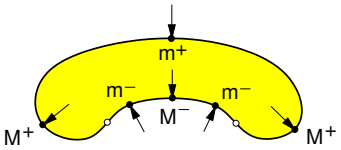
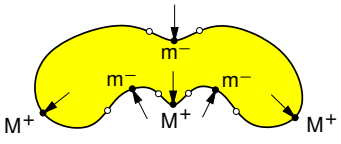
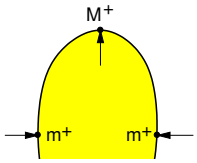
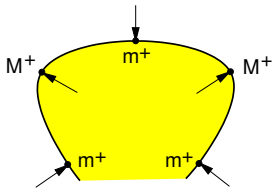
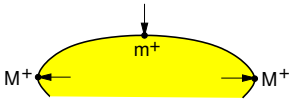
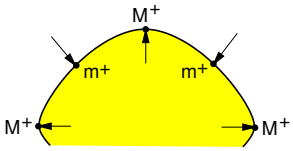
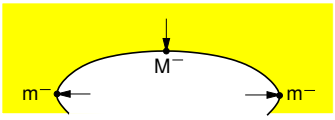
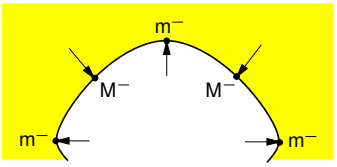
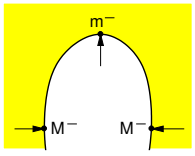
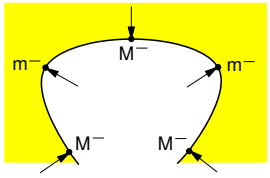
Continuation rules		
$Cm^+ : \; m^+ \rightarrow 0\, m^-\, 0$ $CM^- : \; M^- \rightarrow 0\, M^+\, 0$		
Bifurcation rules		
$BM^+ : \; M^+ \rightarrow M^+\, m^+\, M^+$		
$Bm^+ : \; m^+ \rightarrow m^+\, M^+\, m^+$		
$BM^- : \; M^- \rightarrow M^-\, m^-\, M^-$		
$Bm^- : \; m^- \rightarrow m^-\, M^-\, m^-$		

Table 2.4: Leyton's grammar rules for process continuation and bifurcation

Shape-space

The natural forms that fall within the scope of Leyton's theory have, as their shapes[6], smooth closed planar curves, i.e., curves whose salient points are maxima and minima of curvature. The set of all such curves corresponds to the *shape-space* of the theory. Note that, for mathematical reasons, each shape must have (i) more than three extrema[7], and (ii) an even number of extrema. Using the grammar rules, one can move through shape-space, i.e., from one shape description to another, in a systematic way. Leyton shows there are six meaningful stratifications of shape-space based on the grammar rules, where each "strata-system" is a set of parallel planes (Leyton 1988, Figure 17).

Leyton compares the six grammar rules for process continuation and bifurcation with a set of *codon-grammar* rules based on Hoffman and Richards' contour codons, involving the five original codons, together with their duals (figure-ground reversals). The codon grammar contains level-preserving and level-increasing rules. The level-preserving rules simply replace one codon with another. There are two kinds of level-increasing rules: the first kind replaces an extremum with a codon, while the second kind replaces a codon with two codons. The complete codon grammar consists of eighteen rules, and is therefore significantly larger in size than the process-grammar. After considering the operations of each grammar, Leyton concludes that: "the process-grammar has the considerable advantage of according with intuition concerning the structural relations between shapes... [codons] are essentially *cutting* and *pasting* operations... In contrast, under the process-grammar, any change from one structure to another is expressed purely *developmentally*; i.e. by *growth*." (Leyton 1988, p. 244).

Processes at discontinuities

Hayes & Leyton (1989) extend Leyton's process-based analysis, by considering the formation, via process activity, of first-order discontinuities on silhouette contours. The motiva-

[6]More precisely, the shape of their *silhouettes*.
[7]This constraint is given by the Four-Vertex theorem of differential geometry.

tion for the extension comes from earlier work by Hayes on an ontology for liquids (which forms part of his naive physics manifesto; see (Hayes 1985) for details). The analysis is restricted to the formation of sharp bends and cusps at curvature extrema, i.e., "kinks" (tangent discontinuities) occurring at other places on a contour are not handled by the theory. The presence of a kink corresponds to the presence of a spike on the curvature graph. Spikes that are "positive", i.e., go to $+\infty$, denote outward-pointing kinks, while "negative" spikes ($-\infty$) denote inward-pointing ones. Concerning notation, kinks are represented by attaching a subscripted value, indicating spike parity, to each of the original four extremum-primitive symbols. The kink symbols, then, take the form X^a_b, where X^a is the *curvature component*, a is the *curvature parity* and b is the *spike parity*. If a and b are of the same sign, then the kink is a sharp bend, otherwise it is a cusp, i.e., a cusp is a kink whose curvature graph has a spike of opposite parity to the curvature of the adjoining curve segments. To summarise, the new scheme consists of thirteen symbols (the original five symbols, together with eight new kink symbols introduced by adding spike parity):

$$\overbrace{\mathsf{M}^+\ \mathsf{m}^+\ \mathsf{M}^-\ \mathsf{m}^-\ \mathsf{0}}^{\text{original symbols}}\quad\overbrace{\mathsf{M}^+_+\ \mathsf{m}^+_+\ \mathsf{M}^-_-\ \mathsf{m}^-_-}^{\text{sharp bends}}\quad\overbrace{\mathsf{M}^+_-\ \mathsf{m}^+_-\ \mathsf{M}^-_+\ \mathsf{m}^-_+}^{\text{cusps}}$$

Kinks are seen as being formed by the action of processes. In order to account for the formation of kinks, only one rule needs to be added to the six existing grammar rules. The *kink introduction* rule, K, consists of four transformations, each of which corresponds to the formation of a kink by one of the four kinds of process. The transformations are listed, together with descriptions, in Table 2.5. Examples for each of the four kinds of introductory kinks are given in Figure 2.9 (the figure lies to the left of the traversal directions specified by the arrowheads). The four remaining kink types are essentially secondary types, in that their formation must follow the formation of one of the introductory types, and requires the application and/or *reverse*-application of one or more of the six original grammar rules.

Transformation	*Description*
$M^+ \rightarrow M^+_+$	A protrusion terminates as a sharp bend
$m^+ \rightarrow m^+_-$	A squashing leads to a cusp-like indentation
$M^- \rightarrow M^-_+$	An internal resistance leads to a cusp-like protrusion
$m^- \rightarrow m^-_-$	An indentation terminates as a sharp bend

Table 2.5: The four kink-introduction transformations

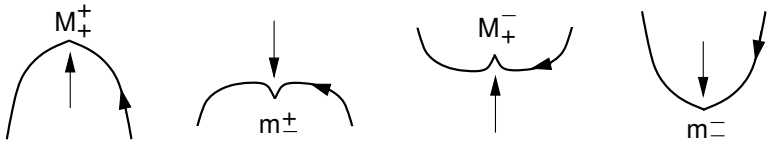


Figure 2.9: The four primary sharp bends and cusps

2.3.3 Structural codings

Cinque & Lombardi (1995) describe a multi-resolution approach to shape description and model matching, based on a process of *heat diffusion*. In a similar way to Rosin (1993), given an image of some entity's silhouette, a smoothing operator is first applied to obtain further images of the entity at different resolution levels. For each smoothed image, a process of diffusion is simulated which yields a description of the silhouette contour. The process consists of the following steps:

1. The contour of the silhouette is heated to a chosen energy value.

2. At each time step a certain amount of heat dissipation is simulated, from the contour in towards the middle of the silhouette.

3. After a certain number of time steps, the diffusion process is stopped.

4. The *temperature values* of the contour pixels are thresholded into discrete ranges, and the contour is segmented and labelled using the symbols that identify each range.

The example given in the paper uses five discrete ranges for temperature values, although a greater or lesser number of ranges could be used. Each of the five ranges is given a symbol, and contours are described by strings of symbols. The symbols (primitives) used are: W ("very concave"), C ("concave"), S ("straight"), X ("convex"), and Y ("very convex"). Two quantitative attributes are associated with each primitive: L_C (the total length of the segment) and S_C (the skewness of the segment). In the paper, the silhouette of a fish is represented at six levels of resolution; the coarsest description is SXCXCSX, while the finest is SCYCSCYCYCYCYCSY. From the six string descriptions, a tree representation is constructed that can be used to match against existing models.

Clearly, the described approach has a strong quantitative aspect, although shape contours *are* described by strings of symbols after a certain amount of numeric processing has been carried out. The discrete ranges chosen are essentially arbitrary and so each qualitative symbol has a rather ill-defined meaning, e.g., it is not clear what "very concave" and "very convex" actually mean.

2.4 Comparisons

In this chapter, we have looked at a number of different schemes for representing two-dimensional shape. We broadly classified the systems according to whether they were region-based or boundary-based. Here we compare the various systems by revisiting the aspects of a shape representation that we considered most relevant to qualitative description, listed at the beginning of the chapter:

- *Computability*

  All of the systems we have considered obtain their descriptions from image data, with the possible exception of Cohn's approach, which was developed from a logical perspective, and one could imagine that descriptions need not originate from quantitative data. Rosin has applied his theory of codons to actual image data with a certain degree of success. It is worth noting that, of the extended set of codons, those

containing straight lines, angles, or cusps are the most difficult to extract reliably. Leyton's extremum primitives, the essential constituents of his process-based theory, are similar to the codons, in that proper detection of the rate of change of curvature is of importance.

- *Scope*

  The range of shapes that can be represented by the region-based systems consists of closed planar curves; open curves cannot be described, since finite open curves do not bound regions. Of the region-based approaches, the symmetry-based representations are capable of describing rectilinear shapes, although the symmetry axes associated with such shapes are not as intuitive as those associated with more "natural" shapes. It is *possible* to describe polygons using Cohn's scheme, but the predicates required are not straightforward. A further limitation, specific to Cohn's scheme, is that all convex shapes have the same (empty) description, and are therefore indistinguishable. The range of shapes that can be represented by the boundary-based systems include both closed *and* open curves, although only Rosin explicitly deals with the representation of the ends of curves. Hoffman & Richards' set of contour codons and Leyton's initial set of extremum primitives are limited to describing smooth, continuously differentiable, closed curves. Hayes and Leyton's cusp extension increases the scope of the extremum primitives by incorporating certain sharp bends and cusps, but there still remain some discontinuity variants that cannot be represented by their scheme.

- *Primitives*

  Under the region-based systems that use symmetry axes, there are two basic constituents of a shape description: the symmetry axis itself, and the radius function. The primitives of Cohn's scheme are the connection relations EC and DC, the $Conv(x)$ operator, and the various predicates that support increased shape discrimination. Of the boundary-based schemes, there are five basic contour codons and four extremum

primitives. Each extremum primitive denotes a particular curvature stationary point. Codons are curve segments bounded by curvature minima that contain zero, one, or two points of zero curvature. Rosin presents an extended set of sixty-three codons. The codons and extremum primitives can be regarded as general descriptors that are not specific to any particular domain.

- *Discriminatory power*

  Potentially, the symmetry-based systems have the greatest amount of discriminatory power, because shapes can be re-created from SAT descriptions if enough quantitative information is kept, i.e., if the exact values of the radius function are preserved, rather than collectively being characterised. Cohn's scheme is able to distinguish between two shapes so long as they differ with respect to their concavities; in contrast, two different convex shapes cannot be distinguished within Cohn's scheme. The basic set of codon contours and extremum primitives have identical discriminatory power, and are able to distinguish two curves that differ with respect to the number and/or type of their curvature extrema. Note that Leyton's "0" primitive is essentially redundant, as its presence can be inferred from the presence of particular curvature extrema, i.e., mathematically, a point of zero curvature is *necessarily* present between M^+ and m^- (for smooth curves).

- *Granularity*

  All of the schemes we have looked at, to a certain degree at least, support the description of a shape at a number of granularity levels. By granularity levels, here, we mean different levels of resolution. We can smooth a shape by a suitable process before deriving a description of the shape; in this way, the issue of granularity is partially separated from the representation itself. Given that, by smoothing a shape, we can derive a number of different descriptions for it, the degree to which a representation supports different levels of granularity is dependent on the ways in which the descriptive elements at different levels can be related. The schemes of Cohn, Rosin,

and Cinque & Lombardi could be said to have the notion of description at different levels of granularity "built in".

- *Incomplete information*

  The region-based systems that generate symmetry axes score poorly here, because noise and occlusion significantly affect the descriptions that are derived by these systems. This is because they are essentially techniques that operate on the shape as a whole, rather than localised portions. The boundary-based schemes, on the other hand, derive their descriptions by curve traversal and are thus less susceptible to negative effects caused by incomplete information.

In summary, the schemes we have looked at differ with respect to the primitives they use and their scopes. The boundary-based schemes of Hoffman & Richards and Leyton attach importance to the curvature extrema of a shape and neither is able to adequately represent all open and closed planar curves, i.e., curves containing straight-line segments, circular-arc segments, tangent discontinuities etc. The two extensions discussed go some way to addressing the deficiency, but neither results in "complete" scope, i.e., neither extension permits a truly general class of open and closed curves to be described. This is understandable, given the motivation behind each of the schemes we have examined. In no instance has the primary goal been to ensure that "all" planar curves can be adequately represented. An important objective of the general theory that we develop in this thesis is to derive a set of shape descriptors that allows a wide range of open and closed curves to be adequately described.

Chapter 3

Qualitative outline theory

In this chapter, we present a boundary-based qualitative scheme for describing planar outlines, consisting of seven curvature-type primitives.[1] *We show how an outline is described by a string of curvature-type symbols and provide the ordering and closure constraints that ensure a given string is instantiable as an outline. We conclude by comparing the scheme with the boundary-based schemes of Chapter 2.*

3.1 Curvature types

The scheme consists of the seven qualitative *curvature types* listed in Table 3.1. The curvature types can be grouped in different ways. The most important distinction, perhaps, being that between the *linelike* types, that contribute to the length of an outline ($\supset$, $\subset$, /), and the *pointlike* types, which do not ($>$, $<$, $\succ$, $\prec$). Another important grouping is *outward* ($\supset$, $>$, $\succ$) versus *inward* ($\subset$, $<$, $\prec$), with / belonging to neither category.

Included as specific types are cusps, which we consider to be qualitatively different from angles. Note that it seems common in the literature to refer to *any* tangent discontinuity as a "cusp", whereas we choose to distinguish between points where the tangents meet from the same direction and points where they do not, referring to the former kind of point

[1] The material in this chapter is based on work published in a paper of the same name (Galton & Meathrel 1999).

as a "cusp" and the latter an "angle". We could have chosen, instead, to combine cusps and angles by having only inward- and outward-pointing "kinks". There are two reasons for not doing so. The principal reason is that cusps are important real-world phenomena, e.g., they are formed when a spherical object lies on a flat surface, when a container is filled with a liquid, and in other common situations. The second reason is that it is not the case that angles and cusps can be treated the same in all situations. An outward-pointing cusp, for example, cannot always be substituted for an outward-pointing angle. In particular, a cusp cannot exist between two straight-line segments. The converse, however, is true: a cusp can always be replaced by an angle of the same orientation.

Notice that there is no separate type for a point of inflection; it was felt unnecessary to include because the presence of such points can always be inferred. In Chapter 8, where we show how the curvature types can be specified in terms of simpler components, the implications of omitting a point-of-inflection type are addressed.

Symbol	*Meaning*
$\supset$	Convex curve segment
$\subset$	Concave curve segment
/	Straight-line segment
$>$	Outward-pointing angle
$<$	Inward-pointing angle
$\succ$	Outward-pointing cusp
$\prec$	Inward-pointing cusp

Table 3.1: The seven qualitative curvature types

The shape illustrated in Figure 3.1 consists of two convex curve segments, together with one each of the six remaining qualitative curvature types, as indicated. The scope of the curvature-type representation is restricted to simple closed planar curves that may include points of tangent discontinuity. Certain kinds of closed curves are excluded. In particular, fractal curves such as the Koch snowflake are disallowed, as are curves that contain points of self-intersection. More formally, define a *uniform curve segment* to be an

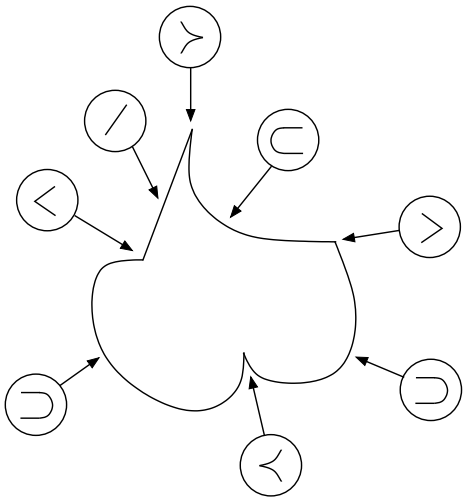


Figure 3.1: A shape annotated with its constituent qualitative curvature types

open segment on which the tangent bearing has derivatives of all orders, and the sign of each derivative remains constant ($+$, 0, or $-$) throughout the segment. A *piecewise uniform curve* is a curve which can be divided into a finite sequence of uniform curve segments and their meeting points. The scope of the curvature-type representation is restricted to piecewise uniform curves that are closed and contain no points of self-intersection.

3.2 Describing outlines

An outline is described by a string of curvature-type symbols that reflects the features encountered during a traversal of the outline. Curvature-type strings that represent outlines are interpreted as cyclic, i.e., the feature represented by the last symbol in the string is followed, round the outline, by the feature represented by the first. In this sense, each curvature-type string may be interpreted as a *ring*. The mapping between strings and rings is many-to-one, that is, a number of different strings may define the same ring. Given an outline and a direction of traversal, different string descriptions are obtained depending on the point at which the traversal is started. However, the different strings

may be considered equivalent because they all correspond to the same ring. The outline in Figure 3.1, for example, when traversed in a clockwise direction, starting from /, yields the string $/ \succ \subset > \supset \prec \supset <$. A different starting point, for example $\prec$, yields the string $\prec \supset < / \succ \subset > \supset$. Since the two strings are cyclically-shifted versions of each other, they must correspond to the same ring and, therefore, represent the same outline type.

3.2.1 Traversal direction and qualitative symmetry

The direction in which an outline is traversed is significant, because the ring descriptions that correspond to the two possible directions will, in general, be different. Note that a reversal of direction does not affect the presence of the particular curvature types that constitute an outline, rather, a change in direction results in a concomitant change in the *ordering* of the curvature types. Consider again the outline in Figure 3.1. If we start at /, as before, but traverse in an *anticlockwise* direction, we get $/ < \supset \prec \supset > \subset \succ$. This string cannot be cyclically shifted so that it matches the string obtained when traversing in a clockwise direction. Hence, it corresponds to a different ring. To summarise, there are two rings associated with an outline, one for each direction of curve traversal. We have referred to the two directions as "clockwise" and "anticlockwise". During clockwise traversal, the region bounded by the outline (the *figure*) lies to the right as we move along the curve. Conversely, for anticlockwise traversal, the figure lies to the left. Note that the relative placement of figure and ground must not change during traversal, i.e., the figure must either be always to the right or always to the left. This is Hoffman & Singh's (1997, p. 45) *Principle of consistent orientation*.

Two outlines that are mirror images of one another will have descriptions, under a particular direction of traversal, that are reversals of one another. This is illustrated by outlines (a) and (b) in Figure 3.2. Assuming clockwise traversal, outline (a) is described by $\supset / >$ and outline (b) by $\supset > /$ (a reversed permutation of (a)'s description). Note that we can only distinguish between (a) and (b) if we are consistent with the traversal direction, i.e., we get the same ring for both outlines if we traverse one clockwise and the other

anticlockwise. If we wished, we could decide that an outline is equivalent to its mirror image, in which case outlines (a) and (b) would be considered identical.

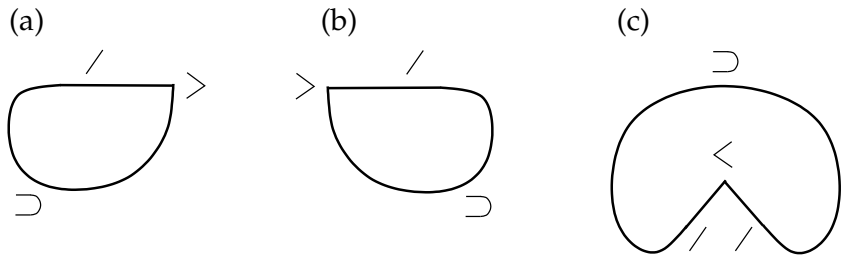


Figure 3.2: Outlines illustrating qualitative symmetry

As a consequence of the qualitative nature of the curvature types, two outlines which are not precise mirror images of one another may still be so regarded in a *qualitative* sense. In particular, if outline X has a description that is a reversed permutation of outline Y (under the same traversal direction), then X may be considered a qualitative mirror-image of Y, and vice versa. What this means is that X and Y may be smoothly transformed, in a way that preserves the existence and ordering of their constituent curvature types, so that they *are* precise mirror images. With respect to a single outline, we can define the notion of *qualitative symmetry*: an outline may be considered qualitatively symmetrical *iff* the ring description obtained via clockwise traversal is the same as that obtained via anticlockwise traversal, in which case the outline can be smoothly transformed (as previously described) so that it has (at least) one axis of symmetry. Outline (c) in Figure 3.2, for example, has one axis of symmetry and is described by $\supset / < /$. All outlines with the description $\supset / < /$ are qualitatively symmetrical because a reversal of $\supset / < /$ yields $/ < / \supset$, which corresponds to the same ring as that given by $\supset / < /$. Every outline that is described by a string of length less than three is necessarily qualitatively symmetrical. Outlines with odd-length descriptions of length three or greater, however, *cannot* be qualitatively symmetrical. Possessing qualitative symmetry is a necessary condition for an outline to *actually have* an axis of symmetry (cf. (Cohn 1995, Section 5)).

3.2.2 Canonical form of a description

Given the set of strings that describe an outline, we can choose one of the string descriptions to be the canonical one. We do this by defining a lexicographic ordering on curvature-type strings and then, for a given outline, choosing that string which occurs earliest in the sequence. Although any ordering of the types would suffice, we will order them as they are listed in Table 3.1, i.e., $\supset$, $\subset$, $/$, $>$, $<$, $\succ$, $\prec$. Since an outline must contain at least one linelike segment, every canonical string must begin with either $\supset$, $\subset$, or $/$. The outline in Figure 3.1, when traversed clockwise, has the canonical description $\supset < / \succ \subset > \supset \prec$. Any candidate description of an outline can be transformed into the canonical description by a cyclic shift (together with a reversal if mirror-image outlines are considered equivalent). Note that, although an outline consisting of N curvature types has N candidate string descriptions (as there are N different places the traversal could start at), the number of *distinct* candidates may be less than N. A triangle, for example, has six candidate descriptions but only two different strings, namely $/ > / > / >$ (the canonical one) and $> / > / > /$. A canonical description, then, does *not* define a unique starting point for traversing an outline, rather, it provides a single string with which to represent an outline.

3.2.3 Outline tracing

One way of obtaining a description of an outline is to *trace* it, noting the change in the direction of the tangent to the curve throughout the traversal. Each of the curvature types may be identified by the effect that it has on the change in tangent bearing, e.g., we know we have a straight-line segment if the tangent bearing remains unchanged over a stretch of curve. Given that we have some method of tracing round an outline whilst keeping track of the change in tangent bearing (with respect to some arbitrary fixed reference point), we can determine the curvature types that constitute an outline's description, as follows:

- Begin a clockwise traversal of the outline, starting at any point on the outline.
- If the tangent bearing changes continuously, then we have one of $\supset$, $\subset$, or $/$, de-

pending on whether the bearing is (in a clockwise sense) increasing, decreasing, or constant.

- A discontinuous change in the tangent bearing signifies the presence of a pointlike type, i.e., one of $>$, $<$, $\succ$, or $\prec$. If the clockwise change is less than 180°, then we have $>$, if it is more than 180°, we have $<$. A change of exactly 180° indicates the presence of a cusp. In order to ascertain whether the cusp is outward-pointing or inward-pointing, we need to check where the section of curve after the cusp is, in relation to the section of curve before the cusp. If the former is to the left as seen from the latter, then we have $\prec$, if to the right, we have $\succ$.

- If, on returning to the starting point, the last curvature type encountered is the same as the first, omit it.

The above procedure is for clockwise traversal. The procedure can be modified for anticlockwise traversal by simply replacing occurrences of "clockwise" with "anticlockwise", and swapping "left" with "right".

3.2.4 Ordering and closure constraints

Up until now, our attention has been focussed on the relationship between an outline and its curvature-type description. If we are given a string, we would like to be able to determine whether or not the string is valid, i.e., whether there actually exists an outline that has the string as one of its candidate descriptions. We address the process of *generating* valid strings, which is also of importance, in the next section. Clearly, a string composed of an arbitrary sequence of curvature-type symbols is not likely to be legal, because it may not include enough symbols to ensure closure, or it may contain subsequences that are unrealisable. There are two kinds of string constraints that need to be considered: *ordering constraints*, that determine how the types may be "put together" to form a string, and *closure constraints*, that specify which types are required in a string in order for it to be realisable as an outline. A cyclically permutable string of curvature-type symbols is subject

to the following ordering constraints:

- It must not contain two consecutive occurrences of the same curvature-type symbol.
- It must contain no two consecutive pointlike types.
- Any occurrence of either $\prec$ or $\succ$ must be adjacent (on at least one side) to an occurrence of $\supset$ or $\subset$ respectively.

To ensure that a string is realisable as an outline, the following closure condition also needs to be met (note that this is an unproven claim):

- To ensure closure, a string must either (i) contain at least one convex curve segment ($\supset$), or (ii) contain at least three outward-pointing kinks (of which there are two kinds: $>$ and $\succ$).

A string is realisable as an outline, and referred to as *valid*, if it adheres to all four constraints listed above. The simplest valid string is $\supset$, which represents convex curvilinear closed curves without tangent discontinuities, e.g., circles, ellipses, etc. Valid strings of length $N \leq 5$ necessarily contain at least one $\supset$, because the presence of three outward-pointing kinks requires the presence of at least three linelike types (a consequence of the second ordering constraint). There are five outline types of length two, and sixteen outline types of length three (or just eight types if reflections are considered equivalent). Exemplar shapes for each of these types are given in (Galton 2000).

3.3 Outline classes

The seven qualitative curvature types, together with the ordering and closure constraints, may be used to define a language containing all of the valid curvature-type strings. Under a particular traversal convention, every outline which falls within the scope of the curvature-type theory has a single ring associated with it, which corresponds to a number of curvature-type strings, each of which is constrained by the ordering and closure

conditions. Let L_f denote the language consisting of the full set of valid strings. The alphabet of L_f, then, is given by the complete set of curvature types: $\{\supset, \subset, /, >, <, \succ, \prec\}$. We use the word "full" when describing L_f because, rather than containing just a *single* string for each possible ring, L_f contains all of the distinct strings that correspond to each possible ring. So, for the ring corresponding to the outline in Figure 3.1, which consists of eight curvature-type symbols, *all* of the eight distinct strings that represent the ring are contained within L_f. Since we can specify, for each ring, a single canonical string description (see Section 3.2.2), a more parsimonious set is that which contains only the canonical strings. Let such a set be denoted by L_c. The languages L_f and L_c, which we may refer to as *outline languages*, can be used to determine whether or not a given curvature-type string is valid. Specifically, a string X is valid if it is contained within L_f, or its canonical form is contained within L_c. Grammars that generate the strings of outline languages can be used to verify that a given string is in a particular outline language; such *outline grammars* can also be used for constructing shapes of a particular outline type.

In Appendix A, a grammar for generating valid string descriptions is presented[2] and its construction is explained. The grammar, which we'll refer to here as G, is regular and constructed in such a way that it generates all of the canonical string descriptions. However, as well as the canonical descriptions, other valid strings are also generated. This is a side-effect of limiting the complexity of the grammar. A grammar that generates all and *only* the canonical strings is necessarily non-regular (a proof is provided in Section A.1.1). The language generated by G may be considered a "compromise" between the outline languages L_f and L_c, since $L(G) \subset L_f$ and $L(G) \supset L_c$.

3.3.1 Sublanguages and subgrammars

By considering subsets of the seven curvature types, we can define sublanguages of the full set of outlines and derive, from the full grammar given in Appendix A, the corresponding outline grammars that generate them. Some important classes of outlines, and the subsets

[2]Consisting of 95 rules, with 35 non-terminal symbols.

they are generated from, are listed in Table 3.2.

Subset	*Outline class*
$\{\supset, \subset, >, <, \succ, \prec\}$	Curvilinear outlines – i.e., no straight-line segments
$\{\supset, \subset, /\}$	Smooth outlines: no cusps or angles
$\{\supset, /, >\}$	Convex outlines
$\{/, >, <\}$	Polygons: no curved segments
$\{\supset, /\}$	Convex smooth outlines
$\{\supset, >\}$	Convex curvilinear outlines
$\{/, >\}$	Convex polygons

Table 3.2: A selection of valid curvature-type subsets

As an example, consider the convex figures, which are generated by a grammar based on a set of three curvature types: $\{\supset, /, >\}$. All convex figures have, as their descriptions, curvature-type strings containing some combination of convex curve segments, straight-line segments, and outward-pointing angles; none of the other four types are permitted, since they each necessitate the presence of a concavity. A regular grammar that generates the language of convex figures can be derived systematically from G, by following the procedure given in Section A.2. The eighteen rules extracted from G that generate the convex figures are listed in Table 3.3.

The rules of the grammar essentially reflect a partitioning of the set of convex figures into (1) the set of convex figures containing a convex curve segment (the descriptions of which are generated by the first two rules of line one together with the middle block of rules) and (2) the set of convex polygons (generated by the last rule on line one together with the last block of rules).

Not all curvature-type subsets are valid. Consider, for example, the subset $\{/, <, \succ\}$, which does not define a class of outlines because, even though it contains an outward-pointing kink (to "ensure closure"), a concave curve segment *must* adjoin an outward-pointing cusp on at least one side. Since the subset doesn't contain $\subset$, none of the curves that are generated from the subset can be closed. It turns out that, of the 127 possible

$$
\begin{array}{lcl}
S & \rightarrow & \supset \mid \supset A_{\supset} \mid /F_{/} \\
 & & \\
A_{\supset} & \rightarrow & /A_{/} \mid > X \\
A_{/} & \rightarrow & \supset A_{\supset} \mid \Lambda \\
X & \rightarrow & \supset A_{\supset} \mid /A_{/} \mid \Lambda \\
 & & \\
F_{/} & \rightarrow & > F_{>} \\
F_{>} & \rightarrow & /G_{/} \\
G_{/} & \rightarrow & > G_{>} \\
G_{>} & \rightarrow & /H_{/} \\
H_{/} & \rightarrow & > H_{>} \\
H_{>} & \rightarrow & /I_{/} \mid \Lambda \\
I_{/} & \rightarrow & > H_{>}
\end{array}
$$

Table 3.3: A regular grammar that generates the convex figures

curvature-type subsets (excluding the empty set), only 62 are valid (in the sense that they define a class of outlines).[3] The valid subsets, along with the rules used to determine them, are listed in Section A.2.

3.4 Quantitative considerations

A consequence of the qualitative nature of the curvature types is that two outlines that look markedly different may nonetheless have the same curvature-type description. An example of this is shown in Figure 3.3, where five different exemplars are given of the description $\supset \prec \supset \prec$. It would not be true to say, perhaps, that the differences between these outlines are purely quantitative, but they are quantitative inasmuch as they have to do with the relative lengths and curvatures of the linelike segments. A partial solution to this deficiency would be to incorporate "semi-qualitative" information into string descriptions. As a first attempt at increasing discriminatory power, we might consider including the following kinds of information:

[3] It is interesting to note that only a small number of these have well-recognised names.

- *The relative orientations of angles and cusps*

  Each pointlike symbol could be annotated with an index such as "U", "D", "L", or "R", depending on whether the angle or cusp is pointing upwards, downwards, to the left, or to the right. The five outlines in Figure 3.3, for example, are all described by $\supset\prec\supset\prec$. By including orientation information as subscripted symbols, the first two outlines are represented as $\supset\prec_D\supset\prec_D$, and the remaining outlines as $\supset\prec_R\supset\prec_L$, $\supset\prec_D\supset\prec_U$, and $\supset\prec_R\supset\prec_U$.

- *The relative lengths of the linelike segments*

  Each of $\supset$, $\subset$, and / could be annotated with a symbol indicating the length of the segment, e.g., "S", "M", and "L", according to whether the segment is short, of medium length, or long.[4]

- *The relative positions of the pointline types*

  For this extension, a curvature-type string would most likely be accompanied by some additional data structure encoding the position of each pointlike element relative to each of the others. Relevant positional schemes include those of Latecki & Röhrig (1993) and Schlieder (1996).

The extensions listed are semi-quantitative inasmuch as they depend on the underlying quantitative information that defines an outline. Because they introduce symbolic information that is non-numeric, each could be considered as essentially a *qualitative* extension. A significant side-effect of introducing such additional information is that the closure constraints would need to be modified accordingly. Different extensions are likely to interact with one another, e.g., fixing the orientations of the pointlike elements could impose constraints on the possible relative lengths of the linelike ones. The detailed working out of these constraints is likely to be problematic.

Even if the above kinds of information are adequately incorporated into the description of an outline, there will still be outlines that, while looking very different from one another,

[4]Note that an annotation scheme such as this, however, is subject to the criticism we levelled earlier at Cinque and Lombardi's "very concave/convex" symbols (see end of Section 2.3.3).

still share the same curvature-type description. This is illustrated by the two outlines in Figure 3.4, which do not look at all similar; it is not clear how, in this case, the two outlines can be distinguished using purely local information.

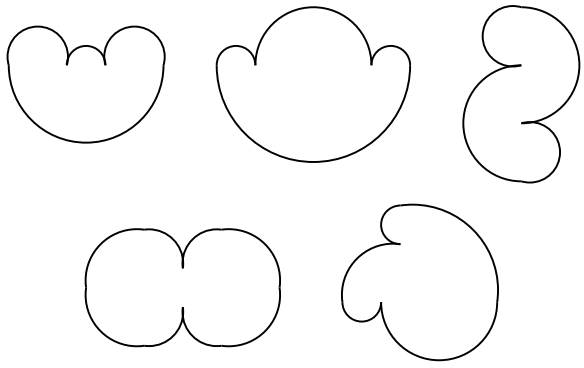


Figure 3.3: Five distinct exemplars of the outline type $\supset \prec \supset \prec$

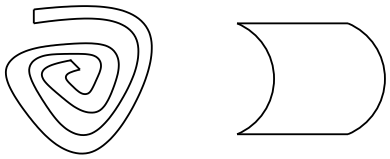


Figure 3.4: Two dissimilar-looking outlines described by $\supset > / > \subset > / >$

3.5 Summary and comparisons

In this chapter, we have presented a boundary-based representation for planar shapes, consisting of seven qualitative curvature types. Throughout the remainder of this thesis we will refer to the representation as "QOT". We have shown how outlines are described by strings, and provided the ordering and closure constraints that are required to determine string validity. Essentially, QOT provides a formal language for describing outlines. A regular grammar that generates all of the canonical string descriptions in the language is given in Appendix A.

We can compare QOT with the two boundary-based schemes of Leyton and Hoffman & Richards. The first important difference relates to the scope of the schemes. QOT encompasses a larger class of shapes, by providing distinct primitives for straight-line segments and tangent discontinuities. In contrast, the other two schemes are essentially limited to the description of smooth curves given by the outline class $\{\supset, \subset\}$. This difference reflects the design goals of the schemes; an important concern of QOT is to provide a set of types that is complete, in the sense that a wide range of outlines can be represented. With the other schemes, recognition of object silhouettes is the goal, and the most salient points on an outline are considered to be curvature extrema. Representing a polygonal outline in QOT is straightforward, since we have distinct types for a straight line and outward/inward pointing angles. Although it would perhaps not be correct to claim that polygons cannot be represented by the other schemes, only an approximation using maxima and minima is possible. The extended set of extremum primitives, that includes "cusps", *does* increase the scope of Leyton's representation, but only certain kinds of cusps are represented, i.e., those that may be formed at curvature extrema. Asymmetrical cusps are not included in the representation.

The second important difference is the discriminatory power of the schemes, i.e., what kinds of curves can be distinguished by QOT but not by the other two schemes, and vice versa. Clearly, those outlines that fall outside the scope of the representations of Leyton and Hoffman & Richards cannot be distinguished from one another in those schemes. In particular, outlines that are not curvilinear and which contain angles and cusps can only be distinguished by QOT. However, even though the other two schemes are restricted to outlines of the class $\{\supset, \subset\}$, by identifying points of curvature maxima and minima they are able to make finer discriminations within the class than QOT can. Two curve segments that differ only as to the number of curvature extrema they contain cannot be distinguished by QOT, whereas under either of the other schemes they are distinguishable. We can see, then, that there is a tradeoff between scope and detail. The scope of QOT is large, in comparison with the scope of the two other schemes, but by having a large scope

and only a small set of primitives, the amount of detail that can be represented by QOT is limited. Note that, interestingly, both the scope and discriminatory power of Leyton's extremum primitives and Hoffman & Richards' contour codons are identical, even though the primitives used by each scheme are different. Essentially, both schemes record the number and ordering of the curvature extrema that make up an outline. For Leyton, the extremum points themselves are important, for Hoffman & Richards, segments of curve containing extrema are attributed greater salience. We have considered how, by adding extra semi-qualitative information, QOT descriptions can possess greater discriminatory power. However, the extensions we discussed do not increase the discriminatory power of QOT with respect to curvature extrema.

Even though there are important differences between the boundary-based schemes we have been considering, each scheme possesses primitives that are based on variations in curvature. This underlying similarity suggests that an analysis of qualitative curvature variation may yield a theory that provides a unifying account of boundary-based schemes. In the next four chapters we undertake such an analysis, and develop a theory of the boundary-based approach to describing shape, which we call *qualitative boundary theory* (QBT). After this, in Chapter 8, we apply QBT by formally specifying the primitives of each of the boundary-based schemes we have been discussing, highlighting some interesting and important differences that underlie their primitives.

Chapter 4

The atomic tokens

In this chapter we derive an unbounded hierarchy of atomic tokens by considering tangent bearing and its successive derivatives. We show how a curve may be described by a string of atomic tokens at a particular level in the hierarchy and how the relevant syntactic constraints can be visually encoded in a graph.

4.1 Deriving the hierarchy of atomic tokens

We begin by considering the curve property of tangent bearing. At each point on a curve, the tangent to the curve at the point is either defined or undefined. By tangent bearing, we mean the angle that a tangent makes with respect to some fixed direction, e.g., "North". Given a curve, we can plot tangent bearing, b, against arc length, s. The rate at which the tangent bearing changes along a curve gives us curvature, i.e., curvature is the first derivative of tangent bearing. We can plot a graph of curvature, c, against arc length, s. If we want, we can obtain a further plot by considering the rate of change of curvature, c', which is the *second* derivative of tangent bearing. We need not stop doing this, there being an infinite number of associated plots for each curve. Our representation is based on a discretisation of b and its successive derivatives (c, c', c'', etc). For tangent bearing we use the quantity space $\{D, U\}$, because we're only interested in whether b is defined

(D) or undefined (U). Curve points where the tangent bearing is undefined are perceived as angles or cusps. Note that, by not distinguishing between different values of defined tangent bearing, we ensure that the resulting representation is invariant with respect to translation, rotation, and uniform scaling transformations. For each of the *derivatives* of tangent bearing, we use the quantity space $\{+, 0, -, U\}$, since a derivative may have a value that is positive, zero, negative, or undefined.

4.1.1 Curve states

Associated with each point P on a curve is a sequence of qualitative values, representing b, c, c', etc. The *complete curve state* at P corresponds to an infinite sequence of values. We write $\partial^k x$ to denote the kth component of the curve state x. Not all component sequences give rise to valid curve states. If a component has a value other than U, then the value of the next component is unconstrained. If, however, a component *does* have the value U, then all subsequent components must also be U. More formally, a curve state x must satisfy the following constraint:

$$(\forall k)(\, \partial^k x = U \;\Rightarrow\; \partial^{k+1} x = U \,)$$

A *partial curve state* at P is any initial n-tuple of the complete curve state. So if $x = \langle D, + \rangle$, for example, then $\partial^1 x = D$, $\partial^2 x = +$, and x is a partial curve state that is assigned to points on a curve where the tangent bearing is defined and the curvature is positive.

We refer to a partial curve state with k components as a level-k state. Given the complete set of states for some level k, it is straightforward to generate the set of states for level $k{+}1$. Each state x, at level k, generates a set of level-$k{+}1$ states, $\mathcal{S}$, as follows (where $y = \partial^k x$):

$$\mathcal{S} \;=\; \begin{cases} \{\, \langle \ldots, y, U \rangle \,\} & \text{if } y = U \\ \{\, \langle \ldots, y, + \rangle,\, \langle \ldots, y, 0 \rangle,\, \langle \ldots, y, - \rangle,\, \langle \ldots, y, U \rangle \,\} & \text{otherwise} \end{cases}$$

There are two partial curve states with one component: $\langle D \rangle$ and $\langle U \rangle$. From these two states we can generate the five states of level 2 and then, from those, the fourteen states of

level 3:

$$
\begin{array}{lcl}
\langle D \rangle & : & \langle D,+\rangle,\ \langle D,0\rangle,\ \langle D,-\rangle,\ \langle D,U\rangle \\
\langle U \rangle & : & \langle U,U\rangle \\
 & & \\
\langle D,+\rangle & : & \langle D,+,+\rangle,\ \langle D,+,0\rangle,\ \langle D,+,-\rangle,\ \langle D,+,U\rangle \\
\langle D,0\rangle & : & \langle D,0,+\rangle,\ \langle D,0,0\rangle,\ \langle D,0,-\rangle,\ \langle D,0,U\rangle \\
\langle D,-\rangle & : & \langle D,-,+\rangle,\ \langle D,-,0\rangle,\ \langle D,-,-\rangle,\ \langle D,-,U\rangle \\
\langle D,U\rangle & : & \langle D,U,U\rangle \\
\langle U,U\rangle & : & \langle U,U,U\rangle
\end{array}
$$

Table 4.1: The partial curve states of levels 1, 2, and 3

For each point on a curve we can assign a partial curve state of k components, so we could theoretically represent a curve by providing a mapping between curve points and partial curve states. However, such a mapping would be infinite and therefore of no practical use. Because our state components take qualitative values, however, certain states may persist over intervals of curve and, therefore, support mappings that are finite. A curve state may have an interval *and/or* a point interpretation. A state that has an interval interpretation may persist over an interval of curve, and a state that has a point interpretation may hold at a single curve point, without holding on any interval adjoining that point. An *atomic token* identifies a particular interpretation of a partial curve state.

4.1.2 Interval and point interpretation

For each set of level-k curve states, we obtain a corresponding set of level-k atomic tokens by considering the allowable interpretations of each state. A state may support an interval interpretation, a point interpretation, or both. An **atomic token** (or "atom") is a particular interpretation of a particular state, and is identified by a *signature* consisting of a sequence of qualitative values. An underlined signature indicates an interval interpretation; a non-underlined signature indicates a point interpretation. The atom D+, for example, represents the interval interpretation of the state $\langle D,+\rangle$, and the atom D+-0 represents the

point interpretation of the state $\langle D, +, -, 0\rangle$.

A curve state may hold at a single point *iff* one of its components is either zero or undefined. This is because if all components are defined, then all of them are continuous, and hence the values $+$ and $-$ can only hold over intervals. A curve state may persist over an interval *iff* none of its components are undefined and, whenever a component has the value zero, the next component also has the value zero. The following predicates, therefore, can be used to determine the interpretations that are supported by a curve state:

$$\begin{aligned}
\textsf{point-interp}(x) &\Leftrightarrow (\exists k)(\partial^k x = U \vee \partial^k x = 0)\\
\textsf{interval-interp}(x) &\Leftrightarrow (\neg\exists k)(\partial^k x = U) \wedge (\forall k)(\partial^k x = 0 \;\Rightarrow\; \partial^{k+1} x = 0)
\end{aligned}$$

4.1.3 Atomic hierarchy

For each set of partial curve states (each level) we can use the predicates point-interp and interval-interp to obtain the corresponding set of atoms. The partial curve states and atoms for level 3 are as follows:

$\langle D, +, +\rangle$:	D++	$\langle D, 0, U\rangle$:	D0U
$\langle D, +, 0\rangle$:	D+0, D+0	$\langle D, -, +\rangle$:	D-+
$\langle D, +, -\rangle$:	D+-	$\langle D, -, 0\rangle$:	D-0, D-0
$\langle D, +, U\rangle$:	D+U	$\langle D, -, -\rangle$:	D--
$\langle D, 0, +\rangle$:	D0+	$\langle D, -, U\rangle$:	D-U
$\langle D, 0, 0\rangle$:	D00, D00	$\langle D, U, U\rangle$:	DUU
$\langle D, 0, -\rangle$:	D0-	$\langle U, U, U\rangle$:	UUU

Table 4.2: The curve states and atoms of level 3

The first four levels of the hierarchy of atomic tokens are shown in Figure 4.1. Each atom in the hierarchy (except the atoms D and U at level 1) is a child of a single parent atom at the previous level, i.e., each atom at level k is derivable from one, and only one, atom at level $k-1$. We call parent atoms that have more than one child *expansive*, and those with only one child *non-expansive*. An interval atom is expansive *iff* its last component is either

$+$ or $-$, otherwise it is non-expansive. A point atom is expansive *iff* its last component is not U, otherwise it is non-expansive. At level 1, then, there is just one non-expansive atom, U, which propagates through the hierarchy as UU, UUU, and so on. The other level-1 atom, D, expands into five atoms at level 2; two of which are non-expansive: D0 and DU.

Any atom is either an interval (I) or a point (P), and either expansive (E) or non-expansive (N), giving us four kinds of atoms: IE, IN, PE, and PN. Each IE atom yields five children: two IE atoms, one IN atom, one PE atom, and one PN atom. Each PE atom yields four children: three PE atoms and one PN atom. The non-expansive atoms, IN and PN, yield a single IN atom and a single PN atom, respectively.

The recursive equations for calculating the numbers of each kind of atom at level k, and the actual figures for levels 1 to 6, are as follows:

$$
\begin{aligned}
\mathrm{IE}(k) &= 2 \times \mathrm{IE}(k-1) \\
\mathrm{IN}(k) &= \mathrm{IN}(k-1) + \mathrm{IE}(k-1) \\
\mathrm{PE}(k) &= 3 \times \mathrm{PE}(k-1) + \mathrm{IE}(k-1) \\
\mathrm{PN}(k) &= \mathrm{PN}(k-1) + \mathrm{PE}(k-1) + \mathrm{IE}(k-1) \\
\mathit{Total}(k) &= \mathrm{IE}(k) + \mathrm{IN}(k) + \mathrm{PE}(k) + \mathrm{PN}(k)
\end{aligned}
$$

Level	IE	IN	PE	PN	*Total*
1	1	0	0	1	2
2	2	1	1	2	6
3	4	3	5	5	17
4	8	7	19	14	48
5	16	15	65	41	137
6	32	31	211	122	396

Table 4.3: Calculating the number of atoms at each level

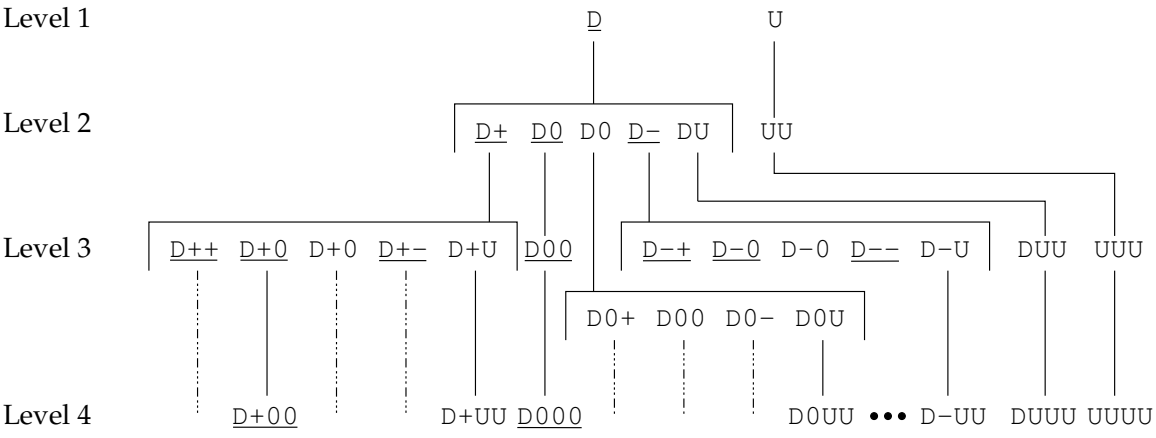


Figure 4.1: The first four levels of the atomic hierarchy

4.1.4 Labelling of the atomic tokens

Atomic signatures are not very easy to read. Therefore, for notational convenience, we assign each atom at level two and above a suitable descriptive label, according to the following convention: the label begins with "P", "Z", "N", or "U", depending on whether the curvature component is positive, zero, negative, or undefined.[1] The remaining components ($c', c'', \ldots$) are denoted by a superscripted string, whose elements are values taken from the set $\{+, 0, -, U\}$. The label is underlined if the atom it is symbolising is an interval. To distinguish between the two distinct kinds of atoms where the curvature component is U, we use the label "$\mathsf{U_b}$" for atoms where the tangent bearing component is also U, and "$\mathsf{U_c}$" for atoms where the tangent bearing is D. Using the new notation, the first three sets of atomic tokens are labelled as follows:

Level 1	
D	D
U	$\mathsf{U_b}$

Level 2	
D+	P
D0	Z
D0	Z
D−	N
DU	$\mathsf{U_c}$
UU	$\mathsf{U_b}$

Level 3							
D++	$\mathsf{P^+}$	D0+	$\mathsf{Z^+}$	D−+	$\mathsf{N^+}$	DUU	$\mathsf{U_c}$
D+0	$\mathsf{P^0}$	D00	$\mathsf{Z^0}$	D−0	$\mathsf{N^0}$	UUU	$\mathsf{U_b}$
D+0	$\mathsf{P^0}$	D00	$\mathsf{Z^0}$	D−0	$\mathsf{N^0}$		
D+−	$\mathsf{P^-}$	D0−	$\mathsf{Z^-}$	D−−	$\mathsf{N^-}$		
D+U	$\mathsf{P^U}$	D0U	$\mathsf{Z^U}$	D−U	$\mathsf{N^U}$		

Table 4.4: The atomic signatures and labels for levels 1, 2, and 3

4.1.5 Kink tokens

We refer to points on a curve where the tangent bearing is undefined as *kink points*, which correspond, perceptually, to angles and cusps.[2] At each level in the hierarchy there exists a single kink-point atom labelled $\mathsf{U_b}$, each component of which takes the value U. Such atoms singularly represent all of the different kinds of points on a curve where the tangent bearing is undefined. Clearly, we would like to be able to distinguish between different

[1] At level 1, D is given the label "D" and U is given the label "$\mathsf{U_b}$", where the "b" signifies that the tangent bearing component is undefined.

[2] A cusp is a point at which two arcs meet from the same direction terminating with a common tangent.

kinds of kink points, otherwise we are losing important curve information. In particular, the two most relevant aspects of a kink point we would like to preserve are its orientation (whether it is inward-pointing or outward-pointing) and its type, i.e., whether it is an angle or a cusp.

In order to distinguish between the different kinds of kink points, we incorporate the following four **kink tokens** into our representation: $U_<$ and $U_>$ (for inward- and outward-pointing angles, respectively), and $U_\prec$ and $U_\succ$ (for inward- and outward-pointing cusps, respectively). These tokens have the same status as U_b, in that they can be thought of as appearing at every level in the hierarchy. Note, however, that kink tokens are not truly *atomic*, since the process by which the atomic tokens are derived does not produce them. Instead, we need to introduce them explicitly.

4.2 String descriptions

A curve is described by a string of atomic tokens taken from a particular level in the hierarchy. By prefixing descriptions with a symbol indicating curve type, both open and closed curves can be represented. We use "$\frown$" for open curves and "$\bigcirc$" for closed ones. Our convention for relating changes in tangent bearing to curvature is that a clockwise change in tangent bearing indicates positive curvature, while an anticlockwise change indicates negative curvature. Thus positive curvature is associated with convex curve segments and negative curvature with concave segments. A curve can be traversed in one of two directions. For a closed curve, the direction is that which preserves the intended figure/ground relationship. For open curves, consistent description dictates that one of the two directions is chosen as the "default" one.

Shown in Figure 4.2 are two curves that have been annotated with atoms from level 3 of the hierarchy. Note that $\underline{P^0}$ refers to a convex circular arc (constant positive curvature over an interval), whereas P^0 refers to a positive curvature extremum; and analogously with $\underline{N^0}$ and N^0. Given a closed curve there will be, in general, a number of different token

strings that describe it. However, as discussed in Section 3.2, we can easily transform one string into another by cyclically shifting it a certain amount. In this sense, all of the strings may be considered equivalent. Any one of sixteen level-3 strings may be used to describe the closed curve in Figure 4.2, two of which are obtained by either starting at $\mathrm{U}_<$ or the uppermost $\underline{\mathrm{Z}^0}$:

$$\bigcirc\, \mathrm{U}_< \, \underline{\mathrm{P}^+} \, \mathrm{P}^0 \, \underline{\mathrm{P}^-} \, \mathrm{Z}^- \, \underline{\mathrm{N}^-} \, \mathrm{N}^0 \, \underline{\mathrm{N}^+} \, \mathrm{U}_\succ \, \underline{\mathrm{Z}^0} \, \mathrm{U}_\mathrm{c} \, \underline{\mathrm{P}^0} \, \mathrm{U}_> \, \underline{\mathrm{Z}^0} \, \mathrm{U}_\prec \, \underline{\mathrm{P}^0}$$

$$\bigcirc\, \underline{\mathrm{Z}^0} \, \mathrm{U}_\mathrm{c} \, \underline{\mathrm{P}^0} \, \mathrm{U}_> \, \underline{\mathrm{Z}^0} \, \mathrm{U}_\prec \, \underline{\mathrm{P}^0} \, \mathrm{U}_< \, \underline{\mathrm{P}^+} \, \mathrm{P}^0 \, \underline{\mathrm{P}^-} \, \mathrm{Z}^- \, \underline{\mathrm{N}^-} \, \mathrm{N}^0 \, \underline{\mathrm{N}^+} \, \mathrm{U}_\succ$$

The importance of specifying a traversal direction for open curves is illustrated by the open curve in Figure 4.2, described by $\frown \underline{\mathrm{P}^-} \, \mathrm{U}_> \, \underline{\mathrm{Z}^0}$. A reversal of direction yields the description $\frown \underline{\mathrm{Z}^0} \, \mathrm{U}_< \, \underline{\mathrm{N}^-}$. Note, however, that under a system where "mirror-reflected" curves are considered equivalent, the direction chosen for traversal is unimportant.

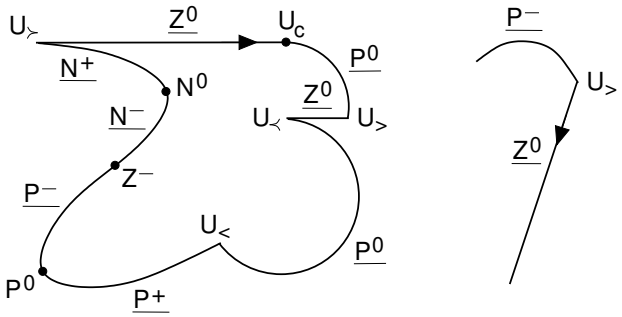


Figure 4.2: Example curves labelled with level-3 atoms

4.2.1 Levels of description

A curve has a description at every atomic level. The most coarse-grained description is at level 1, where only the atoms $\underline{\mathrm{D}}$ and U_b are available. By moving down the hierarchy (incorporating more qualitative components) we get finer-grained descriptions which reflect the increase in discriminatory power that extra components provide.

Given the description of a curve at some level k, we can derive all of the *coarser*-grained

descriptions for the curve, i.e., from the description at level 1 through to the description at level $k-1$. If X denotes the description of a curve, $\mathcal{C}$, at level $k > 1$, then the following straightforward three-step procedure yields the description of $\mathcal{C}$ at level $k-1$:

1. Replace each atom of X with its parent.
2. Iteratively replace substrings of the form xx (where x is a single atom) with x, until there are no more substrings of the form xx.
3. If $\mathcal{C}$ is a closed curve, and the string resulting from the previous step is of length greater than two, and begins and ends with atoms of the same type, then remove the last atom in the string.

By applying the procedure to the ellipse shown in Figure 4.3, we get the level-2 description ○ P, by re-applying the procedure with ○ P as input, we get the most coarse-grained description for an ellipse: ○ D.

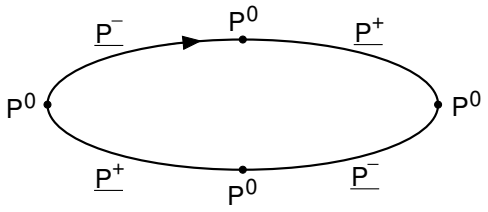


Figure 4.3: An ellipse described using atoms from level 3

Although we can derive all coarser-grained descriptions for a curve given its description at level $k > 1$, it is not possible, in general, to derive any of its finer-grained descriptions. The occasion when it *is* possible to derive a finer-grained description is when the description given consists entirely of atoms that are non-expansive, in which case *all* finer-grained descriptions are accessible. Consider the description of a square at level 2: ○ Z $U_>$ Z $U_>$ Z $U_>$ Z $U_>$. Because both Z and $U_>$ are non-expansive, we can get the "finer-grained" description at level 3 by replacing each atom in the string with its single child

atom. A square's level-5 description, for example, is $\bigcirc\, \underline{\mathsf{Z}^{000}}\, \mathsf{U}_{>}\, \underline{\mathsf{Z}^{000}}\, \mathsf{U}_{>}\, \underline{\mathsf{Z}^{000}}\, \mathsf{U}_{>}\, \underline{\mathsf{Z}^{000}}\, \mathsf{U}_{>}$. Another example would be a shape that has the level-3 description $\frown\, \underline{\mathsf{P}^{0}}$ (i.e., a convex circular-arc segment). For certain shapes, then, there exists a minimal value of k such that the description of the shape at level k contains only non-expansive atoms. For the square, $k = 2$; describing a square at level $k > 2$ is unnecessary, as it leads to no increase in "qualitative precision". For a circle, we need to go to level 3 to achieve this. In this sense, a circle might be regarded as a more complex shape than a square.

4.3 Token-ordering graphs

A curve is described by stringing together those atoms that correspond to the "atomic" qualitative boundary features of the curve. Clearly, some features can occur together and some cannot. A **token-ordering graph** (TOG) encodes the constraints that determine the basic string syntax for a set of tokens. In this section, we show how the transition tables that underlie such graphs can be systematically constructed for each level of the atomic hierarchy.

Given a set of atoms, a TOG is a way of pictorially representing two complementary transition tables that specify which atoms may follow which other atoms in a string. The *interval-interval* ("I-I") table tells us which interval atoms can follow which other interval atoms. The *interval-point-interval* ("I-P-I") table tells us which point atoms and kink tokens can occur in between each pair of interval atoms. It is the construction of these tables, then, that we are primarily interested in.

4.3.1 Compatibility matrices

We make use of two *compatibility matrices* when constructing I-I and I-P-I tables. First, consider the construction of an I-I table for a given level in the hierarchy. Each cell in an I-I table, referred to by $(\text{row } x, \text{column } y)$, contains a tick *iff* the interval atom y can directly follow the interval atom x. Given interval atoms x and y, at level n, we can determine

whether or not y can directly follow x by making use of the matrix given in Table 4.5. The matrix tells us whether or not the kth qualitative components of the atoms x and y are compatible.

	$\partial^k y = +$	$\partial^k y = 0$	$\partial^k y = -$
$\partial^k x = +$	$\top$	$\partial^{k+1} x = -$	$\bot$
$\partial^k x = 0$	$\partial^{k+1} y = +$	$\top$	$\partial^{k+1} y = -$
$\partial^k x = -$	$\bot$	$\partial^{k+1} x = +$	$\top$

Table 4.5: The compatibility matrix for I-I tables

The kth component of y is compatible with the kth component of x, and the predicate $\mathsf{compatible}(k, y, x)$ holds, *iff* the condition in the corresponding cell is satisfied. If $\partial^k x$ and $\partial^k y$ are of opposing signs then they are definitely *in*compatible (signified by a $\bot$ in the appropriate cell), because a derivative cannot change from positive to negative (or vice versa) without taking the value 0 or U in between. If an interval over which ∂^k has the value 0 is followed immediately by an interval over which ∂^k is positive, then in the latter interval it must initially be increasing, i.e., ∂^{k+1} must be positive. This explains the entry for cell $(\partial^k x = 0, \partial^k y = +)$. Analogous explanations hold for the other non-trivial entries in the table. Given interval atoms x and y at level n, then, y can directly follow x *iff* $\mathsf{can\text{-}follow}(y, x)$ holds:

$$\mathsf{can\text{-}follow}(y, x) \quad \Leftrightarrow \quad y \neq x \ \wedge \ (\forall k)(\, 1 < k \leq n \ \Rightarrow \ \mathsf{compatible}(k, y, x)\,)$$

In other words, y and x must be distinct interval atoms (to ensure an underlying state change), and the components of y must be compatible with the components of x. Note that, whenever a condition in the matrix refers to an "unavailable" component of an atom α, i.e., when $k{+}1 > n$, the condition *is* satisfied because the value of $\partial^{n+1}\alpha$ is unconstrained (since it is not specified). As an example, consider the level-3 interval atoms $\underline{\mathsf{P}^+}$ and $\underline{\mathsf{P}^0}$, and whether $\underline{\mathsf{P}^0}$ can directly follow $\underline{\mathsf{P}^+}$. We start by comparing the *second* component of each

atom, since all interval atoms have the same value (D) for their first component. Because both $\partial^2 \underline{\mathsf{P}^+}$ and $\partial^2 \underline{\mathsf{P}^0}$ take the value $+$, the compatibility condition is $\top$, and therefore satisfied. The condition of compatibility for the final component pair evaluates to $\partial^4 \underline{\mathsf{P}^+} = -$, since $\partial^3 \underline{\mathsf{P}^+} = +$ and $\partial^3 \underline{\mathsf{P}^0} = 0$. The condition is satisfied because $\underline{\mathsf{P}^+}$ has no fourth component ($\underline{\mathsf{P}^+}$ is a generalisation of a set of child atoms that includes the level-4 atom $\underline{\mathsf{P}^{+-}}$).

	$\partial^k y = +$	$\partial^k y = 0$	$\partial^k y = -$
$\partial^k x = +$	$\partial^k p = + \ \vee \ \partial^k p = U \ \vee$ $(\partial^k p = 0 \wedge \partial^{k+1} x = -$ $\wedge \partial^{k+1} y = +)$	$\partial^k p = U \vee$ $(\partial^k p = 0 \wedge \partial^{k+1} x = -)$	$\partial^k p = U \ \vee$ $(\partial^k p = 0 \wedge \partial^{k+1} x = -$ $\wedge \partial^{k+1} y = -)$
$\partial^k x = 0$	$\partial^k p = U \vee$ $(\partial^k p = 0 \wedge \partial^{k+1} y = +)$	$\partial^{k-1} p = U$	$\partial^k p = U \vee$ $(\partial^k p = 0 \wedge \partial^{k+1} y = -)$
$\partial^k x = -$	$\partial^k p = U \ \vee$ $(\partial^k p = 0 \wedge \partial^{k+1} x = +$ $\wedge \partial^{k+1} y = +)$	$\partial^k p = U \vee$ $(\partial^k p = 0 \wedge \partial^{k+1} x = +)$	$\partial^k p = - \ \vee \ \partial^k p = U \ \vee$ $(\partial^k p = 0 \wedge \partial^{k+1} x = +$ $\wedge \partial^{k+1} y = -)$

Table 4.6: The compatibility matrix for I-P-I tables

Next, consider the construction of an I-P-I table for a given level in the hierarchy. Each cell (row x, column y) in an I-P-I table contains a list of point atoms that can occur in between interval atoms x and y. For I-P-I tables, we make use of the compatibility matrix given in Table 4.6, which differs from the matrix for I-I tables in that the notion of compatibility this time concerns *three* components rather than just two. Given a point atom p and interval atoms x and y, at level n, the component $\partial^k p$ is compatible with the components $\partial^k x$ and $\partial^k y$, and $\mathsf{compatible}(k, p, x, y)$ holds, *iff* the condition in the corresponding cell is satisfied. A point atom p, then, can occur in between two interval atoms x and y (after x

and before y) *iff* can-occur-between(p, x, y) holds:

$$\begin{aligned}\text{can-occur-between}(p,x,y) \quad \Leftrightarrow \quad & (\exists a, b \leq n)(\, \partial^a p \neq \partial^a x \ \wedge \ \partial^b p \neq \partial^b y \,) \\ & \wedge (\forall k)(\, 1 < k \leq n \ \Rightarrow \ \text{compatible}(k,p,x,y) \,)\end{aligned}$$

In other words, the underlying curve state of p must be distinct from the underlying curve states of x and y, and the qualitative components of p must be compatible with those of x and y. As before, conditions that refer to "unavailable" components are automatically satisfied. To illustrate, we shall use the table to determine which point atoms can come between $x = \underline{\mathsf{P}^-}$ and $y = \underline{\mathsf{P}^0}$. We have $\partial^2 x = \partial^2 y = +$ and $\partial^3 x = -, \partial^3 y = 0$. We want to know the possible values for $\partial^1 p$, $\partial^2 p$ and $\partial^3 p$. Since the definition of can-occur-between imposes no constraints on the first level components, $\partial^1 p$ can be either D or U. For $k = 2$, the top-left cell of the matrix states that $\partial^2 p$ may be any of $+$, U, and 0, but 0 is ruled out because it requires that $\partial^3 x = -$ and $\partial^3 y = +$, which contradicts $\partial^3 y = 0$. Moving on to $k = 3$, we consult the middle cell in the bottom row of the matrix. We see that $\partial^3 p$ may be either U or 0, the additional constraints on the latter value being irrelevant in the present case, as $k + 1 > 3$. Remembering that if any component is U, all subsequent components must be U too, we derive the following candidate component sequences for p:

$$\langle U, U, U \rangle,\ \langle D, +, U \rangle,\ \langle D, +, 0 \rangle,\ \langle D, U, U \rangle$$

These four sequences correspond to the level-3 atoms $\mathsf{U_b}$, $\mathsf{P^U}$, $\mathsf{P^0}$, and $\mathsf{U_c}$. The first condition of can-occur-between rules out $\mathsf{P^0}$, because its underlying curve state is not distinct from that of y. Thus we conclude that only the point atoms $\mathsf{U_b}$, $\mathsf{P^U}$, and $\mathsf{U_c}$ can occur in between the interval atoms $\underline{\mathsf{P}^-}$ and $\underline{\mathsf{P}^0}$.

4.3.2 Transition tables

The set of atoms at level k is the union of a set of interval atoms, $\mathcal{I}_k$, and a set of point atoms, $\mathcal{P}_k$, e.g., at level 2 we have $\mathcal{I}_2 = \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}, \underline{\mathsf{N}}\}$ and $\mathcal{P}_2 = \{\mathsf{Z}, \mathsf{U_c}, \mathsf{U_b}\}$. The transition tables at level k (I-I$_k$ and I-P-I$_k$) each contain $|\mathcal{I}_k|^2$ cells, one for each ordered pair of interval atoms. We refer to a cell (row x, column y) in the I-I$_k$ and I-P-I$_k$ tables by writing

I-I$_k(x, y)$ and I-P-I$_k(x, y)$, respectively. Algorithm 4.1 populates the I-I and I-P-I tables for a given level $k > 1$ in the hierarchy.[3] The "L"-shaped lines in the algorithm indicate the scope of each **for** loop. In the outer loop of the algorithm, each pair of interval atoms $\langle x, y\rangle$ is considered in turn. A tick is added to cell (x, y) of the I-I table if interval y can directly follow interval x (step 1.1). In the inner loop of the algorithm, a point p is placed in cell (x, y) of the I-P-I table if it can occur in between x and y. If p is $\mathsf{U_b}$, then the algorithm places in (x, y) those kink tokens that may occur in between x and y, according to the following rule: inward- and outward-pointing angles can occur in between any two intervals (step 1.2.1.1.1), inward-pointing cusps must be flanked on at least one side by an interval of positive curvature (step 1.2.1.1.2), and outward-pointing cusps must be flanked on at least one side by an interval of negative curvature (step 1.2.1.1.3). This rule is derived from the constraints listed in Section 3.2.4.

Input: k (atomic level)
Output: Populated I-I and I-P-I tables for level k of the atomic hierarchy

```
1       for each ⟨x, y⟩ ∈ I_k × I_k do
1.1       if can-follow(y, x) then I-I_k(x, y) ← √;
1.2       for each p ∈ P_k do
1.2.1       if can-occur-between(p, x, y) then
1.2.1.1       if p = U_b then
1.2.1.1.1       add {U_<, U_>} to I-P-I_k(x, y);
1.2.1.1.2       if ∂²x = + ∨ ∂²y = + then add U_≺ to I-P-I_k(x, y);
1.2.1.1.3       if ∂²x = − ∨ ∂²y = − then add U_≻ to I-P-I_k(x, y);
1.2.1.2       else add p to I-P-I_k(x, y);
```

Algorithm 4.1: Populating the I-I and I-P-I tables for a set of atoms

The I-I and I-P-I tables for level 2 of the atomic hierarchy are given in Table 4.7. The I-I and I-P-I tables for level 3 are given in Tables 4.8 and 4.9 respectively.

[3] At level 1, I-I$_1$(D, D) is empty and I-P-I$_1$(D, D) contains the four kink tokens.

	P	Z	N
P		√	
Z	√		√
N		√	

	P	Z	N
P	Z U_c $U_<$ $U_>$ $U_\prec$	U_c $U_<$ $U_>$ $U_\prec$	Z U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$
Z	U_c $U_<$ $U_>$ $U_\prec$	$U_<$ $U_>$	U_c $U_<$ $U_>$ $U_\succ$
N	Z U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	U_c $U_<$ $U_>$ $U_\succ$	Z U_c $U_<$ $U_>$ $U_\succ$

Table 4.7: I-I and I-P-I tables for the level-2 atoms and kink tokens

	P^+	P^0	P^-	Z^0	N^+	N^0	N^-
P^+		√					
P^0	√		√				
P^-		√		√			
Z^0	√						√
N^+				√		√	
N^0					√		√
N^-						√	

Table 4.8: I-I table for the level-3 atoms

	$\underline{P^+}$	$\underline{P^0}$	$\underline{P^-}$	$\underline{Z^0}$	$\underline{N^+}$	$\underline{N^0}$	$\underline{N^-}$
$\underline{P^+}$	P^0 P^U U_c $U_<$ $U_>$ $U_\prec$	P^U U_c $U_<$ $U_>$ $U_\prec$	P^0 P^U U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\succ$
$\underline{P^0}$	P^U U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\prec$	P^U U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\succ$
$\underline{P^-}$	P^0 P^U Z^0 Z^U U_c $U_<$ $U_>$ $U_\prec$	P^U U_c $U_<$ $U_>$ $U_\prec$	P^0 P^U U_c $U_<$ $U_>$ $U_\prec$	Z^U U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\prec$	Z^0 Z^- Z^U U_c $U_<$ $U_>$ $U_\prec$ $U_\prec$
$\underline{Z^0}$	Z^U U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\prec$	 U_c $U_<$ $U_>$ $U_\prec$	 $U_<$ $U_>$ 	 U_c $U_<$ $U_>$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\succ$	Z^U U_c $U_<$ $U_>$ $U_\succ$
$\underline{N^+}$	Z^+ Z^0 Z^U U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	Z^U U_c $U_<$ $U_>$ $U_\succ$	N^0 N^U U_c $U_<$ $U_>$ $U_\succ$	N^U U_c $U_<$ $U_>$ $U_\succ$	N^0 N^U Z^0 Z^U U_c $U_<$ $U_>$ $U_\succ$
$\underline{N^0}$	 U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\succ$	N^U U_c $U_<$ $U_>$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\succ$	N^U U_c $U_<$ $U_>$ $U_\succ$
$\underline{N^-}$	 U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	 U_c $U_<$ $U_>$ $U_\succ$	N^0 N^U U_c $U_<$ $U_>$ $U_\succ$	N^U U_c $U_<$ $U_>$ $U_\succ$	N^0 N^U U_c $U_<$ $U_>$ $U_\succ$

Table 4.9: I-P-I table for the level-3 atoms and kink tokens

4.3.3 Graph correspondence

An "atomic" TOG, so-called because its nodes represent atomic as opposed to non-atomic tokens, corresponds to a pair of I-I and I-P-I tables and consists of nodes representing atomic tokens and directed edges representing token-ordering constraints. The nodes for interval atoms are shown as rectangles and those for point atoms as circles. A detailed discussion of atomic TOG construction, leading to an algorithm for constructing an atomic TOG from a pair of transition tables, is provided in Appendix B. The atomic TOG for level 2 of the hierarchy is given in Figure 4.4 and the atomic TOG for level 3 in Figure 4.5. Owing to space restrictions, the level-3 atomic TOG is presented as *two* graphs. The lower graph in Figure 4.5 is the main graph which is supplemented by the upper graph incorporating the four kink tokens. The upper graph specifies the allowable kink-token occurrences. The two nodes in the graph labelled $\underline{\mathsf{P}^*}$ and $\underline{\mathsf{N}^*}$ denote, respectively, intervals where the curvature component is positive or negative. The upper graph can be thought of as an *overlay* for the main graph.

Notice that the level-3 atomic TOG is asymmetrical in appearance (because of the placement of the lower Z^{U} node). The graph *can* be re-drawn to look symmetrical, by introducing an extra node for Z^{U} (and re-distributing the eight paths of the form $x \rightarrow \mathsf{Z}^{\mathsf{U}} \rightarrow y$ among three Z^{U} nodes instead of just two), but then the graph no longer has a minimal number of nodes. Note that, with some thought, it *may* be possible to modify the node positions and connections of the minimal graph to attain a symmetrical appearance.

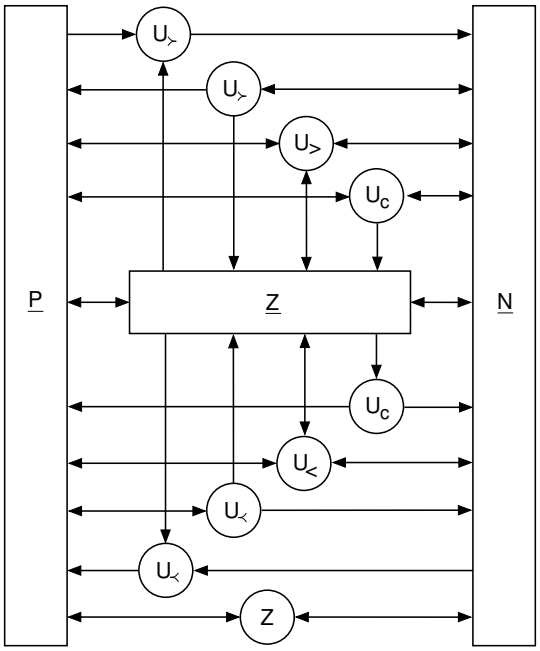


Figure 4.4: Level-2 atomic TOG

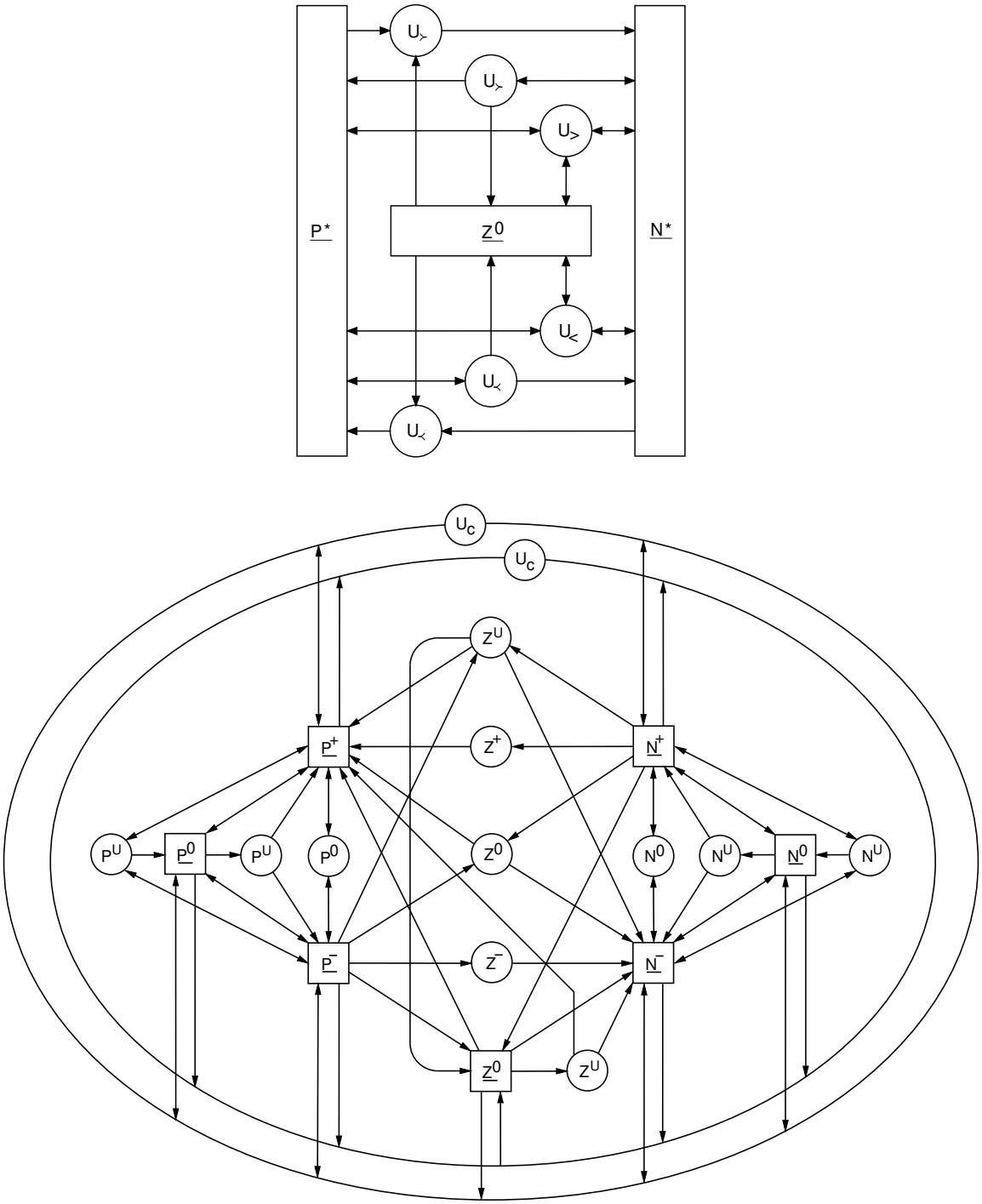


Figure 4.5: Level-3 atomic TOG

4.3.4 String syntax

A TOG (of the atomic *or* non-atomic variety[4]) visually encodes the basic string syntax for a set of tokens. Atomic TOGs, which are categorised according to atomic level, encode the syntax for sets of atoms and kink tokens. A string of tokens $t_1 t_2 \ldots t_n$ is *derivable* from a TOG *iff* the path $t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_n$ exists in the TOG. With reference to the level-2 atomic TOG, for example, the string $\mathsf{U}_{\succ}\,\underline{\mathsf{N}}\,\mathsf{Z}\,\underline{\mathsf{P}}$ *is* derivable, whereas the string $\underline{\mathsf{P}}\,\mathsf{U}_{\succ}\,\underline{\mathsf{Z}}$ is not. A string of tokens is **syntactically valid** *iff* it is derivable from a TOG.

A syntactically valid string may or may not be instantiable as either an open or a closed curve. The string $\frown S$ is valid *iff* S is instantiable as an open curve, and the string $\bigcirc S$ is valid *iff* S is instantiable as a closed curve. Valid strings of the form $\frown X$ may be written as $\diamond X \diamond$, where "$\diamond$" denotes a curve end-point. A string is instantiable as an open curve *iff* (i) it is syntactically valid, (ii) it contains at least one interval atom, and (iii) it starts and finishes with an interval atom, i.e., we consider the interval of points that constitutes an open curve to be topologically open. A string is instantiable as a closed curve *iff* (i) it is syntactically valid, (ii) it contains at least one interval atom, (iii) the first and last atoms are distinct (if the string is of length greater than one), and (iv) it adheres to a set of *closure constraints*. The closure constraints for sets of atomic tokens are beyond the scope of this thesis, although it seems likely (from the analysis of the QOT curvature types in Chapter 8) that the constraints for level 2, at least, would bear a strong resemblance to the QOT closure constraints given in Section 3.2.4.

Listed in Table 4.10 are a number of atomic strings, some of which are syntactically valid and some of which are not. A check of the appropriate atomic TOG provides verification. The table also indicates, for each string X, whether $\frown X$ is valid and whether $\bigcirc X$ is valid. Figure 4.6 provides exemplar curves for those strings in the table that *are* instantiable as open and/or closed curves.

[4] Non-atomic TOGs are covered in Chapter 7.

X	*Syntactically valid?*	$\frown X$ *valid?*	$\bigcirc X$ *valid?*
$\underline{\mathsf{P}}$	√	√	√
$\underline{\mathsf{N}}$	√	√	×
Z	√	×	×
$\underline{\mathsf{P}}\,\underline{\mathsf{N}}$	×	×	×
$\underline{\mathsf{P}}\,\mathsf{Z}^{+}\,\underline{\mathsf{N}}$	×	×	×
$\underline{\mathsf{N}}\,\mathsf{U}_{>}\,\underline{\mathsf{P}}\,\mathsf{U}_{\prec}$	√	×	√
$\underline{\mathsf{Z}^{0}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}^{0}}\,\mathsf{U}_{<}\,\underline{\mathsf{Z}^{0}}\,\mathsf{U}_{>}$ [a]	√	×	×
$\underline{\mathsf{P}^{0}\,\mathsf{P}^{+}}$	√	√	×
$\underline{\mathsf{N}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}$	√	√	√
$\underline{\mathsf{P}^{+}}\,\mathsf{P}^{0}\,\underline{\mathsf{P}^{-}}\,\mathsf{Z}^{0}\,\underline{\mathsf{N}^{+}}\,\underline{\mathsf{N}^{0}}$	×	×	×
$\underline{\mathsf{P}^{-}}\,\underline{\mathsf{Z}^{0}}\,\mathsf{U}_{\mathsf{c}}\,\underline{\mathsf{Z}^{0}}\,\underline{\mathsf{N}^{-}}$	×	×	×
$\mathsf{P}^{0}\,\underline{\mathsf{P}^{-}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}^{0}}\,\mathsf{U}_{>}\,\underline{\mathsf{P}^{+}}$	√	×	√

[a] This string is not instantiable as an open curve because it finishes with a point atom. It is also not instantiable as a closed curve because a polygon must have at least three convex vertices.

Table 4.10: A selection of syntactically valid and invalid atomic strings

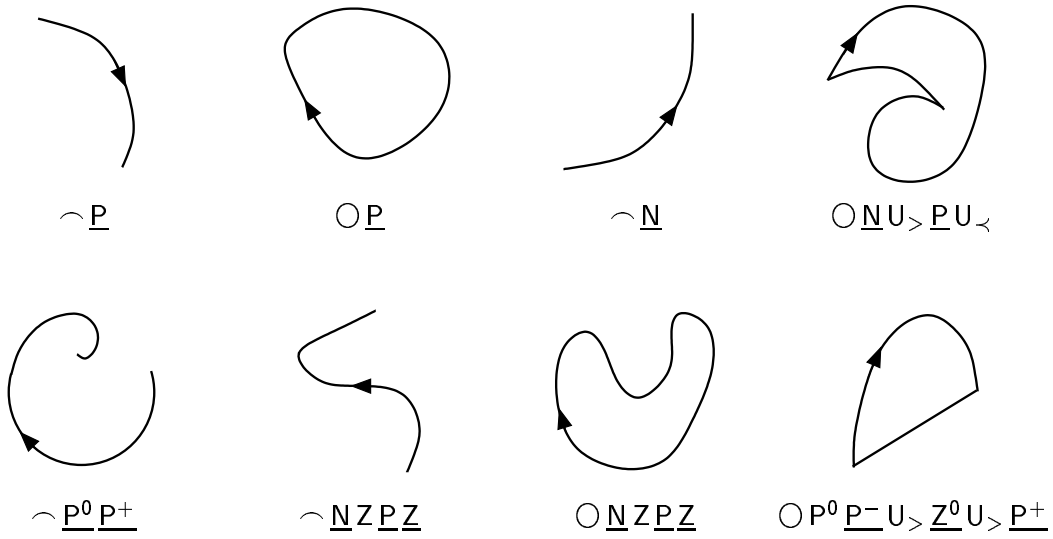


Figure 4.6: Exemplar curves for the instantiable strings in Table 4.10

4.4 Summary

We began the chapter by showing how tangent bearing and its successive derivatives can be characterised by curve states consisting of a number of qualitative components. By considering interval and point interpretations of curve states, we derived sets of atomic tokens. The sets of atomic tokens form an unbounded hierarchy of atoms. Each atom in the hierarchy which has more than one component has a single parent atom at the previous level. As we go down the hierarchy from one level to the next, incorporating more qualitative components, the collection of atoms provided possesses greater discriminatory power. An important consequence of the quantity spaces chosen for the qualitative components is that atomic tokens (and therefore token-string descriptions consisting of atoms) are invariant with respect to translation, rotation, and uniform scaling transformations.

After introducing the atoms, we gave our conventions for curve traversal and introduced a prefix notation for distinguishing between open and closed curves. An open curve has a single description (dependent on traversal direction), whereas a closed curve, in general, has a number of different, but equivalent, string descriptions. We can choose one of the strings to be the canonical one by defining an ordering relation on the set of atomic tokens, and then picking that string description which is lexicographically earliest.

We ended the chapter by showing how atomic TOGs can be constructed for a set of atoms at any level in the hierarchy, from a pair of transition tables (I-I and I-P-I) that are generated, in turn, using two compatibility matrices, one for each table. Nodes in a TOG are associated with tokens and directed edges with token-ordering constraints. A TOG can be used both to verify the syntax of token-string descriptions and to generate syntactically valid strings of tokens. We highlighted the difficulty in determining whether a syntactically valid token string is instantiable as a closed curve, noting that additional constraints are required to ensure closure.

In the next chapter, we develop a model for representing localised curve features that are more abstract than those represented by individual atoms.

Chapter 5

Complex tokens

In this chapter we introduce complex tokens capable of representing non-atomic localised curve features. A complex token is defined as a set of triples, where each triple encodes a leading context, an identity, and a trailing context. Example token specifications are given and the process of fitting complex tokens to atomic strings is described. We conclude the chapter by presenting a set of thirteen relations that may hold between the triples of complex-token specifications.

5.1 Motivation

In the last chapter, we derived a hierarchy of atomic tokens by identifying each atom with an interval or point interpretation of a curve state consisting of a particular number of qualitative components. A curve can be described by a string of atomic tokens, with each token in the string corresponding to an "atomic" feature on the curve. Many curve features, however, are "non-atomic", i.e., there are features that we would like to represent using tokens which do not directly correspond to individual atoms. For example, a curve feature may correspond to a particular *sequence* of atoms, e.g., we might want to identify a "star-point" with the sequence $\underline{Z}\,U_{>}\,\underline{Z}$ (an outward-pointing angle bounded by two straight-line segments). Some curve features may have a number of such "identity sequences", since a curve feature may represent a set of curve-segment classes. Our star-point curve feature,

for example, along with "bump" ($\underline{\mathsf{P}}\,\mathsf{U}_{>}\,\underline{\mathsf{P}}$) and "spike" ($\underline{\mathsf{N}}\,\mathsf{U}_{>}\,\underline{\mathsf{N}}$), may be a member of a more general set of "protrusion" curve features. As well as allowing curve features to be identified with single or multiple identity sequences, there is also a need to take into account the particular *contexts* in which identity sequences may occur. A good example of a curve feature that requires context to be specified is a point of maximum positive curvature, which is one of the primitives of Leyton's system, where it is labelled M^{+}. An M^{+} is identified only with positive stationary points (P^{0} atoms) that are preceded by an interval of positive and increasing curvature ($\underline{\mathsf{P}^{+}}$), and followed by an interval of positive but decreasing curvature ($\underline{\mathsf{P}^{-}}$). In other words, some notion of context is required in order to specify a point of maximum positive curvature.

In the sections that follow, we develop a way of formally specifying *complex tokens* that are capable of representing the kinds of non-atomic curve features we have been discussing.

5.2 Definition of a complex token

A **complex token** is a subset, $\mathcal{S}$, of $\{\diamond, \Lambda\}\Sigma^{*} \times \Sigma^{+} \times \Sigma^{*}\{\diamond, \Lambda\}$, where Σ is the set of tokens at a particular level in the atomic hierarchy[1] and "$\diamond$" is a symbol denoting the end-point of an open curve, such that the following conditions hold:

- Each element in $\mathcal{S}$ is a triple of the form $\langle L, I, T\rangle$, where:
  - L, the *leading context*, is of the form xy, where $x \in \{\Lambda, \diamond\}$ and y is a (possibly empty) string of atomic tokens,
  - I, the *identity*, is a non-empty string of atomic tokens,
  - T, the *trailing context*, is of the form xy, where x is a (possibly empty) string of atomic tokens and $y \in \{\Lambda, \diamond\}$, and

[1] Including the kink tokens.

  - the concatenation of L, I, and T (written $L\,I\,T$) yields a string of atomic tokens that is syntactically valid.

- There is no *redundancy* in $\mathcal{S}$: $\neg(\exists t, t' \in \mathcal{S})(\, t \neq t' \wedge t = \langle b, x, c\rangle \wedge t' = \langle a\,b, x, c\,d\rangle\,)$.

A complex token, then, is specified as a set of triples. Each triple contains an identity string along with the leading and trailing contexts in which the identity must occur in order for it to identify an occurrence of the feature represented by the token.

5.2.1 Example tokens

The curve features described at the start of the chapter (along with "tip-of-star-point" and "tip-of-bump") are specified as complex tokens in Table 5.1. All but two of the tokens have the empty string (Λ) as the leading and trailing contexts of their triple(s). Having the empty string as a context is equivalent to specifying *all possible* contexts of length greater than or equal to one. In other words, the presence of Λ indicates that context is unimportant.

star-point	=	$\{\, \langle \Lambda, \underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}, \Lambda\rangle \,\}$
tip-of-star-point	=	$\{\, \langle \underline{\mathsf{Z}}, \mathsf{U}_{>}, \underline{\mathsf{Z}}\rangle \,\}$
bump	=	$\{\, \langle \Lambda, \underline{\mathsf{P}}\,\mathsf{U}_{>}\,\underline{\mathsf{P}}, \Lambda\rangle \,\}$
tip-of-bump	=	$\{\, \langle \underline{\mathsf{P}}, \mathsf{U}_{>}, \underline{\mathsf{P}}\rangle \,\}$
spike	=	$\{\, \langle \Lambda, \underline{\mathsf{N}}\,\mathsf{U}_{\succ}\,\underline{\mathsf{N}}, \Lambda\rangle \,\}$
protrusion	=	$\{\, \langle \Lambda, \underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}, \Lambda\rangle,\ \langle \Lambda, \underline{\mathsf{P}}\,\mathsf{U}_{>}\,\underline{\mathsf{P}}, \Lambda\rangle,\ \langle \Lambda, \underline{\mathsf{N}}\,\mathsf{U}_{\succ}\,\underline{\mathsf{N}}, \Lambda\rangle \,\}$
M^{+}	=	$\{\, \langle \underline{\mathsf{P}^{+}}, \mathsf{P}^{0}, \underline{\mathsf{P}^{-}}\rangle \,\}$

Table 5.1: A selection of curve features specified as complex tokens

5.2.2 Specification levels

A complex token is specified using the tokens that exist at a particular level, k, in the atomic hierarchy. Given a level-k specification, there necessarily exists an equivalent specification at level $k{+}1$. There *may* also exist an equivalent specification at level $k{-}1$, depending on

the amount of discriminatory power available at that level. For every curve feature that can be represented by a complex token, then, there exists a level, c, below which specification of the curve feature is not possible due to a lack of discriminatory power. Knowing that a curve feature, F, can only be specified at a level greater than, or equal to, c, allows us to make the assertion that F is a level-c curve feature.

Consider star-point as an example. It is specified using atoms at level 2 and is a level-2 curve feature because at level 1 there is no single atom representing a straight-line segment. The *level-3* specification of star-point can easily be derived because the atoms that make up the identity sequence at level 2 ($\underline{\mathsf{Z}}$ and $\mathsf{U}_>$) are both non-expansive. The level 3 specification is therefore $\{\, \langle \Lambda, \underline{\mathsf{Z}^0}\, \mathsf{U}_> \underline{\mathsf{Z}^0}, \Lambda \rangle \,\}$. The same analysis applies to tip-of-star-point. The bump and spike tokens also represent level-2 curve features. This time, however, the equivalent level-3 specifications cannot be derived so easily, since the identity sequence of each token contains two expansive interval atoms ($\underline{\mathsf{P}}$ in the case of bump and $\underline{\mathsf{N}}$ in the case of spike). Taking bump as an example, its identity sequence begins with an interval of positive curvature, represented by $\underline{\mathsf{P}}$ at level 2. At level 3, however, *five* atoms are available ($\underline{\mathsf{P}^+}$, $\underline{\mathsf{P}^0}$, P^0, $\underline{\mathsf{P}^-}$, and P^{U}), allowing different kinds of $\underline{\mathsf{P}}$ to be distinguished. This increase in discriminatory power has the effect that an infinite number of triples are required to specify bump and spike at level 3. In the next section, we extend the notation so that specifications containing an infinite number of triples can be expressed. The tip-of-bump token serves to illustrate that the presence of expansive atoms in a specification doesn't *necessarily* mean that an infinite number of triples are required for the equivalent specification at a higher level. The important difference is that, for tip-of-bump, the $\underline{\mathsf{P}}$s occur in the leading and trailing contexts rather than the identity. Therefore (due to the redundancy condition) we only need to consider, for the leading and trailing contexts, those individual atoms that can, respectively, end and begin an interval of positive curvature. The atoms of interest are $\underline{\mathsf{P}^+}$, $\underline{\mathsf{P}^0}$, and $\underline{\mathsf{P}^-}$, so the level-3 specification of tip-of-bump is as follows:

$$\text{tip-of-bump} \;=\; \{\, \langle x, \mathsf{U}_>, y \rangle \;\mid\; x, y \in \{\underline{\mathsf{P}^+}, \underline{\mathsf{P}^0}, \underline{\mathsf{P}^-}\} \,\}$$

Lastly, consider points of maximum positive curvature, represented by M^+. To identify a stationary point of curvature, the rate of change of curvature must be known, so a point of maximum positive curvature is clearly a level-3 curve feature. Consider the *level-4* specification of M^+. There are four kinds of P^0, namely P^{0+}, P^{00}, P^{0-}, and P^{0U}. The level-3 contexts ($\underline{\mathsf{P}^+}$ and $\underline{\mathsf{P}^-}$) are expansive interval atoms and so we need to identify the atoms at level 4 that may end $\underline{\mathsf{P}^+}$ and begin $\underline{\mathsf{P}^-}$. For $\underline{\mathsf{P}^+}$ we have $\underline{\mathsf{P}^{++}}$, $\underline{\mathsf{P}^{+0}}$, and $\underline{\mathsf{P}^{+-}}$; for $\underline{\mathsf{P}^-}$ we have $\underline{\mathsf{P}^{-+}}$, $\underline{\mathsf{P}^{-0}}$, and $\underline{\mathsf{P}^{--}}$. Let $\mathcal{L} = \{\underline{\mathsf{P}^{++}}, \underline{\mathsf{P}^{+0}}, \underline{\mathsf{P}^{+-}}\}$, $\mathcal{I} = \{\mathsf{P}^{0+}, \mathsf{P}^{00}, \mathsf{P}^{0-}, \mathsf{P}^{0U}\}$, and $\mathcal{T} = \{\underline{\mathsf{P}^{-+}}, \underline{\mathsf{P}^{-0}}, \underline{\mathsf{P}^{--}}\}$. The level-4 specification of M^+ is as follows:

$$\begin{aligned}\mathsf{M}^+ &= \{\, \langle l, i, t\rangle \mid l \in \mathcal{L} \wedge i \in \mathcal{I} \wedge t \in \mathcal{T} \wedge \textsf{can-occur-between}(i, l, t)\,\} \\ &= \{\, \langle \underline{\mathsf{P}^{+-}}, \mathsf{P}^{00}, \underline{\mathsf{P}^{--}}\rangle, \langle \underline{\mathsf{P}^{+-}}, \mathsf{P}^{0-}, \underline{\mathsf{P}^{--}}\rangle, \langle \underline{\mathsf{P}^{+-}}, \mathsf{P}^{0U}, \underline{\mathsf{P}^{--}}\rangle \,\}\end{aligned}$$

5.2.3 Regular expressions and grammars

To facilitate the inclusion of regular expressions and grammars in the specification of a complex token, and to increase legibility, we introduce an alternative way of writing a specification, referred to as *shorthand notation*. A triple $\langle L, I, T\rangle$ is written $L\,[\,I\,]\,T$ in shorthand notation, without writing "Λ" as the empty string. A triple written using shorthand notation is referred to as a "shorthand sequence". The specification of protrusion, for example, consists of three shorthand sequences:

$$\textsf{protrusion} \;=\; \{\, [\,\underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,],\; [\,\underline{\mathsf{P}}\,\mathsf{U}_{>}\,\underline{\mathsf{P}}\,],\; [\,\underline{\mathsf{N}}\,\mathsf{U}_{>}\,\underline{\mathsf{N}}\,]\,\}$$

The expressive power of a shorthand sequence can be substantially increased by allowing any of its three subsequences to be a regular expression or, alternatively, a label referencing a (possibly non-regular) grammar. In this way, a shorthand sequence may correspond to a (possibly infinite) number of triples, rather than just one. As an example, consider the specification of a token representing curvilinear segments of curve that do not contain any points of undefined curvature. Clearly, we need to deal with an infinite number of identities. A curvilinear segment is identified by a level-2 atomic string consisting of a particular combination of $\underline{\mathsf{P}}$, $\underline{\mathsf{N}}$, and Z atoms. The specification in shorthand notation, using a regular expression as the identity subsequence, is as follows:

$$\textsf{curv-seg} \quad = \quad \{\, [(\underline{\mathsf{P}} + \underline{\mathsf{N}})(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{Z}\,\underline{\mathsf{N}})^*] \,\}$$

To present the specification in triple notation we need to list an infinite number of triples, because curv-seg has an infinite number of identities; a separate triple is required for each of the atomic strings in the language defined by the regular expression:

$$\begin{aligned}\textsf{curv-seg} \quad = \quad \{\, &\langle \Lambda, \underline{\mathsf{P}}, \Lambda\rangle,\ \langle \Lambda, \underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}, \Lambda\rangle,\ \langle \Lambda, \underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{N}}, \Lambda\rangle,\ \langle \Lambda, \underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}, \Lambda\rangle,\\ &\langle \Lambda, \underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{N}}, \Lambda\rangle,\ \langle \Lambda, \underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{N}}\,\mathsf{Z}\,\underline{\mathsf{P}}, \Lambda\rangle,\ \langle \Lambda, \underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{N}}\,\mathsf{Z}\,\underline{\mathsf{N}}, \Lambda\rangle,\ \ldots \,\}\end{aligned}$$

An alternative to having a regular expression as one of the three subsequences of a shorthand sequence is to have a label referring to a grammar. The specification for curv-seg, then, can be given with reference to a grammar rather than a regular expression:

$$\textsf{curv-seg} \quad = \quad \{\, [G] \,\} \qquad\qquad G: \quad \begin{aligned} S &\rightarrow \underline{\mathsf{P}}\,A \mid \underline{\mathsf{N}}\,A \\ A &\rightarrow \mathsf{Z}\,S \mid \Lambda \end{aligned}$$

We are now in a position to specify bump at level 3:

$$\textsf{bump} \quad = \quad \{\, [G'] \,\} \qquad\qquad G': \quad \begin{aligned} S &\rightarrow \underline{\mathsf{P}^{+}}\,A \mid \underline{\mathsf{P}^{-}}\,A \mid \underline{\mathsf{P}^{0}}\,B \\ A &\rightarrow \underline{\mathsf{P}^{0}}\,B \mid \mathsf{P}^{0}\,C \mid \mathsf{P}^{\mathsf{U}}\,S \mid \mathsf{U}_{>}\,S' \\ B &\rightarrow \underline{\mathsf{P}^{+}}\,A \mid \underline{\mathsf{P}^{-}}\,A \mid \mathsf{P}^{\mathsf{U}}\,C \mid \mathsf{U}_{>}\,S' \\ C &\rightarrow \underline{\mathsf{P}^{+}}\,A \mid \underline{\mathsf{P}^{-}}\,A \\ S' &\rightarrow \underline{\mathsf{P}^{+}}\,D \mid \underline{\mathsf{P}^{-}}\,D \mid \underline{\mathsf{P}^{0}}\,E \\ D &\rightarrow \underline{\mathsf{P}^{0}}\,E \mid \mathsf{P}^{0}\,F \mid \mathsf{P}^{\mathsf{U}}\,S' \mid \Lambda \\ E &\rightarrow \underline{\mathsf{P}^{+}}\,D \mid \underline{\mathsf{P}^{-}}\,D \mid \mathsf{P}^{\mathsf{U}}\,F \mid \Lambda \\ F &\rightarrow \underline{\mathsf{P}^{+}}\,D \mid \underline{\mathsf{P}^{-}}\,D \end{aligned}$$

Each subsequence of a shorthand sequence yields a set of strings (consisting of just one element if the subsequence *is actually* a single string rather than a regular expression or grammar label). A shorthand sequence $l\,[\,i\,]\,t$, where L, I, and T denote the string sets yielded by l, i, and t respectively, corresponds to the following set of triples:

$$\{\, \langle x, y, z\rangle \ \mid\ x \in L \ \wedge\ y \in I \ \wedge\ z \in T \,\}$$

5.3 Token types

A complex token is a formal specification that represents a curve feature or a class of curve features. The definition we have provided includes only a redundancy constraint (to ensure parsimonious specifications), no constraints are imposed on the kinds of identity strings or contexts that are allowed in a specification. Therefore the definition of a complex token is quite general, enabling a wide range of useful curve features to be specified, such as those given in the previous section and the remainder of this thesis. A side-effect of the general nature of the definition, however, is that curve features that we may refer to as "meaningless" can also be specified. If $\mathsf{tok} = \{ \mathsf{U_c}\,[\underline{\mathsf{Z}}]\,\mathsf{U}_{<},\ [\underline{\mathsf{P}}\,\underline{\mathsf{Z}}]\,\mathsf{U}_{\succ} \}$, for example, then it is not clear that tok actually represents an entity that we would want to refer to as a "curve feature".

A complex token may be classified according to the characteristics of its specification. A specification has a *size*, according to the number of triples it contains and whether it is finite or infinite. All of the tokens we have encountered thus far are finite and contain a small number of triples, except curv-seg, which contains an infinite number of triples, since there are infinitely-many curvilinear segments made up of $\underline{\mathsf{P}}$, $\underline{\mathsf{N}}$, and Z atoms. In addition to set size, complex tokens can also be usefully classified with respect to three other attributes: the *level* of the atomic hierarchy from which the atoms of the specification are taken; whether the feature represented is linelike (e.g., star-point) or pointlike (e.g., M^{+}); and *context*: are the leading and trailing contexts implicit (unimportant), as in the case of curv-seg, for example, or are they explicitly provided (necessary in order to represent the feature), as in the case of tip-of-bump?

In the remainder of this section, we switch our attention to the issue of information loss and preservation, the specification of non-regular curve features, and curve-feature hierarchies.

5.3.1 Information loss and preservation

In choosing to qualitatively characterise curves by representing them using strings of atomic tokens, we sacrifice (”lose”) a certain amount of quantitative information. In particular, information that allows a curve to be precisely reproduced is lost (if the description is obtained from a curve instance), as is information relating to the size, position, and orientation of the linelike boundary features of a curve. The result is that any given string of atoms, rather than corresponding to a specific curve instance, represents a whole class of curves. Such information loss is both inevitable and acceptable, because the purpose of a qualitative scheme for representing shape is to filter out unimportant quantitative detail, leaving only information representing the essential qualities of a shape. For our representational scheme, we have chosen to focus our attention on the characterisation of localised features that can exist on the boundaries of curves.

The issue of information loss is not restricted to the correspondence between strings of atomic tokens and individual curves, however. Information may also be ”lost” in the correspondence between strings of atomic tokens and strings of complex tokens. As far as information loss is concerned, there exist two kinds of complex tokens. An *information preserving* complex token is one that contains a single triple and one, therefore, that corresponds to a single fixed string of atoms. For these kinds of tokens, it is possible to ”recover” the underlying atomic string. Almost all of the complex tokens we have specified thus far have been of the information-preserving variety. Complex tokens which contain more than one triple are *not* information preserving. If a string of complex tokens, for example, contains occurrences of protrusion or curv-seg, then there will exist portions of the underlying atomic description which cannot be precisely determined. In the case of protrusion, which has a specification containing three triples (instances), we can be sure that the underlying portion is one of $\underline{Z}\,U_{>}\,\underline{Z}$, $\underline{P}\,U_{>}\,\underline{P}$, or $\underline{N}\,U_{\succ}\,\underline{N}$. For curv-seg, however, which has infinitely-many instances, our ”choice” for the underlying portion is unlimited.

5.3.2 Non-regular features

One way of categorising a curve feature is by the type of grammar required to list the identities of all the triples of its specification. The identities of all the complex tokens we have considered so far can be specified using regular grammars. A *non-regular feature* is one that requires a grammar with more expressive power than that provided by a regular grammar to specify its identities and/or leading and trailing contexts. Consider a variant of curv-seg, given by curv-seg′, where each curvilinear segment is "atomically symmetrical", i.e., each identity string within the specification is palindromic. A context-free grammar is required in this case:

$$\text{curv-seg}' \;=\; \{\,[G]\,\} \qquad\qquad G: \begin{array}{lcl} S & \rightarrow & \underline{\mathsf{P}} \mid \underline{\mathsf{N}} \mid \underline{\mathsf{P}}\,A\,\underline{\mathsf{P}} \mid \underline{\mathsf{N}}\,A\,\underline{\mathsf{N}} \\ A & \rightarrow & \mathsf{Z} \mid \mathsf{Z}\,S\,\mathsf{Z} \end{array}$$

A further variant of curv-seg would be a token representing curvilinear segments that contain the same number of positive intervals of curvature (P atoms) as negative intervals of curvature (N atoms). In this case, the identity strings required constitute a language which is not regular *or* context-free.

5.3.3 Feature hierarchies

Subtype relationships between complex tokens (and thus curve features) can be established based on set membership. Recall that each triple in the specification of a token represents an instance of the curve feature represented by that token. In other words, if the feature F is represented by the token $\mathcal{T}$, then each element of the specification of $\mathcal{T}$ represents a *kind of* F. Given two curve features, F and F', represented by complex tokens $\mathcal{T}$ and $\mathcal{T}'$ respectively, we can use set-theoretic operations to deduce subtype relationships, as follows:

- If $\mathcal{T} \subset \mathcal{T}'$ then F is a kind of F'.
- If $\mathcal{T} \cap \mathcal{T}' \neq \emptyset$ then there exists a curve feature, F'', represented by $\mathcal{T}'' = \mathcal{T} \cap \mathcal{T}'$, that is both a kind of F *and* a kind of F'.

Referring back to the specifications in Section 5.2.1, we have star-point $\subset$ protrusion, bump $\subset$ protrusion, and spike $\subset$ protrusion, so we can deduce that there are three kinds of protrusion: star-point, bump, and spike. The curvature-specific specifications listed in Table 5.2 produce the feature hierarchy shown in Figure 5.1.[2] In keeping with our convention for node shapes, pointlike features are represented in a curve-feature hierarchy by nodes shown as circles, linelike features by nodes shown as rectangles. The nodes containing question marks indicate that there are other kinds of sv apart from max and min (i.e., curvature inflection points), and that there are kinds of max and min not represented by any tokens in the list (i.e., maxima and minima where the value of curvature is zero).

$$
\begin{aligned}
\mathsf{sv} &= \{\,[\mathsf{P}^0],\ [\mathsf{Z}^0],\ [\mathsf{N}^0]\,\} \\
\mathsf{max} &= \{\underline{\mathsf{P}^+}[\mathsf{P}^0]\underline{\mathsf{P}^-},\ \underline{\mathsf{N}^+}[\mathsf{Z}^0]\underline{\mathsf{N}^-},\ \underline{\mathsf{N}^+}[\mathsf{N}^0]\underline{\mathsf{N}^-}\} \\
\mathsf{min} &= \{\underline{\mathsf{P}^-}[\mathsf{P}^0]\underline{\mathsf{P}^+},\ \underline{\mathsf{P}^-}[\mathsf{Z}^0]\underline{\mathsf{P}^+},\ \underline{\mathsf{N}^-}[\mathsf{N}^0]\underline{\mathsf{N}^+}\} \\
\mathsf{M}^+ &= \{\underline{\mathsf{P}^+}[\mathsf{P}^0]\underline{\mathsf{P}^-}\} \\
\mathsf{M}^- &= \{\underline{\mathsf{N}^+}[\mathsf{N}^0]\underline{\mathsf{N}^-}\} \\
\mathsf{m}^+ &= \{\underline{\mathsf{P}^-}[\mathsf{P}^0]\underline{\mathsf{P}^+}\} \\
\mathsf{m}^- &= \{\underline{\mathsf{N}^-}[\mathsf{N}^0]\underline{\mathsf{N}^+}\}
\end{aligned}
$$

Table 5.2: Complex tokens representing salient curvature points

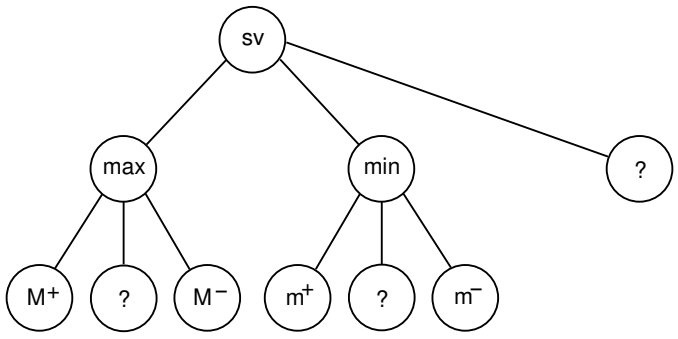


Figure 5.1: The curve-feature hierarchy for the tokens listed in Table 5.2

[2]Where "sv" stands for "stationary value".

5.4 Token fitting

Given an atomic description of a curve, we would like to derive descriptions that consist of complex tokens, rather than atoms, so that we may reason about the curve at a higher level of abstraction. *Token fitting* is a two-stage process by which complex tokens are fitted to a string of atomic tokens. During the initial *triple matching* stage, triples with identities and contexts that match a substring of the atomic description are associated with ("fitted to") that substring. During the *triple elimination* stage, fitted triples are removed if they are determined to be redundant with respect to the other fitted triples present.

5.4.1 Triple matching

Token fitting is carried out with respect to an atomic token-string, S, of length n, and a set of complex tokens, $\mathcal{T}$. The first stage of token fitting involves the consideration of candidate triples taken from each specification in $\mathcal{T}$, and determining, for each candidate, whether it can be fitted to S and, if it can be fitted, the substring(s) of S it can be fitted to. Note that a triple may be fitted to more than one substring, and more than one triple may be fitted to the same substring. Even though the specification of a complex token may contain an infinite number of triples, only a finite number of triples need to be considered from each specification, because S is finite in length. In particular, given a triple t, if a and c are the lengths of its leading and trailing contexts and b is the length of its identity, then t only needs to be considered if $a + b + c \leq n$. There is guaranteed to be a finite number of such triples in each specification. In certain circumstances, it may be possible to reduce the size of the set of candidate triples further by using heuristics based on the particular atoms present in S.

Once the set of candidate triples has been ascertained, the task of determining which triples fit which substrings of S is a straightforward one, involving simple string matching. The only complication is that, as usual, if S is a string that represents a closed curve then it is interpreted as cyclically permutable. The triple matching stage is illustrated for closed

and open curves by Figures 5.2 and 5.3 respectively. Figure 5.2 contains a closed curve described at level 2 and a *ring diagram* showing how triples fit to the various substrings of its description. Here, the set of complex tokens we consider in the token fitting process consists of all the specifications given in Section 5.2. A ring diagram consists of an inner ring of atomic tokens (the curve description) and a number of outer rings showing the *identities* of the fitted triples.

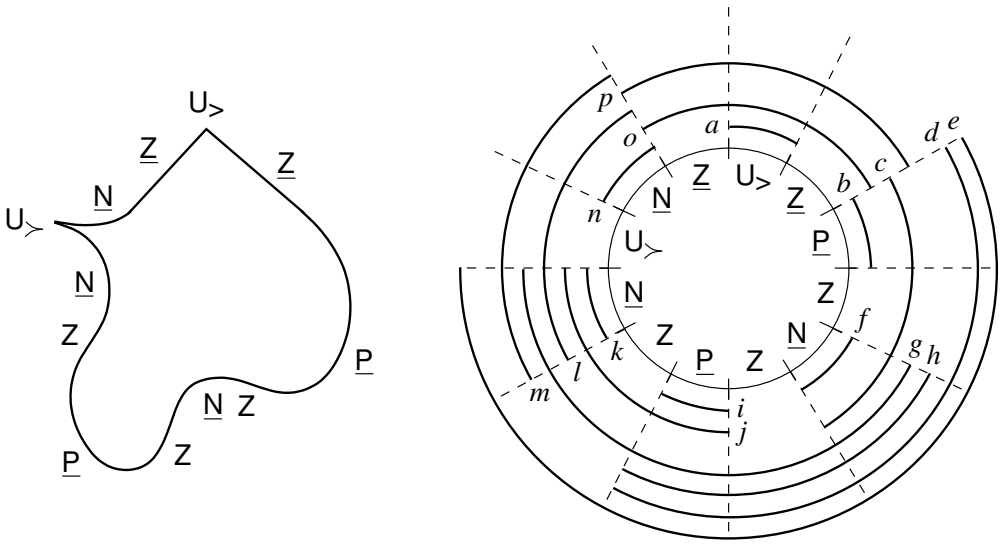


Figure 5.2: Ring diagram showing triples matched to a closed curve

There are sixteen fitted triples in the ring diagram of Figure 5.2, labelled a to p. Each fitted triple identifies an *instance* of one of the complex tokens. Fitted triple a, for example, identifies an instance of the tip-of-star-point curve feature. There are places where more than one triple fits a substring of the description, e.g., the substring $\underline{\mathrm{N}}\,\mathrm{U}_{\succ}\,\underline{\mathrm{N}}$, where l and m identify instances of the spike and protrusion curve features. This is to be expected, of course, since a spike is a kind of protrusion (as spike is a subset of protrusion). Similarly, o and p identify instances of star-point and protrusion that match the substring $\underline{\mathrm{Z}}\,\mathrm{U}_{>}\,\underline{\mathrm{Z}}$. The shape includes two other curve features. An instance of star-point is identified by p and there are eleven instances of curv-seg, identified by $b - k$ and n. A list of the triples fitted

to the description, ordered according to the curve features each identifies, is provided in Table 5.3.

Letter(s)	*Triple fitted*	*Curve feature identified*
o	$\langle \Lambda, \underline{\mathsf{Z}}\ \mathsf{U}_{>}\ \underline{\mathsf{Z}}, \Lambda \rangle$	star-point
a	$\langle \underline{\mathsf{Z}}, \mathsf{U}_{>}, \underline{\mathsf{Z}} \rangle$	tip-of-star-point
l	$\langle \Lambda, \underline{\mathsf{N}}\ \mathsf{U}_{\succ}\ \underline{\mathsf{N}}, \Lambda \rangle$	spike
p	$\langle \Lambda, \underline{\mathsf{Z}}\ \mathsf{U}_{>}\ \underline{\mathsf{Z}}, \Lambda \rangle$	protrusion
m	$\langle \Lambda, \underline{\mathsf{N}}\ \mathsf{U}_{\succ}\ \underline{\mathsf{N}}, \Lambda \rangle$	protrusion
b, i	$\langle \Lambda, \underline{\mathsf{P}}, \Lambda \rangle$	curv-seg
f, k, n	$\langle \Lambda, \underline{\mathsf{N}}, \Lambda \rangle$	curv-seg
c, j	$\langle \Lambda, \underline{\mathsf{P}}\ \mathsf{Z}\ \underline{\mathsf{N}}, \Lambda \rangle$	curv-seg
h	$\langle \Lambda, \underline{\mathsf{N}}\ \mathsf{Z}\ \underline{\mathsf{P}}, \Lambda \rangle$	curv-seg
d	$\langle \Lambda, \underline{\mathsf{P}}\ \mathsf{Z}\ \underline{\mathsf{N}}\ \mathsf{Z}\ \underline{\mathsf{P}}, \Lambda \rangle$	curv-seg
g	$\langle \Lambda, \underline{\mathsf{N}}\ \mathsf{Z}\ \underline{\mathsf{P}}\ \mathsf{Z}\ \underline{\mathsf{N}}, \Lambda \rangle$	curv-seg
e	$\langle \Lambda, \underline{\mathsf{P}}\ \mathsf{Z}\ \underline{\mathsf{N}}\ \mathsf{Z}\ \underline{\mathsf{P}}\ \mathsf{Z}\ \underline{\mathsf{N}}, \Lambda \rangle$	curv-seg

Table 5.3: A list of the fitted triples from the ring diagram of Figure 5.2

Figure 5.3 shows an open curve described at level 3, together with an associated ring diagram showing the triples that match substrings of the curve description. We can see that, for open curves (which do not have cyclically permutable descriptions) the rings in the diagram have a segment "missing", allowing the beginning and end of the description to be identified. There are six fitted triples in the diagram, labelled u to z. A point of maximum positive curvature is identified by w. The fitted triple z identifies an instance of the tip-of-bump curve feature, whose level-3 specification was derived in Section 5.2.2. The remaining fitted triples (u, v, x, and y) all identify particular instances of bump, whose level-3 specification was given in Section 5.2.3. A list of the six fitted triples, ordered according to the curve features each identifies, is provided in Table 5.4.

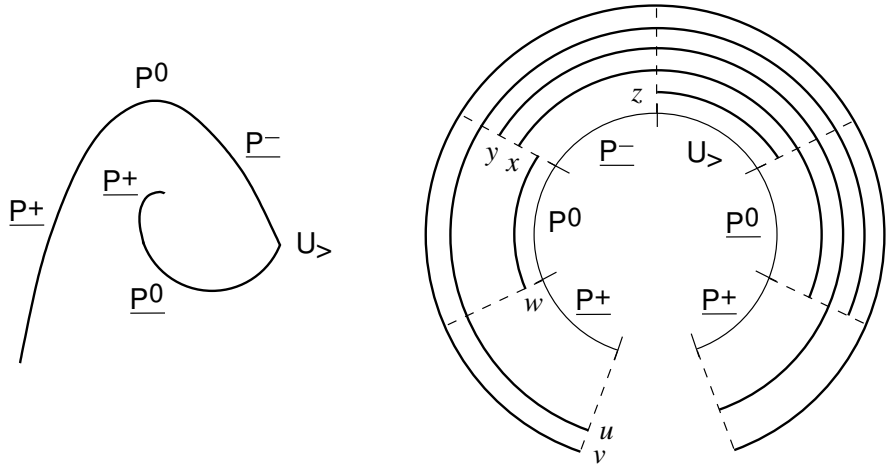


Figure 5.3: Ring diagram showing triples matched to an open curve

Letter(s)	*Triple fitted*	*Curve feature identified*
z	$\langle \underline{\mathsf{P}^-}, \mathsf{U}_>, \underline{\mathsf{P}^0} \rangle$	tip-of-bump
w	$\langle \underline{\mathsf{P}^+}, \mathsf{P}^0, \underline{\mathsf{P}^-} \rangle$	M^+
x	$\langle \Lambda, \underline{\mathsf{P}^-}\, \mathsf{U}_> \underline{\mathsf{P}^0}, \Lambda \rangle$	bump
y	$\langle \Lambda, \underline{\mathsf{P}^-}\, \mathsf{U}_> \underline{\mathsf{P}^0}\, \underline{\mathsf{P}^+}, \Lambda \rangle$	bump
u	$\langle \Lambda, \underline{\mathsf{P}^+}\, \mathsf{P}^0\, \underline{\mathsf{P}^-}\, \mathsf{U}_> \underline{\mathsf{P}^0}, \Lambda \rangle$	bump
v	$\langle \Lambda, \underline{\mathsf{P}^+}\, \mathsf{P}^0\, \underline{\mathsf{P}^-}\, \mathsf{U}_> \underline{\mathsf{P}^0}\, \underline{\mathsf{P}^+}, \Lambda \rangle$	bump

Table 5.4: A list of the fitted triples from the ring diagram of Figure 5.3

5.4.2 Triple elimination

After the triple matching stage comes triple elimination. The purpose of triple elimination is to remove ("unfit") those fitted triples that are considered to be redundant. As an example of redundancy, consider again Figure 5.2. Even though eleven separate instances of curv-seg are identified (by b – k and n), only *two* of the instances of the curv-seg curve feature are really of interest: the instance identified by e and the instance identified by n. The remaining nine fitted triples are redundant, their presence reflecting the fact that a subsegment of a curv-seg is itself a curv-seg. Redundancy is also present in the diagram

of Figure 5.3, in which there is really only a single significant instance of the bump curve feature, given by v. The other three instances (x, y, and u) are redundant.

Redundancy is determined on a per-specification basis, i.e., a triple fitted from one specification in $\mathcal{T}$ cannot be redundant with respect to a fitted triple belonging to a *different* specification. Before we give the conditions that must be satisfied in order for a fitted triple to be removed, we first define what it means for one fitted triple to be covered by another.

Definition 5.1 *A fitted triple, t, is* **covered by** *another fitted triple, t',* iff *the substring of the curve description that the identity of t is fitted to is itself a substring of the substring of the curve description that the identity of t' is fitted to.*

Referring back to Figure 5.2, we can see that, apart from e, *every* fitted triple in the diagram is covered by another fitted triple.[3] A number of the fitted triples are covered by fitted triples from a *different* specification, e.g., a (tip-of-star-point) is covered by o (star-point) and p (protrusion). The same applies in the diagram of Figure 5.3, where v is the only non-covered fitted triple. We are now in a position to define what it means for a fitted triple to be redundant.

Definition 5.2 *Given two fitted triples, t and t', from the same specification, t is* **redundant** iff *(i) t is covered by t', and (ii) t' is not covered by t.*

During triple elimination, all redundant fitted triples are removed ("unfitted"). The results of triple elimination for the two previous ring diagrams are shown in Figure 5.4. For the closed curve, token fitting reveals the presence of two curv-seg features (e and n), two protrusions (m and p), a spike (l), a star-point (o), and a tip-of-star-point (a). For the open curve, token fitting has resulted in three higher-level curve features being identified: bump (v), tip-of-bump (z), and M^+ (w).

As stated previously, token fitting operates on a level-k atomic description. Consequently, the presence of a level-x curve feature (where $x > k$) cannot be detected, because

[3]Given two strings, s and s', if $s = s'$ then s is a substring of s' and s' is a substring of s.

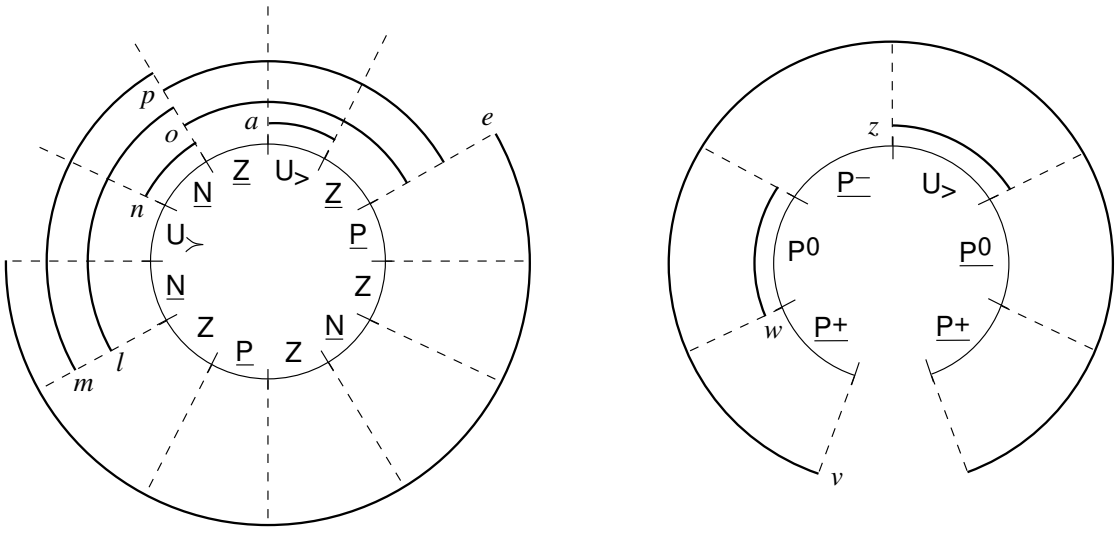


Figure 5.4: Ring diagrams showing the results of triple matching and elimination

the description doesn't contain enough information. This explains, for example, why we could not fit M^+ to the description of the closed curve, even though there are two segments of the curve (given by the $\underline{\mathsf{P}}$ atoms) that *may* include a point of maximum positive curvature. Even if we are unable to fit a token to a description, then, we can still ascertain whether the presence of the curve feature it represents is a *possibility*. In the converse situation, where we have a complex token specified at level $x < k$, we need to derive the equivalent specification at level k before we can fit the token. This explains why curv-seg has not been fitted to the open curve. Note that translating the level-3 description of the open curve into its level 2 equivalent (using the procedure given in Section 4.2.1), yields a description that curv-seg *can* be fitted to ($\underline{\mathsf{P}}\,\mathsf{U}_{>}\,\underline{\mathsf{P}}$).

5.4.3 Token-string descriptions

We have shown how a ring diagram is used to associate instances of complex tokens with substrings of an atomic token-string description. The process of token fitting, then, is essentially one that maps a string of atomic tokens to a ring diagram containing only non-

redundant fitted triples. A token-string description is a string of labels representing curve features. The position of a label in the string corresponding to the position of the represented feature on the curve, i.e., features follow one another along the curve, as encoded by the order of the labels in the string. A token string must contain only token labels, additional symbols, for encoding the relative placement of curve features, are not permitted. Consequently, it is not always the case that a ring diagram, obtained via token fitting, can be mapped to a complex-token string. Consider the final ring diagram, given in Figure 5.4, for the closed curve in Figure 5.2. Since a number of the fitted triples are covered by other fitted triples, it is not clear what the order of the complex tokens in the string should be. If we are forced to attempt the mapping, we end up with a string whose label ordering contradicts the actual positions of the curve features, e.g.:

$$\bigcirc \text{ curv-seg protrusion spike curv-seg protrusion star-point tip-of-star-point}$$

The same problem exists with the ring diagram for the open curve, in which both w and z are covered by v. The most intuitive token-string choice on this occasion being, perhaps, $\frown$ bump M^+ tip-of-bump. It is simply not the case, however, that M^+ occurs *after* bump (in the sense that bump has ended before the occurrence of M^+), although it *is* the case that tip-of-bump occurs after M^+. Clearly, a string of tokens is an inadequate representation for either of the two ring diagrams as they stand. The root of the problem lies in our choice of the complex tokens considered for the token fitting process. In particular, we included tokens that co-occur, such as protrusion and star-point, i.e., the presence of star-point implies a concomitant presence of protrusion, because a star-point is a *kind of* protrusion. If we had omitted protrusion and tip-of-star-point (which is already included in a star-point segment) from the token fitting process, then the final ring diagram for the closed curve would not include m, p, or a, with the result that an unambiguous mapping to a complex-token string is then possible:

$$\bigcirc \text{ curv-seg spike curv-seg star-point}$$

In the next chapter, using the triple relations provided in the following section, we focus our attention on *sets* of complex tokens, and the properties required by a set to ensure that it supports token-string description, i.e., that token fitting using the tokens in the set always yields a ring diagram that can be mapped, unambiguously, to a string of complex tokens.

5.5 Triple relationships

In this section, we present a set of thirteen "may-x" relations that may hold between two triples. The triples may belong to the same specification or to different specifications and more than one of the relations may hold between them. A relation holds between two triples, $t = \langle a, x, b \rangle$ and $t' = \langle c, y, d \rangle$, *iff* there exists a *valid alignment* of the strings $a\,x\,b$ and $c\,y\,d$ such that a particular positional relationship exists between their identity substrings, x and y. Before listing the relations, we introduce the diagrammatic notation used to represent triple alignments.

5.5.1 Bar-diagram notation

A **valid alignment** of two triples, $t = \langle a, x, b \rangle$ and $t' = \langle c, y, d \rangle$, corresponds to one particular way in which the string $a\,x\,b$ can be written above or below the string $c\,y\,d$ (in the manner of a matrix consisting of two adjoining rows of cells) such that the following two conditions are met:

1. Each two-cell column in the matrix either (i) contains one empty and one non-empty cell, or (ii) contains two non-empty cells containing the same atomic token.

2. The string of atomic tokens obtained by starting at the leftmost column of the matrix and finishing at the rightmost column (reading the contents of one of the non-empty cells in each column of the matrix) is syntactically valid.

As an example, one of the valid alignments of [P Z P] and [P Z P Z P Z P] (two triples taken from the specification of curv-seg) is given in Table 5.5. The string of atoms read from the matrix, P Z P Z P Z P Z P, is derivable from the level-2 atomic TOG and therefore syntactically valid. There exist four other valid alignments, corresponding to the leftmost P of the former triple being aligned to one of the four Ps of the latter.

P	Z	P						
		P	Z	P	Z	P	Z	P

Table 5.5: A valid alignment of two curv-seg triples

A class of valid alignments is represented by a *bar diagram*, an example of which is given in Figure 5.5. A bar diagram contains *spring bars*, which represent leading and trailing contexts and are drawn as zigzagged lines, and *fixed bars*, which represent identities and are drawn as solid lines of greater thickness. A spring bar represents a string of length zero or greater (since leading and trailing contexts can be empty). There are two kinds of fixed bar: if a dot is present in the middle of a fixed bar then it represents a string of length one or greater, otherwise it represents a string of length *two* or greater (recall that an identity must be at least one atom in length). A fixed bar is bounded by vertical lines denoting the first and last atoms of an identity string. For dotted fixed bars, then, where the identity string can be a single atom, the vertical lines may denote the same atom. A spring bar has a fixed end, joined to its neighbouring fixed bar, and a free end, whose actual position may be anywhere between the position it is drawn in the diagram and the vertical line bounding the neighbouring fixed bar (the position at which it represents an *empty* leading or trailing context). In Figure 5.5, then, a class of valid triple alignments is represented, whereby the identity of triple t is "started by" the identity of triple t' (which may be one atom in length) and there are no restrictions placed on the relative starting and finishing positions of the leading and trailing contexts of t and t' (other than those implied by the bounding lines of each fixed bar, e.g., the trailing context of t cannot start *before* the trailing

context of t' starts, but it may finish before, after, or at the same place the trailing context of t' finishes). Referring back to the two curv-seg triples, the second of the five possible valid alignments (where the two leftmost P̲s are aligned) is of the kind represented by the bar diagram in Figure 5.5.

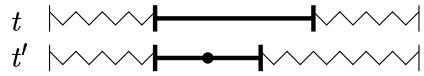


Figure 5.5: An example bar diagram

5.5.2 The may-x relations

The thirteen triple relations are listed in Table 5.6. Each relation has a name of the form "may-x" and (for brevity) an associated short label of the form tr$N$, where N is a value from one to thirteen. The relations are concerned with the valid alignments that exist between two triples and, within a valid alignment, the positional configurations of the two identity strings. As an example, consider the first two relations: may-merge-into (tr1) and may-emerge-from (tr2). Given two triples, t and t', the relations t may-merge-into t' and t' may-emerge-from t hold *iff* there exists a valid alignment of t and t' which is of the kind represented by the bar diagram shown alongside the two relations in the table, i.e., an alignment whereby the first atom of the identity of t' is aligned with the last atom of the identity of t. Note that both identities must be greater than one atom in length, because the fixed bars in the diagram are not dotted. From the valid alignment shown in Table 5.5, for example, we see that [P̲ Z P̲] may-merge-into [P̲ Z P̲ Z P̲ Z P̲] holds and, therefore, [P̲ Z P̲ Z P̲ Z P̲] may-emerge-from [P̲ Z P̲] holds.

A consequence of our atomic sequences being discrete is that we can distinguish between the "merging" of identities (tr1 and tr2) and the "meeting" of identities (tr3 and tr4). The difference is illustrated in Figure 5.6. Either or both of two identities that "meet" are allowed to be one atom in length (the fixed bars in the diagram for tr3 and tr4 are both

dotted). The remaining relations (tr5 to tr13) correspond to the other possibilities that exist for the positional configuration of the identity strings within a valid alignment. Note that we are not interested here in valid alignments where the identities are not in contact, i.e., separated by one or more context atoms.

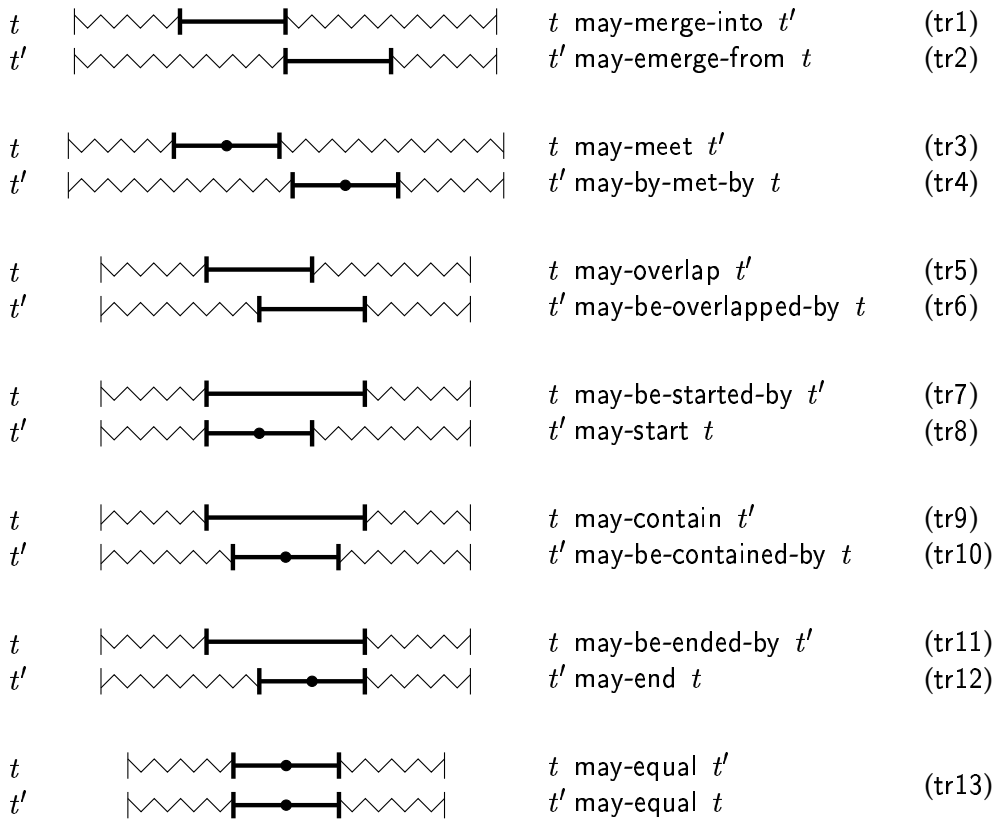


Table 5.6: The thirteen may-x triple relations

Given a pair of triples, a number of valid alignments may exist, or only one may exist, *or*, there may be *no* valid alignments. Consequently, more than one of the relations may hold between a pair of triples, or none of the relations may hold. Between the two curv-seg triples we considered earlier, for example, all of the relations hold except the "meeting" relations (tr3 and tr4), the "overlap" relations (tr5 and tr6), and may-equal (tr13).

The triple relations are loosely analogous to the thirteen interval relations provided

Figure 5.6: The "merging" of identities versus "meeting"

by Allen for the temporal domain (Allen 1984), with identity strings serving the role of intervals. Since an identity is a string of discrete symbols (as opposed to being continuous) we are able to single out the "merging" of two identities as a salient relationship. Also, because we are interested only in those instances where identities are in contact, we do not require the notion of one identity occurring "before" or "after" another. The final difference is that Allen's temporal relations are JEPD, whereas our triple relations are not.

The purpose of the triple relations is to allow us to reason about the possibilities that exist for the relative placement of *fitted triples*. As an example, consider the two triples $t = [\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,]$ and $t' = \mathsf{Z}\,[\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}}$. The four valid alignments of t and t' are given in Table 5.7, along with the triple relations (if any) that hold in each case. Given a level-2 description of a curve, either triple may fit the description at one or a number of places. In the case of the second valid alignment of the triples, the relation t may-merge-into t' holds. This tells us that there exist curve descriptions where both t and t' fit, and where the identity of a fitted instance of t *does* merge into the identity of a fitted instance of t'. In particular, such a merging will occur whenever the substring $\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}$ forms part of a curve description. The may-merge-into relation asserts, then, that given the description of a curve, it *may* be the case that there are fitted instances of the triples where a merging of identities occurs. The reason why a merging of identities will not *necessarily* occur is that the fitting of t does not imply a concomitant fitting of t', and vice versa. Note, however, that if the leading context of t' was $\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}$, and the trailing context of t was $\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}$, then the fitting of one of the triples *would* necessitate the fitting of the other. The third valid alignment of t and t' tells us that the relation t may-overlap t' holds. This means that, given the description of a curve, it may be the case that the identity of a fitted instance of t overlaps the identity of a fitted instance of t'. Given the identities and contexts of t and t',

such an overlapping *will* occur whenever a curve description contains $\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}$ as a substring.

Our discussion of token-string descriptions in Section 5.4.3 highlighted the problems that can arise when mapping a set of fitted triples to a string of complex tokens. In particular, the representation of fitted triples with covered or overlapped identities was shown to be problematic. The way in which an individual token may itself contribute to these problems can be ascertained by checking the relationships that hold between the triples of its specification. In the next chapter, we use the triple relations to reason about individual complex tokens as well as sets of complex tokens.

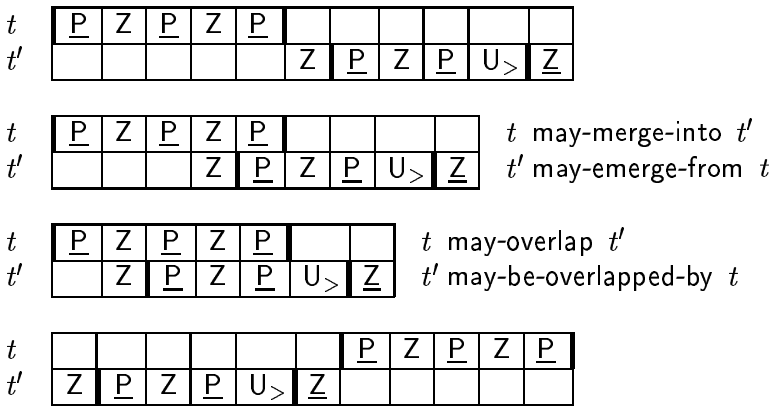


Table 5.7: The four valid alignments of $[\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,]$ and $\mathsf{Z}\,[\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}}$

5.6 Summary

In this chapter, we have developed a notation for specifying complex tokens that are capable of representing non-atomic localised curve features. We have given a number of example specifications and discussed some ways in which complex tokens can be classified, e.g., by set size, level, etc. We have also shown how subtype relationships between tokens can be deduced by set-theoretic operations.

The process of token fitting, by which atomic curve descriptions are translated into

equivalent non-atomic descriptions, consists of two stages, triple matching and triple elimination, and was described in Section 5.4. The triple matching stage fits candidate triples to an atomic description at those positions compatible with the identities and contexts of the triples. During the triple elimination stage, triples that have been fitted to the atomic description by the first stage are considered redundant (and removed), if they are covered by other fitted triples belonging to the *same* token specification.

In the next chapter, we focus our attention on *sets* of complex tokens and give a definition of a *local-feature scheme*. A local-feature scheme is a set of complex tokens which, when fitted to an atomic description, necessarily yield a ring diagram that can be simply mapped to a string of complex tokens.

Chapter 6

Local-feature schemes

In this chapter we consider sets of complex tokens in the context of shape description using token-strings. We provide the constraints on a set of complex tokens which ensure that token fitting yields a ring diagram that can be unambiguously mapped to a string of complex tokens. We then define a local-feature scheme to be a set of complex tokens subject to the constraints.

6.1 Motivation

In our discussion of token fitting in the last chapter (Section 5.4) we considered the mapping from a ring diagram, consisting of non-redundant fitted triples, to a string of complex tokens, and the difficulties associated with such a mapping. In particular, we illustrated that a mapping may yield an ambiguous or inconsistent token-string description if the fitted triples cover or overlap one another in certain ways. In order to ensure that the token fitting process yields a ring diagram that maps to an unambiguous string, we need to carefully choose the set of complex tokens that we want to fit to atomic descriptions. It is the choice of complex tokens used for the fitting process that determines whether unambiguous token-string description is possible or not. In this chapter, we formulate the constraints required for a set of complex tokens to ensure that the fitting process, when using the tokens, *always* yields a ring diagram that maps to an unambiguous string.

Our first task is to identify the types of positional configurations of non-redundant fitted triples that are problematic. To do this, we make use of the ring diagram given in Figure 6.1, which we may refer to as an *anonymous* diagram since the elements of the atomic description are unspecified (we just know that the description consists of twelve atoms). The diagram shows the result of the two-stage token fitting process applied to an atomic description of length twelve. We can deduce, therefore, that if one fitted triple is covered by another then the two triples belong to different specifications (unless the two triples cover *each other*, as d and e do, in which case we cannot make the deduction).[1] For example, f cannot belong to the same specification as either g or a (otherwise g and a would have been removed during the elimination stage), whereas g and a themselves may or may not belong to the same specification. Similarly, a cannot belong to the same specification as h, and g and h may or may not belong to the same specification.

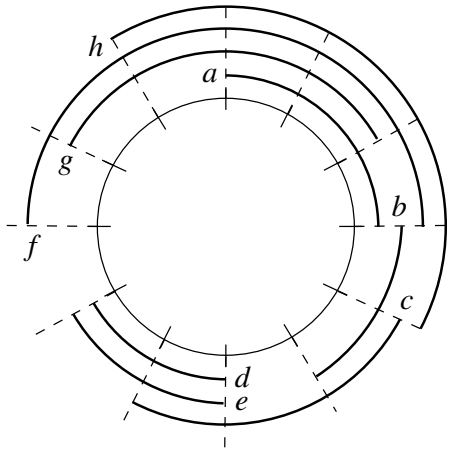


Figure 6.1: Non-redundant triples fitted to an anonymous atomic description

Using the ring diagram of Figure 6.1, we can identify the five distinct kinds of positional configurations that lead to an ambiguous string of complex tokens, as follows:

1. *Overlapping*

   Let us assume that f, g, h, and a belong to the complex tokens F, G, H, and A re-

[1] The reader is referred back to the definitions of **covered by** and **redundant** given in Section 5.4.2.

spectively. Fitted triples f and h have identities that overlap[2], but one could argue that, since the identity of f starts before the identity of h starts and the identity of f finishes before the identity of h finishes, then the substring FH is an unambiguous and adequate description of the presence and ordering of the feature instances represented by f and h, and should therefore form part of the final string. The same argument applies to two other pairs of fitted triples: $\langle g, h \rangle$, and $\langle g, a \rangle$. Consequently, we are claiming that the string should also contain the substrings GH and GA. This is not possible, however, without duplicating token labels (and thereby introducing non-existent feature instances), which is clearly unacceptable. One possible candidate string for describing the presence and ordering of the four feature instances is FGHA, which we might choose because it correctly reflects the relative starting positions of the four identity strings. However, f and g don't overlap and neither do h and a (a fact which is seemingly contradicted by the substrings FG and HA).[3] Also, there is no way of telling that f overlaps h or that g overlaps a. Furthermore, there is nothing in the string which encodes the fact that the four features do actually *overlap*, they could, instead, meet, merge, or even be disjoint.

2. *Starting and ending*

   Assume that f, a, and b belong to the complex tokens F, A, and B, respectively. The identity of a "ends" that of f, i.e., the two identities finish at the same place in the atomic description, with the identity of f starting before that of a. The identity of b fits the two atoms of the atomic description immediately after the identities of f and a. Therefore, the only real choice for the substring describing the presence and ordering of the three feature instances is FAB. The problem here is that the identities of f and b *meet*, as opposed to being separated by the identity of a. This information is lost in the description and, since A occurs in between F and B, perhaps the most intuitive interpretation of the description FAB is that the curve feature represented

[2]The word "overlap" here is to be interpreted as "overlap by more than one atom". We interpret an overlapping by *one* atom as a "merging" (this distinction stems from the triple relations of Section 5.5).

[3]The identities of g and a are, instead, "contained" by the identities of f and h, respectively.

by A occurs in between the features represented by F and B. This contradicts the actual relative placement of the three features.

3. *Containment*

   The purpose of a token-string description is to encode the presence and ordering of curve features, e.g., we would like the string XYZ to reflect the fact that the feature represented by Y occurs *after* the feature represented by X, and that the feature represented by Z occurs after the feature represented by Y. When the identity of a fitted triple *contains* the identity of some other fitted triple, then we get a similar problem to the starting and ending case described previously. Consulting the diagram we can see, for example, that the identity of f contains that of g. The substring FGB seems even more unsatisfactory than FAB, because the identity of g finishes *before* that of f. Also, in principle, the identity of a fitted triple may contain the identities of any number of other fitted triples, thereby increasing the amount of ambiguity.

4. *Equality*

   The two fitted triples d and e are equal, in the sense that they have identities that fit to the same two atoms in the atomic description. Such a configuration is not amenable to token-string description, because we cannot reasonably claim that one of the triples occurs *before* the other. The two possible descriptive sequences, then, DE and ED, are inadequate representations of the presence and placement of the two features.

5. *Special case of meeting and merging*

   A consequence of our atomic descriptions being discrete is that there is a special case of meeting and merging that is problematic, shown by the configuration of identities in Figure 6.2. The problem is similar to that of the starting and ending and containment cases. The sequence XZY, which is the most likely candidate description for the three feature instances, is ambiguous, as the feature represented by Z does not occur in between the features represented by X and Y. The ring diagram contains

one occurrence of the special case, given by h, b, and c.

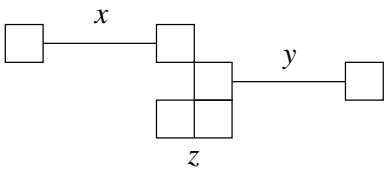


Figure 6.2: The special case of meeting and merging that is problematic

Having identified the kinds of positional configurations that cause ambiguity, an important question that arises is this: what kinds of positional configurations are we left with? The answer is that non-redundant fitted triples are permitted to be disjoint (e.g., as are a and d), to merge (e.g., as do h and b) and to meet (e.g., as do f and b), with the proviso that the positional configuration given by Figure 6.2 (hereafter referred to as **pc5**) does not occur. Our findings can be summarised in the following statement:

> *A mapping from a ring diagram to an unambiguous string of complex tokens is possible if (i) non-redundant fitted triples may only merge, meet, or be disjoint, and (ii) there are no instances of* ***pc5*** *in the diagram.*

Our objective now, then, is to formulate the constraints which ensure that, for a given set of complex tokens, $\mathcal{T}$, token fitting using $\mathcal{T}$ *always* results in a ring diagram that satisfies the two conditions given in the statement.

6.2 Token constraints

We will begin by considering two kinds of constraints. Firstly, the constraints which apply to each individual complex token in a set, and, secondly, the constraints which apply to token *pairs*. In our definition of a *local-feature scheme* (given in Section 6.3) we will incorporate these two kinds of constraints, as well as a final constraint which denies the possibility of **pc5**.

6.2.1 Individual tokens

Consider a complex token, T, which has been fitted to an atomic description, resulting in ring diagram R. The triple elimination stage of the token fitting process ensures that only fitted triples that merge, meet, overlap, are equal, or are disjoint, remain in R. Fitted triples that start or end another fitted triple, or are contained by a fitted triple, are deemed redundant and therefore removed by the token fitting process. This means that we only need to concern ourselves with fitted triples from T that overlap or equal one another. We can eliminate the possibility of equality by enforcing the following condition:

$$(\forall t, t' \in T)(\, t \text{ may-equal } t' \;\Rightarrow\; t = t' \,)$$

This leaves us with the case of fitted triples that overlap, a possibility which we could eliminate by making use of the condition $(\forall t, t' \in T)(\, \neg\, t \text{ may-overlap } t' \,)$. The difference between overlapping and equality, however, is that, whereas fitted triples that are equal are not removed during the triple elimination stage, fitted triples that overlap are, *in certain circumstances*, removed. In other words, overlapping of fitted triples after the triple matching stage of token fitting is acceptable, as long as no overlapping remains at the end of the token fitting process. Therefore, the aforementioned condition is stronger than it needs to be. Furthermore, if it *were* used, then certain tokens, such as curv-seg for example[4], could not be used in token-string descriptions.

Two fitted triples, x and y, that overlap after the triple matching stage, are removed during the triple elimination stage *iff* they are both covered by another fitted triple. It turns out that, if there does not exist a fitted triple that covers *both x and y*, then the fitted triples that together cover x and y must *themselves* overlap. With this in mind, the constraint that we need to enforce can be worded as follows: whenever two fitted triples from T overlap, they must both be covered by a third fitted triple, also from T. As an example, consider curv-seg and two of its triples, $t = [\underline{\text{P}}\,\text{Z}\,\underline{\text{P}}\,\text{Z}\,\underline{\text{P}}\,\text{Z}\,\underline{\text{P}}]$ and $t' = [\underline{\text{P}}\,\text{Z}\,\underline{\text{P}}\,\text{Z}\,\underline{\text{P}}\,\text{Z}\,\underline{\text{P}}\,\text{Z}\,\underline{\text{N}}]$. We have t may-overlap t', as illustrated by the two valid alignments of t and t' shown in Ta-

[4] Refer back to Section 5.2.3 for the specification of curv-seg.

t	P	Z	P	Z	P	Z	P						t may-overlap t'
t'			P	Z	P	Z	P	Z	P	Z	N		t' may-be-overlapped-by t
t	P	Z	P	Z	P	Z	P						t may-overlap t'
t'					P	Z	P	Z	P	Z	P	Z	N	t' may-be-overlapped-by t

Table 6.1: Two of the valid alignments of [P Z P Z P Z P] and [P Z P Z P Z P Z N]

ble 6.1, i.e., there are *two* ways in which t may overlap t'. We need to take account of both ways. The first alignment tells us that, given an atomic description, D, t and t' *will* overlap as fitted triples if D contains one or more instances of the substring P Z P Z P Z P Z P Z N (as explained in Section 5.5). Our constraint requires that a third triple exists in curv-seg which covers both t and t'. The desired triple can be constructed by "merging together" the rows of the alignment matrix (e.g., by moving the first row down over the second) in a manner where the identity cells "dominate" the context cells. The nature of the merging process, and dominance, is make clear in an algorithm presented shortly. In the case of the first alignment of t and t', the constructed triple, c, is [P Z P Z P Z P Z P Z N]. The *second* alignment tells us that t and t' will overlap as fitted triples if D contains one or more instances of the substring P Z P Z P Z P Z P Z P Z N. In this case, the constructed triple, c', is [P Z P Z P Z P Z P Z P Z N]. The two valid alignments of t and t' that are related to t may-overlap t' may be thought of as being *subsumed* by c and c', respectively. The two constructed triples are both in the specification of curv-seg, and so the constraint clearly holds for t may-overlap t'.

We are now in a position to formally specify the overlapping constraint. Let subsumed be a binary predicate taking a set of valid alignments, $\mathcal{A}$, as its first argument and a token specification, T, as its second, such that subsumed holds *iff* every alignment in $\mathcal{A}$ yields a constructed triple that exists in T. The overlapping constraint can be enforced on a token as a whole, then, using the following condition, where *overlap-alignments* is a function returning the set of valid alignments associated with the relationship t may-overlap t':

$$(\forall t, t' \in T)(\, t \text{ may-overlap } t' \;\Rightarrow\; \text{subsumed}(\mathit{overlap\text{-}alignments}(t, t'), T)\,)$$

This condition ensures that, at the end of a token fitting process involving the fitting of T, no two non-redundant fitted triples from T overlap.

The subsumed predicate

An implementation of the subsumed predicate is provided by Algorithm 6.1. The algorithm makes use of the following supporting functions and predicates: $\mathit{width}(A)$ returns the width of alignment matrix A; $\mathit{extract}(A, r, c)$ returns the contents of cell (row r, column c) of matrix A; the three predicates id-atom(A, r, c), lcon-atom(A, r, c), and tcon-atom(A, r, c) hold if the atom in cell (row r, column c) is, respectively, an identity atom, a leading context atom, or a trailing context atom; the addition symbol (+) is used to denote string concatenation and null represents the empty string.

The algorithm contains an outer **for** loop in which each valid alignment is considered in turn (step **1**). For each alignment matrix, a triple is constructed *from* the matrix by simulating a merging of the two rows in the matrix (step 1.2). This triple, specified by the strings L, I, and T, needs to be in X in order for subsumed to hold, and so, if it is not, the algorithm returns false (step 1.3). If all of the constructed triples *are* in X, then the algorithm returns true (step **2**). To illustrate the merging operation carried out by the algorithm, consider the valid alignment of $\underline{\mathsf{P}^0}\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,]\,\mathsf{P}^0$ and $\underline{\mathsf{P}^+}\,\mathsf{P}^0\,[\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\mathsf{Z}^-$ shown in Table 6.2.

	1	2	3	4	5	6	7	8	9
1	$\underline{\mathsf{P}^0}$	$\underline{\mathsf{P}^+}$	P^0	$\underline{\mathsf{P}^-}$	P^0	$\underline{\mathsf{P}^+}$	P^0		
2		$\underline{\mathsf{P}^+}$	P^0	$\underline{\mathsf{P}^-}$	P^0	$\underline{\mathsf{P}^+}$	P^0	$\underline{\mathsf{P}^-}$	Z^-

Table 6.2: An alignment of $\underline{\mathsf{P}^0}\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,]\,\mathsf{P}^0$ and $\underline{\mathsf{P}^+}\,\mathsf{P}^0\,[\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\mathsf{Z}^-$

Inputs: $\mathcal{A}$ (set of valid alignment matrices), X (token specification)
Output: true or false

```
1       for each A ∈ 𝒜 do
1.1         L ← I ← T ← null;
1.2         for i ← 1 to width(A) do
1.2.1           if id-atom(A, 1, i) then
1.2.1.1             I ← I + extract(A, 1, i); next for;
1.2.2           if id-atom(A, 2, i) then
1.2.2.1             I ← I + extract(A, 2, i); next for;
1.2.3           if lcon-atom(A, 1, i) then
1.2.3.1             L ← L + extract(A, 1, i); next for;
1.2.4           if lcon-atom(A, 2, i) then
1.2.4.1             L ← L + extract(A, 2, i); next for;
1.2.5           if tcon-atom(A, 1, i) then
1.2.5.1             T ← T + extract(A, 1, i); next for;
1.2.6           if tcon-atom(A, 2, i) then
1.2.6.1             T ← T + extract(A, 2, i); next for;
1.3         if ⟨L, I, T⟩ ∉ X then return false;
2       return true;
```

Algorithm 6.1: The subsumed predicate

We begin our analysis at step 1.1 of the algorithm, with A as the valid alignment matrix of Table 6.2. Step 1.1 initialises the identity and leading and trailing context strings of the to-be-constructed triple. In step 1.2, the variable i is set to loop from 1 to the width of A, which is 9, i.e., the algorithm traverses the matrix from the leftmost column to the rightmost column. For each column, a check is made on the content-type of each of the two cells in the column, resulting in a single atom being added to either L, I, or T. Identity atoms *dominate* context atoms, since, if *either* cell of a column contains an identity atom, then that atom is added to I and nothing is added to L or T (steps 1.2.1 and 1.2.2). This is the case for columns 2 to 8 in our example. If step 1.2.3 is reached, i.e., a column does not

contain any identity atoms, then either: one of the cells is empty, or both cells contain an atom of the same type.[5] Leading-context atoms are identified by steps 1.2.3 and 1.2.4 and added to L, trailing-context atoms are identified by steps 1.2.5 and 1.2.6 and added to T. When step 1.3 is reached, the triple constructed from the alignment matrix, $\langle L, I, T \rangle$, takes the form depicted in Table 6.3.

$\underline{\mathsf{P}^0}$	$\underline{\mathsf{P}^+}$	P^0	$\underline{\mathsf{P}^-}$	P^0	$\underline{\mathsf{P}^+}$	P^0	$\underline{\mathsf{P}^-}$	Z^-

Table 6.3: The triple constructed from the alignment matrix given in Table 6.2

MAMO categorisation

We are now in a position to bring together the constraints on an individual complex token which ensure that, when the token is fitted to an atomic description, the ring diagram that results only contains non-redundant fitted triples that merge, meet, or are disjoint, as specified by the statement given earlier (p. 121). A complex token, T, is a MAMO ("merge and meet only") token[6] *iff*:

- $(\forall t, t' \in T)(\, t \text{ may-equal } t' \;\Rightarrow\; t = t' \,)$,
- $(\forall t, t' \in T)(\, t \text{ may-overlap } t' \;\Rightarrow\; \text{subsumed}(\mathit{overlap\text{-}alignments}(t, t'), T) \,)$, and
- $(\forall t, t', t'' \in T)(\, t \text{ may-meet } t' \;\Rightarrow\; \neg(t \text{ may-merge-into } t'' \;\wedge\; t'' \text{ may-merge-into } t') \,)$.

The third condition is included so that **pc5** cannot occur with triples from the same token specification. By including the third condition in the MAMO definition, in addition to its inclusion in the definition of a local-feature scheme (as we will see later), we can assert that all MAMO tokens are candidates for inclusion in a local-feature scheme.

[5] Assuming a valid alignment in which the identities are not disjoint, i.e., overlap, merge, or meet.

[6] The name reflects the fact that, after fitting the token to an atomic description, the only relations that can exist between the fitted triples are of the merging and meeting kind (disjointness is not defined as a relation).

An example: determining whether or not curv-seg is a MAMO token

As an example, consider the curv-seg complex token and whether or not it is MAMO. Recall that the specification of curv-seg is $\{\,[(\underline{\mathsf{P}} + \underline{\mathsf{N}})(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{Z}\,\underline{\mathsf{N}})^*]\,\}$. Let $\mathcal{L}$ denote the language of strings given by the regular expression. The first MAMO condition is clearly satisfied. The third condition is also satisfied, because **pc5** requires the existence of a triple with an identity of length two. No such triples exist in curv-seg, which contains only triples with odd-length identities. To show that the second condition is satisfied, we make use of the following properties that characterise the identity string of each triple in curv-seg:

- the atoms $\underline{\mathsf{P}}$ or $\underline{\mathsf{N}}$ appear at all odd-numbered positions in the string, and
- the atom Z appears at all even-numbered positions in the string.

We need to prove that all triple pairs, $\langle t, t' \rangle$, of curv-seg, that may overlap, are necessarily subsumed, i.e., that fitted instances of t and t' that *do overlap* are necessarily removed by triple elimination. In other words, we have to show that a triple covering t and t' must exist in curv-seg. The covering triple (denoted by t'') must be of the form $\langle \Lambda, I, \Lambda \rangle$, because t and t' have leading and trailing contexts that are empty. The identity of t'' must be a string that exists in $\mathcal{L}$, because I is the string obtained by concatenating the identity string of t with that portion of the identity string of t' that isn't overlapped by the identity string of t. Such a concatenation yields a string of atoms that has the two properties listed above. Having these properties is a necessary and sufficient condition for a string to belong to $\mathcal{L}$. The triple t'' must always exist in curv-seg, so the second MAMO condition is also satisfied. We can conclude, therefore, that curv-seg *is* a MAMO token.

6.2.2 Pairs of tokens

The triple elimination stage of token fitting removes fitted triples from a ring diagram on a per-token basis, leaving non-redundant fitted triples. The removal of a fitted triple depends only on other fitted triples from the *same* specification. Therefore, the MAMO

constraints for individual tokens do not prevent fitted triples from *different* specifications from overlapping, starting or ending, or being equal to another fitted triple. In order to ensure that the non-redundant triples that remain after triple elimination only merge, meet, or are disjoint, we also need to constrain the relations that may hold between triples of token *pairs*. Specifically, we need to make sure that the only relations that hold between triples from different specifications are the merging and meeting relations (tr1 to tr4). To do this we define a "non-interference" relation, $\odot$, such that, if $T \odot T'$ holds, then fitted triples of T may only merge, meet, or be disjoint from fitted triples of T':

$$
\begin{aligned}
T \odot T' \quad &\equiv \quad (\forall t \in T, t' \in T')(\, \neg\, t \text{ TRIP-REL } t' \,) \\
t \text{ TRIP-REL } t' \quad &\equiv \quad t \text{ may-overlap } t' \;\vee\; t \text{ may-be-overlapped-by } t' \vee \\
&\qquad\quad t \text{ may-be-started-by } t' \;\vee\; t \text{ may-start } t' \vee \\
&\qquad\quad t \text{ may-contain } t' \;\vee\; t \text{ may-be-contained-by } t' \vee \\
&\qquad\quad t \text{ may-be-ended-by } t' \;\vee\; t \text{ may-end } t' \vee \\
&\qquad\quad t \text{ may-equal } t'
\end{aligned}
$$

In the condition that is used to define the non-interference relation, triple relations tr5 to tr13 need to be explicitly disallowed (using TRIP-REL), because more than one of the thirteen relations may hold between a given pair of triples (although only one of the relations can hold for each valid alignment).[7] The $\odot$ relation is irreflexive, symmetric, and non-transitive.

6.3 Definition of a local-feature scheme

We have formulated constraints for individual complex tokens and pairs of complex tokens, to ensure that token fitting yields fitted triples that only merge, meet, or are disjoint. Using these constraints, we are now in a position to constrain *sets* of complex tokens. A **local-feature scheme** (LFS) is a finite set of complex tokens, $\mathcal{S}$, such that:

- every complex token in $\mathcal{S}$ is specified at the same atomic level,

[7] I.e., "$\neg\, t$ TRIP-REL t" cannot be replaced with "t tr1 $t' \;\vee\; t$ tr2 $t' \;\vee\; t$ tr3 $t' \;\vee\; t$ tr4 t'", because the set of triple relations is not JEPD.

- each complex token in $\mathcal{S}$ is MAMO,
- $(\forall T, T' \in \mathcal{S})(\, T \neq T' \;\Rightarrow\; T \odot T' \,)$, and
- $(\forall t, t', t'' \in \bigcup \mathcal{S})(\, t \text{ may-meet } t' \;\Rightarrow\; \neg(t \text{ may-merge-into } t'' \;\wedge\; t'' \text{ may-merge-into } t') \,)$.

The final condition of the definition rules out any "global" occurrences of **pc5**, i.e., occurrences where the three fitted triples do not all belong to the same token specification. Note that this condition subsumes the analogous **pc5**-related MAMO condition, as t, t', and t'' may all belong to the same token in $\mathcal{S}$. The reason for this apparent redundancy is that, by incorporating the **pc5**-related condition in the definition of a MAMO token, we can make the assertion that every MAMO token is a candidate for inclusion in an LFS.

An LFS, then, provides a set of shape descriptors for representing shapes, such that each description is an unambiguous string of complex tokens encoding the presence and ordering of curve features on the bounding curve of a shape.

6.4 An example: representing polygonal shapes

As an example of LFS specification, and the choosing of a set of complex tokens for describing shapes of a particular kind, consider the representation of polygonal figures using the complex tokens specified in Table 6.4.

Token specification	*Description*
$v_o \;=\; \{\, \underline{\mathsf{Z}}\,[\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}} \,\}$	Outward-pointing vertex
$v_i \;=\; \{\, \underline{\mathsf{Z}}\,[\,\mathsf{U}_{<}\,]\,\underline{\mathsf{Z}} \,\}$	Inward-pointing vertex
$/ \;=\; \{\, [\,\underline{\mathsf{Z}}\,] \,\}$	Straight edge
$> \;=\; \{\, [\,\underline{\mathsf{Z}}\mathsf{U}_{>}\underline{\mathsf{Z}}\,] \,\}$	Outward-pointing "V-segment"
$< \;=\; \{\, [\,\underline{\mathsf{Z}}\mathsf{U}_{<}\underline{\mathsf{Z}}\,] \,\}$	Inward-pointing "V-segment"

Table 6.4: Complex tokens for representing polygonal shapes

All five of the complex tokens are MAMO and each one is therefore a candidate for

inclusion in an LFS. Note that the *complete* set of five tokens does not define an LFS, because of the following token interference:

$$\begin{array}{lcl}
\underline{\mathsf{Z}}\,[\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}} \text{ may-be-contained-by } [\,\underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,] & \Rightarrow & \neg(v_o \odot >) \\
\underline{\mathsf{Z}}\,[\,\mathsf{U}_{<}\,]\,\underline{\mathsf{Z}} \text{ may-be-contained-by } [\,\underline{\mathsf{Z}}\,\mathsf{U}_{<}\,\underline{\mathsf{Z}}\,] & \Rightarrow & \neg(v_i \odot <) \\
[\,\underline{\mathsf{Z}}\,] \text{ may-start } [\,\underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,] \;\wedge\; [\,\underline{\mathsf{Z}}\,] \text{ may-end } [\,\underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,] & \Rightarrow & \neg(/ \odot >) \\
[\,\underline{\mathsf{Z}}\,] \text{ may-start } [\,\underline{\mathsf{Z}}\,\mathsf{U}_{<}\,\underline{\mathsf{Z}}\,] \;\wedge\; [\,\underline{\mathsf{Z}}\,] \text{ may-end } [\,\underline{\mathsf{Z}}\,\mathsf{U}_{<}\,\underline{\mathsf{Z}}\,] & \Rightarrow & \neg(/ \odot <)
\end{array}$$

Let's assume that we would like to specify an LFS capable of distinguishing between polygonal shapes that differ as to the number, and configuration, of inward- and outward-pointing vertices. Therefore, of the $2^5 = 32$ subsets of the five tokens, only four of the subsets are of interest to us: $\{v_o, v_i\}$, $\{v_o, v_i, /\}$, $\{>, <\}$, and $\{>, <, /\}$. The first three subsets each define an LFS, but the fourth subset doesn't. The reason why the fourth subset doesn't is that the non-interference condition doesn't hold between / and either of > and <, since the triple of / may both start and end the triples of > and <. We can choose any of the following LFSs, then, as our representation for polygonal shapes: $POLY_1 = \{v_o, v_i\}$, $POLY_2 = \{v_o, v_i, /\}$, or $POLY_3 = \{>, <\}$. Shown in Figure 6.3 is a simple four-sided polygon, with the level-2 atomic description $\underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,\mathsf{U}_{<}$, together with ring diagrams showing how the five complex tokens are fitted to its description. From the three diagrams in the figure, we can see that the shape is described as $v_o\, v_o\, v_o\, v_i$ under $POLY_1$, $/\, v_o\, /\, v_o\, /\, v_o\, /\, v_i$ under $POLY_2$, and $>\, >\, >\, <$ under $POLY_3$.

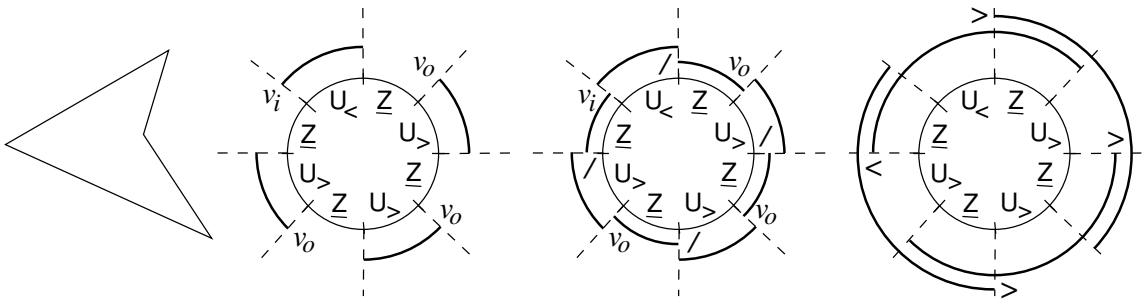


Figure 6.3: Fitting complex tokens to a polygonal shape

6.5 Summary

This chapter has been concerned with the mapping from a ring diagram to an unambiguous string of complex tokens. We started by showing that, without constraining the set of tokens used in the token fitting process, it may not be possible to adequately represent the resultant set of non-redundant fitted triples using a token-string description. We then formulated constraints for individual tokens and pairs of tokens. Using these constraints, we defined a local-feature scheme (LFS) as a set of complex tokens that *can* be used to represent shapes using unambiguous token-string descriptions, i.e., tokens which, when fitted to an atomic description, necessarily yield non-redundant fitted triples that may only merge, meet, or be disjoint.

Later on, in Chapter 8, we show how each of the boundary-based schemes we looked at in Chapter 2 can be defined as an LFS. In the next chapter, we focus our attention on the problem of constructing a token-ordering graph for an LFS.

Chapter 7

Token-ordering graphs

A token-ordering graph visually encodes the syntactic constraints for a set of tokens. It consists of nodes associated with tokens and directed edges representing ordering constraints between tokens. In this chapter, we address the problem of constructing a token-ordering graph from a set of complex tokens defining a local-feature scheme.

7.1 Preliminaries

The purpose of a token-ordering graph (TOG) is to provide a visual encoding of the basic string syntax specific to a particular set of tokens. In Chapter 4, we considered *atomic* TOGs, i.e., graphs associated with sets of atomic tokens taken from the unbounded atomic hierarchy. As mentioned previously, an algorithm for constructing an atomic TOG for a given level of the atomic hierarchy, from a pair of I-I and I-P-I tables, is given in Appendix B. In the present chapter, we are concerned with the construction of TOGs that encode string syntax for sets of *complex* tokens. Specifically, as a result of our analysis of unambiguous token-string description in the previous chapter, we are interested only in those sets of complex tokens that define local-feature schemes (LFSs). Regardless of whether the syntax encoded by a TOG is for a set of atomic tokens or complex tokens, the meaning of a TOG, and its constituent elements, is the same in each case, i.e., nodes

are associated with tokens and edges represent the ordering constraints between tokens. The two kinds *do* differ, however, with respect to their construction. An atomic TOG is constructed for a set of atomic tokens from a pair of transition tables, whose contents are derived from the sequences of qualitative components identified with the atoms themselves (Algorithm 4.1, p. 84). In the case of a set of atomic tokens, the associated TOG may be interpreted as a state-transition graph, since each atom represents an interval or point interpretation of a curve state. The construction of a *non-atomic* TOG is significantly different, stemming from the fact that complex tokens are sets of string-triples, rather than interpreted curve states.

7.1.1 LFS restriction

In order to simplify the construction of a TOG from an LFS, we will restrict ourselves to LFSs whose tokens do not contain any triples with leading or trailing contexts of length greater than one. The principal reason for this restriction is to rule out the possibility of certain token-occurrence dependencies which are not easily represented by a TOG. As an example, consider an LFS that includes the token specifications $\mathsf{A} = \{\, x = \mathsf{Z}\,[\,\underline{\mathsf{P}}\,]\,\mathsf{Z}\,\underline{\mathsf{N}}\,\}$ and $\mathsf{B} = \{\, y = \underline{\mathsf{P}}\,[\,\mathsf{Z}\,]\,\underline{\mathsf{N}}\,\}$. We have x may-by-met-by y, and, since x has a trailing context that extends to the *end* of y's trailing context, when x fits to an atomic description, y necessarily also fits, i.e., each occurrence of "A" in a token-string description must be followed by a "B". Such a dependency requires graph nodes associated with *strings* of token labels, or some other extension to the TOG formalism. More elaborate dependencies are possible with contexts of even greater length. Our restriction to contexts of length one eliminates the possibility of such problematic dependencies. The restriction is considered acceptable because it doesn't rule out *any* of the LFSs considered in this thesis, each of which contains only tokens with triple-contexts of length at most one atom.

7.1.2 Use of LFS_1

In the discussion of TOG construction that follows, we use LFS_1 as an illustrative example. Its specification is given in Table 7.1 and includes the tokens of $POLY_2$ (from Section 6.4), together with the specification for curv-seg (aliased as "C" here).

$$
\begin{array}{rcl}
LFS_1 = \{ \quad v_o & = & \{\,\underline{\mathsf{Z}}\,[\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}}\,\}, \\
v_i & = & \{\,\underline{\mathsf{Z}}\,[\,\mathsf{U}_{<}\,]\,\underline{\mathsf{Z}}\,\}, \\
/ & = & \{\,[\,\underline{\mathsf{Z}}\,]\,\}, \\
\mathsf{C} & = & \{\,[\,(\underline{\mathsf{P}}+\underline{\mathsf{N}})(\mathsf{Z}\,\underline{\mathsf{P}}+\mathsf{Z}\,\underline{\mathsf{N}})^{*}\,]\,\} \quad \}
\end{array}
$$

Table 7.1: The specification of LFS_1

7.1.3 Ring-diagram sets and non-atomic TOGs

In Chapter 5, we showed how the process of token fitting yields a ring diagram from an atomic description and a set of complex tokens. A ring diagram contains an inner ring (the atomic description) and one or more outer rings which show how the complex tokens fit the description. The ring diagram in Figure 7.1, for example, shows the result of fitting the tokens of LFS_1 to the description $\bigcirc\, \mathsf{U}_{>}\, \underline{\mathsf{Z}}\, \underline{\mathsf{P}}\, \mathsf{Z}\, \underline{\mathsf{N}}\, \mathsf{U}_{>}\, \underline{\mathsf{Z}}\, \underline{\mathsf{P}}\, \mathsf{U}_{>}\, \underline{\mathsf{Z}}\, \mathsf{U}_{<}\, \underline{\mathsf{Z}}$. In this case, there are eight equivalent and unambiguous complex-token descriptions that the diagram maps to. Starting at the rightmost $\underline{\mathsf{P}}$ of the inner ring (and going clockwise), we get the description $\bigcirc\, \mathsf{C}\,/\,\mathsf{C}\,/\,v_i\,/\,v_o\,/$. The seven other (equivalent) descriptions are obtained by starting at different positions on the inner ring.

Consider an LFS, $\mathcal{T}$, where each token is specified using atoms at level k in the atomic hierarchy. The token fitting process can be used to fit the tokens of $\mathcal{T}$ to a valid string, s, of the form $\frown X$ or $\bigcirc X$, where X is a syntactically valid string of atomic tokens (i.e., derivable from the level-k atomic TOG) and s is subject to the conditions for validity given in Section 4.3.4. The fitting of $\mathcal{T}$ to s yields a single ring diagram, $R(\mathcal{T}, s)$, which maps to a set of complex-token descriptions, $D(R(\mathcal{T}, s))$. For example, if $\mathcal{T} = LFS_1$ and $s =$

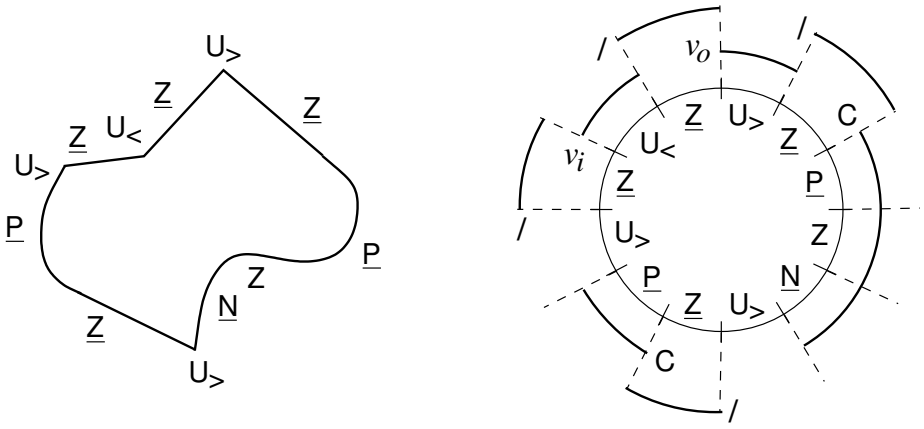


Figure 7.1: Fitting the tokens of LFS_1 to ○ U> Z P Z N U> Z P U> Z U< Z

○ U> Z P Z N U> Z P U> Z U< Z, then we have:

$$D(\,R(\,\{v_o, v_i, /, \mathsf{C}\},\, s\,)\,) \;=\; \{\quad \bigcirc\, v_o \,/\, \mathsf{C} \,/\, \mathsf{C} \,/\, v_i \,/,\quad \bigcirc\, / \,\mathsf{C} \,/\, \mathsf{C} \,/\, v_i \,/\, v_o,\quad \bigcirc\, \mathsf{C} \,/\, \mathsf{C} \,/\, v_i \,/\, v_o \,/,$$
$$\bigcirc\, / \,\mathsf{C} \,/\, v_i \,/\, v_o \,/\, \mathsf{C},\quad \bigcirc\, \mathsf{C} \,/\, v_i \,/\, v_o \,/\, \mathsf{C} \,/,\quad \bigcirc\, / \,v_i \,/\, v_o \,/\, \mathsf{C} \,/\, \mathsf{C},$$
$$\bigcirc\, v_i \,/\, v_o \,/\, \mathsf{C} \,/\, \mathsf{C} \,/,\quad \bigcirc\, / \,v_o \,/\, \mathsf{C} \,/\, \mathsf{C} \,/\, v_i \quad \}$$

Let $\mathcal{S}$ denote the set of *all* valid strings of the form $\frown X$ or $\bigcirc X$ that are derivable from the level-k atomic TOG. From each $s \in \mathcal{S}$, then, we get a ring diagram $R(\mathcal{T}, s)$ by fitting the tokens of $\mathcal{T}$ to s, from which, in turn, we get a set of complex-token descriptions, $D(R(\mathcal{T}, s))$. A set of atomic descriptions, $\mathcal{S}$, then, when considered with respect to a set of complex tokens, $\mathcal{T}$, leads to a set of ring diagrams, $\mathcal{R}$, which, in turn, leads to a set of complex-token descriptions, $\mathcal{D}$:

$$\mathcal{S} \;\rightsquigarrow\; \mathcal{R} = \{\, R(\mathcal{T}, s) \;\mid\; s \in \mathcal{S}) \,\} \;\rightsquigarrow\; \mathcal{D} = \bigcup \{\, D(R(\mathcal{T}, s)) \;\mid\; s \in \mathcal{S} \,\} \tag{7.1}$$

The complex-token strings in $\mathcal{D}$ are exactly those that should be derivable from the non-atomic TOG constructed for the LFS $\mathcal{T}$, with respect to the set of atomic descriptions contained in $\mathcal{S}$. In the sections that follow, we describe one method by which non-atomic TOGs, with different "scopes", can be constructed for a restricted LFS.

7.2 Scope graphs

The construction of a TOG for an LFS is carried out *with respect to* a set of atomic descriptions, because the form of a constructed non-atomic TOG (i.e., the nodes and connections it contains) depends on the set of complex-token strings that should be derivable from it (given by $\mathcal{D}$ in equation 7.1). As equation 7.1 indicates, the content of $\mathcal{D}$ depends, ultimately, on the atomic descriptions contained in $\mathcal{S}$. Up until now, we have assumed that $\mathcal{S}$ is obtained from the level-k atomic TOG (where k is the atomic level at which the tokens of an LFS are specified). In actual fact, $\mathcal{S}$ can be *any* set of atomic descriptions. The purpose of $\mathcal{S}$ is to specify a *scope* (range) of curves that an LFS is capable of representing. A **scope graph** is an atomic TOG that is a "subset" of a level-k atomic TOG, in the sense that the set of atomic descriptions obtainable from it is a subset of the full set of level-k atomic descriptions.

A scope graph is used to define the scope of curves considered in the construction of a TOG for an LFS. We write TOG(L, G) to refer to the TOG constructed for L with respect to the scope of curves given by graph G. In our construction of a TOG for LFS_1, we will use the graph given in Figure 7.2 (G_1), which restricts the scope to curves that do not contain any cusps, or any points where the tangent bearing is defined but the curvature is not.

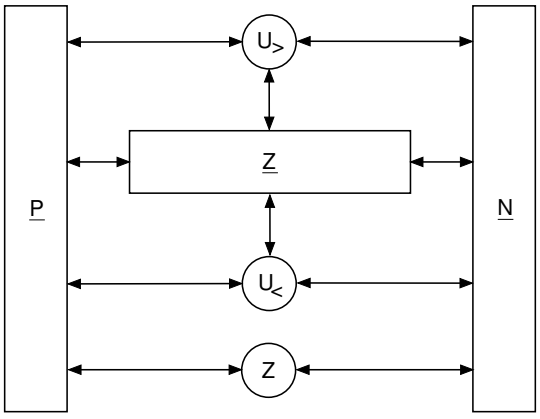


Figure 7.2: An example scope graph, G_1, for LFS_1

7.3 Stages of TOG construction

A non-atomic TOG is constructed for an LFS, using a scope graph, G, in three stages. In the first stage, the LFS is "prepared" so that each of its tokens contains only triples that are compatible with G. An algorithm is provided for converting each triple of each token specification into *explicit-context form*, and for filtering out triples that do not fit any of the atomic descriptions derivable from G. Empty tokens (those whose triples have *all* been filtered out) are removed from the LFS. In the second stage, graph nodes are created for each token of the prepared LFS. A given token may require the creation of more than one graph node, depending on the triples contained in its specification. Each node of a non-atomic TOG is associated with a (possibly infinite) set of triples. In the final stage of construction, the nodes of the graph are connected together, according to the sets of triples associated with each node.

7.3.1 LFS preparation

To simplify TOG construction, we convert triples that contain either one or two empty strings into equivalent triples that are in *explicit-context form*. A triple is in explicit-context form *iff* each of its three component strings (leading context, identity, and trailing context) is non-empty. As an example, consider the token $/ = \{\, [\,\underline{\mathsf{Z}}\,] \,\}$ from LFS_1, which has leading and trailing contexts that are implicit. As stated earlier, in Section 5.2.1, having the empty string as a context is equivalent to specifying *all possible* contexts of length greater than or equal to one. For /, then, we have $\{\, [\,\underline{\mathsf{Z}}\,] \,\} \equiv \{\, x\,[\,\underline{\mathsf{Z}}\,]\,y \mid \mathsf{derivable}(x\,\underline{\mathsf{Z}}\,y, G_1) \,\}$, where $\mathsf{derivable}(S, G)$ is a predicate that holds *iff* the atomic string $S = s_1 s_2 \ldots s_n$ is derivable from graph G, i.e., *iff* the path $s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n$ exists in G. Importantly, because of the redundancy constraint in the definition of a complex token (see Section 5.2), we only need to consider those strings of the form $x\,\underline{\mathsf{Z}}\,y$ where x and y are a single atom in length. In order to accomodate curve end-points (denoted by "$\diamond$"), we assume the temporary existance of two extra nodes in G (labelled "$\diamond$"), one of which is reachable from all interval

nodes and has no outgoing connections, the other having no incoming connections and connected to all interval nodes.[1] A conversion of / into explicit-context form, then, with respect to G_1, yields the following triples:

$\underline{\mathsf{P}}\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{P}}$	$\underline{\mathsf{N}}\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{P}}$	$\mathsf{U}_>\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{P}}$	$\mathsf{U}_<\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{P}}$	$\diamond\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{P}}$
$\underline{\mathsf{P}}\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{N}}$	$\underline{\mathsf{N}}\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{N}}$	$\mathsf{U}_>\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{N}}$	$\mathsf{U}_<\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{N}}$	$\diamond\,[\,\underline{\mathsf{Z}}\,]\,\underline{\mathsf{N}}$
$\underline{\mathsf{P}}\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_>$	$\underline{\mathsf{N}}\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_>$	$\mathsf{U}_>\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_>$	$\mathsf{U}_<\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_>$	$\diamond\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_>$
$\underline{\mathsf{P}}\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_<$	$\underline{\mathsf{N}}\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_<$	$\mathsf{U}_>\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_<$	$\mathsf{U}_<\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_<$	$\diamond\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_<$
$\underline{\mathsf{P}}\,[\,\underline{\mathsf{Z}}\,]\,\diamond$	$\underline{\mathsf{N}}\,[\,\underline{\mathsf{Z}}\,]\,\diamond$	$\mathsf{U}_>\,[\,\underline{\mathsf{Z}}\,]\,\diamond$	$\mathsf{U}_<\,[\,\underline{\mathsf{Z}}\,]\,\diamond$	$\diamond\,[\,\underline{\mathsf{Z}}\,]\,\diamond$

As well as the conversion of triples with implicit context into explicit-context form, the preparation of an LFS includes the removal of triples that are not compatible with the scope graph under consideration. A triple $x\,[\,I\,]\,y$ is incompatible *iff* the string $x\,I\,y$ is not derivable from the scope graph. In the case of LFS_1 and scope graph G_1, there are no triples in the specifications of the tokens in LFS_1 that are incompatible with G_1. This is principally because we have chosen G_1 so that it defines a scope of curves that is well-suited to LFS_1. If we remove the $\mathsf{U}_>$ node from G_1 (and its associated connections), for example, then the triple of v_o would become incompatible with G_1 and therefore removed, resulting in the removal of v_o itself from LFS_1 (since it would contain no triples). A greater effect, in terms of triple removal, is caused by the removal of the node for Z: if Z is removed (while retaining all the other nodes in G_1, including $\mathsf{U}_>$), the infinite specification for C reduces to the 50-element set $\{x\,[\,\underline{\mathsf{P}}\,]\,y,\ x\,[\,\underline{\mathsf{N}}\,]\,y\ \mid\ x, y \in \{\underline{\mathsf{Z}}, \mathsf{Z}, \mathsf{U}_>, \mathsf{U}_<, \diamond\}\}$.

A procedural account of the preparation process is provided by Algorithm 7.1, which converts and removes triples from a set of tokens, $\mathcal{T}$, with respect to a scope graph, G. The algorithm operates as follows. For each $T \in \mathcal{T}$, a temporary result set, A, is initialised, and then each triple contained in the specification of T is considered in turn (step 1.2). Triples that are incompatible with G are simply ignored (step 1.2.1) and thus filtered out of the specification (an action that is effected by step 1.3 or 1.4). A triple that *is* derivable from G

[1] Recall that the description of an open curve must start and finish with interval atoms.

is either (i) converted to explicit-context form if it has empty strings for one or both of its contexts, with the resulting triple(s) copied to A (steps 1.2.1.2 to 1.2.1.4), or, (ii) already in explicit-context form and therefore copied directly to A (step 1.2.1.5). After every triple in T has been considered, A is checked for emptyness. If A *is* empty, then T is removed from $\mathcal{T}$ (step 1.3), otherwise the content of T is replaced by that of A (step 1.4). The output of the algorithm is the modified version of $\mathcal{T}$.

The prepared version of LFS_1, with respect to G_1, is given in Table 7.2. We write $L(G)$ to denote an LFS, L, that has been prepared using scope graph G. The modified version of LFS_1 is therefore referred to as $LFS_1(G_1)$.

Inputs: $\mathcal{T}$ (set of complex tokens), G (scope graph)
Output: a modification of $\mathcal{T}$

1 **for each** $T \in \mathcal{T}$ **do**
1.1 $A \leftarrow \{\}$;
1.2 **for each** $\langle u, I, v\rangle \in T$ **do**
1.2.1 **if** $\mathsf{derivable}(u\, I\, v, G)$ **then**
1.2.1.1 $B \leftarrow \{\, \langle x, I, y\rangle \mid length(x) = length(y) = 1 \ \wedge\ \mathsf{derivable}(x\, I\, y, G)\,\}$;
1.2.1.2 **if** $u = \Lambda \ \wedge\ v = \Lambda$ **then** $A \leftarrow A \cup B$;
1.2.1.3 **if** $u \neq \Lambda \ \wedge\ v = \Lambda$ **then** $A \leftarrow A \cup \{\, \langle x, I, y\rangle \mid \langle x, I, y\rangle \in B \ \wedge\ x = u\,\}$;
1.2.1.4 **if** $u = \Lambda \ \wedge\ v \neq \Lambda$ **then** $A \leftarrow A \cup \{\, \langle x, I, y\rangle \mid \langle x, I, y\rangle \in B \ \wedge\ y = v\,\}$;
1.2.1.5 **if** $u \neq \Lambda \ \wedge\ v \neq \Lambda$ **then** $A \leftarrow A \cup \{\, \langle u, I, v\rangle\,\}$;
1.3 **if** $A = \emptyset$ **then** $\mathcal{T} \leftarrow \mathcal{T} \setminus \{T\}$;
1.4 **else** $T \leftarrow A$;
2 **return** $\mathcal{T}$;

Algorithm 7.1: Preparing an LFS for TOG construction

7.3.2 Node creation

After an LFS has been prepared, so that all of its token specifications contain triples that are in explicit-context form and compatible with the scope graph, the next stage in the

$$
\begin{array}{rcl}
LFS_1(G_1) = \{ \quad v_o & = & \{\, \underline{\mathsf{Z}}\,[\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}}\,\}, \\
v_i & = & \{\, \underline{\mathsf{Z}}\,[\,\mathsf{U}_{<}\,]\,\underline{\mathsf{Z}}\,\}, \\
/ & = & \{\, x\,[\,\underline{\mathsf{Z}}\,]\,y \mid x, y \in \{\underline{\mathsf{P}}, \underline{\mathsf{N}}, \mathsf{U}_{>}, \mathsf{U}_{<}, \diamond\}\,\}, \\
\mathsf{C} & = & \{\, x\,[\,(\underline{\mathsf{P}} + \underline{\mathsf{N}})(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{Z}\,\underline{\mathsf{N}})^*\,]\,y \mid x, y \in \{\underline{\mathsf{Z}}, \mathsf{Z}, \mathsf{U}_{>}, \mathsf{U}_{<}, \diamond\}\,\} \quad \}
\end{array}
$$

Table 7.2: The prepared specification of LFS_1, with respect to G_1

construction process is to create the nodes for the graph. Each node of a non-atomic TOG is associated with a (possibly infinite) set of triples, such that the triples associated with any particular node belong to the same token specification. The task of the node creation stage is to map the set of tokens of an LFS to a set of graph nodes. We need to decide, therefore, the nodes that are required in the graph for each token. One possibility is to assign a node to each individual triple of every complex token. This solution may suffice for simple specifications, containing relatively few triples, but it is clearly inadequate for specifications that contain an *infinite* number of triples: a TOG must contain a finite number of nodes. Even if the number of nodes assigned *is* finite, we don't want to have graphs that contain many more nodes than are necessary. At the other end of the scale, it is not always possible to assign just a single graph node to each token, as evidenced by the construction of *atomic* TOGs (see Appendix B).

Node-connection transitivity

The existence of a connection between two graph nodes depends on the triples associated with the nodes. We have a directed edge (connection) from a node N to a node N' *iff* there exists a triple associated with N that *connects to* a triple associated with N'. There are different types of node connections and these will be defined in due course, here we're just interested in the fact that node connections depend on connections between *triples*. We write $\mathsf{conn}(t, t')$ to denote that triple t connects to triple t'. More formally, then, we have $N \rightarrow N' \;\Leftrightarrow\; (\exists t \in N, t' \in N')(\,\mathsf{conn}(t, t')\,)$, where "$\rightarrow$" denotes a connection from one

node to another.[2] In order to ensure that, if there is a valid path from N to N' ($N \rightarrow N'$) and a valid path from N' to N'' ($N' \rightarrow N''$), then the path $N \rightarrow N' \rightarrow N''$ is also valid, the following node-connection transitivity condition must hold:

$$(\forall x \in N, i, j \in N', y \in N'') \\ [\,(\,\mathsf{conn}(x,i)\ \wedge\ \mathsf{conn}(j,y)\,)\ \Rightarrow\ (\exists k \in N')(\,\mathsf{conn}(x,k)\ \wedge\ \mathsf{conn}(k,y)\,)\,] \qquad (7.2)$$

For node-connection transitivity to be preserved, then, we need to make sure that, if a triple x in N connects to a triple i in N', and a triple j in N' connects to a triple y in N'', then there must exist a triple k in N' such that x connects to k and k connects to y. As an example, consider the tokens $\mathsf{X} = \{\,\underline{\mathsf{P}^+}\,[\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^-}\,\}$, $\mathsf{Y} = \{\,\mathsf{P}^0\,[\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_>,\ \mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,]\,\mathsf{U}_>\,\}$, and $\mathsf{Z} = \{\,\underline{\mathsf{P}^+}\,[\,\mathsf{U}_>\,]\,\underline{\mathsf{N}^-}\,\}$. We write $N_k(\alpha)$ to refer to the kth node of token α. If we assume that a single node is assigned to each token, then we have $N_1(\mathsf{X}) \rightarrow N_1(\mathsf{Y})$ and $N_1(\mathsf{Y}) \rightarrow N_1(\mathsf{Z})$, because the single triple of X connects to the first triple of Y (since $\underline{\mathsf{P}^+}\,[\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^-}$ may-by-met-by $\mathsf{P}^0\,[\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_>$) and the second triple of Y connects to the single triple of Z (since $\mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,]\,\mathsf{U}_>$ may-by-met-by $\underline{\mathsf{P}^+}\,[\,\mathsf{U}_>\,]\,\underline{\mathsf{N}^-}$).[3] However, the path $N_1(\mathsf{X}) \rightarrow N_1(\mathsf{Y}) \rightarrow N_1(\mathsf{Z})$ (the existence of which is implied by $N_1(\mathsf{X}) \rightarrow N_1(\mathsf{Y})$ and $N_1(\mathsf{Y}) \rightarrow N_1(\mathsf{Z})$) is invalid, because the triple of Y that the triple of X connects to is *different* from the triple of Y that connects to the triple of Z (see Figure 7.3), and there does not exist a third triple in Y (the k of equation 7.2) that the triple in X connects to and which connects to the triple of Z. Consequently, it is clear that the node mapping is invalid and that token Y requires two nodes: $N_1(\mathsf{Y}) = \{\,\mathsf{P}^0\,[\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_>\,\}$ and $N_2(\mathsf{Y}) = \{\,\mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,]\,\mathsf{U}_>\,\}$.

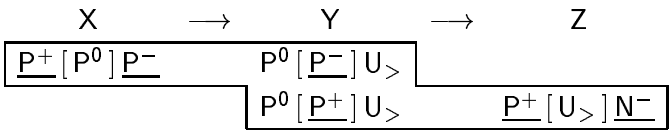


Figure 7.3: An example of an invalid node mapping

[2]For notational convenience, each node is interpreted as a set of associated triples (hence the use of "$\in$").

[3]Two triples that meet give rise to the type of node connection we refer to as "strong" (the identities joining together without any intervening context).

Note that, if an LFS consisted of just X and Y, then we *could* assign a single node to Y, because node-connection transitivity would not be compromised. Strictly speaking, then, the nodes required by a token of an LFS depend on the other tokens in the LFS. To simplify node assignment, however, we adopt a strategy whereby nodes are assigned to a token, T, without the need to consider specifications of tokens other than T. The strategy is based on an analysis of those substrings of a triple that play an active part in the determination of a triple connection.

Starting and finishing substrings

Two triples may connect together in one of three ways: their identities may merge, meet, or be disjoint. The two "tightest" connections, of merging and meeting, are illustrated in Figure 7.4, with shaded squares representing context atoms.

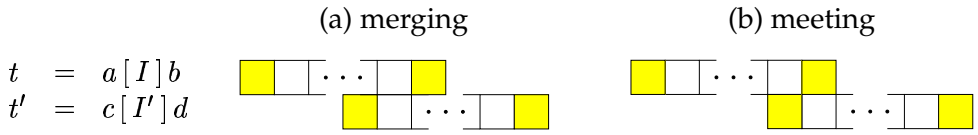


Figure 7.4: The merging and meeting of two triples

In the case of two triples merging, such that $t = a\,[\,I\,]\,b$, $t' = c\,[\,I'\,]\,d$, and the relation t may-merge-into t' holds (Figure 7.4(a)), the three-atom rightmost substring of $a\ I\ b$ matches the three-atom leftmost substring of $c\ I'\ d$. In other words, only the last three atoms of $a\ I'\ b$ and the first three atoms of $c\ I'\ d$ need to be considered in order to determine that t merges with (and thus connects to) t'. In the case of two triples *meeting* (Figure 7.4(b)), we need only consider rightmost and leftmost substrings of length *two*. If two triples connect together, but their identities do not merge *or* meet, i.e., are disjoint, then we *still* need to consider substrings of length two. The reason for this is that, for two triples, t and t', to connect, such that their identities are disjoint, it must not be the case that there exists a triple that "fits" in between the identities of t and t'. To determine this, we need to know

the last atom of the identity of t and the first atom of the identity of t'. Consider, for example, the triples $t = \underline{\mathsf{P}^+}\,[\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^-}$, $t' = \underline{\mathsf{P}^-}\,[\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^+}$, and $t'' = \mathsf{P}^0\,[\,\underline{\mathsf{P}^-}\,]\,\mathsf{P}^0$. The trailing-context atom of t matches the leading-context atom of t', suggesting that t *does* connect to t' such that their identities are disjoint. The triple t'', however, fits in between the identities of t and t', and therefore t does *not* connect to t'. If we only consider substrings of length *one* (i.e., the $\underline{\mathsf{P}^-}$ from t and the $\underline{\mathsf{P}^-}$ from t') then there is not enough information to determine whether t'' fits in between the identities of t and t'. With substrings of length two, we get $\underline{\mathsf{P}^0\,\mathsf{P}^-}$ from t and $\underline{\mathsf{P}^-}\,\mathsf{P}^0$ from t', giving us the information we require. It is not necessary to consider substrings of length *greater* than two, because the leading and trailing contexts of a triple (which we have restricted to be of length one) cannot match any more than one atom of another triple's identity.

In summary, to determine that a triple t connects to a triple t', we only need to consider the rightmost substring of $a\,I\,b$ of length x and the leftmost substring of $c\,I'\,d$ of length x, where $x = 2$ (for connections where the identities meet or are disjoint) or $x = 3$ (for connections where the identities merge). Any remaining atoms of $a\,I\,b$ and $c\,I'\,d$ are immaterial to the determination of triple connections involving t or t'. We refer to the leftmost substring of a triple as a *starting substring* and the rightmost substring as a *finishing substring*. A triple, t, then, may be characterised by the pair $\langle s, f\rangle$, where s is its starting substring and f its finishing substring. So, if $t = \underline{\mathsf{P}^+}\,[\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^-}$, we have $\langle \underline{\mathsf{P}^+}\,\mathsf{P}^0, \mathsf{P}^0\,\underline{\mathsf{P}^-}\rangle$ for $x = 2$ and $\langle \underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}, \underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\rangle$ for $x = 3$.

Preserving node-connection transitivity

We are now in a position to derive the constraint on a set of graph-node triples that ensures node-connection transitivity is preserved. To simplify the analysis that follows, we will restrict our attention to starting and finishing substrings of length two, i.e., those atoms of a triple that play a part in determining the existence of "meeting" and "disjoint" triple connections. Consider a set of triples, T, associated with a node, N. Each triple in T has an associated pair of starting and finishing substrings that characterises it. The set of all such

pairs, T_{ch}, gives us a finite *characterisation* of T, i.e., T_{ch} will be finite, even if T is not. The token, C, of LFS_1, for example, is characterised by the following set of pairs:

$$\begin{aligned}\mathsf{C}_{ch} = \{\, \langle s, f\rangle \;\mid\; s \;&\in\; \{\underline{\mathsf{Z}}\,\underline{\mathsf{P}},\ \mathsf{Z}\,\underline{\mathsf{P}},\ \mathsf{U}_{>}\,\underline{\mathsf{P}},\ \mathsf{U}_{<}\,\underline{\mathsf{P}},\ \diamond\,\underline{\mathsf{P}},\ \underline{\mathsf{Z}}\,\underline{\mathsf{N}},\ \mathsf{Z}\,\underline{\mathsf{N}},\ \mathsf{U}_{>}\,\underline{\mathsf{N}},\ \mathsf{U}_{<}\,\underline{\mathsf{N}},\ \diamond\,\underline{\mathsf{N}}\} \wedge \\ f \;&\in\; \{\underline{\mathsf{P}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{P}}\,\mathsf{Z},\ \underline{\mathsf{P}}\,\mathsf{U}_{>},\ \underline{\mathsf{P}}\,\mathsf{U}_{<},\ \underline{\mathsf{P}}\,\diamond,\ \underline{\mathsf{N}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{N}}\,\mathsf{Z},\ \underline{\mathsf{N}}\,\mathsf{U}_{>},\ \underline{\mathsf{N}}\,\mathsf{U}_{<},\ \underline{\mathsf{N}}\,\diamond\}\ \}\end{aligned}$$

Different pairs of a characterisation set may contain the same starting or finishing substrings (but not the same starting *and* finishing substrings, of course). For example, the pairs $\langle \mathsf{Z}\,\underline{\mathsf{P}},\ \underline{\mathsf{N}}\,\diamond\rangle$ and $\langle \mathsf{Z}\,\underline{\mathsf{P}},\ \underline{\mathsf{P}}\,\underline{\mathsf{Z}}\rangle$ in C_{ch} share the same starting substring. Let $\mathit{distinct\text{-}ss}(X)$ and $\mathit{distinct\text{-}fs}(X)$ be functions that return, respectively, the number of *distinct* starting and finishing substrings in the characterisation set X. For C_{ch}, we have $\mathit{distinct\text{-}ss}(\mathsf{C}_{ch}) = 10$ and $\mathit{distinct\text{-}fs}(\mathsf{C}_{ch}) = 10$. Given a characterisation set, X, if the number of distinct starting substrings in X, multiplied by the number of distinct finishing substrings in X, equals the total number of *pairs* in X, then it must be the case that every $\langle s, f\rangle$ *combination* of distinct starting (s) and finishing (f) substrings is included in X. This is precisely the condition required to ensure that a set of triples preserves node-connection transitivity:

Definition 7.1 *A set of triples, T, is* **transitivity-preserving** iff *its characterisation set, T_{ch}, is such that $|T_{ch}| = \mathit{distinct\text{-}ss}(T_{ch}) \times \mathit{distinct\text{-}fs}(T_{ch})$.*

If a complex token, T, in a prepared LFS, is a transitivity-preserving set of triples, then T can be assigned a single graph node, otherwise, T must be partitioned into a finite number of subsets, such that each subset is transitivity-preserving and associated with a separate graph node. Note that, if T consists of a *single* triple, then it is necessarily transitivity-preserving, since $|T_{ch}| = 1 = 1 \times 1$. The complex token C, even though it consists of an infinite number of triples, requires only a single node, because $|\mathsf{C}_{ch}| = 100 = \mathit{distinct\text{-}ss}(\mathsf{C}_{ch}) \times \mathit{distinct\text{-}fs}(\mathsf{C}_{ch})$. The token Y, in our earlier example, requires two nodes, because $\mathsf{Y}_{ch} = \{\, \langle \mathsf{P}^0\,\underline{\mathsf{P}^-},\ \underline{\mathsf{P}^-}\,\mathsf{U}_{>}\rangle, \langle \mathsf{P}^0\,\underline{\mathsf{P}^+},\ \underline{\mathsf{P}^+}\,\mathsf{U}_{>}\rangle\,\}$, $\mathit{distinct\text{-}ss}(\mathsf{Y}_{ch}) = 2$, $\mathit{distinct\text{-}fs}(\mathsf{Y}_{ch}) = 2$, and $2 \neq 2 \times 2$. The algorithm given in Appendix B for partitioning a set of context pairs (Algorithm B.1, p. 228) can also be used to indirectly partition a set of triples, by supplying, as input, the corresponding characterisation set, so that the function $\mathit{lcon}(p)$ returns the starting substring of pair p and the function $\mathit{tcon}(p)$ returns the

finishing substring. As an example, consider a prepared LFS that includes a token for representing points of zero curvature where the rate of change of curvature is defined: $\mathsf{Zero} = \{\, \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^-\,]\,\underline{\mathsf{N}^-},\ \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{P}^+},\ \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{N}^-},\ \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^+\,]\,\underline{\mathsf{P}^+},\ \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{N}^-},\ \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{P}^+}\,\}$. The corresponding characterisation set is as follows:

$$\begin{aligned}\mathsf{Zero}_{ch} = \{\ &\langle \underline{\mathsf{P}^-}\,\mathsf{Z}^-,\ \mathsf{Z}^-\,\underline{\mathsf{N}^-}\rangle, \langle \underline{\mathsf{P}^-}\,\mathsf{Z}^0,\ \mathsf{Z}^0\,\underline{\mathsf{P}^+}\rangle, \langle \underline{\mathsf{P}^-}\,\mathsf{Z}^0,\ \mathsf{Z}^0\,\underline{\mathsf{N}^-}\rangle,\\ &\langle \underline{\mathsf{N}^+}\,\mathsf{Z}^+,\ \mathsf{Z}^+\,\underline{\mathsf{P}^+}\rangle, \langle \underline{\mathsf{N}^+}\,\mathsf{Z}^0,\ \mathsf{Z}^0\,\underline{\mathsf{N}^-}\rangle, \langle \underline{\mathsf{N}^+}\,\mathsf{Z}^0,\ \mathsf{Z}^0\,\underline{\mathsf{P}^+}\rangle\ \}\end{aligned}$$

Zero_{ch} contains four distinct starting substrings and four distinct finishing substrings, so Zero requires more than one node, because $|\mathsf{Zero}_{ch}| = 6 \neq 4 \times 4$. The output of Algorithm B.1, with Zero_{ch} as input, is a partition of Zero_{ch}, such that each subset of the partition characterises a subset of Zero's triples that *is* transitivity-preserving, as listed in Table 7.3. The Zero token requires three nodes, then, one for each subset of the partition:

$$\begin{aligned} N_1(\mathsf{Zero}) &= \{\, \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^-\,]\,\underline{\mathsf{N}^-}\,\},\\ N_2(\mathsf{Zero}) &= \{\, \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{P}^+},\ \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{N}^-},\ \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{N}^-},\ \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{P}^+}\,\},\ \text{and}\\ N_3(\mathsf{Zero}) &= \{\, \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^+\,]\,\underline{\mathsf{P}^+}\,\}. \end{aligned}$$

Subset of partition of Zero_{ch}	*Characterised triple(s)*
$\{\, \langle \underline{\mathsf{P}^-}\,\mathsf{Z}^-,\ \mathsf{Z}^-\,\underline{\mathsf{N}^-}\rangle\,\}$	$\{\, \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^-\,]\,\underline{\mathsf{N}^-}\,\}$
$\{\, \langle \underline{\mathsf{P}^-}\,\mathsf{Z}^0,\ \mathsf{Z}^0\,\underline{\mathsf{P}^+}\rangle, \langle \underline{\mathsf{P}^-}\,\mathsf{Z}^0,\ \mathsf{Z}^0\,\underline{\mathsf{N}^-}\rangle,$ $\langle \underline{\mathsf{N}^+}\,\mathsf{Z}^0,\ \mathsf{Z}^0\,\underline{\mathsf{N}^-}\rangle, \langle \underline{\mathsf{N}^+}\,\mathsf{Z}^0,\ \mathsf{Z}^0\,\underline{\mathsf{P}^+}\rangle\,\}$	$\{\, \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{P}^+},\ \underline{\mathsf{P}^-}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{N}^-},\ \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{N}^-},\ \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^0\,]\,\underline{\mathsf{P}^+}\,\}$
$\{\, \langle \underline{\mathsf{N}^+}\,\mathsf{Z}^+,\ \mathsf{Z}^+\,\underline{\mathsf{P}^+}\rangle\,\}$	$\{\, \underline{\mathsf{N}^+}\,[\,\mathsf{Z}^+\,]\,\underline{\mathsf{P}^+}\,\}$

Table 7.3: The three subsets resulting from a partitioning of Zero_{ch}

In the above analysis, we have assumed starting and finishing substrings of length two. Exactly the same reasoning also applies to substrings of length three. Our choice of substring length for node creation depends on the nature of the LFS we are considering. If an LFS contains tokens that yield non-redundant fitted triples that only meet or are disjoint (i.e., do not merge), then we can use substrings of length two.[4] If, on the other hand, an LFS

[4] We could use substrings of length three for such an LFS, but this would, in general, result in more nodes being created than are necessary.

contains tokens which yield non-redundant fitted triples that *do* merge, then we must use substrings of length three. We shall refer to an LFS of the former kind as a "no-merging" LFS and one of the latter kind as a "merging" LFS. Most of the LFSs considered in the remainder of this thesis are of the "no-merging" variety, and so, unless stated otherwise, an LFS is assumed to be of this kind and we will use starting and finishing substrings of length two for node creation.

Node assignment algorithm

The method of assigning nodes to tokens, described above, is formalised as Algorithm 7.2. The algorithm takes as input a set of complex tokens, $\mathcal{T}$, and returns a set of graph nodes, $\mathcal{N}$. Its operation is as follows. Each token, $T \in \mathcal{T}$, is considered in turn (step **2**). First, the characterisation set for T is generated, by extracting the starting and finishing substrings of length x from each triple in T, with the result stored in A (steps 2.1 and 2.2). Next, the characterisation set, A, is partitioned using Algorithm B.1, with the result assigned to $\mathcal{P}$ (step 2.3). A node is then created for each subset, P, of the partition of A (step 2.5.2). Each node is assigned a *pair of sets*, as opposed to a set of triples. The first element of the pair is the set of distinct starting substrings of P, and the second element is the set of distinct finishing substrings of P. This is the minimal amount of information required in order to characterise (and thus recover) the associated set of transitivity-preserving triples. The nodes created for each $T \in \mathcal{T}$ are added to the result set, $\mathcal{N}$ (step 2.5.4).

An application of the algorithm, with $\mathcal{T} = LFS_1(G_1)$ as input, yields the set of nodes $\mathcal{N}(LFS_1(G_1))$, listed in Table 7.4. Figure 7.5 shows a partial version of TOG(LFS_1, G_1) after the node creation stage and before the addition of node connections. In keeping with the convention adopted for *atomic* TOGs, a node is depicted as circular if all of its associated triples have identities that are single point atoms, otherwise it is depicted as rectangular. The nodes for v_o and v_i are therefore shown as circles and the nodes for / and C are shown as rectangles.

Inputs: $\mathcal{T}$ (set of complex tokens), x (substring length)
Output: $\mathcal{N}$ (set of graph nodes)

1 $\mathcal{N} \leftarrow \{\}$;
2 **for each** $T \in \mathcal{T}$ **do**
2.1 $A \leftarrow \{\}$;
2.2 **for each** $\langle u, I, v \rangle \in T$ **do**
2.2.1 $A \leftarrow A \cup \{ \langle \mathit{leftmost\text{-}substr}(u\, I\, v, x),\ \mathit{rightmost\text{-}substr}(u\, I\, v, x) \rangle \}$;
2.3 $\mathcal{P} \leftarrow \mathit{partition}(A)$;
2.4 $i \leftarrow 0$;
2.5 **for each** $P \in \mathcal{P}$ **do**
2.5.1 $i \leftarrow i + 1$;
2.5.2 create $N_i(T)$;
2.5.3 $N_i(T) \leftarrow \langle \{ s \mid \langle s, \alpha \rangle \in P \}, \{ f \mid \langle \beta, f \rangle \in P \} \rangle$;
2.5.4 $\mathcal{N} \leftarrow \mathcal{N} \cup \{N_i(T)\}$;
3 **return** $\mathcal{N}$;

Algorithm 7.2: Creation of the graph nodes for a restricted and prepared LFS

$$
\begin{aligned}
\mathcal{N}(LFS_1(G_1)) \;=\; \{\; & N_1(v_o) = \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{>}\},\ \{\mathsf{U}_{>}\,\underline{\mathsf{Z}}\} \,\rangle, \\
& N_1(v_i) = \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{<}\},\ \{\mathsf{U}_{<}\,\underline{\mathsf{Z}}\} \,\rangle, \\
& N_1(/) \;= \langle\, \{\underline{\mathsf{P}}\,\mathsf{Z},\ \underline{\mathsf{N}}\,\mathsf{Z},\ \mathsf{U}_{>}\,\underline{\mathsf{Z}},\ \mathsf{U}_{<}\,\underline{\mathsf{Z}},\ \diamond\underline{\mathsf{Z}}\},\ \{\mathsf{Z}\,\underline{\mathsf{P}},\ \mathsf{Z}\,\underline{\mathsf{N}},\ \underline{\mathsf{Z}}\,\mathsf{U}_{>},\ \underline{\mathsf{Z}}\,\mathsf{U}_{<},\ \underline{\mathsf{Z}}\diamond\} \,\rangle, \\
& N_1(\mathsf{C}) \;= \langle\, \{\underline{\mathsf{Z}}\,\underline{\mathsf{P}},\ \mathsf{Z}\,\underline{\mathsf{P}},\ \mathsf{U}_{>}\,\underline{\mathsf{P}},\ \mathsf{U}_{<}\,\underline{\mathsf{P}},\ \diamond\underline{\mathsf{P}},\ \underline{\mathsf{Z}}\,\underline{\mathsf{N}},\ \mathsf{Z}\,\underline{\mathsf{N}},\ \mathsf{U}_{>}\,\underline{\mathsf{N}},\ \mathsf{U}_{<}\,\underline{\mathsf{N}},\ \diamond\underline{\mathsf{N}}\}, \\
& \qquad \{\underline{\mathsf{P}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{P}}\,\mathsf{Z},\ \underline{\mathsf{P}}\,\mathsf{U}_{>},\ \underline{\mathsf{P}}\,\mathsf{U}_{<},\ \underline{\mathsf{P}}\diamond,\ \underline{\mathsf{N}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{N}}\,\mathsf{Z},\ \underline{\mathsf{N}}\,\mathsf{U}_{>},\ \underline{\mathsf{N}}\,\mathsf{U}_{<},\ \underline{\mathsf{N}}\diamond\} \,\rangle\; \}
\end{aligned}
$$

Table 7.4: The set of graph nodes required for $LFS_1(G_1)$

Figure 7.5: A partial version of TOG(LFS_1, G_1) containing only the nodes

7.3.3 Connecting the nodes

The final stage of the TOG construction process involves connecting together the nodes of the graph. Consider two nodes, N and N' (not necessarily distinct). We define three types of connections, of different *strengths*, that can exist from N to N': *strong*, *weak*, and *remote*. The connection types are not mutually exclusive, i.e., more than one type may exist from N to N'. The existence of each type of connection, for a given pair of nodes, is determined by considering the set of triples associated with each node. Recall from Algorithm 7.2 that, for reasons of efficiency, rather than assigning to a node its *actual* set of associated triples, X (which may be infinite), we assign a pair of (necessarily finite) sets, $\langle S, F \rangle$, where S is the set of distinct starting substrings in X and F is the set of distinct finishing substrings in X. For $N_1(/)$ of TOG(LFS_1, G_1), for example, we have $S = \{\underline{\mathsf{P}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{N}}\,\underline{\mathsf{Z}},\ \mathsf{U}_{>}\,\underline{\mathsf{Z}},\ \mathsf{U}_{<}\,\underline{\mathsf{Z}},\ \diamond\,\underline{\mathsf{Z}}\}$ and $F = \{\underline{\mathsf{Z}}\,\underline{\mathsf{P}},\ \underline{\mathsf{Z}}\,\underline{\mathsf{N}},\ \underline{\mathsf{Z}}\,\mathsf{U}_{>},\ \underline{\mathsf{Z}}\,\mathsf{U}_{<},\ \underline{\mathsf{Z}}\,\diamond\}$, which characterise the 25-element set of triples associated with $N_1(/)$: $X = \{\, x\,[\,\underline{\mathsf{Z}}\,]\,y \mid x, y \in \{\underline{\mathsf{P}}, \underline{\mathsf{N}}, \mathsf{U}_{>}, \mathsf{U}_{<}, \diamond\}\,\}$. The existence of one or more of the connection types from $N = \langle S, F \rangle$ to $N' = \langle S', F' \rangle$ is determined by considering the finishing substrings, F, of N, and the starting substrings, S', of N', as follows (where $\xrightarrow{S}$, $\xrightarrow{W}$, and $\xrightarrow{R}$ denote strong, weak, and remote connections respectively):[5]

$$N \xrightarrow{S} N' \quad \Leftrightarrow \quad (\exists f \in F, s' \in S')(\,\mathsf{strong\text{-}conn}(f, s')\,) \tag{7.3}$$

$$N \xrightarrow{W} N' \quad \Leftrightarrow \quad (\exists f \in F, s' \in S')(\,\mathsf{weak\text{-}conn}(f, s', \mathcal{T})\,) \tag{7.4}$$

$$N \xrightarrow{R} N' \quad \Leftrightarrow \quad (\exists f \in F, s' \in S')(\,\mathsf{remote\text{-}conn}(f, s', \mathcal{T}, G)\,) \tag{7.5}$$

The sets F and S' may contain substrings characterising triples that can *end* or *begin* an open curve, i.e., $f \in F$ may end with "$\diamond$" or $s \in S'$ may begin with "$\diamond$". In the S and F sets assigned to $N_1(/)$, for example, S contains $\diamond\,\underline{\mathsf{Z}}$ and F contains $\underline{\mathsf{Z}}\,\diamond$. When node connections are being determined, such starting and finishing substrings are simply ignored, because there cannot be a connection *to* a triple that *starts* a curve, or *from* a triple that *ends* a curve. We are now in a position to define the predicates strong-conn, weak-conn, and remote-conn.

[5] An inspection of equations 7.4 and 7.5 reveals that the determination of a weak connection requires a reference to $\mathcal{T}$ (the set of LFS tokens), and the determination of a remote connection requires a reference to $\mathcal{T}$ and the scope graph, G.

Strong connections

Consider two triples, t and t', characterised by $\langle s, f \rangle$ and $\langle s', f' \rangle$, respectively. The former triple *strongly* connects to the latter, i.e., $\mathsf{strong\text{-}conn}(f, s')$ holds, *iff* t may merge into, or may be met by, t'. In other words, two triples strongly connect if they may join together without their identities being separated by one or more non-identity atoms. In terms of starting and finishing substrings, then, t strongly connects to t' *iff* $f = s'$. For a no-merging LFS, f and s' will be of length two and for a merging LFS, f and s' will be of length three. In the former case, f and s' being equal implies a meeting of triples and, in the latter case, a merging, as shown by the alignment diagrams of Figure 7.6.

$$\mathsf{strong\text{-}conn}(f, s') \quad \Leftrightarrow \quad f = s' \tag{7.6}$$

As an example, consider $N_1(v_o)$ of TOG(LFS_1, G_1). There exists a strong connection from $N_1(v_o)$ to $N_1(/)$, since the single distinct finishing substring assigned to $N_1(v_o)$, $\mathsf{U}_{>}\,\underline{\mathsf{Z}}$, is also an element of the set of distinct starting substrings assigned to $N_1(/)$. This is the only strong connection *from* $N_1(v_o)$, since $\mathsf{U}_{>}\,\underline{\mathsf{Z}}$ is not an element of the sets of distinct starting substrings assigned to $N_1(v_i)$ and $N_1(\mathsf{C})$. There are five other strong connections in the graph: $N_1(/) \xrightarrow{S} N_1(v_o)$, $N_1(/) \xrightarrow{S} N_1(v_i)$, $N_1(v_i) \xrightarrow{S} N_1(/)$, $N_1(/) \xrightarrow{S} N_1(\mathsf{C})$, and $N_1(\mathsf{C}) \xrightarrow{S} N_1(/)$. The completed TOG is shown later on, in Section 7.4, with strong connections drawn as solid lines.

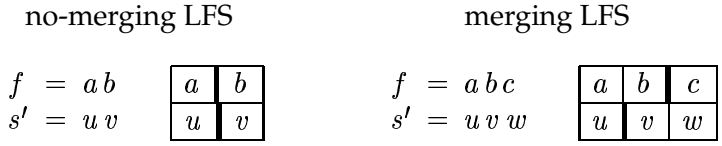


Figure 7.6: Substring equality implies a meeting or a merging

Weak connections

A triple, t, *weakly* connects to a triple, t', where t is characterised by $\langle s, f \rangle$ and t' by $\langle s', f' \rangle$, *iff* (i) the single trailing-context atom of t is the same as the single leading-context atom of t', (i.e., the triples join together such that their identities are separated by a single atom), and (ii) there does not exist a triple, from any token specification of the prepared LFS, that fits in between the identities of t and t'. Let $f = a\,b\,c$, $s' = u\,v\,w$, and $\mathcal{T}$ be the set of tokens of the prepared LFS. If the prepared LFS is of the no-merging kind, then $a = w = \Lambda$. The predicate weak-conn, then, is defined as follows:

$$\text{weak-conn}(f = a\,b\,c, s' = u\,v\,w, \mathcal{T}) \quad \Leftrightarrow \quad c = u \ \wedge \ \langle b, c, v \rangle \notin \bigcup \mathcal{T} \tag{7.7}$$

The alignment diagram for weak connections is shown in Figure 7.7. An example of a weak connection in TOG(LFS_1, G_1) is given by $N_1(/)$ and $N_1(\mathsf{C})$. There exists a weak connection from $N_1(/)$ to $N_1(\mathsf{C})$, because $f = b\,c = \underline{\mathsf{Z}}\,\mathsf{U}_>$ is a distinct finishing substring of $N_1(/)$, $s' = u\,v = \mathsf{U}_>\,\underline{\mathsf{P}}$ is a distinct starting substring of $N_1(\mathsf{C})$, $c = u$, and $\langle b, c, v \rangle$ (the triple $\underline{\mathsf{Z}}\,[\,\mathsf{U}_>\,]\,\underline{\mathsf{P}}$) is not contained in any token specification of $LFS_1(G_1)$. Two other weak connections exist in the graph: $N_1(\mathsf{C}) \overset{W}{\rightarrow} N_1(/)$ and $N_1(\mathsf{C}) \overset{W}{\rightarrow} N_1(\mathsf{C})$. The weak connections reflect the fact that v_o and v_i represent only outward- and inward-pointing angles that are adjoined, on both sides, by straight-line segments, i.e., v_o and v_i were specified so as to represent vertices of *polygonal* shapes.[6] The scope graph, however, since it is not restricted to polygonal shapes, generates descriptions of curves that contain *other* kinds of angles, not represented by any token in LFS_1.

In a TOG, weak connections are drawn as dashed lines (see Figure 7.9).

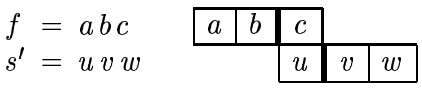


Figure 7.7: Alignment diagram for weak connections

[6] Recall that the complex tokens v_o, v_i, and / of LFS_1 were taken from $POLY_2$.

Remote connections

For weak connections, the identities of two triples are separated by a *single* non-identity atom. With a *remote* connection, two triples join together such that their identities are separated by *more* than a single atom, i.e., a string of atoms, S, of length two or greater. A remote connection only exists, however, if there exists an S such that there does *not* exist a triple, from any token specification of the prepared LFS, that fits S. First, we define what it means for a triple to *fit* a string of atomic tokens:

Definition 7.2 *The triple* $\langle l, I, t\rangle$ **fits** *the string of atomic tokens,* S, iff $\mathsf{substring}(l\,I\,t, S)$ *holds, where* $\mathsf{substring}(x, y)$ *is a predicate that holds* iff x *is a substring of* y.[7]

Consider, then, two triples, t and t', characterised by $\langle s, f\rangle$ and $\langle s', f'\rangle$ respectively. Let $f = a\,b\,c$, $s' = u\,v\,w$, $\mathcal{T}$ be the set of tokens of the prepared LFS, and G the related scope graph. If the LFS is of the no-merging variety, then we have $a = w = \Lambda$. The identities of t and t' may be separated by exactly two atoms *iff* the path $b \to c \to u \to v$ exists in G (i.e., *iff* the string $b\,c\,u\,v$ is derivable from G), where b is the last atom of the identity of t and v is the first atom of the identity of t'. Note that b and v need to be included in the path (we can't just check for the presence of $c \to u$), due to the fact that certain atoms in the scope graph may be represented by more than one node. As an example, consider the case where $f = bc = \underline{\mathsf{Z}}\,\mathsf{U_c}$ and $s' = u\,v = \underline{\mathsf{Z}}\,\underline{\mathsf{P}}$, with G the level-2 atomic TOG (p. 88). The context atoms, c and u, do connect together, since $\mathsf{U_c} \to \underline{\mathsf{Z}}$ exists in G. However, the path $\underline{\mathsf{Z}} \to \mathsf{U_c} \to \underline{\mathsf{Z}} \to \underline{\mathsf{P}}$ does *not* exist in G, because a $\mathsf{U_c}$ atom cannot be flanked, on *both* sides, by a $\underline{\mathsf{Z}}$ atom. In general, the identities of t and t' can be separated by all strings of the form $c\,X\,u$, where $length(X) \geq 0$ and $b\,c\,X\,u\,v$ is derivable from G (see Figure 7.8). Triple t remotely connects to triple t' *iff* there exists an X such that no triple from any specification of the LFS fits the string $b\,c\,X\,u\,v$:

$$\mathsf{remote\text{-}conn}(f = a\,b\,c, s' = u\,v\,w, \mathcal{T}, G) \;\Leftrightarrow\; (\exists X)$$
$$(\,\mathsf{derivable}(b\,c\,X\,u\,v, G) \;\wedge\; \neg(\exists\,\langle l, I, t\rangle \in \bigcup\mathcal{T})(\,\mathsf{substring}(l\,I\,t,\; b\,c\,X\,u\,v)\,)\,) \qquad (7.8)$$

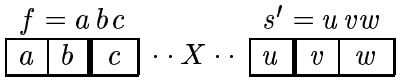


Figure 7.8: Alignment diagram for remote connections

In a TOG, remote connections are drawn as dotted lines. There are *no* connections of the remote kind in TOG(LFS_1, G_1) (see Figure 7.9). We can establish this fact, *without* considering starting and finishing substrings, as follows. Every occurrence of Z, P, and N, in an atomic description obtained from G_1, is represented by one of the tokens of LFS_1. In particular, occurrences of Z are represented by /, and occurrences of P and N are represented by C. Since Z, P, and N account for all of the interval atoms in G_1, it follows that all atoms in a description that are not represented (i.e., not associated with the identity of any fitted triple) are point atoms. No two point atoms can occur consecutively in a description, therefore, all unrepresented portions of a description must be of length one. Consequently, only strong and weak connections are possible, since a remote connection requires the existence of an unrepresented portion of length *greater* than one.

To establish the non-existence of remote connections computationally, we need to consider starting and finishing substrings. As an example, consider the determination of $N_1(/) \xrightarrow{R} N_1(\mathsf{C})$, which exists *iff* remote-conn holds when given a finishing substring, f, of $N_1(/)$, and a starting substring, s', of $N_1(\mathsf{C})$. We will consider just one of the candidate pairs: $\langle f = \mathsf{Z}\,\mathsf{U}_>, s' = \mathsf{Z}\,\mathsf{P}\rangle$. For f and s', string X of equation 7.8 cannot be empty, because $\mathsf{U}_> \rightarrow \mathsf{Z}$ does not exist in G_1. If X is of length one, then X must be either P or N, giving us $\alpha = b\,c\,\mathsf{P}\,u\,v = \mathsf{Z}\,\mathsf{U}_>\,\mathsf{P}\,\mathsf{Z}\,\mathsf{P}$ and $\beta = b\,c\,\mathsf{N}\,u\,v = \mathsf{Z}\,\mathsf{U}_>\,\mathsf{N}\,\mathsf{Z}\,\mathsf{P}$. However, triples $\langle \mathsf{U}_>, \mathsf{P}, \mathsf{Z}\rangle$ and $\langle \mathsf{U}_>, \mathsf{N}, \mathsf{Z}\rangle$ of C fit α and β respectively. Therefore X, if it *does* exist, must be of length two or more. Upon inspection of G_1, it is clear that the string $b\,c\,X$, where X is greater than, or equal to, two atoms in length, must begin with one of the following twelve substrings:

[7] Including the case where $x = y$.

$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{P}}\,\mathsf{U}_>$	$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{Z}}\,\mathsf{U}_>$	$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{N}}\,\mathsf{U}_>$
$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{P}}\,\mathsf{Z}$	$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{Z}}\,\mathsf{P}$	$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{N}}\,\mathsf{Z}$
$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{P}}\,\mathsf{U}_<$	$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{Z}}\,\mathsf{N}$	$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{N}}\,\mathsf{U}_<$
$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{P}}\,\mathsf{Z}$	$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{Z}}\,\mathsf{U}_<$	$\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{N}}\,\mathsf{Z}$

For each string in the first or third column, there exists a triple in C that fits to it: $\langle \mathsf{U}_>, \underline{\mathsf{P}}, \mathsf{U}_> \rangle$ fits $\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{P}}\,\mathsf{U}_>$, $\langle \mathsf{U}_>, \underline{\mathsf{P}}, \underline{\mathsf{Z}} \rangle$ fits $\underline{\mathsf{Z}}\,\mathsf{U}_>\,\underline{\mathsf{P}}\,\underline{\mathsf{Z}}$, etc. Similarly, for each of the four strings in the second column, there exists a triple in / that fits to it. We conclude that the X of equation 7.8 does not exist for $f = \underline{\mathsf{Z}}\,\mathsf{U}_>$ and $s' = \underline{\mathsf{P}}\,\mathsf{Z}$. The same result (i.e., non-existence of X) is obtained for each of the other candidate pairs of finishing and starting substrings, allowing us to assert the non-existence of $N_1(/) \xrightarrow{R} N_1(\mathsf{C})$.

The types of connections present in a TOG provide information about the descriptive capabilities of the underlying LFS. In general, the presence of weak and/or remote connections in a TOG may suggest a weakness in the design of an LFS, or the use of an inappropriate scope graph. The usefulness of the connection types will become apparent in the next chapter, where we analyse the boundary-based schemes of Chapter 2.

7.4 Construction algorithm

The complete procedure for constructing a non-atomic TOG for a restricted LFS is given by Algorithm 7.3. The algorithm takes, as input, the set of complex tokens of a restricted LFS, $\mathcal{T}$, a scope graph, G, and the length of starting and finishing substrings to use, x. The algorithm operates as follows. In the first and second steps, $\mathcal{T}$ is prepared (using Algorithm 7.1) and nodes are assigned to each of its tokens (using Algorithm 7.2). The remainder of Algorithm 7.3 adds the node connections to the graph, by considering each pair of nodes in turn (step 3) and their starting and finishing substrings (step 3.1). The three predicates strong-conn, weak-conn, and remote-conn are used to determine the existence of node connections (steps 3.1.1.1 to 3.1.1.3). Step 3.1.1 ensures that finishing substrings that end a curve and starting substrings that begin a curve are ignored. The result of the al-

gorithm is TOG($\mathcal{T}, G$), a graph encoding the ordering constraints for the tokens of $\mathcal{T}$ with respect to the scope defined by G.

The completed TOG(LFS_1, G_1) is shown in Figure 7.9, together with a table detailing its node connections. The presence of weak connections in TOG(LFS_1, G_1) indicates the possibility that not all atoms of an atomic description, after token fitting, belong to the identity of a fitted triple. This will be the case for any description that contains an angle ($\mathsf{U}_>$ or $\mathsf{U}_<$) that is not bounded on both sides by a straight-line segment ($\underline{\mathsf{Z}}$). For an example of such a description, the reader is referred back to Figure 7.1 (p. 135).

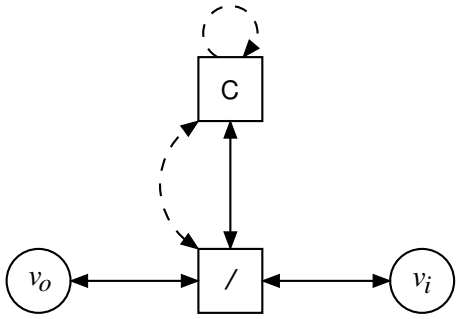
Inputs: $\mathcal{T}$ (set of LFS complex tokens), G (scope graph), x (substring length)
Output: TOG($\mathcal{T}, G$)

1 $\mathcal{T} \leftarrow prepare(\mathcal{T}, G)$;
2 $\mathcal{N} \leftarrow create\text{-}nodes(\mathcal{T}, x)$;
3 **for each** $\langle\, N \leftarrow \langle S, F\rangle,\ N' \leftarrow \langle S', F'\rangle \,\rangle \in \mathcal{N} \times \mathcal{N}$ **do**
3.1 **for each** $\langle\, f \leftarrow a\,b\,c,\ s' \leftarrow u\,v\,w \,\rangle \in F \times S'$ **do**
3.1.1 **if** $c \neq \diamond \wedge u \neq \diamond$ **then**
3.1.1.1 **if** strong-conn(f, s') **then** create $N \xrightarrow{S} N'$;
3.1.1.2 **if** weak-conn$(f, s', \mathcal{T})$ **then** create $N \xrightarrow{W} N'$;
3.1.1.3 **if** remote-conn$(f, s', \mathcal{T}, G)$ **then** create $N \xrightarrow{R} N'$;

Algorithm 7.3: Constructing a non-atomic TOG for an LFS

7.5 Example TOGs

In this section, we give TOGs for the three LFSs specified in Section 6.4 for representing polygonal shapes: $POLY_1$, $POLY_2$, and $POLY_3$. For their construction, we use the scope graph, G_p, of Figure 7.10. The TOGs for $POLY_1$, $POLY_2$, and $POLY_3$ are shown in Figures 7.11, 7.12, and 7.13 respectively, along with the associated prepared LFS specification and the set of graph nodes assigned to each token.



	$N_1(v_o)$	$N_1(v_i)$	$N_1(/)$	$N_1(\mathsf{C})$
$N_1(v_o)$			S	
$N_1(v_i)$			S	
$N_1(/)$	S	S		SW
$N_1(\mathsf{C})$			SW	W

Figure 7.9: TOG(LFS_1, G_1) and its table of node connections

The differences between the three TOGs reflect the subtle differences in the specifications of the schemes. Any pair of polygonal shapes that have different atomic descriptions can be distinguished by each of the three schemes. $POLY_1$ describes a polygon with minimal information, since only vertices are represented. In $POLY_2$, straight-line segments are explicitly included in the representation, so $POLY_2$ is, in a sense, more complete than $POLY_1$. $POLY_3$, which is a *merging* scheme, combines the properties of $POLY_1$ and $POLY_2$, since it is minimal (in terms of the number of tokens in a description) and also complete, because vertices *and* straight-line segments are included. $POLY_3$ may be interpreted as a constructive scheme, in which shapes are formed by arranging convex and concave "parts" (> and <).

An LFS designed to represent only polygonal shapes is unable to adequately represent shapes of a different kind, such as those that *can* be adequately described by LFS_1, for example. To illustrate this, we give a TOG for $POLY_2$ constructed with respect to G_1 (see Figure 7.14). TOG$(POLY_2, G_1)$ is, in fact, fully connected with respect to both weak and remote connections, i.e., between any pair of nodes there exists a weak and a remote connection. The presence of so many remote connections confirms the fact that $POLY_2$ is inadequate for representing the scope of curves defined by G_1.

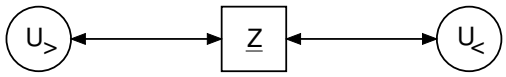
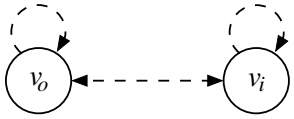


Figure 7.10: The scope graph, G_p, for polygonal shapes

$$\begin{aligned} POLY_1(G_p) \;=\; \{\, v_o \;&=\; \{\, \underline{\mathsf{Z}}\,[\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}}\,\}, \\ v_i \;&=\; \{\, \underline{\mathsf{Z}}\,[\,\mathsf{U}_{<}\,]\,\underline{\mathsf{Z}}\,\}\,\} \end{aligned}$$

$$\begin{aligned} \mathcal{N}(POLY_1(G_p)) \;=\; \{\, N_1(v_o) \;&=\; \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{>}\},\, \{\mathsf{U}_{>}\,\underline{\mathsf{Z}}\}\,\rangle, \\ N_1(v_i) \;&=\; \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{<}\},\, \{\mathsf{U}_{<}\,\underline{\mathsf{Z}}\}\,\rangle\,\} \end{aligned}$$

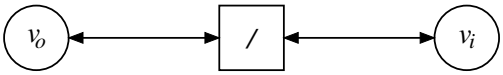


	$N_1(v_o)$	$N_1(v_i)$
$N_1(v_o)$	W	W
$N_1(v_i)$	W	W

Figure 7.11: $POLY_1(G_p)$, its assigned nodes, and TOG($POLY_1, G_p$)

$$\begin{aligned} POLY_2(G_p) \;=\; \{\, v_o \;&=\; \{\, \underline{\mathsf{Z}}\,[\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}}\,\}, \\ v_i \;&=\; \{\, \underline{\mathsf{Z}}\,[\,\mathsf{U}_{<}\,]\,\underline{\mathsf{Z}}\,\}, \\ / \;&=\; \{\, x\,[\,\underline{\mathsf{Z}}\,]\,y \;\mid\; x, y \in \{\mathsf{U}_{>}, \mathsf{U}_{<}, \diamond\}\,\}\,\} \end{aligned}$$

$$\begin{aligned} \mathcal{N}(POLY_2(G_p)) \;=\; \{\, N_1(v_o) \;&=\; \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{>}\},\, \{\mathsf{U}_{>}\,\underline{\mathsf{Z}}\}\,\rangle, \\ N_1(v_i) \;&=\; \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{<}\},\, \{\mathsf{U}_{<}\,\underline{\mathsf{Z}}\}\,\rangle, \\ N_1(/) \;&=\; \langle\, \{\mathsf{U}_{>}\,\underline{\mathsf{Z}},\, \mathsf{U}_{<}\,\underline{\mathsf{Z}},\, \diamond\underline{\mathsf{Z}}\},\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{>},\, \underline{\mathsf{Z}}\,\mathsf{U}_{<},\, \underline{\mathsf{Z}}\diamond\}\,\rangle\,\} \end{aligned}$$

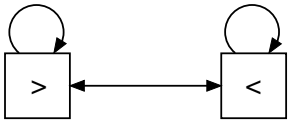


	$N_1(v_o)$	$N_1(v_i)$	$N_1(/)$
$N_1(v_o)$			S
$N_1(v_i)$			S
$N_1(/)$	S	S	

Figure 7.12: $POLY_2(G_p)$, its assigned nodes, and TOG($POLY_2, G_p$)

$$
\begin{aligned}
POLY_3(G_p) \;=\; \{\, > \;&=\; \{\, x\,[\,\underline{\mathsf{Z}}\,\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,]\,y \;\mid\; x,y \in \{\mathsf{U}_{>},\mathsf{U}_{<},\diamond\}\,\},\\
< \;&=\; \{\, x\,[\,\underline{\mathsf{Z}}\,\mathsf{U}_{<}\,\underline{\mathsf{Z}}\,]\,y \;\mid\; x,y \in \{\mathsf{U}_{>},\mathsf{U}_{<},\diamond\}\,\}\,\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{N}(POLY_3(G_p)) \;=\; \{\, N_1(>) \;&=\; \langle\, \{\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,\mathsf{U}_{>},\ \mathsf{U}_{<}\,\underline{\mathsf{Z}}\,\mathsf{U}_{>},\ \diamond\,\underline{\mathsf{Z}}\,\mathsf{U}_{>}\},\\
&\qquad \{\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,\mathsf{U}_{>},\ \mathsf{U}_{>}\,\underline{\mathsf{Z}}\,\mathsf{U}_{<},\ \mathsf{U}_{>}\,\underline{\mathsf{Z}}\,\diamond\}\,\rangle,\\
N_1(<) \;&=\; \langle\, \{\mathsf{U}_{>}\,\underline{\mathsf{Z}}\,\mathsf{U}_{<},\ \mathsf{U}_{<}\,\underline{\mathsf{Z}}\,\mathsf{U}_{<},\ \diamond\,\underline{\mathsf{Z}}\,\mathsf{U}_{<}\},\\
&\qquad \{\mathsf{U}_{<}\,\underline{\mathsf{Z}}\,\mathsf{U}_{>},\ \mathsf{U}_{<}\,\underline{\mathsf{Z}}\,\mathsf{U}_{<},\ \mathsf{U}_{<}\,\underline{\mathsf{Z}}\,\diamond\}\,\rangle\,\}
\end{aligned}
$$

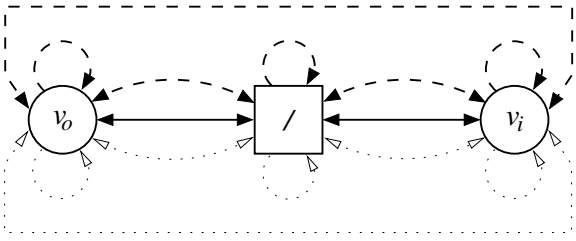


	$N_1(>)$	$N_1(<)$
$N_1(>)$	S	S
$N_1(<)$	S	S

Figure 7.13: $POLY_3(G_p)$, its assigned nodes, and TOG$(POLY_3, G_p)$

$$
\begin{aligned}
POLY_2(G_1) \;=\; \{\, v_o \;&=\; \{\,\underline{\mathsf{Z}}\,[\,\mathsf{U}_{>}\,]\,\underline{\mathsf{Z}}\,\},\\
v_i \;&=\; \{\,\underline{\mathsf{Z}}\,[\,\mathsf{U}_{<}\,]\,\underline{\mathsf{Z}}\,\},\\
/ \;&=\; \{\, x\,[\,\underline{\mathsf{Z}}\,]\,y \;\mid\; x,y \in \{\underline{\mathsf{P}},\underline{\mathsf{N}},\mathsf{U}_{>},\mathsf{U}_{<},\diamond\}\,\}\,\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{N}(POLY_2(G_1)) \;=\; \{\, N_1(v_o) \;&=\; \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{>}\},\ \{\mathsf{U}_{>}\,\underline{\mathsf{Z}}\}\,\rangle,\\
N_1(v_i) \;&=\; \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{<}\},\ \{\mathsf{U}_{<}\,\underline{\mathsf{Z}}\}\,\rangle,\\
N_1(/) \;&=\; \langle\, \{\underline{\mathsf{P}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{N}}\,\underline{\mathsf{Z}},\ \mathsf{U}_{>}\,\underline{\mathsf{Z}},\ \mathsf{U}_{<}\,\underline{\mathsf{Z}},\ \diamond\,\underline{\mathsf{Z}}\},\\
&\qquad \{\underline{\mathsf{Z}}\,\underline{\mathsf{P}},\ \underline{\mathsf{Z}}\,\underline{\mathsf{N}},\ \underline{\mathsf{Z}}\,\mathsf{U}_{>},\ \underline{\mathsf{Z}}\,\mathsf{U}_{<},\ \underline{\mathsf{Z}}\,\diamond\}\,\rangle\,\}
\end{aligned}
$$



	$N_1(v_o)$	$N_1(v_i)$	$N_1(/)$
$N_1(v_o)$	WR	WR	SWR
$N_1(v_i)$	WR	WR	SWR
$N_1(/)$	SWR	SWR	WR

Figure 7.14: $POLY_2(G_1)$, its assigned nodes, and TOG$(POLY_2, G_1)$

7.6 Summary

In this chapter, we have developed a three-stage procedure for constructing non-atomic TOGs, culminating in Algorithm 7.3. A TOG is constructed for an LFS with respect to a class of curves, specified by a scope graph. To simplify the construction process, we restricted ourselves to those LFSs whose tokens have leading and trailing contexts of length at most one atom. In the first stage of the process, an LFS is converted into explicit-context form and any triples inconsistent with the scope graph are removed. The second stage assigns graph nodes to each token of the LFS, while ensuring that node-connection transitivity is not compromised. The final stage of the process connects the nodes of the graph together, according to the set of triples associated with each node. We defined three connection *strengths*. Strong connections indicate the existence of triples that merge or meet, such that their identities join together without any intervening non-identity atoms. Weak and remote connections indicate the existence of triples that may join together more loosely, in the sense that their identities are separated by one or a number of non-identity atoms. The types of connections present in a TOG reflect the descriptive suitability of the LFS with respect to the scope used in the construction of the TOG. It was stated at the beginning of the chapter that atomic and non-atomic TOGs have the same meaning. The choice of solid lines for representing strong node connections in non-atomic TOGs is consistent with the appearance of node connections in atomic TOGs.

We finished the chapter by revisiting the polygonal schemes defined in Section 6.4. A non-atomic TOG was constructed for each of the three schemes, with respect to an appropriate scope graph. In the next chapter, we construct non-atomic TOGs for the boundary-based schemes of Chapters 2 and 3, using them as a basis for comparing the discriminatory power of the schemes.

Chapter 8

Analysis of existing schemes

In this chapter, we revisit the qualitative boundary-based schemes of Chapters 2 and 3. We specify the primitives of each scheme as complex tokens and show that each scheme is a local-feature scheme (LFS). We also construct, for each scheme, token-ordering graphs (TOGs) that can be used to compare the relative discriminatory power of the schemes.

8.1 Contour codons

We will begin by considering the set of five contour codons given by Richards & Hoffman (1987), i.e., the set $\{0^+, 0^-, 1^+, 1^-, 2\}$, which includes a subdivision of the original 0 codon[1] into 0^+ and 0^-, reflecting the fact that curves bounded by minima that contain no points of zero curvature can be either convex or concave. We looked at these descriptors and some of their properties in Section 2.3.1, as part of our review of the boundary-based approach to qualitatively representing shape. For ease of reference, the exemplar curve segments that the codons represent are reproduced in Figure 8.1. Recall that the choice of label for each codon reflects the number of points of zero curvature contained within the curve segment. Segments that don't contain any points of zero curvature are either convex (0^+) or concave (0^-). Curve segments containing a single point of zero curvature are distinguished by

[1] Found in (Hoffman & Richards 1982).

noting whether the point of zero curvature occurs before (1^-) or after (1^+) the point of maximum curvature also contained within the segment.

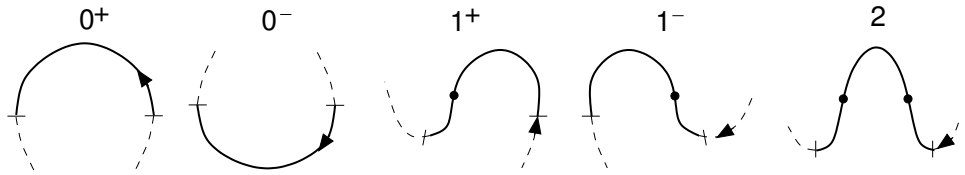


Figure 8.1: Hoffman and Richards' contour codons

Curvature plots corresponding to the five codon types are given in Figure 8.2, where curvature is denoted by c and arc length by s. A fact that is not made clear by the plots is that maxima and minima are not allowed to be points of zero curvature. For if they were, then every one of the five types could contain two points of zero curvature, e.g., imagine shifting the plot of 0^+ in Figure 8.2 down so that its two minima coincide with the s-axis. Also disallowed are curvature inflection points where the gradient is zero. In effect, then, curvature is only permitted to be zero at those points where the rate of change of curvature is *non-zero*. This constraint is enforced by the following rule:

$$c(s) = 0 \quad \Rightarrow \quad c'(s) \neq 0 \tag{8.1}$$

Also disallowed are points of curvature discontinuity, where the value of curvature is undefined. Curves are assumed to be smooth, such that there exists no point where either the curvature or the rate of change of curvature is undefined.

In order to show that the set of contour codons form an LFS, we must specify each of the contour codons as a complex token and then verify that (i) each token is MAMO, and (ii) no two tokens interfere. Definitions for these constraints are given in Section 6.2. We are then in a position to construct TOGs encoding the ordering constraints for the set of contour codons with respect to particular scope graphs.

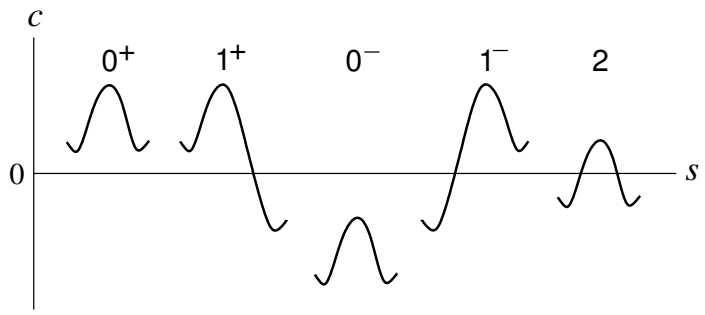


Figure 8.2: A curvature plot of Hoffman and Richards' codons

8.1.1 LFS specification

Codons are characterised by points of maximum, minimum, and zero curvature. An extremum point (a maximum or minimum of curvature) is identified by detecting that the rate of change of curvature (c') is zero. Therefore, the codons need to be specified at level 3 or above in the atomic hierarchy, because the qualitative value of c' is given by the third component of a curve state. Naturally, we choose level 3, so as to minimise the complexity of our specifications. In order to distinguish a maximum from a minimum, the gradient before and after the extremum must be considered. A positive maximum, for example, is a positive stationary point where the curvature before the point is increasing and the curvature after the point is decreasing: $\underline{\mathsf{P}^+}\,[\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^-}$. For a minimum, curvature decreases before the stationary point and increases afterwards, giving $\underline{\mathsf{P}^-}\,[\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^+}$. Each codon represents a class of curve segments identified by a particular string of atoms. Consider, for example, the codon type $\mathsf{0}^+$, represented by the leftmost curvature-variation pattern in Figure 8.2. An occurrence of $\mathsf{0}^+$ is given by the occurrence of a positive minimum, followed by a positive maximum, followed by another positive minimum, i.e., the string of atoms $\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}$. There is a one-to-one correspondence between this string and occurrences of $\mathsf{0}^+$. Similar strings can be given for each of the four other codon types.

We have all the information we need to define the contour codons as complex tokens. Before we can formulate the specifications, however, we need to decide which strings of

atoms should constitute the codon identities and which atoms, if any, should constitute the leading and trailing contexts. There is an ambiguity: should the minima that bound a codon form part of the codon's identity, or should they be treated instead as contextual information? There seems no good reason to prefer either of the two possibilities, so we shall consider them both in the analysis that follows. Set CC_e contains codon specifications with the minima *excluded* from the identity strings. The minima are given, instead, as the leading and trailing contexts of each specification. In CC_i, the minima are included in the identity string of each specification.

$$
\begin{array}{rcl}
CC_e = \{\ 0^+ & = & \{\, \mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\mathsf{P}^0\,\},\\
0^- & = & \{\, \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{N}^0\,\},\\
1^+ & = & \{\, \mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,]\,\mathsf{N}^0\,\},\\
1^- & = & \{\, \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\mathsf{P}^0\,\},\\
2 & = & \{\, \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,]\,\mathsf{N}^0\,\}\ \ \}
\end{array}
$$

$$
\begin{array}{rcl}
CC_i = \{\ 0^+ & = & \{\, [\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,]\,\},\\
0^- & = & \{\, [\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,\mathsf{N}^0\,]\,\},\\
1^+ & = & \{\, [\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0\,]\,\},\\
1^- & = & \{\, [\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,]\,\},\\
2 & = & \{\, [\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0\,]\,\}\ \ \}
\end{array}
$$

Table 8.1: Alternative sets of contour-codon specifications: CC_e and CC_i

We now need to check that both CC_e and CC_i define an LFS. First, consider CC_e. We need to verify that each token in CC_e is MAMO and that no two tokens in CC_e interfere. Recall that a token is MAMO ("merge and meet only") so long as, when it is fitted to an atomic description, its non-redundant fitted triples only merge, meet, or are disjoint. In addition, the problematic triple configuration **pc5** must not exist (see Figure 6.2, p. 121). Every token specification contains a single triple. Consider the triple in CC_e for the codon type 0^-: $t = \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{N}^0$. The single valid alignment that exists between t and itself is shown in Table 8.2. In the alignment, the identities are disjoint, so none of the thirteen

triple relations hold between t and itself. The 0^- token is therefore MAMO. The codon type 0^+ has a similar valid alignment and is also MAMO. The triples of the remaining three tokens do not have *any* valid alignments when considered independently. We conclude, then, that all five tokens of CC_e are MAMO, as required. In general, any complex token, T, that contains a single triple, t, is MAMO, unless t may-overlap t holds, in which case the token *cannot* be MAMO, because a triple that overlaps itself cannot, in addition, subsume the overlapping of its own identities. The problematic configuration of **pc5** cannot exist in T, because **pc5** involves the merging *and* meeting of triples: a single triple is unable to both merge with itself *and* meet itself. Furthermore, a triple with an identity string of length less than four cannot overlap itself.[2] These results give rise to the following two lemmas:

Lemma 8.1 *If a complex token, T, contains a single triple, t, and $\neg\, t$ may-overlap t holds, then T is MAMO.*

Lemma 8.2 *If a complex token, T, contains a single triple, t, and the length of the identity string of t is less than four, then T is MAMO.*

N^0	$\underline{N^+}$	N^0	$\underline{N^-}$	N^0				
				N^0	$\underline{N^+}$	N^0	$\underline{N^-}$	N^0

Table 8.2: The single valid alignment of $N^0\,[\,\underline{N^+}\,N^0\,\underline{N^-}\,]\,N^0$ and itself

Having ascertained that all of the complex tokens in CC_e are MAMO, we need to verify that the *non-interference* relation, defined in Section 6.2.2 and symbolised by $\odot$, holds between each pair of distinct tokens in CC_e. Given two distinct tokens, T and T', $T \odot T'$ holds (T and T' do not interfere) *iff* the only relations that hold between the triples of T and T' are of the merging and meeting variety, i.e., tr1 to tr4. For CC_e, *none* of the triple relations hold between triples from different token specifications. Therefore, there can be

[2]For a triple to overlap another triple, the identities of both triples must be at least three atoms in length. For a single triple with an identity of length three to overlap itself, the first atom of the identity must be the same as the second atom. This is not possible, because an atom cannot occur consecutively in a string.

no interference between any of the tokens. Consequently, there can be no "global" occurrence of **pc5**, which requires merging and meeting relations to hold. We conclude, then, that CC_e is an LFS of the *no-merging* variety.

The specifications of the alternative set, CC_i, only differ with respect to the inclusion of the bounding minima in the identities of the codons. The contexts of each token are left empty, indicating that all possible contexts are allowed. When determining whether or not each token is MAMO, we find that the single triples of 0^+, 0^-, and 2 may self-merge (which is permissible), and that for 1^+ and 1^- there exists no valid alignment. When we consider the relations that hold between the triples of different tokens, we find that only the merging relation holds. We conclude, then, that CC_i is an LFS of the *merging* variety.

8.1.2 TOG construction

To show how contour codons connect together, we will construct two pairs of TOGs. The first pair will be constructed with respect to a scope graph that is representative of the class of curves that are intended to be described by contour codons. The second pair of TOGs will be constructed with respect to the full set of curves that can be described by strings of atoms at level 3 in the atomic hierarchy.

Restricted scope

The scope graph that will be used for the construction of the first pair of non-atomic TOGs for CC_e and CC_i is given in Figure 8.3. The graph, which we label G_s, represents smooth curves with continuously varying curvature, such that (i) there are no tangent-bearing discontinuities, and (ii) the first two derivatives of tangent-bearing are defined at every point. Comparing G_s with the level-3 atomic TOG (Figure 4.5, p. 89), we can see that it is similar, in terms of the atoms used and the connections present, to the subgraph constituted by the nine-node rectangular region in the centre of the atomic TOG. The absence of Z^0 and its connections is accounted for by rule 8.1. There are two nodes each for P^0 and N^0, disallowing curves that contain points of curvature inflection where the value of curvature is

positive or negative (see Figure 8.4). It is not clear whether Hoffman and Richards allow such stationary points within a codon segment, and so, because their omission simplifies the specification of the codons, we do not allow them. If we *were* to include non-zero points of curvature inflection within codon identities, then the specification of each codon would contain an *infinite* number of triples. The specification of 0^+, for example, would be given by $\{\, P^0\, [\, (\underline{P^+}\, P^0)^+\, \underline{P^-}\, (P^0\, \underline{P^-})^*\,]\, P^0\, \}$.

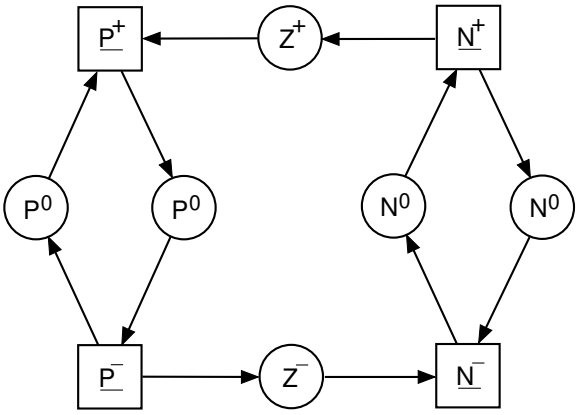


Figure 8.3: The restricted scope graph, G_s, for contour codons

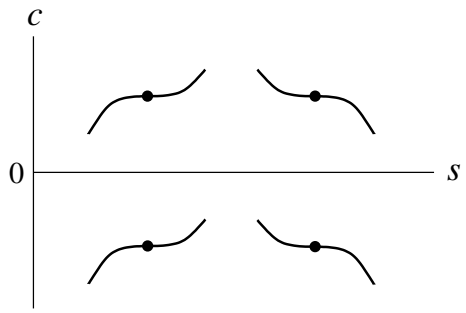


Figure 8.4: Non-zero points of curvature inflection

The first stage of the TOG construction process requires that CC_e and CC_i are prepared with respect to G_s. This involves the conversion of triples into explicit-context form and the removal of triples not consistent with G_s. Our choice of scope graph ensures that the

triples of CC_e and CC_i are *all* consistent with G_s. The triples of CC_e are already in explicit-context form, so we have $CC_e(G_s) = CC_e$. The tokens of CC_i *do* need to be converted, but the conversion is straightforward because, for each triple, only one possibility exists for its leading and trailing context atoms. The token 0^+, for example, has an identity that starts with $\mathsf{P}^0\,\underline{\mathsf{P}^+}$ and ends with $\underline{\mathsf{P}^-}\,\mathsf{P}^0$. Consulting the scope graph, we can see that the only allowable leading context is $\underline{\mathsf{P}^-}$ and the only allowable trailing context is $\underline{\mathsf{P}^+}$. The prepared set of CC_i specifications is given in Table 8.3.

$$
\begin{array}{rcl}
CC_i(G_s) = \{\ 0^+ & = & \{\underline{\mathsf{P}^-}\,[\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^+}\},\\
0^- & = & \{\underline{\mathsf{N}^-}\,[\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,\mathsf{N}^0\,]\,\underline{\mathsf{N}^+}\},\\
1^+ & = & \{\underline{\mathsf{P}^-}\,[\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0\,]\,\underline{\mathsf{N}^+}\},\\
1^- & = & \{\underline{\mathsf{N}^-}\,[\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^+}\},\\
2 & = & \{\underline{\mathsf{N}^-}\,[\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0\,]\,\underline{\mathsf{N}^+}\}\quad\}
\end{array}
$$

Table 8.3: The prepared specification of CC_i, with respect to G_s

The second stage of the construction process involves the mapping of LFS tokens to graph nodes, as detailed in Section 7.3.2. In that section, we noted that a token specification consisting of a single triple only requires one node. Since each of our codon specifications contain only one triple, node assignment is straightforward. The nodes assigned to the tokens of CC_e and CC_i are listed in Table 8.4. Recall that each node has an associated pair of sets, $\langle S, F\rangle$, where S is a set of distinct starting substrings and F is a set of distinct finishing substrings. For CC_e, which is a no-merging LFS, the substrings are all of length two; for CC_i, which is a merging LFS, they are all of length three. Contour codons are curve segments, rather than point features, so all of the nodes in the TOGs we construct will be rectangular, rather than circular, in appearance.

The final stage of the construction process involves the determination of the node connections in each graph. We will start with the TOG for CC_e. First, consider strong connections. A strong connection exists from N to N' *iff* one of the finishing substrings of N equals one of the starting substrings of N'. Consulting the set of nodes $\mathcal{N}(CC_e(G_s))$

$$
\begin{array}{rcl}
\mathcal{N}(CC_e(G_s)) = \{ & N_1(0^+) & = \langle\, \{\mathsf{P}^0\,\underline{\mathsf{P}^+}\},\, \{\underline{\mathsf{P}^-}\,\mathsf{P}^0\} \,\rangle, \\
& N_1(0^-) & = \langle\, \{\mathsf{N}^0\,\underline{\mathsf{N}^+}\},\, \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\} \,\rangle, \\
& N_1(1^+) & = \langle\, \{\mathsf{P}^0\,\underline{\mathsf{P}^+}\},\, \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\} \,\rangle, \\
& N_1(1^-) & = \langle\, \{\mathsf{N}^0\,\underline{\mathsf{N}^+}\},\, \{\underline{\mathsf{P}^-}\,\mathsf{P}^0\} \,\rangle, \\
& N_1(2) & = \langle\, \{\mathsf{N}^0\,\underline{\mathsf{N}^+}\},\, \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\} \,\rangle \quad \}
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{N}(CC_i(G_s)) = \{ & N_1(0^+) & = \langle\, \{\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\},\, \{\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\} \,\rangle, \\
& N_1(0^-) & = \langle\, \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\},\, \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\} \,\rangle, \\
& N_1(1^+) & = \langle\, \{\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\},\, \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\} \,\rangle, \\
& N_1(1^-) & = \langle\, \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\},\, \{\underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\} \,\rangle, \\
& N_1(2) & = \langle\, \{\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\},\, \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\} \,\rangle \quad \}
\end{array}
$$

Table 8.4: The nodes for TOG(CC_e, G_s) and TOG(CC_i, G_s)

in Table 8.4, it is clear that there are no strong connections in the graph for CC_e. A weak connection exists whenever two triples may join together such that their identities are separated by a single non-identity atom, and no other triple fits between their identities. There are several weak connections in the graph. As an example, here is how we determine the existence of a weak connection from $N_1(1^+)$ to $N_1(2)$:

- $N_1(1^+)$ contains the single finishing substring $f = bc = \underline{\mathsf{N}^-}\,\mathsf{N}^0$ and $N_1(2)$ contains the single starting substring $s = u\,v = \mathsf{N}^0\,\underline{\mathsf{N}^+}$. We have $c = u$, which means that the identities of the triple(s) characterised by f and s *do* join together such that they are separated by a single non-identity atom (in this case N^0).

- The second condition requires that no triple fits between the identities of the triples characterised by f and s. We must check that the triple $\langle b, c, v\rangle$ ($\underline{\mathsf{N}^-}\,[\,\mathsf{N}^0\,]\,\underline{\mathsf{N}^+}$) does not exist in any specification of CC_e. It does not, therefore there *is* a weak connection from $N_1(1^+)$ to $N_1(2)$.

We can carry out a similar analysis for the remaining twenty-four pairs of nodes, with the result that, for each pair of nodes, it is known whether or not there exists a weak

connection. All we have left to do, then, is to determine whether any *remote* connections exist in the graph. Here is how we determine whether a remote connection exists from $N(0^+)$ to $N(1^-)$:

- From $N(0^+)$ we get $f = bc = \underline{\mathsf{P}^-}\,\mathsf{P}^0$ and from $N(1^-)$ we get $s = u\,v = \mathsf{N}^0\,\underline{\mathsf{N}^+}$. A remote connection exists from $N(0^+)$ to $N(1^-)$ *iff* there exists an X such that the string $\alpha = \underline{\mathsf{P}^-}\,\mathsf{P}^0\,X\,\mathsf{N}^0\,\underline{\mathsf{N}^+}$ is derivable from G_s, and no triple in any specification of CC_e fits α.

- To see what the possibilities for X are, we need to consult G_s. Note that the second atom of α corresponds to the leftmost P^0 in G_s, because it is preceded by $\underline{\mathsf{P}^-}$. Similarly, the penultimate atom of α corresponds to the leftmost N^0 in G_s. The string X, then, defines a path in G_s connecting the leftmost node for P^0 to the leftmost node for N^0. Starting at the leftmost P^0, the next node in the path must be $\underline{\mathsf{P}^+}$, the node after that must be the rightmost P^0, and the node after that must be $\underline{\mathsf{P}^-}$. At $\underline{\mathsf{P}^-}$ we can either return to the leftmost P^0 or we can go to Z^-. If we go to Z^-, then we reach the leftmost N^0 via $\underline{\mathsf{N}^-}$. We have established, then, that X must either start with $x = \underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0$ or $x' = \underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0$.

- If X begins with x, then $\alpha = \underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0 \cdots \mathsf{N}^0\,\underline{\mathsf{N}^+}$, in which case the string corresponding to the triple of 0^+, i.e., $\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{P}^0$, fits α. If, on the other hand, X begins with x', then $\alpha = \underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0 \cdots \mathsf{N}^0\,\underline{\mathsf{N}^+}$, and the string corresponding to the triple of 1^+, i.e., $\mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0$, fits α. We have shown that there does *not* exist an X such that no triple in any specification of CC_e fits α. We conclude, therefore, that there is no remote connection from $N(0^+)$ to $N(1^-)$.

Applying the same kind of argument to the remaining pairs of nodes, we find that the graph contains no remote connections. Table 8.5 summarises the connection types that exist between the nodes of $\mathrm{TOG}(CC_e, G_s)$. The complete TOG constructed for CC_e with respect to G_s is given on the left of Figure 8.5. Using the TOG, we can determine the

syntax of the strings of the CC_e LFS. We see that three of the nodes have loops, indicating that three of the five codons may occur immediately adjacent to themselves (0^+, 0^-, and 2). The absence of any strong connections means that, given an atomic description of a curve within the scope, not all of the atoms in the description will belong to the identity of a codon. Specifically, the minimum points are considered distinct from the actual curve segments represented by the codons. The absence of any remote connections means that, for all curves within scope, all of the atoms of their atomic descriptions are accounted for by either the identity or the context of a codon.

	$N_1(0^+)$	$N_1(0^-)$	$N_1(1^+)$	$N_1(1^-)$	$N_1(2)$
$N_1(0^+)$	W		W		
$N_1(0^-)$		W		W	W
$N_1(1^+)$		W		W	W
$N_1(1^-)$	W		W		
$N_1(2)$		W		W	W

Table 8.5: The node connections for TOG(CC_e, G_s)

The construction of the TOG for CC_i closely mirrors the construction just described for CC_e. The main difference being that certain triples within CC_i may merge, because of the difference in the way the codons are specified. Consequently, where we had a weak connection in the TOG for CC_e, we have a *strong* connection in the TOG for CC_i. The TOG for CC_i is given on the right of Figure 8.5.

Full scope

The "full" scope is provided by the level-3 atomic TOG (Figure 4.5, p. 89), which we will refer to as G_{f3}. The TOGs for CC_e and CC_i are constructed in exactly the same manner as before. Bearing in mind that G_s is essentially a subgraph of G_{f3}, i.e., all the paths through the former are present in the latter, and the fact that scope graphs do not need to be consulted when determining strong or weak connections, CC_e and CC_i do not need

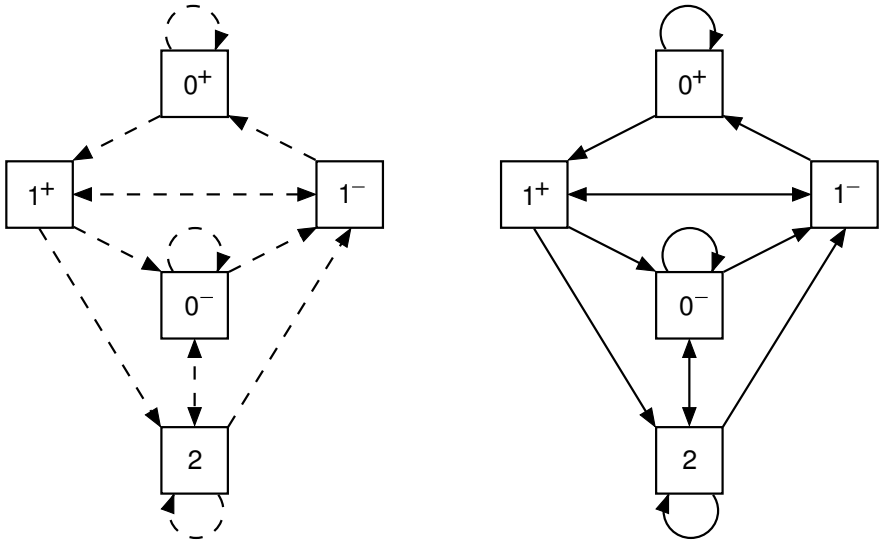


Figure 8.5: TOG(CC_e, G_s) and TOG(CC_i, G_s)

to be re-prepared ($CC_e(G_{f3}) = CC_e(G_s), CC_i(G_{f3}) = CC_i(G_s)$[3]) and the configuration of strong and weak connections will be the same as before. We need only concern ourselves, therefore, with remote connections. A consequence of the additional atoms and, therefore, path possibilities provided by G_{f3}, is that the constructed TOGs for CC_e and CC_i contain a large number of remote connections. In fact, each TOG is fully connected with respect to remote connections, i.e., there exists a remote connection between *every* pair of nodes. As an example, let's consider again whether there is a remote connection from $N(0^+)$ to $N(1^-)$ in the TOG for CC_e, this time using G_{f3} as the scope graph instead of G_s:

- A remote connection exists *iff* there exists an X such that $\alpha = \underline{\mathsf{P}^-}\,\mathsf{P}^0\,X\,\mathsf{N}^0\,\underline{\mathsf{N}^+}$ is derivable from G_{f3}, and no triple in any specification of CC_e fits α.

- In G_{f3}, P^0 and N^0 require only one node each, because non-zero points of curvature inflection *are* allowed in G_{f3}. We need to consider the valid paths that con-

[3]Strictly speaking, CC_i *should* be re-prepared, because its leading and trailing contexts are implicit, and, as non-zero points of curvature inflection *are* allowed in G_{f3}, there are, in each case, *two* possible interval atoms that can be assigned to the leading and trailing contexts of a triple. However, to preserve the essential difference between the TOGs constructed for CC_e and CC_i (weak versus strong connections) we choose, here, to assert that $CC_i(G_{f3})$ contains the same specifications as $CC_i(G_s)$.

nect P^0 to N^0. One possible path is $\mathsf{P}^0 \rightarrow \underline{\mathsf{P}^-} \rightarrow \mathsf{U}_> \rightarrow \underline{\mathsf{N}^+} \rightarrow \mathsf{N}^0$, giving us $\alpha = \underline{\mathsf{P}^-}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{U}_>\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}$. There does not exist a triple in CC_e that fits this string, so we conclude that there *does* exist a remote connection from $N(0^+)$ to $N(1^-)$.

Although there are, in fact, an infinite number of different paths that imply a remote connection from $N(0^+)$ to $N(1^-)$, we only need to find a *single* path in order to verify the existence of a remote connection. We can construct an analogous value of X for *any* pair of nodes in the graph, as follows. The associated α for a pair of nodes takes the form $b\,c\,X\,u\,v$. Assign the string $b\,\mathsf{U}_>\,v$ to X. The resultant α will be such that no triple in CC_e fits it. For the pair $\langle N_1(2), N_1(0^-)\rangle$, for example, we get $\underline{\mathsf{N}^-}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,\mathsf{U}_>\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}$. The TOGs for CC_e and CC_i, constructed with respect to G_{f3}, are given in Figure 8.6.

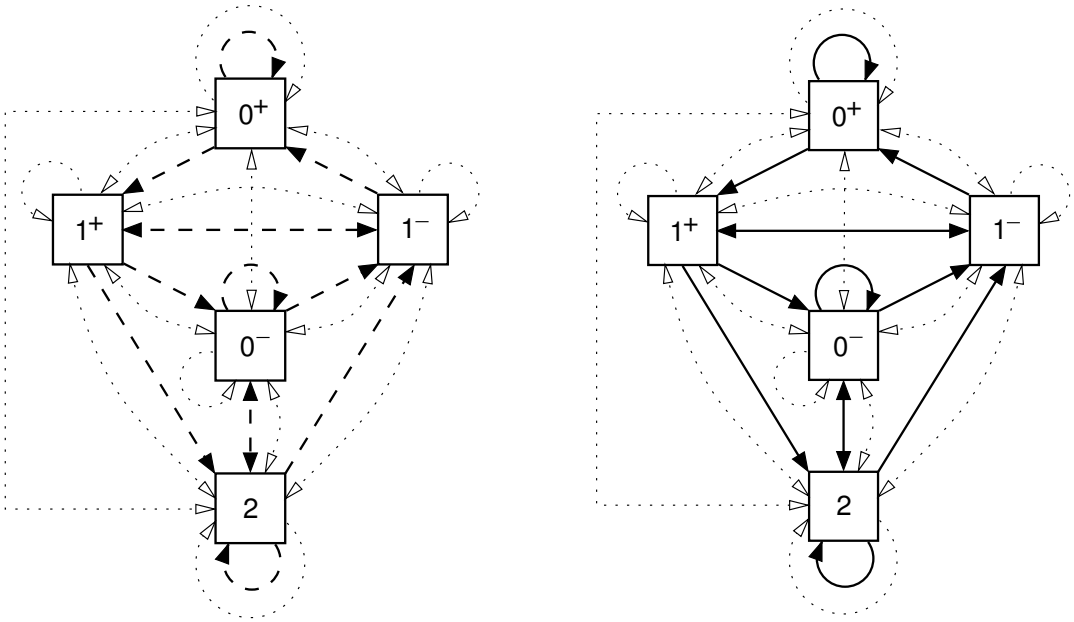


Figure 8.6: TOG(CC_e, G_{f3}) and TOG(CC_i, G_{f3})

The number of remote connections that the TOGs now contain suggest that contour codons are inadequate for representing certain curves that fall within the full scope of the theory of atoms. Kink points, straight-line segments, and circular-arc segments, for example, are not represented. Also not represented are the atomic curve features represented

by $\underline{\mathsf{P}^0}$, P^{U}, Z^0, Z^{U}, $\underline{\mathsf{N}^0}$, N^{U}, and U_{c}. Shapes that contain such features *can* be described using strings of codons (by simply ignoring atoms that have no representation under the contour-codon scheme), but there will be a significant amount of information loss. In some cases, the description of a shape in terms of codons may just be "Λ" (the empty string), e.g., every polygon has an empty description. This property of CC_e and CC_i is not to be considered a deficiency when the purpose of the schemes is taken into account. However, it does highlight a lack of discriminatory power with respect to certain shapes that one might want to represent.

8.1.3 The extended set of codons

Rosin (1993) provides a set of contour codons that includes Hoffman & Richards' original six (the five shown earlier, plus "∞" for representing straight-line segments), together with a large number of additions. The primary motivation for the expansion is to increase the expressive power of codons, in order to meet a design criterion, given by Rosin, for contour representations: "The representation should be adequately expressive so that sufficient shapes can be represented by unique (i.e., distinguishing) descriptions" (p. 286). In total, fifty-eight new codons are introduced, of which seven are required for representing adjoining straight lines, twelve for representing open curves, seven for representing curves containing no minima, and thirty-two for representing "semi-cusps". Rosin's scheme, then, consists of sixty-four primitives in all. As pointed out in Section 2.3.1, circular-arc segments are not represented and no distinction is made between cusps and angles; Rosin-style cusps are formed by connecting together two semi-cusp codons. All of the codons in the extended set can be specified as complex tokens using atoms at level 3 in the atomic hierarchy (see Appendix C for a full list of the specifications). Figure 8.7 shows a selection of codons from the extended set and the corresponding complex-token specifications.

The discriminatory power provided by Rosin's extended set of codons is much greater than that provided by Hoffman and Richards' original set of codons. However, there are still curve features not represented by Rosin's scheme, so it cannot really be used as a

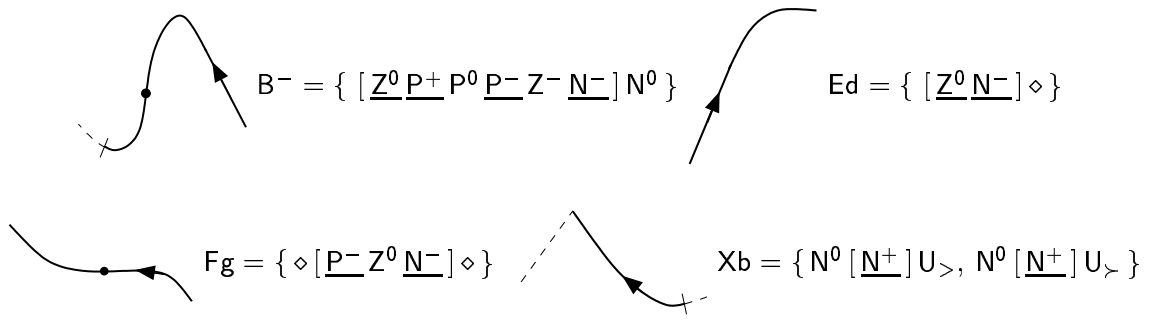


Figure 8.7: A selection of Rosin's codons and their specifications

general scheme for describing shape in two dimensions. Interestingly, the twenty atoms at level 3 in the atomic hierarchy (the derived atoms together with the four kink tokens) have greater discriminatory power than the sixty-four primitives comprising the extended set of contour codons.

8.2 Extremum primitives

Leyton's scheme, which we looked at in Section 2.3.2, contains primitives representing curvature extrema; his choice of primitives is motivated by the idea that process activity explains the shape of certain natural forms. A different kind of deformational process is associated with each of the four kinds of curvature extremum, as listed in Table 2.3. The primitives M^+ and m^+ are associated with protrusion and squashing processes respectively, while M^- and m^- are associated with internal resistance and indentation respectively. The annotated curvature plot of Figure 8.8 shows the assignment of primitives to curvature extrema.

In addition to the four primitives representing curvature extrema, Leyton includes a fifth primitive, $\mathsf{0}$, representing points where the curvature is zero. Although Leyton does not explicitly disallow stationary points of curvature that are not extrema, we make the assumption that such points are not found on the curves ordinarily described by Leyton's primitives. In other words, we assume that rule 8.1 holds and that Leyton's primitives

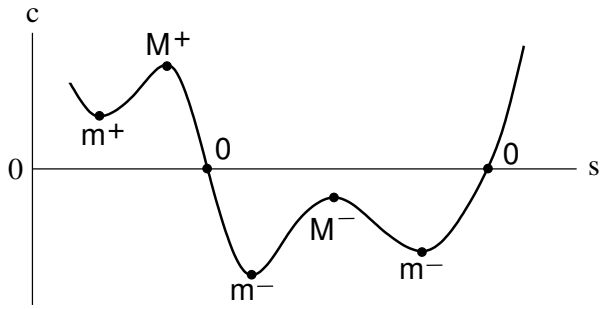


Figure 8.8: Curvature plot showing the primitives of Leyton's scheme

are therefore designed to represent the same class of curves as the contour codons. When constructing the TOG for Leyton's scheme, then, we will use G_s as the scope graph.

8.2.1 LFS specification

A curvature extremum corresponds to a stationary point of curvature within a particular context. The "squashing" primitive (m^+), for example, represents a positive stationary point (P^0) preceded by an interval of decreasing positive curvature ($\underline{\mathsf{P}^-}$) and followed by an interval of increasing positive curvature ($\underline{\mathsf{P}^+}$). The primitive m^+ is therefore specified by a single triple: $\underline{\mathsf{P}^-}\,[\,\mathsf{P}^0\,]\,\underline{\mathsf{P}^+}$. There are two kinds of zero-curvature point, representing the two possible ways in which the sign of curvature can change, i.e., from positive to negative or vice versa. Consequently, whereas each extremum primitive requires only one triple in its specification, the $\mathsf{0}$ primitive requires two. The five primitives of Leyton's scheme are specified in Table 8.6.

The set LEY is an LFS, because all of its tokens are MAMO, and there is no interference between tokens. The extremum primitives are MAMO, since each primitive has an identity consisting of a single atom (lemma 8.2). The $\mathsf{0}$ primitive is also MAMO, because its triples do not overlap and identity-equality is not possible. Tokens that represent extrema with the same curvature sign have identical identities but, importantly, their contexts differ. Such tokens, therefore, do not interfere when fitted to an atomic description.

$$
\begin{array}{rcl}
LEY = \{ \quad \mathsf{M}^{+} & = & \{\underline{\mathsf{P}^{+}}[\mathsf{P}^{0}]\underline{\mathsf{P}^{-}}\}, \\
\mathsf{m}^{+} & = & \{\underline{\mathsf{P}^{-}}[\mathsf{P}^{0}]\underline{\mathsf{P}^{+}}\}, \\
\mathsf{M}^{-} & = & \{\underline{\mathsf{N}^{+}}[\mathsf{N}^{0}]\underline{\mathsf{N}^{-}}\}, \\
\mathsf{m}^{-} & = & \{\underline{\mathsf{N}^{-}}[\mathsf{N}^{0}]\underline{\mathsf{N}^{+}}\}, \\
\mathsf{0} & = & \{\underline{\mathsf{P}^{-}}[\mathsf{Z}^{-}]\underline{\mathsf{N}^{-}}, \underline{\mathsf{N}^{+}}[\mathsf{Z}^{+}]\underline{\mathsf{P}^{+}}\} \quad \}
\end{array}
$$

Table 8.6: The token specifications for Leyton's scheme

8.2.2 TOG construction

We will construct two TOGs for LEY. The first will be constructed with respect to G_s, and the second with respect to G_{f3} (the level-3 atomic TOG, p. 89).

Restricted scope

The triples of LEY are already in explicit-context form and consistent with G_s, so the preparation stage is redundant. Only one node is required for each of the four extremum primitives, because they each contain just a single triple. The $\mathsf{0}$ primitive, however, requires two nodes, because if both triples were associated with the same node then node-connection transitivity could not be guaranteed. The triples of $\mathsf{0}$ give rise to a set, S, of two distinct starting substrings and a set, F, of two distinct finishing substrings, as follows: $S = \{\underline{\mathsf{P}^{-}}\mathsf{Z}^{-}, \underline{\mathsf{N}^{+}}\mathsf{Z}^{+}\}$ and $F = \{\mathsf{Z}^{-}\underline{\mathsf{N}^{-}}, \mathsf{Z}^{+}\underline{\mathsf{P}^{+}}\}$. In order to ensure the presence of all possible combinations of distinct starting and finishing substrings, the node would need to be associated with at least $2 \times 2 = 4$ triples. Since $\mathsf{0}$ contains only two triples, this is not possible. Consequently, two nodes are assigned to $\mathsf{0}$. The complete set of nodes required for TOG(LEY, G_s) is given in Table 8.7.

Now we are in a position to connect the nodes of the graph together. There is no merging or meeting of any triples in LEY, and so the graph doesn't contain any strong connections. There are, however, a number of weak connections. For example, $N_1(\mathsf{M}^{+})$ weakly connects to $N_1(\mathsf{0})$, because (i) the distinct finishing substring associated with $N_1(\mathsf{M}^{+})$ has

$$\begin{aligned}\mathcal{N}(LEY(G_s)) = \{\quad & N_1(\mathsf{M}^+) &=& \langle \{\underline{\mathsf{P}^+}\,\mathsf{P}^0\}, \{\mathsf{P}^0\,\underline{\mathsf{P}^-}\}\rangle,\\ & N_1(\mathsf{m}^+) &=& \langle \{\underline{\mathsf{P}^-}\,\mathsf{P}^0\}, \{\mathsf{P}^0\,\underline{\mathsf{P}^+}\}\rangle,\\ & N_1(\mathsf{M}^-) &=& \langle \{\underline{\mathsf{N}^+}\,\mathsf{N}^0\}, \{\mathsf{N}^0\,\underline{\mathsf{N}^-}\}\rangle,\\ & N_1(\mathsf{m}^-) &=& \langle \{\underline{\mathsf{N}^-}\,\mathsf{N}^0\}, \{\mathsf{N}^0\,\underline{\mathsf{N}^+}\}\rangle,\\ & N_1(\mathsf{0}) &=& \langle \{\underline{\mathsf{P}^-}\,\mathsf{Z}^-\}, \{\mathsf{Z}^-\,\underline{\mathsf{N}^-}\}\rangle,\\ & N_2(\mathsf{0}) &=& \langle \{\underline{\mathsf{N}^+}\,\mathsf{Z}^+\}, \{\mathsf{Z}^+\,\underline{\mathsf{P}^+}\}\rangle\quad\}\end{aligned}$$

Table 8.7: The nodes for TOG(LEY, G_s)

a trailing-context atom ($\underline{\mathsf{P}^-}$) that matches the leading-context atom of the distinct starting substring associated with $N_1(\mathsf{0})$, and (ii) the triple $\mathsf{P}^0\,[\,\underline{\mathsf{P}^-}\,]\,\mathsf{Z}^-$ does not exist in LEY. The graph does not contain any remote connections. Here, for example, is how we determine that there is *not* a remote connection from $N_1(\mathsf{m}^+)$ to $N_1(\mathsf{M}^-)$:

- $N_1(\mathsf{m}^+)$ contains the distinct finishing substring $\mathsf{P}^0\,\underline{\mathsf{P}^+}$ and $N_1(\mathsf{M}^-)$ contains the distinct starting substring $\underline{\mathsf{N}^+}\,\mathsf{N}^0$. A remote connection exists from $N_1(\mathsf{m}^+)$ to $N_1(\mathsf{M}^-)$, then, *iff* there exists an X such that the string $\alpha = \mathsf{P}^0\,\underline{\mathsf{P}^+}\,X\,\underline{\mathsf{N}^+}\,\mathsf{N}^0$ is derivable from G_s, and no triple in any specification of LEY fits α.

- Consulting G_s, the minimal path from $\underline{\mathsf{P}^+}$ to $\underline{\mathsf{N}^+}$ is $\underline{\mathsf{P}^+} \to \mathsf{P}^0 \to \underline{\mathsf{P}^-} \to \mathsf{Z}^- \to \underline{\mathsf{N}^-} \to \mathsf{N}^0 \to \underline{\mathsf{N}^+}$, giving $X = \mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0$ and $\alpha = \mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\,\mathsf{N}^0$. Three tokens fit to α in this case: M^+, $\mathsf{0}$, and m^-. We need to show that at least one token fits to *all possible* paths.

- Inspecting G_s, it is clear that any non-minimal path from $\underline{\mathsf{P}^+}$ to $\underline{\mathsf{N}^+}$ must start with $\underline{\mathsf{P}^+} \to \mathsf{P}^0 \to \underline{\mathsf{P}^-}$ and end with $\underline{\mathsf{N}^-} \to \mathsf{N}^0 \to \underline{\mathsf{N}^+}$, i.e., X must start with $\mathsf{P}^0\,\underline{\mathsf{P}^-}$ and end with $\mathsf{N}^0\,\underline{\mathsf{N}^+}$. For every possible value of X, then, $\alpha = \mathsf{P}^0\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\cdots\underline{\mathsf{N}^-}\,\mathsf{N}^0\,\underline{\mathsf{N}^+}\mathsf{N}^0$. The triples of M^+ and m^- always fit such a string. We conclude, then, that there is no remote connection from $N_1(\mathsf{m}^+)$ to $N_1(\mathsf{M}^-)$.

The connection information for LEY is given in Table 8.8, and the corresponding TOG in Figure 8.9. The nodes in the TOG are shown as circles, as each token represents a point

curve feature. We can see, as we would expect, that TOG(LEY, G_s) encodes information pertaining to the order in which curvature extrema can occur in a plot of curvature against arc length. The 0 primitive is redundant, in the sense that its presence can be inferred from the order of the extrema, e.g., given M^+ followed by m^- we know that, so long as the curve being described is within scope, a point of zero curvature must be present in between the two extrema.

	$N_1(\mathsf{M}^+)$	$N_1(\mathsf{m}^+)$	$N_1(\mathsf{M}^-)$	$N_1(\mathsf{m}^-)$	$N_1(0)$	$N_2(0)$
$N_1(\mathsf{M}^+)$		W			W	
$N_1(\mathsf{m}^+)$	W					
$N_1(\mathsf{M}^-)$				W		
$N_1(\mathsf{m}^-)$			W			W
$N_1(0)$				W		
$N_2(0)$	W					

Table 8.8: The node connections for TOG(LEY, G_s)

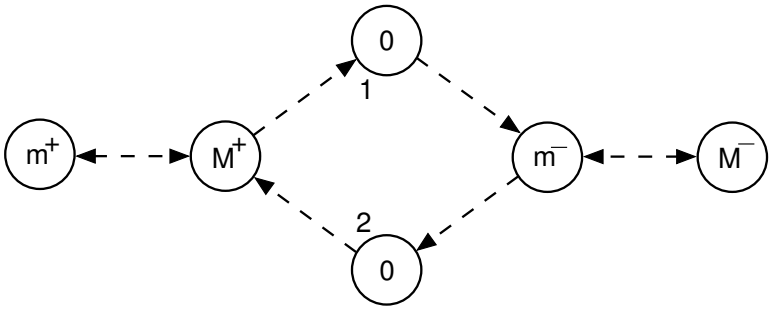


Figure 8.9: TOG(LEY, G_s)

Full scope

The construction of the TOG for LEY with respect to the level-3 atomic TOG (G_{f3}) follows the same pattern as when we considered the contour codons and full scope. There are

no strong connections, and the weak connections remain unchanged in the graph. The graph is fully connected with remote connections, indicating, correctly, that the primitives of LEY are not capable of adequately representing the full scope of curves. The TOG constructed for LEY using G_{f3} is given in Figure 8.10.

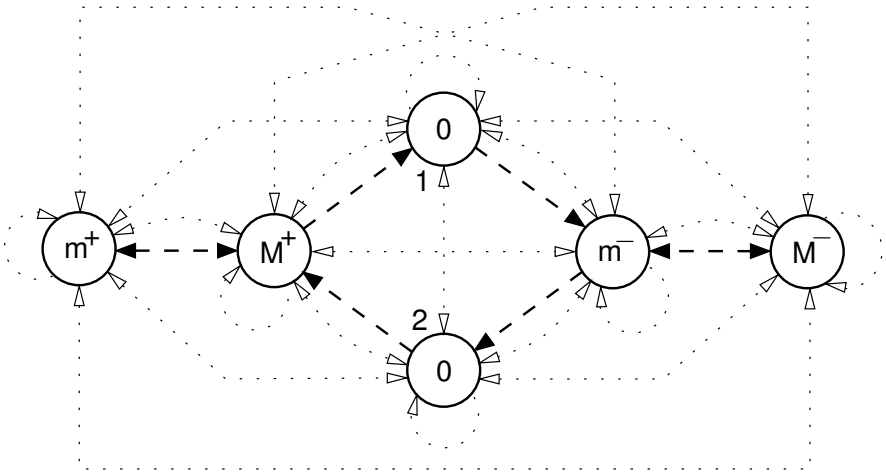


Figure 8.10: TOG(LEY, G_{f3})

8.3 Curvature types

In Chapter 3, we covered "Qualitative outline theory" (QOT), which uses a set of seven qualitative curvature types for describing outlines. In Appendix A, we provide a regular grammar that generates all of the canonical strings of curvature-type symbols. In this section, we show how each of the curvature types can be specified as a complex token and that the seven curvature types collectively define an LFS. We then construct a TOG for the curvature types with respect to the full scope of curves given by the level-2 atomic TOG, comparing the constructed TOG with the grammar given in the appendix.

8.3.1 LFS specification

There are three linelike types (/, ⊃, and ⊂) and four pointlike types (>, <, ≻, and ≺). Our aim is to specify each of the types as a complex token using atoms with as few qualitative components as possible. First, consider the linelike types. The straight-line type, /, represents intervals of zero curvature; so / can be defined at level 2 and identified with all occurrences of $\underline{\mathsf{Z}}$. Convex curve segments, which are represented by ⊃, correspond to segments of curve which do not contain any straight-line segments or concavities. In terms of curvature variation, a convex curve segment is an interval over which curvature is non-negative and, if the curvature is zero, it is zero only at an isolated point. In other words, a convex curve segment does not simply correspond to intervals over which the curvature is wholly positive (i.e., $\underline{\mathsf{P}}$); within the segment, there may be points where the curvature is undefined (as when two circular-arc segments of positive curvature are smoothly joined, such that the tangent bearing is defined at the join but the curvature is not) and there may be points where the curvature momentarily takes the value zero (corresponding to the kinds of minima disallowed by the schemes of Hoffman & Richards and Leyton). Such point occurrences are illustrated in Figure 8.11 by a convex curve segment with the description $\underline{\mathsf{P}}\,\mathsf{U}_c\,\underline{\mathsf{P}}\,\mathsf{Z}\,\underline{\mathsf{P}}$. The type ⊃, then, corresponds to strings of atoms that begin and end with an interval of positive curvature ($\underline{\mathsf{P}}$) and contain zero or more points of undefined curvature (U_c) and zero or more points of zero curvature (Z). Segments of curve that are *concave*, and represented by ⊂, are analogous to convex curve segments in their atomic composition, with occurrences of $\underline{\mathsf{P}}$ replaced with $\underline{\mathsf{N}}$.

The four pointlike types can be specified at any level in the hierarchy. There are two types for angles: outward-pointing (>) and inward-pointing (<). Similarly, there is a type representing outward-pointing cusps (≻) and another for representing inward-pointing ones (≺). All seven of the curvature types, then, can be specified using atoms at level 2 of the atomic hierarchy. The set of specifications, QOT, is given in Table 8.9. All of the curvature types, apart from the convex and concave curve-segment types, represent occurrences of a *single* level-2 atom.

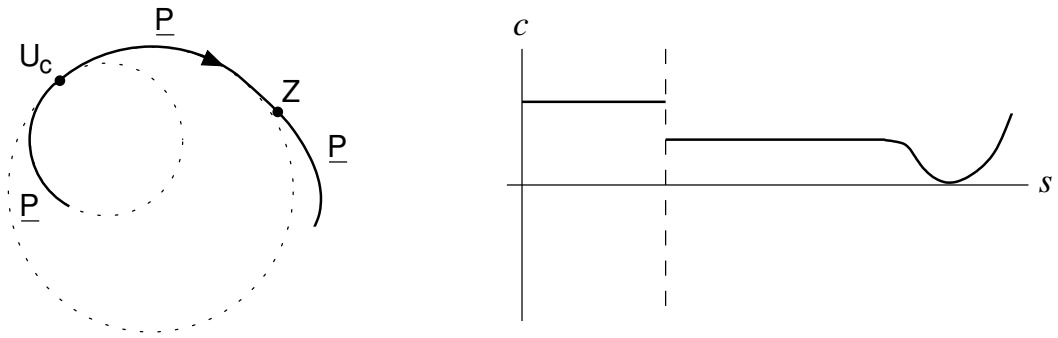


Figure 8.11: An exemplar curvature plot of the $\supset$ curvature type

$$
\begin{aligned}
QOT = \{ \quad \supset &= \{ [\underline{\mathsf{P}}\,(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{U_c}\,\underline{\mathsf{P}})^*\,] \},\\
\subset &= \{ [\underline{\mathsf{N}}\,(\mathsf{Z}\,\underline{\mathsf{N}} + \mathsf{U_c}\,\underline{\mathsf{N}})^*\,] \},\\
/ &= \{ [\underline{\mathsf{Z}}] \},\\
> &= \{ [\mathsf{U}_{>}] \},\\
< &= \{ [\mathsf{U}_{<}] \},\\
\succ &= \{ [\mathsf{U}_{\succ}] \},\\
\prec &= \{ [\mathsf{U}_{\prec}] \} \quad \}
\end{aligned}
$$

Table 8.9: The token specifications for the curvature types of QOT

Now we need to check that QOT is indeed an LFS. Five of the seven tokens ($/, >, <, \succ$, and $\prec$) have identities that consist of single atoms, so they must all be MAMO (lemma 8.2). The two remaining tokens, $\supset$ and $\subset$, are also MAMO by virtue of the subsumed predicate holding whenever two triples may overlap. We will prove this for $\supset$:

Consider a valid alignment, A, of two triples in $\supset$, t and t', such that t may-overlap t' holds. The identity of t is $\underline{\mathsf{P}}\,(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{U_c}\,\underline{\mathsf{P}})^a$, where $a \geq 2$. The first $\underline{\mathsf{P}}$ of the identity of t' is aligned with one of the $\underline{\mathsf{P}}$s of the identity of t. Since t may-overlap t' holds, the identity of t' must extend beyond the identity of t. The string of atoms that extends beyond the identity of t is of the form $(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{U_c}\,\underline{\mathsf{P}})^b$, where $b \geq 1$. The valid alignment A is illustrated in Table 8.10, where x is used to denote an occurrence of Z or $\mathsf{U_c}$. For subsumed$(\{A\}, \supset)$ to

hold, there must exist a triple in $\supset$ whose identity is $\underline{\mathsf{P}}\,(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{U_c}\,\underline{\mathsf{P}})^a\,(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{U_c}\,\underline{\mathsf{P}})^b$, which simplifies to $\underline{\mathsf{P}}\,(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{U_c}\,\underline{\mathsf{P}})^c$, where $c \geq 3$. As this is clearly the case given the specification of $\supset$, we conclude that $\mathsf{subsumed}(\{A\}, \supset)$ must hold and that, therefore, $\supset$ is MAMO. $\square$

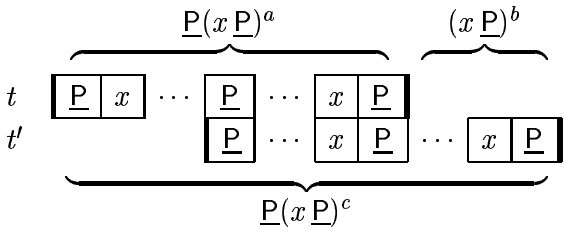


Table 8.10: A valid alignment of t $\mathsf{may\text{-}overlap}$ t', where $t, t' \in \supset$

We have verified that all of the tokens in QOT are MAMO. All that remains is to check that no two tokens of QOT interfere. A prerequisite for two tokens, T and T', to interfere is that a triple relation holds between $t \in T$ and $t' \in T'$. However, if T contains only triples with identities that consist of atoms taken from some set A, and T' contains only triples with identities that consist of atoms taken from some set B, then, if $A \cap B = \emptyset$, none of the thirteen triple relations can hold between t and t', so T and T' cannot interfere. Inspecting the specifications of the tokens in QOT, using this heuristic, reveals that no two tokens interfere. We conclude, therefore, that QOT is an LFS.

8.3.2 TOG construction

The set of curvature types is designed to be complete, in the sense that all of the curves that fall within the scope of our theory have non-empty curvature-type descriptions. More specifically, every point on a curve should form part of the identity, or constitute the entire identity, of exactly one of the seven curvature types. When constructing the TOG, then, we will use the level-2 atomic TOG as the scope graph (Figure 4.4, p. 88), which we will refer to as G_{f2}. Our expectation is that the constructed TOG for QOT will contain only strong connections.

Preparation of QOT

All of the specifications in QOT have implicit leading and trailing contexts, so every token needs to be converted into explicit-context form. As an example, consider /, and the conversion of $[\,\underline{\mathsf{Z}}\,]$ into a set, A, of corresponding triples, with leading and trailing contexts of length one. The conversion is carried out using G_{f2}. Each path in G_{f2} of the form $x \to \underline{\mathsf{Z}} \to y$ yields an element, $x\,[\,\underline{\mathsf{Z}}\,]\,y$, of A. Recall that, during triple conversion, the scope graph is assumed to contain two extra nodes for curve end-points (represented by "$\diamond$") that are connected to all of the interval-atom nodes in the graph (see Section 7.3.1). The specification for /, then, is converted into the following set of explicit-context triples:

$$\{\, x\,[\,\underline{\mathsf{Z}}\,]\,y \;\mid\; x, y \in \{\underline{\mathsf{P}}, \underline{\mathsf{N}}, \mathsf{U}_{>}, \mathsf{U}_{<}, \mathsf{U}_{\succ}, \mathsf{U}_{\prec}, \mathsf{U}_{\mathsf{c}}, \diamond\}\,\}$$

All of the other tokens in QOT are converted in a similar manner. As well as triple conversion, the preparation of an LFS involves a check to ensure that every triple is compatible with the scope graph; G_{f2} in this case. Clearly, the triples of QOT *must* be compatible with G_{f2}, because there cannot exist a valid string of level-2 atoms that is *not* derivable from the level-2 atomic TOG. The prepared specification of QOT is given in Table 8.11.

$$\begin{aligned}
QOT(G_{f2}) \;=\; \{\;\; \supset \;&=\; \{\, x\,[\,\underline{\mathsf{P}}(\mathsf{Z}\,\underline{\mathsf{P}} + \mathsf{U}_{\mathsf{c}}\,\underline{\mathsf{P}})^*\,]\,y \;\mid\; x, y \in \{\underline{\mathsf{Z}}, \mathsf{Z}, \mathsf{U}_{>}, \mathsf{U}_{<}, \mathsf{U}_{\succ}, \mathsf{U}_{\prec}, \mathsf{U}_{\mathsf{c}}, \diamond\}\,\},\\
\subset \;&=\; \{\, x\,[\,\underline{\mathsf{N}}(\mathsf{Z}\,\underline{\mathsf{N}} + \mathsf{U}_{\mathsf{c}}\,\underline{\mathsf{N}})^*\,]\,y \;\mid\; x, y \in \{\underline{\mathsf{Z}}, \mathsf{Z}, \mathsf{U}_{>}, \mathsf{U}_{<}, \mathsf{U}_{\succ}, \mathsf{U}_{\prec}, \mathsf{U}_{\mathsf{c}}, \diamond\}\,\},\\
/ \;&=\; \{\, x\,[\,\underline{\mathsf{Z}}\,]\,y \;\mid\; x, y \in \{\underline{\mathsf{P}}, \underline{\mathsf{N}}, \mathsf{U}_{>}, \mathsf{U}_{<}, \mathsf{U}_{\succ}, \mathsf{U}_{\prec}, \mathsf{U}_{\mathsf{c}}, \diamond\}\,\},\\
> \;&=\; \{\, x\,[\,\mathsf{U}_{>}\,]\,y \;\mid\; x, y \in \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}, \underline{\mathsf{N}}\}\,\},\\
< \;&=\; \{\, x\,[\,\mathsf{U}_{<}\,]\,y \;\mid\; x, y \in \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}, \underline{\mathsf{N}}\}\,\},\\
\succ \;&=\; \{\, x\,[\,\mathsf{U}_{\succ}\,]\,y \;\mid\; (x = \underline{\mathsf{N}} \wedge y \in \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}, \underline{\mathsf{N}}\}) \vee (x \in \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}\} \wedge y = \underline{\mathsf{N}})\,\},\\
\prec \;&=\; \{\, x\,[\,\mathsf{U}_{\prec}\,]\,y \;\mid\; (x = \underline{\mathsf{P}} \wedge y \in \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}, \underline{\mathsf{N}}\}) \vee (x \in \{\underline{\mathsf{Z}}, \underline{\mathsf{N}}\} \wedge y = \underline{\mathsf{P}})\,\}\;\;\}
\end{aligned}$$

Table 8.11: The prepared specification of QOT, with respect to G_{f2}

Note that our original specifications were not restricted to closed curves, so the linelike types, each of which may both begin and end an open curve, include triples with the curve end-point symbol as their leading and/or trailing context(s).

Determining the graph nodes

The set of curvature types closely resembles the set of level-2 atomic tokens, in the sense that five of the curvature types are essentially aliases for level-2 atoms. In particular, / represents $\underline{\mathsf{Z}}$, and the four kink tokens, $\mathsf{U}_{>}$, $\mathsf{U}_{<}$, $\mathsf{U}_{\succ}$, and $\mathsf{U}_{\prec}$ are represented by $>$, $<$, $\succ$, and $\prec$ respectively. As we would expect, then, the number of nodes in the level-2 atomic TOG for the aforementioned atoms reflects the number of nodes required by each of the five curvature types in TOG(QOT, G_{f2}). In particular, /, $>$, and $<$ require one node each, whereas $\succ$ and $\prec$ require two nodes each. As a reminder of the node creation procedure, consider the assignment of nodes to $\succ$. First, the characterisation set for $\succ$ is generated, by extracting the starting and finishing substrings from each triple of the prepared specification of $\succ$:

$$\succ_{ch} = \{\, \langle \underline{\mathsf{N}}\,\mathsf{U}_{\succ}, \mathsf{U}_{\succ}\,\underline{\mathsf{P}} \rangle,\ \langle \underline{\mathsf{N}}\,\mathsf{U}_{\succ}, \mathsf{U}_{\succ}\,\underline{\mathsf{Z}} \rangle,\ \langle \underline{\mathsf{N}}\,\mathsf{U}_{\succ}, \mathsf{U}_{\succ}\,\underline{\mathsf{N}} \rangle,\ \langle \underline{\mathsf{P}}\,\mathsf{U}_{\succ}, \mathsf{U}_{\succ}\,\underline{\mathsf{N}} \rangle,\ \langle \underline{\mathsf{Z}}\,\mathsf{U}_{\succ}, \mathsf{U}_{\succ}\,\underline{\mathsf{N}} \rangle \,\}$$

Next, $\succ_{ch}$ is partitioned using Algorithm B.1, yielding two subsets, one of which contains the first three elements of $\succ_{ch}$ (as listed) and the other of which contains the last two elements. A node is then created for each of the subsets, as follows:

$$\begin{aligned} N_1(\succ) &= \langle\, \{\underline{\mathsf{N}}\,\mathsf{U}_{\succ}\},\ \{\mathsf{U}_{\succ}\,\underline{\mathsf{P}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\} \,\rangle \\ N_2(\succ) &= \langle\, \{\underline{\mathsf{P}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\succ}\},\ \{\mathsf{U}_{\succ}\,\underline{\mathsf{N}}\} \,\rangle \end{aligned}$$

The two remaining curvature types, $\supset$ and $\subset$, require one node each, bringing the total number of nodes required by TOG(QOT, G_{f2}) to nine. The complete set of nodes is given in Table 8.12. In the constructed graph, the nodes for the linelike types are shown as rectangles, and those for the pointlike types as circles.

Connecting the nodes

As expected, there are a number of strong connections in the constructed TOG. QOT is a no-merging LFS, so all of the strong connections are of the meeting kind. As an example, consider the strong connections that exist from $N_1(/)$ to the other nodes in the graph. The node $N_1(/)$ is associated with the following set, F, of distinct finishing substrings:

$$
\begin{array}{rcl}
\mathcal{N}(QOT(G)) = \{\ N_1(\supset) & = & \langle\, \{\underline{\mathsf{Z}}\,\underline{\mathsf{P}},\ \mathsf{Z}\,\underline{\mathsf{P}},\ \mathsf{U}_{\mathsf{c}}\,\underline{\mathsf{P}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{P}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{P}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{P}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{P}},\ \diamond\,\underline{\mathsf{P}}\},\\
 & & \quad \{\underline{\mathsf{P}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{P}}\,\mathsf{Z},\ \underline{\mathsf{P}}\,\mathsf{U}_{\mathsf{c}},\ \underline{\mathsf{P}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{P}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{P}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{P}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{P}}\diamond\}\,\rangle,\\
N_1(\subset) & = & \langle\, \{\underline{\mathsf{Z}}\,\underline{\mathsf{N}},\ \mathsf{Z}\,\underline{\mathsf{N}},\ \mathsf{U}_{\mathsf{c}}\,\underline{\mathsf{N}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{N}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{N}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{N}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{N}},\ \diamond\,\underline{\mathsf{N}}\},\\
 & & \quad \{\underline{\mathsf{N}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{N}}\,\mathsf{Z},\ \underline{\mathsf{N}}\,\mathsf{U}_{\mathsf{c}},\ \underline{\mathsf{N}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{N}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{N}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{N}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{N}}\diamond\}\,\rangle,\\
N_1(/) & = & \langle\, \{\underline{\mathsf{P}}\,\underline{\mathsf{Z}},\ \underline{\mathsf{N}}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\mathsf{c}}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{Z}},\ \diamond\,\underline{\mathsf{Z}}\},\\
 & & \quad \{\underline{\mathsf{Z}}\,\underline{\mathsf{P}},\ \underline{\mathsf{Z}}\,\underline{\mathsf{N}},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\mathsf{c}},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{Z}}\diamond\}\,\rangle,\\
N_1(\succ) & = & \langle\, \{\underline{\mathsf{P}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{N}}\,\underline{\mathsf{U}_{\succ}}\},\ \{\mathsf{U}_{\succ}\,\underline{\mathsf{P}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\}\,\rangle,\\
N_1(\prec) & = & \langle\, \{\underline{\mathsf{P}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{N}}\,\underline{\mathsf{U}_{\prec}}\},\ \{\mathsf{U}_{\prec}\,\underline{\mathsf{P}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{N}}\}\,\rangle,\\
N_1(\succ) & = & \langle\, \{\underline{\mathsf{N}}\,\mathsf{U}_{\succ}\},\ \{\mathsf{U}_{\succ}\,\underline{\mathsf{P}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\}\,\rangle,\\
N_2(\succ) & = & \langle\, \{\underline{\mathsf{P}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\succ}\},\ \{\mathsf{U}_{\succ}\,\underline{\mathsf{N}}\}\,\rangle,\\
N_1(\prec) & = & \langle\, \{\underline{\mathsf{P}}\,\mathsf{U}_{\prec}\},\ \{\mathsf{U}_{\prec}\,\underline{\mathsf{P}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{Z}},\ \mathsf{U}_{\prec}\,\underline{\mathsf{N}}\}\,\rangle,\\
N_2(\prec) & = & \langle\, \{\underline{\mathsf{Z}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{N}}\,\mathsf{U}_{\prec}\},\ \{\mathsf{U}_{\prec}\,\underline{\mathsf{P}}\}\,\rangle\quad \}
\end{array}
$$

Table 8.12: The nodes for TOG(QOT, G_{f2})

$$F = \{\, \underline{\mathsf{Z}}\,\underline{\mathsf{P}},\ \underline{\mathsf{Z}}\,\underline{\mathsf{N}},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\mathsf{c}},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\prec},\ \underline{\mathsf{Z}}\diamond\,\}$$

There is a strong connection from $N_1(/)$ to node X *iff* at least one of the substrings in F is contained in the set of distinct starting substrings, S', associated with X. For $X = N_2(\succ)$, we have $S' = \{\underline{\mathsf{P}}\,\mathsf{U}_{\succ},\ \underline{\mathsf{Z}}\,\mathsf{U}_{\succ}\}$. The element $\underline{\mathsf{Z}}\,\mathsf{U}_{\succ}$ of S' is shared by F, so $N_1(/) \xrightarrow{S} N_2(\succ)$ exists. The six strong connections in TOG(QOT, G_{f2}) that originate from $N_1(/)$ are listed in Table 8.13. Altogether there are thirty strong connections in the graph and, as expected, there are no strong connections between any of the nodes representing pointlike types. Interestingly, there are no strong connections between the nodes for $\supset$ and $\subset$ either, as we expected there to be. Instead, the nodes for $\supset$ and $\subset$ are linked by two connections of the *weak* kind. Consider the determination of a weak connection from $N_1(\supset)$ to $N_1(\subset)$. The set of distinct finishing substrings of $N_1(\supset)$ contains the two substrings $f_1 = b_1\, c_1 = \underline{\mathsf{P}}\,\mathsf{U}_{\mathsf{c}}$ and $f_2 = b_2\, c_2 = \underline{\mathsf{P}}\,\mathsf{Z}$, and the set of distinct starting substrings of $N_1(\subset)$ contains the two substrings $s'_1 = u_1\, v_1 = \mathsf{U}_{\mathsf{c}}\,\underline{\mathsf{N}}$ and $s'_2 = u_2\, v_2 = \mathsf{Z}\,\underline{\mathsf{N}}$. Taking f_1 and s'_1 first, we have $c_1 = u_1$, so triples characterised by f_1 and s'_1 can be joined together such that their identities are

Connection	$f \in F = s' \in S'$
$N_1(/) \overset{S}{\rightarrow} N_1(\supset)$	$\underline{\mathsf{Z}}\,\underline{\mathsf{P}}$
$N_1(/) \overset{S}{\rightarrow} N_1(\subset)$	$\underline{\mathsf{Z}}\,\underline{\mathsf{N}}$
$N_1(/) \overset{S}{\rightarrow} N_1(>)$	$\underline{\mathsf{Z}}\,\mathsf{U}_{>}$
$N_1(/) \overset{S}{\rightarrow} N_1(<)$	$\underline{\mathsf{Z}}\,\mathsf{U}_{<}$
$N_1(/) \overset{S}{\rightarrow} N_2(\succ)$	$\underline{\mathsf{Z}}\,\mathsf{U}_{\succ}$
$N_1(/) \overset{S}{\rightarrow} N_2(\prec)$	$\underline{\mathsf{Z}}\,\mathsf{U}_{\prec}$

Table 8.13: The strong connections originating from $N_1(/)$ in TOG(QOT, G_{f2})

separated by the single atom $\mathsf{U_c}$. The triple $\underline{\mathsf{P}}\,[\,\mathsf{U_c}\,]\,\underline{\mathsf{N}}$, however, does not exist in QOT, which means that there *does* exist a weak connection from $N_1(\supset)$ to $N_1(\subset)$. In addition to f_1 and s'_1, the substrings f_2 and s'_2 *also* imply $N_1(\supset) \overset{W}{\rightarrow} N_1(\subset)$. There exist analogous substrings in the set of distinct starting substrings of $N_1(\supset)$ and the set of distinct finishing substrings of $N_1(\subset)$ that imply a weak connection in the *other* direction: $N_1(\subset) \overset{W}{\rightarrow} N_1(\supset)$. There are four other weak connections in the graph, involving the node for /. The six weak connections of TOG(QOT, G_{f2}) are detailed in Table 8.14.

	Connection	$f \in F$	$s' \in S'$	*Triple not in QOT*
1	$N_1(/) \overset{W}{\rightarrow} N_1(\supset)$	$\underline{\mathsf{Z}}\,\mathsf{U_c}$	$\mathsf{U_c}\,\underline{\mathsf{P}}$	$\underline{\mathsf{Z}}\,[\,\mathsf{U_c}\,]\,\underline{\mathsf{P}}$
2	$N_1(/) \overset{W}{\rightarrow} N_1(\subset)$	$\underline{\mathsf{Z}}\,\mathsf{U_c}$	$\mathsf{U_c}\,\underline{\mathsf{N}}$	$\underline{\mathsf{Z}}\,[\,\mathsf{U_c}\,]\,\underline{\mathsf{N}}$
3	$N_1(\supset) \overset{W}{\rightarrow} N_1(/)$	$\underline{\mathsf{P}}\,\mathsf{U_c}$	$\mathsf{U_c}\,\underline{\mathsf{Z}}$	$\underline{\mathsf{P}}\,[\,\mathsf{U_c}\,]\,\underline{\mathsf{Z}}$
4	$N_1(\supset) \overset{W}{\rightarrow} N_1(\subset)$	$\underline{\mathsf{P}}\,\mathsf{U_c}$ $\underline{\mathsf{P}}\,\mathsf{Z}$	$\mathsf{U_c}\,\underline{\mathsf{N}}$ $\mathsf{Z}\,\underline{\mathsf{N}}$	$\underline{\mathsf{P}}\,[\,\mathsf{U_c}\,]\,\underline{\mathsf{N}}$ $\underline{\mathsf{P}}\,[\,\mathsf{Z}\,]\,\underline{\mathsf{N}}$
5	$N_1(\subset) \overset{W}{\rightarrow} N_1(/)$	$\underline{\mathsf{N}}\,\mathsf{U_c}$	$\mathsf{U_c}\,\underline{\mathsf{Z}}$	$\underline{\mathsf{N}}\,[\,\mathsf{U_c}\,]\,\underline{\mathsf{Z}}$
6	$N_1(\subset) \overset{W}{\rightarrow} N_1(\supset)$	$\underline{\mathsf{N}}\,\mathsf{U_c}$ $\underline{\mathsf{N}}\,\mathsf{Z}$	$\mathsf{U_c}\,\underline{\mathsf{P}}$ $\mathsf{Z}\,\underline{\mathsf{P}}$	$\underline{\mathsf{N}}\,[\,\mathsf{U_c}\,]\,\underline{\mathsf{P}}$ $\underline{\mathsf{N}}\,[\,\mathsf{Z}\,]\,\underline{\mathsf{P}}$

Table 8.14: The six weak connections in TOG(QOT, G_{f2})

All that remains is to show that there are no remote connections in TOG(QOT, G_{f2}). There are 81 potential remote connections that need to be checked.[4] As an illustrative example of how these could be checked, we will prove that no remote connection exists from $N_2(\succ) = \langle S, F\rangle$ to $N_1(<) = \langle S', F'\rangle$:

We have $F = \{\mathsf{U}_{\succ}\,\underline{\mathsf{N}}\}$ and $S' = \{\underline{\mathsf{P}}\,\mathsf{U}_{<}, \underline{\mathsf{Z}}\,\mathsf{U}_{<}, \underline{\mathsf{N}}\,\mathsf{U}_{<}\}$. Therefore, there are three sets of strings that are relevant to the existence of $N_2(\succ) \xrightarrow{R} N_1(<)$: $\alpha = \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\,X\,\underline{\mathsf{P}}\,\mathsf{U}_{<}$, $\beta = \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\,X'\,\underline{\mathsf{Z}}\,\mathsf{U}_{<}$, and $\gamma = \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\,X''\,\underline{\mathsf{N}}\,\mathsf{U}_{<}$. We will consider each set of strings in turn:

- $\alpha = \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\,X\,\underline{\mathsf{P}}\,\mathsf{U}_{<}$

  The substring X cannot be of length zero, because $\underline{\mathsf{N}} \rightarrow \underline{\mathsf{P}}$ does not exist in G_{f2}. Every X of length greater than zero must begin with an atom taken from the set $A = \{\underline{\mathsf{Z}}, \mathsf{Z}, \mathsf{U}_{>}, \mathsf{U}_{<}, \mathsf{U}_{\succ}, \mathsf{U}_{\prec}, \mathsf{U}_{\mathsf{c}}\}$. Therefore, α must be of the form $\mathsf{U}_{\succ}\,\underline{\mathsf{N}}\,a \cdots \underline{\mathsf{P}}\,\mathsf{U}_{<}$, where $a \in A$. Every triple of the form $\mathsf{U}_{\succ}\,[\,\underline{\mathsf{N}}\,]\,a$ is an element of the specification of $\subset$. Therefore, $\subset$ fits to *all* of the strings given by α.

- $\beta = \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\,X'\,\underline{\mathsf{Z}}\,\mathsf{U}_{<}$

  The substring X' *can* be empty, because $\underline{\mathsf{N}} \rightarrow \underline{\mathsf{Z}}$ exists in G_{f2}. In which case, the triples $\mathsf{U}_{\prec}\,[\,\underline{\mathsf{N}}\,]\,\underline{\mathsf{Z}}$ (from $\subset$) and $\underline{\mathsf{N}}\,[\,\underline{\mathsf{Z}}\,]\,\mathsf{U}_{<}$ (from /) fit β. Apart from this difference, we may use the same reasoning as for the strings given by α to conclude that $\subset$ fits to all of the strings given by β.

- $\gamma = \mathsf{U}_{\succ}\,\underline{\mathsf{N}}\,X''\,\underline{\mathsf{N}}\,\mathsf{U}_{<}$

  The same reasoning applies here as for the set of paths given by α, with "$\underline{\mathsf{P}}$" replaced by "$\underline{\mathsf{N}}$", and "α" by "γ".

For each of the three sets of paths, we have shown that at least one token fits to every path in the set. We may therefore conclude that $N_2(\succ) \xrightarrow{R} N_1(<)$ does not exist.

An enumeration of the connections in TOG(QOT, G_{f2}) is provided in Table 8.15; the graph itself is given in Figure 8.12.

[4] Note that we can prove that no remote connections exist in TOG(QOT, G_{f2}) using a high-level argument similar to that given for TOG(LFS_1, G_1), on page 152.

	$N_1(/)$	$N_1(\supset)$	$N_1(\subset)$	$N_1(>)$	$N_1(<)$	$N_1(\succ)$	$N_2(\succ)$	$N_1(\prec)$	$N_2(\prec)$
$N_1(/)$		SW	SW	S	S		S		S
$N_1(\supset)$	SW		W	S	S		S	S	
$N_1(\subset)$	SW	W		S	S	S			S
$N_1(>)$	S	S	S						
$N_1(<)$	S	S	S						
$N_1(\succ)$	S	S	S						
$N_2(\succ)$			S						
$N_1(\prec)$	S	S	S						
$N_2(\prec)$		S							

Table 8.15: The node connections for TOG(QOT, G_{f2})

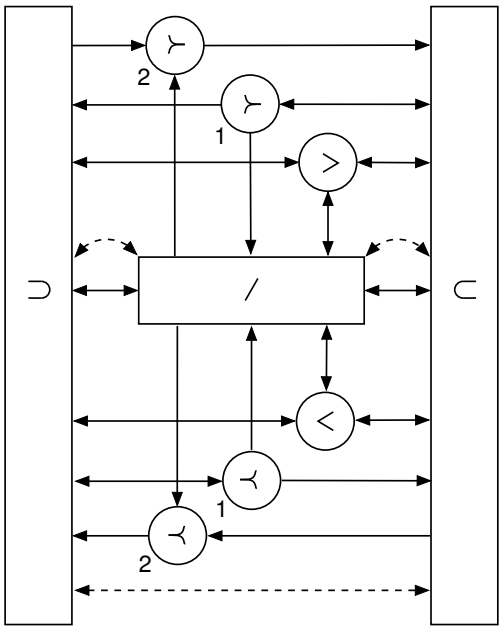


Figure 8.12: TOG(QOT, G_{f2})

The absence of any remote connections in TOG(QOT, G_{f2}), together with the large number of strong connections compared with weak connections, suggests that QOT adequately represents the scope of curves defined by G_{f2}. Since G_{f2} corresponds to "full" scope, QOT may be regarded as a *general* scheme, because it is capable of representing a wide range of curves. This is not surprising, because the set of curvature types provided by QOT is very similar to the set of atoms and kink tokens provided at level 2 of the atomic hierarchy. This similarity is reflected by the fact that TOG(QOT, G_{f2}) bears a strong resemblance to G_{f2} (p. 88).

The presence of weak connections in TOG(QOT, G_{f2}) suggests some kind of weakness or deficiency in QOT. Closer inspection of Table 8.14 reveals that there are two level-2 atoms that are not fully accounted for by the tokens of QOT: $\mathsf{U_c}$ and Z. In our discussion of QOT in Chapter 3, we noted that a separate curvature type for a point of inflection is not necessary, as its presence can be inferred from adjacent occurrences of $\supset$ and $\subset$. The decision to omit points of inflection accounts for weak connections 4 and 6 in the table. There are two kinds of curve-inflection points, those where the curvature smoothly passes through zero (Z), and those where the tangent bearing is defined but the curvature is not ($\mathsf{U_c}$). The latter kind is illustrated by the section of curve shown on the right of Figure 8.13, formed by joining together two circular-arc segments.[5]

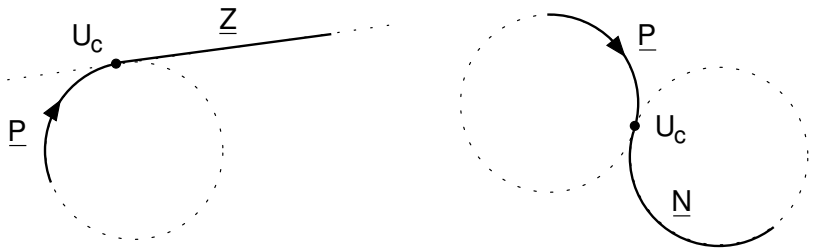


Figure 8.13: Occurrences of $\mathsf{U_c}$ not accounted for by QOT

The remaining four weak connections are due to unrepresented occurrences of $\mathsf{U_c}$ be-

[5]Note that, for a $\mathsf{U_c}$ inflection, the adjoining segments of curve need not be circular arcs. The segments just need to join together such that the tangent bearing is defined at the join, but the curvature is not.

tween a straight-line segment and an interval of positive or negative curvature. An instance of such a point, preceded by an interval of positive curvature and followed by a straight-line segment (weak connection 3), is shown on the left of Figure 8.13.

The existence of weak connections in a non-atomic TOG signify that, when the tokens of the underlying LFS are fitted to an atomic description, there may be atoms that are not accounted for, i.e., atoms which do not form part of the identity of any fitted triple. It is important to note that the points left unaccounted for by QOT are not points that may be regarded as perceptually salient. For this reason, their omission may be considered acceptable, and need not be seen as a deficiency. The weak connections could be removed, if necessary, by introducing one or more extra curvature types into the QOT scheme. One possibility is to include a curve-inflection type, Inf, together with a type representing $\mathsf{U_c}$s that occur between straight-line segments and intervals of positive or negative curvature (which we refer to here as X)[6], giving us "$QOT9$":

$$\begin{aligned} \mathsf{Inf} &= \{\underline{\mathsf{P}}[\mathsf{U_c}]\underline{\mathsf{N}}, \underline{\mathsf{P}}[\mathsf{Z}]\underline{\mathsf{N}}, \underline{\mathsf{N}}[\mathsf{U_c}]\underline{\mathsf{P}}, \underline{\mathsf{N}}[\mathsf{Z}]\underline{\mathsf{P}}\} \\ \mathsf{X} &= \{\underline{\mathsf{Z}}[\mathsf{U_c}]\underline{\mathsf{P}}, \underline{\mathsf{Z}}[\mathsf{U_c}]\underline{\mathsf{N}}, \underline{\mathsf{P}}[\mathsf{U_c}]\underline{\mathsf{Z}}, \underline{\mathsf{N}}[\mathsf{U_c}]\underline{\mathsf{Z}}\} \end{aligned}$$

The ordering constraints encoded by TOG(QOT, G_{f2}) are consistent with those given in Section 3.2.4. The constructed TOG relates to the grammar given in Appendix A in the following way: the set of strings that can be generated from the TOG by starting at $N_1(\supset)$ and tracing a route that ends at a node *other* than $N_1(\supset)$, is the same as the set of strings generated by the first two rules of the grammar, together with the block of rules of the form $A_\alpha \rightarrow \beta$ and $W \rightarrow \beta$.

8.4 Summary

In this chapter, we have analysed the three qualitative boundary-based schemes from Chapters 2 and 3, and shown that each one of them is an LFS. The specifications of the complex tokens for each scheme was shown to be straightforward; no contexts of length

[6]The author has been unsuccessful in finding a succinct label for such points, providing evidence, perhaps, that points of type X are fairly esoteric and can be left unspecified.

greater than one atom were needed by any of the tokens, in keeping with the restriction given in Section 7.1.1.

The range of curves adequately described by the schemes was compared by constructing TOGs with respect to particular scope graphs. For the contour codons and extremum primitives, we constructed TOGs with respect to the full scope of curves (given by the level-3 atomic TOG) and found that the graphs were fully connected with remote connections, indicating inadequate descriptive power with respect to the full scope of curves. In particular, polygonal shapes and curves with points of undefined tangent bearing and/or curvature cannot be adequately described using contour codons or extremum primitives. We also constructed TOGs for the contour codons and extremum primitives with respect to a restricted scope graph, representing the class of curves implied by the author's of the two schemes. The resultant TOGs, in this case, contained *no* remote connections, indicating the suitability of the scope graph for both schemes. For the curvature types, we constructed a TOG with respect to the full scope of curves given by the level-2 atomic TOG. The absence of any remote connections in the constructed graph, in this case, indicating that the set of curvature types can adequately represent a wider range of curves than either the set of contour codons or the set of extremum primitives.

One interesting difference between the schemes is given by information loss. Any description of a curve in terms of contour codons or extremum primitives can be translated into a single atomic description at level 3, i.e., each codon and extremum primitive corresponds to a finite fixed string of atoms. This is not the case with the curvature types. The two curve-segment types correspond to an infinite number of atomic strings, so their presence in a string of curvature types does *not* correspond to a single string of atoms. In this sense, a curvature-type description loses information about the underlying atomic description of a curve. However, this is exactly the property which provides the set of curvature types with the capability to adequately represent a wide range of curves, while still consisting of a small and manageable number of primitives.

Chapter 9

Evaluation

In this chapter, we evaluate the theory of boundary-based shape representation developed in Chapters 4 to 7 of this thesis: qualitative boundary theory. The theory essentially consists of three parts. Here, we discuss the strengths and weaknesses of each part of the theory in turn.

9.1 Summary of qualitative boundary theory

In Chapters 4 to 7 of this thesis, we have developed a theory of the boundary-based approach to the qualitative representation of two-dimensional shape, which we call *qualitative boundary theory* (QBT). In the previous chapter, we applied QBT by showing how existing boundary-based schemes are ultimately specifiable in terms of the basic shape descriptors provided by QBT. The basic building blocks of the theory, together with the support the theory provides for the specification of higher-level curve features, constitute a framework in which boundary-based representations of shape, such as the various schemes we have looked at, can be formally specified. QBT consists of three *parts*:

- *The atomic shape descriptors*

  We began, in Chapter 4, by deriving an unbounded hierarchy of atomic tokens based on combinations of qualitative components. Our components included tangent bearing and its successive derivatives with respect to arc length, i.e., curvature, the rate

of change of curvature, etc. These atomic tokens are the basic building blocks for representing the boundary of shapes and form the foundation of the theory. Shape features that are not expressible in terms of atomic tokens cannot be represented by any scheme derived from the theory.

- *Support for higher-level curve description*

  In Chapter 5, we began by showing that there are curve features which we would like to represent that do not correspond to single atomic tokens, such as curvature maxima and minima, for example. We developed a means by which strings of atomic tokens can be grouped together to represent higher-level curve features. These "complex" tokens utilise the notions of *identity* and *context*. A curve feature may have one or a (possibly infinite) number of atomic identities and each of its identities is associated with a set of contexts. Complex tokens do not increase the discriminatory power provided by the atomic tokens, rather, their purpose is to associate single labels with non-atomic curve features, i.e., they enable us to explicitly represent localised curve features of greater abstraction.

  Our concern in Chapter 6 was to consider how complex tokens can be collected together to form a boundary-based scheme for representing shape. We interpreted a single coherent scheme as one that represents a shape by a string of tokens, such that the features represented by the tokens do not overlap round the boundary of a shape. We defined a local-feature scheme (LFS) as one that meets this criterion. In Chapter 8, we were able to highlight the underlying similarities of existing boundary-based schemes, by showing that each is specifiable as an LFS.

- *Token-ordering graphs*

  In Chapter 4, token-ordering graphs (TOGs) were introduced, in the context of the atomic tokens. A TOG visually encodes the ordering constraints for a set of atomic *or* complex tokens. Nodes in a TOG are associated with tokens and directed edges with ordering constraints between tokens. A procedure is provided in Appendix B

for constructing a TOG for a set of atoms at a given level of the atomic hierarchy. In Chapter 7, we focussed our attention on the problem of constructing a TOG for a set of complex tokens defining an LFS. Such *non-atomic* TOGs make explicit the ordering constraints implicit in the token specifications contained in an LFS.

In the sections that follow, we focus on each part of the theory in turn, highlighting its strengths and weaknesses, and relating it to work from the literature.

9.2 The atomic shape descriptors

9.2.1 Design

The basic units provided by the theory for representing shapes are atomic tokens. Each atomic token (or *atom*) is identified with a sequence of qualitative components representing a partial curve state. The first value in a sequence represents tangent bearing (b), the second curvature (c), the third the rate of change of curvature (c'), and so on. We use two simple quantity spaces as our sets of allowable qualitative values. For tangent bearing, we choose only to distinguish between points where it is defined (D) and points where it is not (U), so we use the space $\{D, U\}$. Points where the tangent bearing is undefined correspond to curve features referred to as kinks. For all of the successive derivatives of tangent bearing (c, c', ...), we use the space $\{+, 0, -, U\}$, so we can distinguish between positive, zero, negative, and undefined values of a derivative. Our choice of quantity spaces ensures that atomic descriptions are invariant with respect to translation, rotation, and uniform scaling transformations. The hierarchy of atomic tokens is generated by considering only those sequences of qualitative components that are *valid* (see Section 4.1.1) and is unbounded because there is no limit imposed on the length of a sequence of qualitative components.

Although we combine qualitative components with a view to deriving descriptors for representing shape that characterise the tangent-bearing function[1], the analysis of qualitative components given in Chapter 4 provides a general technique for the qualitative

[1] Also known as the ψ-s curve (Ballard & Brown 1982).

representation of real-valued variables and their derivatives (linear functions). We will have more to say on this in the next chapter, when we highlight possible directions for further work (Section 10.3).

Completeness

By starting with tangent bearing and using quantity spaces for tangent bearing and its successive derivatives, such that there is a qualitative value for every possible underlying numeric value of each component, the set of atoms at each level of the hierarchy is *complete*, in the sense that an atomic description of a curve (at any level of the hierarchy) accounts for *all* of the points on the curve, i.e., every point is associated with exactly one atom of the description. The completeness of the sets of basic shape descriptors, with respect to a fairly general scope of curves, is more comprehensive than that provided by any of the existing boundary-based schemes we have looked at.

A consequence of completeness is the existence of atoms that are not perceptually salient. The atom U_c, for example, which exists at every level in the hierarchy, represents points where the tangent bearing is defined but the curvature is not. A U_c is not a point that can easily be identified on a curve by just inspecting the appearance of the curve. The number of point atoms that contain at least one undefined component increases as we move down the hierarchy, e.g., at level 2 there is just one (U_c) whereas at level 3 there are four (U_c, P^U, Z^U, and N^U). Although required for completeness, the existence of such atoms can, in fact, complicate the specification of curve features. Evidence for this is provided by the specifications of the curve-segment types of QOT (see Section 8.3.1).

Scope of curves represented

As a consequence of attending solely to the way in which tangent bearing changes locally along the bounding curve of a shape, the class of shapes that can be adequately represented by atomic tokens is restricted. To ensure that atomic descriptions are finite, we need to restrict our attention to piecewise uniform curves, as we did for QOT in Chapter 3.

We do not need to restrict ourselves to curves that are closed, however. As with QOT, it makes sense to disregard self-intersecting curves, because points of self-intersection are not taken into account by QBT. The problems that arise when describing curves containing crossing or "touching" points can be highlighted with reference to the three digits shown in Figure 9.1.

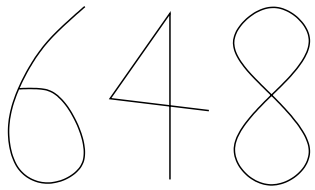


Figure 9.1: Curves that contain crossing and touching points

The first digit, "6", contains a touching point and the second digit, "4", contains a crossing point. It would be incorrect to say that these curves *cannot* be represented, but in any atomic description the touching and crossing points *themselves* are not represented. The third digit, "8", differs from the first two in that it is a closed curve. An additional problem arises in this case with respect to the assignment of figure and ground. If figure is interpreted as the two regions enclosed by the curve, then the relationship between the direction of curve traversal and the side of the curve that the figure lies to is compromised. To adequately represent such *critical points* (Freeman 1978), our theory needs to be extended in some way.

Kink tokens

At every level of the hierarchy there exists just a single atom representing kink points, labelled $\mathsf{U_b}$. Inward- and outward-pointing kinks, therefore, are not distinguishable. As a consequence, the discriminatory power of the *derived* atoms is limited for shapes that have tangent discontinuities. We could have chosen to use a different quantity space for tangent bearing. Instead of $\{D, U\}$, we could have used $\{D, U_+, U_-\}$, with the value U_+

indicating an outward-pointing kink and U_- an inward-pointing one; or vice versa. In the analysis of Hayes & Leyton (1989), all kinks are formed at extrema of curvature and are essentially interpreted as maxima and minima of curvature that go to infinity. Their *spike parity* of a kink reflects the sign of the infinite value. For Hayes and Leyton, undefined curvature presupposes undefined tangent bearing. In our theory, we make the distinction between a point where both the tangent bearing and the curvature are undefined and a point where the curvature is undefined but the tangent bearing is *defined*. If we did use the extended quantity space $\{D, U_+, U_-\}$ then, although the orientation of a kink point would be known, we would still have no way of determining its type, i.e., whether it is an angle or a cusp. For Hayes & Leyton (1989) this is not so important, since they do not readily distinguish between angles and cusps; instead, they consider only the orientation of a kink. Rather than use an extended quantity space, we chose to introduce a set of four kink tokens separately (Section 4.1.5). The fact that our analysis of qualitative components does not yield *all* of the desirable building blocks could be considered a weakness in our approach. Although the kink tokens are not atomic in the sense that they are not derived in the same way as the other atoms, they may be regarded as *non-derived atoms*, because they effectively replace $\mathsf{U_b}$ at every level of the atomic hierarchy.

9.2.2 Discriminatory power

Recall that discriminatory power relates to the capability that a set of shape descriptors has for distinguishing between two given shapes. Each level of the atomic hierarchy provides a complete set of descriptors for representing a shape. The first level, consisting as it does only of $\underline{\mathsf{D}}$ and the four kink tokens, provides the least amount of discriminatory power. Even so, level 1 is still capable of a certain amount of useful discrimination, e.g., polygons that differ as to the number of sides and angle types have different level-1 descriptions. If we are interested solely in the number and type of tangent discontinuities that a shape has, then descriptions at level 1 would suffice. In the second level of the hierarchy the expansive interval $\underline{\mathsf{D}}$, from the first level, yields five atoms that are capable, collectively, of

describing the different *kinds* of "D" intervals. At level 2, the extra component associated with each atom provides a qualitative value for curvature, so that a segment of curve can be classified as straight, convex, or concave. As we move further down the atomic hierarchy, the number of qualitative components increases and, therefore, so does the available discriminatory power. We learnt in the previous chapter that we only need to go as far as level 3 to specify the primitives of existing boundary-based schemes. This does not mean to say that atoms with more than three components are not useful, rather, it reflects the design criteria of the schemes we have looked at. Representations that are used for shape recognition tasks using image data may sensibly restrict themselves to the first three components of the curve state, because it is difficult to extract derivatives of a higher order.

An important characteristic of the hierarchy of atoms is that it provides discriminatory power of a *certain kind*. Given two different shapes, they may be indistinguishable regardless of the level at which one decides to describe them. A simple example is that two different four-sided convex polygons cannot be distinguished, e.g., a square cannot be distinguished from a rhombus. Also, certain kinds of individual shapes cannot be adequately described at *any* level of the hierarchy, such as the shape given in the last section of Chapter 3. The kind of discriminatory power provided by the atoms may be considered a limitation, serving to indicate that the atomic shape descriptors are unsuitable for certain kinds of tasks that require shape in two dimensions to be represented. There are two main reasons why atomic descriptions are inadequate in certain circumstances:

- *Atoms represent curve features that are localised*

  Each atom in a description is associated with a single point or an interval of points on the bounding curve of some shape. An atomic description represents a string of localised curve features that "join together" to form an open or closed curve. Information that might be regarded as global, or concerning the whole region enclosed by the boundary (in the case of a closed curve), such as symmetry or some measure of elongation, is lost in an atomic description. We cannot distinguish, for example, between ellipses that have different radii. Interestingly, the global property of con-

vexity *can* be determined from a description, because it requires only knowledge of curvature sign and the distinction between inward- and outward-pointing kinks.

- *Atoms are associated with sequences of components that are non-numeric*

  The atomic shape descriptors are based on quantity spaces and designed to be qualitative in nature. For this reason, purely quantitative differences between shapes are not represented at all. In particular, the length of arc associated with any given interval atom is information not encoded by a description, and therefore "lost". Note, however, that it is not strictly correct to claim that all components have non-numeric, qualitative values. The qualitative value "0", rather than representing a range of values (as $+$ and $-$ do), represents a single value. The presence of 0 and U in the quantity space used for derivatives of tangent bearing accounts for the concept of "absolute" qualitative precision that we encountered in Section 4.2.1. The availability of 0 also accounts for the fact that a circle is the only shape precisely described by an atomic description ($\bigcirc\ \underline{\mathsf{P}^0}$).

9.2.3 Computability

One question that naturally arises concerns the computability of our atomic descriptions. Where do atomic descriptions come from? In the next chapter, we will consider the applicability of the atomic tokens as shape descriptors. One likely candidate for an application of QBT is high-level computer vision: the recognition of objects based on the boundaries of their silhouettes. If we consider raw image data to be the source of our atomic descriptions, then it would seem that only the initial levels of the hierarchy contain atoms that may be reliably extracted. Atoms with more than three components, because of noise and the amount of data available, cannot be feasibly identified. As well as the number of derivatives being an issue, identifying kink points is not straightforward, and distinguishing between angles and cusps is even more problematic. Although there are significant problems associated with the extraction of atomic descriptions from image data, they are

not of primary concern to us here. Importantly, we can easily *generate* atomic descriptions at *any* level of the hierarchy, since the rules for string syntax are provided by the atomic TOGS. In other words, we do not need image data to process descriptions that contain atoms with greater than three components.

9.3 Support for higher-level curve description

Atomic tokens are not, in themselves, adequate for representing all kinds of local curve features. We demonstrated this at the beginning of Chapter 5 by considering a couple of example curve features and showing that single atoms were not capable of representing them (Section 5.1). By "higher-level" curve descriptions we mean strings of *non-atomic* tokens. The purpose of representing higher-level (more abstract) curve features is to have more meaningful descriptions that make important qualitative aspects of a shape explicit. Support for higher-level descriptions is provided in Chapter 5 in the form of a model, based on the notions of identity and context, that allows curve features of greater abstraction to be specified as *complex tokens*. Further support is then provided in Chapter 6, in the form of the definition of an LFS (Section 6.3). An LFS is a constrained set of complex tokens that supports the description of shapes by strings of tokens, ensuring that no two represented features on the bounding curve of a shape overlap.

9.3.1 Model for abstract curve features

Through examples, it was established that there are two important aspects of an abstract curve feature that need to be modelled in order to represent it. The first aspect is the identities of the feature and is given by the curve segments and/or points that the feature corresponds to. An identity is represented by a string of atoms. The second important aspect of an abstract curve feature is the contexts in which the identities of the feature can exist. A context is split into two parts, a leading part and a trailing part. Each part is also represented by a string of atoms. An abstract curve feature is modelled by a complex token,

whose specification consists of a (possibly infinite) set of triples, each of which includes an identity string and related leading and trailing context strings. A complex token is identified by a label that is used to symbolise its presence in a description.

The model proposed has been used successfully, in the previous chapter, to specify the primitives of existing boundary-based schemes from the literature. In fact, we have not encountered a localised curve feature in a qualitative boundary-based scheme that cannot be specified using identities and contexts. In specifying primitives as complex tokens, we can define their essential characteristics in a formal manner, leading to a greater understanding of the precise meaning of a primitive. In the case of the contour codons, we were able to highlight an ambiguity regarding the status of bounding minima (Section 8.1.1). Such an ambiguity would need to be taken into account by a shape system containing knowledge of the contour codons.

Feature space

A complex token represents a localised curve feature. The set of all complex tokens, therefore, defines the space of curve features that can be represented by QBT. Within this space, only a small number of curve features have been specified in this thesis. Some features have simple finite specifications, e.g., curvature extrema, whilst others require specifications that contain an infinite number of triples, such as the curve-segment curvature types of QOT. A consequence of the lack of constraints imposed on the definition of a complex token is that "meaningless" curve features can be specified just as easily as useful ones. Although we have considered the classification of complex tokens and made reference to the notion of curve-feature hierarchies (see Section 5.3), we have not provided any systematic method of specifying curve features that are necessarily "non-meaningless".[2] A closer examination of feature space may uncover such a method or, perhaps, yield additional constraints that there is good reason to impose on the definition of a complex token.

[2]One difficulty, of course, is that one person's meaningless feature may be another's meaning*ful* feature.

9.3.2 Token-string description

In Chapter 5 we covered the process of *token fitting*, by which triples from the specifications of complex tokens are associated with substrings of an atomic description. A *ring diagram* shows the result of fitting a set of complex tokens to a particular atomic description. The examples of token fitting we gave in Section 5.4 highlighted the difficulties that can arise in mapping a ring diagram to an unambiguous token-string description. Recall that a token-string description is a string of labels representing curve features; no additional symbols are allowed in a string to encode the relative placement of the features (see Section 5.4.3). The existing boundary-based schemes we looked at in Chapters 2 and 3 all represent a shape by token-string descriptions. In Leyton's scheme, for example, there is a symbol attributed to each of the four kinds of curvature extrema, together with a symbol representing points of zero curvature. A shape is described by a string consisting only of these symbols. The order of the symbols in a string correspond to the order of the extrema and points of zero curvature round the boundary of the shape described by the string.

In Chapter 6, we focussed our attention on *sets* of complex tokens and the problem of ensuring that a set of complex tokens, when fitted to an atomic description, necessarily yields a ring diagram that unambiguously maps to a string of complex tokens. Our analysis of the positional configurations of fitted triples concluded that non-redundant fitted triples should be allowed to merge, meet, or be disjoint. From the analysis, constraints on individual complex tokens and pairs of complex tokens were defined, leading to the definition of an LFS (Section 6.3). In the previous chapter, we showed that each of the existing qualitative boundary-based schemes is an LFS.

When two non-redundant fitted triples of an LFS merge together, their identities overlap by a single atom, i.e., a single point or interval atom is *shared* by both identities. The two possibilities are shown by the curve segments in Figure 9.2. On the left of the figure, two of Hoffman and Richards' contour codons, 0^- and 1^-, as specified in CC_i (see Section 8.1.1), are shown fitted to a curve segment. The dots identify points of minimum curvature. Recall that, in CC_i, the minima that bound each codon are *included* in the identity of each

token specification, i.e., $0^- = \{\ [\,N^0\ \underline{N^+}\ N^0\ \underline{N^-}\ N^0\,]\ \}$ and $1^- = \{\ [\,N^0\ \underline{N^+}\ Z^+\ \underline{P^+}\ P^0\ \underline{P^-}\ P^0\,]\ \}$. We see that, for the segment of curve shown, 0^- merges with 1^-, such that the point atom N^0 is shared by both curve features. We can think of the features, then, as being connected at point x on the curve. The segment of curve on the right of Figure 9.2 gives an example of a merging involving *interval* atoms. The two tokens of $POLY_3$, $>$ and $<$, are shown fitted to a polygonal curve segment. Recall that $>$ is specified as $\{\ [\,\underline{Z}U_{>}\underline{Z}\,]\ \}$ and $<$ as $\{\ [\,\underline{Z}U_{<}\underline{Z}\,]\ \}$. We see that $>$ merges with $<$, such that a single straight line (labelled y) separates the vertices of $>$ and $<$. On this occasion, because the atom being shared is an interval, there is no single point connecting the two curve features, rather, both features share a single interval, i.e., we do not have a definite point of separation.

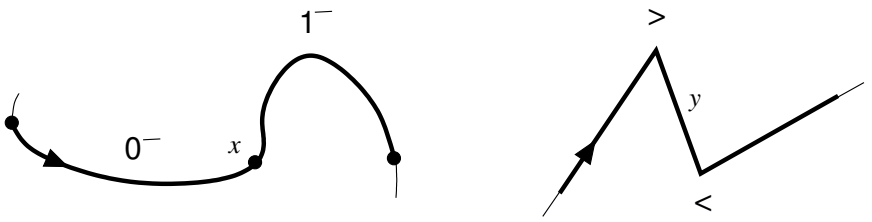


Figure 9.2: The sharing of identity atoms: points versus intervals

We have chosen, in this thesis, to focus our attention on the representation of shape using token-strings. We have not looked at other, more sophisticated descriptive techniques, where a shape is represented using some structure that is richer than a simple linear sequence of atomic or complex tokens. We will return to the issue of a richer representational structure in Section 10.3 of the next chapter, as part of our consideration of extensions to QBT. Currently, there is no provision for incorporating additional information of a quantitative nature into the description of a shape. This can be considered a limitation as it restricts the applicability of QBT. A capability for including quantitative information may allow the specification of the kinds of domain-specific vocabularies of high-level shape descriptors advocated by Saund (1992). Rather than representing a shape (of some object) using a general set of building-block primitives designed to correspond to part structure

(such as the contour codons or Biederman's geons), Saund takes the view that shape descriptors should be "tuned to the structural regularities and constraints of specific shape domains" (p. 75). Saund's high-level shape descriptors represent constellations of "shape tokens"[3]. A shape is described by a *Scale-Space Blackboard* holding shape tokens localised in position, orientation, and scale.

9.4 Token-ordering graphs

9.4.1 Expressive power

A set of tokens provides an alphabet of symbols for representing shapes using string descriptions. We identified two kinds of constraints associated with a set of tokens: *ordering constraints* and *closure constraints* (Section 3.2.4). The ordering constraints define the basic syntax for token-string descriptions. Any description that adheres to the ordering constraints is realisable as a section of curve (an open curve or part of an open curve[4]). If a description adheres to the closure constraints as well, then it is also realisable as a closed curve. In QOT, for example, the alphabet is $\{\supset, \subset, /, >, <, \succ, \prec\}$, and the ordering and closure constraints, given in Section 3.2.4, tell us that the description $\succ / \supset \subset$ is realisable as a section of curve *and* a closed curve.

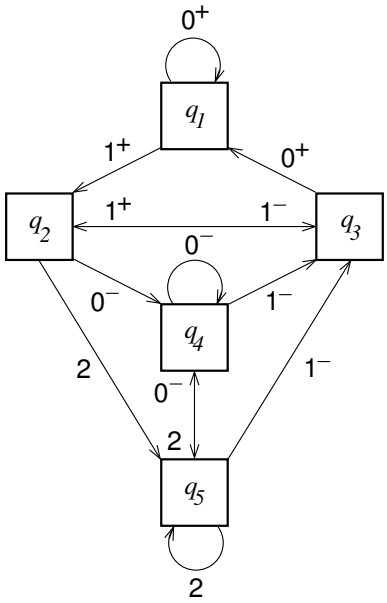
A token-ordering graph (TOG) visually encodes the ordering constraints that exist for a particular set of tokens. A TOG may be interpreted as a grammar that generates syntactically valid token-string descriptions. In Section 8.3.2, we drew a comparison between the TOG constructed for QOT and the full QOT grammar given in Appendix A, noting that the essential differences between the two are accounted for by the closure constraints not being encoded by the TOG. Specifically, the TOG constructed for QOT, when interpreted as a grammar, generates strings that are not generated by the grammar in the appendix,

[3] A shape token "marks the occurrence in [an] image of a shape fragment (e.g. corner, edge, blob), or some configuration of such fragments" (Saund 1992, p. 74).

[4] In Section 4.3.4, we stipulated that open curves begin and end with interval atoms, so a description that is syntactically valid may not be realisable as a *whole* open curve.

because a TOG is unable to encode any closure constraints that may be required by a set of tokens.

A TOG, consisting as it does of a finite number of nodes and directed edges, is essentially equivalent to a finite automaton (FA). The two graphs constructed for the contour codons with respect to restricted scope, for example (Figure 8.5, p. 170), are both equivalent to the finite-state machine shown in Figure 9.3. Nodes map to states and directed edges to state transitions. A node in a TOG associated with token T maps to a state, q, in the machine, such that every state transition *to* q is labelled T. All of the states in the machine are accepting states, apart from q_0 (not shown in Figure 9.3), which is designated the start state. Transitions exist in the machine from q_0 to each of the other states. Note that information pertaining to the strength of node connections is lost in the conversion of a TOG to its equivalent FA. This explains why the TOGs constructed for CC_e and CC_i are equivalent to the same machine. An alternative finite-state machine for Hoffman and Richards' contour codons, which computes codon descriptions from sequences of curvature extrema, is given by Richards et al. (1988, Figure 11).



	0^+	1^+	1^-	0^-	2
q_1	q_1	q_2	—	—	—
q_2	—	—	q_3	q_4	q_5
q_3	q_1	q_2	—	—	—
q_4	—	—	q_3	q_4	q_5
q_5	—	—	q_3	q_4	q_5

Figure 9.3: The finite-state machine equivalent to TOG(CC_e, G_s) and TOG(CC_i, G_s)

Just as, for a given regular language, there will be, in general, more than one FA that accepts it, so there will be, in general, more than one TOG capable of encoding the ordering constraints for a set of tokens. The construction algorithm for *atomic* TOGs, given in Appendix B, generates graphs that are both deterministic[5] and minimal (in terms of the number of nodes *and* the number of connections). The algorithm we provide in Chapter 7, for constructing *non-atomic* TOGs, does *not* ensure minimality (in terms of nodes *or* connections), although it does result in a graph that is deterministic.

9.4.2 Construction

The construction of TOGs for sets of atomic tokens is relatively straightforward and is covered in Appendix B. In Chapter 4, we derived the TOGs for levels 2 and 3 of the atomic hierarchy. Constructing non-atomic TOGs for sets of complex tokens is much less straightforward, because a complex token is specified by a (possibly infinite) set of string triples. In Chapter 7, we presented a solution to the construction problem for sets of complex tokens defining *restricted* LFSs. A restricted LFS contains only complex tokens whose triples have leading and trailing contexts of at most one atom in length. Our solution specifies three stages of construction: LFS preparation, node determination, and connection determination. In dealing only with restricted LFSs, we are able to adopt a strategy whereby the nodes in a graph are determined by considering each complex token in isolation, which simplifies the construction process. A side-effect of the strategy is that the set of graph nodes is not guaranteed to be minimal. Although our solution to the construction of a non-atomic TOG only admits complex tokens whose contexts are significantly restricted, we have still been able to construct TOGs for every LFS of interest, including, of course, the TOGs for the existing boundary-based schemes. This provides additional evidence, perhaps, that our definition of a complex token is under-constrained.

In a TOG, a node N is connected to a node N' if the curve feature associated with N' may follow the curve feature associated with N, without a curve feature associated with

[5]The meaning of "deterministic" in this context is explained in Section B.1.

any other node necessarily occurring in between. The construction procedure of Chapter 7 introduces the notion of connection strength, resulting in three connection types: strong, weak, and remote. A strong connection exists if feature identities may merge or meet, a weak connection exists if feature identities may be separated by a *single* atom, and a remote connection exists if feature identities may be separated by *more than* one atom. The connection types are not mutually exclusive, i.e., more than one type of connection may hold between two given nodes. Note that, in an atomic TOG, all of the connections are drawn as solid lines and are thus *strong* connections and consistent with the notion of connection strength. By utilising different kinds of connections, a non-atomic TOG is capable of visually reflecting the descriptive power of a set of complex tokens with respect to a particular scope of curves. This property was found to be useful in Chapter 8, as part of our analysis of the relative discriminatory power of existing boundary-based schemes. We were able to highlight, for example, a subtle deficiency in the set of QOT complex tokens: we expected every connection in the TOG constructed for QOT to be of the strong variety, but the graph actually contains six *weak* connections. The reason why weak connections exist in the TOG is that certain occurrences of the atoms Z and $\mathsf{U_c}$ are not accounted for by any of the curvature types of QOT.

Chapter 10

Conclusions and further work

In this final chapter we reflect on the work as a whole and then present the main contributions made by it. We finish by highlighting some potential paths for future research.

10.1 Reflections on the work as a whole

This thesis has been concerned with the qualitative representation of shape. In Chapter 1, we discussed the difficulties associated with the general representation of shape, concluding that the development of a single all-encompassing representational scheme is not feasible. We chose to restrict our attention to the qualitative representation of two-dimensional shape, in the context of AI. There are two basic approaches to the qualitative representation of two-dimensional shape: region-based description and boundary-based description. The central claim of the thesis was first stated in Section 1.2, and is as follows:

> *An analysis of qualitative curvature variation, based on tangent bearing, provides an adequate theoretical underpinning for the boundary-based approach to the qualitative representation of two-dimensional shape, leading to a theory which (i) provides a unifying account of, and (ii) generalises, existing qualitative boundary-based schemes.*

In Chapters 2 and 3 we looked at a number of existing region-based and boundary-based schemes, to indicate the essential differences between the two approaches, and to

show that existing boundary-based schemes differ with respect to rationale and the set of primitives provided. Hoffman and Richards' contour codons, for example, were designed for a theory regarding the perception of figure-ground reversal, Leyton's primitives result from the association of process activity with curvature extrema, and Galton and Meathrel's curvature types were designed as a minimal set of high-level descriptors for characterising shape. After reviewing existing schemes, we developed our general theory of the boundary-based approach to the qualitative representation of two-dimensional shape: *qualitative boundary theory* (QBT). The theory was developed in a stepwise fashion in Chapters 4 to 7. First, we derived an unbounded hierarchy of basic shape descriptors from an analysis of qualitative curvature variation, and introduced token-ordering graphs (TOGs) and an algorithm for their construction for sets of basic descriptors (Chapter 4); next, we formulated a model for complex tokens capable of specifying abstract localised curve features (Chapter 5); in Chapter 6 we identified the constraints on a set of complex tokens to ensure that token-string description is not compromised, which led to the definition of a local-feature scheme (LFS); finally, in Chapter 7 we provided an algorithm for constructing TOGs for a restricted class of LFSs. As detailed in Section 9.1, QBT consists of three parts: the basic shape descriptors, support for higher-level curve description, and a theory of TOGs.

Support for thesis claim

QBT is shown to provide a unifying account of existing boundary-based schemes by the analyses provided in Chapter 8, where each of the boundary-based schemes of Chapters 2 and 3 is specified as an LFS and its primitives, therefore, as complex tokens. The account given is a *unifying* one in the sense that each of the existing schemes is shown to be ultimately specifiable in terms of atomic tokens.

It is clearly the case that QBT also generalises existing qualitative boundary-based schemes. The unbounded hierarchy of atomic tokens, together with the model provided for representing more complex curve features, essentially define a *space* of representable

primitives. This space, in turn, defines a space of LFSs, in which are located the boundary-based schemes whose primitives are the contour codons, extremum primitives, and curvature types. QBT, then, is a *general* theory that supports the specification of application-specific boundary-based schemes.

We conclude that the thesis claim is strongly supported by the general theory developed in Chapters 4 to 7 and the analysis of existing boundary-based schemes carried out in Chapter 8.

Limitations of QBT

In the previous chapter, we highlighted the strengths and weaknesses of each of the three parts of QBT. The main limitations of QBT relate to discriminatory power and scope. Atomic tokens represent *localised* curve features and are associated with sequences of *non-numeric* components (see Section 9.2.2). In addition, there is no provision for the inclusion of quantitative information. This has an effect on the kinds of applications that boundary-based schemes derived from QBT can be used for. The scope of curves that can be adequately represented by QBT also affects the applicability of the derivable boundary-based schemes. Curves that contain "touching" or self-intersection points are not accounted for by QBT (see Section 9.2.1), therefore certain applications, such as handwriting recognition and diagrammatic reasoning, which are likely to require such points to be represented, are not very well supported by the theory. We pick up on these limitations in the section on further work at the end of the chapter.

Despite these limitations, we have demonstrated that, within its own terms, QBT is a powerful and expressive theory.

The framework provided by QBT, in which qualitative boundary-based schemes can be formally specified, contrasts with that proposed by Clementini & Felice (1997), in the domain of geographical information systems (GIS). The emphasis of Clementini & Felice's framework for qualitative shape description is on *global* properties of shape, whereas our emphasis has been on *localised* shape properties.

10.2 Main contributions

The work presented in this thesis makes the following contributions:

- *Qualitative analysis of boundary-based differential curve structure*

  We derived our hierarchy of atomic shape descriptors by using qualitative values for tangent bearing and its successive derivatives. A consideration of "qualitative calculus" based on these values yielded a small number of simple rules, by which sets of atomic tokens containing the same number of qualitative components are specifiable. This leads directly to the unbounded hierarchy, each level of which provides a complete set of atomic tokens with a certain level-specifc degree of discriminatory power. A curve has a description at each level of the hierarchy. Higher-numbered levels, those that incorporate a greater number of qualitative components, provide greater discriminatory power than lower-numbered levels. The hierarchy provides new boundary-based building blocks that are not tied to any particular domain of application and which may be used, as has been shown in this thesis, to specify the primitives of representational schemes for shape that *are* application-specific.

- *A model for specifying abstract localised curve features*

  It was recognised, in Chapter 5, that many interesting curve features cannot be represented by single atomic tokens. For this reason, we developed a model of complex tokens capable of specifying non-atomic localised curve features. The model is based on the notions of identity and context – two properties of a curve-feature instance that were shown to be necessary components of any adequate model of higher-level features. The model provided, and its notation, is intuitive, which is reflected by the succinct nature of the specifications for the majority of features that we have considered, e.g., a positive maximum of curvature is simply $\{\,\underline{\mathsf{P}^{+}}\,[\,\mathsf{P}^{0}\,]\,\underline{\mathsf{P}^{-}}\,\}$. Even the small number of specifications we encountered that contained infinitely many triples could be simply written, using appropriate regular expressions.

- *Unifying account of existing boundary-based schemes*

  In Chapter 8, we analysed the boundary-based schemes from Chapters 2 and 3. We showed how the primitives of each scheme can be specified as complex tokens, and that each scheme is an LFS. Our unifying account yielded a number of results. In specifying the contour codons as complex tokens, we found that we needed to address an ambiguity regarding the inclusion or otherwise of bounding minima within the identity of each codon. We constructed TOGs for each scheme, which showed that the scope of curves adequately represented by the contour codons is the same as that represented by the extremum primitives. The TOG constructed for QOT revealed that certain points on a curve are not accounted for by the set of qualitative curvature types. The same graph, when compared with the TOGs constructed for the contour codons and extremum primitives, with respect to *full scope*, showed that QOT is capable of adequately representing a much wider range of shapes than either of the other two schemes, although for certain curves the other schemes can make discriminations that cannot be made by QOT. The common tradeoff between scope and detail, therefore, is highlighted by our analysis.

- *The specification of graphs that encode token-ordering constraints*

  A set of atomic or complex tokens provides an explicit alphabet over which an implicit language of shape descriptions exists. Such a language is subject to ordering and closure constraints. We have developed a graph notation for visually encoding the ordering constraints for a set of tokens, where nodes represent tokens and directed edges the ordering constraints between them. In Appendix B we provided a procedure for constructing graphs for sets of atomic tokens, and in Chapter 7 a procedure for constructing graphs for sets of complex tokens. Atomic TOGs are important for the specification of complex tokens, because they are used to verify that triples contain strings of atomic tokens that are syntactically valid. In our analysis of existing schemes, we demonstrated that non-atomic TOGs are useful in comparing the discriminatory power of different schemes, with respect to specific scopes.

10.3 Directions for further work

We finish the chapter by highlighting some directions for further work based on QBT.

Further analysis of boundary-based curve features

The set of triples that may constitute the specification of a complex token is not constrained to any great extent (we have just a single condition to ensure that there are no *redundant* triples in the set). Our main objective was to provide a model general enough to enable all of the localised curve features we have encountered to be specified. This was achieved, with the side-effect that almost *all* legal specifications represent curve features in only a contrived technical sense, i.e., most "curve features" don't resemble meaningful features at all. The structure of *feature space* was only briefly touched upon. In Section 5.3.3, we showed how to determine subtype relationships between curve features represented by complex tokens, using set-theoretic operations. For example, a positive maximum of curvature is a *kind of* curvature stationary point, because the single triple specifying the former is a member of the set of triples specifying the latter.

A more detailed analysis of boundary-based curve features may yield additional constraints on the definition of a complex token, and a greater understanding of the structure of feature space. In turn, it may be possible for useful curve features to be specified *systematically*. One example of this would be a query-based system, in which a user enters criteria that characterise a class of curve features, resulting in the system automatically generating the appropriate specifications. In order to retrieve Hoffman and Richards' contour codons, for example, we would request all curve segments that are bounded by curvature minima but which do not *contain* any curvature minima.[1]

[1]Note that we are inadvertently implying here that the "correct" specification of the contour codons is given by CC_e (of Section 8.1.1).

Increasing the scope of shapes that can be represented

Although the scope of QBT includes a wide range of shapes (in the form of open and closed curves of a reasonably general type) we pointed out in Section 9.2.1 that certain curve points are not well handled by the theory. In particular, touching and crossing points are not represented in any way. Applications that require such points to be represented are currently poorly supported by QBT, and any extension that includes an adequate representation of these points is therefore desirable.

A natural question to pose, given a general theory for describing two-dimensional shape, is whether or not it can be used, in some (possibly modified) form, for representing *three*-dimensional shape. The nature of QBT is such that an adequate transition to three dimensions does not seem possible. The theory relies on the fact that a boundary of a two-dimensional shape is one-dimensional and can therefore be described using a linear string of symbols. In three dimensions this is not possible. There is no obvious analogue of an atomic curve feature in three dimensions[2], so it is not easy to see how the theory can be applied to three-dimensional shape.

Increasing discriminatory power

There are a number of ways to increase the discriminatory power of QBT. By improving QBT's capability for distinguishing shapes, a wider range of applications would be supported by the theory. An indication of the possibilities available for increasing discriminatory power is given by the following extensions:

- *Inclusion of additional qualitative information*

  More information of a qualitative kind could be associated with atomic tokens. Some of the possibilities for this were described in the context of QOT, in Section 3.4. The set of kink tokens could be enlarged to encompass the distinction between acute, right-angled, and obtuse angles, for example. The relative length, orientation, and

[2] Although a selection of three-dimensional QOT primitives *have* been suggested by Galton (2000).

position of curve features represented by atomic tokens could also be incorporated in a qualitative fashion. One possibility would be to encode the additional information using predicates such as longer-than, same-length, opposite, collinear, etc.

- *Inclusion of quantitative information*

  Descriptions could be augmented with more precise information of a numerical kind. This can be done at different levels, e.g., ratios could be used (thereby preserving the property of invariance with respect to uniform scaling), feature-specific quantitative shape measures could be included, and exact values could be specified, for angle size, segment length, etc. Applications concerned with the *generation* of shapes, in particular, are likely to require the inclusion of quantitative information.

- *Integration with the region-based approach*

  As suggested by Richards & Hoffman (1987, footnote 1), it may be desirable to represent a shape with a description that consists of boundary-based information, together with region-based information associated with one of the axis-based analyses. The region-based part of the description, with its radius function and elementary descriptors, such as "worm" and "flare", would serve as the quantitative model of the shape, and the boundary-based part as the related qualitative model. The exact way in which the two parts would be related, however, is unclear. Richards & Hoffman hypothesise a mapping via "a suitable list of indexed parameters".

Note that most of the extensions listed here affect the kind of structures used for shape descriptions, i.e., simple token-string descriptions are not likely to suffice.

Determination of closure constraints

In Chapter 3, we provided the simple closure constraint for descriptions consisting of the curvature-type primitives of QOT, although the correctness of the constraint has yet to be formally proven. Knowledge of the closure constraints for a set of tokens enables us to verify that a given description is instantiable as a closed curve. We have not addressed

the problem of deriving the closure constraints for a set of atomic or complex tokens. The similarity between the set of QOT primitives and the atomic tokens at level 2 of the atomic hierarchy suggests that the closure constraints for the level-2 atoms may be as easily specified as the closure constraints for QOT. However, it is not clear what the closure constraints are for any of the other levels in the hierarchy, or what general rules are required to derive the necessary constraints for a given level. An interesting observation is that the closure constraints for QOT are such that they can be built into a grammar for generating outlines that is *regular*. It is not clear whether the closure constraints for each level in the hierarchy can also be encoded in a regular grammar or whether, instead, a non-regular grammar is required in the general case.

Incorporating QBT into a larger system

We have seen how QBT can be used to analyse existing boundary-based schemes and to formally specify abstract localised curve features. A logical next step is to consider building a spatial reasoning or high-level vision system which includes some or all of the theory of boundary-based representation of shape developed in this thesis.

Our development of QBT did not include a discussion of the multi-scale representation of shapes using atomic descriptions. This is an obvious next step to consider, especially if our interest is in the building of a high-level vision system. A multi-scale representation, similar to that provided by Rosin (1993) for contour codons, would allow the atomic features representing noise or the unimportant finer details of a shape to be distinguished from the more significant features that characterise a shape (and therefore disregarded if necessary).

Re-application of developed techniques

Finally, QBT is founded on a particular discretisation of tangent bearing and its successive derivatives. Essentially, we have been dealing with the representation of a single real-valued variable over time, where the variable is tangent bearing and the arc length of a

curve is analogous to time. Therefore, QBT may actually be re-interpreted as a theory of the qualitative representation of a variable as it changes over time. It is possible, then, that the techniques and formalisms developed in this thesis can be usefully *re-applied* in an area other than the representation of two-dimensional shape. One such area of application, for example, might be the description of the "profiles" of different processes ("temporal shape").

Appendix A

QOT material

A regular grammar is derived that generates valid strings for the full set of outlines, including all of the canonical ones. We explain why a grammar that generates all and only *the canonical strings cannot be regular. This appendix also contains an enumeration of the valid curvature-type subsets and a procedure for constructing subset-specific grammars.*

A.1 A grammar for the full set of outlines

Here we provide a regular grammar that generates all of the canonical string descriptions and also other valid descriptions that are not canonical. In the next section we explain why a grammar that generates all and only the canonical descriptions cannot be regular. As a first step to constructing the grammar, we observe that the full set of outlines can be partitioned into three subsets:

1. outlines containing at least one $\supset$,
2. outlines that do not contain a $\supset$, but contain at least one $\subset$, and
3. outlines that do not contain any instances of $\supset$ or $\subset$, i.e., polygonal outlines.

The grammar is constructed in three parts. Each part generates the canonical strings (and certain other non-canonical strings) for one of the three subsets. The complete set of

grammar rules is given in Table A.1 (where "Λ" denotes the empty string).

Firstly, consider the valid strings that describe the outlines of subset 1. Every canonical string must begin with $\supset$ and, apart from the description consisting of a single $\supset$, end with a symbol distinct from $\supset$. This constraint is enforced by the organisation of the grammar rules. A string is generated by successively adding symbols according to the ordering constraints given in Section 3.2.4. Note that, because each string begins with a convex curve segment, the *closure* constraint is automatically satisfied by the initial symbol in the string. The first block of rules in the grammar (those of the form $A_\alpha \rightarrow \beta$ and $W \rightarrow \beta$), together with the first two starting rules ($S \rightarrow \supset$ and $S \rightarrow \supset A_\supset$), generate all of the valid strings beginning with $\supset$.

Next, consider the valid strings that describe the outlines of subset 2. Every canonical string must begin with $\subset$ and contain at least three convex points. As before, the ordering constraints are enforced by the organisation of the grammar rules. However, this time we must ensure closure by "remembering" when a convex point is added to the string, and not allowing a string to be terminated until it contains three such points. This is achieved by having "cascading" blocks of rules; where each block ensures that a convex point of some kind is added and then "passes control" to the next block. The second block of rules in the grammar ($B_\alpha \rightarrow \beta$ through to $E_\alpha \rightarrow \beta$), together with the starting rule $S \rightarrow \subset B_\subset$, generate all of the valid strings that begin with $\subset$ and which do not contain any convex curve segments.

Lastly, we need to consider the valid strings that describe the polygonal outlines of subset 3. Once again, to ensure that each string contains at least three outward-pointing angles, we require a cascading mechanism. The last block of rules in the grammar, together with the starting rule $S \rightarrow /F_/$, generate the valid strings representing the polygonal outline-types.

A.1.1 Generating only the canonical strings

Here we show that a grammar that generates *all and only* the canonical strings for the full set of outlines must necessarily be non-regular. Consider the set of polygonal outlines that contain exactly two inward-pointing vertices that are not adjacent. Let $\mathcal{S}$ be the set of canonical strings describing such outlines. Let A and B denote the subsequences $/>$ and $/<$, respectively. Each string in $\mathcal{S}$ is of the form A^xBA^yB, where $x + y \geq 3$ (for closure) and, to ensure that the string is canonical, $x \geq y$. In other words, when generating a string in $\mathcal{S}$ we must ensure that the longest sequence of outward-pointing vertices is encoded at the *beginning* of the string. An application of the pumping lemma shows that a regular grammar that generates all strings of the form A^xBA^yB, that adhere to the constraints $x + y \geq 3$ and $x \geq y$, also generates at least one string of the same form, but where x is *less than* y. Such strings are not canonical. We conclude, therefore, that a grammar capable of generating all and only the canonical strings cannot be regular.

$$
\begin{array}{lcl}
S & \rightarrow & \supset \mid \supset A_{\supset} \mid \subset B_{\subset} \mid /F_{/} \\
\\
A_{\supset} & \rightarrow & /A_{/} \mid \subset A_{\subset} \mid > W \mid < W \mid \succ A_{\succ} \mid \prec W \\
A_{\subset} & \rightarrow & \supset A_{\supset} \mid /A_{/} \mid > W \mid < W \mid \succ W \mid \prec A_{\prec} \mid \Lambda \\
A_{/} & \rightarrow & \supset A_{\supset} \mid \subset A_{\subset} \mid > W \mid < W \mid \succ A_{\succ} \mid \prec A_{\prec} \mid \Lambda \\
A_{\succ} & \rightarrow & \subset A_{\subset} \\
A_{\prec} & \rightarrow & \supset A_{\supset} \\
W & \rightarrow & \supset A_{\supset} \mid /A_{/} \mid \subset A_{\subset} \mid \Lambda \\
\\
B_{\subset} & \rightarrow & /B_{/} \mid > X \mid < B_{<} \mid \succ X \\
B_{/} & \rightarrow & \subset B_{\subset} \mid > X \mid < B_{<} \mid \succ B_{\succ} \\
B_{<} & \rightarrow & /B_{/} \mid \subset B_{\subset} \\
B_{\succ} & \rightarrow & \subset C_{\subset} \\
X & \rightarrow & /C_{/} \mid \subset C_{\subset} \\
C_{\subset} & \rightarrow & /C_{/} \mid > Y \mid < C_{<} \mid \succ Y \\
C_{/} & \rightarrow & \subset C_{\subset} \mid > Y \mid < C_{<} \mid \succ C_{\succ} \\
C_{<} & \rightarrow & /C_{/} \mid \subset C_{\subset} \\
C_{\succ} & \rightarrow & \subset D_{\subset} \\
Y & \rightarrow & /D_{/} \mid \subset D_{\subset} \\
D_{\subset} & \rightarrow & /D_{/} \mid > Z \mid < D_{<} \mid \succ Z \\
D_{/} & \rightarrow & \subset D_{\subset} \mid > Z \mid < D_{<} \mid \succ D_{\succ} \\
D_{<} & \rightarrow & /D_{/} \mid \subset D_{\subset} \\
D_{\succ} & \rightarrow & \subset E_{\subset} \\
Z & \rightarrow & /E_{/} \mid \subset E_{\subset} \mid \Lambda \\
E_{\subset} & \rightarrow & /E_{/} \mid > Z \mid < Z \mid \succ Z \\
E_{/} & \rightarrow & \subset E_{\subset} \mid > Z \mid < Z \mid \succ E_{\succ} \mid \Lambda \\
E_{\succ} & \rightarrow & \subset E_{\subset} \\
\\
F_{/} & \rightarrow & > F_{>} \mid < F_{<} \\
F_{<} & \rightarrow & /F_{/} \\
F_{>} & \rightarrow & /G_{/} \\
G_{/} & \rightarrow & > G_{>} \mid < G_{<} \\
G_{<} & \rightarrow & /G_{/} \\
G_{>} & \rightarrow & /H_{/} \\
H_{/} & \rightarrow & > H_{>} \mid < H_{<} \\
H_{<} & \rightarrow & /H_{/} \\
H_{>} & \rightarrow & /I_{/} \mid \Lambda \\
I_{/} & \rightarrow & > H_{>} \mid < H_{>}
\end{array}
$$

Table A.1: A regular grammar for the full set of outlines

A.2 Valid curvature-type subsets

A subset of the seven qualitative curvature types is valid *iff*:

- the set contains at least one symbol taken from $\{\supset, \subset, /\}$,
- if $\supset$ is not present, then at least one of $\{>, \succ\}$ must be present,
- if the set contains $\succ$ then it must also contain $\subset$, and
- if the set contains $\prec$ then it must also contain $\supset$.

The 62 valid curvature-type subsets are listed in Table A.2.

A.2.1 Procedure for subgrammar construction

Given a valid curvature-type subset, S, a subgrammar can be derived from the full grammar that generates valid strings representing the outline types denoted by S. We construct a grammar for S by selecting the appropriate subset of the rules comprising the full grammar, as follows:

1. If $\supset \in S$ then select $S \to \supset$, $S \to \supset A_{\supset}$, and all rules of the form $\Psi_\alpha \to \beta\Phi$ and $\Psi_\alpha \to \Lambda$, where $\Psi \in \{A, W\}$ and $\alpha, \beta \in S$.
2. If $\subset \in S$ and $S \cap \{>, \succ\} \neq \emptyset$ then select $S \to \subset B_{\subset}$ and all rules of the form $\Psi_\alpha \to \beta\Phi$ and $\Psi_\alpha \to \Lambda$, where $\Psi \in \{B, C, D, E, X, Y, Z\}$ and $\alpha, \beta \in S$.
3. If $/, > \in S$ then select $S \to /F_{/}$ and all rules of the form $\Psi_\alpha \to \beta\Phi$ and $\Psi_\alpha \to \Lambda$, where $\Psi \in \{F, G, H, I\}$ and $\alpha, \beta \in S$.

1-element set	*4-element sets*	*5-element sets*
$\{\supset\}$	$\{\supset, \subset, /, >\}$	$\{\supset, \subset, /, >, <\}$
	$\{\supset, \subset, /, <\}$	$\{\supset, \subset, /, >, \succ\}$
2-element sets	$\{\supset, \subset, /, \succ\}$	$\{\supset, \subset, /, >, \prec\}$
$\{\supset, \subset\}$	$\{\supset, \subset, /, \prec\}$	$\{\supset, \subset, /, <, \succ\}$
$\{\supset, /\}$	$\{\supset, \subset, >, <\}$	$\{\supset, \subset, /, <, \prec\}$
$\{\supset, >\}$	$\{\supset, \subset, >, \succ\}$	$\{\supset, \subset, /, \succ, \prec\}$
$\{\supset, <\}$	$\{\supset, \subset, >, \prec\}$	$\{\supset, \subset, >, <, \succ\}$
$\{\supset, \prec\}$	$\{\supset, \subset, <, \succ\}$	$\{\supset, \subset, >, <, \prec\}$
$\{\subset, >\}$	$\{\supset, \subset, <, \prec\}$	$\{\supset, \subset, >, \succ, \prec\}$
$\{\subset, \succ\}$	$\{\supset, \subset, \succ, \prec\}$	$\{\supset, \subset, <, \succ, \prec\}$
$\{/, >\}$	$\{\supset, /, >, <\}$	$\{\supset, /, >, <, \prec\}$
	$\{\supset, /, >, \prec\}$	$\{\subset, /, >, <, \succ\}$
3-element sets	$\{\supset, /, <, \prec\}$	
$\{\supset, \subset, /\}$	$\{\supset, >, <, \prec\}$	*6-element sets*
$\{\supset, \subset, >\}$	$\{\subset, /, >, <\}$	$\{\supset, \subset, /, >, <, \succ\}$
$\{\supset, \subset, <\}$	$\{\subset, /, >, \succ\}$	$\{\supset, \subset, /, >, <, \prec\}$
$\{\supset, \subset, \succ\}$	$\{\subset, /, <, \succ\}$	$\{\supset, \subset, /, >, \succ, \prec\}$
$\{\supset, \subset, \prec\}$	$\{\subset, >, <, \succ\}$	$\{\supset, \subset, /, <, \succ, \prec\}$
$\{\supset, /, >\}$		$\{\supset, \subset, >, <, \succ, \prec\}$
$\{\supset, /, <\}$		
$\{\supset, /, \prec\}$		*7-element set*
$\{\supset, >, <\}$		$\{\supset, \subset, /, >, <, \succ, \prec\}$
$\{\supset, >, \prec\}$		
$\{\supset, <, \prec\}$		
$\{\subset, /, >\}$		
$\{\subset, /, \succ\}$		
$\{\subset, >, <\}$		
$\{\subset, >, \succ\}$		
$\{\subset, <, \succ\}$		
$\{/, >, <\}$		

Table A.2: An enumeration of the 62 valid curvature-type subsets

Appendix B

Constructing atomic TOGs

In this appendix, we first describe how an atomic TOG is constructed from a pair of I-I and I-P-I tables, and then we provide a construction algorithm.

B.1 Discussion

As an example of TOG construction, we will use the I-I and I-P-I tables for level 2 of the atomic hierarchy and consider how a corresponding graph may be specified from them. For convenience, the level-2 tables are reproduced in Table B.1.

We construct the TOG in two stages. In the first stage, using the I-I table, nodes are created for the interval atoms and connected together. In the second stage, using the I-P-I table, nodes are created for the point atoms and then connected to the appropriate interval nodes. Recall that a point atom can only be present in between two interval atoms. Consequently, there will be no edges in the graph that directly connect one point atom to another. Figure B.1 shows the result of the first stage using the I-I_2 table. One node is created for each of the three interval atoms and the nodes are connected together according to the four ticks present in the table. For the second stage, the simplest strategy is to create a new graph node for each occurrence of a point atom in a cell of the I-P-I table. Consulting I-P-I_2, then, we would need to create four nodes for Z, eight for $\mathsf{U_c}$, nine each for $\mathsf{U_<}$ and

$U_>$, and five each for $U_\prec$ and $U_\succ$. In total, the TOG would consist of 43 nodes (forty for the six point atoms and three for the three interval atoms). To indicate the nature of the resulting graph, Figure B.2(a) shows the connected interval nodes together with the eight nodes for U_c, and Figure B.2(b) shows the interval nodes along with the four nodes representing Z. In each case, the point nodes are connected to the interval nodes according to the occurrences of the point atoms in the I-P-I_2 table.

	P	Z	N
P		√	
Z	√		√
N		√	

	P	Z	N
P	Z U_c $U_<$ $U_>$ $U_\prec$	U_c $U_<$ $U_>$ $U_\prec$	Z U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$
Z	U_c $U_<$ $U_>$ $U_\prec$	$U_<$ $U_>$	U_c $U_<$ $U_>$ $U_\succ$
N	Z U_c $U_<$ $U_>$ $U_\prec$ $U_\succ$	U_c $U_<$ $U_>$ $U_\succ$	Z U_c $U_<$ $U_>$ $U_\succ$

Table B.1: I-I and I-P-I tables for the level-2 atoms and kink tokens

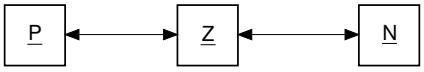


Figure B.1: Connected TOG nodes for the interval atoms of level 2

The graphs in Figure B.2 can be regarded as *non-deterministic* because, in each graph, more than one node representing the point atom is reachable from each of the interval nodes. In graph (b), for example, we can go from node P to either of two Z nodes, one of which takes us back to P and the other of which takes us to N. A non-deterministic TOG leads to inefficiency in the parsing of token strings. In the present case, consider the string P Z N. If we scan the string from left to right, then when we reach Z we need to either look

ahead to see *which* Z we have encountered, or we may need to backtrack if we choose the "wrong" one. A parse using graph (a) is potentially even more inefficient, since its P and N interval nodes are each connected to *three* different U_c nodes.

By inspecting the four Z nodes of graph (b) we can deduce that, in fact, only one node is required for the Z point atom. The reason for this is that an interval atom that immediately precedes Z does not constrain the choice of interval atoms that may immediately follow Z. If we call the interval that precedes a point the *leading context* of the point, and the interval that follows it the *trailing context*, then the reason Z requires only one node is that its leading and trailing contexts are independent.[1] In contrast, the eight nodes for U_c cannot be replaced by just one node, because the leading and trailing contexts of U_c are *not* independent. Specifically, if the leading context is Z, then the only possibilities for the trailing context are P and N (i.e., Z is not allowed), whereas, if the leading context is either of P or N, then the trailing context can be P, N, *or* Z. It is not possible to enforce these dependencies using a single node.

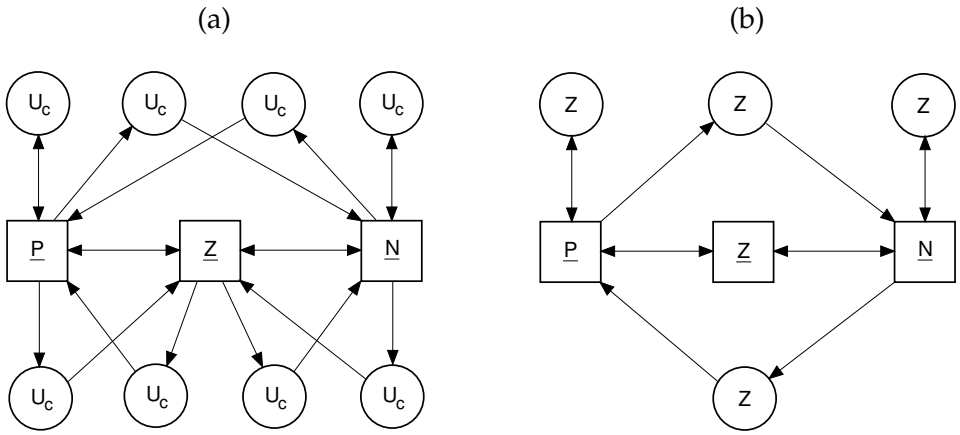


Figure B.2: Level-2 partial TOGs incorporating the U_c and Z point atoms

In order to determine how many nodes are required for a given point atom, and to ensure that the graph remains deterministic, we need to take a closer look at the leading

[1]Context independence also accounts for the fact that interval atoms only require one node in an atomic TOG.

and trailing contexts of the atom. Consider, first, the point atom Z. Each occurrence of Z in I-P-I_2 is specified by a *context pair*, $\langle l, t \rangle$, where l is a leading context of Z and t is a trailing context. For Z, we have the pairs $\langle \underline{P}, \underline{P} \rangle$, $\langle \underline{P}, \underline{N} \rangle$, $\langle \underline{N}, \underline{P} \rangle$, and $\langle \underline{N}, \underline{N} \rangle$. There are two distinct leading contexts ($\underline{P}$ and $\underline{N}$) and two distinct trailing contexts (also $\underline{P}$ and $\underline{N}$). There are four pairs in all and, importantly, each one accounts for a particular permutation of the possible leading and trailing contexts. As the number of context pairs equals the number of permutations, we deduce that the leading and trailing contexts of Z are indeed independent, and that, therefore, only one node is required for Z. Now let's consider U_c. We extract the following eight context pairs:

$$\langle \underline{P}, \underline{P} \rangle,\ \langle \underline{P}, \underline{Z} \rangle,\ \langle \underline{P}, \underline{N} \rangle,\ \langle \underline{Z}, \underline{P} \rangle,\ \langle \underline{Z}, \underline{N} \rangle,\ \langle \underline{N}, \underline{P} \rangle,\ \langle \underline{N}, \underline{Z} \rangle,\ \langle \underline{N}, \underline{N} \rangle$$

There are three distinct leading contexts ($\underline{P}$, $\underline{Z}$, and $\underline{N}$) and three distinct trailing contexts (also $\underline{P}$, $\underline{Z}$, and $\underline{N}$). This time, the number of pairs is *not* equal to the number of permutations of the distinct leading and trailing contexts ($8 \neq 3 \times 3$). The "missing" context pair is $\langle \underline{Z}, \underline{Z} \rangle$. We have established, therefore, that more than one node is required for U_c. Now we need to decide exactly how many nodes *are* required, and how each node is related to the set of context pairs. We can think of each node in the graph representing a point atom as being associated with a subset of the context pairs of the atom. For point atoms having a single node, the full set of context pairs is associated with the node. To ensure that the graph is deterministic, context pairs that have the same leading context must be associated with the *same* node. Our first step, therefore, is to partition the set of context pairs into a number of subsets, such that each subset contains all of the context pairs that share a particular leading context. For U_c, then, we get the three subsets $\{\langle \underline{P}, \underline{P} \rangle, \langle \underline{P}, \underline{Z} \rangle, \langle \underline{P}, \underline{N} \rangle\}$, $\{\langle \underline{Z}, \underline{P} \rangle, \langle \underline{Z}, \underline{N} \rangle\}$, and $\{\langle \underline{N}, \underline{P} \rangle, \langle \underline{N}, \underline{Z} \rangle, \langle \underline{N}, \underline{N} \rangle\}$. We could now create three nodes for U_c, one per subset. Inspection of the subsets, however, reveals that the first and third subsets can be combined, since they share the same trailing contexts. Therefore, only two nodes are required for U_c, one being associated with $\{\langle \underline{P}, \underline{P} \rangle, \langle \underline{P}, \underline{Z} \rangle, \langle \underline{P}, \underline{N} \rangle, \langle \underline{N}, \underline{P} \rangle, \langle \underline{N}, \underline{Z} \rangle, \langle \underline{N}, \underline{N} \rangle\}$ and

the other with $\{\langle \underline{Z}, \underline{P}\rangle, \langle \underline{Z}, \underline{N}\rangle\}$. The final result is shown in Figure B.3. In the next section we formalise the procedure we have described in this section, giving an algorithm that constructs a deterministic (and minimal[2]) atomic TOG from the I-I and I-P-I tables for a given level of the atomic hierarchy. The full atomic TOGs for levels 2 and 3 are given in Chapter 4, as Figures 4.4 and 4.5 respectively.

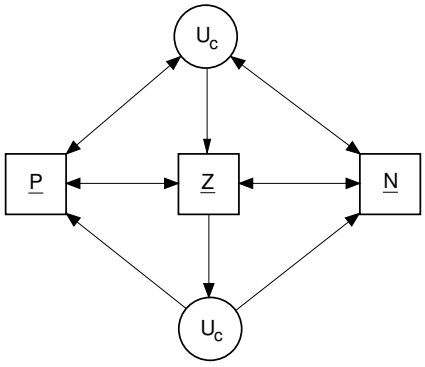


Figure B.3: Deterministic Level-2 partial TOG incorporating $\mathsf{U_c}$

B.2 Construction algorithm

The construction algorithm, which we will present shortly, makes use of an algorithm for partitioning a set of context pairs, according to the procedure described in the last section, i.e., grouping the pairs according to their leading contexts, and then combining those sets of pairs that share the same trailing contexts. Algorithm B.1 formalises the partitioning procedure. The functions *lcon* and *tcon* return, respectively, the leading and trailing elements of a context pair. To illustrate the steps of the algorithm, consider the kink token $\mathsf{U}_{\succ}$. Consulting I-P-I_2, we have the following set, $\mathcal{S}$, of context pairs for $\mathsf{U}_{\succ}$:

$$\langle \underline{P}, \underline{N}\rangle,\ \langle \underline{Z}, \underline{N}\rangle,\ \langle \underline{N}, \underline{P}\rangle,\ \langle \underline{N}, \underline{Z}\rangle,\ \langle \underline{N}, \underline{N}\rangle$$

Step **1** initialises the result set, $\mathcal{P}$. Step **2** creates a number of sets of trailing contexts,

[2]In terms of the number of nodes *and* the number of edges.

each of which is associated with a particular leading context. We get $\mathcal{T}_{\underline{\mathsf{P}}} = \{\underline{\mathsf{N}}\}$, $\mathcal{T}_{\underline{\mathsf{Z}}} = \{\underline{\mathsf{N}}\}$, and $\mathcal{T}_{\underline{\mathsf{N}}} = \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}, \underline{\mathsf{N}}\}$. In step **3**, sets of context pairs are added to the result set $\mathcal{P}$, as single elements (step 3.2; note the extra pair of curly brackets). Each $\mathcal{T}_x$ is considered in turn:

- $\mathcal{T}_{\underline{\mathsf{P}}}$: $y = \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}\}$, $\mathcal{P} = \{\, \{\langle \underline{\mathsf{P}}, \underline{\mathsf{N}}\rangle, \langle \underline{\mathsf{Z}}, \underline{\mathsf{N}}\rangle\}\,\}$
- $\mathcal{T}_{\underline{\mathsf{Z}}}$: $y = \{\underline{\mathsf{P}}, \underline{\mathsf{Z}}\}$, $\mathcal{P} = \{\, \{\langle \underline{\mathsf{P}}, \underline{\mathsf{N}}\rangle, \langle \underline{\mathsf{Z}}, \underline{\mathsf{N}}\rangle\}\,\}$
- $\mathcal{T}_{\underline{\mathsf{N}}}$: $y = \{\underline{\mathsf{N}}\}$, $\mathcal{P} = \{\, \{\langle \underline{\mathsf{P}}, \underline{\mathsf{N}}\rangle, \langle \underline{\mathsf{Z}}, \underline{\mathsf{N}}\rangle\}, \{\langle \underline{\mathsf{N}}, \underline{\mathsf{P}}\rangle, \langle \underline{\mathsf{N}}, \underline{\mathsf{Z}}\rangle, \langle \underline{\mathsf{N}}, \underline{\mathsf{N}}\rangle\}\,\}$

The final step of the algorithm returns $\mathcal{P} = \{\, \{\langle \underline{\mathsf{P}}, \underline{\mathsf{N}}\rangle, \langle \underline{\mathsf{Z}}, \underline{\mathsf{N}}\rangle\}, \{\langle \underline{\mathsf{N}}, \underline{\mathsf{P}}\rangle, \langle \underline{\mathsf{N}}, \underline{\mathsf{Z}}\rangle, \langle \underline{\mathsf{N}}, \underline{\mathsf{N}}\rangle\}\,\}$ for the set of context pairs, $\mathcal{S}$, of $\mathsf{U}_{\succ}$. Since $|\mathcal{P}| = 2$, we know that two nodes are required for $\mathsf{U}_{\succ}$ in the atomic TOG; one node is created per element of $\mathcal{P}$.

Input: $\mathcal{S}$ (set of context pairs)
Output: $\mathcal{P}$ (a partition of $\mathcal{S}$)

1 $\mathcal{P} \leftarrow \{\}$;
2 **for each** $s \in \mathcal{S}$ **do**
2.1 $\quad \mathcal{T}_{lcon(s)} \leftarrow \{\, tcon(x \in \mathcal{S}) \mid lcon(x) = lcon(s)\,\}$;
3 **for each** $\mathcal{T}_x$ **do**
3.1 $\quad y \leftarrow \{\, x\,\} \cup \{\, z \mid \mathcal{T}_z = \mathcal{T}_x\,\}$;
3.2 $\quad \mathcal{P} \leftarrow \mathcal{P} \cup \{\, \{\, s \in \mathcal{S} \mid lcon(s) \in y\,\}\,\}$;
4 **return** $\mathcal{P}$;

Algorithm B.1: Partitioning a set of context pairs

The full TOG construction procedure is formalised as Algorithm B.2. During steps **1** and **2**, nodes are created for each interval atom at level k (step 1.1) and connected together according to the contents of I-P-I$_k$ (step 2.1). A node created for atom x is denoted by $N(x)$. For $k = 2$, after step **2** of the algorithm the TOG is the graph shown in Figure B.1. The remainder of the algorithm is concerned with the creation of nodes for the point atoms and the connections involving such nodes (step **3**). Each point atom, p, in $\mathcal{P}_k$ is considered

in turn. First, in steps 3.1 through to 3.2.1, the set of context pairs for p, $\mathcal{S}$, is established, by considering the occurrences of p in the I-P-I$_k$ table. Next, in step 3.3, $\mathcal{S}$ is partitioned according to Algorithm B.1, with the result returned as set $\mathcal{X}$. New nodes are then created for each element, $\mathcal{C}$, of $\mathcal{X}$ (step 3.5.2) and each is connected to the appropriate existing interval nodes, according to the leading and trailing elements of the context pairs contained in $\mathcal{C}$ (steps 3.5.3.1 and 3.5.3.2).

Input: k (atomic level)
Output: An atomic TOG for level k of the atomic hierarchy

```
1       for each i ∈ I_k do
1.1         create N(i);
2       for each ⟨x, y⟩ ∈ I_k × I_k do
2.1         if I-I_k(x, y) = √ then connect N(x) to N(y);
3       for each p ∈ P_k do
3.1         S ← {};
3.2         for each ⟨x, y⟩ ∈ I_k × I_k do
3.2.1           if I-P-I_k(x, y) contains p then S ← S ∪ { ⟨x, y⟩ };
3.3         X ← partition(S);
3.4         j ← 0;
3.5         for each C ∈ X do
3.5.1           j ← j + 1;
3.5.2           create N_j(p);
3.5.3           for each c ∈ C do
3.5.3.1             connect N(lcon(c)) to N_j(p);
3.5.3.2             connect N_j(p) to N(tcon(c));
```

Algorithm B.2: Constructing an atomic TOG from a pair of I-I and I-P-I tables

Appendix C

Specifications for Rosin's codons

Rosin (1993) has developed an extensive set of contour codons for representing curves. The set includes Hoffman and Richards' original codons, together with a large number of additional codons that allow straight lines, cusps, and open curves to be represented. In this appendix, we provide complex-token specifications for each of Rosin's codons.

All of the codons in Rosin's extended set can be specified as complex tokens. The specifications are given in Tables C.1 to C.6, according to the order they are presented in (Rosin 1993). Rosin's set includes Hoffman and Richards' original set of codons (Table C.1), additional codons required to adjoin straight lines (Table C.2), codons that allow open curves to be represented (Tables C.3 and C.4), and codons that allow angles and cusps to be represented (Tables C.5 and C.6).

$$
\begin{aligned}
0^{+} &= \{\, P^{0}\,[\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,]\,P^{0}\,\} \\
0^{-} &= \{\, N^{0}\,[\,\underline{N^{+}}\,N^{0}\,\underline{N^{-}}\,]\,N^{0}\,\} \\
1^{+} &= \{\, P^{0}\,[\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,Z^{-}\,\underline{N^{-}}\,]\,N^{0}\,\} \\
1^{-} &= \{\, N^{0}\,[\,\underline{N^{+}}\,Z^{+}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,]\,P^{0}\,\} \\
2 &= \{\, N^{0}\,[\,\underline{N^{+}}\,Z^{+}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,Z^{-}\,\underline{N^{-}}\,]\,N^{0}\,\} \\
\infty &= \{\, [\,\underline{Z^{0}}\,]\,\}
\end{aligned}
$$

Table C.1: Hoffman and Richards' original set of codons

$$
\begin{aligned}
A^{+} &= \{\, P^{0}\,[\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,\underline{Z^{0}}\,]\,\} \\
A^{-} &= \{\, [\,\underline{Z^{0}}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,]\,P^{0}\,\} \\
B^{+} &= \{\, N^{0}\,[\,\underline{N^{+}}\,Z^{+}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,\underline{Z^{0}}\,]\,\} \\
B^{-} &= \{\, [\,\underline{Z^{0}}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,Z^{-}\,\underline{N^{-}}\,]\,N^{0}\,\} \\
C &= \{\, [\,\underline{Z^{0}}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,\underline{Z^{0}}\,]\,\} \\
D^{+} &= \{\, N^{0}\,[\,\underline{N^{+}}\,\underline{Z^{0}}\,]\,\} \\
D^{-} &= \{\, [\,\underline{Z^{0}}\,\underline{N^{-}}\,]\,N^{0}\,\}
\end{aligned}
$$

Table C.2: Codons for adjoining straight lines

$$
\begin{aligned}
0a &= \{\, N^{0}\,[\,\underline{N^{+}}\,N^{0}\,\underline{N^{-}}\,]\,\diamond\,\} \\
1a &= \{\, P^{0}\,[\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,Z^{-}\,\underline{N^{-}}\,]\,\diamond\,\} \\
1b &= \{\, P^{0}\,[\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,]\,\diamond\,\} \\
1c &= \{\, P^{0}\,[\,\underline{P^{+}}\,]\,\diamond\,\} \\
2a &= \{\, N^{0}\,[\,\underline{N^{+}}\,Z^{+}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,Z^{-}\,\underline{N^{-}}\,]\,\diamond\,\} \\
2b &= \{\, N^{0}\,[\,\underline{N^{+}}\,Z^{+}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,]\,\diamond\,\} \\
2c &= \{\, N^{0}\,[\,\underline{N^{+}}\,Z^{+}\,\underline{P^{+}}\,]\,\diamond\,\} \\
2d &= \{\, [\,\underline{N^{-}}\,N^{0}\,\underline{N^{+}}\,]\,\diamond\,\} \\
Ea &= \{\, [\,\underline{Z^{0}}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,]\,\diamond\,\} \\
Eb &= \{\, [\,\underline{Z^{0}}\,\underline{P^{+}}\,P^{0}\,\underline{P^{-}}\,Z^{-}\,\underline{N^{-}}\,]\,\diamond\,\} \\
Ec &= \{\, [\,\underline{Z^{0}}\,\underline{P^{+}}\,]\,\diamond\,\} \\
Ed &= \{\, [\,\underline{Z^{0}}\,\underline{N^{-}}\,]\,\diamond\,\}
\end{aligned}
$$

Table C.3: Codons for representing the ends of open curves

$$
\begin{aligned}
\mathsf{Fa} &= \{\diamond[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\diamond\}\\
\mathsf{Fb} &= \{\diamond[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}]\diamond\}\\
\mathsf{Fc} &= \{\diamond[\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\diamond\}\\
\mathsf{Fd} &= \{\diamond[\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}]\diamond\}\\
\mathsf{Fe} &= \{\diamond[\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}]\diamond\}\\
\mathsf{Ff} &= \{\diamond[\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}]\diamond\}\\
\mathsf{Fg} &= \{\diamond[\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}]\diamond\}
\end{aligned}
$$

Table C.4: Codons for representing curves with no curvature minima

$$
\begin{aligned}
\mathsf{Xa} &= \{\mathsf{P}^0[\underline{\mathsf{P}^+}]\mathsf{U}_>,\ \mathsf{P}^0[\underline{\mathsf{P}^+}]\mathsf{U}_\succ\}\\
\mathsf{Xb} &= \{\mathsf{N}^0[\underline{\mathsf{N}^+}]\mathsf{U}_>,\ \mathsf{N}^0[\underline{\mathsf{N}^+}]\mathsf{U}_\succ\}\\
\mathsf{Xc} &= \{\mathsf{P}^0[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_>,\ \mathsf{P}^0[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\succ\}\\
\mathsf{Xd} &= \{\mathsf{N}^0[\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_>,\ \mathsf{N}^0[\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\succ\}\\
\mathsf{Xe} &= \{\mathsf{N}^0[\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}]\mathsf{U}_>,\ \mathsf{N}^0[\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}]\mathsf{U}_\succ\}\\
\mathsf{Xf} &= \{[\underline{\mathsf{Z}^0}]\mathsf{U}_>,\ [\underline{\mathsf{Z}^0}]\mathsf{U}_\succ\}\\
\mathsf{Xg} &= \{\diamond[\underline{\mathsf{P}^+}]\mathsf{U}_>,\ \diamond[\underline{\mathsf{P}^+}]\mathsf{U}_\succ,\ \diamond[\underline{\mathsf{P}^-}]\mathsf{U}_>,\ \diamond[\underline{\mathsf{P}^-}]\mathsf{U}_\succ\}\\
\mathsf{Xh} &= \{\diamond[\underline{\mathsf{N}^+}]\mathsf{U}_>,\ \diamond[\underline{\mathsf{N}^+}]\mathsf{U}_\succ,\ \diamond[\underline{\mathsf{N}^-}]\mathsf{U}_>,\ \diamond[\underline{\mathsf{N}^-}]\mathsf{U}_\succ\}\\
\mathsf{Xi} &= \{\mathsf{U}_>[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_>,\ \mathsf{U}_>[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\succ,\ \mathsf{U}_\succ[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_>,\ \mathsf{U}_\succ[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\succ\}\\
\mathsf{Xj} &= \{\mathsf{U}_<[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_<,\ \mathsf{U}_<[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\prec,\ \mathsf{U}_\prec[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_<,\ \mathsf{U}_\prec[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\prec\}\\
\mathsf{Xk} &= \{\mathsf{U}_<[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_>,\ \mathsf{U}_<[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\succ,\ \mathsf{U}_\prec[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_>,\ \mathsf{U}_\prec[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\succ\}\\
\mathsf{Xl} &= \{\mathsf{U}_>[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_<,\ \mathsf{U}_>[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\prec,\ \mathsf{U}_\succ[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_<,\ \mathsf{U}_\succ[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\mathsf{U}_\prec\}\\
\mathsf{Xm} &= \{\mathsf{U}_>[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\diamond,\ \mathsf{U}_\succ[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}]\diamond\}\\
\mathsf{Xn} &= \{\mathsf{U}_>[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}]\diamond,\ \mathsf{U}_\succ[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}]\diamond\}\\
\mathsf{Xo} &= \{\mathsf{U}_>[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\underline{\mathsf{Z}^0}],\ \mathsf{U}_\succ[\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\underline{\mathsf{Z}^0}]\}\\
\mathsf{Xp} &= \{\mathsf{U}_>[\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}]\diamond,\ \mathsf{U}_\succ[\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}]\diamond\}
\end{aligned}
$$

Table C.5: Codons for representing convex angles and cusps

$$
\begin{aligned}
\mathsf{Va} &= \{\, \mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,]\,\mathsf{U}_{<},\ \mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vb} &= \{\, \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,]\,\mathsf{U}_{<},\ \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vc} &= \{\, \mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_{<},\ \mathsf{P}^0\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vd} &= \{\, \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_{<},\ \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,\mathsf{Z}^+\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Ve} &= \{\, \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{<},\ \mathsf{N}^0\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vf} &= \{\, [\,\underline{\mathsf{Z}^0}\,]\,\mathsf{U}_{<},\ [\,\underline{\mathsf{Z}^0}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vg} &= \{\, \diamond\,[\,\underline{\mathsf{P}^+}\,]\,\mathsf{U}_{<},\ \diamond\,[\,\underline{\mathsf{P}^+}\,]\,\mathsf{U}_{\prec},\ \diamond\,[\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_{<},\ \diamond\,[\,\underline{\mathsf{P}^-}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vh} &= \{\, \diamond\,[\,\underline{\mathsf{N}^+}\,]\,\mathsf{U}_{<},\ \diamond\,[\,\underline{\mathsf{N}^+}\,]\,\mathsf{U}_{\prec},\ \diamond\,[\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{<},\ \diamond\,[\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vi} &= \{\, \mathsf{U}_{>}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{>},\ \mathsf{U}_{>}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\succ},\ \mathsf{U}_{\succ}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{>},\ \mathsf{U}_{\succ}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\succ}\,\} \\
\mathsf{Vj} &= \{\, \mathsf{U}_{<}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{<},\ \mathsf{U}_{<}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\prec},\ \mathsf{U}_{\prec}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{<},\ \mathsf{U}_{\prec}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vk} &= \{\, \mathsf{U}_{<}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{>},\ \mathsf{U}_{<}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\succ},\ \mathsf{U}_{\prec}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{>},\ \mathsf{U}_{\prec}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\succ}\,\} \\
\mathsf{Vl} &= \{\, \mathsf{U}_{>}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{<},\ \mathsf{U}_{>}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\prec},\ \mathsf{U}_{\succ}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{<},\ \mathsf{U}_{\succ}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\mathsf{U}_{\prec}\,\} \\
\mathsf{Vm} &= \{\, \mathsf{U}_{<}\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\diamond,\ \mathsf{U}_{\prec}\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,]\,\diamond\,\} \\
\mathsf{Vn} &= \{\, \mathsf{U}_{<}\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,]\,\diamond,\ \mathsf{U}_{\prec}\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\mathsf{Z}^-\,\underline{\mathsf{N}^-}\,]\,\diamond\,\} \\
\mathsf{Vo} &= \{\, \mathsf{U}_{<}\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\underline{\mathsf{Z}^0}\,],\ \mathsf{U}_{\prec}\,[\,\underline{\mathsf{P}^+}\,\mathsf{P}^0\,\underline{\mathsf{P}^-}\,\underline{\mathsf{Z}^0}\,]\,\} \\
\mathsf{Vp} &= \{\, \mathsf{U}_{<}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\diamond,\ \mathsf{U}_{\prec}\,[\,\underline{\mathsf{N}^+}\,\mathsf{N}^0\,\underline{\mathsf{N}^-}\,]\,\diamond\,\}
\end{aligned}
$$

Table C.6: Codons for representing concave angles and cusps

Bibliography

Allen, J. F. (1984), 'Towards a general theory of action and time', *Artificial Intelligence* **23**, 123–154.

Ballard, D. H. (1981), 'Strip trees: A hierarchical representation for curves', *Communications of the ACM* **24**(5), 310–321.

Ballard, D. H. & Brown, C. M. (1982), *Computer Vision*, Prentice-Hall, Inc.

Biederman, I. (1987), 'Recognition-by-components: A theory of human image understanding', *Psychological Review* **94**(2), 115–147.

Biederman, I. (1988), Aspects and extensions of a theory of human image understanding, *in* Z. Pylyshyn, ed., 'Computational processes in human vision: An interdisciplinary perspective', Norwood, NJ: Ablex, pp. 370–428.

Blum, H. & Nagel, R. N. (1978), 'Shape description using weighted symmetric axis features', *Pattern Recognition* **10**, 167–180.

Boyle, R. D. & Thomas, R. C. (1988), *Computer Vision: A First Course*, Blackwell Scientific Publications.

Brady, M. (1983), Criteria for representations of shape, *in* J. Beck, B. Hope & A. Rosenfeld, eds, 'Human and machine vision', Academic Press, Inc., pp. 39–84.

Brady, M. & Asada, H. (1984), 'Smoothed local symmetries and their implementation', *The International Journal of Robotics Research* **3**(3), 36–61.

Chase, S. C. (1989), 'Shapes and shape grammars: from mathematical model to computer implementation', *Environment and planning B: Planning and Design* **16**, 215–242.

Chung, J.-M. & Ohnishi, N. (1997), Chain of circles for matching and recognition of planar shapes, *in* 'Proceedings of 15th IJCAI', Vol. 2, Morgan Kaufmann Publishers, Inc., pp. 1482–1487.

Cinque, L. & Lombardi, L. (1995), 'Shape description and recognition by a multiresolution approach', *Image and Vision Computing* **13**, 599–607.

Clementini, E. & Felice, P. D. (1997), 'A global framework for qualitative shape description', *GeoInformatica* **1**, 11–27.

Cohn, A. G. (1995), A hierarchical representation of qualitative shape based on connection and convexity, *in* A. U. Frank & W. Kuhn, eds, 'Spatial Information Theory: A Theoretical Basis for GIS', Vol. 988 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 311–326.

Cohn, A. G. (1997), Qualitative spatial representation and reasoning techniques, *in* G. Brewka, C. Habel & B. Nebel, eds, 'Proceedings of KI-97', Vol. 1303 of *LNAI*, Springer-Verlag, pp. 1–30.

Forbus, K. D. (1990), Qualitative physics: Past, present, and future, *in* D. S. Weld & J. de Kleer, eds, 'Readings in Qualitative Reasoning about Physical Systems', Morgan Kaufmann Publishers, Inc., pp. 11–39.

Freeman, H. (1974), 'Computer processing of line-drawing images', *Computing Surveys* **6**(1), 57–97.

Freeman, H. (1978), 'Shape description via the use of critical points', *Pattern Recognition* **10**, 159–166.

Freksa, C. & Röhrig, R. (1993), Dimensions of qualitative spatial reasoning, *in* N. P. Carreté & M. G. Singh, eds, 'Qualitative Reasoning and Decision Technologies', CIMNE, pp. 483–492.

Galton, A. P. (2000), *Qualitative Spatial Change*, Oxford University Press.

Galton, A. P. & Meathrel, R. C. (1999), Qualitative outline theory, *in* T. Dean, ed., 'Proceedings of 16th IJCAI', Vol. 2, Morgan Kaufmann Publishers, Inc., pp. 1061–1066.

Hayes, P. J. (1985), The second naive physics manifesto, *in* J. R. Hobbs & R. C. Moore, eds, 'Formal Theories of the Commonsense World', Ablex Publishing Corporation, Norwood, NJ, pp. 1–36.

Hayes, P. J. & Leyton, M. (1989), Processes at discontinuities, *in* 'Proceedings of 11th IJCAI', Vol. 2, Morgan Kaufmann Publishers, Inc., pp. 1267–1272.

Hernández, D. (1994), *Qualitative Representation of Spatial Knowledge*, Vol. 804 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.

Hernández, D., Clementini, E. & Felice, P. D. (1995), Qualitative distances, *in* A. U. Frank & W. Kuhn, eds, 'Spatial Information Theory: A Theoretical Basis for GIS', Vol. 988 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 45–57.

Herskovits, A. (1997), Language, spatial cognition, and vision, *in* O. Stock, ed., 'Spatial and Temporal Reasoning', Kluwer Academic Publishers, pp. 155–202.

Hoffman, D. D. & Richards, W. A. (1982), Representing smooth plane curves for recognition: implications for figure-ground reversal, *in* 'Proceedings of AAAI-82', American Association for Artificial Intelligence, pp. 5–8.

Hoffman, D. D. & Richards, W. A. (1984), 'Parts of recognition', *Cognition* **18**, 65–96.

Hoffman, D. D. & Singh, M. (1997), 'Salience of visual parts', *Cognition* **63**(1), 29–78.

Karinthi, R. R. & Nau, D. S. (1989), Using a feature algebra for reasoning about geometric feature interactions, *in* 'Proceedings of 11th IJCAI', Vol. 2, Morgan Kaufmann Publishers, Inc., pp. 1219–1224.

Kulpa, Z. (1994), 'Diagrammatic representation and reasoning', *Machine Graphics & Vision* **3**(1/2), 77–103.

Latecki, L. & Röhrig, R. (1993), Orientation and qualitative angle for spatial reasoning, *in* R. Bajcsy, ed., 'Proceedings of 13th IJCAI', Morgan Kaufmann Publishers, Inc., pp. 1544–1549.

Lee, D. T. (1982), 'Medial axis transformation of a planar shape', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **4**(4), 363–369.

Leyton, M. (1988), 'A process-grammar for shape', *Artificial Intelligence* **34**, 213–247.

Leyton, M. (1989), 'Inferring causal history from shape', *Cognitive Science* **13**, 357–387.

Marr, D. & Nishihara, H. K. (1978), 'Representation and recognition of the spatial organization of three-dimensional shapes', *Proceedings of the Royal Society of London, Series B* **200**, 269–294.

Meathrel, R. C. & Galton, A. P. (2000), Qualitative representation of planar outlines, *in* W. Horn, ed., 'Proceedings of 14th ECAI', IOS Press, pp. 224–228.

Meathrel, R. C. & Galton, A. P. (2001), A hierarchy of boundary-based shape descriptors, *in* B. Nebel, ed., 'Proceedings of 17th IJCAI', Morgan Kaufmann Publishers, Inc., pp. 1359–1364.

Mokhtarian, F. & Mackworth, A. (1986), 'Scale-based description and recognition of planar curves and two-dimensional shapes', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**(1), 34–43.

Mokhtarian, F., Abbasi, S. & Kittler, J. (1996), 'Efficient and robust retrieval by shape content through curvature scale space'. Proc. International Workshop on Image DataBases and MultiMedia Search.

Piazzalunga, U. & Fitzhorn, P. (1998), 'Note on a three-dimensional shape grammar interpreter', *Environment and planning B: Planning and Design* **25**, 11–30.

Pizer, S. M., Oliver, W. R. & Bloomberg, S. H. (1987), 'Hierarchical shape description via the multiresolution symmetric axis transform', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **9**(4), 505–511.

Pratt, I. (1999), 'Shape representation using fourier coefficients of the sinusoidal transform', *Journal of Mathematical Imaging and Vision* **10**, 221–235.

Prusinkiewicz, P. & Lindenmayer, A. (1990), *The Algorithmic Beauty of Plants*, Springer-Verlag.

Randell, D. A., Cui, Z. & Cohn, A. G. (1992), A spatial logic based on regions and connection, *in* 'Proc. 3rd Int. Conf. on Knowledge Representation and Reasoning', pp. 165–176. Cambridge MA, Oct. 1992.

Richards, W. & Hoffman, D. D. (1987), Codon constraints on closed 2D shapes, *in* M. A. Fischler & O. Firschein, eds, 'Readings in Computer Vision: Issues, Problems, Principles, and Paradigms', Morgan Kaufmann Publishers, Inc., pp. 700–708.

Richards, W., Dawson, B. & Whittington, D. (1988), Encoding contour shape by curvature extrema, *in* W. Richards, ed., 'Natural Computation', The MIT Press, pp. 83–98.

Rosin, P. L. (1993), 'Multiscale representation and matching of curves using codons', *CVGIP: Graphical Models and Image Processing* **55**(4), 286–310.

Rozenberg, G. & Salomaa, A. (1980), *The Mathematical Theory of L Systems*, Academic Press.

Saund, E. (1992), 'Putting knowledge into a visual shape representation', *Artificial Intelligence* **54**, 71–119.

Schlieder, C. (1996), Qualitative shape representation, *in* P. A. Burrough & A. U. Frank, eds, 'Geographic Objects with Indeterminate Boundaries', London: Taylor & Francis, pp. 123–140.

Shoham, Y. (1985), Naive kinematics: one aspect of shape, *in* 'Proceedings of 9th IJCAI', Vol. 1, Morgan Kaufmann Publishers, Inc., pp. 436–442.

Stiny, G. (1980), 'Kindergarten grammars: designing with froebel's building gifts', *Environment and planning B: Planning and Design* **7**, 409–462.

Weld, D. S. & de Kleer, J., eds (1990), *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann Publishers, Inc.

Witkin, A. P. (1983), Scale-space filtering, *in* A. Bundy, ed., 'Proceedings of 8th IJCAI', Vol. 2, pp. 1019–1022.