# Studies on the Formal Semantics of Pictures

**Dejuan Wang**

# Studies on the Formal Semantics of Pictures

ILLC Dissertation Series 1995-4

institute *for* logic, language *and* computation

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam
phone: +31-20-5256090
fax: +31-20-5255101
e-mail: illc@fwi.uva.nl

# Studies on the Formal Semantics of Pictures

Academisch Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam,
op gezag van de Rector Magnificus
Prof.dr P.W.M. de Meijer
in het openbaar te verdedigen in de
Aula der Universiteit
(Oude Lutherse Kerk, ingang Singel 411, hoek Spui)
op woensdag 17 januari 1995 te 15.00 uur

door

Dejuan Wang

geboren te Chongqing China.

Promotor: Prof.dr.ir. R.J.H. Scha
Faculteit der Letteren
Universiteit van Amsterdam
Spuistraat 134
1012 VB Amsterdam

Copromotor: Dr. J.R Lee
Human Communication Research Center
Universiteit van Edinburgh
2 Buccleuch Place
Edinburgh
Schotland

CIP gegevens



ISBN: 90-74795-21-8

# Contents

# Acknowledgments

The work on my thesis started under the supervision of John Lee in EdCAAD (Edinburgh Computer-Aided Architecture Design, Edinburgh University). After two years, I moved to Amsterdam and carried on the work in the Computational Linguistics Department of Amsterdam University under the supervision of Remko Scha.

I wish to thank the many people who have helped me in the course of this work. First of all, Remko Scha, my PhD promotor and supervisor, has provided many suggestions and much helpful advice. John Lee, my PhD copromoter, gave me inspiration and encouragement. He has also made various contributions to this work. Special thanks go to Don Sannella, my second supervisor in Edinburgh. Zhaohui Luo gave me substantial help, teaching me type systems and algebraic program specification. He always gives essential comments on my work. Thanks also go to Luis Pineda, Ewan Klein, Keith Stenning and Piero Mussio for discussions and valuable comments on my work. Special thanks go to David Beaver who read the entire draft of the thesis and helped me in correcting many mistakes against the English language. Henk Zeevat translated the abstract into Dutch. He also gave many useful suggestions on my work.

Colleagues both in Edinburgh and Amsterdam helped me in lots of ways. James Lothian helped me with his Gizmo system, in which the GAR system is implemented. Margaret McDougall gave me constant good care and Ian Thurlbeck for providing system support. Mehdi and Arie helped me with the facilities in the Department. Special thanks to my room mates in Amsterdam for their friendly support.

I owe many thanks to Sissi Burstall but she cannot hear this any more. Her great care of me not only made me feel at home during the first couple of years living in Edinburgh, but also gave me encouragement to overcome the language barrier so that I could start research work again. Thanks also go to Rod Burstall, Irene Neilson, Yangling Xiang, Chengzhi Peng, Yiqun Gu, Fangjing Xu, Maria, Inge and Iet for friendship, many invitations and taking care.

Parts of this thesis were previously published in:

- Dejuan Wang, John Lee & Henk Zeevat (1994) Reasoning with Diagrammatical Representations. In: *Diagrammatic Reasoning* editor H. Narayanan, AAAI press and MIT press (to appear).
- Dejuan Wang & John Lee (1993) Visual Reasoning: its Formal Semantics and Applications. *Journal of Visual Languages and Computing* 4, 327-356.
- Dejuan Wang & John Lee (1993) Pictorial Concepts and a Concept-supporting Graphical System. *Journal of Visual Languages and Computing* 4, 177-199.
- Dejuan Wang, John Lee & Henk Zeevat (1993) G-morphisms and Reasoning with Diagrammatic Representations. In IJCAI Workshop on Principles of Hybrid Representation and Reasoning. IJCAI'93, Chambery, France, September, pp. 39-49.
- Dejuan Wang (1992) Graphical Communication and Reasoning: an interpretation-based approach. In *AAAI Workshop on Communicating Scientific and Technical Knowledge.* July, pp. 43-50.
- Dejuan Wang & John Lee (1991) Graphics-Assisted Reasoning. In: *EUROGRAPHICS Workshop on Computer Graphics and Mathematics.* October, pp. 59-78.

Amsterdam Summer Dejuan Wang
July, 1994.

# 1 Introduction

## 1.1 Background

Pictures provide us with a language which vividly expresses spatial concepts like geometrical shapes, sizes, positions and spatial relations. People can read this language immediately because we are in such a space and spatial concepts are part of our nature.

When we reason about a problem or when we explain something in a non-graphical domain, we quite naturally draw pictures and give meanings to the graphical objects and their spatial relations in the pictures (*e.g.* circles represent sets). Then the observation of graphical objects and their spatial properties becomes the observation of the problem being tackled. This often makes some features of the problem appear trivial and obvious, where they would otherwise have required a considerable amount of work and might never have been discovered.

I will use the term *visual communication* in this thesis for communication involving the use of pictures. Intuitively, visual communication involves three parts: a graphical domain (the pictures), an application domain (the problems) and a link which associates the graphical domain with the application domain (the semantics of the pictures).

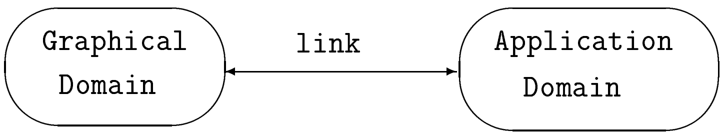



Figure 1.1: The three parts of visual communication.

The link plays an important role in visual communication. Without the link, pictures only carry spatial information. It is the link which transfers spatial information into information about the application domain, so that pictures can be helpful in communication. Furthermore, establishing such a link is not a trivial matter. First, the link should associate pictures with the subject matter in the application domain in a way which people view as natural. For instance, the use of Venn Diagrams representing set theory is natural, as is also attested by its popularity. However, diagrams in which lines represent sets and t-joint lines represent elements are hardly popular. Second, the link should associate pictures with the subject matter in the application domain in a way which is not misleading. Imagine, for example, that spatial inclusion between circles is used to represent the father-son relation. Then, the fact that circle A is included in circle B, may mean that John is Bill's father. That circle B is included in circle C means that Charles is John's father. Now the picture suggests the inference that Charles is Bill's father, but this is not what we want. Apparently, the link associates this picture and the application domain in an incorrect way.

## 1.2 Topic

There is a growing interest in computer systems that support visual communication. Since the essential property of such systems is that they support a proper semantics of pictures, studying the semantics of pictures (*i.e.* the link in Figure 1.1) has become important. Some research is directed towards looking for natural graphical representations by studying human psychology. And other research is focussed on the logical foundations for a cognitive theory of picture semantics (*e.g.* how Euler circles are used to solve syllogisms [57]; or see other works in [49] [1]).

However, there is no study of the general *methods* by means of which the link can be specified and by means of which the correctness of the semantics of pictures can be defined. When I say methods, I mean methods in the computer science tradition. I.e. effective formal frameworks for the specification of picture description languages, for describing pictures in such languages, for the specification of the interpretation of pictures (the link in Figure 1.1) and for carrying out the reasoning that such interpreted pictures support. This will be the business of this PhD thesis.

Although results of this thesis have led to implementations (See Section 10.2, 15.1 and 15.5.2), the primary aim of the thesis is theoretical. The general methods I am interested in will be formal and logic-based, but the aim is not to provide a "logic of pictures" (it is not clear that this is possible). The study of the meaning of graphical representations is closely connected to the study of semantics in general (*e.g.* Natural Language semantics) and has interesting connections to the theory of knowledge and human reasoning. However, these questions will not be emphasized in this thesis, as they fall outside my competence. What is emphasized in this thesis is providing formal frameworks for specifying picture semantics. Computer systems that support general visual communication [1] can be designed on the basis of such

[1] *General visual communication* used in this thesis means that the application domain and the semantics of pictures are not fixed. Most current visual communication systems do not support general visual communication because the semantics of the picture is predefined. Some of them can

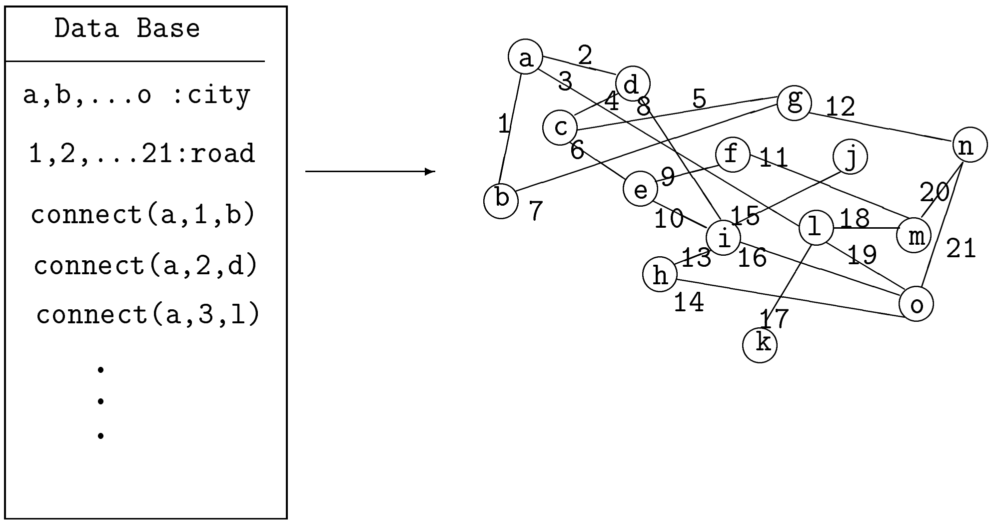
frameworks.

In particular, this thesis concerns two frameworks for specifying the link in Figure 1.1, namely, *interpretations* and *picture specifications*.

An *interpretation* approach assigns meanings to each graphical entity explicitly, and the semantics of a picture is determined from the meanings of its components and their spatial relations. A *picture specification* approach assigns meanings to pictures in an implicit way, *i.e.* it creates picture classes according to the requirements of the problems being tackled in the application domain.

## 1.3 Examples of the two approaches

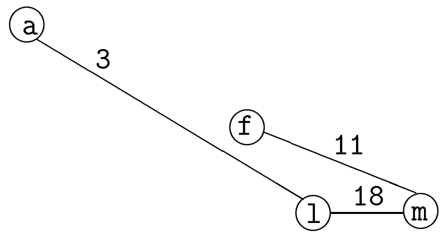
Let's see some examples to get a rough impression of these two approaches. The interpretation approach specifies the semantics of a picture by interpreting graphical objects and their spatial properties as the objects and properties of an application domain. Visual communication can then be realised by reading the geometrical properties and relations in the picture. For example, we may interpret *circles* as *cities*, *lines* as *roads* and the property that *two circles are connected by a line* as the property that *the road connects the two cities*. We may also interpret other graphical entities like *the path between two circles* as *the path between two cities* and so on. Information about traffic connections between cities can then be represented and, furthermore, certain problems may be solved by graphical inference. For instance, the information on the traffic connections between cities in a data base can be represented as follows.



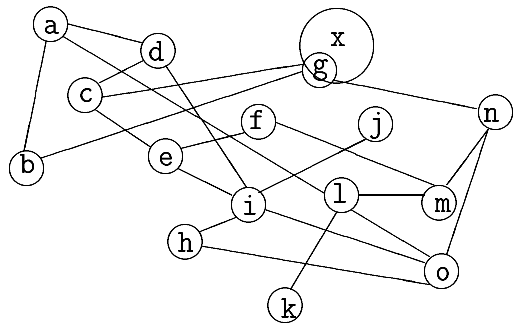

We can ask questions like *What is the shortest path, in terms of number of roads traversed, between city a and city f*? The system may answer the question by reasoning over the picture according to the interpretation (*i.e.* it understands that the

be used in different application domains but the application domain cannot be arbitrary and must fit the predefined picture semantics.

shortest path between cities is the shortest path between circles) and by illustrating the result.




On the other hand, by means of the picture specification approach, a class of pictures is specified so that the meanings are implicitly incorporated in the picture specification. For example, we may specify a class, say *Traffic-map*, whose pictures consist of a set of (non-overlapping) circles and a set of lines for each of which there are two circles such that the two end-points of the line connect with each of the circles. Having specified such a class of pictures, the supporting system treats the new class *Traffic-map* in the same way as other classes of pictures like *Circle*, *Rectangle etc*, so that we may draw (or the supporting system generates) pictures of the class *Traffic-map* to represent implicitly some special traffic maps. We may also manipulate the picture by adding (or deleting) circles or lines. What we manipulate pictures that belong to the class *Traffic-map*, the supporting system may tell us whether or not the resulting drawing is an object of *Traffic-map*, or it may maintain the specification by rejecting or adjusting the resulting picture if it fails to satisfy the specification of *Traffic-map*. For instance, if we add a city $x$ to the following picture, the system might reject the addition, or put the circle $x$ somewhere else, because the position we chose breaks the constraint that the circles do not overlap each other.




The picture specification approach is not, properly speaking, an approach to the semantics of pictures, but a syntactic approach. However, people who work in picture specification consider picture specifications to be an approach to the semantics of pictures (see [29] for example). The reason is that, in practice, a class of pictures is always specified with the purpose to represent the subject matter of an application domain. The subject matter determines the syntactic construction of the picture class. Thus, meanings are implicitly carried by the structure of the pictures.

One may wonder why we need two approaches instead of only one. The reason is that they suit different kinds of visual communication. The interpretation approach is more suitable for visual reasoning which emphasizes manipulations of pictures. Graphical objects, graphical operations and graphical properties play different roles and can be combined together in a flexible way. Consider using Venn diagrams to illustrate the distributive law of set operations $\cup$ and $\cap$. The validity of the equation

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

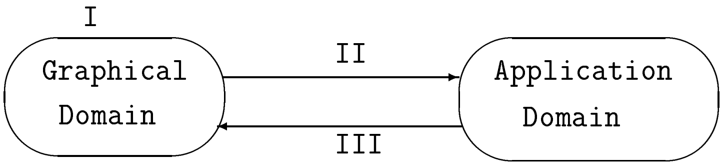
is illustrated step by step demonstrating graphical inference over circles and other closed curves (the representation of $B \cap C$ and of $A \cup (B \cap C)$ for example). If the meanings of a picture cannot be decomposed into the meanings of its components, such manipulations over the picture would be unnatural and difficult.

The picture specification approach, however, is more suited for another kind of visual communication (such as design) in which pictures are used to represent special objects in an application domain. For example, when designing a cooker, the designer may specify a class *Cooker* whose pictures consist of a rectangle with two to four circles inside it, and where the distances between circles are greater than some unit. What she needs for this design task is to find a particular picture in the class *Cooker*. Maintenance of the specification of the class *Cooker* by the system can be helpful for her. For instance, if she drags one circle toward another, she does not need to worry about whether or not the two circles are too close, for the system guarantees that the distance is always greater than the unit. This use of pictures is concentrated on the objects of a picture class (*e.g.* design can be viewed as looking for 'better' objects in a class of pictures).

In principle, these two approaches can be fitted into one framework where the picture specification is for creating complex picture classes and the other for giving meanings to picture classes. A useful aspect of such a combination would be that constraints on pictures can be derived from facts about the application domain. Moreover, the constraints and the domain would naturally serve as each other's correctness criteria. The reason for not carrying out this program is practical. Current picture specification systems tend to be concerned with very complex graphical objects. For developing semantics we prefer to keep graphical objects relatively simple. In future work it may well turn out that combined systems can be fruitfully developed.

## 1.4 Structure of the thesis

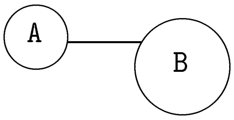
The model in Figure 1.1 can now be enriched as in Figure 1.2 where the link in Figure 1.1 became two arrows. One arrow, from a graphical domain to an application domain, represents the interpretation approach which explicitly interprets graphical entities as objects and their relations in the application domain. Another one, from an application domain to a graphical domain, represents the picture specification approach which specifies new classes of pictures in the graphical domain according to the requirements of the problem solving in the application domain.




| | |
|---|---|
| Part I | Geometrical characterisation of pictures |
| Part II | Interpretations:<br>interpreting graphical entities as objects and their relations in the application domain |
| Part III | Picture specifications:<br>specifying new picture classes according to the requirements of the application domain |

Figure 1.2: This Thesis

In order to study the interpretation approach, we need to start by studying pictures. This includes: providing a formal framework for the specification of picture description languages and characterising what one can 'see' in a picture. I use an example to illustrate this. Consider the following picture.




Tom says: "Let *circles* be *sets*, *two circles connected by a line* be the *two sets have common elements* and *one circle is bigger than another* be *the set has more elements than the another set*." Then the above picture becomes $A \cap B \neq \emptyset$ and $\mid B \mid > \mid A \mid$. On another occasion, Bill says: "Let *circles* be *cities* and the *lines connecting circles* be the *roads connecting cities*. Then the picture becomes *There is a road connecting city A and city B*.

There are interesting issues in this example. For instance, both of them were giving different meanings to the graphics (circles were sets for Tom and cities for Bill). In which way are the meanings associated with the picture? How do we know that the different meanings assigned correctly reflect the problems in the two application domains?

Furthermore, there are other issues in this example, which must be considered before the interpretation issue is studied. First, there is a language used to describe pictures, *e.g.* circles, lines, circles connected by lines, one circle is bigger than another. What kind of language is this? Second, if the language is fixed, is it guaranteed that the description of the picture in the language is always the same? For example, Tom says: "Circles A and B are connected by a line and circle B is bigger than A". Bill says: "There is a line connecting circle A and B". What causes them to describe the same picture in a different way? Does Bill also see that circle B is bigger than circle A? What does that mean for him? Do they also see that circle A is to the left of circle B and higher than circle B and the many other geometrical facts in the pictures? Do they see that for any circle $x$ and $y$, if $x$ is connected with $y$ by a line then $y$ must be connected with $x$ by the same line? Correctly characterising what they see in the picture is so important because what they see in the picture becomes the observation of the properties of sets for Tom and the road traffic between cities for Bill. The study of interpretations must be based on proper answers to the above questions about pictures.

Based on these considerations, this thesis consists of three parts (illustrated in Figure 1.2): (1) geometrical characterisation of pictures, (2) interpretation and (3) picture specification.

In Part I, we study the geometrical characterisation of pictures, *i.e.* how to describe pictures by means of a geometrical theory. The main tasks in this part are as follows: first, to provide a general framework for specifying picture description languages (*i.e.* a language structure for describing graphics which is able to express those features of pictures which are used in visual communication); second, to develop a formal way to give pictures a geometrical characterisation based on a picture description language.

Order sorted signatures [24] are used to give a general structure to picture description languages. Languages with this structure can be highly expressive. A picture description language with this structure can express graphical objects (both simple objects like circles and lines and complex objects like an object emerging by two overlapping circles) and spatial properties of (and relations between) graphical objects. Graphical objects are divided into different sorts and a partial order relation is built upon the sorts to characterise the inheritance relationship between graphical objects. Given a picture description language, a picture can be described as a set of graphical objects and spatial properties of the graphical objects which satisfy certain conditions (see Section 4.1 to Section 4.3). The conditions reflect my understanding about what one can "see" in a picture in visual communication so that they serve as a basis for retrieving information in a picture to contribute to visual communication. This abstract characterisation of picture description is the most essential step towards an understanding of the interpretation of pictures and the use of pictures to represent real world problems in visual communication.

In Part II, interpretations are studied on the basis of the results of Part I. I give a formal model of an interpretation and investigate how pictures can be used in visual communication by means of interpretations.

Interpretation is captured by a generalised notion of renaming — a mapping from names of graphical objects and their possible properties to those in the representation language of the real world problem domain. Technically, this is formalised by the notion of a signature morphism [23] which gives meanings to a subset of the names of graphical entities. The formal interpretation framework not only helps us to understand what an interpretation is, but, more importantly, it also serves as a structure to guide assigning semantics to pictures in the design of visual communication systems. More precisely, it verifies whether or not an interpretation is syntactically correct. Three properties of interpretations are studied, which, on the one hand, answer the question of how pictures are used in visual communication and, on the other, serve as a theory of semantic correctness for interpretations.

An environment supporting visual communication is discussed, which emphasizes that the meaning of pictures is embedded in their use. The meanings of pictures are not fixed by system designers, but instead, they are specified by the user with the help of a support system. Three kinds of visual communication via interpretations are discussed; illustrative visual communication, demonstrative visual communication and reasoning with diagrammatic representations. An experimental system GAR (Graphics-Assisted Reasoning) which supports demonstrative visual communication is presented. Reasoning with diagrammatic representations is studied further using the notion of a G-morphism (Graphical morphism).

In Part III, picture specifications are studied. The main tasks of this part are as follows. First, I develop a new method for picture specifications. The new method should be powerful in the sense that richer picture structures can be specified, and systematical in the sense that systems designed on the basis of this method can support users in specifying classes of pictures without using a programming language. Second, applications are presented to verify the theoretical approach.

I present a new and structured approach to the construction of classes of pictures. The theoretical foundations of this approach are given by a type-theoretic framework, which provides a rich specification language with abstraction mechanisms by means of which classes of pictures with sophisticated structures can be specified in a clear and natural way. In particular, the approach allows us to study, define and use classes at a high level of abstraction so that powerful and useful operations over classes can be defined and systematically used to form complicated classes. An experimental system CSGS (Concept Supporting Graphical System), which supports users in specifying classes of pictures directly, is described to prove that the approach provides a solid computational basis for the design of computer system supporting picture specification. Three kinds of visual communication by means of picture specifications are classified: geometrical constraint maintenance, design and computational art. Further treatment of computational art is given and an experimental system CINONG which can synthesise some simple computational art programs is presented.

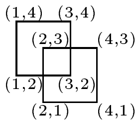
The relationships between the three parts are illustrated in Figure 1.2. As we described above, picture description languages presented in Part I are typed languages which divide graphical objects into various sorts like *Circle, Line, Polygon etc.* By means of picture specifications, new sorts (*e.g. Cooker*) can be dynamically created and added to a language for various applications. Therefore, apart from their main purpose (*i.e.* assigning meanings to pictures), picture specifications can also be used to enrich a picture description language in a dynamic way. Assigning meanings to pictures by interpretations is more general in the sense that the second approach (*i.e.* meanings are given by picture specifications) can be viewed as its special case. It can be understood as follows: (1) specifying a class of picture (*Cooker*) and (2) interpreting the new sort (*Cooker*) to the class of objects (cookers) in the application domain. Part I of this thesis is the basis of Part II. Conceptually, Part I and Part

III are closely related to each other, but technically, Part III is self-contained and can be read separately.

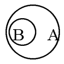
## 1.5 Some terminology

In this section, I briefly introduce some concepts around visual communication, which tend — when they are not properly distinguished — to lead to confusion.

- *Graphical representation languages*:
  A graphical representation language is not a symbolic language, it is a class of pictures which represent events or things. Consider some road builders working on a highway. There are two signposts 100 meters away to indicate this event. On one signpost, is written "Attention, road building 100m ahead". On the other one, a drawing of a red triangle in which a man is digging. The event of road building is described by natural language in the first signpost, and by a graphical representation language (traffic signs) in the second.
- *Visual Languages*:
  Visual Languages are graphical representation languages but restricted to either visual programming languages or program visualisation [10]. Flow charts are example of programs written in a visual programming language and visualising the information in a data base is an example of program visualisation.
- *Picture description languages*:
  A picture description language is a symbolic language for describing pictures. Example expressions in such languages are: "Circle A is inside square B" or "inside(circle(A),square(B))".
- *Graphical inference*:
  Graphical inference is for computing graphical objects formed by other graphical objects (emergent objects), and inferring properties of graphical objects. For instance, given the descriptions of two squares $A$ and $B$:




$$A = ((1,2)\ (3,2)\ (3,4)\ (1,4))$$
$$B = ((2,1)\ (4,1)\ (4,3)\ (2,3))$$

  another square $C = ((2,2)\ (3,2)\ (3,3)\ (2,3))$ is obtained by graphical inference. Properties of $A$ and $B$ can also be derived by graphical inference, *e.g.* $overlap(A, B)$.
- *Visual reasoning*:
  Visual reasoning is reasoning about problems in an application domain by means of pictures and their properties, and relationships derived from graphical inference (*c.f.* [9]). For example, a fact $B \subset A$ is obtained from a picture: 

  if the *inside* relation is interpreted as $\subset$ and *circles* as *sets*.
- *Visual communication*:
  Visual communication in this thesis means communication involving the use

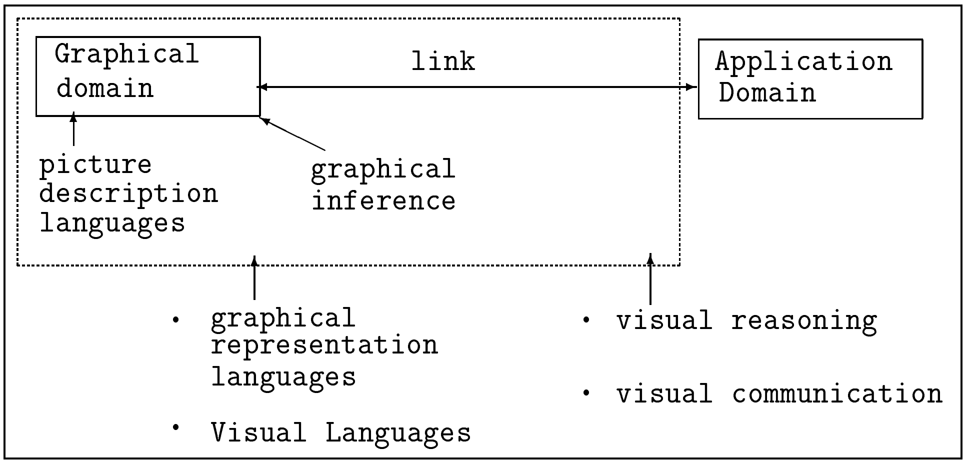



Figure 1.3: Some terminology.

of pictures. It is rather like the communication involving the use of natural languages in the sense that both pictures and natural languages carry meanings in another domain. Visual communication is the process of conveying the meaning from one agent to another by means of the graphical representation. The fact that a graphical representation has a meaning is a precondition for visual communication. There is an important difference with natural languages: meanings for graphical representation are not fixed conventionally, but generally come about in the use of graphical representations by people.

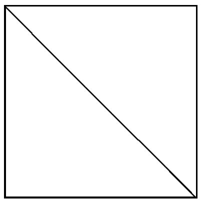
I include visual art under the label of visual communication (see Section 15.5) since graphical representation in visual art often has a meaning in the same sense. However, these meanings need not be presented and different meanings for the same graphical representation are allowed. This means we can use the same techniques in the computational handling of visual art, especially constructions, but the conveying of information is not a primary function, we do not need to assure or to guarantee that the user pick up the meaning as represented in the system.

# Part I

# Geometrical Characterisation of Pictures

# 2 Introduction to Part I

In this part, picture descriptions are studied. What is a picture description? Consider the following picture.



It may be described as: *two triangles sharing one side*, or *a square with a diagonal line*, or *five lines*, and so on. There are some problems that arise in such descriptions. First, although each of these descriptions seems to be acceptable, there is no reason to choose this one instead of the others. Second, none of these descriptions seems to be complete in the sense that no more details can be added to the description. For instance, the first one can be enriched by describing the triangles as *right angled* triangles, by describing the shared side as *hypotenuse*, by describing the length of each side, its orientation, its position in the plane. Thinking in this way, we may find it hard to obtain a description in which one can say that every detail in the picture has been included. However, when we use pictures to communicate with each other we are never troubled by such problems. In certain circumstances, we know how to describe a picture and we also know to what extent we need to describe it. For instance, if we use *squares* to represent *sets* and a *diagonal line* to represent the fact that *a set is empty*, we will describe the picture as *a square with a diagonal line*. Though it is clear that the diagonal line divides the square into two triangles, this does not interest us. If we use *triangles* to represent *sets* and *two triangles sharing one side* to represent *two sets having the same size*, the above picture will be described as *two triangles sharing one side*. Furthermore, if the *right angled* property is also used, *e.g.* to represent *integer sets*, the description will be *two right angled* *triangles sharing one side.*

What can we conclude from this example? It shows that a picture can be described in different ways. Our choice of the descriptions is directed by how we intend to use the picture. Choosing the proper description of pictures in virtue of their intended interpretations seems our natural ability. It seems also one of the reasons why visual communication is so popular with us.

Now the question is how picture descriptions can be formalised so that a solid basis for a visual communication support environment can be provided. This is the subject studied in this part. I present a formal approach which is used for describing pictures. It is my intention that this approach can be used to describe correctly what one can 'see' in a picture in visual communication.

The approach consists of two parts, an abstract notion of a picture description language and a set of principles for giving particular pictures a geometrical characterisation. The first part (picture description language) is related to various work on picture descriptions. The main distinction between my work and most other work results from having different purposes in describing pictures. Much work in this field is aimed at picture recognition and generation [18] [17] [41] [51] and constraint based picture specification [6] [29]. Such work is mainly concerned with describing pictures on the basis of current computer graphical techniques to serve directly some kind of visualisation; or with describing pictures on the basis of some application domain for recognition of the objects from photographs *etc.* However, my purpose is to formalise the way in which human beings 'read' pictures in visual communication. What I am interested in are those aspects of pictures that are used in communication. Therefore, techniques for computer graphics and visualisation will not be studied; instead, I will be mainly concerned with features of pictures relevant to their use as visual communication languages.

As the above example tells us, a picture can be 'read' in different ways according to the ways in which we intend to use it. This suggests that we are concerned with the abstract structure of picture description languages but not with a particular language. The structure is abstract in the sense that for a particular application, suitable graphical entities can be entered into the structure to give a particular picture description language for a special purpose. A picture description language consists of a graphical signature and a graphical theory. The expressions of a language are generated by a graphical signature and the geometrical meanings of the language expressions are characterised by a graphical theory over the graphical signature. The graphical signature I will use is an order sorted signature [24] which can express classes of graphical objects as well as individual ones, which reflects the inheritance relationship between classes of pictures and which gives a natural way of describing possible computations between graphical entities. The graphical signature embodies a well ordered structure which, on the one hand, reflects the internal relationships between graphical objects and on the other provides a natural way to associate pictures with real world problems via interpretations (see Part II). The importance of having a graphical theory as part of a picture description language is in giving a logical characterisation of graphical inference. It is one of my assumptions that the properties of graphical objects one can see in a picture can be derived by the algorithms for graphical inference.

An algebraic approach to picture description used by Pineda in his system *GRAFLOG* [52] provided some inspiration for the approach in this thesis. There are, however, some divergences. The picture description language in *GRAFLOG* is based on many sorted algebraic structures [25]. My picture description language is based on an order sorted algebra. Many sorted algebra is developed for program specifications. In order to avoid the various problems such as those resulting from partial functions in many sorted algebras, order sorted algebras were developed (for details see [24]). Apart from this consideration, in *GRAFLOG*, the meanings of the graphical entities in the language are given by geometrical algorithms and mine is given by a set of logical formulas (in a graphical theory) which describe the behaviour of geometrical algorithms in practice. My approach makes it natural and possible to give pictures a geometrical characterisation based on the picture description language in an accurate way. However, by means of the approach adopted in *GRAFLOG*, this would be difficult.

The work on the descriptions of particular pictures based on a picture description language is new as far as I know. Accuracy is emphasised in the sense that the description of a picture should correctly reflect what one 'sees' in the picture, no more and no less. This has been achieved by presenting a set of principles by means of which a picture is described as a set of graphical objects and a set of formulas representing properties of (and relations between) the graphical objects. The set of principles consists of the following aspects: (1) a *partial graphical signature* reflecting the fact that for a particular picture there are only some graphical objects in it; (2) *basic facts* (a basic fact is either an atomic formula or its negation) reflecting the fact that one can only 'see' basic facts in a picture (see Section 4.1); (3) *consistency* which guarantees that the basic facts included in the description of a picture are only what the picture tells us (see Section 4.2), and (4) *maximality* which guarantees that the basic facts included in the description of a picture are all that the picture tells us (see Section 4.3).

# 3 Picture description languages

In this section, an abstract notion of picture description language is presented. The notion is intended to account for the use of pictures in visual communication.

Using pictures in visual communication, one can 'see' in a picture not only a set of basic graphical objects (*e.g.* circles) and their properties (*e.g.* one circle is inside the other) but also emergent graphical objects (*e.g.* a closed curve emerging from two overlapping circles) and their properties. Therefore, a picture description language should be able to describe not only basic but also emergent graphical objects and their possible properties. Furthermore, one often uses a class of graphical objects (*e.g.* Venn diagrams) to represent a class of objects (*e.g.* sets) in the application domain. And several different classes are often used in visual communication, *e.g.* a class of circles represent cities and a class of lines represent the roads between cities. Therefore, it is natural to use a typed language. Finally, many classes of graphical objects have certain inheritance relationships between them in the sense that one class (*e.g.* that of circles) is a subclass of another (*e.g.* the class of closed curves). A picture description language reflecting such natural relationships can support natural choices of graphical representations and their implementation.

Based on the above considerations, I will use order-sorted logical languages[1] (*c.f.* [24]) as picture description languages, which naturally satisfy such requirements. The syntax of an order-sorted language is presented by an *order-sorted signature*, from which the expressions of the language are generated.

## 3.1 Graphical signature

Informally, an order-sorted signature of a picture description language consists of the following:

- a set of *sorts*,

[1]For generality, except for the order-sorted syntactical structure, I do not fix a logical system here. See, for example, [22, 23] and [45] for discussions of 'general logics'. However, the order-sorted first-order logic is assumed in the examples.

| | |
|---|---|
| $\mathcal{S}$ | *Point*, *Circle*, *Closed_curve*, *Graph*, *Null* |
| $\leq$ | $Null \leq s \leq Graph$, for all $s \in \mathcal{S}$<br>$s \leq s$ for all $s \in \mathcal{S}$<br>$Circle \leq Closed\_curve$ |
| $\mathcal{F}$ | $a, b, c, d$ : *Point*; $A, B, C, D$ : *Circle*; $E, F$ : *Closed_curve*;<br>$\perp$ : *Null*;<br>*overlap* : $Circle \times Circle \rightarrow Closed\_curve$;<br>*merge* : $Closed\_curve \times Closed\_curve \rightarrow Graph$;<br>... ... |
| $\mathcal{P}$ | $in_1, outside$ : $Point \times Circle$;<br>$in_2, overlapping$ : $Circle \times Circle$;<br>... ... |

Figure 3.1: An example graphical signature $\Sigma_0 = (\mathcal{S}, \leq, \mathcal{F}, \mathcal{P})$.

- a set of *constants* (*i.e.* nullary function symbols),
- a set of *function symbols*,
- a set of *predicate symbols* and
- a partial order between the sorts.

The sorts stand for classes of graphical objects. The constants and function symbols, together with a set of variables, constitute the class of terms representing graphical objects. Predicate symbols are used to construct formulas which represent properties of and relations between graphical objects. Formulas which do not have free variables are called *sentences.* The partial order between sorts expresses the subclass relationship between graphical objects.

The following gives a precise definition of a graphical signature.

**3.1.1.** Definition. (order-sorted signature and graphical signature)
*An* order-sorted signature $\Sigma$ *is a quadruple* $\Sigma = (\mathcal{S}, \leq, \mathcal{F}, \mathcal{P})$*, where*

- $(\mathcal{S}, \leq)$ *is a partially ordered set of* sorts*, with a largest sort and a smallest sort.*
- $\mathcal{F}$ *is a set of* function symbols *(including constants), each of which is associated with a principal type of the form* $s_1 \times ... \times s_n \rightarrow s$*, where* $n \geq 0$ *and* $s_i, s \in \mathcal{S}$*. In* $\mathcal{F}$*, there is a constant* $\perp$ *with the smallest sort as its principal type.*
- $\mathcal{P}$ *is a set of* predicate symbols*, each of which is associated with a principal domain of the form* $s_1 \times s_2 \times ... \times s_n$*, where* $n \geq 0$ *and* $s_i \in \mathcal{S}$*.*

*A* graphical signature *is just an order-sorted signature, whose sorts, function symbols and predicate symbols are called* graphical sorts *(with the largest* Graph *and the smallest* Null*),* graphical operations *and* graphical predicates*, respectively.*

In the following, I give some further explanations, using the example graphical signature in Figure 3.1.

### 3.1.1 Graphical sorts and partial order relation

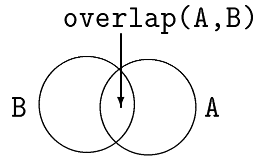
The graphical sorts represent classes of graphical objects (*e.g.* *Circle* for the class of circles). The partial order over the graphical sorts describes a subclass relationship between the classes of graphical objects. For instance, '$Circle \leq Closed\_curve$' expresses that every circle is a closed curve. A function symbol with principal type $s_1 \times ... \times s_n \to s$ also has $s'_1 \times ... \times s'_n \to s'$ as its type, if $s_i \geq s'_i$ and $s \leq s'$. Similarly, a predicate symbol with principal domain $s_1 \times ... \times s_n$ also has $s'_1 \times ... \times s'_n$ as its domain, if $s_i \geq s'_i$. For instance, for the signature in Figure 3.1, *merge* also has $Circle \times Circle \to Graph$ as its type.

### 3.1.2 Largest and smallest sorts

Every graphical signature has a largest sort *Graph*, a smallest sort *Null* and a constant $\perp$ of type *Null*. $\perp$ represents a *null* object and *Graph* stands for the class of arbitrary graphical objects. Note that every graphical sort has $\perp$ as an object and hence is non-empty. The sorts *Graph* and *Null* play an important role in avoiding partial function problems. For example, applying *overlap* to two non-overlapping circles forms a null object, and merging two arbitrary closed curves together may not form an object of any graphical sort except *Graph* (see the operation *merge* in Figure 3.1). By introducing a largest sort, a picture description language becomes complete in the sense that any picture can be denoted by expressions of the language. If there were not a largest sort in a graphical signature, then no matter how many sorts the signature has, there would be pictures containing graphical objects which do not belong to any of the sorts, and the picture description language would fail to describe such pictures. However, if a graphical signature has the largest sort, even if it has only few other sorts it will be able to describe any kind of picture because any graphical objects in the picture at least have the largest sort *Graph* as their sort.

### 3.1.3 Basic and emergent graphical objects

The graphical function symbols denote the possible operations over graphical objects; they are used to build the *terms* of the language. The terms in a graphical signature denote expressible graphical objects. Examples of terms in the graphical signature in Figure 3.1 are: $a$, $A$, $\perp$, $overlap(A, B)$, $overlap(\perp, A)$, $merge(overlap(A, B), A)$, $merge(A, E)$. There are two kinds of graphical objects, basic graphical objects and emergent graphical objects. All the constants in the set of function symbols (*e.g.* points $a$, $b$, circles $A$, $B$) represent basic graphical objects and applying a function symbol to terms forms emergent graphical objects (*e.g.* $overlap(A, B)$ represents an emergent closed curve if $A$ and $B$ are overlapping or a null object otherwise).




### 3.1.4 Properties of graphical objects

The predicate symbols (more precisely, the atomic formulas generated from them) describe the basic properties of and relationships between graphical objects. For example, in the signature in Figure 3.1, the atomic formulas, $in_1(a, A)$ and $overlapping(\perp, A)$, can be generated, which describe properties of graphical objects like *point a is inside circle A*. $overlapping(\perp, A)$ represents that the null graphical object $\perp$ and circle $A$ overlap. In fact, such property (or its negation) cannot be seen in real pictures. This is a side-effect resulting from introducing the constant $\perp$, which we use in order to avoid having partial functions instead of total ones.

Formulae like

$$\forall x, y, z : Circle.\ in_2(x, y) \wedge in_2(y, z) \Rightarrow in_2(x, z)$$

can also be generated to represent the transitivity of the 'inside' relation between circles.

### 3.1.5 How are graphical signatures built?

Definition 3.1.1 only gives the structure of a graphical signature. Following this structure one may build any kind of graphical signature. Although the notion of graphical signature is purely syntactic, the choice of certain graphical sorts, function symbols and predicate symbols in one's graphical signature must be based on semantic considerations. In practice, a graphical signature is built as consisting of all of the possible sorts, function symbols and predicate symbols which one may wish to use for a certain application. Therefore, once a graphical signature is selected, the graphical objects and their properties which are of interest are those expressible in the signature. For instance, if one chooses the graphical signature in Figure 3.1, one would not be interested in talking about triangles, although they may be regarded as arbitrary graphs of type *Graph*. If one builds an Euler inspired picture description language for solving syllogisms, closed curves and various operations and predicates over closed curves (*e.g.* overlapping, merge) will be included in the graphical signature. However, if the application is about diagram chasing [2] (see the system GROVER [53]) functions and predicates related to circles do not need to be included in the graphical signature.

**Notation** Let $\Sigma$ be a signature. Then $T(\Sigma)$, $F(\Sigma)$ and $At(\Sigma)$ will be used to denote the sets of terms, formulas and atomic formulas in $\Sigma$, respectively.

## 3.2 Graphical inference: an axiomatic characterisation

A graphical signature presents the syntax of the picture description language. In the above explanation of the picture description language, we pretended that there was a 'common-sense' understanding of the graphical sorts, operations and predicates. In fact, the real meanings of the symbols and expressions in the language are determined by the associated engine of graphical inference.

Graphical inference is used to compute the graphical objects formed by graphical operations such as *overlap* and to infer the properties of graphical objects in a picture such as whether a triangle is inside a circle. In practice, graphical inference is realised by geometrical algorithms. In other words, graphical operations and predicates are implemented by programs which give an (operational) semantics to graphical expressions in the language. A theoretical characterisation of graphical inference may be given in different ways, one of which is to give an axiomatic semantics, that is to characterise by a logical theory the general properties of all pictures. This latter approach is suitable for studying how to give pictures semantics in visual communication, and is adopted here.

More specifically, I assume that graphical inference is axiomatizable by a logical theory over the graphical signature, called *the graphical theory* of the picture description language. Let $\Sigma$ be the graphical signature of a picture description language. Then the graphical theory $\mathcal{T}$ is a set of logical formulas over $\Sigma$ which is consistent and closed under the consequence relation of the underlying logical system and characterises graphical inference.

Formally, a graphical theory is just a consistent logical theory. However, the intention here is that the graphical theory $\mathcal{T}$ gives an adequate axiomatic characterisation of the general properties of pictures and graphical inference (the behaviour of graphical algorithms in particular). Being closed under logical inference, a graphical theory is necessarily infinite, but it usually (and always in practice) has finite presentations.[2] It should also be emphasized that the graphical theory contains *only* the general properties that all pictures (intended models) share, but *not* the specific properties. For example, the presentation of the graphical theory over the graphical signature in Figure 3.1 would contain (among others) the formulas listed in Figure 3.2, where it is assumed that the variables are universally quantified over appropriate sorts. But, the theory does not contain any of the atomic facts such as $in_1(a, A)$.

It is useful to emphasize that I think of the graphical theory as fixed once and for all. For a given picture description language we can restrict the graphical theory to the signature of the language. Also it is possible to allow for preferred modes of graphical functions and predicates (*e.g.* take *in* between circles as *reflexive* or not).

[2]It is an interesting topic to study what kind of presentation of graphical theory is suitable for implementation of a system of visual communication. Some related research develops suitable graphical theories for specific applications, *e.g.* the study of Venn Diagrams [59] and the discussion of Euler circles and syllogisms [56].

$$
\mathcal{T} \quad
\begin{array}{rcl}
in_1(x,y) & \Rightarrow & \neg(outside(x,y)) \\
in_2(x,y) & \Rightarrow & overlapping(x,y) \\
overlapping(x,y) & \Rightarrow & overlapping(y,x) \\
in_1(x,y) \wedge in_2(y,z) & \Rightarrow & in_1(x,z) \\
\ldots\ldots & &
\end{array}
$$

Figure 3.2: Example formulas in a graphical theory.

The graphical theory intuitively expresses the geometrical truth about the graphical screen.

In summary, a picture description language $\mathcal{L}$ is completely presented by a graphical signature together with the graphical theory that characterises graphical inference. From now on, I shall use the following notation:

**Notation** Let $\mathcal{L}$ be a picture description language. Then, $\Sigma(\mathcal{L})$ and $\mathcal{T}(\mathcal{L})$ will be used to denote the graphical signature and the graphical theory of $\mathcal{L}$, respectively; that is, $\mathcal{L} = (\Sigma(\mathcal{L}), \mathcal{T}(\mathcal{L}))$.

# 4

# The geometrical characterisation of pictures

Pictures are expressed by terms in our picture description language and can be viewed as models of the graphical theory of the language. In order to describe pictures, I shall introduce a notion of a *situation* [1], based on which I give the geometrical characterisation needed to express *what one can see in a picture.*

There are two aspects that must be characterised to describe the information that a picture carries: (a) the graphical objects that occur in the picture and (b) the properties those graphical objects have. For the first, one has to delimit those graphical objects that occur in the picture and exclude those that do not appear in it. To meet this demand, I introduce the notion of a *subsignature.*

**4.0.1.** DEFINITION. (Subsignature) *A signature* $\Sigma = (\mathcal{S}, \leq, \mathcal{F}, \mathcal{P})$ *is a* subsignature *of another,* $\Sigma' = (\mathcal{S}', \leq', \mathcal{F}', \mathcal{P}')$*, if and only if:*

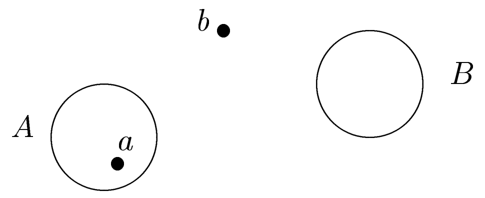
1. $\mathcal{S}$ *is a subset of* $\mathcal{S}'$ *and contains the largest and smallest sorts* $Graph$ *and* $Null$*;*
2. $\leq$ *is the restriction of* $\leq'$ *over* $\mathcal{S}$*;*
3. $\mathcal{F}$ *and* $\mathcal{P}$ *are subsets of* $\mathcal{F}'$ *and* $\mathcal{P}'$*, respectively.*

*I shall write* $\Sigma \sqsubseteq \Sigma'$ *when* $\Sigma$ *is a subsignature of* $\Sigma'$*.*

The graphical objects that occur in a picture can be presented by the terms generated from a subsignature of the graphical signature. For example, consider the picture shown in Figure 4.1(1). The graphical objects in it correspond to the subsignature in Figure 4.1(2) (of the graphical signature in Figure 3.1).

The properties of the graphical objects in a picture are to be characterised axiomatically. I introduce a notion of 'situation' to characterise several important ideas based on which the geometrical semantics of pictures is given. A situation is a maximal set of formulas expressing basic properties over a subsignature which are consistent with respect to the graphical theory (graphical inference). The notions of

[1]'Situation' as used here is unrelated to situation theory as developed by Jon Barwise and John Perry [4].




(1) A picture $\mathcal{G}_0$ whose geometrical semantics is given by $(\Sigma_{\mathcal{G}_0}, \mathcal{A}_{\mathcal{G}_0})$.

| | | |
|---|---|---|
| $\Sigma_{\mathcal{G}_0}$ | $\mathcal{S}$ | $Point, \quad Circle, \quad Graph, \quad Null$ |
| | $\leq$ | $Null \leq s \leq Graph$ and $s \leq s$, for all $s \in \mathcal{S}$. |
| | $\mathcal{F}$ | $a, \; b \; : \; Point; \quad A, \; B \; : \; Circle; \quad \perp \; : \; Null.$ |
| | $\mathcal{P}$ | $in_1, \; outside \; : \; Point \times Circle;$<br>$in_2, \; overlapping \; : \; Circle \times Circle.$ |

(2) The subsignature for picture $\mathcal{G}_0$ in (1).

| | |
|---|---|
| $\mathcal{S}_{\mathcal{G}_0}$ | $in_1(a, A), \; \neg outside(a, A), \; outside(b, A), \; \neg in_1(b, A),$<br>$\neg in_1(a, B), \; outside(a, B), \; \neg in_1(b, B), \; outside(b, B),$<br>$\neg in_2(A, B), \; \neg in_2(B, A), \; \neg overlapping(A, B), in_1(\perp, B),$...... |

(3) A situation for the picture $\mathcal{G}_0$ in (1).

| | |
|---|---|
| $\mathcal{A}_{\mathcal{G}_0}$ | $in_1(a, A), \; \neg outside(a, A), \; outside(b, A), \; \neg in_1(b, A),$<br>$\neg in_1(a, B), \; outside(a, B), \; \neg in_1(b, B), \; outside(b, B),$<br>$\neg in_2(A, B), \; \neg in_2(B, A), \; \neg overlapping(A, B)$ |

(4) The set of basic sentences for picture $\mathcal{G}_0$ in (1).

Figure 4.1: An example picture and its geometrical semantics

basic fact and of situation are given in the following definitions, followed by explanations why the latter gives an adequate characterisation of pictorial properties.

**4.0.2.** DEFINITION. (basic facts) *Let* $\Sigma$ *be an order-sorted signature. A* basic fact *over* $\Sigma$ *is either an atomic sentence over* $\Sigma$ *or the negation of an atomic sentence over* $\Sigma$*. The set of basic facts over* $\Sigma$ *is denoted by* $\mathcal{B}(\Sigma)$*.*

$$\mathcal{B}(\Sigma) = \{P, \neg P \mid P \in F(\Sigma) \textit{ is an atomic sentence}\}.$$

**4.0.3.** DEFINITION. (situation) *Let* $\mathcal{L}$ *be a picture description language and* $\Sigma \sqsubseteq \Sigma(\mathcal{L})$ *be a subsignature of the graphical signature of* $\mathcal{L}$*. A* situation $S$ *over* $\Sigma$ *is a maximal set of basic facts over* $\Sigma$ *consistent with respect to the graphical theory* $\mathcal{T}(\mathcal{L})$*, that is,* $S$ *is a set of formulas over* $\Sigma$ *such that*

1. $S \subseteq \mathcal{B}(\Sigma)$;
2. $S$ *is consistent with respect to* $\mathcal{T}(\mathcal{L})$*, i.e.* $\mathcal{T}(\mathcal{L}) \cup S \not\vdash false$*; and*
3. $S$ *is maximal, i.e. for any set* $M \subseteq \mathcal{B}(\Sigma)$ *of basic facts over* $\Sigma$*,* $M = S$ *if* $S \subseteq M$ *and* $M$ *is consistent with respect to* $\mathcal{T}(\mathcal{L})$*.*

A situation over a subsignature contains the properties of a picture whose graphical objects are characterised by the subsignature. For instance, the situation in Figure 4.1(3) contains the properties that the picture shown in Figure 4.1(1) has.

Without explanation, it might be not very clear how a situation is formed. In Section 3.2, I emphasised that a graphical theory does not contain any of the atomic facts. This also means that a graphical theory does not contain any concrete geometrical information about graphical objects, *e.g.* the position of point $a$, the center and the radius of circle $A$. Adding concrete geometrical information about certain graphical objects to the graphical theory deduces a set of atomic facts which forms the situation.

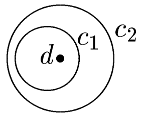
The notion of situation captures three important aspects of the properties of those graphical objects occurring in a picture: basic facts, consistency and maximality.

## 4.1 Basic facts

The first condition, basic facts, captures the following important observation:

- *The properties that can be seen in a picture are the basic ones (expressed by the basic facts).*

A picture is only a concrete model of the graphical theory. In a particular picture, one can only see the basic properties of the graphical objects, but *not* those general properties of graphical inference which are characterised by the graphical theory. For example, consider the following picture.



One can see that the dot is inside both of the circles and the smaller circle is inside the larger one. However, one can *not* see in the above picture the general properties such as 'for any dot $x$ and any circles $y$ and $z$, $x$ is inside $z$ if $x$ is inside $y$ and $y$ is inside $z$', which is expressed by the following logical formula (*c.f.* Figure 3.2):

$$\forall x : Dot\ \forall y, z : Circle.\ in_1(x, y) \wedge in_2(y, z) \Rightarrow in_1(x, z).$$

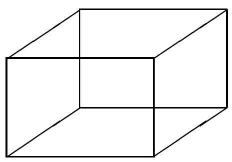
Some people may argue that they can 'see' such general properties in a picture. In our view, however, such general properties are *not* what they *see* in a picture, but rather what they *realise* by associating them with the basic facts they see in the picture, which are particular instances of the general properties. Such a possibility of associating the basic facts seen in particular pictures with some general properties by generalisation is one of the interesting and important aspects of visual communication, which allows people to *illustrate* (but not to prove) some general principles such as general algebraic laws about set operations using particular pictures such as Venn diagrams (see Section 10.2 for such a use of pictures).

A remark here is necessary. I have only included the basic facts in the geometrical semantics of a picture, but excluded those properties about the graphical objects in a picture expressed by other composite sentences such as $in_1(d, c_1) \wedge in_2(c_1, c_2) \Rightarrow in_1(d, c_2)$. It is arguable whether the logical consequences of the basic facts can be seen in a picture. It may be reasonable to say that the conjunction of depictable basic facts can be seen in a picture. I think that this is already covered by the set of depictable basic properties. However, it would be more questionable whether one can see a disjunctive fact (and similarly, an implication). So, I think that the characterisation via the basic facts is adequate and does not lose generality.

## 4.2 Consistency

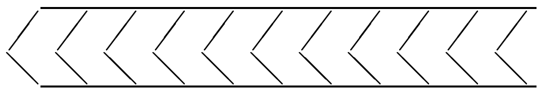
The *consistency* condition for a situation reflects the fact that pictures cannot convey contradictory graphical information. In other words, for a basic property represented by an atomic sentence $P$ about some objects in a picture (say, a point is inside a circle), it is impossible to see both $P$ and $\neg P$ in the picture. E.g. it would be absurd to see a point both as inside and as outside a circle.

Some may hold that there are pictures which convey contradictory information. For instance, a reasonable graphical theory should reflect the fact that in a 3D space, for any plane there is at most only one of its sides which can be seen. However, if we consider the following picture



we may be confused by sometimes seeing the top sides of the horizontal planes and other time seeing their reverse sides. Does this mean that pictures may convey contradictory information? In my view, such a contradiction does not directly come from the basic facts conveyed by a picture but is formed by another process. For the above picture, the inconsistent observation results from moving the apparent horizon above the cube at one time and below it at another time. That means that the cube has been viewed twice and each time in a different reference system which should be represented by different graphical theories. It is clear that in one reference system we can only see either the top or its reverse side.

One distinction between inconsistency and optical illusion can be mentioned here. A picture may cause an optical illusion but will never convey inconsistent information. Optical illusions can be described as: a property $P$ in a picture is not recognised as such but rather as its negation. But this does not mean that both $P$ and $\neg P$ exist in the picture. For example, consider the following picture.

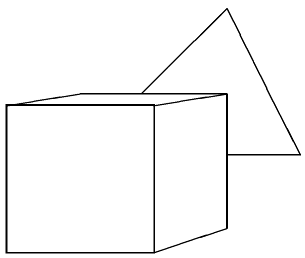


The optical illusion in this picture results from the set of arrows which causes us to see a pair of non-parallel lines which are in fact parallel according to the geometrical definition. However, this only means that the picture shows one property which *we* read as its negation rather than as both the property and its negation.

## 4.3 Maximality

Finally, pictures do give us *maximal* information about the properties of the graphical objects, as characterised in the third condition in the definition of a situation. In other words, for a basic property represented by an atomic sentence $P$ about some objects in a picture, either $P$ or $\neg P$ can be seen in the picture. This consideration leads to the maximality condition for situations.

The distinction between maximality and the perceptual Gestalt that a picture may have can be mentioned here. Consider the following picture.



Usually, people view the shape connected with the cube as a triangle behind the cube, though it is just a partial triangle. One may be confused by our claim of maximality if one considers such a perceptual Gestalt, saying that pictures do not always tell us maximal information: in this example, the shape seen by us is a triangle but in the picture it is only a partial triangle. However, in our view, the complete triangle is *not* what we *see* in the picture, but what we *guess*. What we see in the picture is indeed a partial triangle. The above perceptual Gestalt can be described by introducing a special picture description language whose signature includes the following subsorts of the sort *Triangle*:

- *Complete-triangle* stands for the class of complete triangles.
- *Gestalt-triangle* stands for the class of incomplete triangles which are viewed as triangles by perceptual Gestalt.

$x$ is a Gestalt triangle if (1) $x$ is a path of three lines, (2) the two end-points of $x$ connect with an object $y$, and (3) the meeting point which is obtained by extending the two end-lines of $x$ is inside $y$. Thus, we describe triangles in a picture by looking not only for complete triangles but also for the Gestalt triangles. However, the above may not be a sound way to deal with such a perceptual Gestalt, because there are many different situations causing the existence of a partial triangle and it may be very difficult to have a general definition of what an incomplete triangle is. Here, I only point out the difference between perceptual Gestalt and the maximality condition for situations.

Maximality, together with the consistency condition, implies the completeness of basic facts of a situation. Also note that a situation $S$ over $\Sigma$ is necessarily closed with respect to the graphical theory $\mathcal{T}(\mathcal{L})$ of the picture description language in the sense that $\{ P \in \mathcal{B}(\Sigma) \mid \mathcal{T}(\mathcal{L}) \cup S \vdash P \} \subseteq S$, that is, *every basic property of the graphical objects in a picture is included in a situation provided that it is a logical consequence of the graphical theory $\mathcal{T}(\mathcal{L})$ and the basic facts in the situation.*[2]

## 4.4 Invisible properties

The above notion of situation has essentially captured the geometrical semantics of pictures, except for a minor point to be discussed connecting the 'null' graphical objects represented by the constant $\perp$. Because of this, a picture may correspond to more than one situation because, for any invisible property involving null objects such as $in_1(\perp, A)$, there are (at least) two different situations corresponding to the same picture, one containing $in_1(\perp, A)$ and the other its negation $\neg in_1(\perp, A)$. Since the properties of any null graphical object (say, $in_1(\perp, A)$) can not be seen in a picture, one is not interested in these properties. To deal with this, we can define an observational equivalence between the situations over the same subsignature $\Sigma$ by taking the sort *Null* as a non-observable sort (*c.f.* the study of observational equivalence in program specification; see, for example, [54].) Then, a picture $\mathcal{G}$ corresponds to an equivalence class $[S_{\mathcal{G}}]$ of the situations corresponding to the picture $\mathcal{G}$.

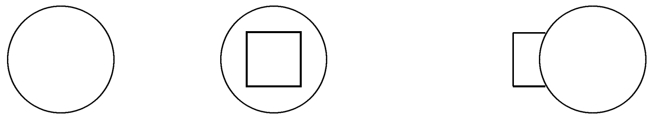
[2] This can be simply proved as follows. Any $P \in \mathcal{B}(\Sigma)$ being a logical consequence of $\mathcal{T}(\mathcal{L}) \cup S$, $\mathcal{T}(\mathcal{L}) \cup S \cup \{P\}$ is logically consistent, by the consistency of the underlying logical system. Therefore, $S \cup \{P\}$ is consistent with respect to $\mathcal{T}(\mathcal{L})$, and hence $P \in S$ by the maximality of the situation $S$.

Taking into account the above consideration, the geometrical characterisation of a picture $\mathcal{G}$, whose graphical objects are characterised by a subsignature $\Sigma_{\mathcal{G}}$, is given by the following set of basic facts:

$$\mathcal{A}_{\mathcal{G}} = \bigcap [S_{\mathcal{G}}],$$

where the invisible properties about the null graphical objects are excluded by taking the intersection of the equivalent situations subject to the observational equivalence (taking *Null* as a non-observable sort).

When we build a picture description language, the 'null' graphical object ($\perp$) should be treated in a careful way. Besides some obvious ways in which the 'null' graphical object is made (*e.g.* overlap two separated circles), overlapping a filled area and an object may also make the 'null' object. How to characterise such an overlapping in the graphical theory should be considered carefully so that the geometrical characterisation of pictures meets the requirements of visual communication. It is my opinion that the graphical objects used in visual communication should always be seen by the users. It does not make sense from a visual point of view to think that there is a square inside a filled circle (as in picture (1)), unless it can be seen, as in picture (2), or it can be partially seen and can be recognised as a square according to the perceptual Gestalt, as in picture (3).



(1) (2) (3)

Therefore, I suggest that the drawing system make every drawing visible (see the above picture (2) as an example) in a visual communication supporting environment. Otherwise, the graphical theory should give correct axioms so that the geometrical characterisation of a picture correctly reflects what one can see in the picture. For example, if one object covers another, the later should be deleted from (or marked as a null object in) the subsignature.

To summarise: a picture is given a formal geometrical characterisation by capturing its graphical objects by means of a subsignature and characterising its properties axiomatically as above.

**4.4.1.** Definition. (the geometrical characterisation of pictures) *Let $\mathcal{L}$ be a picture description language and $\mathcal{G}$ a picture. The geometrical characterisation of $\mathcal{G}$ in $\mathcal{L}$ is given as a pair:*

$$[\![\mathcal{G}]\!] = (\Sigma_{\mathcal{G}}, \mathcal{A}_{\mathcal{G}})$$

*where $\Sigma_{\mathcal{G}}$ is the subsignature of $\Sigma(\mathcal{L})$ characterising the graphical objects of $\mathcal{G}$, and $\mathcal{A}_{\mathcal{G}}$ is the set of basic facts as defined above.*

For example, the geometrical characterisation of the picture in Figure 4.1(1) is given by the subsignature $\Sigma_{\mathcal{G}_0}$ in Figure 4.1(2) and the set of depictable basic properties $\mathcal{A}_{\mathcal{G}_0}$ in Figure 4.1(4). It might be helpful to point out the relations between

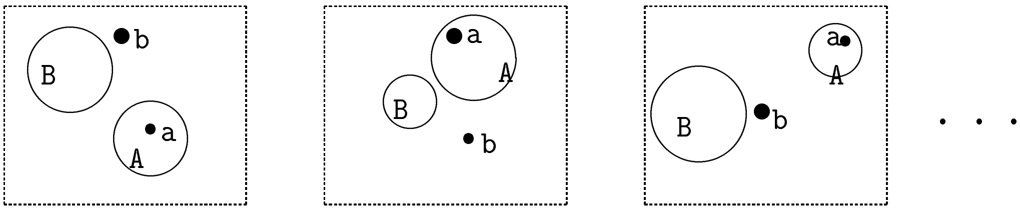



Figure 4.2: Example pictures whose geometrical characterisation can also be given by $\Sigma_{\mathcal{G}_O}$ and $\mathcal{A}_{\mathcal{G}_O}$ in Figure 6.

the geometrical characterisation of pictures and real pictures. Given a picture and a picture description language, the geometrical characterisation of the picture is completely fixed. Not for every picture description language however it holds that the characterisation completely determines the picture. For example, if the graphical signature is the one in Figure 3.1, $\mathcal{A}_{\mathcal{G}_O}$ in Figure 4.1(4) describes not only the picture given in Figure 4.1(1), but also other pictures, as in Figure 4.2. Because, the signature in Figure 3.1 is not capable to distinguish them, it is only used for the applications where position and size is not important.

# 5

# Conclusion of Part I

In this part, I have studied how to formally describe pictures. Figure 5.1 gives an illustration of the relations between picture description languages, geometrical characterisations of pictures and pictures themselves.

A picture description language $\mathcal{L}_i$ consists of a graphical signature $\Sigma(\mathcal{L}_i)$ and a graphical theory $\mathcal{T}(\mathcal{L}_i)$. Different signatures and theories form different languages. In practice, we build a particular picture description language according to what kind of graphical objects and properties we wish to use.

The expressions of $\mathcal{L}_i$ are generated by the graphical signature $\Sigma(\mathcal{L}_i)$. There are *sorts*, *function* and *predicate* symbols in $\Sigma(\mathcal{L}_i)$. There is a partial order relation over the sorts.

The geometrical meaning of the sorts, functions and predicates in $\Sigma(\mathcal{L}_i)$ are characterised by the graphical theory $\mathcal{T}(\mathcal{L}_i)$. Note that $\mathcal{T}(\mathcal{L}_i)$ only contains general properties of graphical symbols. For example, $\mathcal{T}(\mathcal{L}_i)$ may contain the property: if circle $X$ is inside circle $Y$ then $X$ must not be outside $Y$. But whether or not a particular circle is inside (or outside) another (say $in(C_1, C_2)$ or $out(C_1, C_2)$) is not included in $\mathcal{T}(\mathcal{L}_i)$. Therefore, $T(\mathcal{L}_i)$ cannot be directly used to describe a picture.

A picture $P_j$ is described in $\mathcal{L}_i$ as $(\Sigma_{P_j}, \mathcal{A}_{P_j})$. $\Sigma_{P_j}$ is a subsignature of $\Sigma(\mathcal{L}_i)$ and is used to generate the description of the graphical objects in $P_j$. $\mathcal{A}_{P_j}$ is a set of basic facts which describe the properties of and relations between the graphical objects in $P_j$. $\mathcal{A}_{P_j}$ is an intersection of a set of equivalent situations $[S]$. There may be some basic facts which are invisible like $in(\perp, C)$ in the situations, but there are no such invisible facts in $\mathcal{A}_{P_j}$. A situation in the class $[S]$ is a maximal set of basic facts over $\Sigma_{P_j}$ which is consistent with respect to $\mathcal{T}(\mathcal{L}_i)$. $P_j$ can be characterised in many different ways corresponding to different picture description languages. However, there is only one description of $P_j$ in a given picture description language.

The abstract characterisation of picture description, studied in this part, is the most essential step towards an understanding of the interpretation of pictures and the use of pictures to represent real world problems in visual communication.

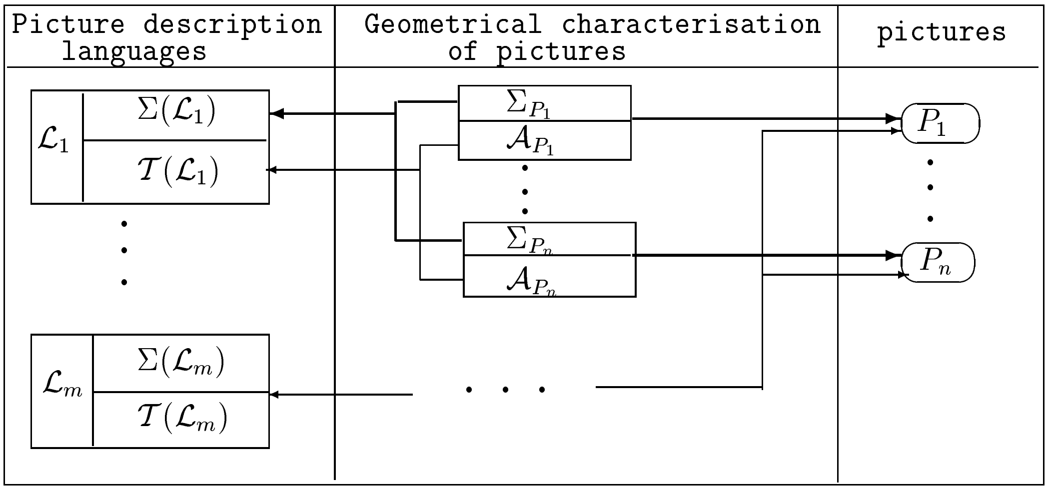



Figure 5.1: An illustration of the relations between picture description languages, pictures and their geometrical characterisation.

# Part II

# Interpretation

# 6 Introduction to Part II

In Part I, I studied how to give pictures a geometrical characterisation regardless of their use. There is a variety of meanings in the use of pictures in visual communication and the meanings can either be assigned to pictures through an explicit interpretation or be embedded into the construction of a more complex structure of pictures (*i.e.* picture specifications). In this part, I study how to give pictures semantics by interpretation.

Consider the following diagrams which are often used to illustrate the validity of the distributive law of the set operations $\cup$ and $\cap$:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

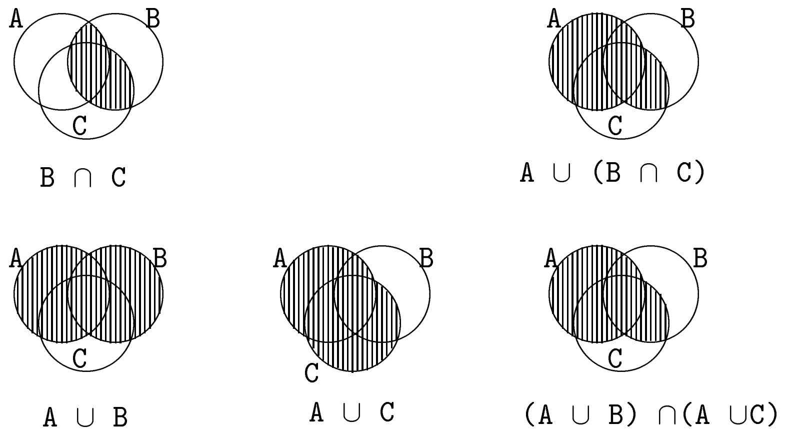



Figure 6.1: The distributive law of $\cup$ and $\cap$ is illustrated by means of Venn Diagrams.

It is clear that these diagrams help us to realise the validity of the distributive law because they are interpreted as the set operations. I.e. the closed curves in the diagrams represent sets, two closed curves' overlapping represents $\cap$ and the merging of two closed curves represents $\cup$. It is the interpretations, not the diagrams themselves, that associate the meanings of sets and set operations with the diagrams. This suggests a natural model of the use of pictures in visual communication where graphical objects and their spatial properties in a picture are interpreted as objects and relations in the application domain. Then the subject matter in the application domain can be investigated by means of manipulation of pictures.

It is often said that a picture is worth a thousand words, but it seems hard to say which picture is worth which thousand words. In fact, if this could be figured out, an universal graphical representation language could have been defined and we would study very different questions in the research of visual communication. However, according to our experience of using pictures in communication, a picture can be worth many different thousands of words. It depends on how we assign meanings to pictures. I.e. it depends on the interpretations. This interpretation makes the picture worth this thousand words and that interpretation makes it that thousand. Furthermore, if an interpretation is not proper, the picture may be worth nothing. The role of the interpretation is obviously a key factor in understanding the semantics of pictures and the process of visual communication.

Picture semantics has been studied by many other researchers. Much work in this field involves the study of the semantics on the basis of a preexistant interpretation. For instance, the meanings of Euler's circles in their use to solve syllogisms [56], the research on Venn diagrams [59], the logical study of Higraphs (which are a combination of Venn diagrams, Euler circles and a diagrammatic mechanism for representing relations) [27], and the work in the field of visual languages (see the collection in the Journal of Visual Language and Computing [8] for example). These studies are very important for those kinds of visual communication in which the graphical representation of the application domain has been well established (*e.g.* Venn diagrams and set theory, Euler's circles and syllogisms, various charts and binary relations, flow charts and programs, and so on). However, there are few studies aimed in the direction that pictures are assigned meanings by explicit interpretation. In his work [52], Pineda gave a formal framework which allows the expression of certain properties of an interpretation, however, the notion of an interpretation as such is not systemantically addressed: neither syntactic correctness nor semantic appropriateness of an interpretation can be defined. The study of interpretation is important for providing a computational environment to support general visual communication where pictures do not have fixed meanings but only pictorial information. It is the interpretation which relates pictorial information with various issues in application domains. This means that without interpretations pictures have no meanings but only pictorial information, or in other words, pictures have a variety of meanings in their uses in visual communication.

This *variety* of meanings in use seems to be essential in understanding the general forms of visual communication. The 'variety of meanings' does *not* simply refer to the fact that the graphical entities in a picture may be assigned (or understood) to have many different meanings. In the use of pictures in communication based on one's *intended* interpretation of the graphical entities, the information that a picture carries is subject to a certain kind of *abstraction*. One often emphasizes some of the properties of the graphical objects, but at the same time intentionally ignores (or abstracts away from) the other properties. For instance, using Venn diagrams to illustrate the algebraic laws about sets, one ignores irrelevant properties of the diagrams such as the sizes of the circles. In that case, one does not (and should not) even think that the fact that two circles are overlapping implies that the represented sets have common elements. Thus, when using pictures in communication, one does not and should not regard all of the information one can see in the pictures as being relevant to the subject matter. We understand this phenomenon by saying that the intended interpretation should concern only the relevant information and ignore the irrelevant. It is this possibility of abstraction in interpretation that enables pictures to be useful in visual communication.

In visual communication, usually, it is not just a picture as a whole that is assigned a meaning; more importantly, the component graphical entities such as subpictures, emergent graphical objects and their possible properties, must be regarded as having meanings so that they contribute information in communication. For example, using a Venn diagram to illustrate the distributive law of set operations $\cup$ and $\cap$, the emergent graphical object formed by the overlapping of two circles represents $B \cap C$, and what is obtained by merging this emergent graphical object with another circle represents $A \cup (B \cap C)$.

In summary, the notion of interpretation should reflect the following three considerations.

- The same class of graphical objects and properties may be interpreted and hence used to represent different objects and properties in different applications.
- The notion of interpretation should allow the flexibility of reflecting the uses of pictures where the communication concerns and concentrates on only a part of the properties of the graphical objects in a picture and ignores the rest.
- What can be interpreted (used) in communication includes not only the simple graphical components of a picture and their properties, but also the more complicated components such as emergent graphical objects and their properties.

In the rest of this part, I first give a formal investigation of interpretation and visual communication via interpretation, and then study how pictures are correctly used in communication by discussing various properties of interpretations which determine whether or not the meaning borne by the picture under the interpretation reflects the user's intention. On the basis of the theoretical framework, I discuss its applications. A computer-based visual communication environment is presented based on the framework. Three kinds of visual communication are investigated. They are: illustrative visual communication, demonstrative visual communication and reasoning with diagrammatical representations. I present the system GAR (Graphics-Assisted Reasoning), designed and implemented on the basis of the formal framework, to support demonstrative visual communication. I also present a theoretical framework for supporting reasoning with diagrammatical representations.

# 7

# Interpretations

In this section, a formal notion of interpretation which satisfies the requirements discussed in the last section is presented.

Intuitively, an interpretation maps graphical entities to entities in an application domain in a coherent way so that it respects the necessary relationships between objects. The simplest way to formalise such a notion of interpretation in the framework from Part I is to consider the notion of signature morphisms (renaming maps),[1] which map sorts to sorts, function symbols to function symbols and predicate symbols to predicate symbols. However, in practice, it is sometimes very helpful to use a single graphical function/predicate symbol to represent a composed function (term) or predicate (propositional function). For example, we may need this generality to allow a binary graphical operation with principal type $s \times s \to s$ to be used to represent a composed operation $\lambda xy.(\overline{\overline{x} \cap \overline{y}})$ in set theory. This can be done by allowing the interpretation of a function symbol (predicate symbol) into a 'generalised term (formula)'. Roughly speaking, a generalised term (formula) is a term where $\lambda$-notation is used to represent functions (propositional functions). See Appendix A for the definition. A generalised term may have an 'arrow sort' (*e.g.* $\lambda xy.\overline{\overline{x} \cap \overline{y}} \ : \ set \times set \to set$) and a generalised formula may be a propositional function over some domain.

**Notation** Let $\Sigma$ be a signature. Then, $GT(\Sigma)$ and $GF(\Sigma)$ denote the sets of generalised terms and generalised formulas, respectively.

**7.0.2.** Definition. (Interpretation) *Let $\mathcal{L}$ be a picture description language and $\Sigma = (\mathcal{S}, \leq, \mathcal{F}, \mathcal{P})$ an order-sorted signature (of the application language). An* inter-

[1] It may be the case that reasoning in an application domain is based on a logical system which is different from the logical system for graphical inference. If that is the case, to interpret graphical inference in the application domain, one has to consider a map from the logic for graphical inference to that in the application domain. See, for example, Meseguer's discussion in [45] on general logics and interpretations between them. Here, for simplicity, I consider the case where the same logical system is used. This is adequate for current purpose.

| | | |
|---|---|---|
| $\Sigma_{\mathcal{I}}$ | $\mathcal{S}_{\mathcal{I}}$ | $Point,\ Circle,\ Graph,\ Null$ |
| | $\leq_{\mathcal{I}}$ | $Null \leq s \leq Graph$ and $s \leq s$, for all $s \in \mathcal{S}$. |
| | $\mathcal{F}_{\mathcal{I}}$ | $a, b\ :\ Point;\ \ A, B\ :\ Circle;\ \ \perp\ :\ Null.$ |
| | $\mathcal{P}_{\mathcal{I}}$ | $in_1\ :\ Point \times Circle;$ |
| $\Sigma$ | $\mathcal{S}$ | $Elem,\ Set,\ \ldots\ \ldots$ |
| | $\leq$ | $\ldots\ \ldots$ |
| | $\mathcal{F}$ | $a, b, c, d\ :\ Elem;\ \ A, B, C, D\ :\ Set;\ \ldots\ \ldots$ |
| | $\mathcal{P}$ | $\in\ :\ Elem \times Set;\ \ \subseteq\ :\ Set \times Set;\ \ldots\ \ldots$ |
| $\mathcal{I}$ | $\mathcal{I}_{\mathcal{S}}$ | $Point \mapsto Elem,\ \ Circle \mapsto Set,\ \ldots\ \ldots$ |
| | $\mathcal{I}_{\mathcal{F}}$ | $A \mapsto A,\ \ B \mapsto B,\ \ a \mapsto a,\ \ b \mapsto b,$ |
| | $\mathcal{I}_{\mathcal{P}}$ | $in_1 \mapsto \in.$ |

Figure 7.1: An interpretation.

pretation $\mathcal{I}$ *from the picture description language to* $\Sigma$

$$\mathcal{I}\ :\ \Sigma(\mathcal{L}) \rightharpoonup \Sigma$$

*is a quadruple* $(\Sigma_{\mathcal{I}}, \mathcal{I}_{\mathcal{S}}, \mathcal{I}_{\mathcal{F}}, \mathcal{I}_{\mathcal{P}})$, *where*

- $\Sigma_{\mathcal{I}} = (\mathcal{S}_{\mathcal{I}}, \leq_{\mathcal{I}}, \mathcal{F}_{\mathcal{I}}, \mathcal{P}_{\mathcal{I}})$ *is a subsignature of the graphical signature* $\Sigma(\mathcal{L})$*, called the* signature of $\mathcal{I}$*;*
- $\mathcal{I}_{\mathcal{S}}\ :\ \mathcal{S}_{\mathcal{I}} \rightarrow \mathcal{S}$ *is a function which preserves the partial order between sorts and the largest and smallest sorts;*
- $\mathcal{I}_{\mathcal{F}}\ :\ \mathcal{F}_{\mathcal{I}} \rightarrow GT(\Sigma)$ *is a type-preserving function; and*
- $\mathcal{I}_{\mathcal{P}}\ :\ \mathcal{P}_{\mathcal{I}} \rightarrow GF(\Sigma)$ *is a domain-preserving function.*

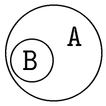
Figure 7.1 describes an example subsignature $\Sigma_{\mathcal{I}}$ of the graphical signature $\Sigma_0$ in Figure 3.1, a signature $\Sigma$ concerning sets and an interpretation $\mathcal{I}\colon \Sigma_0 \rightharpoonup \Sigma$ whose signature is $\Sigma_{\mathcal{I}}$.

## 7.1 Partial mapping ($\Sigma_{\mathcal{I}}$)

The above notion of interpretation from a picture description language to an application domain reflects the requirements I have described in the last section. In particular, an interpretation is a *partial* map over the graphical signature in the sense that it is defined only over a part of the graphical signature (the signature of the interpretation). This reflects the second aspect of use of pictures and allows the user to specify an interpretation that allows him/her to concentrate on certain kinds of graphical objects and properties but ignore the others. In other words, choosing an interpretation over a certain subsignature can provide the necessary control over how pictures are used in communication. This also sheds light on the implementation of a supporting system, where the implemented picture description language is rather rich for a wide range of applications and, for a particular application, only a part of the picture description language is under consideration.

## 7.2 Sorts ($\mathcal{I}_\mathcal{S}$)

Mapping graphical sorts to sorts in an application domain naturally reflects the idea that in visual communication we always use a class of graphical objects to represent a class of objects in an application domain [2]. Note that graphical objects of the same sort cannot be interpreted to represent different sorts of objects by a single interpretation, for to do so would only causes confusion. For example, by mapping some circles into sets, other circles into elements and *inside* into the *subset* relation for sets or the *membership* relation for elements, the following picture confuses us because whether B is an element or a subset of A is not clear.




An interpretation must also respect the subtyping relation (*i.e.* be partial order preserving) which guarantees the accuracy of the use of classes of pictures in representing classes of objects in an application domain. For example, mapping circles to men and closed curves to persons is only valid when the class of men is considered as a subclass of that of persons in the application domain. Some people may wish to use circles to represent men and closed curves to represent cats, and yet deny that this means that all men are cats. Actually, if they are to be consistent when they use closed curves to represent cats, the class of closed curves must already be specified as the class in which circles are excluded. In fact, during visual communication, people use conventions according to their needs (like circles are excluded in the class of closed curves in the above example), no matter they are aware or unaware of these. And in my formal framework, such conventions determine certain picture description languages and interpretations.

## 7.3 Objects and properties ($\mathcal{I}_\mathcal{F}$ and $\mathcal{I}_\mathcal{P}$)

Graphical objects (and their spatial properties) are allowed to be used in visual communication by means of the mapping of graphical function symbols (and graphical predicate symbols). $\mathcal{I}_\mathcal{F}$ ($\mathcal{I}_\mathcal{P}$) must be type (domain) preserving in the sense that the mapping between functions (predicates) must be consistent with the mapping between sorts. More precisely, if $f$ is a function symbol with type $s_1 \times s_2 \times \ldots \times s_n \to s$ then $\mathcal{I}_\mathcal{F}(s_1) \times \mathcal{I}_\mathcal{F}(s_2) \times \ldots \times \mathcal{I}_\mathcal{F}(s_n) \to \mathcal{I}_\mathcal{F}(s)$ must be the type of $\mathcal{I}_\mathcal{F}(f)$ in the application domain. For example, if *Circle* is interpreted as *Set* then all the nullary

[2] Here it is assumed that the underlying graphical signature $\Sigma(\mathcal{L})$ is rich enough to support any choice of picture classes. However, in practice, it may be the case that there is no suitable class of graphical objects in the picture description language which can be used to represent the class of objects in the application domain. This can be solved by introducing a picture specification mechanism (see Part III) which supports users in creating new classes of graphical objects (*i.e.* in adding new sorts into the graphical signature) according to the requirements of the problem solving.

function symbols (*i.e.* the representations of graphical objects) with sort *Circle* must be mapped into objects of sort *Set*. Furthermore, emergent graphical objects are allowed to be used by the mapping of n-ary (n $\geq$ 1) function symbols such as mapping $overlap : Closed\_curve \times Closed\_curve \rightarrow Closed\_curve$ into $\cap : Set \times Set \rightarrow Set$. A function (predicate) symbol can be mapped to a generalised term (formula). For example, if $\mathcal{I}_{\mathcal{S}}(s) = set$ and $union : s \times s \rightarrow s$, we may have $\mathcal{I}_{\mathcal{F}}(union) = \lambda xy.(\overline{\overline{x} \cap \overline{y}})$.

Given an interpretation as defined above, it is extended to the terms and formulas over the signature of $\mathcal{I}$ (*c.f.* [23]). For example,

$$I(merge(A, overlap(B, C))) = \mathcal{I}(merge)(\mathcal{I}(A), \mathcal{I}(overlap)(\mathcal{I}(B), \mathcal{I}(C)))$$

I will use the following notation.

**Notation** Let $\mathcal{I}$ be an interpretation. For any set $M$ of terms or formulas over $\Sigma_{\mathcal{I}}$,

$$\mathcal{I}(M) =_{\text{df}} \{\, \mathcal{I}(m) \mid m \in M \,\}.$$

# 8

# A logical characterisation of visual communication

Based on the geometrical characterisation of pictures studied in Part I and the above notion of interpretation, I am now ready to give visual communication via interpretations a logical characterisation. In other words, I want to make clear how graphical objects and their properties are used in reasoning about an application domain.

Roughly speaking, we can treat pictures, an application domain theory $\mathcal{T}$ and an interpretation $\mathcal{I}$ as a 'reasoning' system (a visual communication reasoning system). Characterising visual communication is to answer the question: *Given a picture $\mathcal{G}$, a domain theory $\mathcal{T}$ and an interpretation $\mathcal{I}$, what is a conclusion in this 'reasoning' system?* From Part I, the picture $\mathcal{G}$ can be geometrically characterised as $[\![\mathcal{G}]\!] = (\Sigma_\mathcal{G}, \mathcal{A}_\mathcal{G})$, where $\mathcal{A}_\mathcal{G}$ is a set of basic facts deduced from $\mathcal{G}$, which characterise what one 'sees' in the picture. However, in visual communication, not all the facts in $\mathcal{A}_\mathcal{G}$ but only those, which we choose to represent the subject matter in the application domain, interest us. Therefore, we only take that part of $\mathcal{A}_\mathcal{G}$ which can be formed from the interpretation signature, *i.e.* $\mathcal{A}_\mathcal{G} \cap F(\Sigma_\mathcal{I})$, into account. Interpreting that part, *i.e.* $AF = \mathcal{I}(\mathcal{A}_\mathcal{G} \cap F(\Sigma_\mathcal{I}))$, we get a set of facts $AF$ in the application domain. Each fact in $AF$ is a conclusion in this 'reasoning' system. Furthermore, taking $AF$ and the domain theory $\mathcal{T}$ as premisses, all the conclusions in the domain reasoning system can also be viewed as conclusions in this visual communication reasoning system.

More precisely, in the following we characterise this by defining a consequence relation which incorporates graphical inference as well as domain reasoning in such a way that the properties deduced from pictures are incorporated in reasoning about the application domains.

**8.0.1.** DEFINITION. (logical characterisation of visual communication)
*Let $\mathcal{L}$ be a picture description language, $\mathcal{T}$ the logical theory over a signature $\Sigma$ of the application language. Then,*

1. *A* state of visual communication *(in $\mathcal{T}$) is a pair*

$$\sigma = (\mathcal{I}, \mathcal{G})$$

*consisting of an interpretation $\mathcal{I}$ from $\mathcal{L}$ to $\Sigma$ whose signature is $\Sigma_{\mathcal{I}}$ and a picture $\mathcal{G}$ whose geometrical characterisation is $[\![\mathcal{G}]\!] = (\Sigma_{\mathcal{G}}, \mathcal{A}_{\mathcal{G}})$.*
2. *The* consequence relation for visual communication $\vdash^{\sigma}$*, determined by a state of visual communication $\sigma$, is defined as follows: for any sentence $A$ over $\Sigma$,*

$$\vdash^{\sigma} A \quad \textit{iff} \quad \mathcal{T} \cup \mathcal{I}(\mathcal{A}_{\mathcal{G}} \cap F(\Sigma_{\mathcal{I}})) \vdash A.$$

Consider the picture and its geometrical characterisation in Figure 4.1 (for easy reading, they have been copied into Figure 8.1) and the interpretation in Figure 7.1, Figure 8.1 lists:

- $F(\Sigma_{\mathcal{I}})$: the set of formulas over the interpretation signature;
- $\mathcal{A}_{\mathcal{G}_O} \cap F(\Sigma_{\mathcal{I}})$: the intersection between the formulas over $\Sigma_{\mathcal{I}}$ ($F(\Sigma_{\mathcal{I}})$) and the basic facts deduced from the picture ($\mathcal{A}_{\mathcal{G}_O}$) and
- $\mathcal{I}(\mathcal{A}_{\mathcal{G}_O} \cap F(\Sigma_{\mathcal{I}}))$: the facts in the application domain deduced from the picture.

According to the above logical characterisation of visual communication, the properties of graphical objects in a picture that are 'carried over' by an interpretation to an application domain are those basic facts in the geometrical characterisation of the picture which are interpreted by the interpretation. Put in another way, those and only those *interpreted* depictable properties of the *interpreted* graphical objects of the picture are used in communication about the application domain. Those graphical objects and properties which are either uninterpreted or those which cannot be seen in the picture do not have any effect in domain reasoning. For instance, $\neg overlapping(A, B)$ in the picture in Figure 8.1 does not play any role in the communication because *overlapping* is not interpreted; although it can be seen in the picture it is ignored. Those properties, like $in_1(\perp, A)$, that are interpreted but cannot be seen in the picture, do not affect the communication either. It may be worth remarking that I do not assume the subsignature corresponding to the picture ($\Sigma_{\mathcal{G}}$) to be a subsignature of the signature of the interpretation ($\Sigma_I$); this allows a picture used in visual communication to contain some graphical objects that are not interpreted and hence not used in the communication. This results in a certain robustness and is useful in practice. Following the above logical characterisation of visual communication, not only those facts deduced from the picture (*i.e.* $\mathcal{I}(\mathcal{A}_{\mathcal{G}} \cap F(\Sigma_{\mathcal{I}}))$) are conclusions but also those sentences concluded from the application domain theory and the facts carried by the picture (*i.e.* $\mathcal{T} \cup \mathcal{I}(\mathcal{A}_{\mathcal{G}} \cap F(\Sigma_{\mathcal{I}}))$). Considering the example in Figure 8.1, if there is a formula: $\forall x : Elm(x \in A \rightarrow x \leq 100)$ in the application domain theory, then $a \leq 100$ is also a conclusion under the consequence relation for the state of the visual communication shown in Figure 8.1. This is useful in some applications, especially, in theorem proving where some general properties in the application domain may be applied to the facts deduced from pictures to obtain new facts. We will see this in Section 9.3 where the Pythagoras theorem is proved using both the facts carried in the picture and the axioms of triangles and squares in the application domain theory.

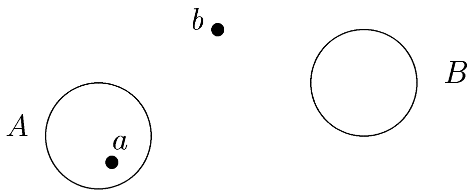
| | |
|---|---|
| $\mathcal{G}_O$ |  |
| $\mathcal{A}_{\mathcal{G}_o}$ | $in_1(a,A),\ \neg outside(a,A),\ outside(b,A),\ \neg in_1(b,A),$ $\neg in_1(a,B),\ outside(a,B),\ \neg in_1(b,B),\ outside(b,B),$ $\neg in_2(A,B),\ \ \neg in_2(B,A),\ \neg overlapping(A,B)$ |
| $F(\Sigma_{\mathcal{I}})$ | $in_1(a,A),\ \neg in_1(a,A),\ in_1(a,B),\ \neg in_1(a,B),$ $in_1(b,A),\ \neg in_1(b,A),\ in_1(b,B),\ \neg in_1(b,B),$ $in_1(\bot,A),\ \neg in_1(\bot,A),\ in_1(\bot,B),\ \neg in_1(\bot,B),$ ...... |
| $\mathcal{A}_{\mathcal{G}_o} \cap F(\Sigma_{\mathcal{I}})$ | $in_1(a,A),\ \neg in_1(a,B),\ \neg in_1(b,A),\ \neg in_1(b,B)$ |
| $\mathcal{I}(\mathcal{A}_{\mathcal{G}_o} \cap F(\Sigma_{\mathcal{I}}))$ | $a \in A,\ a \notin B,\ b \notin A,\ b \notin B$ |

Figure 8.1: Explanation of logical characterisation of visual communication.

# 9 Correct uses of pictures in communication

Interpretations are used in visual communication to capture the different uses (and hence different possible meanings) of graphical entities. A correct use of pictures in communication is reflected in the choice of interpretations, *i.e.* the suitable choice of graphical entities to represent entities in the application domain. The correctness of interpretation has two aspects, syntactic and semantic correctness, as discussed below.

## 9.1 Syntactic correctness of interpretations

By syntactic correctness I mean that the interpretation must satisfy the definition of interpretation (Definition 7.0.2). When using a visual communication supporting system such as GAR to be discussed in Section 10.2, the system requires the user to specify interpretations according to the formal structure of interpretations described in its definition. Whether a given specification is syntactically correct can be checked automatically by the supporting system, which prevents the user from making many mistakes.

An example will show how the syntax-checking for interpretations can be helpful. Consider the graphical operation *overlap* (in Figure 3.1), whose principal type is

$$Circle \times Circle \rightarrow Closed\_curve.$$

With the partial order specified in Figure 3.1, the types of *overlap* must be of the form $A \times B \rightarrow C$ whose domain sorts ($A$ and $B$) are less than or equal to *Circle* and range sort ($C$) is greater than or equal to $Closed\_curve$. Any mapping which is supposed to be an interpretation but which does not satisfy this will be considered as syntactically incorrect. The guarantee of such syntactic correctness of meaning specification is helpful to avoid certain misuses of pictures in communication. For instance, in using Venn diagrams, a beginner might have thought that sets are represented by circles.

Such mistakes can be prevented by syntax-checking. To explain, if one interpreted *Circle* as *Set*, there would be no sensible graphical operation that can be used to represent the set operations such as intersection ($\cap$). The graphical operation *overlap* could not be used to represent $\cap$ since in general overlapping two circles makes a closed curve, rather than a circle. A correct interpretation would be to represent sets by closed curves, which include circles since *Circle* is a subsort of $Closed\_curve$. Then the interpretation mapping *overlap* to $\cap$ is syntactically correct.

## 9.2 Semantic properties of interpretations

The absence of syntactically incorrect interpretations can exclude some mistakes which happen quite often in visual communication. However, even when an interpretation is syntactically correct, it may be semantically incorrect in that it still does not conform with one's intentions. Intuitively, a semantically correct interpretation guarantees that the properties of graphical objects which are depictable in a picture are, under the interpretation, among the properties of the depicted objects in the application domain. The semantic correctness of interpretation is the basis of the correct use of pictures in communication. However, handling semantic correctness is more complex then handling syntactic correctness. Semantic correctness can only be achieved by the user through a good understanding of the application domain as well as the picture description language. The supporting system can provide some help to guide the user to have a semantically correct interpretation, but does not have a general method for checking semantic correctness. Furthermore, for different applications, there are different demands about how much of the application domain should be represented by pictures. For example, if pictures are used to illustrate the set of facts in a data base, the set of facts derived from the representing picture should be exactly the same as the set of facts in the data base. If the application is about design, it may be desirable that facts which do not exist in the application domain can be introduced. After all, design is a process of exploration. Due to such considerations, semantic correctness should not be defined once and for all. Instead, various semantic properties of interpretation should be discussed to explain how pictures can be used correctly in visual communication to reflect the user's intention and hence to obtain adequate (and intended) results. In particular, three properties that an interpretation may have are defined on the basis of the logical characterisation of visual communication.

**9.2.1.** Definition. (Consistency, Soundness and Conservativity)
*Let $\mathcal{L}$ be a picture description language, $\mathcal{T}$ a domain theory over signature $\Sigma$ of the application language, $\mathcal{I} : \Sigma(\mathcal{L}) \rightharpoonup \Sigma$ an interpretation whose signature is $\Sigma_{\mathcal{I}}$, and $\mathcal{G}$ a picture with geometrical semantics $[\![\mathcal{G}]\!] = (\Sigma_{\mathcal{G}}, \mathcal{A}_{\mathcal{G}})$. Then,*

1. *$\mathcal{I}$ is* consistent *with respect to $\mathcal{G}$ and $\mathcal{T}$ if and only if*

$$\mathcal{T} \cup \mathcal{I}(\mathcal{A}_{\mathcal{G}} \cap F(\Sigma_{\mathcal{I}})) \ \textit{ is consistent if } \mathcal{T} \textit{ is consistent.}$$

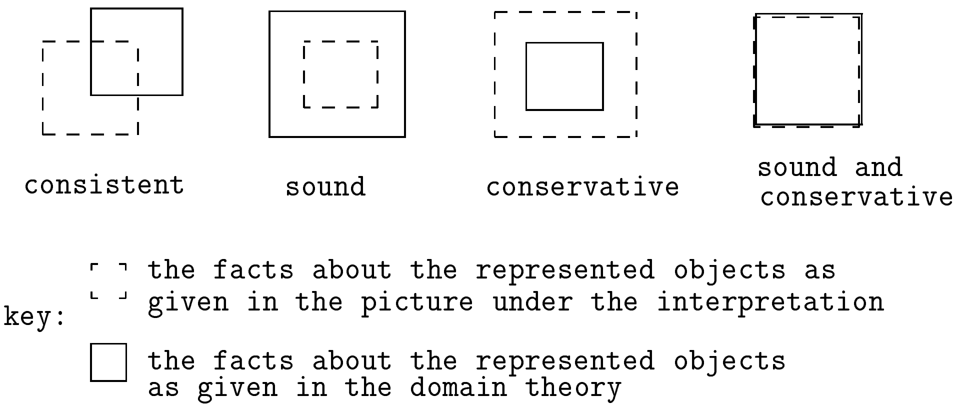



Figure 9.1: An illustration of the three properties

*2. $\mathcal{I}$ is* sound *with respect to $\mathcal{G}$ and $\mathcal{T}$ if and only if*

$$\mathcal{I}(\mathcal{A}_\mathcal{G} \cap F(\Sigma_\mathcal{I})) \subseteq \mathcal{T}.$$

*3. $\mathcal{I}$ is* conservative *with respect to $\mathcal{G}$ and $\mathcal{T}$, if and only if*

$$\{\, A \in At(\Sigma_\mathcal{I}) \cap At(\Sigma_\mathcal{G}) \mid \mathcal{I}(A) \in \mathcal{T} \,\} \subseteq \mathcal{A}_\mathcal{G},$$

*where $At(\Sigma_\mathcal{I})$ and $At(\Sigma_\mathcal{G})$ are the sets of atomic sentences over $\Sigma_\mathcal{I}$ and $\Sigma_\mathcal{G}$, respectively.*

Intuitively, the above three properties of interpretations reflect the relationship between the properties that a picture presents and those in the logical theory of the application domain. *Consistency* of an interpretation means that the information derived by interpretation of the picture does not introduce any contradictions into the domain theory. However, a consistent interpretation may introduce new facts into the application domain theory which are not originally derivable and, on the other hand, the information the interpretation carries may not cover all of the properties of the corresponding objects in the application domain theory. *Soundness* means that all of the properties that an interpretation carries from a picture are already derivable from the domain theory. *Conservativity* means that the properties carried by an interpretation include all of the properties of the corresponding objects in the domain theory. Without explanation, it might be not clear why I use the word *conservativity* instead of *completeness* here. Usually, completeness is used to discuss a property holding between a logical system and its models and conservativity is used to compare systems, *i.e.* the relation between a system and its subsystems. In visual communication, I do not treat pictures as models of application domain theories though in some applications they may behave like models. Instead I treat them as graphical representations of application domain theories. The properties of interpretations are investigated by comparing two systems, where one corresponds to the interpretation related part of the application domain and the other is the system obtained by interpreting the set of formulas deduced from a picture. If an

| | | |
|---|---|---|
| $\Sigma(\mathcal{L})$ | $\mathcal{S}$ | $Circle,\ Line,\ \dots\ \dots$ |
| | $\leq$ | $\dots\ \dots$ |
| | $\mathcal{F}$ | $C_1, C_2, C_3, \dots\ :\ Circle,\ L_1,\ L_2,\ L_3,\ \dots\ :\ Line, \dots$ |
| | $\mathcal{P}$ | $line\_connect\ :\ Circle \times Circle,$<br>$in\ :\ Circle \times Circle,$<br>$t\_joint\ :\ Line \times\ Line$ |
| $\mathcal{T}(\mathcal{L})$ | $line\_connect(x, y) \Leftrightarrow line\_connect(y, x)$<br>$in(x, y)\ \wedge\ in(y, z) \Rightarrow in(x, z)$<br>$\dots\ \dots$ | |
| $\Sigma$ | $\mathcal{S}'$ | $Set,\ \dots\ \dots$ |
| | $\leq'$ | $\dots\ \dots$ |
| | $\mathcal{F}'$ | $a, b, c\ :\ Set,\ \dots\ \dots$ |
| | $\mathcal{P}'$ | $\subseteq:\ Set \times Set,\ \dots\ \dots$ |
| $\mathcal{T}$ | $\subseteq$ is the partial order generated by<br>$a \subseteq b \subseteq c.$ | |

Figure 9.2: A picture description $\mathcal{L} = (\Sigma(\mathcal{L}), \mathcal{T}(\mathcal{L}))$ and a domain signature and theory.

interpretation is conservative (and also consistent) the system which results from the picture can be viewed as a conservative expansion of the application domain system. If an interpretation is both sound and conservative, the facts that an interpretation carries coincide with the properties of the corresponding objects in the domain theory. Figure 9.1 gives an illustration of these three properties, where a solid box represents the facts of the represented objects in the domain theory and a dash box represents the facts an interpretation carries from the representing graphical objects in a picture.

Now, I give examples to explain the above properties of interpretations. Let $\mathcal{T}(\mathcal{L})$ be the graphical theory over $\Sigma(\mathcal{L})$ and $\mathcal{T}$ be the domain theory over $\Sigma$ as described in Figure 9.2. Consider three interpretations $\mathcal{I}_i : \Sigma(\mathcal{L}) \rightharpoonup \Sigma$ and their properties.

1. $\mathcal{I}_1$ interprets the graphical sort *Circle* as *Set* and $line\_connect(X, Y)$ as $\mathcal{I}(X) \subseteq \mathcal{I}(Y)$, for $X, Y : Circle$, as in Figure 9.3. $\mathcal{I}_1$ is inconsistent with respect to the picture in Figure 9.3 and $\mathcal{T}$. The geometrical characterisation of the picture in Figure 9.3 (based on the picture description language in Figure 9.2) contains the following formulas:

$$line\_connect(C_1, C_2), \quad line\_connect(C_2, C_1),$$

$$line\_connect(C_2, C_3), \quad line\_connect(C_3, C_2).$$

After interpreting them by $\mathcal{I}_1$, we get the following formulas in the application domain:

$$a \subseteq b,\ b \subseteq a,\ b \subseteq\ c,\ c \subseteq\ b;$$

$b \subseteq a$ and $c \subseteq b$ contradict the formulas $\neg b \subseteq a$ and $\neg c \subseteq b$ in the domain theory $\mathcal{T}$ in Figure 9.2. The inconsistency of $\mathcal{I}_1$ results from the use of the

| | | |
|---|---|---|
| $\Sigma_{\mathcal{I}}$ | $\mathcal{S}_{\mathcal{I}}$ | $Circle, \ldots$ |
| | $\leq_{\mathcal{I}}$ | |
| | $\mathcal{F}_{\mathcal{I}}$ | $C_1,\ C_2,\ C_3:\ Circle;$ |
| | $\mathcal{P}_{\mathcal{I}}$ | $line\_connect Circle \times Circle;$ |
| $\mathcal{I}_1$ | $\mathcal{I}_{\mathcal{S}}$ | $Circle \mapsto Set,$ |
| | $\mathcal{I}_{\mathcal{F}}$ | $C_1 \mapsto a,\quad C_2 \mapsto b,\quad C_3 \mapsto c$ |
| | $\mathcal{I}_{\mathcal{P}}$ | $line\_connect \mapsto \subseteq.$ |

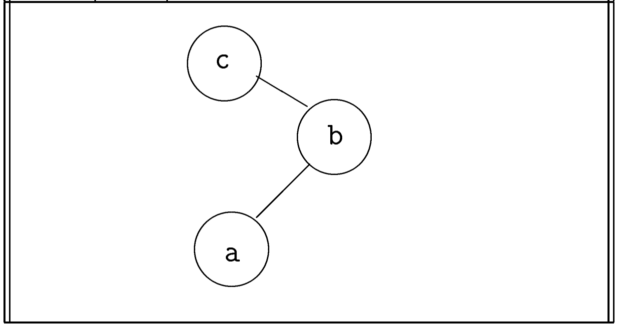



Figure 9.3: $I_1$ is inconsistent with $\mathcal{T}$ and the picture.

symmetric binary predicate *line_connect* in the graphical theory to represent the asymmetric predicate $\subseteq$ in the domain language.

2. $\mathcal{I}_2$ interprets the graphical sort *Line* as *Set* and *t_joint* as $\subseteq$, as in Figure 9.4. Then, $\mathcal{I}_2$ is sound (and hence consistent) with respect to the picture in Figure 9.4 and $\mathcal{T}$. But it is not conservative, for there are some facts in the application domain (*e.g.* $a \subseteq a$, $a \subseteq c$ *etc.*) which have no corresponding representation in the picture in Figure 9.4. The non-conservativity results from *t_joint* being neither transitive nor reflexive in the graphical theory.
3. $\mathcal{I}_3$ interprets *Circle* as *Set* and *in* as $\subseteq$, as in Figure 9.5. $\mathcal{I}_3$ is sound and conservative with respect to the picture in Figure 9.5 and $\mathcal{T}$.

The above examples illustrate the three properties of interpretations. However, the examples may give a false impression that the semantic correctness of an interpretation is determined by the coincidence of the properties of the graphical entities with those in the application domain. In the examples, when the properties between the graphical predicate and the $\subseteq$ relation are conflicting (*e.g.* *line_connect* is symmetric but $\subseteq$ is asymmetric) the interpretation ($\mathcal{I}_1$) is inconsistent; when the set of the properties of the graphical predicate is a proper subset of the properties of $\subseteq$ (*e.g.* *t_joint* is asymmetric) the interpretation ($\mathcal{I}_2$) is sound but not conservative; only when the properties of both are the same (*e.g.* *in* is reflexive, asymmetric and transitive, and $\subseteq$ also has these three properties) the interpretation ($\mathcal{I}_3$) is both sound and conservative. So it seems easy to believe that the relations between the two sets of properties determine the properties of an interpretation. However, this is not right. We know that the relation *father* (*e.g.* $father(A, B)$ means A is B's father) is not transitive, and the graphical predicate *in* in the picture description language in Figure 9.2 is transitive. Nevertheless, if $\mathcal{I}$ interprets *in* into *father*, $\mathcal{I}$ is

| | | |
|---|---|---|
| $\Sigma_{\mathcal{I}}$ | $\mathcal{S}_{\mathcal{I}}$ | $Line, ...$ |
| | $\leq_{\mathcal{I}}$ | |
| | $\mathcal{F}_{\mathcal{I}}$ | $L_1,\ L_2,\ L_3:\ Line;$ |
| | $\mathcal{P}_{\mathcal{I}}$ | $t\_joint\ :\ Line \times Line;$ |
| $\mathcal{I}_2$ | $\mathcal{I}_{\mathcal{S}}$ | $Line \mapsto Set,$ |
| | $\mathcal{I}_{\mathcal{F}}$ | $L_1 \mapsto a,\quad L_2 \mapsto b,\quad L_3 \mapsto c$ |
| | $\mathcal{I}_{\mathcal{P}}$ | $t\_joint \mapsto \subseteq.$ |

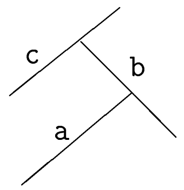



Figure 9.4: $I_2$ is sound with $\mathcal{T}$ and the picture.

sound and conservative with respect to $\mathcal{T}$ and the picture in Figure 9.6.

Once again, it is emphasized that we can only see *basic* facts in a picture as discussed in Section 4.1, and in visual communication, only *basic* facts in the picture are interpreted in the application domain as I defined in Section 8. In the case in Figure 9.6, only the basic facts (*e.g.* $in(Circle_{Mary}, Circle_{Tom})$) are interpreted in the application domain (*e.g.* $father(Tom, Mary)$). The general properties are not interpreted in the application domain. For instance, the transitivity of the *in* relation $\forall x, y, z : Circle(in(x, y) \wedge in(y, z) \rightarrow in(x, y))$ does not mean that $\forall x, y, z : Person(father(y, x) \wedge father(z, y) \rightarrow father(z, x))$.

However, this certainly does not mean that the properties of a graphical entity have nothing to do with the entity it interprets in the application domain. If the graphical entity has the same general properties with its represented entity, it is easy to build an isomorphic graphical representation; otherwise, for some facts in the application domain it may not be possible to find a correct representation.

The above analysis provides a formal understanding of the semantic correctness of interpretations. Semantic correctness cannot be automatically checked; technically, it is undecidable, and pragmatically, the supporting system has no way to get a complete grasp of the semantics of the application domain. However, such an understanding is important for providing useful guidance to help users, both conceptually and by machine, to choose correct interpretations. For instance, a supporting system may give the user some useful information about the graphical entities chosen by the user, such as the symmetry property of *line_connect* in the interpretation $\mathcal{I}_1$ and in this case the user will realise that something explicitly directional, such as *arrow_connection*, should be used.

| | | |
|---|---|---|
| $\Sigma_{\mathcal{I}}$ | $\mathcal{S}_{\mathcal{I}}$ | $Circle, ...$ |
| | $\leq_{\mathcal{I}}$ | |
| | $\mathcal{F}_{\mathcal{I}}$ | $C_4,\ C_5,\ C_6 :\ Circle;$ |
| | $\mathcal{P}_{\mathcal{I}}$ | $in\ :\ Circle \times Circle;$ |
| $\mathcal{I}_3$ | $\mathcal{I}_{\mathcal{S}}$ | $Circle \mapsto Set,$ |
| | $\mathcal{I}_{\mathcal{F}}$ | $C_4 \mapsto a,\quad C_5 \mapsto b,\quad C_6 \mapsto c$ |
| | $\mathcal{I}_{\mathcal{P}}$ | $in \mapsto \subseteq$ |

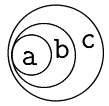



Figure 9.5: $I_3$ is sound and conservative with $\mathcal{T}$ and the picture.

| | |
|---|---|
| $\mathcal{T}$ | $father(Tom, Mary),\ father(John, Scott),\ father(John, Lina)$ |


Figure 9.6: $in$ is transitive and $father$ is anti-transitive, but the interpretation of $in$ to $father$ is sound and conservative with respect to $\mathcal{T}$ and the picture.

## 9.3 Pythagoras' Theorem: an example of abstract reasoning

Having discussed the correct use of pictures in visual communication in rather abstract terms and with artificial examples, I discuss, in this section, how the notion of interpretation is critically used in a well-known but more subtle example of visual reasoning — the use of graphical objects to represent arbitrary objects and properties in a proof of Pythagoras' Theorem.

One may distinguish two different kinds of reasoning about an application domain. The first may be called '*concrete reasoning*', where properties of particular objects are concerned, for example when one wants to show that a particular set is a subset of another. The second kind of reasoning, which I may call '*abstract reasoning*', is used to prove some properties that an arbitrary object of a certain kind has, for example to prove Pythagoras' theorem which concerns *arbitrary* right angled triangles.

For concrete reasoning, one uses particular graphical objects in a picture to represent particular objects in the domain, and graphical inference under an interpretation provides a direct answer to the questions one may ask about the represented objects. For example, whether or not set $A$ is a subset of $B$ ($A \subseteq B$) can be answered through

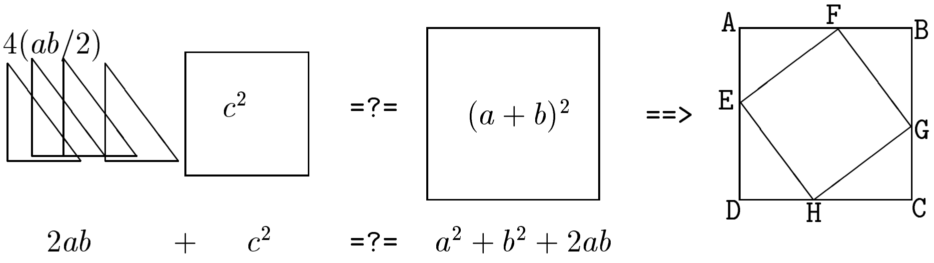



Figure 9.7: Outline of proof of Pythagoras' Theorem.

graphical inference of whether or not circle $A$ is inside $B$ ($in(A,B)$), if *Circle* is interpreted as *Set* and *in* as $\subseteq$.

However, since pictures are particular models of the graphical theory, it is natural to ask whether and how one can use pictures to help in proving universal properties in abstract reasoning. For example, can we use pictures to help in proving Pythagoras' theorem? If so, how can one regard an arbitrarily chosen (but particular) right angled triangle in a picture as an arbitrary right angled triangle? It is obvious that, if we are not careful, it is easy to deduce incorrect conclusions from such a use of pictures in reasoning; for example, if the particular right angled triangle representing the arbitrary right angled triangle happens to be an isosceles triangle, one might get the absurd conclusion that every right angled triangle is an isosceles triangle.

In the following, the example of proving Pythagoras' Theorem is considered in the setting of visual communication. I shall give a brief analysis and explain how pictures may be used to help abstract reasoning. In particular, it is emphasized that interpretations are the key tool in understanding and controlling how pictures (and which parts of them and what properties of them) are used in reasoning about the application domain.

Pythagoras' Theorem asserts that in a right angled triangle, if the hypotenuse has length $c$ and the other two sides have lengths $a$ and $b$, then $a^2 + b^2 = c^2$. An elegant proof is obtained by forming two squares using four identical right angled triangles (see Figure 9.7) and equating areas. That is, one shows that the picture contains two squares, and then the fact that the area of the smaller square is equal to that of the outer minus four times the area of the right angled triangle gives the proof of the theorem.

It is obvious that, in such an argument, the picture plays an essential role, without which one could not proceed with the argument (or even find it). However, on the other hand, it is not completely clear how the picture helps in the proof and what information provided by it is used in the proof. For example, the particular fact that one of the sides in a triangle is longer than another, is not (and should not) be used in the proof.

Using such a picture to prove Pythagoras' theorem, we must understand that the triangles in the picture represent an arbitrary right angled triangle in the application domain. Therefore, particular properties such as that one of the edges is longer

| | | |
|---|---|---|
| $\Sigma_{\mathcal{I}}$ | $\mathcal{S}$ | $Angle,\ Edge,\ ...$ |
| | $\leq$ | $...$ |
| | $\mathcal{F}$ | $EAF,\ FBG,\ GCH,\ HDE\ :\ Angle,\quad EF,\ FG,\ GH,\ HE\ :\ Edge$<br>$AFE,\ BGF,\ CHG,\ DEH\ :\ Angle,\quad AE,\ BF,\ CG,\ DH\ :\ Edge$<br>$AEF,\ BFG,\ CGH,\ DHE\ :\ Angle,\quad AF,\ BG,\ CH,\ DE\ :\ Edge$<br>$EFG,\ FGH,\ GHE,\ HEF\ :\ Angle$ |
| | $\mathcal{P}$ | $...$ |
| $\Sigma'$ | $\mathcal{S}'$ | $R'Triangle,\ S'quare,\ A'ngle,\ E'dge,\ ...$ |
| | $\leq'$ | $...$ |
| | $\mathcal{F}'$ | $R\ :\ R'Triangle,\quad C\ :\ S'quare$<br>$RA,\ RB,\ RC,\ C_1,\ C_2,\ C_3,\ C_4\ :\ Angle$<br>$Ra,\ Rb,\ Rc\ :\ Edge$ |
| | $\mathcal{P}'$ | $...$ |
| $\mathcal{T}'$ | $Rc$, $Ra$ and $Rb$ are the three sides of $R$ where $Rc$ is the hypotenuse.<br>$RC$, $RA$ and $RB$ are the three angles of $R$.<br>$RA + RB = RC = C_1 = C_2 = C_3 = C_4 = \pi/2 \qquad (1)$<br>axioms about triangles and squares<br>... ... | |
| $\mathcal{I}$ | $\mathcal{I}_{\mathcal{S}}$ | $Angle \mapsto A'ngle,\qquad Edge \mapsto E'dge,\ ...$ |
| | $\mathcal{I}_{\mathcal{F}}$ | $EAF,\ FBG,\ GCH,\ HDE\ \mapsto RC,\qquad EF,\ FG,\ GH,\ HE \mapsto Rc$<br>$EFG \mapsto C_1,\quad FGH \mapsto C_2,\ GHE \mapsto C_3,\quad HEF \mapsto C_4$<br>$AFE,\ BGF,\ CHG,\ DEH \mapsto RA,\qquad AE,\ BF,\ CG,\ DH \mapsto Ra$<br>$AEF,\ BFG,\ CGH,\ DHE \mapsto RB,\qquad AF,\ BG,\ CH,\ DE \mapsto Rb$ |

Figure 9.8: The graphical signature, domain signature and theory, and the interpretation for proving Pythagoras' Theorem.

than another should not be used in domain reasoning[1]. This can be controlled by means of interpretation, since, according to the logical characterisation of visual communication, only the interpreted pictorial components (and their interpreted properties) can be used. In other words, giving a correct and intended interpretation is a most important step in using pictures in communication.

In the interpretation (see Figure 9.8) for proving Pythagoras' theorem, numerical attributes like *length* of edges are not interpreted (that is, they are not included in the signature of the interpretation), since an arbitrary right angled triangle represented by the triangles in the picture does not necessarily have the particular (numerical) properties that these pictorial objects may have. This prevents us from deducing incorrect conclusions from graphical inference. Having given such an interpretation, one can use the picture in Figure 9.7 to guide and help to find the proof as sketched above. For instance, in order to prove that (the area represented by) $ABCD$ is a square, it is necessary to prove that (the lines represented by) $AF$ and $FB$ are

[1]Such a possible fallacy in a proof of Pythagoras' theorem has also been pointed out by Barwise and Etchemendy in [5]. Here, I analyse this phenomenon and make it clear in my framework.

collinear. To do this, one may point at the picture to prove that the sum of degrees of (the angles represented by) $AFE$, $BFG$ and $EFG$ is equal to $\pi$. Note that, by pointing at the picture and using '$ABCD$', '$AF$', '$AFE$' etc., the developer of the proof is really referring to the corresponding objects in the domain theory which are represented by these graphical objects! For example, by pointing at the angles $AFE$, $BFG$ and $EFG$, the visual reasoning supporting system gives the angles $RA$, $RB$ and $C_1$ in the application domain according to the interpretation, the axiom (1) in the domain theory $\mathcal{T}'$ results in the conclusion that the sum of the three angles is equal to $\pi$, so $AF$ and $FB$ are collinear [2]. In other words, by pointing at graphical objects one is referring to the images of these graphical objects under the interpretation. Through this, the picture plays an essential role guiding the development or Explanation of the proof.

Note that, based on the interpretation in Figure 9.8, there is no way the user can, for example, use the size of an angle in the picture (say $AFE$) to refer to the size of the represented angle in the domain theory, since the attribute *angle size* (except for the *right angle*) is not interpreted. Similarly, the attribute *length* is not interpreted, either. Otherwise, if *degree* and *length* were interpreted, one would be able to 'prove' the Pythagoras theorem simply by computing the values of the graphical attributes (say, the lengths of the edges of a right angled triangle which may be 3, 4 and 5 respectively) and calculate that $c^2 = a^2+b^2$ ($5^2 = 3^2+4^2$); this certainly does not prove the theorem in general. In my terminology, such an interpretation is only *consistent*, but *not sound*; it carries unintended information (like the numerical attributes) over to the application domain and makes the represented triangle become a particular triangle, and no longer an arbitrary one. In conclusion, pictures are useful in both concrete and abstract reasoning. In fact, interpretations may also provide ways to deal with both abstract and concrete reasoning. When abstract reasoning is considered, interpretations should be given more carefully, especially with respect to the geometrical attributes of graphical objects.

[2]Note, not only are the facts carried in the picture but also the axioms about triangles and squares in the application domain theory used in the reasoning (see the definition and the discussion in Section 8).

# 10 Applications

I have discussed interpretations from a theoretical point of view. Now the question arises as to how interpretations should be incorporated into computer systems for visual communication. In the standard approach, the interpretations of visual expressions such as icons and presentation diagrams are given fixed interpretations predetermined by system designers. This approach assumes that the built-in meanings of the visual expressions have a common-sense understanding shared by a community, and is suitable and proves to be successful for those applications where particular graphical representation methods can be widely accepted (*e.g.* the use of visual expressions such as data flow charts to represent computational processes in visual programming languages [30], bar-charts to represent various kinds of relations in visualisation systems [40], and icons to represent classes and objects in databases [33]), or can be set up by experts for their long-term use (*e.g.* designing a set of icons to represent cell populations [47]).

However, in more general visual communication (such as visual reasoning, teaching), the meanings of visual expressions are usually embodied in their uses (*c.f.* Wittgenstein's view on meanings [62]). There is a wide range of choices of visual expressions in such visual communication. The choice of visual representations is itself a highly creative activity and depends a great deal on the current problem and the individuals involved in communication, and cannot be reasonably determined by the designer of the supporting system.

Based on such considerations, I present a system structure (which I call *interpretation-based structure*) to support visual communication. The notable feature of the interpretation-based structure is that the interpretations are not to be mixed up with the other components of the system, but will exist as an independent module providing support for users choosing their own graphical representations according to their current problems and intentions. The system structure consists of the following main components (see Figure 10.1):

- The *interpretation mechanism* helps users to set up the intended and correct interpretation. Under the control of the interpretation mechanism, the syntactic

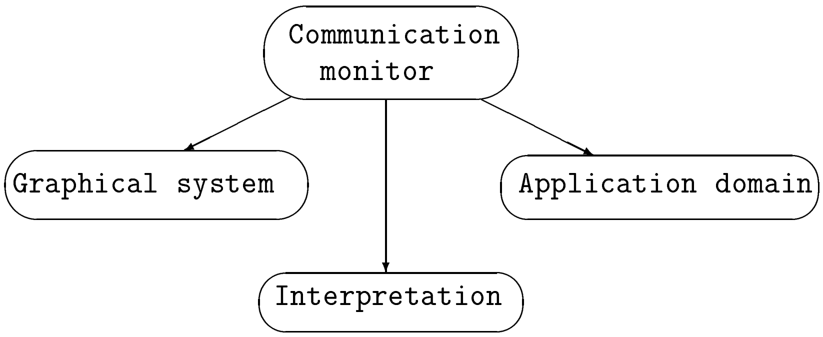



Figure 10.1: The main components of the supporting system.

correctness of a user's interpretation must be guaranteed.

- The *graphical system* consists of an order-sorted picture description language and a graphical inference engine, which provides users with a friendly graphical environment for drawing and graphical inference.
- The *application domain* provides users with a language for (non-graphical) description and problem-solving in their applications;
- The *communication monitor* controls and realises the various kinds of visual communication.

In the following, firstly I discuss three kinds of visual communication which can be supported by such a system structure. Secondly, I present a concrete system, GAR, which is implemented on the basis of the theoretical conclusions from Part I and Part II, and which supports one kind of visual communication (demonstrative visual communication). Thirdly, I give a further theoretical treatment of another kind of visual communication (reasoning with diagrammatic representations).

## 10.1 Categories of visual communication

Visual communication supported by the interpretation-based structure can be classified into three kinds, (1) illustrative visual communication: graphical illustration or presentation of facts in an application domain; (2) demonstrative visual communication: stepwise graphical demonstration of how a problem in an application domain is solved by graphical inference; (3) reasoning with diagrammatic representations: the conclusion of an inferential problem is treated as analogous to pictures representing the conclusion, that are derived from the pictures representing the premises.

A simple example of illustrative visual communication is shown in Figure 10.2, where the points on the x-axis represent the names of cars, the numbers on the y-axis the prices and the axis bars represent the *cost* relation between cars and their prices. After this interpretation, the cost relation in the data base can be graphically represented as the picture in Figure 10.2. Illustrative visual communication is the most basic form of visual communication. Many visual systems can be put into this

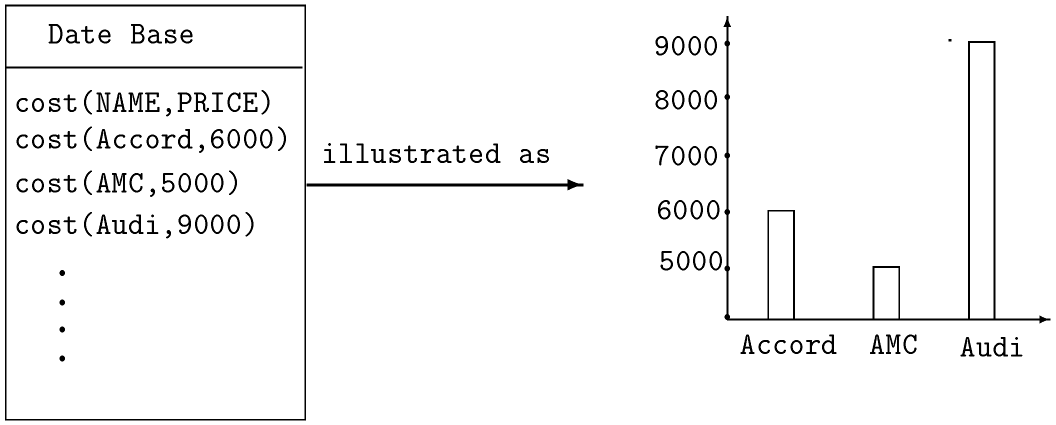



Figure 10.2: An example of illustrative visual communication.

category (*e.g.* *COOL* [32], *APT* [40]). The main problems to be solved in order to support such visual communication are how to arrange graphical objects on the computer screen; this is related to the work on picture generation and constraint solving which is outside the subject studied in this thesis.

Demonstrative visual communication is more complicated than illustrative communication, for not only are the basic facts in the application domain graphically represented but also the truth of a proposition is illustrated by demonstrating graphical inference. Using Venn diagrams to illustrate the validity of the laws of set operations belongs to this kind of visual communication. For example, consider the illustration of the validity of the distributive law of set operation $\cup$ and $\cap$ in Figure 6.1. The validity of the formula

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

is illustrated by graphically demonstrating what $B \cap C$ is, what $A \cup (B \cap C)$ is *etc.* successively reaching the conclusion that the left-hand side of the formula is equal to the right-hand side.

The third kind of visual communication (reasoning with diagrammatic representation) is to use diagrams in real world reasoning in such a way that an inference is represented by some set of diagrams showing the premises and another showing the possible conclusions. Solving syllogisms by the use of Euler circles is a typical example of this kind of visual communication. Figure 10.3 shows an example of this kind of visual communication, where one picture represents the premiss *all A are B*, another *some C are not B* and the third picture which follows from the two premiss pictures by applying some kind of rule (which is called G-morphism in [15] and which I shall discuss in Section 10.3) represents *some C are not A*, which is just the conclusion of the two premisses by means of syllogism.

Illustrative visual communication is the basic category among the three kinds of visual communication. A system supporting either of the other two kinds of commu-

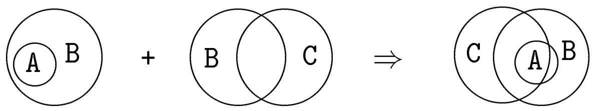



Figure 10.3: *all A are B* + *some C are not B* $\Rightarrow$ *some C are not A*. An example of reasoning with diagrammatic representations.

nication must be able to support illustrative visual communication. Though there are many interesting technical points in the implementation of such a supporting system, they are mainly related to research topics in computer graphics, and their theoretical relevance to this thesis is trivial. Therefore, I will not discuss illustrative visual communication further here. In the following, firstly, I present the experimental system GAR. Secondly, I present a theoretical treatment of reasoning with diagrammatic representations based on geometrical characterisation of pictures and the notion of interpretation.

## 10.2 GAR: a Graphics-Assisted Reasoning system

In this section, I describe the experimental system, GAR (Graphics-Assisted Reasoning),[1] designed and implemented based on the formal study of visual communication via interpretations. GAR is designed to support demonstrative visual communication.

A significant feature of GAR that differs from most of the other systems (eg [5, 53]) is that users are allowed to give their own interpretations to graphical entities and the system uses the interpretation given by the user to support visual communication.

Following the system structure in Figure 10.1, GAR is designed to consist of the following components. A *graphical system* provides an order-sorted picture description language and implements graphical inference by geometrical algorithms. In the current experimental implementation, I have chosen a fairly small graphical signature intended to consider visual communication in the domain for demonstrating the validities of set operations. The implementation is intended to be extendable to a much larger picture description language. An *interpretation system* allows users to specify interpretations, and guarantees the syntactic correctness of user-specified interpretations. A *visual reasoning mechanism* supports visual reasoning based on the interpretations given by the user.

The interface of the GAR system is shown in Figure 10.4, where the current graphical signature is displayed with pictures as examples in window (1) to help users to understand the geometrical meanings of the graphical entities; window (2) is used by the user to draw original pictures for visual communication, and window (3) is used by the system to demonstrate steps of graphical inference so that the user can understand how the problems are solved by inspecting the demonstration of the corresponding processes of graphical inference. In the following, I present examples using diagrams to illustrate the various laws of the set operations $\cup$, $\cap$ and $-$, to give an intuitive impression of how the GAR system helps users in visual communication.

For example, the user may give his interpretation by interpreting $Closed\_curve$ as *set*, *overlap* as *cap* (*i.e.* $\cap$ ), *merge* as *cup* (*i.e.* $\cup$) and $difference$ as $-$, as shown in Figure 10.5. Then the user can draw three circles on the screen of window (2), which are understood by both the user and GAR system as three sets, $a$, $b$ and $c$, according to the interpretation given above.

Now, the GAR system cannot only answer the user's various queries about the operations *cap*, *cup* and $-$ over the three sets, but demonstrate the process of problem-solving by graphical inference in the language of the application domain (set theory in this case). For instance, the user may enter the following query:
$cup(a, cap(b, c)) == cap(cup(a, b), cup(a, c))$? as shown in Figure 10.6. To respond, besides giving a "Yes/No" answer in the form of *Left side* $==$ *Right side*, the GAR system also provides a comprehensible illustration of *how* the conclusion is deduced, to explain *why* it is true (see Figure 10.7). The user may also ask various other questions such as the associative law for *cup* (or *cap*), DeMorgan's law *etc.* and the system will give responses (see Figure 10.8 and Figure 10.9).

[1] GAR is implemented on top of the Gizmo system [35].

Figure 10.4: The user interface of the GAR system.

Figure 10.5: The interpretation given by the user.

Figure 10.6: The user enters a query.

Figure 10.7: The GAR system answers the query in Figure 10.6.

Figure 10.8: The user enters another query.

Figure 10.9: The GAR system answers the query in Figure 10.8.

## 10.3 G-morphism and reasoning with diagrammatical representations

In this section, *reasoning with diagrammatic representations* will be studied. A well-known example is syllogistic reasoning using Euler circles. In figure 10.10, two diagrams represent the premisses and another one the conclusion.

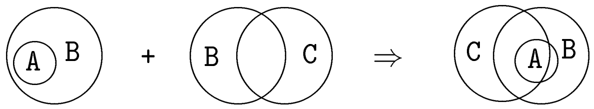



Figure 10.10: *all A are B + some C are not B ⇒ some C are not A*

A computational model of reasoning with diagrammatic representation must have the following three components (see in Figure 10.11): (1) a geometrical characterisation of the premiss pictures (which represent the premiss) and the conclusion pictures (which represent the conclusions), (2) an interpretation which associates the pictures to inference about the application domain, (3) G-morphisms (the relations between premiss and conclusion pictures), and (4) a logical characterisation of reasoning with diagrammatic representations.

The geometrical characterisation (and interpretations) of pictures have been studied in terms of the model in which only one picture is involved in a visual communication state. But now there are two groups of pictures involved in the reasoning. The geometrical descriptions (and the interpretations) to each of them should be consistent with each other. In the following, therefore, I will introduce some conventions on the descriptions and the interpretations of the pictures, respectively. Based on this, G-morphisms are studied by introducing some mapping conditions by means of which mapping rules can be made to produce the possible conclusion pictures from the premiss pictures. For example, a rule, which unifies the same labeled objects in two diagrams, forms the third diagram from the former two in Figure 10.10 (here the circles labelled with $B$ are unified). I will then give a logical characterisation of reasoning with diagrammatic representations, which includes the meaning of a conclusion picture and, if there is more than one conclusion picture, the relationship between each of them and the real domain. Finally, I present examples which further illustrate the theoretical study.

### 10.3.1 Picture descriptions

A G-morphism is a relation which can be thought of as a transformation allowing premiss diagrams to be combined in certain ways, and to take a number of different forms corresponding to different consistent representations of the information contained in the premisses.

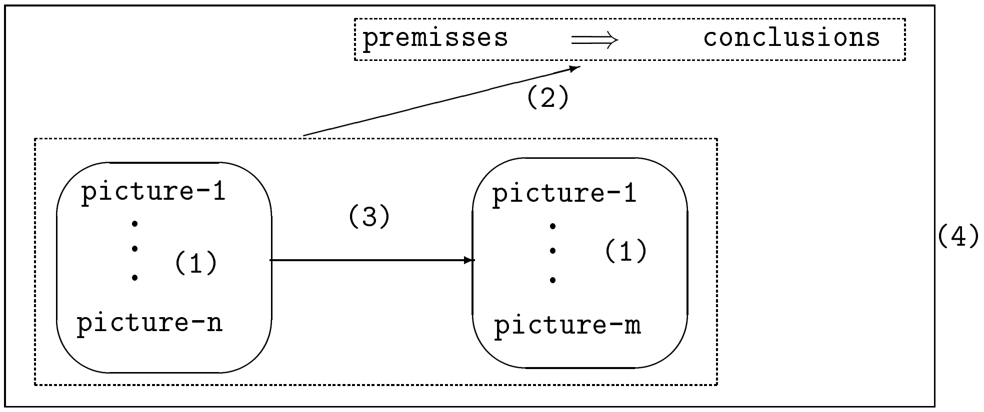



Figure 10.11: A model of reasoning with diagrammatic representations.

When a G-morphism transfers a set of diagrams to another, it maps each graphical object in the premiss diagrams into another object in the conclusion diagram. A direct representation of such a mapping would be as a set of pairs like $(O_i, O_j)$, which indicates that the graphical object $O_i$ is mapped to $O_j$. However, if we consider the use of G-morphism in real world reasoning, in most applications a real world object is represented by several different graphical objects (in different diagrams). For instance, in Figure 10.10, the set $B$ is represented by three different circles whose sizes may be the same but their relative spatial positions are different from each other. Though they are different graphical objects, they are recognised to be (representations of) the same object in that application because of the label $B$ attached to each of them. When we look at a G-morphism (*e.g.* the transformation between diagrams in Figure 10.10) we usually view a label and the graphical objects attached to it as a whole (*e.g.* $B$ and the circles labeled $B$ in Figure 10.10) [2].

Therefore, I give the following conventions to the description of the pictures in G-morphisms.

**Convention**:

(1) There is a *label* attached to each graphical object in a diagram. More precisely, I use $(g, l)$ : to represent a graphical object, where $g$ is the name of the graphical object (*i.e.* a term in the picture description language) and $l$ is a label attached to the graphical object. For instance, consider the diagram in Figure 10.12. If we relate the graphical objects $p_1$, $p_2$, $C_1$, $C_2$ and $overlap(C_1, C_2)$ with labels $a$, $b$, $A$,

[2] In the early sections, sometimes, a label attached with a graphical object is the name of the graphical object (*e.g.* the picture in Figure 4.1 (1))and other times, it is the name of the represented object in the application domain (*e.g.* pictures in Figure 9.3, 9.4). When only one picture is used in a visual communication state it is not important to distinguish the name of a graphical object and the name of the represented object, because in most cases we can use the same name for both. But the distinction becomes important in the study of the G-morphism because several different graphical objects (or one graphical object) may represent one object (or several different objects) in the application domain.

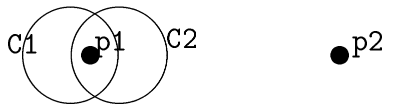



Figure 10.12: C1, C2, p1 and p2 are names of the graphical objects.

$B$ and $C$ respectively, then the objects are re-represented as $(p_1, a)$, $(p_2, b)$, $(C_1, A)$, $(C_2, B)$ and $(overlap(C_1, C_2), C)$. For any diagram $\mathcal{D}$, I use $BL_{\mathcal{D}}$ (*basic* label set) to represent the set of the labels attached to the original graphical objects and $EL_{\mathcal{D}}$ (*emergent* labels set) to represent the emergent graphical objects in the diagram, *e.g.* $BL_{\mathcal{D}} = \{a, b, A, B\}$ is the basic label set and $EL_{\mathcal{D}} = \{C\}$ the emergent label set of the above diagram.

(2) For any diagram $\mathcal{D}$ and label $l$, $l$ appears in $\mathcal{D}$ at most once. Usually labels are the names of the represented objects in an application domain. This convention means we always use one graphical object to represent a domain object. One may wonder whether it should be allowed to have several graphical objects represent a single domain object, have, for instance, both a square and a triangle represent a house. In our view, it is more natural to treat the square and the triangle as one (complex) graphical object instead of two graphical objects. If we must have several graphical objects to represent a single domain object in one diagram, we can still start by different labels for each of the graphical objects and then map all of the labels to the represented object. For this reason, the graphical objects in a diagram can be represented unambiguously by just their labels. In the following, if there is no ambiguity, constants (*i.e.* the representation of basic graphical objects) in a graphical signature will also be represented by just their labels. The labels for emergent objects can be formed from the labels of their components. For example, if $(C_1, a)$ and $(C_2, b)$ are two overlapping circles, $(overlap((C_1, a), (C_2, b)), overlap(a, b))$ may be used to represent the emergent closed curve, and $overlap(a, b)$ is the new label attached to it.

(3) There are no labels attached to an empty graphical object. A graphical object may be transformed away into nothing, in which case its associated label disappears. This happens when premiss pictures are inconsistent with each other, I will discuss this in the definition of G-morphism.

(4) In Part I, we saw that giving a picture a geometrical characterisation must be based on an underlying graphical signature and a graphical theory over the signature. The geometrical characterisation of each picture in a G-morphism must be based on a single graphical signature and theory.

**Notation** Let $\mathcal{L}$ be a picture description language, $P(\mathcal{L})$ denotes a set of pictures such that: for each picture $p \in P(\mathcal{L})$ whose geometrical characterisation is

$[\![p]\!] = (\Sigma_p, \mathcal{A}_p)$, $\Sigma_p \sqsubseteq \Sigma(\mathcal{L})$ and $\mathcal{A}_p$ is the intersection of the equivalent situations which are consistent and maximal with respect to $\mathcal{T}(\mathcal{L})$.

### 10.3.2 Interpretations

For each picture in a G-morphism, there is an interpretation which relates the picture with an application domain. It seems reasonable that all the diagrams in a G-morphism should be interpreted by means of a common interpretation. But this might not be always true, *e.g.* in Figure 10.10 the circle labeled $B$ in the second premiss diagram has exactly the same geometrical shape and relative spatial position as the one labeled $C$ in the conclusion picture, but this graphical object has been interpreted as a different set in each picture. Fortunately, there is a label for each graphical object. what appears to be an identical graphical object can be viewed as a different one by having a different label. This avoids conflicting interpretations which might exist in applications. Therefore, interpretations for each picture will be consistent in terms of the labels instead of the graphical objects. This leads the following convention.

**Convention**
Let $\mathcal{L}$ be a picture description language, $\Sigma$ a signature and $\mathcal{I}$ an interpretation from $\mathcal{L}$ to $\Sigma$ whose signature is $\Sigma_{\mathcal{I}}$. $P(\mathcal{L}_{\mathcal{I}})$ denote a set of pictures such that:

1. $P(\mathcal{L}_{\mathcal{I}})$ is $P(\mathcal{L})$;
2. for any two terms, $t_i = (O_i, l_i)$ and $t_j = (O_j, l_j)$, generated from $\Sigma_{\mathcal{I}}$, if the labels $l_i = l_j$ then $\mathcal{I}(t_i) = \mathcal{I}(t_j)$ (*i.e.* for those graphical objects whose labels are the same must be interpreted in the same object in the application domain);
3. for each picture $P \in P(\mathcal{L}_{\mathcal{I}})$, whose geometrical characterisation is $[\![P]\!] = (\Sigma_P, \mathcal{A}_P)$, $\mathcal{A}_P^{\mathcal{I}}$ denotes $\mathcal{A}_P \cap F(\Sigma_{\mathcal{I}})$.

### 10.3.3 G-morphism

**10.3.1.** Definition. (G-morphism)
*Let $\mathcal{L}$ be a picture description language, $\Sigma$ an application domain signature and $\mathcal{I}$ an interpretation from $\mathcal{L}$ to $\Sigma$, whose signature is $\Sigma_{\mathcal{I}}$. A G-morphism $G$ in $\mathcal{T}(\mathcal{L})$ with $\mathcal{I}$ is a pair $(P_1(\mathcal{L}_{\mathcal{I}}), P_2(\mathcal{L}_{\mathcal{I}}))$, where $P_1(\mathcal{L}_{\mathcal{I}})$ are the premise diagrams and $P_2(\mathcal{L}_{\mathcal{I}})$ are the conclusion diagrams which satisfy the following conditions:*

- Identity*: For any $P_2 \in P_2(\mathcal{L}_{\mathcal{I}})$, the basic labels that occur in $P_2$ must occur somewhere among the basic labels in the premiss diagrams, i.e.*

$$BL_{P_2} \subseteq \bigcup_{P_1 \in P_1(\mathcal{L}_{\mathcal{I}})} BL_{P_1}.$$

*New labels are never introduced, unless they come from new emergent objects.*

- Consistency: *For any $P_1 \in P_1(\mathcal{L}_\mathcal{I})$ and $P_2 \in P_2(\mathcal{L}_\mathcal{I})$ the basic facts in $P_1$ and $P_2$ must be consistent with each other, i.e.*

$$\mathcal{A}^{\mathcal{I}}_{P_1} \cup \mathcal{A}^{\mathcal{I}}_{P_2} \textit{ is consistent.}$$

- Maximality: *Any basic fact in a premiss diagram which is consistent with a conclusion diagram (and also with all the other premiss diagrams) must occur in that conclusion diagram. Namely,*

$$\forall P_2 \in P_2(\mathcal{L}_\mathcal{I}), \quad \forall P_1 \in P_1(\mathcal{L}_\mathcal{I})$$

$$\forall A \in \mathcal{A}^{\mathcal{I}}_{P_1}, \quad \{A\} \cup \mathcal{A}^{\mathcal{I}}_{P_2} \textit{ is consistent and}$$

$$\forall P_1' \in P_1(\mathcal{L}_\mathcal{I}), \quad \{A\} \cup \mathcal{A}^{\mathcal{I}}_{P_2} \cup \mathcal{A}^{\mathcal{I}}_{P_1'} \textit{ is consistent}$$

$$\Rightarrow A \in \mathcal{A}^{\mathcal{I}}_{P_2}.$$

The *Identity* condition is justified, because in practice the label of a graphical object will be used to indicate the (semantic) relationship between that object and some object in the real world. A G-morphism transformation can change the geometrical representation of the object in the application domain but should not lose (or change) the link (*i.e.* the label) between the object and its graphical representation. But new emergent graphical objects can be introduced in conclusion pictures, and these play an important role in many applications (see the examples in Section 10.3.5). Besides this, a label is also allowed to be mapped to an empty object (*i.e.* it disappears). The reason is that if a G-morphism obeys the *consistency* condition, a conclusion picture may not convey all the information in the premise diagrams. For example, there might be two premise pictures, $P_1$ and$P_2$, with two circles labeled as $A$ and $B$ overlapping in $P_1$ but not overlapping in $P_2$ (suppose *overlapping* has been considered in the interpretation). Then the conclusion diagrams must be single object diagrams in which only one of $A$ and $B$ appears (see Figure 10.13) if the consistency condition is satisfied.

*Consistency* tells us that the conclusion diagram should not contain any facts which are contradictory with its premises and this is also the most basic condition which should hold for a reasoning system.

The above two conditions only tell us what kinds of objects and properties should not be included in a conclusion diagram. A trivial way to satisfy these two conditions is to define a G-morphism which maps any diagrams to an empty picture. It is obvious that we also need some condition which limits a G-morphism to preserve all the proper objects and their properties. This leads us consider maximality.

*Maximality* guarantees that objects and their spatial properties in the premise diagrams should be mapped to the conclusion diagram as much as possible.

Figure 10.14 gives an example of a G-morphism which maps two premise diagrams, $P_1$ and $P_2$, to four conclusion diagrams. Figure 10.15 gives another example where the map is not a G-morphism with respect to the same graphical theory.

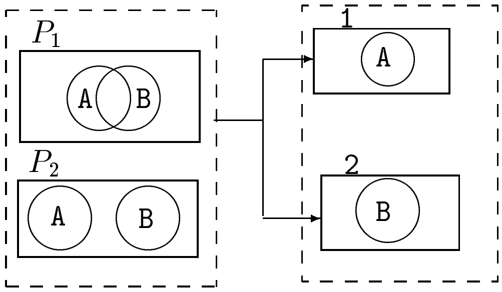



Figure 10.13: *Premise diagrams $P_1$ and $P_2$ are contradictory with each other if the predicate* overlapping *is considered in the picture description language and the interpretation. In order to satisfy the consistency condition, either A (2) or B (1) does not exist in the conclusion diagrams.*

| | | |
|---|---|---|
| $\Sigma$ | $\mathcal{S}$ | $Circle,$ |
| | $\leq$ | ....... |
| | $\mathcal{F}$ | $A\ B\ C\ :\ Circle.$ |
| | $\mathcal{P}$ | $overlapping,\ in\ :\ Circle \times Circle;$ |
| $\Sigma'$ | $\mathcal{S}'$ | $Set,\ ...\ ...$ |
| | $\leq'$ | ... ... |
| | $\mathcal{F}'$ | $A,\ B,\ C\ :\ Set;\ ...\ ...$ |
| | $\mathcal{P}'$ | $\cap\ :\ Set \times Set\ ;\neq \emptyset\ :\ Set; ...\ ...$ |
| $\mathcal{I}$ | $\mathcal{I}_{\mathcal{S}}$ | $Circle \mapsto Set,\ ...\ ...$ |
| | $\mathcal{I}_{\mathcal{F}}$ | $A \mapsto A,\ \ B \mapsto B,\ C \mapsto C$ |
| | $\mathcal{I}_{\mathcal{P}}$ | $overlapping \mapsto \lambda xy.(x \cap y \neq \emptyset)\ ;\ in \mapsto \subset.$ |

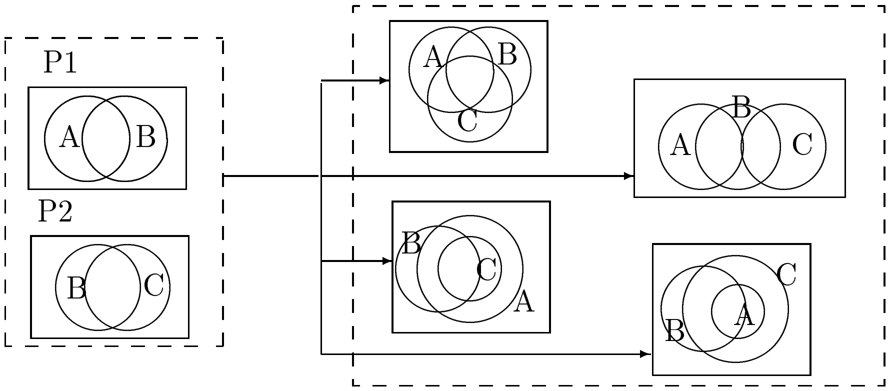



Figure 10.14: *An example G-morphism*

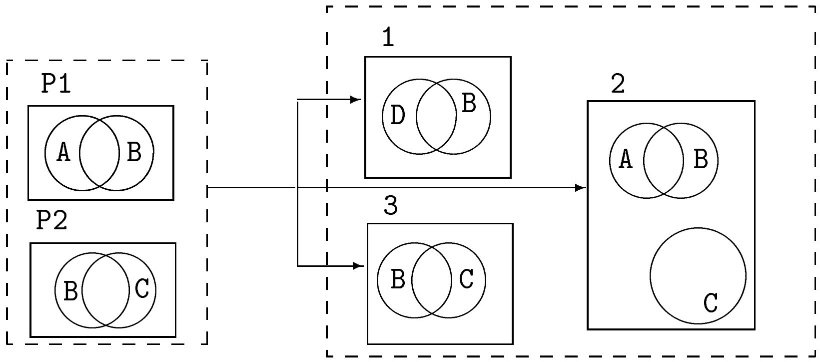



Figure 10.15: *These are not G-morphisms with respect to the graphical theory and the interpretation in Figure 10.14 because: (1) does not satisfy the identity condition; (2) is not consistent and (3) is not maximal.*

### 10.3.4 Reasoning with diagrammatic representations

In the previous section, I discussed the relations between premiss and conclusion pictures mainly from a graphical point of view. Though interpretations are considered in the definition of G-morphism, it served only as a filter to ignore those parts in a picture which the user is not interested. In this section, I study what the conclusion pictures really mean in an application domain, *i.e.* the formal semantics of reasoning with diagrammatic representations.

Considering a consequence relation for reasoning with diagrammatic representations, the meaning of reasoning with diagrammatic representations can be characterised as follows.

**10.3.2.** Definition. (reasoning with diagrammatic representations)
*Let $\mathcal{L}$ be a picture description language, $\mathcal{T}$ the logical theory over a signature $\Sigma$ of the application language, $\mathcal{I}$ an interpretation from $\mathcal{L}$ to $\Sigma$, $G = (P(\mathcal{L}_\mathcal{I}), P'(\mathcal{L}_\mathcal{I}))$ a G-morphism. The* consequence relation $\vdash^G$ determined by $G$, *is defined as follows: for any sentence $A$ over $\Sigma$*

$$\vdash^G A \quad \textit{iff} \;\; \bigvee\{\bigwedge \mathcal{I}(\mathcal{A}_P^\mathcal{I}) \mid P \in P'(\mathcal{L}_\mathcal{I})\} \;\vdash\; A$$

*$A$ is a theorem iff it consists of the disjunction of expressions giving the interpretation for each conclusion diagram (these expressions each being formed by conjoining the interpretations of the diagram's basic facts).*

From this we see that conclusion diagrams must be inconsistent with each other and the conclusion of reasoning with diagram representations is an exclusive disjunction rather than the conjunction of all the facts in each conclusion diagram.

The meaning of the reasoning in Figure 10.14 is:

$$\begin{aligned}\vdash^G \quad & (A \cap C \neq \emptyset \ \wedge A \cap B \neq \emptyset \ \wedge B \cap C \neq \emptyset) \vee \\ & (A \cap C = \emptyset \ \wedge A \cap B \neq \emptyset \ \wedge B \cap C \neq \emptyset) \vee \\ & (A \subset C \ \wedge A \cap B \neq \emptyset \ \wedge B \cap C \neq \emptyset) \vee \\ & (C \subset A \wedge A \cap B \neq \emptyset \ \wedge B \cap C \neq \emptyset).\end{aligned}$$

Note that this neglects certain areas which appear salient in the diagram, e.g. $A \cap B \cap C$. The reason can be seen from the definition of the original graphical signature and theory in Figure 10.14), where it can be seen that *overlapping* is defined only between pairs of circles; there is no predicate representing relations between three circles or between circles and closed-curves. This emphasizes the need for choosing a particular picture description language for a particular purpose, as discussed in Part I.

Besides carefully selecting the graphical signature and the theory, to correctly use G-morphisms in real world reasoning we also need reasonable interpretations. A correct interpretation should guarantee that the basic facts in each conclusion diagram under the interpretation are consistent with the application domain theory.

The conclusion of reasoning with diagrammatic representations is an (exclusive) disjunction. That means (1) any one of the conclusion diagrams can be true, but (2) in a specific model, only one of them can be true. So the set of conclusion diagrams can match a set of *specific* models, or all of them together can be seen to satisfy a model which, in the sense defined by Stenning and Oberlander [57], has *limited abstraction*.

In order to make the meanings represented by the set of conclusion diagrams more easily accessible to the user, we can also exploit Stenning and Oberlander's notion of a *minimal case of animation*, by dynamically transforming one conclusion diagram into another in a cyclical fashion. This is defined here as the following sequence of diagram transformations:

**10.3.3.** DEFINITION. (conclusion displaying cycle)
*Let $G = (P, P')$ a G-morphism, where, $P' = \{P'_0, P'_1, ..., P'_{m-1}\}$. A conclusion displaying cycle of G is a diagram transformation sequence, from $P'_i$ to $P'_{mod(i+1,m)}$, $i = \{0, 1, ..., m-1\}$*

## 10.3.5 Examples

In this section, some examples are presented to provide further clarification of the above formal description of diagrammatic reasoning, and to show how it might be used in solving real problems.

### Using Euler circles to solve syllogisms

Consider the syllogism in which *All A are B* and *Some C are not B* are premisses and one of the valid conclusions is *Some C are not A*. Using Euler circles, this relation between the premisses and the conclusion can be represented in Figure 10.16(2). Now I describe how this conclusion can be reached on the basis of the formal approach which has been presented.

Suppose the underlying graphical signature is in Figure 10.16(1) and I assume that the graphical theory over this signature gives the 'common-sense' meanings to the graphical symbols, which I omit. An interpretation is in Figure 10.16(1). The premisses *All A are B* and *Some C are not B* are represented by diagrams $P_1$ and $P_2$ in Figure 10.16(2) respectively. $P_1'$, $P_2'$ and $P_3'$ in Figure 10.16(2) are the conclusion pictures which are chosen by satisfying the definition of G-morphism.

Let $S = \{in(A,B), overlapping(B,C), overlapping(C,B)\}$, the geometrical semantics of each conclusion diagram is as follows:

$$\begin{aligned}
\mathcal{A}_{P_1'} &= \{in(A,C), out(C,A)\} \cup S \\
\mathcal{A}_{P_2'} &= \{overlap(A,C), overlap(C,A)\} \cup S \\
\mathcal{A}_{P_3'} &= \{separate(A,C), separate(C,A)\} \cup S
\end{aligned}$$

According to the Definition 10.3.2, the following conclusion in the application domain is obtained under the interpretation in Figure 10.16(1):

$$\begin{aligned}
&(all_are(A,C) \ \wedge \ some_are_not(C,A)) \vee \\
&(some_are_not(A,C) \ \wedge \ some_are_not(C,A)) \vee \\
&(all_are_not(A,C) \ \wedge \ all_are_not(C,A))
\end{aligned}$$

and, since *all_are_not* implies *some_are_not* (a rule in the application domain), this implies that

$$some\_are\_not(C,A) \wedge \ (all\_are(A,C) \vee \ some\_are\_not(A,C) \vee \ all\_are\_not(A,C))$$

which tell us that *Some C are not A* is always true (see the animation diagram in Figure 10.16(2)) and this is also the conclusion obtained by means of the traditional way of using Euler circles to solve this problem.

### Using diagrams to solve resultant force

In this section, I give another example, where emergent graphical objects are used in reasoning.

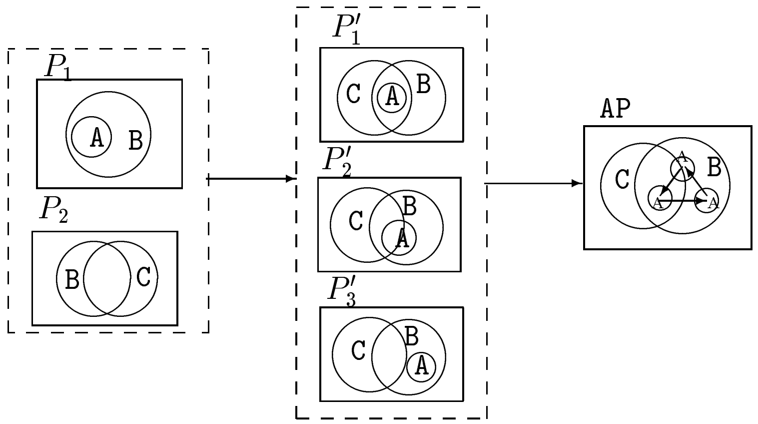
Suppose two forces $a$ (5kg and $90^o$) and b (5kg and $0^o$) apply on an object $O$, the question is what the resultant force on the object $O$ is.

Solving this problem by means of G-morphism, first I assume a graphical signature (the graphical theory over this signature is omitted), the application domain signature and an interpretation in Figure 10.17(1). The premises that force $a$ applies on the object $O$ and $b$ on $O$, are represented as two premiss diagrams in Figure 10.17(2). By satisfying the three conditions of a G-morphism, a conclusion diagram is obtained (see Figure 10.17(2)).

The graphical object labeled as $a+b$ in the conclusion diagram is an emergent graphical object, which represents the resultant force applied on the object $O$. It should be noted that such an emergent graphical object is an inevitable outcome according to my theoretical framework. Using G-morphisms the conclusion diagram may only consist of the two arrows $a$ and $b$ (see $C_1$ in Figure 10.17(2)). However, when we give the diagram a geometrical characterisation (studied in Part I) the graphical subsignature reflects all the possible graphical objects in the diagram. For

| | | |
|---|---|---|
| $\Sigma$ | $\mathcal{S}$ | $Circle;\ ...$ |
| | $\mathcal{F}$ | $A,\ \ B,\ \ C\ \ :\ Circle, ...$ |
| | $\mathcal{P}$ | $in,\ \ overlap\ :\ Circle \times Circle$<br>$out,\ \ separate\ :\ Circle \times Circle$ |
| $\Sigma'$ | $\mathcal{S}'$ | $Set,\ ..$ |
| | $\mathcal{F}'$ | $A,\ \ B,\ \ C\ \ :\ Set,\ ...$ |
| | $\mathcal{P}'$ | $some\_are\_not\ \ :\ \ Set \times Set$<br>$all\_are,\ \ all\_are\_not\ \ :\ \ Set \times Set,\ ...$ |
| $\mathcal{I}$ | $\mathcal{I}_{\mathcal{S}}$ | $Circle \mapsto Set$ |
| | $\mathcal{I}_{\mathcal{F}}$ | $A \mapsto A,\ \ B \mapsto B,\ \ C \mapsto C$ |
| | $\mathcal{I}_{\mathcal{P}}$ | $out,\ \ overlap \mapsto some\_are\_not$<br>$in \mapsto all\_are,\ separate \mapsto all\_are\_not$ |

*(1) An interpretation from $\Sigma$ to a syllogism signature $\Sigma'$.*

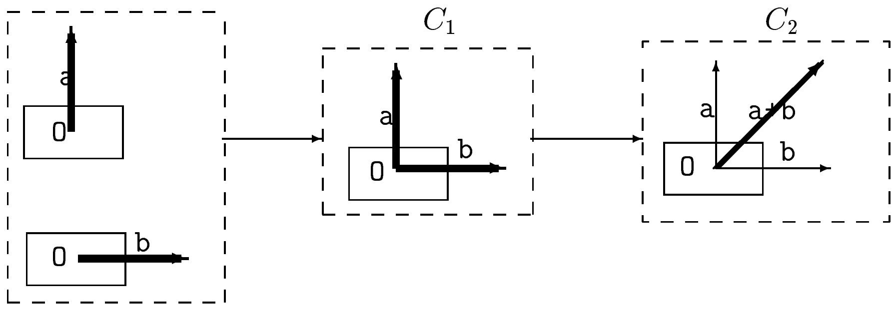



*(2) The G-morphism. AP is a representation of the animating diagram of the conclusion diagrams.*

Figure 10.16: *An example of using G-morphism to solve a syllogism.*

| | | |
|---|---|---|
| $\Sigma$ | $\mathcal{S}$ | $Box,\ Arrow,\ ThinArrow,...$ |
| | $\leq$ | $ThinArrow\ \leq\ Arrow,\ ..$ |
| | $\mathcal{F}$ | $O\ :\ Box,\ (A_1,a),\ (A_2,b)\ :\ Arrow,$<br>$(TA_1,a),\ (TA_2,b)\ :\ ThinArrow$<br>$diagonal : Arrow \times Arrow \rightarrow (Arrow, ThinArrow, ThinArrow)$ |
| | $\mathcal{P}$ | $box\_arrow\_connected : Box \times Arrow$<br>$angle\_of\_arrow : Arrow \times Real$<br>$length\_of\_arrow : Arrow \times Reals\ ..$ |
| $\Sigma'$ | $\mathcal{S}'$ | $Object,\ Force,\ Component,...$ |
| | $\leq$ | $Component\ \leq\ Force,\ ...$ |
| | $\mathcal{F}'$ | $O\ :\ Object,\quad a,\ b\ \ :\ \ Force$<br>$resultant : Force \times Force \rightarrow (Force, Component, Component)$ |
| | $\mathcal{P}'$ | $force\_on\_object : Object \times Force$<br>$orientation\_of\_force : Force \times Real$<br>$quantity\_of\_force : Force \times Real$ |
| $\mathcal{I}$ | $\mathcal{I}_{\mathcal{S}}$ | $Box \mapsto Object,\ Arrow \mapsto Force,\ ThinArrow \mapsto Component,\ ..$ |
| | $\mathcal{I}_{\mathcal{F}}$ | $O \mapsto O,\ (A_1,a),(TA_1,a) \mapsto a,\ (A_2,b),(TA_2,b) \mapsto b,$<br>$diagonal \mapsto resultant$ |
| | | $box\_arrow\_connected \mapsto force\_on\_object$<br>$angle\_of\_arrow \mapsto orientation\_of\_force$<br>$length\_of\_arrow \mapsto quantity\_of\_force$ |

*(1) An interpretation between $\Sigma$ and $\Sigma'$.*




*(2) The G-morphism for the resultant force problem.*

Figure 10.17: *An example of using diagrams to solve resultant forces applied on objects, where emergent graphical objects are used in the problem solving.*

this case, the subsignature for the conclusion diagram not only contains arrows $a$ and $b$ but also the emergent object formed by applying the graphical function, *diagonal*, to $a$ and $b$. By drawing it explicitly, an expected conclusion diagram is obtained (see $C_2$ in Figure 10.17(2)).

In this graphical signature, there is a special kind of arrow, $ThinArrow$ which is a subsort of $Arrow$, corresponding to $Component$ forces which is a subsort of $Force$ in the application domain. The graphical function *diagonal* produces a diagonal line which meanwhile changes the pair of arrows in its domain into thin arrows. This prevents the conclusion diagram from producing infinite arrows by applying the function *diagonal* to each new arrow and the existing arrows.

In the graphical signature in Figure 10.17(1), labels $a$ and $b$ relate both arrows and thin arrows. The full representations of the four arrows are $(A_1, a)$, $(A_2, b)$, $(TA_1, a)$ and $(TA_2, b)$. $(A_1, a)$ and $(TA_1, a)$ are interpreted as force $a$, and $(A_2, b)$ and $(TA_2, b)$ as $b$. This example also shows that it is not necessary to preserve the sorts of the graphical objects in a G-morphism. It suffices to preserve the labels of the graphical objects (*e.g.* an arrow $(A_1, a)$ is mapped to a thin arrow $(TA_1, a)$). However, in this case, if the mapping changes the size or orientation of the arrow it won't be a G-morphism any more for that change does not satisfy the consistency condition.

# 11 Conclusion of Part II

The meanings of pictures in various uses are determined by interpretations of graphical entities which are formalised by a general notion of renaming (a signature morphism). Based on the geometrical characterisation of pictures and the notion of interpretation, a logical (and formal) understanding of visual communication is given as a consequence relation which characterises how the basic facts obtained from a picture under an interpretation are used.

This formal understanding suggests how pictures can be correctly used in communication, which is explained by studying three semantic properties of interpretation. Pictures convey completely the basic facts in the application domain, when they are used with a sound and conservative interpretation; they correctly represent part of the real world when being used with a sound interpretation; they may carry new information to the application domain when being used with an interpretation which is only consistent.

A visual communication environment is presented on the basis of the formal understanding. The distinction between this environment and others consists in its interpretation mechanism. This makes it possible to associate pictures with various meanings according to the user's intention. In this sense, this environment is more general. Types of visual communication supported by this environment include: illustrative visual communication, demonstrative visual communication and reasoning with diagrammatic representations.

Illustrative visual communication is the basic form of visual communication and many existing visual communication systems can be put into this form. By designing an illustrative visual communication system on the basis of my visual communication environment, the important aspect is that users are allowed to choose graphical representations.

Demonstrative visual communication is more complex because apart from graphically illustrating the facts in the application domain, graphical inference is used substantially to explain the issues in the application domain. A system GAR is presented for verifying this idea. GAR, which works on a very small picture description language in the current stage, successfully realises demonstrative visual communication. This has been presented by examples which show how the user interprets graphical entities as set operations and how GAR answers various queries from the user by stepwise graphical inference.

The central idea about GAR, as well as the interpretation framework presented in this thesis, is that interpretation is not fixed. Similar claims can be seen also in other people's work. For example, "The interpretation of the edges is left open-ended." in the logical study of Higraphs [27] and "Interpretation can be given by users." in the COOL system [32]. But if we consider Higraph or the visual tool of the COOL system within the framework presented in this thesis, both of them are systems with predefined interpretations. Higraph has syntactic structure and semantic interpretation, where edges represent binary relations. "Leave the interpretation open" means that at one occasion, an edge represents one binary relation (*e.g.* $P_1(x, y)$ means dog $x$ bits animal $y$), in another occasion, it represents another binary relation (*e.g.* $p_2(x, y)$ means $y$ buys equipments from $x$). So does the COOL system. We may describe this as: the interpretation is *relatively* arbitrary.

Pineda's system GRAFLOG [52] was also aimed in the direction that the interpretation is given by the user. In GRAFLOG, users can draw pictures and write prolog programs to assign meanings to the pictures. For example, the user draws a square (labeled as *English*) and a triangle (labeled as *Luis*) which is inside the square, and then she writes the following prolog sentences:
subject(x) :- square(x).
student(x) :- triangle(x).
study(x,y) :- student(x), subject(y), in(x,y).
After these, she may ask *GRAFLOG* questions like: study(Luis,x)? and the system's answer is: x= English. The difference between GRAFLOG and GAR in the aspect of interpretation is that in the GAR system, there is an interpretation mechanism which supports users to give their interpretations and guarantees the syntactical correctness of users' interpretations, but this is not in the GRAFLOG system. Although Pineda's formal framework allows the expression of certain properties of an interpretation, the notion of an interpretation as such is not systemantically addressed: neither syntactic correctness nor semantic appropriateness can be defined.

The main work involved in understanding reasoning with diagrammatic representations is the notion of G-morphisms. A G-morphism is a meaning-preserving transformation relationship between pictures and can be used to represent inferential problems in the application domain. Further work on this subject will involve designing a system to support reasoning with diagrammatic representations. This will be developed on the top of the GAR system by extending the picture description language, implementing G-morphisms and animating the conclusion pictures.

The current definition of the G-morphism has its limitations. The example of *counting the tangled forest*, demonstrated by the system *BITPICT* [19], can be viewed as a reasoning with diagrammatic representations. In the example, a picture of a tangled forest is mapped to another by mapping a curve to a dot and then dots to Roman numerals which is the number of the trees in the tangled forest. Though this mapping can be described in my framework by choosing a proper picture description language and interpretation, we cannot get each of the conclusion diagrams by just looking for them to meet the definition of the G-morphism. In order to solve problems like this, a framework is needed for putting mapping rules in a G-morphism according to the application. It would be nice to develop such a framework on the basis of the current version of the G-morphism.

From the experience in the design and implementation of GAR and developing the further theoretical treatment for reasoning with diagrammatic representations, the current formal framework is quite general for understanding and implementing visual communication systems. However, it has its limitations. For example, I have not considered the use of graphical entities to represent subject matter such as events and actions as used in various visual programming languages and graphical interfaces, but rather have concentrated on the use of pictures to help users understand how problems in an application domain are solved by visual communication. It is not clear how the uses of graphical objects to represent actions *etc.* should be related to the kinds of visual communication I am considering.

# Part III

# Picture Specifications

# 12

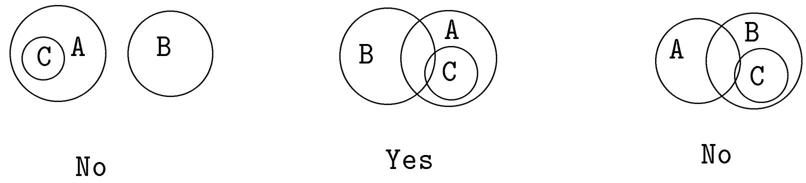
# Introduction to Part III

In Part II, I have studied picture semantics by the investigation of interpretations. In this part, I study giving pictures semantics by means of picture specifications. Picture specification techniques allow certain types of pictures to be organised together to represent a class of objects in an application domain.

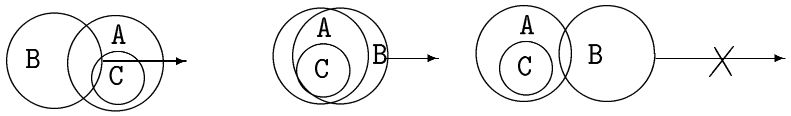
First, we look at an example to get an impression of (1) how to give pictures meanings using specifications and (2) how a class of specified pictures can be used in visual communication. Imagine that we want to use a picture to represent sets $A$, $B$ and $C$ such that $A \cap B \neq \emptyset$ and $C \subset A$. Instead of giving an explicit interpretation (*e.g.* *Circle* as *Set, inside* as $\subset$ and *overlapping* as $\lambda xy(x \cap y \neq \emptyset)$) as we studied in Part II, here, we specify a class of pictures *Three-Sets-Relation* to represent these three sets. Pictures in class *Three-Sets-Relation* consist of three circles and satisfy certain geometrical constraints. I.e:

```
Class Name:   Three-Sets-Relation
Components:   A:  Circle
              B:  Circle
              C:  Circle
Constraints:  A and B overlap & C is inside A.
```

Having specified the new class Three-Sets-Relation, the underlying graphical supporting system should be able to recognise whether or not a drawing is an object in this class (I call such ability *concept checking*) and protect the drawing from those modifications which destroy the constraints (this ability is called *constraint maintenance*). For instance, the system can tell users whether or not the following three pictures represent $A \cap B \neq \emptyset$ and $C \subset A$ by concept checking, where *Yes* means the picture represents the proposition and *No* means it does not.

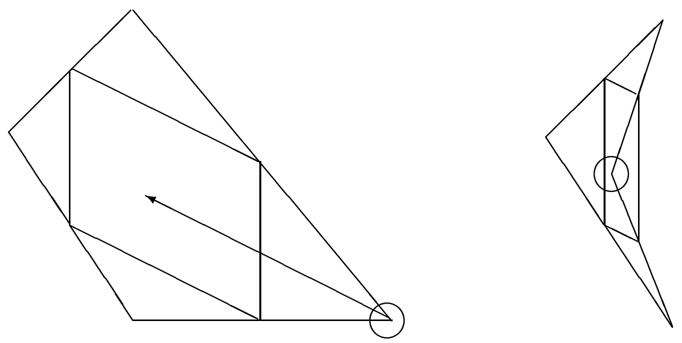



Furthermore, by means of constraint maintenance, alternative graphical representations of this proposition can be demonstrated by changing the size and position of each of the objects in a picture. For instance, we may change the position of the circle B by dragging it towards the right. When it is in the position showing in the last picture below, further dragging in the same direction is not possible without destroying the constraint (*A and B overlap*).




This behaviour of pictures in visual communication is analogous to the use of data-types in a typed programming language. In a typed programming language, there are some system built-in data types like *Integer, Real, Character.* while in the graphical system, there are some built-in classes of pictures like *Line, Circle, Rectangle.* In the programming language, users can build their own data types, like specifying a data type *Age* to range over a subset of the natural numbers with a smallest number 0 and a largest number, say, 200, while in the graphical system, users also can specify their own classes of pictures such as Three-Sets-Relation. If a variable $A$ is declared to have *Age* as its type, the system will maintain its type. For instance, $A :=$ -5 will be treated as an illegal sentence (this is similar to concept checking in the graphical system) and if $A = 190$ then $A := A + 12$ won't be executed by the system since the result would no longer be a member of the set *Age* (this is similar to constraint maintenance in the graphical system).

The main purpose of introducing data types into programming languages is to prevent various errors. However, introducing classes into graphical systems is aimed at using pictures as a meaningful mode of interaction with computers so that a kind of visual communication can be realised. Using pictures in such a meaningful way started with ThingLab [6]. For example, the validity of the quadrilateral theorem can be demonstrated in ThingLab. The quadrilateral theorem asserts that the quadrilateral formed by connecting the four mid-points of the edges in an arbitrary quadrilateral is a parallelogram. First, ThingLab helps the user to specify a class of quadrilaterals with connected midpoints. Then the user can change the quadrilateral by dragging one of its vertices and the system adjusts the picture through maintaining the mid-points constraint. Because arbitrary changes of the quadrilateral do not effect the parallelogram property, the validity of the theorem is strongly suggested.



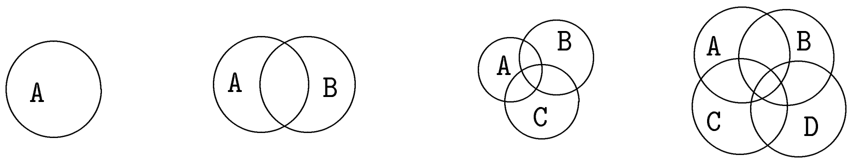
As has been stated, there are two main aspects to this use of pictures in communication, one is specifying classes of pictures and the other is constraint maintenance. It is the former which gives meanings to pictures therefore this is what is studied in this part. As mentioned in Section 1.3, specifying a class of pictures is not to assign meanings to the picture class directly, but to specify the syntactic structure of the picture class. However, in practice, a class of pictures is always specified for representing a subject matter of an application domain. Thus the meanings of the subject matter are implicitly embedded in the syntactic structure of the picture class. Therefore, the picture specification technique is considered as an approach to pictures semantics.

To specify a class of pictures we need: (1) to specify the components of the class (*e.g.* three circles in the class Three-Sets-Relation) and (2) the constraints on the components (*e.g.* two circles overlap and the third circle is inside another one in the class Three-Sets-Relation). The existing techniques for picture specifications are mainly concerned with specifying the constraints (which, together with the constraint maintenance has become a specialisation [34]). As to say how to specify the structure of the components, only one possibility has been considered so far, *i.e.* putting several objects together (as we put three circles together in the class *Three-Sets-Relation*).

Though putting components together covers many applications, there are some real world problems which cannot be naturally represented by means of this procedure. For instance, it is hard to represent the following proposition by means of putting components together.

$$A\ is\ a\ set\ of sets\ satisfying:\ \forall X, Y \in \mathcal{A}(X \cap Y \neq \emptyset).$$

The reason is that the number of the elements in $\mathcal{A}$ is not fixed. Though all the following pictures are correct in the sense that they are objects of the class which represents the set $\mathcal{A}$,




such a class cannot be specified unless a *list* structure (or *set* structure) is introduced. The structure of Venn Diagrams is another example. A Venn Diagram consists of an arbitrary number of closed curves (see the definition of Venn Diagram given by [61]). Such a structure cannot be described by combining a fixed number of closed curves. Therefore, it is natural to describe the class *Venn-Diagram* as consisting of pictures whose components are represented as a *list* of closed curves.

There is another kind of picture class which cannot be described by simply putting components together. As a simple example, one may describe the top of a table as being either a circle or a square; in other words, the class *Top-of-table* is meant to be the disjoint union of the classes *Circle* and *Square*. This means a disjunctive structure *or* is needed for such an application. Furthermore, if a set of pictures is intended to represent a set of such table tops, then a more complicated structure which combines *list* and *or* together is needed. Looking at more applications, we realise that the structure obtained by just putting components together is far less than is required in visual communication, and that it is important to create more powerful picture specification languages for visual communication.

Besides the expressive power of a picture specification language, it is also important to emphasize the naturalness and understandability of the method for specifying pictures — important considerations for users who have to define classes and use them correctly. For instance, complicated classes should be specifiable in a structured way. As in the use of a typed programming language, a user may first define some data structures and then combine them to get a more complicated data structure, instead of defining the complicated one from scratch. In the use of pictures, it is also helpful to allow users to build their classes in such a way. For instance, a designer may begin by specifying *Cooker, Table etc.* and then specify *Kitchen* in which some of the components are those previously specified. In this way, she may specify more classes like *Bath room, Living room etc.* and finnaly, perhaps, a more complicated class *House* can be specified on the basis of all the specified classes. It is clear that in order to provide such functionality, the picture specification technique should be able to manipulate *classes* (in other words, treat *classes* as first class citizens) instead of only working with the system's built-in classes. However, the existing picture specification techniques do not satisfy this need.

Thus, there are several requirements to be met by a picture specification language. Firstly, it should allow a natural and clear description of the structure of pictures, which in some cases is rather complicated and needs *powerful mechanisms.* Secondly, it should allow one to construct classes of pictures in a *structured way* so that one may construct more complicated classes from existing ones and understand the former based on the latter. Thirdly, the language should have a *simple and coherent semantics* on which the understanding is based. Finally, but no less importantly, the graphical support system should allow the user to define and use picture classes easily without requiring programming expertise.

To meet the above requirements, I present a new approach to the construction of specifications of classes of pictures. This approach provides us with a rich specification mechanism in which various kinds of picture structures can be defined in a clear and natural way.

My approach is based on a type-theoretic framework. The reason for using a type-theoretic approach instead of other formal specification approaches is that classes of pictures and their properties should be treated as first class citizens to meet our requirements. More specifically, there are functional operations defined on the set of classes to produce new classes. This can be naturally achieved by a type-theoretical approach. I will explain this in more detail in the later sections.

In the following, I will use *pictorial concept* instead of *class*. A pictorial concept in this thesis not only has a class of pictures as its objects but also has various predicates as its attributes. This makes it clear how classes of pictures are used in visual communication. For example, consider a kitchen design. The client may give some requirements which specify what things are inside the kitchen (*e.g.* a cooker, a refrigerator *etc.*) and what conditions the things should satisfy (*e.g.* the distance between the cooker and the refrigerator should not be less than 1 meter). The client's requirements together with other common knowledge about kitchens may be used as a specification of the class of kitchens. But this is only the start of the design process. There are many particular 'kitchens' which satisfy the specification. We may conceive of the designer's task as that of choosing one of these. The one chosen is different from others in the same class not because it has a different specification but because it has different attributes. The attributes defined over the class *Kitchen* can be the area of a kitchen, the length and width, the position of the dinner table, the geometrical shape of the table, the kitchen *etc.* It is the values of the attributes which distinguish graphical objects from each other within the same class and it is also the attributes which make communication (*e.g.* the kitchen design) meaningful and interesting. Some attributes can be recognised by looking at the picture (*e.g.* the geometrical shape of the table) and others must be measured or calculated (*e.g.* the size). It is nice if the visual communication support system allows the user to specify attributes over classes and if it answers the user's queries about them. This leads me to consider a *class* and a set of *attributes* together as a *pictorial concept*.

In the rest of this part, firstly, a type-theoretic framework is introduced and some motivation is given for choosing a type-theoretic approach to pictorial concepts. Secondly, pictorial concepts and the ways of constructing them are studied on the basis of the type-theoretic framework. Thirdly, applications are discussed. An experimental system CSGS (a Concept-Supporting Graphical System) is described, to verify the theoretic framework. CSGS supports the users' definition of pictorial concepts without requiring programming expertise. Three kinds of visual communication on the basis of picture specifications are discussed; they are geometric constraint maintenance, design and computational art.

# 13 A type-theoretic approach to pictorial concepts

In this section, I introduce a type-theoretic framework which is used to describe and construct pictorial concepts. Firstly, I explain why type theory is used as the formal tool, and then I present the type-theoretical framework.

## 13.1 Motivation

Type theories are formal systems based on typed lambda-calculi, studied by logicians with interests in the foundations of mathematics and also further developed by computer scientists with interests in programming and specification languages, and theorem-proving (*c.f.* [43] [20] [13] [3] [37] [39]). Type theories give rise to uniform languages with both computational power (as a rich functional programming language) and logical power (as a logic by the Curry-Howard principle of propositions-as-types [14] [31]) and have hence been studied recently as powerful uniform languages for programming, specification and theorem-proving (*c.f.* [50] [38] among others). There are already many implementations of proof development systems and program development environments based on type theories, which have attracted growing interest in computer science (*c.f.* [12] [36]).

A notable feature of type theory, as compared with other programming or specification languages, is that it can provide strong expressive power and abstraction mechanisms to modularise various entities under consideration. It is this aspect that has been exploited to a large extent (in for example, [38], [39]) to develop a theory of program specification and development. Our development of a type-theoretic approach to pictorial concepts and concept construction has been much inspired by this. The ideas from type theory have led to a simple but important notion of pictorial concept and a rich language for natural and clear description of pictorial concepts in which many sophisticated picture structures can be properly characterised by means of the type structures. The standard (operational) semantics of type theory based on its proof-theoretic results (*c.f.* [43]) gives a simple and direct understanding of pictorial concepts and what is meant by a picture satisfying a concept, as we shall explain.

In our theory, a *pictorial concept* consists of a type (picture type) and a predicate (a propositional function in type theory) over the picture type. The picture type gives the structure of (the representations of) the pictures in the concept and the predicate specifies the logical constraints that any picture in the concept should satisfy. In particular, we introduce several useful *concept operations* by means of which one can define various pictorial concepts in a structured way. These concept operations take pictorial concepts as arguments and generate new concepts whose pictures have more sophisticated structures or satisfy new logical constraints. In particular, besides operations that put concepts together and that narrow down concepts by constraints as mentioned above, concept operations corresponding to disjoint union and list-structures are examples which show that more concept operations can be defined in the type-theoretic framework. Further ways of using dependent types and type universes in type theory to develop other useful concept operations (*e.g.* structure sharing and parameterised concepts as discussed in [38] in the context of program specification) are also briefly discussed. It is shown systematically what kinds of *attributes* concerning a concept can be defined and used in applications or in defining new concepts. Special features of pictorial concepts are discussed and some interesting structural relationships between concepts are analyzed. One of the benefits is that, in the type-theoretic framework, concepts are first-class citizens over which operations can be defined: this allows a more abstract treatment of pictorial concepts than in many other systems (where only *basic primitives* are first-class) and gives a good basis for the design of the supporting system.

## 13.2 The type theory

In this section, I give an informal description of the type theory used in this part (for a formal definition, see Appendix B). The type theory is essentially the higher-order $\lambda$-caculus [21] extended by (predicative) type constructors for product types, disjoint union and list types together with several basic types.

Intuitively, one may think of a *type* as a set. One of the main assertions one can make in a type theory is that '$a$ is an object of type $A$', usually denoted as $a : A$. The class of types in our type system contains the following types and is closed under several type operations (see Figure 13.1).

1. Some basic data types, for example, the type of natural numbers $N$ and the type of floating point numbers $Real$.
2. *Logical propositions* are types. In a type theory, we use the Curry-Howard principle of propositions-as-types [31] to introduce logical formulas as the types of their proofs. A proposition $P$ is *provable* if it has a proof, *i.e.* there exists $p$ such that $p$ is an object of $P$.
3. All propositions belong to the type $Prop$. We only point out that $Prop$ is (a type universe) closed under the logical operations ($P \supset Q$, $P\&Q$, $P \vee Q$, $\forall x :$

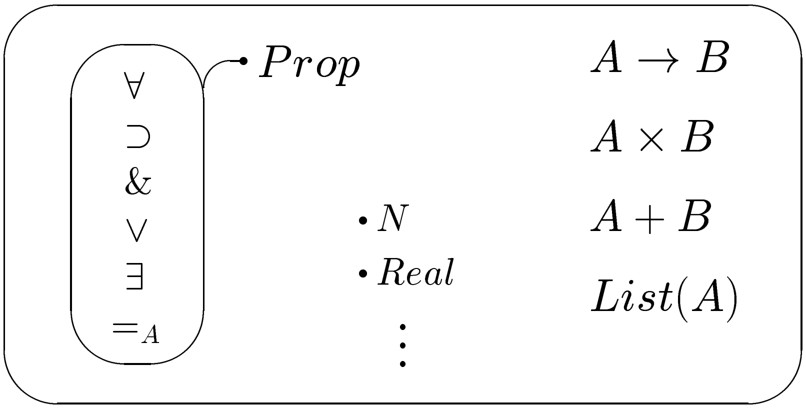



Figure 13.1: Types in the type system.

$A.\ P(x)$ and $\exists x : A.\ P(x)$) and an equality ($a =_A b$, so-called Leibniz equality[1] over each type $A$. These provide a consistent (higher-order intuitionistic) logic embedded in the type system.

The class of types is closed under the following type constructors:

- *Arrow types*:
  If $A$ and $B$ are types, so is the arrow type $A \to B$, which is the function space whose objects are functions from $A$ to $B$. Functions are introduced as $\lambda$-expressions whose evaluation rule is the usual $\beta$-conversion rule, *i.e.* $(\lambda x.b(x))(a) = b(a)$.
  As an example, for any type $A$, $A \to Prop$ is a type, the type of the predicates (propositional functions) over $A$.
- *Product types*:
  If $A$ and $B$ are types, so is the product type $A \times B$, whose objects are the pairs $(a, b)$, where $a$ and $b$ are objects of type $A$ and $B$, respectively. Type $A \times B$ intuitively stands for the set $\{\,(a,b) \mid a \in A \text{ and } b \in B\,\}$. There are two elimination operators (projections) $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$ which satisfy the definitional equalities $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$.
- *Sum types*:
  If $A$ and $B$ are types, so is the sum type $A+B$, whose objects are of the form $\mathbf{inl}(a)$ and $\mathbf{inr}(b)$, where $a$ and $b$ are objects of type $A$ and $B$, respectively. $A+B$ intuitively stands for the disjoint union of $A$ and $B$. **inl** and **inr** are the injections from $A$ to $A+B$ and from $B$ to $A+B$, respectively. Functions with $A+B$ as domain can be defined via an elimination operator **case** which satisfies the evaluation rules $\mathbf{case}(f, g)(\mathbf{inl}(a)) = f(a)$ and $\mathbf{case}(f, g)(\mathbf{inr}(b)) = g(b)$, where $f$ is a function with $A$ as domain and $g$ is a function with $B$ as domain.
- *List types*:
  If $A$ is a type, so is $List(A)$, the type of the finite lists of objects of type

[1]Leibniz equality says that two objects (of the same type $A$) are equal if and only if they are indiscernible from each other with respect to all predicates.

$A$. There are the usual list constructors **nil** and $\mathbf{cons}(a, l)$. An elimination operator $\mathbf{rec}_{List}$ allows one to define functions with $List(A)$ as domain by case analysis as follows:

$$\begin{aligned} F(\mathbf{nil}) &=_{\mathrm{df}} c, \\ F(\mathbf{cons}(a, l)) &=_{\mathrm{df}} f(a, l, F(l)). \end{aligned}$$

where $c$ is an object of some type $C$ and $f$ is a function that takes three arguments of type $A$, $List(A)$ and $C$ respectively, and return an value of type $C$. The function $F$ thus defined is formally denoted by $\mathbf{rec}_{List}(c, f)$ and satisfies the evaluation rules $\mathbf{rec}_{List}(c, f)(\mathbf{nil}) = c$ and $\mathbf{rec}_{List}(c, f)(\mathbf{cons}(a, l)) = f(a, l, \mathbf{rec}_{List}(c, f)(l))$.
As an example, one can define the function *length* of type $List(A) \to N$, which takes a list as its argument and returns the length of the list as result:

$$\begin{aligned} length(\mathbf{nil}) &=_{\mathrm{df}} 0, \\ length(\mathbf{cons}(a, l)) &=_{\mathrm{df}} length(l) + 1. \end{aligned}$$

Formally, *length* is defined as

$$\mathbf{rec}_{List}(0, \lambda x : A\ \lambda l : List(A)\ \lambda r : N.r + 1),$$

which by the evaluation rules, satisfies the above equations.

Each type (except $Prop$) is inductively defined in the sense that, intuitively, the principle of (primitive) recursion and structural induction is reflected by the elimination rule(s) associated with its elimination operator(s). Discussion of the proof-theoretic properties of the type system is outside the range of this thesis. We only remark that the type system satisfies the strong normalisation property (a proof can be given by extending Girard-Tait's reducibility method) and type-checking (and type inference) is decidable — there is a straightforward algorithm to check whether $a$ is an object of type $A$ for any $a$ and $A$.

# 14 Pictorial concepts

Intuitively, a pictorial concept is an abstract principle specifying a class of pictures each of which consists of certain graphical objects which have some structural relationship and satisfy certain constraints. For example, the class of points in an $n \times n$ plane can be specified by a pictorial concept *Point* whose objects are the pairs of natural numbers which are less than or equal to $n$; in other words, by this definition of *Point*, a point is represented by a pair of natural numbers $(n_1, n_2)$ which satisfies the constraints $n_1 \leq n$ and $n_2 \leq n$. Besides specifying the class of pictures belonging to it, a definition of a pictorial concept is complete only if one has also specified various attributes and how the concept relates to the other pictorial concepts and data types. A simple example of an attribute of the concept *Point* would be a function which computes the distance between two points.

In general, a pictorial concept consists of a type, which specifies the structure of (the representations of) the pictures in the concept, and a predicate over the type which specifies the logical constraints that the pictures in the concept should satisfy. More precisely, a *pictorial concept* $C$ is

$$C = (\mathcal{T}_C, \Phi_C)$$

where $\mathcal{T}_C$ is a type, called the *picture type* of $C$ and $\Phi_C$ is a predicate over $\mathcal{T}_C$ (*i.e.* a propositional function of type $\mathcal{T}_C \rightarrow Prop$), called the *logical constraints* of $C$.

A pictorial concept determines a class of (representations of) pictures. We say that a picture $c$ *is in concept* $C$ if $c$ is an object of type $\mathcal{T}_C$ and $\Phi_C(c)$ is provable (*i.e.* $c$ satisfies the logical constraints). Intuitively, a pictorial concept $C$ specifies the following class of pictures:

$$\{ c \in \mathcal{T}_C \mid \Phi_C(c) \}$$

*Concept-checking* is to check whether (a representation of) a picture is in some pictorial concept.

Based on the general form of pictorial concepts, the class of pictorial concepts is inductively generated by certain concept operations, starting from a set of basic pictorial concepts.

## 14.1 Basic pictorial concepts

The *basic pictorial concepts* are those basic kinds of graphical objects provided by a drawing system, like points, lines, circles, *etc.*[1] These basic types of graphical objects can be represented as pictorial concepts definable in the type theory. For example, we can define a basic pictorial concept of points in an $n \times n$ plane as

$$Point = (\mathcal{T}_{Point}, \Phi_{Point})$$

where

- The picture type $\mathcal{T}_{Point}$ is defined as

$$\mathcal{T}_{Point} = N \times N$$

  where, N is the type of natural numbers, and $\times$ is the product type constructor.
- The logical constraint $\Phi_{Point}$ is of type $Point \to Prop$ defined as, for any pair of natural numbers $p = (i, j)$,

$$\Phi_{Point}(p) = (i < n) \ \& \ (j < n)$$

In this way, points are represented by pairs of natural numbers. One can define various attributes of a pictorial concept; for instance, an attribute *distance* which computes the distance between two arbitrary points can be defined as a function of type $Point \to Point \to Real$.

Other basic pictorial concepts can be introduced in a similar way. For example, lines can be represented by pairs of different points (two end points), which constitute the pictorial concept $Line = (\mathcal{T}_{Line}, \Phi_{Line})$ defined as:

$$\mathcal{T}_{Line} = \mathcal{T}_{Point} \times \mathcal{T}_{Point}$$

$$\Phi_{Line}(p, q) = \Phi_{Point}(p) \ \& \ \Phi_{Point}(q) \ \& \ (\neg Eq(p, q))$$

where, $Eq$ is the equality relation over $\mathcal{T}_{Point} = N \times N$ , *i.e.* $Eq(p, q)$ means that the points $p$ and $q$ are identical.
Similarly, circles may be represented by pairs of a point (center) and a floating-point number (radius), which constitute the concept $Circle$ defined as:

$$\mathcal{T}_{Circle} = \mathcal{T}_{Point} \times \mathcal{T}_{Real}$$

$$\Phi_{Circle}(p, r) = \Phi_{Point}(p) \ \& \ (r > 0)$$

Usually, a picture type $\mathcal{T}_C$ represents the components of pictures in $C$. An exception may be made for the picture types of basic pictorial concepts. In this case, the picture type of a basic pictorial concept $C$ is the representation of the components of the pictures in $C$. For instance, $\mathcal{T}_{Circle} = \mathcal{T}_{Point} \times \mathcal{T}_{Real}$ does not mean that a

[1] In an implementation of a concept-supporting graphical system, there are choices to be made concerning which concepts should be basic. This can be allowed to be dependent on the drawing system on which the concept-supporting system is based on.

circle consists of a point and a floating point number. Here, it means that a circle can be represented as a point (standing for the center) and a floating point number (standing for the radius). Furthermore, it is assumed that there are no equivalence relations between basic pictorial concepts. Even if the definitions of two basic pictorial concepts are the same, they are different basic pictorial concepts. For instance, $Square$ can be defined as:

$$\mathcal{T}_{Square} = \mathcal{T}_{Point} \times \mathcal{T}_{Real}$$

$$\Phi_{Square}(p, r) = \Phi_{Point}(p) \ \& \ (r > 0)$$

which is the same as the definition of $Circle$, but they are not the same basic pictorial concepts, otherwise they should have the same name. For two basic pictorial concepts which have the same definition, their images in the screen should not be the same (see *drawing attributes* in Section 15.5.1) and neither their attributes (*e.g.* the area of a circle $(p, r)$ is different from the area of a square $(p, r)$).

## 14.2 Operations on pictorial concepts

In the type-theoretic framework, unlike the situation in most programming languages, pictorial concepts (as pairs described above) are first-class citizens over which functional operations can be defined. This allows us to introduce useful *concept operations* to generate new pictorial concepts. The concept operations defined below provide structured ways to define pictorial concepts and, when implemented in a concept-supporting system, support a natural approach to user-defined concepts.

The first two concept operations correspond to the usual ways of putting concepts (or picture specifications) together and extending a concept by further logical constraints (*c.f.* [6]). The concept operation **Ext** allows one to 'extend' the definition of a pictorial concept by adding more logical constraints.

**14.2.1.** DEFINITION. (**Ext**) *Let* $C = (\mathcal{T}_C, \Phi_C)$ *be a pictorial concept and* $P$ *a predicate over* $\mathcal{T}_C$. *Then* $\mathbf{Ext}(C, P)$ *is the pictorial concept defined as follows:*

$$\mathcal{T}_{\mathbf{Ext}(C,P)} = \mathcal{T}_C$$

*and, for any* $c$ *of type* $\mathcal{T}_C$,

$$\Phi_{\mathbf{Ext}(C,P)}(c) = \Phi_C(c) \ \& \ P(c)$$

The new concept $\mathbf{Ext}(C, P)$ is formed by adding a new logical constraint $P$ to the pictorial concept $C$. For example, having defined a pictorial concept *Polygon* whose pictures are represented by the finite lists of lines satisfying certain constraints, one may define a concept *Triangle* of triangles by adding a logical constraint $P$ expressing that the length of the finite list of lines is three:

$$Triangle = \mathbf{Ext}(Polygon, P).$$

The concept operation $\otimes$ is used to put concepts together to form a new concept.

**14.2.2.** DEFINITION. ($\otimes$ ) *Let $C = (\mathcal{T}_C, \Phi_C)$ and $C' = (\mathcal{T}_{C'}, \Phi_{C'})$ be pictorial concepts. Then, $C \otimes C'$ is the pictorial concept defined as follows:*

$$\mathcal{T}_{C\otimes C'} = \mathcal{T}_C \times \mathcal{T}_{C'}$$

*where $\mathcal{T}_C \times \mathcal{T}_{C'}$ is the product type whose objects are pairs $(c, c')$ with $c$ and $c'$ being objects of type $T_C$ and $T_{C'}$, respectively, and for any $(c, c')$ of type $\mathcal{T}_C \times \mathcal{T}_{C'}$,*

$$\Phi_{C\otimes C'}(c, c') = \Phi_C(c) \;\&\; \Phi_{C'}(c')$$

The new pictorial concept $C \otimes C'$ puts concepts $C$ and $C'$ together in such a way that a picture in $C \otimes C'$, represented by $(c, c')$ where $c$ and $c'$ are (representations of) pictures in $C$ and $C'$ respectively, is the picture obtained by putting the pictures $c$ and $c'$ together on the screen. For example, a picture in the concept $Line \otimes Line$ consists of two lines. In practice, this operation is often used together with the extension operation **Ext** to build up new concepts. For instance, a concept *ParLines* each of whose pictures consists of two parallel lines can be defined as

$$ParLines = \mathbf{Ext}(Line \otimes Line, Par)$$

where *Par* is the logical constraint expressing that the two lines are non-collinear and parallel.

Although **Ext** and $\otimes$ can be used to define many kinds of useful pictorial concepts, there are some others which are useful in many applications but can not be constructed only by means of these two concept operations. Here we define two other especially important concept operations, $\oplus$ and **List**.

**14.2.3.** DEFINITION. ($\oplus$) *Let $C = (\mathcal{T}_C, \Phi_C)$ and $C' = (\mathcal{T}_{C'}, \Phi_{C'})$ be pictorial concepts. Then, $C \oplus C'$ is the pictorial concept defined as follows:*

$$\mathcal{T}_{C\oplus C'} = \mathcal{T}_C + \mathcal{T}_{C'}$$

*where $\mathcal{T}_C + \mathcal{T}_{C'}$ is the disjoint union type each of whose objects is either of type $T_C$ or $T_{C'}$, and for any $c$ of type $\mathcal{T}_C + \mathcal{T}_{C'}$,*

$$\Phi_{C\oplus C'}(c) = \mathbf{case}(\Phi_C, \Phi_{C'})(c)$$

*where* **case** *is the case operator which satisfies*

$$\mathbf{case}(\Phi_C, \Phi_{C'})(c) = \begin{cases} \Phi_C(c) & \textit{if } c \in T_C \\ \Phi_{C'}(c) & \textit{if } c \in T_{C'} \end{cases}$$

The new pictorial concept $C \oplus C'$ puts concepts $C$ and $C'$ together in another way such that the class of pictures represented by $C \oplus C'$ is the (disjoint) union of the classes of pictures represented by $C$ and $C'$. For example, one may define a concept of table-tops as

$$Top\_of\_table = Circle \oplus Rectangle$$

to describe the intention that the top of a table is (represented by) either a circle or a rectangle. Another simple example is that, in a graphical system with colours, there may be several basic concepts for lines (say, red lines, blue lines, ..., and yellow lines). One can define a concept of coloured lines as $RedLine \oplus BlueLine \oplus ... \oplus YellowLine$[2].

The concept operation **List** constructs a concept whose pictures are (represented by) finite lists of the pictures satisfying the argument concept.

**14.2.4.** DEFINITION. (**List**) *Let* $C = (\mathcal{T}_C, \Phi_C)$ *be a pictorial concept. Then* $\mathbf{List}(C)$ *is the pictorial concept defined as follows:*

$$\mathcal{T}_{\mathbf{List}(C)} = List(\mathcal{T}_C)$$

*where,* $List(\mathcal{T}_C)$ *is the type of finite lists of objects of type* $\mathcal{T}_C$*, and for any list* $l$ *of type* $List(\mathcal{T}_C)$*,*

$$\Phi_{\mathbf{List}(C)}(l) = \begin{cases} \mathbf{true} & \textit{if } l = \mathbf{nil} \\ \Phi_C(a) \& \Phi_{\mathbf{List}(C)}(l') & \textit{if } l = \mathbf{cons}(a, l') \end{cases}$$

A list in $\mathbf{List}(C)$ represents the picture which consists of the pictures represented by the elements in the list. The empty list in $\mathbf{List}(C)$ represents the 'empty picture' with nothing shown on the screen. For example, a concept of polygons may be defined as

$$Polygon = \mathbf{Ext}(\mathbf{List}(Line), P)$$

where $P$ is the logical constraint expressing that the lines in the list should form a polygon. One can define a concept of concentric circles in a similar way. Pictures consisting of indefinite numbers of sub-pictures belonging to different concepts can be characterised by a concept of the form $\mathbf{List}(C_1 \oplus ... \oplus C_n)$. For example, $\mathbf{List}(RedLine \oplus BlueLine \oplus ... \oplus YellowLine)$ specifies a class of pictures consisting of an arbitrary number of colour lines. Some possible objects of concept $\mathbf{List}(RedLine \oplus BlueLine \oplus YellowLine)$ are shown in Figure 14.1. If one line covers another in a list (*e.g.* $l_{red}$ covers $l_{blue}$ in $[l_{red}, l_{blue}]$), the covered line may not be seen (*e.g.* $l_{blue}$ becomes invisible) or a mixing colour line appears and the two original lines are invisible (*e.g.* a purple line appears and $l_{red}$ and $l_{blue}$ are invisible), depending on the underlying drawing systems. If one wants to avoid this, one may extend the concept by a constraint which specifies that there are no covered lines in a list of this concept, namely,

$$Colour\text{-}Lines = \mathbf{Ext}(\mathbf{List}(RedLine \oplus BlueLine \oplus YellowLine), P) \quad \textit{and}$$

for all $L$ of type $\mathbf{List}(RedLine \oplus BlueLine \oplus YellowLine)$, $P(L)$ is defined as

$$\forall l_1 l_2 (L = [L_1 l_1 L_2 l_2 L_3]) \neg (cover(l_1, l_2) \vee cover(l_2, l_1)).$$

By means of the type constructors of the underlying type theory, other useful operations can be defined for structured concept construction. In particular, structure

[2]Semantically, both $\otimes$ and $\oplus$ are associative (and commutative); so brackets can be omitted.

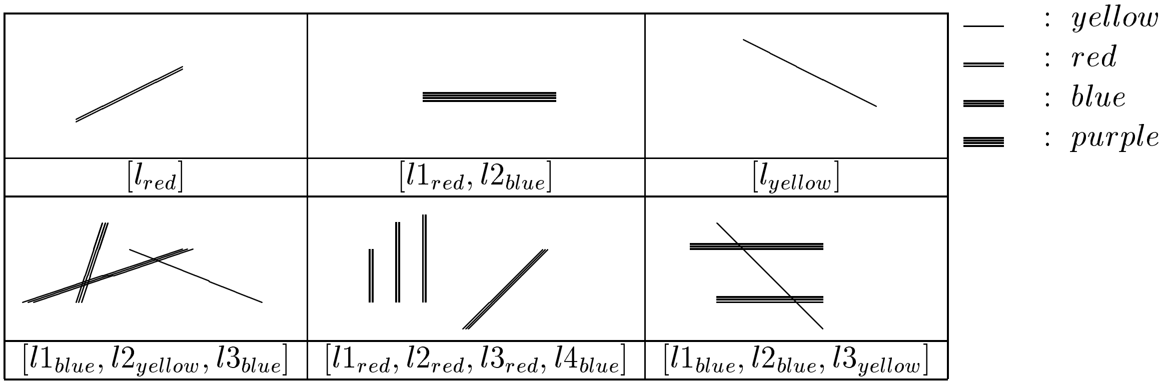



Figure 14.1: Some objects of the concept $\mathbf{List}(RedLine \oplus BlueLine \oplus YellowLine)$.

sharing of the component pictures in a more complicated picture structure can be elegantly characterised by defining a concept operation by means of the dependent sum types, which generalises the concept operation $\otimes$ to the dependent case. For instance, we might define a concept *Flat* whose pictures consist of a bed room (in *Bedroom*) and a bath room (in *Bathroom*), where the bed room and the bath room share one wall (in *Wall*). Although this example seems simple, a systematic treatment of this kind of problem is not simple. This sort of concepts can not be specified simply by $\otimes$, $\oplus$ and **List**, because a component (*i.e.* the wall) ties up both of the sub-concepts. Such structure sharing is difficult to describe in most programming languages unless special sharing constraints are available (*c.f.* the language ML [28]). Another useful mechanism for structuring concepts is parameterisation, or parameterised concepts (*c.f.* Sketchpad [60]) which in the type-theoretic framework are simply functions with pictorial concepts as values (concept operations can be viewed as special parameterised concepts), as concepts are first-class citizens in this setting. These issues will be discussed in Section 16.

## 14.3 Structural relationships between pictorial concepts

Having characterised pictorial concepts as being inductively generated by concept operations from basic concepts, some interesting structural relationships between concepts become clear, which are important in understanding, using and implementing pictorial concepts. In particular, we can define the *sub-concept* and *sub-structure* relationships as follows.

(a) *Sub-concept.* A pictorial concept $C$ is a *sub-concept* of another concept $C'$ if they have the same picture type and the pictures in $C$ are also pictures in $C'$. Typical examples of sub-concepts are concepts defined by adding logical constraints to other concepts — $\mathbf{Ext}(C, P)$ is a sub-concept of $C$. For example, the concept of triangles defined as $\mathbf{Ext}(Polygon, P)$ is a sub-concept of *Polygon*, where $P$ expresses that the length of the finite list of lines is three. In Part I, sub-sort relations were introduced between graphical sorts. The sub-sort relation is just the same as the sub-concept relation. For example, $Triangle$ is a sub-concept of $Polygon$ and in the graphical signatures introduced in Part I, the sort $Triangle$ can be a sub-sort of the sort $Polygon$.

(b) *Sub-structure.* Considering the concept operations we have introduced above, the sub-structure relation $\prec$ may be characterised as the smallest partial order $\prec$ such that

1. $C \prec C \otimes C'$ and $C' \prec C \otimes C'$;
2. $C \prec C \oplus C'$ and $C' \prec C \oplus C'$;
3. $C \prec \mathbf{List}(C)$; and
4. If $C \prec C'$, then $C \prec \mathbf{Ext}(C', P)$.

For example, *Line* stands in the sub-structure relation with *ParLines* and with *Polygon*.

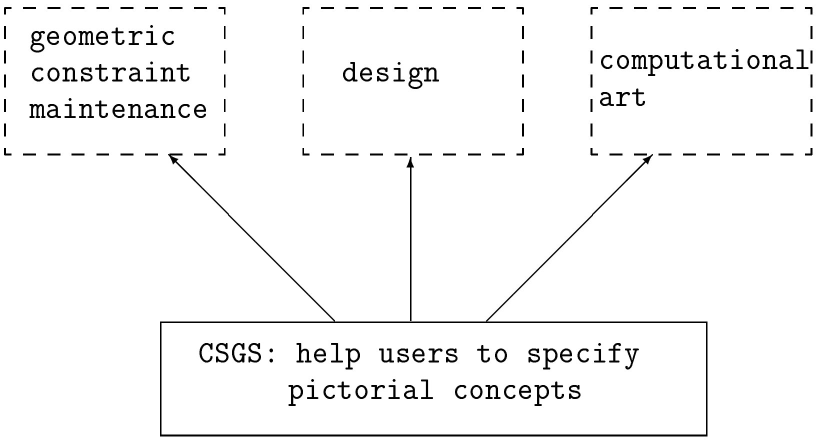
In a concept-supporting graphical system (like CSGS in Section 15.1.2), the above structural relationships indicate how new (user-defined) concepts are defined in a structured and hierarchical way.

## 14.4 Attributes

In practice, a definition of a pictorial concept is complete only if one has also specified various attributes to reflect which properties are relevant for the visual communication and to reflect how the concept relates to the other pictorial concepts and data types so that the supporting system can manipulate pictures based on such logical relations. Analogous to the method for constructing pictorial concepts, a set of basic attributes of the basic pictorial concepts can be predefined by the system designer, from which the attributes of the composed pictorial concepts can be inductively generated. An example of this is the generation of the *drawing attributes*, which relate the representation of a picture to the picture drawn on a computer screen. The objects in a pictorial concept can be drawn according to their drawing attributes. A formal definition of *drawing attributes* will be given in Section 15.5 in the context of the study of computational art.

# 15 Applications

I have presented a type-theoretical approach to pictorial concepts. In this section, some possible applications are discussed. Firstly, I present an experimental system CSGS which helps users to specify pictorial concepts without programming. The system will be presented by giving an example of how to use CSGS and discussing the implementation issues. The purpose of implementing CSGS is (1) to verify the theoretical framework and (2) to provide a platform to support various kinds of visual communication via picture specifications. Secondly, I divide visual communication via picture specifications into three categories: (1) geometric constraint maintenance, (2) design and (3)computational art. The relationship between the three kind of visual communication and CSGS is shown as follows.




CSGS is the basis because each kind of visual communication considered here needs picture specifications. Besides picture specifications, geometrical constraint maintenance and design need the functionality of constraint maintenance, and computational art needs the functionality of picture generation in a special way. The former two categories will be discussed in a rather general way and the latter, computational art, will be studied in some detail. A small system will be presented which generates simple computational art systems according to users' specifications.

## 15.1 CSGS: a Concept-Supporting Graphical System

A Concept-Supporting Graphical System, CSGS, has been designed based on the theoretical framework to support users in defining pictorial concepts in an easy and natural way. In the following, I first show an example of how to use CSGS to specify pictorial concepts. Then implementation issues are discussed.

### 15.1.1 Using CSGS: a simple example

In this section, I present an example of using the system to define a simple pictorial concept *Kitchen*. The example does not exploit the full expressive power of the theory of pictorial concepts implemented by CSGS; it is an example which only uses the concept operations $\otimes$ and *Ext*. The example is chosen for simplicity and the system works in a similar way for other concept operations such as disjoint union ($\oplus$) and list construction (**List**).

CSGS provides users with a drawing window, a dialogue window, an information displaying window and a set of function keys, as shown in Figure 15.1, where the user has defined a concept *Kitchen*. The picture in the drawing window is an example kitchen drawn by the user and the information displaying window shows *Kitchen*'s English description produced by the system. The system-generated internal representation of *Kitchen* is in Figure 15.2, which is an OBJ3 module including a concept-checking program used by the system to check whether a representation of a picture satisfies the definition of *Kitchen*.

Using CSGS, the user can define a new concept from the existing concepts using concept operations either by menu selection or by drawing pictures as examples. For example, suppose that the user has defined concepts *Cooker*, *Table-set*, *Sink*, *Refrigerator* and *Room*, and wants to specify a new concept, *Kitchen*, such that any kitchen consists of a room containing a cooker, a sink, a refrigerator and a table-set, and where the distance between the refrigerator and the cooker is not less than 5 units. Specifying *Kitchen* by menu selection, the user first clicks the operation $\otimes$ and then uses the menu [1] to select the concepts *Room*, *Table-set*, *Refrigerator*, *Sink* and *Cooker*, putting them together to form the structure of the *Kitchen* concept. (Alternatively, by drawing examples, he may draw a picture in the drawing window (see Figure 15.1) to indicate the components of a kitchen.) The system then shows in the display window the structure of the Kitchen concept as follows:

[1]Menus will appear (and then dispear) when they are needed. For instance, when the functionality of specifying a concept is chosen the menu which lists all predefined concepts will appear and then dispear when the specification is finished.

Figure 15.1: A concept-supporting graphical system.

| |
|---|
| Kitchen: |
| 1. Room |
| 2. Table-set |
| 3. Refrigerator |
| 4. Sink |
| 5. Cooker |

and asks the user to select some of them to specify further logical constraints about their relationships (with the operation **Ext**). Upon selecting number 3 and 5, the user is supplied a list of the defined attributes of *Refrigerator* and *Cooker*, which includes, for instance, the following entries.

| |
|---|
| 1. distance-refrigerator-cooker = ? |
| 2. distance-refrigerator-cooker >= ? |
| 3. distance-refrigerator-cooker <= ? |

By selecting number 2 the user is further asked to give a value (say 5 units), for the minimal distance between the refrigerator and the cooker in a kitchen. He may also add more constraints by selecting logical operators and attributes, *e.g. table-set-is-in-room* to indicate that the table-set should be inside the room, and hence complete the specification of the concept *Kitchen*[2].

[2]The underlying system allows arbitrary such attributes and constraints to be defined for and between any concepts, based on existing defined relationships. For instance, the distance between objects of two concepts can be defined in terms of existing procedures applying to the components of those concepts.

```
obj KITCHEN is
using ROOM TABLE-SET REFRIGERATOR SINK COOKER .
sort Kitchen .
op kitchen : Room Table-set Refrigerator Sink Cooker -> Kitchen .
op kitchen1 : Kitchen -> Room .
op kitchen2 : Kitchen -> Table-set .
op kitchen3 : Kitchen -> Refrigerator .
op kitchen4 : Kitchen -> Sink .
op kitchen5 : Kitchen -> Cooker .
op kitchen? : Kitchen -> Bool .
var X1 : Room . var X2 : Table-set . var X3 : Refrigerator .
var X4 : Sink . var X5 : Cooker .
eq kitchen1(kitchen(X1,X2,X3,X4,X5)) = X1 .
eq kitchen2(kitchen(X1,X2,X3,X4,X5)) = X2 .
eq kitchen3(kitchen(X1,X2,X3,X4,X5)) = X3 .
eq kitchen4(kitchen(X1,X2,X3,X4,X5)) = X4 .
eq kitchen5(kitchen(X1,X2,X3,X4,X5)) = X5 .
var X : Kitchen .
eq kitchen?(X) = room?(kitchen1(X)) and
table-set?(kitchen2(X)) and refrigerator?(kitchen3(X)) and
sink?(kitchen4(X)) and cooker?(kitchen5(X)) and
distance-refrigerator-cooker(kitchen3(X),kitchen5(X)) >= 5 and
table-set-is-in-room(kitchen2(X),kitchen1(X)) and
refrigerator-is-in-room(kitchen3(X),kitchen1(X)) and
sink-is-in-room(kitchen4(X),kitchen1(X)) and
cooker-is-in-room(kitchen5(X),kitchen1(X)) .
endo
```

Figure 15.2: The system-generated internal representation of *Kitchen*.

Figure 15.3: Concept-checking.

When the specification of a new concept finishes, the system generates the internal representation of the defined concept and its corresponding concept-checking algorithm (*e.g. Kitchen* in Figure 15.3). The new concept (*e.g. Kitchen*) can then be employed by users to define other concepts (say, a concept *House*) and by the system to check whether a drawing satisfies a concept or not.

Such *concept-checking* is an important support for the designer. During the design process, drawings are changed constantly and any change may cause a drawing to fail to satisfy the concept to which the drawing should belong. For example, the picture in Figure 15.1 is a kitchen; however, if it is changed to the picture shown in Figure 15.3, the system will then tell the user that it is no longer a kitchen because the constraint *distance-refrigerator-cooker* $>=$ 5 is not satisfied (see Figure 15.3), although it still looks like a kitchen. Usually, for one design task, a set of alternative design sketches are needed. This means that designers may draw more than one sketch for a design concept. In such a case, concept-checking also helps the designer by checking whether each of the sketches he draws is, for example, a kitchen and, if not, why not.

### 15.1.2 Implementation

The design and the experimental implementation of the concept-supporting graphical system CSGS is based on the ideas and theoretical framework described above. I describe the implementation by considering the following three aspects of the system: 1) representation of pictorial concepts; 2) implementation of concept operations and a structured approach to concept-checking; 3) further aspects of support for user-

```
1  obj D-LINE is
2  using POINT .
3  sort D-line .
4  op d-line : Point Point -> D-line .
5  op d-line1 : D-line -> Point .
6  op d-line2 : D-line -> Point .
7  op d-line? : D-line -> Bool .
8  var X1 X2 : Point .
9  eq d-line1(d-line(X1,X2)) = X1 .
10 eq d-line2(d-line(X1,X2)) = X2 .
11 var X : D-line .
12 eq d-line?(X) = point?(d-line1(X)) and point?(d-line2(X)) .
13 endo
```

Figure 15.4: The OBJ3 object representing concept D-line.

defined concepts.

### Representation of pictorial concepts

An important part of the system is the implementation of pictorial concepts. We have used OBJ3 [26] as the internal representation language for pictorial concepts. OBJ3 provides mechanisms to express modules (called *objects*), supporting abstract data types, parameterised programming and subsort relationships, which is useful in the implementation.

In the implementation of CSGS, a pictorial concept is represented as an OBJ3 object. For example, a concept of (possibly degenerate) lines is represented by the object shown in Figure 15.4.

In order to generate automatically the internal representation of user-defined concepts and the corresponding concept-checking algorithm, a concept $C = (\mathcal{T}_C, \Phi_C)$ is uniformly represented as an OBJ3 object (with a uniform notation) which consists of:

- *Object name*: the name of the OBJ3 object corresponding to the name of the concept (given by the user in the case of user-defined concepts), *e.g.* `D-LINE` in line 1 of Figure 15.4.
- *Major sort*: the sort corresponding to the picture type $\mathcal{T}_C$, *e.g.* `D-line` in line 3 of Figure 15.4.
- *Constructor*: the operation generating (the representations of) the objects of picture type $T_C$, *e.g.* `d-line` in line 4 of Figure 15.4. In this case, as described earlier, the type of `d-line` is constructed as `Point` $\times$ `Point`; line 2 lists the sorts of objects which will be used in the construction.
- *Selectors*: operations corresponding to the elimination operator(s) for the type constructor which is used in the definition of the picture type, if $C$ is defined

```
obj TRIANGLE is
using POLYGON .
sort Triangle .
op triangle : Line-list -> Triangle .
op triangle1 : Triangle -> Line-list .
op triangle? : Triangle -> Bool .
var L : Line-list .
eq triangle1(triangle(L)) = L .
var X : Triangle .
eq triangle?(X) = polygon?(polygon(triangle1(X)))
and length-of-list(triangle1(X)) == 3 .
endo
```

Figure 15.5: The representation of *Triangle*.

by $\otimes$ *etc.* (For $\oplus$, see later.) For example, `d-line1` and `d-line2` in line 5 and 6 of Figure 15.4.

- *Checker*: the operation representing the concept-checking algorithm, which corresponds to the logical constraints $\Phi_C$. For example, `d-line?` in line 7 of Figure 15.4.
- *Attributes*: Attributes are represented as operations. For a basic pictorial concept, there are a set of pre-defined attributes and, for a user-defined concept, the attributes of the concepts used to define it are all inherited. New attributes can also be defined.

### Implementation of concept operations

Based on the uniform representation described above, internal representations of pictorial concepts defined by the concept operations are generated in the following way.

- $\mathbf{Ext}(C, P)$: The internal representation of $\mathbf{Ext}(C, P)$ is obtained from that of $C$ by replacing the corresponding names respectively according to the name of the extended concept given by the user, and augmenting the definition (algorithm) of the checker as the conjunction of the new logical constraint with that for the original checker of $C$.[3] For the example of *Triangle* in Section 14.2, suppose that the constructor and selector for *Polygon* are:

  ```
  polygon : Line-list -> Polygon .
  polygon1 : Polygon -> Line-list .
  ```

  then *Triangle* is represented as in Figure 15.5.

[3]Ideally, extension by logical constraints would correspond to *subsorts with constraints*. However, the current version of OBJ3 [26] does not support this. Therefore, we have to use *checker* to handle the logical constraints and concept-checking.

```
obj TABLE is
using RECTANGLE-AND-CIRCLE-LIST .
using RECTANGLE .
using CIRCLE .
sort Table .
op table : Rectangle-and-circle-list -> Table .
op table1 : Table -> Rectangle-and-circle-list .
op table? : Table -> Bool .
var X : Rectangle-and-circle-list .
eq table1(table(X)) = X .
var X : Table .
eq table?(X) = length-of-list(table1(X)) == 1
and (circle?(head(table1(X)))
or rectangle?(head(table1(X)))) .
endo
```

Figure 15.6: The representation of *Table*.

- $C \otimes C'$: The concepts $C$ and $C'$ have the sub-structure relationship with $C \otimes C'$, as discussed in Section 14.3. The domain sort of the constructor and the range sorts of selectors can be determined by the representations of $C$ and $C'$. The object `D-LINE` in Figure 15.4 is such an example, which is the OBJ3 representation of *Point* $\otimes$ *Point*, with `POINT` representing the concept *Point*.
- $C \oplus C'$: Although $\oplus$ corresponds to the sub-structure relationship as well, there is no direct encoding for such a sub-structure relationship in OBJ3. To deal with this problem, a parameterised object `List` is used. For such a case, CSGS first produces a new object `C-AND-C'-LIST` (by means of the mechanism of *views* provided by OBJ3). Then, in the object representing $C \oplus C'$, the constructor of $C \oplus C'$ is defined as from sort `C-and-C'-list` to the sort of the concept and the checker checks whether a picture passes the checker of $C$ or the checker of $C'$. For example, the concept *Table* $=$ *Circle* $\oplus$ *Rectangle* is represented as in Figure 15.6.
- **List**$(C)$: This is represented by parameterised object `List`. The implementation is straightforward.

Based on these, the system generates automatically the internal representation of pictorial concepts and their concept-checking algorithms.

As the reader may have already noticed, concept-checking in CSGS is done in a hierarchical and structured way. To check whether (the representation of) a picture is in a concept, the system checks whether its sub-components are in the concepts which have a sub-structure relation with the concept. For example, checking whether $p$ is in $C \otimes C'$ involves checking whether the representation of $p$ is a pair $(c, c')$ and whether $c$ and $c'$ are respectively in $C$ and $C'$. This is similar for the other concept operations. This structured approach reduces the problem of concept-checking to concept-checking for the basic concepts and the extra logical constraints introduced in extensions. This natural strategy of proof search has been incorporated in the above implementation of pictorial concepts.

### More on user-defined pictorial concepts

How the user specifies a concept in CSGS has been explained through a simple example in Section 15.1. Here, we discuss further how the system supports user-defined concepts.

*Open and complete concepts.* The CSGS system provides two modes of concept specifications, called *open* and *complete*. The open mode of concept specification allows the user to develop a concept gradually in a top-down fashion. A pictorial concept specified under the open mode is a partial specification which is not allowed to be used for concept checking or to be used for specifying other pictorial concepts under the complete mode. Further specifications can be added into a partially specified pictorial concept until the definition of the concept is complete, which is made clear by selecting the *complete* mode. The default mode is the *complete* mode. Example of using these two modes in concept specification will be shown in Section 15.4.

*Defining concepts from existing ones.* The general way to define new pictorial concepts in CSGS is to select concept operations and their argument concepts from existing concepts (either by selecting from the menu or drawing example pictures). CSGS provides four functions for this, corresponding to the concept operations defined in Section 14.2 (see Figure 15.1). After the concept operation is selected, the system shows a menu of the existing concepts in the display window for the user to select the argument concepts. For operations $\otimes$ and $\oplus$, the user may also select more than two concepts, due to the associativity of these two concept operations. Since extension by logical constraints often follows one of the other three operations, we combine the extension operation with each of the others as a single step of specification (as in the case of specifying *Kitchen* in Section 15.1). Also, as the operation $\otimes$ (putting together) followed by an extension of logical constraints is often used, the CSGS implementation takes it as a default, if no other operation is selected. In the current implementation, we only consider two logical operators, `and` and `or` (see Figure 15.1). In defining attributes, the user can select one of them to connect the existing logical attributes to form new logical constraints.

Extension by logical constraints and definition of attributes are similar to the above; hence we omit detailed discussion, but only point out that specifying attributes is also important because they relate concepts to each other so that further concepts may be defined by referring to them (*e.g.* in our *Kitchen* example, the attribute `is-in` between table-sets (or cookers) and rooms is used in the definition of *Kitchen*).

## 15.2 Categories of visual communication

I have presented a concept supporting graphical system CSGS. In this section, the use of such a system to support various kinds of visual communication is considered. Visual communication via picture specifications may be divided into three kinds, (1) *geometrical constraint maintenance*, (2) *design* and (3) *computational art*. This distinction between three categories reflects the way in which we use the objects in a pictorial concept in visual communication. Geometrical constraint maintenance is to help users to realise the validity of a theorem by means of showing a number of arbitrary objects in a pictorial concept; design is looking for a 'good' object in a pictorial concept and computational art may be viewed as randomly producing objects in a pictorial concept.

## 15.3 Geometric Constraint maintenance

The validities of many abstract theorems can be demonstrated by means of geometric constraint maintenance. The basic principle is: understanding the *generality* of a theorem (which is embedded into the construction of the represented pictorial concept) by seeing a number of random objects in the pictorial concept. This is similar to random sampling in examining the quality of products.

The validity of the quadrilateral theorem demonstrated by means of Thinglab (see Section 12) is an example of this kind of visual communication. The quadrilateral theorem says that the quadrilateral formed by connecting the four mid-points of the edges in an arbitrary quadrilateral is a parallelogram. In order to help people to realise the validity of this theorem, we may specify a pictorial concept Q-T (see in Figure 15.8) to represent the Quadrilateral Theorem.

To verify the validity of the quadrilateral theorem, it should be shown that every object of the concept Q-T has the property that the inner quadrilateral of the object is a parallelogram. However, we cannot enumerate every object to verify the validity of the theorem. Instead we must be satisfied by random sampling. The supporting system shows an object of Q-T to the user. Then the user is allowed to change the object in arbitrary ways. For example, the user may change the outside quadrilateral by dragging one of its vertex and the supporting system adjusts the changed picture, maintaining the constraints, so that another object of the concept Q-T is produced (see in Figure 15.7).

## 15.4 Design

We may understand a design task as looking for a special object in the pictorial concept which specifies the product requirements. However, a pictorial concept related to a design task can be very complex. For instance, even the simple *Kitchen* concept specified in Section 15.1 is quite big compared to the concepts used in geometric constraint maintenance. Furthermore, design is a process of exploration. At the start of a design task, a designer rarely has a complete idea of the resulting model.

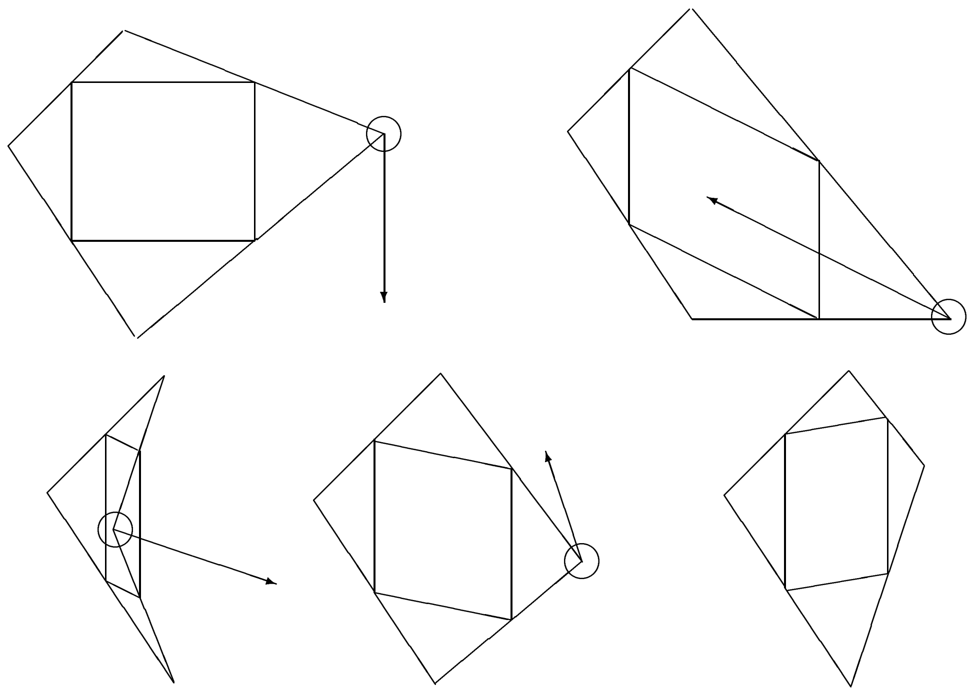


Figure 15.7: Example geometrical constraint maintenance showing the validity of the quadrilateral theorem.

Therefore, it is important to allow the user to develop concepts in a top-down style so that the details of the design may be filled in at a later stage. In order to achieve this, the CSGS system provides a special mode of concept specification, called *open* concept specification (see Figure 15.1). The open mode of concept specification allows the user to develop a concept gradually in a top-down fashion. For example, the user may want to design a concept of *House* for which there is as yet no detailed plan. Then, using the open mode, an *open concept House* can be introduced. This is implemented in the system by introducing an auxiliary concept, `PICTURE-LIST`, which is transparent to the user and is satisfied by any pictures which satisfy the currently existing concepts; its concept-checker is simply defined as *true*. CSGS generates the representation for the open concept *House* in Figure 15.9. Then, the user can refine the concept specification gradually, for example, by introducing new open concepts *Kitchen* and *Bathroom* as components of the *House* concept. A combination of top-down and bottom-up designs can be used (*e.g.* the *Kitchen* concept can be defined as shown earlier) until the definition of the concept is complete, which is made clear by selecting the *complete* mode (see Figure 15.1). Then the system deletes the corresponding part for `PICTURE-LIST` from the representation and adds the complete definition to the list of existing (complete) concepts.

Supporting the user looking for the object of the concept is not simple at all. Concept checking and constraint maintenance are the main techniques related to

```
obj Q-T is
using POLYGON4 . sort Q-t .
op q-t : Polygon4 Polygon4 -> Q-t . op q-t1 : q-t -> Polygon4 .
op q-t2 : q-t -> Polygon4 . op q-t? : q-t -> Bool .
var X1 X2 : Polygon4 . eq q-t1(q-t(X1,X2)) = X1 . eq q-t2(q-t(X1,X2)) = X2 .
var X : Q-t .
eq q-t?(X) = polygon4?(q-t1(X)) and polygon4?(q-t2(X)) and
((eq-point(p1-polygon4(q-t2(X)),mp1-polygon4(q-t1(X))) and
eq-point(p2-polygon4(q-t2(X)),mp2-polygon4(q-t1(X))) and
eq-point(p3-polygon4(q-t2(X)),mp3-polygon4(q-t1(X))) and
eq-point(p4-polygon4(q-t2(X)),mp4-polygon4(q-t1(X)))) or
(eq-point(p2-polygon4(q-t2(X)),mp1-polygon4(q-t1(X))) and
eq-point(p3-polygon4(q-t2(X)),mp2-polygon4(q-t1(X))) and
eq-point(p4-polygon4(q-t2(X)),mp3-polygon4(q-t1(X))) and
eq-point(p1-polygon4(q-t2(X)),mp4-polygon4(q-t1(X)))) or
(eq-point(p3-polygon4(q-t2(X)),mp1-polygon4(q-t1(X))) and
eq-point(p4-polygon4(q-t2(X)),mp2-polygon4(q-t1(X))) and
eq-point(p1-polygon4(q-t2(X)),mp3-polygon4(q-t1(X))) and
eq-point(p2-polygon4(q-t2(X)),mp4-polygon4(q-t1(X)))) or
(eq-point(p4-polygon4(q-t2(X)),mp1-polygon4(q-t1(X))) and
eq-point(p1-polygon4(q-t2(X)),mp2-polygon4(q-t1(X))) and
eq-point(p2-polygon4(q-t2(X)),mp3-polygon4(q-t1(X))) and
eq-point(p3-polygon4(q-t2(X)),mp4-polygon4(q-t1(X))))) .
endo
```

Figure 15.8: A representation of the quadrilateral theorem. POLYGON4 is a predefined concept standing for quadrilaterals; p$i$-polygon4 returns the $i$th vertex of a quadrilateral; mp$i$-polygon4 returns the mid-point between the $i$th and the $i+1$th vertex, and eq-point is a predicate which checks whether or not the positions of two points are close enough.

```
obj HOUSE is
using PICTURE-LIST .
sort House .
op house : Picture-list -> House .
op house1 : House -> Picture-list .
op house? : House -> Bool .
var X : Picture-list .
eq house1(house(X)) = X .
var X : House .
eq house?(X) = true .
endo
```

Figure 15.9: The representation of the open concept, *House*.

this subject. Maintaining a simple concept like the quadrilateral can help visual communication in an effective way, but maintaining a complicated one may not be practical because of the long time consumed by the computation. Besides such technical considerations, many other factors also need to be considered. For instance, whether or not the constraints should be maintained automatically by the system depends on the particular design task. For example, for circuit design, it may be very helpful to maintain constraints automatically. However, for architecture design, it may be more acceptable to maintain constraints by consulting the user (*i.e.* the designer).

Many applications can be put into this category, which include mechanical design, architectural design, user-interface design, network design *etc.* The discussion of this category is outside the scope of this thesis. It is only intended to show that the picture specification techniques developed in this part can be of some help in building a computer aided design system.

## 15.5 Computational art

In this thesis, by *computational art*, I mean works of art produced by a computer system. The basic principle adopted by many computational art systems is *drawing by chance*. On the basis of this principle, some systems draw pictures in an abstract style and others in a realistic style (see the pictures in Figure 15.10 as examples, which (1) is from [42], (2) is from [55] and (3) is from [44]).

Putting computational art into the picture specification environment, we may understand it as consisting of the following three steps: (1) specifying a pictorial concept; (2) randomly producing objects of the concept and (3) choosing the 'good' objects as the works of the art.

For instance, the works produced by one of Mandelbrot's algorithms [42] (also independently developed by Morrellet [46] and by DeVries [16]) simply consist of a set of lines (see the picture in Figure 15.10 (1) as an example). I.e. there is a pictorial

Figure 15.10: Example pictures produced by computational art systems.

concept whose picture structure is a line list without any logical constraint.

$$Line\text{-}picture = (\mathcal{T}_{Line\text{-}picture}, \Phi_{Line\text{-}picture})$$

$$\mathcal{T}_{Line\text{-}picture} = \mathbf{List}(Line) \quad \& \quad \Phi_{Line\text{-}picture} = true$$

(where a *line* has no starting point or end point). To randomly produce an object in this concept, a random number [4] is created as the length ($L$) of the list, which determines how many lines the picture has; then repeating $L$ times to draw each line. To draw a line, all the attributes related to a line need to be randomly created, like the starting point, the end point (which determine the position and the orientation of a line) and the width of the line. If there are constraints (*e.g.* the x-axis value of each line's starting point is 150), they should be incorporated into the drawing step (*e.g.* when randomly creating the starting point, only the y-axis value is created and the x-axis value is always 150 or randomly create a starting point and check whether or not the x-axis value is 150). The work *three men in a botanic garden* drawn by system *AARON* [44] in Figure 15.10 (3) can be understood in a similar way. But the pictorial concept *Three-Men-In-Garden* is a complicated concept which consists of a list of *Plant* and three *Man*. The basic pictorial concepts used in *AARON* are different from many others. Their objects are hand-drawn-like curves which make the pictures drawn by *AARON* more like 'real' drawings. In the system [55], more complicated pictorial concepts are used such as parameterised concepts (which we have not developed yet), the embedded use of list structures and there is a wide variety of basic pictorial concepts used in the system.

By understanding computational art in such a way, I will propose how to build a system which can generate computational art using the pictorial concepts specified by the user. On the basis of the system CSGS, the main thing which this system should do is *randomly drawing objects in a pictorial concept*. This leads us to consider the relationship between the representation of a picture and the picture drawn on a screen as a (drawing) attribute of the pictorial concept which the picture belongs to.

## 15.5.1 Drawing attributes

Suppose a computer screen is an $n \times n$ plane (*i.e.* a bitmap). The picture on the screen can be characterised by predicates over the pictorial concept *Point*, *i.e.* by functions of type

$$Picture = Point \rightarrow Prop$$

Usually, one may describe pictures as the predicate which represents the drawing function of the picture. For example, a line, $L = (p1, p2)$, corresponding to the

[4] A high (and low) threshold is always needed for creating a random number. But there is no upper bound for the lengths of the objects in the concept *Line-picture*. However, in practice, if the high threshold is big enough we shall be satisfied. For instance, if the random numbers are created between 0 to 2000, the results should be quite acceptable because, when the number of lines in a picture get above 2000, the density of the lines will be too big to tell one picture from another.

equation $y = ax + b$ (where $a$ and $b$ can be derived from $p1$ and $p2$) can be described by the predicate $L_{a,b} : Point \to Prop$ defined as, for any points $p = (x_p, y_p)$,

$$L_{a,b}(p) = (y_p =_{Real} ax_p + b) \;\&\; between(p, p1, p2)$$

where, the predicate *between* checks whether $p$ is within the boundary determined by $p1$ and $p2$. We can associate with a pictorial concept $C$ a *drawing attribute* which is a function of type

$$\mathrm{Drawing}_C \;:\; \mathcal{T}_C \to Picture$$

which consists of two parts and has the form: for any object $c$ of type $\mathcal{T}_C$ and any point $p$,

$$\mathrm{Drawing}_C(c)(p) = \Phi_C(c) \;\&\; \mathrm{draw}_C(c)(p)$$

where $\mathrm{draw}_C : \mathcal{T}_C \to Picture$ is the characterisation of the drawing functions for the pictures in $C$. For example, for the concept *Line*, one may define, for any $l : \mathcal{T}_{Line}$ and any point $p$,

$$\mathrm{Drawing}_{Line}(l)(p) = \Phi_{Line}(l) \;\&\; \mathrm{draw}_{Line}(l)(p)$$

where $\mathrm{draw}_{Line}(l) = L_{a,b}$ is as defined above with $a$ being the gradient of $l$ and $b$ the intercept of $l$, which can be computed from the representation $l$. The drawing attributes for the other basic pictorial concepts can be defined in a similar way.

Due to our inductive definition and understanding of pictorial concepts, drawing attributes for pictorial concepts defined by concept operations can be inductively defined using the drawing attributes of basic concepts.

**15.5.1.** DEFINITION. (inductive generation of drawing attributes)
*Let* $\mathrm{Drawing}_C$ *and* $\mathrm{Drawing}_{C'}$ *be the drawing attributes of* $C$ *and* $C'$*, respectively. According to the general form of a drawing attribute, we only have to define* draw *for the following:*

- $\mathbf{Ext}(C, P)$*: for any* $c : \mathcal{T}_C$*,*

$$\mathrm{draw}_{\mathbf{Ext}(C,P)}(c) = \mathrm{draw}_C(c)$$

- $C \otimes C'$*: for any objects* $c'' = (c, c') : \mathcal{T}_C \times \mathcal{T}_{C'}$*,*

$$\mathrm{draw}_{C\otimes C'}(c'') = \mathrm{draw}_C(c) \vee \mathrm{draw}_{C'}(c')$$

  *(Note that the logical* $\vee$ *here gives the characterisation of* 'draw $c$' and 'draw $c'$'*.)*

- $C \oplus C'$*: for any object* $c'' : \mathcal{T}_C \oplus \mathcal{T}_{C'}$*,*

$$\mathrm{draw}_{C\oplus C'}(c'') = \begin{cases} \mathrm{draw}_C(c) & \textit{if } c'' = \mathbf{inl}(c) \\ \mathrm{draw}_{C'}(c') & \textit{if } c'' = \mathbf{inr}(c') \end{cases}$$

- $\mathbf{List}(C)$*: for any* $l : List(\mathcal{T}_C)$*,*

$$\mathrm{draw}_{\mathbf{List}(C)}(l) = \begin{cases} \lambda p.\mathbf{false} & \textit{if } l = \mathbf{nil} \\ \mathrm{draw}_C(c) \vee \mathrm{draw}\mathbf{List}(C)(l') & \textit{if } l = \mathbf{cons}(c, l') \end{cases}$$

  *(So, to draw the picture represented by a list, draw all of the pictures represented by the elements in the list.)*

Sum up the above:

- An object in a pictorial concept $C$ is drawn by means of: (1) a system built-in method if $C$ is basic pictorial concept; (2) inductively according to the above definition if $C$ is a concept defined by concept operations.
- An object in a pictorial concept is found by means of: (a) randomly create an object $o$ with picture type $\mathcal{T}_C$; (b) if $\Phi_C(o)$ is true then draw $o$ else go to (a).

The above tells us that it is *theoretically* feasible to find and draw objects in a pictorial concept. But this may cost too much time because many objects randomly created may not satisfy the logical constraints. When implementing a system, it is more practical to look for an object with some guidance of the logical constraints. For instance, guided by the constraint that the x-axis value of each line's starting point is 150, we only need randomly create the y-axis value of each line's starting point and leave the x-axis value 150, instead of randomly create both x-axis and y-axis values and then checking the constraint. However, when logical constraints become complicated, more intelligent methods are needed to choose objects and for some cases, there may not be a proper way in which constraints can be considered in the choice of the objects. I leave this as a further research topic.

### 15.5.2 An experimental system: CINONG

I have implemented an experimental system CINONG which can generate simple computational art systems according to the pictorial concepts specified by users. CINONG is implemented on the basis of CSGS extended with the generation of drawing programs.

CINONG provides users an interface for specifying pictorial concepts. Specifying a pictorial concept in CINONG is similar with in CSGS. But more basic pictorial concepts and attributes than those in CSGS are introduced in CINONG for the consideration of art systems, like *line size*, *filling pattern*, *free-hand draw style*.

The result of specifying a pictorial concept is that CINONG returns a program which can draw random objects in the specified concept. For example, in order to obtain a program which can draw those line pictures produced by one of Mandelbrot's algorithms (see Figure 15.10(1)), the user needs only answer the following questions asked by CINONG.

<table>
<tr><th>Questions (CINONG)</th><th>Answers (User)</th></tr>
<tr><td>The name of the concept?</td><td>Line-picture</td></tr>
<tr><td>Choose a concept operator:<br>T(⊗), O(⊕), L(List) or D(Default)</td><td>L</td></tr>
<tr><td>Choose pictorial concept(s) for the operator.<br><table><tr><td>1</td><td>Line-Segment</td></tr><tr><td>2</td><td>Line</td></tr><tr><td>3</td><td>Circle</td></tr><tr><td>4</td><td>Box</td></tr><tr><td></td><td>...</td></tr></table></td><td>2</td></tr>
<tr><td>Any constraints? Y/N</td><td>N</td></tr>
<tr><td>Which drawing style do you want?<br>M(Machine draw) or H(Hand-like draw)</td><td>M</td></tr>
</table>

After the above interaction, CINONG generates a program which can draw random objects in *Line-picture*. CINONG gives the program to the user by laying a new button named *Line-picture* on the interface. Pressing the button, a picture which consists of random number of lines with random positions and orientations is drawn. An example picture drawn by the program is shown in Figure 15.11(1).

If the answer is "Y" for the above question "Any constraints? Y/N", further questions will be asked by CINONG to form the constraints. In the circumstance of specifying *Line-picture*, the following possible constraints will be provided to the user.

| | |
|---|---|
| 1 | X value of the start point of the line |
| 2 | Y value of the start point of the line |
| 3 | X value of the end point of the line |
| 4 | Y value of the end point of the line |
| 5 | line size |
| 6 | the length of the list |

By choosing 1 and giving a value (say 150), CINONG generates a program which draws pictures consisting of random number of lines each of which starts from the vertical line $x = 150$ (an example picture is shown in Figure 15.11(2)).

Programs which draw complex pictures can also be generated by CINONG provide that the corresponding pictorial concepts are specified. For instance, inspired by children's cartoon *Teenage Mutant Hero Turtle*, we can specify a concept *Mutant-Turtle*. The program generated by the CINONG can draw one turtle in a random way. For there should be four such turtles in the cartoon, we may specify another concept *Mutant-Turtles* by: first, choosing concept operator **List**, second, choosing *Mutant-Turtle* as the argument for **List** and then specifying the constraint that the length of the list is 4. Furthermore, we may specify other concepts (*e.g. Plant*) and then another concept *Mutant-Turtle-with-Plants* simply by choosing concept operator $\otimes$ and *Plant* and *Mutant-Turtles* as the arguments of $\otimes$. We get a program which can draw mutan turtles with plants. Example pictures drawn by the programs are shown in Figure 15.12.

CINONG provides two kinds of drawing style, free hand drawing and machine drawing. The line pictures in Figure 15.11 are drawn in the style of machine drawing and those turtles are drawn in free hand drawing style. More examples are shown in Figure 15.13.

Figure 15.11: The above picture is drawn by a program which is generated by CINONG according to the pictorial concept *Line-picture* = (**List**(*Line*), *true*). The program which draws the below picture is generated according to the pictorial concept (**List**(*Line Segment*), $P$), where $P$ expresses that the x-value of each line's starting point is 150.

Figure 15.12: Example pictures drawn by the programs generated by CINONG according to pictorial concepts: Mutant-Turtles (1,3,5); Turtle-plants (2) and Turtles-plants (4).

Figure 15.13: Pictures drawn by programs which are generated by CINONG according to various pictorial concepts.

# 16

# Conclusion of Part III

In this part, a type-theoretic approach to pictorial concepts and concept construction has been presented. A concept-supporting system including its implementation has been described to verify the theoretical framework and support various sort of visual communication. Three kinds of visual communication on the basis of picture specifications have been discussed.

One can find in the literature many approaches to the specification or definition of classes of pictures. Some of these are given more in terms of descriptions of programs or implementations of systems (*e.g.* [6]), and others more in abstract theoretical terms (*e.g.* [29]). We distinguish our approach from the others by indicating its relative abstractness, leading to explicitness and generality. The method of concept construction presented in this part not only supports more flexible construction of classes of pictures (*e.g.* by means of disjoint union and list) but more importantly, it provides a new systematic and abstract approach for further development in this direction (see below). It is my hope that this will broaden the application areas of visual communication on the basis of picture specifications.

One of the main contributions of the type-theoretic approach to pictorial concepts presented in this part is the development of a new and richer method to represent and manipulate various structures in picture specification by means of computational (or non-logical) type structures. In particular, the notion of pictorial concept enhances the distinction between the description of the picture structure and that of the logical constraints. Moreover, pictorial concepts in this framework (represented as pairs, rather than modules as in other programming languages) are first-class objects over which operations can be defined. This allows us to specify complicated picture structures, enriching the traditional ways of putting concepts together to form new concepts, by adding various new concept operations such as disjoint union ($\oplus$) and list-construction (**List**). Correlatively, we obtain a higher-level approach to the manipulation of pictorial concepts — the abstract objects — as well as the pictures they abstract over.

Many notions of concept-construction are apparent in work on type-theoretic approaches to program specification [38], [7]. We have applied type theory to pictorial concepts and their construction, but there are several important differences compared with the work in program specification, both in motivation and in technical development. First, pictorial concepts and program specifications have rather different semantic interpretations, in that the former are specifying external entities (pictures on the screen), which is reflected in the implementation of the concept-supporting system, while the latter are concerned with the internal syntactic objects in the language. Second, some of the concept operations (disjoint union and list-construction) we have developed in this part, while being very useful in the construction of pictorial concepts, are seemingly less useful in program construction. Finally, although verifying the correctness of either a program or a picture, with respect to its specification, is a theorem-proving problem which is in general undecidable, some form of automatic concept-checking is fundamentally required in applications of pictures as visual expressions (especially in reasoning). Automatic program verification, in complex cases, is not a practical possibility: program verification is actually done by people rather than programs, and program derivations from proofs are based on explicit proof objects. Graphical concepts, on the other hand, are generally much more tractable. In this respect, the work on structured generation of the concept-checking algorithms for user-defined concepts is a new application.

Spatial operators are often considered in visual languages to build complex visual elements from basic ones. Such spatial operators are similar to some of the concept operations presented in this part. For example, the three spatial image operators in [11], & (spatial overlay), + (horizontal concatenation) and $\wedge$ (vertical concatenation), which are used to build complex icons from basic icons, may be represented as $\mathbf{Ext}(C_1 \otimes C_2, P)$ in our framework, where $C_1$ and $C_2$ represent concepts of basic icons and $P$ expresses that two icons in $C_1$ and $C_2$ are appropriately related, depending on which of &, + and $\wedge$ is being treated. However, due to the difference of motivation, spatial operators only represent several fixed combinations of graphical objects for particular applications and do not support flexible concept construction in general.

In CSGS, users may specify pictorial concepts by presenting examples and answering questions. This relates to the idea of example-based programming and the strategy of question-and-answer (see [48]), and may be called an approach to example-based specification (of concepts). An objective of the work reported in this part is the development of a theoretical framework on the basis of which the CSGS system is able to support users in specifying various kinds of pictorial concepts in the mode of question-and-answer and by presenting examples. It may be possible to study and develop techniques for example-based program specification based on similar ideas.

The type-theoretic framework for pictorial concepts presented in this thesis is rather general in the sense that, using the same notion of pictorial concept, further development is possible which can enrich the type structures of the language so that new picture structures can be described. Among other things, we have been considering *type universes* and *dependent types*, which suggest new type structures to introduce into the theory.

- *Parameterisation* allows users to construct parameterised (or generic) concepts,

which are not only very useful tools for the *re-use* of pictorial concepts, but also relevant to creating and generating new pictorial concepts. For example, (the representations of) polygons in a pictorial concept *Polygon* and closed-curves in *Closed-curve* may share a common structure, *i.e.* both a polygon and a closed curve may be represented by a list of objects (lines and arcs, respectively) with some logical constraints. We can specify a parameterised concept *Poly(X)* as $\mathbf{Ext}(\mathbf{List}(X), Cl(X))$, where $X$ is a concept parameter and $Cl(X)$ is a predicate over $\mathcal{T}_{\mathbf{List}(X)}$ expressing the logical constraints that the objects in the list representation form a closure. Then, we have $Polygon = Poly(Line)$ and `Closed-curve = Poly(Arc)`. Furthermore, one may combine other concept operations to create more interesting concepts, *e.g.* $Poly(Line \oplus Arc)$, which specifies the class of closures whose components are either lines or arcs.
Such a parameterisation mechanism requires type universes to be introduced into the type theory (*c.f.* [43][39]), which would provide interesting applications in concept construction and re-use. Although the idea of parameterisation in picture specification languages is not new (having appeared even in Sketchpad [60], for example), we believe that our parameterisation mechanism based on type theory will provide a more systematic and powerful way to specify parameterised concepts than the existing approaches. Firstly, type-theoretic parameterisation is purely functional, as concepts (and types) are themselves first-class citizens that can be used as values as well as arguments of functions. Secondly, the set of concept operations presented in this part allows users to describe richer picture structures and clear concept constructions. Based on this, the parameterisation mechanism is more powerful than one may have expected. Thirdly, not only can the picture structure be parameterised, but also the logical formulas which are used to describe the constraints of a pictorial concept. This can be achieved theoretically by means of the type-theoretical framework and, in implementation, by means of the implementation language OBJ3 which provides a parameterisation mechanism more powerful than one finds in other languages such as Ada or the usual object-oriented programming languages.
- *Type dependency* allows users to describe pictures consisting of parts of certain other pictures. For example, we may consider the pictures which consist of parts of two intersecting circles and whose existence is dependent on the two circles. The issue of such dependency is closely related to the study of *emergent* graphical objects[58], which are important in understanding the processes of visual reasoning (*e.g.* the objects emerging from the overlapping circles are commonly used to represent intersections of sets in reasoning with Venn diagrams), and are useful in design (*e.g.* extra spaces emerging from the existence of other rooms, walls *etc.* in architecture design, see [58] for details). A clean and special treatment of emergent pictures is very desirable. An elegant mechanism for this purpose may be developed by means of dependent types (dependent function spaces, in particular). For instance, some classes of emergent objects can be described as functions which have dependent function

spaces as types. How this can be fully exploited to deal with emergent objects and their manipulation is under continuing investigation.

Type dependency also allows users to create pictorial concepts whose pictures share common sub-pictures with certain other concepts. For example, one may want to define a concept *House*; a house consists of a kitchen (of concept *Kitchen*) and a room (of concept *Room*), and the kitchen and the room share a wall (of concept *Wall*). In such a case, it is not enough to use logical equality in constraints to assert that the wall of the kitchen and the wall of the room are the same, since the *sameness* required in such a case means *syntactic* or *computational* identity. Using dependent types (dependent sum types, in particular) in type theory can provide a coherent treatment of such structure-sharing.

Besides parameterisation and type dependency as discussed above, one may also consider other concept operations, for example, in developing recursive structures of pictures. These are topics for further exploration.

I divided visual communication via picture specifications into three categories on the basis of the use of the objects in a pictorial concept. Constraint maintaining visual communication and designing are well known subjects studied by many others with the emphasis on dealing with constraints. Applying picture specifications in computational art is new. The idea of CINONG is providing users a convenient environment for creating computational art systems without programming. The implementation is however just in its early stage, more basic pictorial concepts, system built-in attributes and concept operators are needed for generating more interesting computational art systems. Developing suitable algorithms to generate drawing programs is also needed for further exploration. This may be related to various efforts in the area of picture generation.

# Appendix A

# The definition of generalized terms/formulae

In Section7, I introduced *Generalised Term* and *Generalised Formulae* for defining interpretations. In the following, a formal definition of these two concepts is given.

**A.0.2.** DEFINITION. (generalized terms/formulae)
*Let* $\Sigma = (\mathcal{S}, \leq, \mathcal{F}, \mathcal{P})$ *be a signature and* $T_\Sigma(X)$ $(F_\Sigma(X))$ *be the set of terms (formulae) over* $\Sigma$ *with variables in* $X$*. Then,* $GT(\Sigma)$ *and* $GF(\Sigma)$*, the sets of generalized terms and generalized formulae over* $\Sigma$*, are defined as follows:*

1. $GT(\Sigma)$ *is the* $\mathcal{S}'$*_indexed set, where* $\mathcal{S}' = \mathcal{S} \cup \{s_1 \times ... \times s_n \rightarrow s \mid s_i, s \in \mathcal{S}\}$ *such that:*
   - $T(\Sigma) \subseteq GT(\Sigma)$ *and for* $t, t' \in T(\Sigma)$, $t =_{GT(\Sigma)} t'$ *iff* $t =_{T(\Sigma)} t'$
   - $\mathcal{F} \subseteq GT(\Sigma)$
   - *for* $s \in \mathcal{S}$, $X = \{x_1, x_2, ..., x_n\}$, $t \in T_\Sigma(X)_s$, $\lambda x_1 x_2 ... x_n.t \in GT(\Sigma)_{s_1 \times s_2 \times ... \times s_n \rightarrow s}$ *where* $s_i$ *is the sort of* $x_i$*, and, for* $t_i \in T(\Sigma)_{s_i}$, $(\lambda x_1 x_2 ... x_n.t)(t_1, ..., t_n) =_{CT(\Sigma)} [t_1, t_2, ..., t_n/x_1, x_2, ...x_n]t$.
2. $GF(\Sigma)$ *is the* $\mathcal{S}'$*_indexed set, where* $\mathcal{S}' = \mathcal{S} \cup \{s_1 \times s_2 \times ... \times s_n \mid s_i \in \mathcal{S}(\Sigma)\}$ *such that:*
   - $F(\Sigma) \subseteq GF(\Sigma)$*, and for* $A, A' \in F_\Sigma, A =_{GF(\Sigma)} A'$ *iff* $A =_{F_\Sigma} A'$
   - $\mathcal{P} \subseteq GF(\Sigma)$
   - *for* $X = \{x_1, x_2, ...x_n\}$ *and* $A \in F_\Sigma(X), \lambda x_1 x_2 ... x_n.A \in GF(\Sigma)_{s_1,...s_n}$ *and, for* $t_i \in T(\Sigma)_{s_i}$, $(\lambda x_1 x_2 ... x_n.A)(t_1, ...t_n) =_{GF(\Sigma)} [t_1, ...t_n/x_1, ..., x_n]A$.

□

# Appendix B

# A formal presentation of the type system

I give a formal formulation of (the core of) the type system we have informally explained in section 13.2. The formulation is presented by means of Martin-Löf's logical framework (see [50], Part III). We shall use **Type** instead of **Set** as used in [50]. We also drop the lifting operator *El* to write $A$ for $El(A)$, and use infix notations for the type constructors like $\to$, $\times$ and $+$.

## B.1 Function space

$$\begin{aligned} \to &: (\mathbf{Type})(\mathbf{Type})\mathbf{Type} \\ \lambda &: (A, B : \mathbf{Type})((A)B)A \to B \\ \mathbf{app} &: (A, B : \mathbf{Type})(A \to B)(A)B \end{aligned}$$

For $A, B : \mathbf{Type}$, $a : A$ and $f : (A)B$,

$$\mathbf{app}(A, B, \lambda(A, B, f), a) = f(a) \ : \ B(a)$$

## B.2 Product

$$\begin{aligned} \times &: (\mathbf{Type})(\mathbf{Type})\mathbf{Type} \\ \mathbf{pair} &: (A, B : \mathbf{Type})(A)(B)A \times B \\ \mathbf{eval} &: (A, B : \mathbf{Type})(C : (A \times B)\mathbf{Type})((x : A)(y : B)C(\mathbf{pair}(A, B, x, y))) \\ &\quad\ (z : A \times B)C(z) \end{aligned}$$

For $A, B : \mathbf{Type}$, $C : (A \times B)\mathbf{Type}$, $f : (x : A)(y : B)C(\mathbf{pair}(A, B, x, y))$, $a : A$ and $b : B$,

$$\mathbf{eval}(A, B, C, f, \mathbf{pair}(A, B, a, b)) = f(a, b) \ : \ C(\mathbf{pair}(A, B, a, b))$$

## B.3 Sum

$$
\begin{array}{rcl}
+ & : & (\mathbf{Type})(\mathbf{Type})\mathbf{Type} \\
\mathbf{inl} & : & (A, B : \mathbf{Type})(A)A + B \\
\mathbf{inr} & : & (A, B : \mathbf{Type})(B)A + B \\
\mathbf{case} & : & (A, B : \mathbf{Type})(C : (A + B)\mathbf{Type}) \\
 & & ((x : A)C(\mathbf{inl}(A, B, x)))((y : B)C(\mathbf{inr}(A, B, y))) \\
 & & (z : A + B)C(z)
\end{array}
$$

For $A, B : \mathbf{Type}$, $C : (A{+}B)\mathbf{Type}$, $f : (x : A)C(\mathbf{inl}(A, B, x))$, $g : (y : B)C(\mathbf{inr}(A, B, y))$, $a : A$ and $b : B$,

$$\mathbf{case}(A, B, C, f, g, \mathbf{inl}(A, B, a)) = f(a) \ : \ C(\mathbf{inl}(A, B, a))$$

$$\mathbf{case}(A, B, C, f, g, \mathbf{inr}(A, B, b)) = g(b) \ : \ C(\mathbf{inr}(A, B, b))$$

## B.4 List

$$
\begin{array}{rcl}
\mathit{List} & : & (\mathbf{Type})\mathbf{Type} \\
\mathbf{nil} & : & (A : \mathbf{Type})\mathit{List}(A) \\
\mathbf{cons} & : & (A : \mathbf{Type})(A)(\mathit{List}(A))\mathit{List}(A) \\
\mathbf{rec}_{\mathit{List}} & : & (A : \mathbf{Type})(C : (\mathit{List}(A))\mathbf{Type}) \\
 & & (C(\mathbf{nil}(A)))((x : A)(y : \mathit{List}(A))(C(y))C(\mathbf{cons}(A, x, y))) \\
 & & (z : \mathit{List}(A))C(z)
\end{array}
$$

For $A : \mathbf{Type}$, $C : (\mathit{List}(A))\mathbf{Type}$, $c : C(\mathbf{nil}(A))$,
$f : (x : A)(y : \mathit{List}(A))(C(y))C(\mathbf{cons}(A, x, y))$, $a : A$ and $l : \mathit{List}(A)$,

$$\mathbf{rec}_{\mathit{List}}(A, C, c, f, \mathbf{nil}(A)) = c \ : \ C(\mathbf{nil}(A))$$

$$\mathbf{rec}_{\mathit{List}}(A, C, c, f, \mathbf{cons}(A, a, l)) = f(a, l, \mathbf{rec}_{\mathit{List}}(A, C, c, f, l)) \ : \ C(\mathbf{cons}(A, a, l))$$

## B.5 Propositions and proofs

$$
\begin{array}{rcl}
\mathit{Prop} & : & \mathbf{Type} \\
\mathbf{Prf} & : & (\mathit{Prop})\mathbf{Type} \\
\forall & : & (A : \mathbf{Type})((A)\mathit{Prop})\mathit{Prop} \\
\Lambda & : & (A : \mathbf{Type})(P : (A)\mathit{Prop})((x : A)\mathbf{Prf}(P(x)))\mathbf{Prf}(\forall(A, P)) \\
\mathbf{App} & : & (A : \mathbf{Type})(P : (A)\mathit{Prop})(\mathbf{Prf}(\forall(A, P)))(a : A)\mathbf{Prf}(P(a))
\end{array}
$$

For $A : \mathbf{Type}$, $P : (A)Prop$, $f : (x : A)\mathbf{Prf}(P(x))$ and $a : A$,

$$\mathbf{App}(A, P, \Lambda(A, P, f), a) = f(a) \ : \ \mathbf{Prf}(P(a))$$

**Remark** The other logical operators and the propositional equality can be defined by means of $\forall$ as in higher-order logic.

## B.6 The type of natural numbers

We only define the type of natural numbers here as an example of data types.

$$\begin{array}{rcl} N & : & \mathbf{Type} \\ 0 & : & N \\ \mathbf{succ} & : & (N)N \\ \mathbf{rec}_N & : & (C : (N)\mathbf{Type})(C(0))((x : N)(C(x))C(\mathbf{succ}(x)))(y : N)C(y) \end{array}$$

For $C : (N)\mathbf{Type}$, $a : C(0)$, $f : (x : N)(C(x))C(\mathbf{succ}(x))$ and $n : N$,

$$\mathbf{rec}_N(C, a, f, 0) = a \ : \ C(0)$$

$$\mathbf{rec}_N(C, a, f, \mathbf{succ}(n)) = f(n, \mathbf{rec}_N(C, a, f, n)) \ : \ C(\mathbf{succ}(n))$$

# Bibliography


[1] Gerard Allwein and Jon Barwise, editors. *Working Papers on Diagrams and Logic*. IULG-93-24. Indiana University Logic Group, May 1993.

[2] M. A. Arbib and E. G. Manes. *Arrows, Structure and Functors — the Categorical Imperative*. Academic Press, New York, 1975.

[3] H.P. Barendregt. Introduction to generalized type systems. Proc. of the 3rd Italian Conf. on Theoretical Computer Science, Mandera., 1989.

[4] J. Barwise. *Situations and attitudes*. MIT, 1983.

[5] Jon Barwise and John Etchemendy. Visual information and valid reasoning. CSLI report, Stanford., 1990.

[6] A. Borning. The programming language aspects of thinglab, a constraint-oriented simulation laboratory. *ACM Trans. Program. Lang. Syst.*, 3, 1981.

[7] R. Burstall and J. McKinna. Deliverables: an approach to program development in the calculus of constructions. Technical report ECS-LFCS-91-133, LFCS, Dept of Computer Science, Edinburgh University, 1991.

[8] S. K. Chang, editor. *Journal of Visual Language and Computing*. Academic Press. from 1990.

[9] Shi-Kou Chang. Visual reasoning for information retrieval from very large database. *Journal of Visual Language and Computing*, 1(1):41–58, 1990.

[10] Shi-Kou Chang and Stefano Levialdi. The editors' foreword. *Journal of Visual Language and Computing*, 1(1):1–2, 1990.

[11] Shi-Kou Chang, Michael J. Tauber, Bing Yu, and Jing-Sheng Yu. A visual language compiler. *IEEE Transaction on Software Engineering*, 15(5):506–525, 1989.

[12] R.L. Constable et al. *Implementing Mathematics with the NuPRL Proof Development System*. Pretice-Hall, 1986.

[13] Th. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3), 1988.

[14] H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North Holland Publish Company.

[15] M. Dastani. The role of images in the process of reasoning. Computational

Linguistic Dept. Amsterdam University, 1992.
[16] DeVries. Random objectivations. *Villanuova sul Clisi, Italy: Edizioni Amodulo*, 1972.
[17] D. A. Duce and E. V. C. Fielding. Towards a formal specification of the gks output primitive. In *Proceedings of Eurographics 86*. North-Holland, Amsterdam, 1986.
[18] K. S. Fu. *Syntactic Methods in Pattern Recognition*. Academic Press, New York, 1974.
[19] George W. Furnas. Reasoning with diagrams only. In *AAAI Spring Symposium Series, Symposium: Reasoning with Diagrammatic Representations*, pages 118–123. Stanford University, 1992.
[20] J. Girard, Y. Lafont, and P. Taylor. *Proof and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University, 1989.
[21] J.-Y. Girard. *Interpretation fouctionelle et elimination des coupures de l'arithmetique d'ordre superieur*. PhD thesis, Universite Paris VII, 1972.
[22] J. Goguen and R. Burstall. Introducing institutions. *LNCS 164*, 1984.
[23] J. Goguen and R. Burstall. Institutions: abstract model theory for specification and programming. *Joural of the Association for Computing Machinery*, 39(1):95–146, January 1992.
[24] J. Goguen and J. Meseguer. Order-sorted algebra 1: Equational deduction for multiple inheritance, polymorphism, overloading and partial operations. Technical report SRI-CSL-89-10, SRI International, 1989.
[25] J. Goguen, J. Thaticher, and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current trends in programming methodology*. Prentice-Hall, 1978.
[26] J. A. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, SRI, 1988.
[27] Eric Hammer. Representing relations diagrammatically. In Gerard Allwein and Jon Barwise, editors, *Working Papers on Diagrams and Logic*, IULG-93-24, pages 78–119. Indiana University Logic Group, May 1993.
[28] R. Harper, R. Milner, and M. Tofte. The semantics of standard ml. Technical Report ECS-LFCS-87-36, LFCS, Dept of Computer Science, University of Edinburgh, 1987.
[29] Richard Helm and Kim Marriott. A declarative specification and semantics for visual languages. *Journal of Visual Language and Computing*, 2:311–331, 1991.
[30] Daniel D. Hils. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Language and Computing*, 3(1):69–101, 1992.
[31] W. A. Howard. The formulae-as-types notion of construction. In J. Hindley and J. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic*. Academic Press, 1980.
[32] T. Kamada. *Visualizing Abstract Objects and Relations: a constraint-based approach*. World Scientific, 1989.
[33] K.Tsuda, A. Yoshitaka, M. Hirakawa, M. Tanaka, and T. Ichikawa. Icon-

icbrowser: An iconic retrieval system for object-oriented databases. *Journal of Visual Language and Computing*, 1(1):59–76, 1991.

[34] W. Leler. *Constraint Programming Languages: their specification and generation.* Addison-Wesley Publishing Company, 1988.

[35] James Lothian and Neil Leslie. The gizmo system reference mannual. Technical report, EdCAAD, University of Edinburgh, 1992.

[36] Z. Luo and R Pollack. *LEGO Proof Development System: User's Manual.* LFCS Technical Notes ECS-LFCS-92-211, Dept. of Computer Science, Edinburgh University, 1989.

[37] Zhaohui Luo. *An Extended Calculus of Constructions.* PhD thesis, Department of Computer Science, Edinburgh University, 1990.

[38] Zhaohui Luo. Program specification and data refinement in type theory. *Mathematical Structure in Computer Science*, 3(3), 1993.

[39] Zhaohui Luo. *Computation and reasoning: a type theory for computer science.* Oxford University Press, 1994.

[40] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2), 1986.

[41] W. R. Mallgren. *Formal Specification of Interactive Graphics Programming Language.* The MIT Press, Cambrighe, Massachusetts, 1983.

[42] B. Mandelbrot. Fractals. *Form, chance and dimension, San Francisco: W.H. Freeman and Company*, 1977.

[43] Per Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

[44] Pamela McCorduck. *AARON's code: Meta-Art, Artificial Intelligence and the work of Harold Cohen.* W. H. Freeman, 1991. New York.

[45] J. Meseguer. General logics. In H.-D. Ebbinghaus et al, editor, *Logic Colloquium 1987.* North-Holland, 1989.

[46] Francois Morellet. *Nationalgalerie Berlin.* Berlin Baden-Baden Paris, 1977.

[47] P. Mussio, M. Pietrogrande, and M. Protti. Simulation of hepatological models: a study in visual interactive exploration of scientific problems. *Journal of Visual Language and Computing*, 2:75–95, 1991.

[48] Brad A Myers. Demonstrational interfaces: A step beyond direct manipulation. In D. Diaper and N. Hammond, editors, *People and Computers VI*, pages 11–30. Cambridge University Press, 1991.

[49] N. Hari Narayanan, editor. *Symposium: Reasoning with Diagrammatic Representations.* Stanford University, March 1992.

[50] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: an introduction.* Oxford University Press, 1990.

[51] T. Onodera and S. Kawai. A formalization of the specification and systematic generation of computer graphics systems. *Visual Computer*, 1986.

[52] L. A. Pineda. *GRAFLOG: a Theory of Semantics for Graphics with Applications to Human-Computer Interaction and CAD Systems.* PhD thesis, University of Edinburgh, 1990.

[53] D. B. Plummer and S. C. Bailin. Proofs and pictures proving the diamond lemma with the grover theorem proving. In *AAAI Spring Symposium Series, Sympo-*

*sium: Reasoning with Diagrammatic Representations*, pages 102–107. Stanford University, 1992.

[54] D. Sannella and A. Tarlecki. On observational equivalence and algebraic specification. *Computer and System Science*, 34, 1987.

[55] R. Scha. Artificial art. *Informatie en informatiebeleid*, 6(4):73–80, 1988. reprinted in ZeeZucht, 1991, PP. 29-34.

[56] Keith Stenning. Logic as a foundation for a cognitive theory of modality allocation. Human communication research center, Edinburgh University, 1993.

[57] Keith Stenning and Jon Oberlander. Implementing logics in diagrams. In *AAAI Spring Symposium Series, Symposium: Reasoning with Diagrammatic Representations*, pages 91–95. Stanford University, 1992.

[58] George Stiny. What designers do that computers should. In M. McCullough, W. J. Mitchell, and P. Purcell, editors, *The Electronic Design Studio*, pages 17–30. MIT Press, Cambridge MA, 1990.

[59] Shin Sun-Joo. A situation-theoretic account of valid reasoning with venn diagrams. *Situation Theory and its Applications*, 2, 1991. Stanford: CSLI.

[60] Ivan E. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System*. PhD thesis, MIT, Cambridge, 1963.

[61] J. Venn. *Symbolic logic*. New York, Burt Franklin, 1971[1886]. 2nd ed.

[62] Ludwig Wittgenstein. *Philosophical Investigations*. Basil Blackwell, 1988.

# Index

# List of symbols

# Samenvatting

Zoals uitdrukkingen van natuurlijke talen een semantiek hebben, hebben ook grafische voorstellingen een semantiek, als ze gebruikt worden als hulpmiddel in de menselijke communicatie. Er is echter een verschil tussen de semantiek van grafische voorstellingen en de semantiek van natuurlijke talen: de semantiek van grafische voorstellingen is niet eens en voor altijd vastgelegd. Een en hetzelfde plaatje kan geheel verschillende betekenissen hebben als het in verschillende omstandigheden wordt gebruikt.

Dit proefschrift houdt zich bezig met de vraag hoe grafische voorstellingen betekenis krijgen in het gebruik van zulke voorstellingen in communicatie. Het doel van deze studie is het ontwikkelen van formele kaders waarbinnen computationele systemen kunnen worden ontworpen om visuele communicatie te ondersteunen.

In het bijzonder onderzoek ik twee semantische benaderingen van grafische voorstellingen: de methode van semantische interpretatie en de methode van conceptuele specificatie. De interpretatiemethode kent aan iedere grafische entiteit een expliciete betekenis toe en de semantiek van de gehele voorstelling wordt bepaald door de betekenis van de componenten en hun ruimtelijke relaties. De specificatiemethode kent op impliciete wijze betekenis toe aan een grafische voorstelling door het definiëren van klassen van voorstellingen volgens de eisen van het toepassingsgebied dat wordt gerepresenteerd.

Dit proefschrift bestaat uit drie delen. In het eerste deel onderzoek ik hoe grafische vorstellingen geometrisch kunnen worden gekarakteriseerd. Er wordt een formeel kader aangegeven voor het ontwikkelen van een beschrijvende grafische taal en vervolgens worden een methode aangegeven om in zo'n taal een grafische voorstelling te beschrijven.

In het tweede gedeelte wordt de interpretatiemethode onderzocht. Er wordt een definitie van interpretatie ontwikkeld uitgaande van de resultaten van deel 1. Deze definitie dient als criterium om te controleren of een interpretatie syntactisch correct is. Verder wordt een logische karakterisering ontwikkeld van het juiste gebruik van een interpretatie in visuele communicatie en van de correctheid van de interpretatie met betrekking tot de bedoelingen van de gebruiker. Hierna worden verschillende toepassingsmogelijkheden besproken, uitmondend in de beschrijving van een specifiek systeem voor redeneren met grafische ondersteuning beschreven: het GAR-systeem. Tenslotte wordt een algemene benadering van het redeneren met diagrammatische voorstellingen ontwikkeld.

In het derde deel gaat het om de specificatie van grafische voorstellingen. Ik breid de traditionele specificatiemethode uit door de toevoeging van grafische conceptoperatoren die de vorming van bepaalde klassen van voorstellingen vereenvoudigen ten opzichte van de traditionele methode. Een aantal toepassingen worden besproken, waaronder toepassingen op computationele kunst. Het gaat in het laatste geval om de experimentele systemen CSGS en CINONG. CSGS ondersteunt het definiëren van grafische concepten door een gebruiker en CINONG synthetiseert eenvoudige computerkunstprogramma's.

# Titles in the ILLC Dissertation Series:

*Transsentential Meditations; Ups and downs in dynamic semantics*
Paul Dekker
ILLC Dissertation series, 1993-1

*Resource Bounded Reductions*
Harry Buhrman
ILLC Dissertation series, 1993-2

*Efficient Metamathematics*
Rineke Verbrugge
ILLC Dissertation series, 1993-3

*Extending Modal Logic*
Maarten de Rijke
ILLC Dissertation series, 1993-4

*Studied Flexibility*
Herman Hendriks
ILLC Dissertation series, 1993-5

*Aspects of Algorithms and Complexity*
John Tromp
ILLC Dissertation series, 1993-6

*The Noble Art of Linear Decorating*
Harold Schellinx
ILLC Dissertation series, 1994-1

*Generating Uniform User-Interfaces for Interactive Programming Environments*
Jan Willem Cornelis Koorn
ILLC Dissertation series, 1994-2

*Process Theory and Equation Solving*
Nicoline Johanna Drost
ILLC Dissertation series, 1994-3

*Calculi for Constructive Communication, a Study of the Dynamics of Partial States*
Jan Jaspars
ILLC Dissertation series, 1994-4

*Executable Language Definitions, Case Studies and Origin Tracking Techniques*
Arie van Deursen
ILLC Dissertation series, 1994-5

*Chapters on Bounded Arithmetic & on Provability Logic*
Domenico Zambella
ILLC Dissertation series, 1994-6

*Adventures in Diagonalizable Algebras*
V. Yu. Shavrukov
ILLC Dissertation series, 1994-7

*Learnable Classes of Categorial Grammars*
Makoto Kanazawa
ILLC Dissertation series, 1994-8

*Clocks, Trees and Stars in Process Theory*
Wan Fokkink
ILLC Dissertation series, 1994-9

*Logics for Agents with Bounded Rationality*
Zhisheng Huang
ILLC Dissertation series, 1994-10

*On Modular Algebraic Prototol Specification*
Jacob Brunekreef
ILLC Dissertation series, 1995-1

*Investigating Bounded Contraction*
Andreja Prijatelj
ILLC Dissertation series, 1995-2

*Relativized Algebras of Relations and Arrow Logic*
Maarten Marx
ILLC Dissertation series, 1995-3

*Study on the Formal Semantics of Pictures*
Dejuan Wang
ILLC Dissertation series, 1995-4