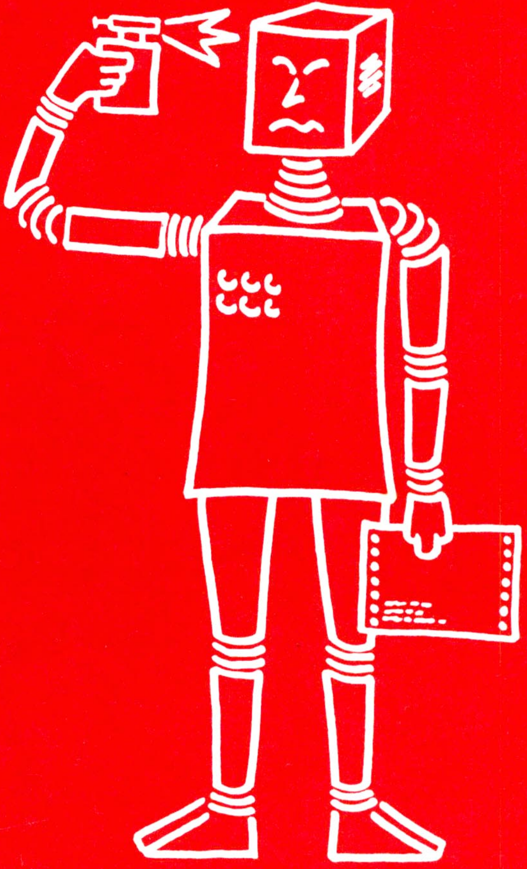
Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
D-6750 Kaiserslautern
West Germany

Universität Karlsruhe
Institut für Informatik I
Postfach 6380
D-7500 Karlsruhe
West Germany



SCENELAB

Scene Labelling by a Society of Agents
A Distributed Constraint Propagation System

Michael Reinfrank

MEMO SEKI-85-06 August 1985

Abstract

This paper describes SCENELAB, a computer system for labelling line drawings of scenes in simple polyhedral worlds. The key idea behind SCENELAB is to bring together the concept of contraint-based filtering algorithms and the paradigm of societies of cooperating agents. The problem of finding labellings for pictures drawn from blocks world scenes has been taken as a sample application. Clearly, this makes SCENELAB no vision system, but we claim that a system designed along these lines could be part of a real vision system.

Following e.g. Alan Mackworth, we argue that constraint exploitation on resp. between various representational levels is a key technique of 'seeing things'. Furthermore, constraints and constraint propagation neatly fit into the framework of societies of agents, realized by asynchronuously concurrent processing units and message passing mechanisms.

SCENELAB, as it is actually running, can be used to specify and solve arbitrary labelling problems that can be seen as instances of a particular class of simple constraint problems, based on finite, pseudo-transitive binary constraints. However, it is felt that the overall approach generalizes to arbitrary constraint problems.

Emphasis is given to a mathematical model of the problem and its solution, to be able to specify the reasoning techniques of SCENELAB, and to identify the class of problems it can handle. I tried to shed some light onto the methodological background of SCENELAB, which seems necessary to judge the achievements and disachievements of the present work.

After some introductory chapters on the key concepts involved in SCENELAB, (scene) labelling problems, constraint propagation, and societies of agents, an overview on both the structure and behavior of SCENELAB is given in part B of the paper. In part C, then, an algebraic model is introduced, which serves as a base for discussing several approaches to labelling problems, namely Waltz's original landmark algorithm, a synchronized parallel solution suggested by Azriel Rosenfeld, and clearly, the present approach. A proof of the correctness of SCENELABs algorithms is included. This proof takes into account the specifities of systems of asynchronously communicating agents where no global state is observable.

Author's current address

Michael Th. Reinfrank
Kieferweg 4
6730 Neustadt-Hambach
West Germany

Table of Contents

0 Preface

This paper describes SCENELAB, a fully implemented computer system for labelling line drawings of scenes in simple polyhedral worlds. The key idea behind SCENELAB is to bring together the concept of contraint-based filtering algorithms and the paradigm of societies of cooperating agents. The problem of finding labellings for pictures drawn from blocks world scenes has been taken as a sample application. Clearly, this makes SCENELAB no vision system, but we claim that a system designed along these lines could be part of a real vision system.

Following e.g. Alan Mackworth [Mackworth-83], we argue that constraint exploitation on resp. between various representational levels is a key technique of 'seeing things'. Furthermore, constraints and constraint propagation neatly fit into the framework of societies of agents, realized by asynchronuously concurrent processing units and message passing mechanisms.

SCENELAB, as it is actually running, can be used to specify and solve arbitrary labelling problems that can be seen as instances of a particular class of simple constraint problems, based on finite, pseudo-transitive binary constraints. However, it is felt that the overall approach generalizes to arbitrary constraint problems.

Emphasis is given to a mathematical model of the problem and its solution, to be able to specify the reasoning techniques of SCENELAB, and to identify the class of problems it can handle. I tried to shed some light onto the methodological background of SCENELAB, which seems necessary to judge the achievements and disachievements of the present work.

After some introductory chapters on the key concepts involved in SCENELAB, (scene) labelling problems, constraint propagation, and societies of agents, an overview on both the structure and behavior of SCENELAB is given in part B of the paper. In part C, then, an algebraic model is introduced, which serves as a base for discussing several approaches to labelling problems, namely Waltz's original landmark algorithm [Waltz-72], a synchronized parallel solution suggested by Azriel Rosenfeld [Rosenfeld|Hummel|Zucker-76], and clearly, the present approach. A proof of the correctness of SCENELABs algorithms is included. This proof takes into account the specifities of systems of asynchronously communicating agents where no global state is observable.

1 Scene Analysis

1.1 A Note on Computer Vision

This is a paper on distributed constraint propagation rather than on vision. Nevertheless, the problem of labelling line drawings plays a central role throughout the paper, so I feel some general remarks on computer vision in order. This is simply to place the problems addressed into their proper context, to be able to judge what has and what has not been achieved by the work presented herein.

Natural vision is considered to be a kind of intelligent information processing, hence vision is an integral part of artificial intelligence, too. However, natural vision is not understood well enough to directly copy the visual machinery of biological seeing into some artificial hardware. Therefore, AI research strives for a better understanding of the key issues involved in 'seeing things', and focusses on making a computer be rather than simulate a seeing machine.

Research thus far has provided the key insight that vision relies both on massive computation and knowledge-based processing. I.e., vision is neither a matter of pattern analysis only, nor can it be hoped that some elaborate problem solving component can completely replace extensive low level computations.

The purpose of scene analysis is not only to process a 2D image but also to understand that image in terms of a 3D scene it has been made of. Therefore, scene analysis is often referred to as to image understanding to contrast it with image processing [Cohen|Feigenbaum-82]. It seems to be a matter of fact that the gap between a 2D image and a 3D interpretation cannot be come across by one single large step. However, it is still subject to a rather controverse discussion what the intermediate level representations should be, and how processing should proceed (bottom-up, top-down, mixed heterarchical control ...). One of the currently most popular approaches has been suggested by David Marr [Marr-82]. Marr separates the following representational levels (see figure 1.1):

- A digitalized 2D image, a so-called pixel (for picture elements) array, representing brightness values, i.e. the powers per unit area sensored by a camera.

- A primal sketch, where some low level computations have made facts about brightness changes, texture, etc. explicit.

- A 2-1/2D sketch, including e.g. information about surface orientation

- A 3D world model, representing facts about volumes. These models are frequently based on Binford's generalized cylinders [Binford-82].

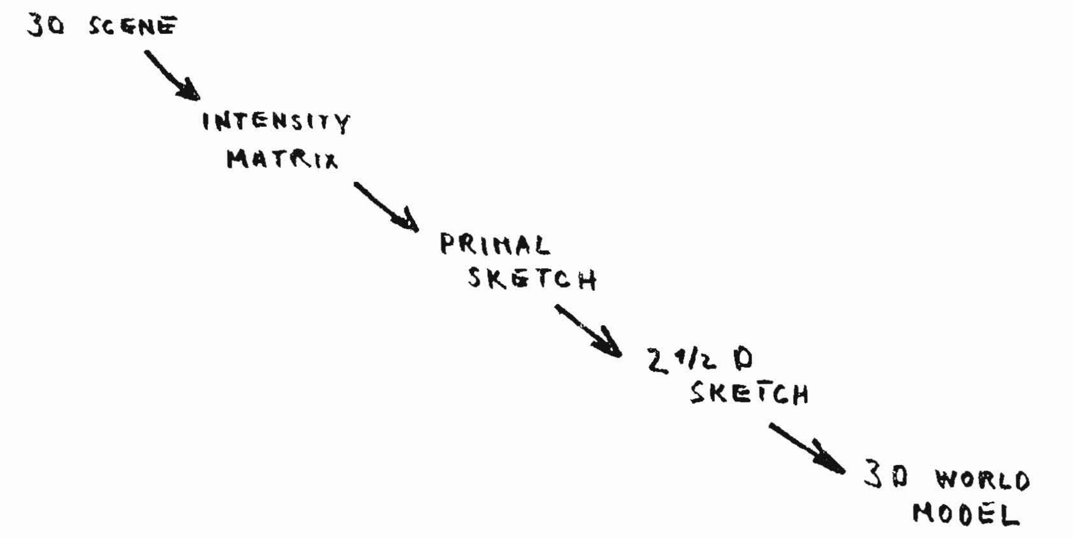


Figure 1.1: Scene analysis proceeds through several intermediate levels of representation

One particularly important point made by Marr (see also [Mackworth-83]) is that an image inherently underconstrains the scene(s) it might represent, and that the process of vision can be considered as exploring various kinds of constraints within and/or between different levels of abstraction, to increasingly restrict the equivalence class of scenes that could provide a satisfactory interpretation for the image.

I do not pursue the discussion of general issues in computer vision here. Some introductory readings on vision can be found e.g. in [Cohen|Feigenbaum-82] and [Winston-84]. David Marr's book VISION [Marr-82] provides a detailed treatment of his theory on vision. Some state of the art papers are gathered in a special issue of the AI-journal on this subject [AI17-81]. Alan Mackworth [Mackworth-83] reviews the achievements of computer vision from a global perspective.

1.2 Perfect Drawings of Trihedral Scenes

Throughout this paper, I use a very simple model of the vision process that relies on several restrictive assumptions. First of all, the world in consideration is a toy blocks world, composed of opaque polyhedral objects. We suppose that at every vertex of a complex object that may consist of several such polyeders, exactly three surfaces meet. I.e. polyeders with cracks, and some particular alignments of polyeders are excluded (figure 1.2).

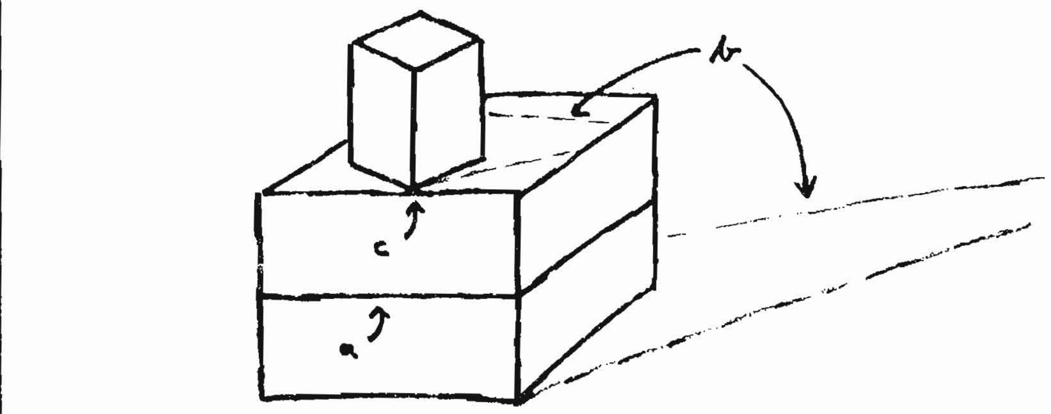


Figure 1.2: Non-trihedral object. Crack-line (a), shadows (b), more than three surfaces meeting at one vertex (c).

Furthermore, we assume that the choice of viewpoint is such that a minor movement of the eye cannot lead to significant changes of the image, e.g. no axis of projection may fall into a plane defined by a surface (figure 1.3). Finally, we assume that no significant brightness change is due to a shadow in the scene, or to some borderline between two differently colored regions of one surface.

Given a 3D scene in such a trihedral world, we use only one intermediate level representation between its 2D image and its interpretation, namely a line drawing showing the edges, regions, and vertices of the scene from a particular point of view. In this context, then, the purpose of scene analysis is to find an interpretation of that drawing (figure 1.4).

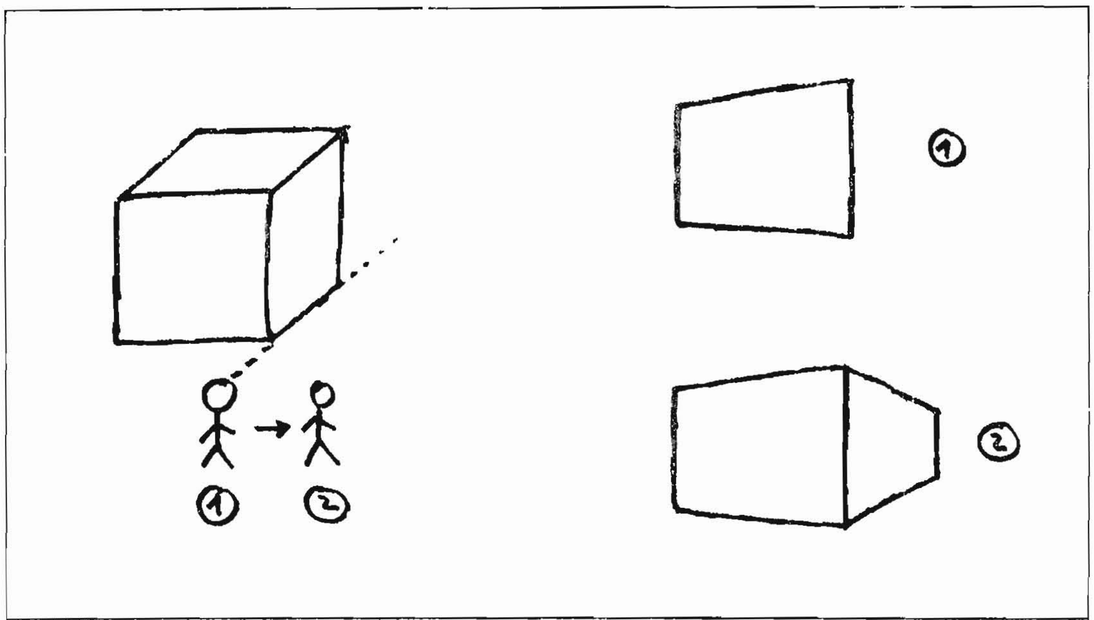


Figure 1.3: An accidental alignment.

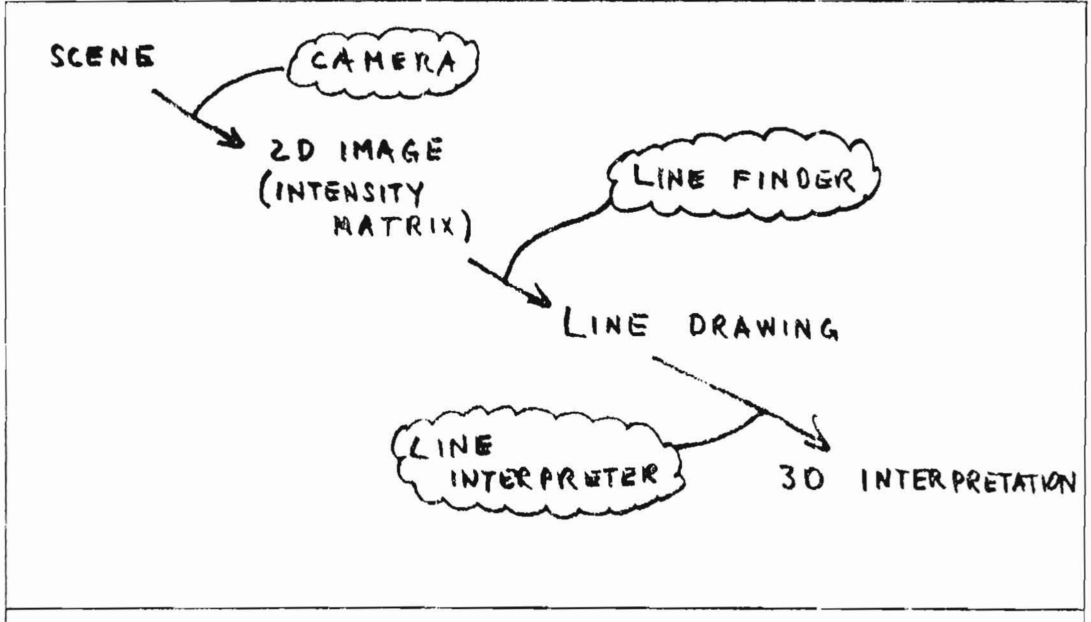


Figure 1.4: Our simplified model uses only one intermediate level representation.

It should be clear that in any reasonably complex real world environment, none of these restrictions will be met. Note that things get even more complicated when e.g. sequences of images from time-varying scenes have to be interpreted. However, some of the key problems of vision can be studied in such simple worlds, and the basic concepts (not their technical realizations) like e.g. constraint exploitation that apply in simple worlds often generalize to real worlds. Alan Mackworth [Mackworth-77] surveys a number of scene analysis systems that work in simple worlds.

1.3 The Relation of Representation

Computer vision is closely related to natural language understanding, which is another field of major interest in AI. As M.B. Clowes has pointed out in his landmark publication 'On Seeing Things' [Clowes-72], some concepts that have proven fruitful for semantic-based natural language processing can be very usefully introduced into the area of scene analysis. In particular, this is true for the distinction made between expressions like the sound of a spoken r, a written r, etc..., and their corresponding abstractions, here the letter r. Taking pattern from Clowes, we say that a 3D scene belongs to an abstract scene domain, while drawings of that scene belong to an expressive picture domain. The relation between a scene and its expression in terms of the picture domains primitives like lines, points, and regions, is called - again following Clowes - the Relation of Representation.

In the sequel, our considerations will rely on an additional assumption about a given line drawing. A drawing is assumed to be perfect in the sense that each of its junctions, regions, and lines corresponds to resp. exactly one vertex, surface, or edge, and that every such scene element that can be seen from the current viewpoint is represented in the drawing. The perfectness assumption is even more demanding than the simplicity assumptions about trihedral scenes, since it imposes some unrealistic requirements on the sensoric machinery and the line extracting module of a vision system.

Notice that a clean distinction between the scene domain and the picture domain, although it may seem a little bit too sophisticated, is essential for the understanding of the capabilities of labelling procedures, as discussed in this paper. Such algorithms work only in the picture domain, and any interpretation based on labellings heavily relies on the descriptive adequacy of the picture primitives (the labels included) chosen, and on a proper relation between a drawing and the corresponding scene.

2 Labelled Line Drawings

2.1 The General Idea

Given a line drawing, we are seeking a description in terms like "there is a small block in front of a big block, a tetraeder stands on that big block ...". One big step towards such an interpretation is frequently accomplished by finding an assignment of some meaningful labels to picture elements. The key ideas hereby are:

- Labels stand for properties of the objects in a 3D scene being represented by the elements of a line drawing. An attachment of a label to such an element means that the corresponding scene element has a particular property.

- There are natural constraints upon the properties of elements that stand in a particular relationship. This means that only certain combinations of labels are admissible for the corresponding picture elements, while other combinations would denote a physically impossible situation.

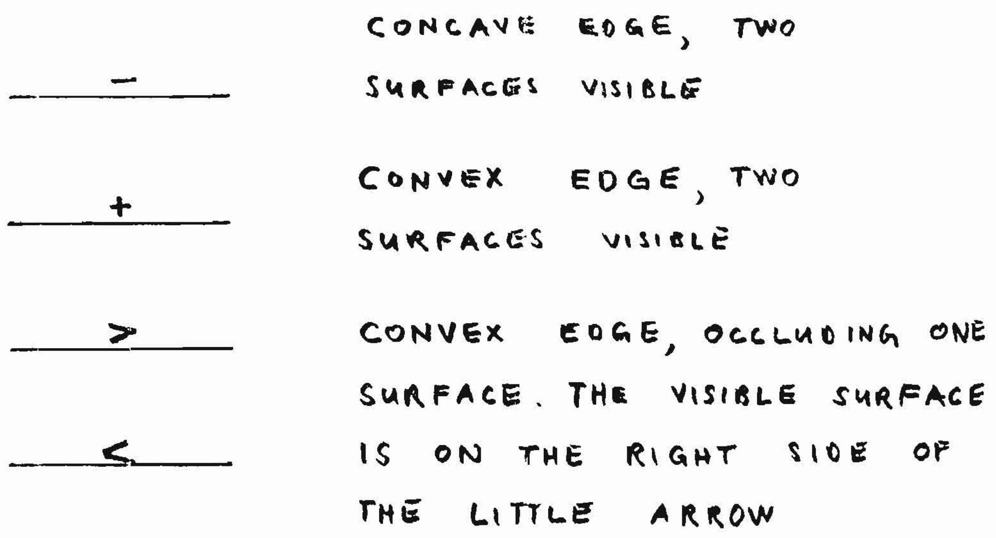
In order to realize this idea, two major problems have to be solved. Firstly, an appropriate choice of properties to be represented must be made, and adequate representations in terms of labels must be found. Furthermore, the constraints upon the occurences of (combinations) of these properties must be elaborated. Secondly, given such a collection of labels and corresponding constraints, methods must be designed to find labellings for line drawings that satisfy all of these constraints.

Clearly, the choice of properties and labels should be such that it is easy to derive readable descriptions of a scene from a labelled drawing. In the present paper, we are mainly concerned with the problem of finding all the admissible labellings of a line drawing using a predefined set of labels and related constraints.

2.2 The Huffman-Clowes Label Set

In 1971, D. A. Huffman [Huffman-71] and M. B. Clowes [Clowes-71] - independently - published two papers on scene analysis that can be regarded as the antecedents of many subsequently developed labelling procedures. In fact, their systems, too, had some predecessors as e.g. the SEE-program by A. Guzman [Guzman-68], but Huffaman and Clowes were the first to make the very principles explicit that underly such procedures.

Huffman and Clowes restricted their considerations mainly to the properties of edges in trihedral scenes. Basically, an edge can be concave or convex. Given a fixed viewpoint, one sees either both or none of the surfaces meeting at a concave edge, i.e. one label suffices to mark concave edges. The case is different for a convex edge: it may be invisible, or one can see one or both of the surfaces meeting at the edge. Furthermore, if only one surface is visible, it may be on one of two different sides. These three cases are reflected by three different labels for convex edges. The complete set of four line labels is shown in figure 2.1.



CONCAVE EDGE, TWO SURFACES VISIBLE

CONVEX EDGE, TWO SURFACES VISIBLE

CONVEX EDGE, OCCLUDING ONE SURFACE. THE VISIBLE SURFACE IS ON THE RIGHT SIDE OF THE LITTLE ARROW

Figure 2.1: The Huffman/Clowes segment labels

For reasons that soon will become clear, we adopt the convention that we always look at a segment from the point of view of one of its two end-junctions. We therefore temporarily use pairs of labels for segments, one for each junction in consideration. The property of being resp. a concave edge or a convex edge with two surfaces visible is independent from the viepoint. However, a region that is on the right side of a segment when viewed from one junction is on the left side when viewed from the opposite junction, and vice versa. To make the viepoint explicit, we replace the little arrow labels by o> and >o, and speak of ingoing and outgoing segments w.r.t. that particular junction.

It is easy to see that the properties chosen are both exhaustive and exclusive in the sense that every edge, in combination with a given point of view, must have one and only one of these properties. I.e., every line must be labelled with exactly one of the following pairs of labels: (+,+), (-,-), (o>,>o), (>o,o>). Given an assignment of such pairs of labels to segments, we can orientate the segments according to the following definition:

- If a segment is labelled (+,+) or (-,-), choose an arbitrary but fixed direction.

- A segment with labels (o>,>o) or (>o,o>) is directed from the junction where it is outgoing to the junction where it is ingoing, i.e. from o> to >o.

Thus, having a unique labelling, we can use the singular labels shown in figure 2.1 instead of pairs of labels. For arrow-labelled segments we implicitely assume that the segment has the same direction as the arrow, and hence that the visible surface is on the right side of the segment when the movement of eye follows the direction given by the arrow. Figure 2.2 shows the labelled drawings of a cube and of a small stair.

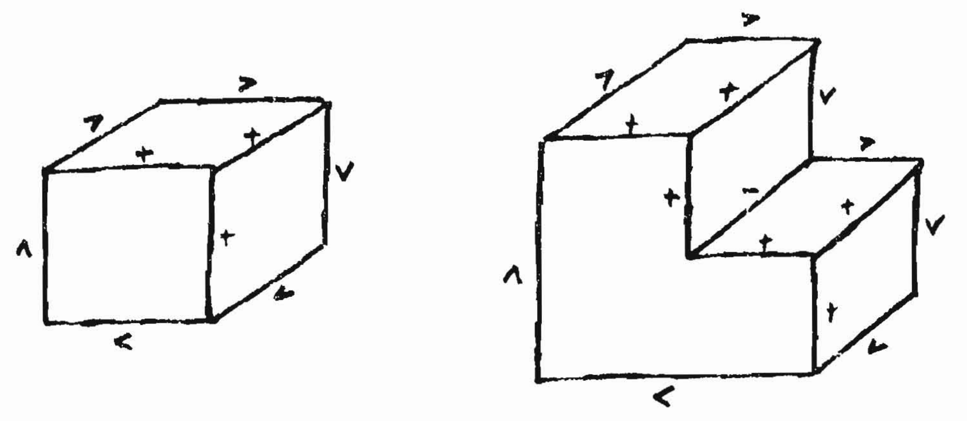


Figure 2.2: Labelled line drawings

2.3 Admissible Huffman-Clowes Labellings

The very central idea behind labelling algorithms now is the following: the edges meeting at a trihedral vertex can only exhibit some combinbations of the properties represented by labels, for physical reasons, and these are ususally only a small subset of all combinatorially possible combinations.

As every trihedral vertex joins exactly three surfaces, it subdivides the space into eight octants. The follwoing procedure can be used to systematically derive all possible combinations of labels at junctions representing such a vertex. Consider all combinations of 1 to 7 of the resulting octants being filled with opaque material, and place the observer into each of the remaining free octants to look at the vertex (figure 2.3).

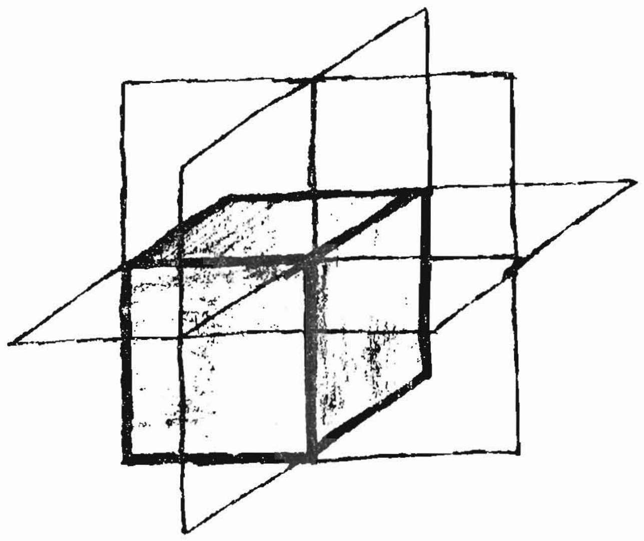


Figure 2.3: A trihedral vertex subdivides the space into eight octants

It turns out hat trihedral junctions fall into one of four different classes, properly separable by some geometric features (figure 2.4). Notice that all but one of these figures, the FORK, allow for the definition of a unique ordering of the segments involved. This ordering is based on the fact that ther segments of a junction concavely bound at most one region, which can serve as a starting point for a clockwise order. For a FORK, we can choose one of three possible cyclic permutations of an arbitrary but fixed clockwise ordering. Figure 2.5 then shows all of the eighteen possible combinations of segment labels at a given junction, using the Huffman/Clowes label set. The combination of segment labels at a junction is said to be a (composite) junction label, and a list of admissible junction labels is referred to as to a label dictionary.

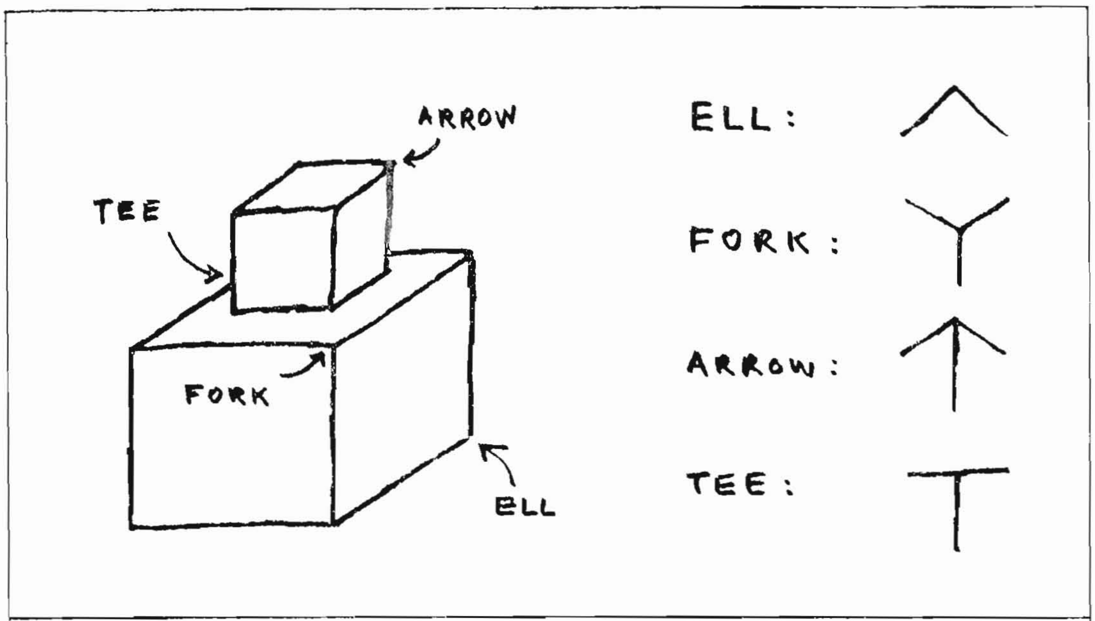


Figure 2.4: Trihedral junctions fall into one of four possible classes

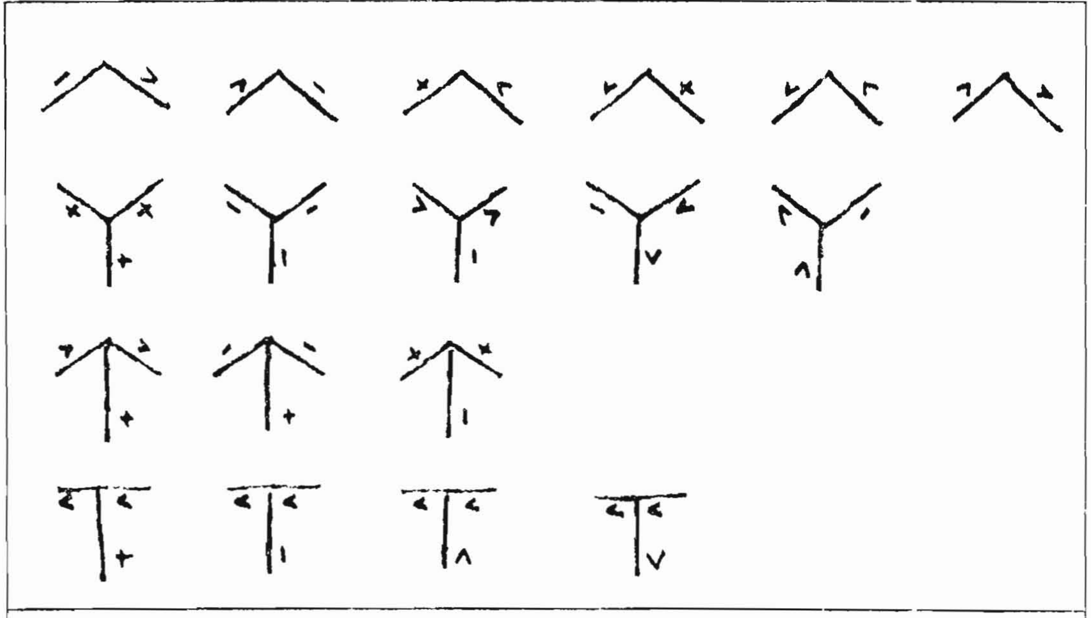


Figure 2.5: The complete Huffman/Clowes label dictionary

Note that e.g. an ARROW can only be labelled in 3 different ways, although there are 64 combinatorial possibilities. The problem of scene labelling then states as:

- Given a label dictionary and a line drawing, find an assig- ment of junction labels to junctions such that every junction is labelled with a unique label from the appropriate class, and such that any pair of neighboring junctions induces a pair of compatible segment labels for the segment joining them.

We now could expect that every drawing that is labelld this way represents a possible real world scene. Unfortunately, this is not the case, as shown in figure 2.6. However, every drawing of a trihedral scene has at least one admissible labelling. I.e., having an admissible labelling is a necessary but not a sufficient condition for a drawing to depict a possible scene. Sugihara [Sugihara-82] argued that a sufficient condition can only be achieved when information about surface orientation is included. Other authors as e.g. Draper [Draper-81] came to similar results.

As we have yet mentioned, the information given by an image, and especially by such a highly abstracted representation as a line drawing, inherently underconstrains the scene. It should be clear that a labelled line drawing stands for an entire euivalence class of scenes, since e.g. no difference is made between a flat or a peaked arrow. Moreover, many aspects of scenes such as color and texture are not represented in our simple models. There are also line drawings with more than one admissible labellings (figure 2.7), i.e. the equivalence classes induced by unlabelled drawings are not so fine-grained as those induced by labelled drawings.

However, there is one point to be made: elaborate vision systems, too, make extensive use of similar constraint directed representation and reasoning techniques, labels standing e.g. for depth, illumination, and orientation information, or even for specific objects in the domain of discourse. Tenenbaum and Barrow [Tenenbaum|Barrow-76] use region labels like 'door-knob' and 'door', a possible constraint being that a region labelled 'door-knob' must be within a region labelled 'door'.

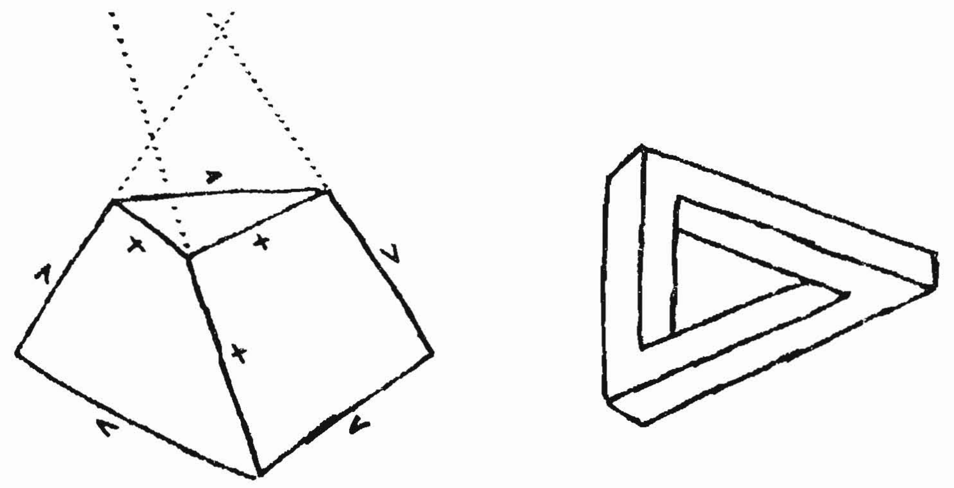


Figure 2.6: An impossible object with an admissible labelling (left), and another impossible object, having no admissible labelling. (Figures due to Sugihara [Sugihara-82]).

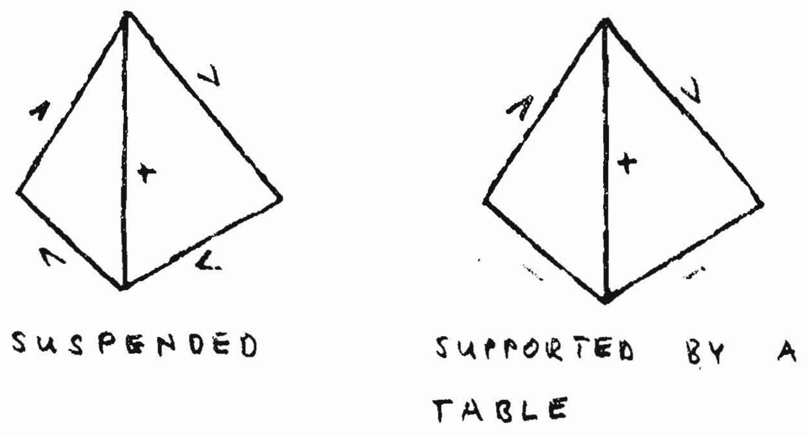


Figure 2.7: A line drawing mith multiple admissible labellings

3 Constraints and Constraint Propagation

3.1 Waltz-Filtering

Early labelling procedures used standard search strategies to find admissible labellings, like e.g. depth-first search with backtracking [Huffman-71]. The structure of the search tree thereby depends upon the order in which the procedure attempts to label the junctions. In 1972, David Waltz [Waltz-72], [Winston-75] tried to loosen the restrictions imposed by a trihedral world by a substantial extension of the Huffman/Clowes label set. The introduction of labels e.g. for crack lines and for region illumination led to an explosive growth of the number of possible composite junction labels that now ranged at more than 3,000. Standard search methods usually fail to find solutions in such a giant search space within an acceptable time.

It was the merit of Waltz to recognize that the admissibility of labellings mainly depends on local features, and that therefore, a locally restricted search can drastically prune the search space. A label from a given universe of potentially possible labels for a junction of a certain class can only contribute to an admissible labelling if at least one compatible label can be found for each adjacent junction. The key idea of the Waltz-algorithm is to exploit this restriction and to eliminate every label that can not be part of an admissible labelling for these local reasons. Properly speaking, the Waltz-algorithm works as follows:

```
(1) for every junction x of the drawing do

    (1.1) assign the set of physically possible labels,
          as given by a label dictionary, to x

    (1.2) mark x visited

    (1.3) adjust the labelling of x w.r.t. all of its
          neighbors that have been previously visited,
          i.e. remove from x every label $\sigma$ for which
          there is some neighbor y of x without any
          label compatible with $\sigma$

    (1.4) recursively do until no more deletions occur
            for all marked neighbors y of x, adjust the
            labelling of y w.r.t. that one of x.
```

We call a set valued labelling locally consistent at a junction x if every label σ of x can consistently reside at x in the sense that every neighbor y of x has at least one compatible label ρ. As will be shown in a later chapter, the Waltz-algorithm correctly determines the maximal set-valued labelling being contained in the original labelling given by the dictionary that enjoys this local consistency property for every junction.

Unfortunately, local consistency does not guarantee that the remaining labels allow for the construction of a unique admissible labelling. We will show that the result of the Waltz-algorithm necessarily comprises every admissible labelling of a drawing, while not every inadmissible labelling is filtered out. Thus, the relevance of Waltz-like algorithms is to scale down to a manageable size the number of possibilities to be considered by a costly global search strategy. Recently, some authors [Gaschnig-79], [Nudel-83] have suggested to use hybrid algorithms that combine filtering and tree search. For an overview of such consistency providing procedures, see e.g. [Mackworth|Freuder-85].

3.2 Constraint Propagation

The Waltz-filtering procedure derives constraints upon possible labellings by local computations and then it propagates these restrictions towards junctions in the neighborhood. This is in fact the key idea of the constraint propagation paradigm which nowadays plays a prominent role in AI. Constraint problems usually exhibit the following structure: There is a number of variables VARS and corresponding domains DOMS, a variable v taking values from an associated domain dom(v). A constraint relation is an n-ary relation on domains, i.e.:

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n$$

A constraint instance or, simply, a constraint is a named triple of the form <NAME,REL,VAR-LIST>, where NAME is a unique symbolic name, VAR-LIST is an n-tuple $(v_1, v_2, \ldots, v_n) \in VARS^n$ of variables, and REL is an n-ary constraint relation on the appropriate domains. A constraint network, then, is composed of a number of such constraints.

It is often convenient to represent such networks graphically as follows: there are two types of nodes, variable nodes and constraint nodes, one for each variable and constraint in consideration. The constraint nodes are linked exactly to those variables mentioned in its variable list (see figure 3.1).

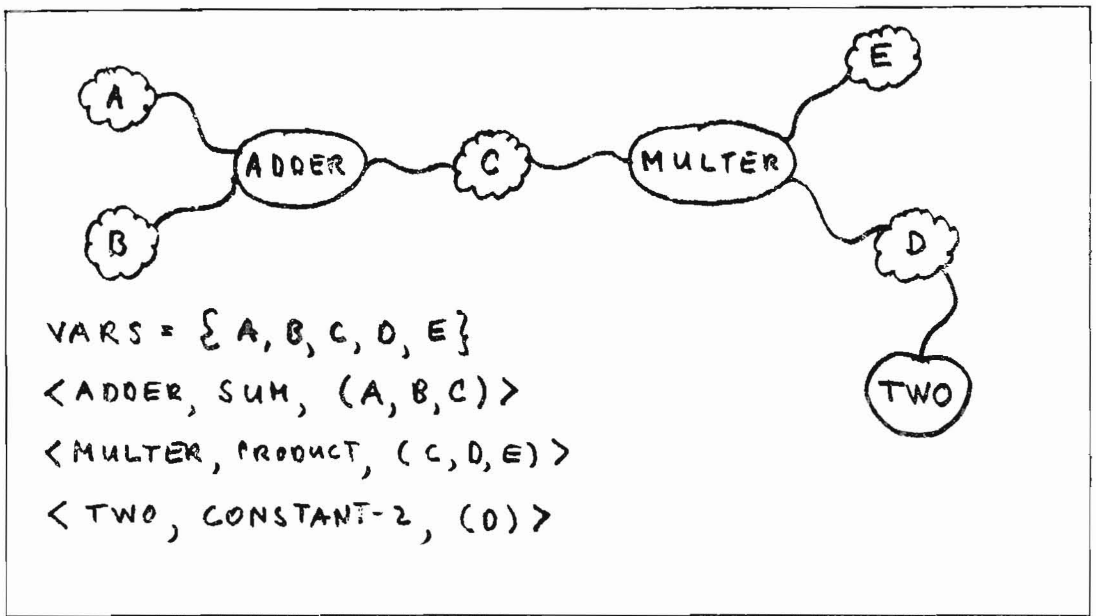


Figure 3.1: Constraint nodes and variable nodes in a constraint network

A single-valeued assignment of values to variables is a partial function

$$\begin{aligned} ass: \ & VARS \rightarrow DOMS \\ & v \mapsto ass(v) \in dom(v) \end{aligned}$$

Definition 3.1 [admissible assignments]

An assignment ass is admissible w.r.t. a constraint $<N,R,(v_1,v_2,...,v_n)>$ if and only if it is defined for every variable in the list and

$$(ass(v_1),ass(v_2),...,ass(v_n)) \in REL$$

holds. Admissibility extends to constraint networks in that an assignment is admissible w.r.t. to a network of constraints if and only if it is admissible w.r.t. to every single constraint of the network.

■

A constraint network thus can be seen as a declarative specification of all admissible assignments to its variables. Given a partial assignment to some of the variables, a procedure that finds admissible completions, if any, can be used to perform arbitrary computations. Figure 3.1 shows a VISICALC-like [Steele-80], [Sussman|Steele-80] network representing the arithmetic equations A+B=C, C*D=E, and D=2. The constraint relations in that example, however, are very simple. For every n-ary constraint, (n-1) values uniquely determine the remaining n-th value. We call a relation R almost simple if for a 'reasonably large' number of (n-1)-tuples there is exactly one completed n-tuple belonging to R. For the remaining tuples, there may be no or several different such completions. Given a network of almost-simple constraints, and a partial assignment, admissible completions can be found by constructive propagation algorithms of the following form:

- Whenever all but one variables in the list of a constraint have definite values, compute the corresponding value for that variable, and make it available at all other constraints the variable is involved in.

- When all the variables in a list have definite values, check that they satisfy the constraint relation.

A constraint problem given by a network of constraints along with an initial partial assignment may be both overconstrained and underconstrained. I.e. there may be no solution (= admissible completion) at all, or there may be several solutions.

Moreover, when trying to find a solution by local, constructive propagation algorithms, several difficulties may arise. The propagation procedure may fail to find an admissible completion either because at some simple constraint, less than (n-1) values are known, or because at some non-simple constraint, no unique value can be deduced. The former case can be handled by introducing and propagating unknowns, but that requires the evaluation and simplification of symbolic expressions. The latter case can be handled by making choices, but, as multiple choices may turn out to be mutually incompatible, a conflict handling mechanism is needed.

If the constraint relations are highly non-simple, making choices and recovering from erroneous choices becomes infeasible. In this case, destructive propagation algorithms along the lines of Waltz-filtering apply as a pre-processor. Such algorithms handle set-valued assignments

$$\begin{aligned} \text{sv-ass}: \ & VARS \rightarrow \mathcal{P}(DOMS) \\ & v \mapsto \text{sv-ass}(v) \subseteq dom(v) \end{aligned}$$

For such set-valued assignments, the definition of local consistency replaces admissibility.

Definition 3.2 [local consistency of set-valued assignments]

An assignment sv-ass is locally consistent at a variable v if and only if for every $y \in$ sv-ass(v) and for every constraint $<N,R,(w_1,\ldots,w_{i-1},v,w_{i+1},\ldots,w_n)>$ where v occurs in the variable list, there are values $x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n$, where $x_j \in$ sv-ass(w_j), such that $(x_1,\ldots,x_{i-1},y,x_{i+1},\ldots,x_n) \in R$

∎

Destructive propagation algorithms now proceed as follows:

- Initially assign to every variable its entire domain, or some appropriate subset thereof.

- If, at some constraint, a restriction upon the values for a variable can be computed, remove the corresponding values, and make this restriction available at all other constraints involved.

Such algorithms can be used to derive assignments that are locally consistent at every variable. As we have yet mentioned, this does not necessarily imply that the resulting set-valued assignment comprises admissible single-valued assignments. Clearly, if the domains and relations are infinite, managing set-valued assignments requires some effective finite representation of infinite sets.

In a later chapter, we will see that the scene labelling problem fits into the general scheme of constraint problems in two different ways, where either the junctions or the segments are considered as variables, the domains being given by resp. the total of composite junction labels or simple segment labels. We will see that the specifities of constraint relations induced by labelling problems allow for an efficient representation and corresponding propagation algorithms.

3.3 Some Related Work

Guy Steele from the M.I.T. has developed and implemented a general-purpose constraint system for almost-simple, integer valued constraints, based on constructive, local propagation [Steele-80]. The truth maintenance system by David McAllester [McAllester-80] works with constraints upon the truth values of propositional formulae, as induced by the truth value tables for logical connectives. The basic structure of this system is closely related to Steele's system. The M.I.T. research on constraints has been deeply influenced by the pioneering work of Stallman and Sussman [Stallman|Sussman-79] on the use of constraint propagation for electrical circuit analysis and design. A number of succesor systems like e.g. QUAL by Johan deKleer [deKleer-79] uses similar techniques. Another system for handling algebraic constraints is due to James Gosling [Gosling-83] from CMU.

The ISIS-group at the robotics institute of CMU used constraints to attack the job-shop-scheduling problem [Fox et al-83] which is known to be NP-complete. Constraint-directed reasoning is also popular in planning [Stefik-81] and simulation [Borning-79].

The combinatorial aspects of finite constraints are discussed in two papers by Haralick and Shapiro [Haralick|Shapiro-79,80] on what they call the 'Consistent Labelling Problem'. Among others, the work of Mackworth [Mackworth-77], Montanari [Montanari-82], and Freuder [Freuder-82] is also important, not to forget the original Waltz-paper [Waltz-72]. John Gaschnig [Gaschnig-79] provides an extensive discussion of the efficiency of several search algorithms, including Waltz-filtering, w.r.t. constraint problems. So does Bernard Nudel [Nudel-83].

The relaxation labelling approach taken by Rosenfeld, Hummel, and Zucker [Rosenfeld|Hummel|Zucker-76] can be seen as the continuous equivalent of discrete labelling algorithms.

4 Societies of Agents

As I have mentioned in an previous chapter, no overall control strategy like e.g. hierarchical top-down or bottom-up approaches has been proven to be superior for vision. However, one thing is clear: vision involves massive computation and reasoning processes on various representational levels. It seems straightforward to attack this complexity problem by concurrent processing and to exploit a parallel vs. sequential processing speed-up and, in fact, some existing vision systems as well as concepts for further developments make extensive use of parallelism. The nature of parallel solutions depends on the level of detail on which they are applied.

- For low level image processing, massively parallel hardware architectures such as e.g. SIMD array processors apply, using e.g. one processer per pixel to perform uniform operations. A survey on parallel processing for low level vision can be found in [Davis|Rosenfeld-80].

- Concerning high level vision, a specialization of a few processors to some speficic aspects of the vision problem can be made, e.g. concurrently and interactively co-operating line-finders and drawing-interpreters, or e.g. specialists for illumination, texture etc...

- On intermedtiate levels such as line drawings, frequently algorithms on graph structures allow for the introduction of parallelism. The nodes of such a graph stand for components of the drawing like regions or junctions, and the links reflect the adjacency of these components. Following Douglass [Douglass-82], we call such algorithms image graph algorithms.

4.1 The Paradigm

For discussing intermediate and high-level parallelism, the conceptual view of societies of co-operating agents has become increasingly important. Such a society is composed of a number of independent units, the agents, that do some local problem solving of their own, and that exchange results and requests by a (mostly asynchronous) communication mechanism. There is no superior instance to guide the overall flow of reasoning. This paradigm relates to the object-oriented programming approach, which plays a prominent role not only within the AI community, and to algebraic specification techniques using abstract data types. Thus, the concept of societies of agents can be seen as the problem solving-level counterpart of related concepts on the programming language resp. the abstract specification level.

I now briefly outline a simple model of societies of agents, to be refined in a later chapter, which serves as a basis for an informal introduction of the language CSSA, given in the next section. We assume the acquaintances between agents to be directed, i.e. the topologic structure of such a society is that of a directed graph. At a given time instant, each agent is either inactive, being in a stable state from a given universe of possible states it may be within, or it is busy, performing a non-separable transaction from one stable state to another. Such transactions can only started by requests from outside the agent, i.e. by the receipt of a message. During such a transaction, an agent is free to send messages to acquainted agents. Message transmission is considered time-consuming, with unknown but non-null transmission times. Messages are only accepted in stable states. We therefore assume some implicit buffering mechanism, a so-called mailbox, to store messages until the addressee is ready to accept them. Intuitively, this buffer can be regarded as a mailbox, the basic action cycle of an agent being to look in his mailbox and select a message, to work on this message, possibly issuing some messages of his own, and then to look again into the mailbox (figure 4.1).

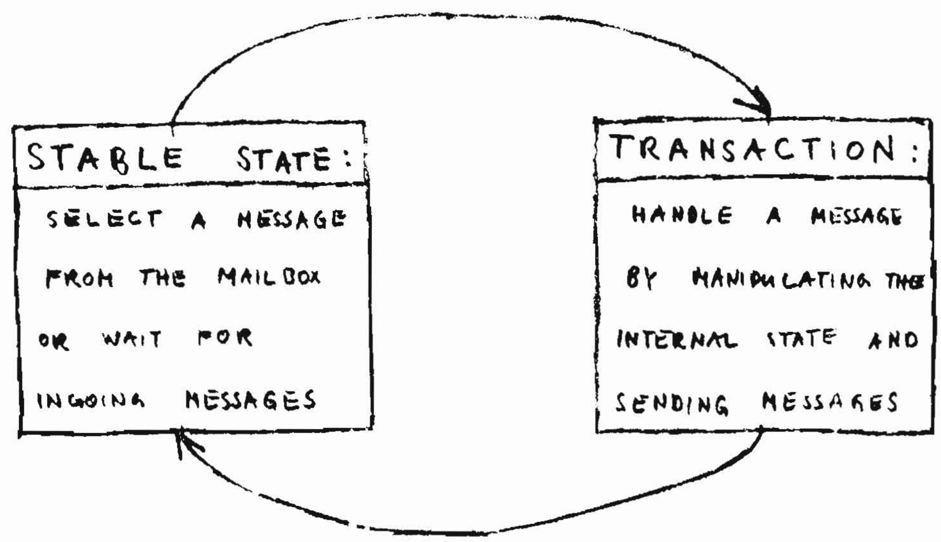

STABLE STATE:
SELECT A MESSAGE FROM THE MAILBOX OR WAIT FOR INGOING MESSAGES
TRANSACTION:
HANDLE A MESSAGE BY MANIPULATING THE INTERNAL STATE AND SENDING MESSAGES

Figure 4.1: Action cycle of an agent

We do not assume any regular transmission order as e.g. first-sent/first-received or similar orders, but we require that every message sent is eventually received. Note that this requirement involves the message passing technique, mechanisms for selecting a message from a mailbox, as well as the internal behavior of the agents themselves in that we must exclude infinite transactions.

We do not go into the details of a proper, elaborate semantical theorie of such societies of agents. Indeed, I do not know any such theory that is completely satisfactory, but two problems of particular importance should be addressed: the starting problem, and the termination problem. The starting problem is the following: where do the agents come from and as they may not act spontaneously, where do the initially activating messages come from. We assume that agents can be created by other agents, and that there is one superior meta-agent being able to act spontaneously. Note that this makes the network completely dynamic, together with the hitherto unmetioned postulate that acquaintances be communicable.

The termination problem is twofold: when does an agent end to exist, and what does it mean that a society of agents is terminated? In chapter 9, we will see that a proper specification of 'termination' is a non-trivial affair, and that checking for such a condition is both a delicate theoretical problem and an issue of practical importance. Basically, the difficulties result from the absence of a common, global system state. Properly speaking, the existence of such a state can be postulated but it is not directly observable. We will not consider the problem of 'dying' agents, but assume that every agent exists at least as long no appropriately defined termination condition holds and is detected.

4.2 CSSA:Computer System for Societies of Agents

It is not the purpose of this paper to discuss the concrete realization of a society of agents. However, I think that an informal introduction of the implemetantion language of SCENELAB, CSSA, will help to clarify some of its aspects. CSSA [Beilken|Mattern|Spenke-82] is a fully implemented Computer System for (realizing) Societies of Agents.

The behavior of an individual CSSA-agent is determined by a so-called script, formulated in an imperative language in the tradition of PASCAL. The local state of an agent is given by the values of variables, state transitions are realized by the side-effects of operations. An agent is able to perform a number of such operations, where the effects of an operation are modifications of the agents variables and, possibly, the sending of messages. From a programming language point of view, an operation is a block of code, possibly structured into subblocks such as procedures or functions, and possibly having some local variables, quite similar to a usual procedure. An operation may only be invoked by a message that is composed of the operations name, and a list of actual parameters that matches the formal parameter list. An CSSA-agents execution cycle corresponds to the cycle shown in figure 4.1.

The user itself is an integral part of a CSSA-society, in the form of a special interface agent . This interface agent is the only agent that may take actions of his own, and hence plays the role of the meta-agent mentioned in the preceding section. All other agents must be created by a special generative command, according to some pre-defined script. The initial states of these agents are defined by implicit default values or by explicitely specified values of their variables. The resulting network is dynamic in that agents may not only be created but also aborted, and in that acquaintances can be communicated at run-time.

An early version of CSSA has been running on a single-CPU SIEMENS BS2000 machine. The CSSA-code is translated to an intermediate virtual stack machine, running on a SIMULA multiple processor simulation system. This system simulates an asynchronous processing environment with flexible parameters, such as number and interconnection of physical processors and message transmission routes and times etc... A distributed operating system is running on the simulator.

Currently, a realization of CSSA on a real multicomputer network composed of sevarl 32-Bit Charles River machines is under development, and a first prototype has now been completed. This reasearch and development is part of the INKAS-project at Kaiserslautern University [Nehmer et al-85]. A brief overview on CSSA can be found in [Mattern|Beilken-85]. For full details, especially for the many aspects of CSSA not mentioned here, see [Beilken|Mattern|Spenke-82]. A collection of CSSA-programming examples is given in [Voss-82].

5 SCENELAB

5.1 The Key Idea and the System Kernel

The key idea of SCENELAB is to realize Waltz-like filtering procedures by a society of cooperating agents. The problem of labelling line drawings from trihedral scenes has been taken as sample application area to exploit this idea.

The kernel of SCENELAB is constituted by a network of agents that is 'isomorphic' to the line drawing in that there is one agent per junction, agents that work on adjacent junctions being acquainted to each other (figure 5.1).

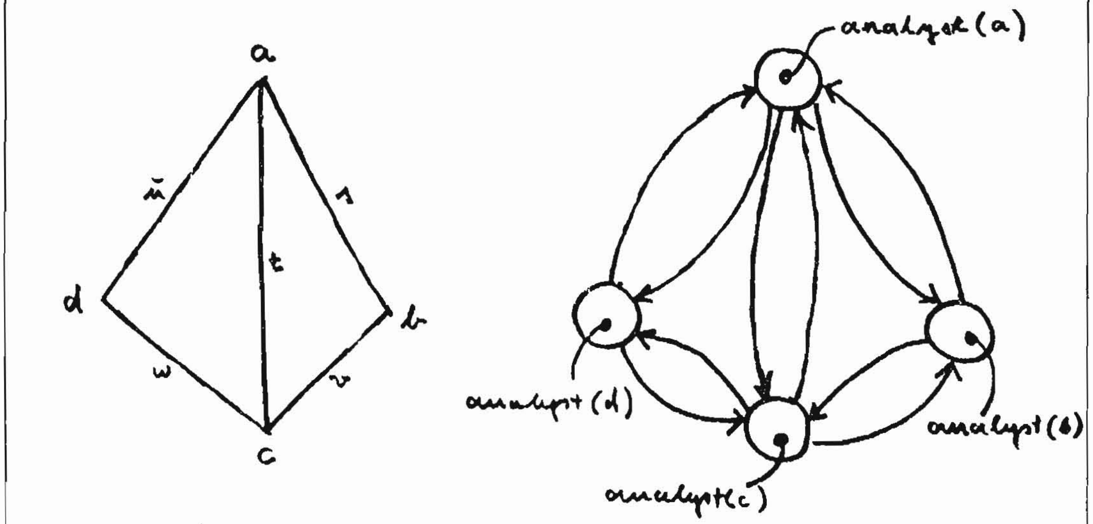


Figure 5.1: SCENELAB works with a network of cooperating agents that is isomorphic to the line drawing

Basically, the task of such an agents is to keep the labelling of its own junction compatible with those of all of its neighbors.

This goal is achieved by the following realization of a Waltz-filtering algorithm. Initially, every agent is supplied with a list of potentially possible junction labels for the type of junction it is processing. These labels restrict the possibilities of assigning segment labels to the segments joining at that junction, as seen from that particular viewpoint. If some specific segment label is ruled out by the current set of junction labels, the agent sends a message to the agent working on the opposite junction of the segment in consideration. This message says, the addresse should remove from its current label set all those junction labels that induce a segment label for that segment that corresponds to the label that has been eliminated. When receiving such a message, an agent performs the appropriate deletions and then checks whether these deletions impose additional constraints upon some other segments, i.e. whether some label has become impossible for a segment while it has been possible before the transaction. If so, the agent forwards corresponding messages to the neighbors involved (figure 5.2).

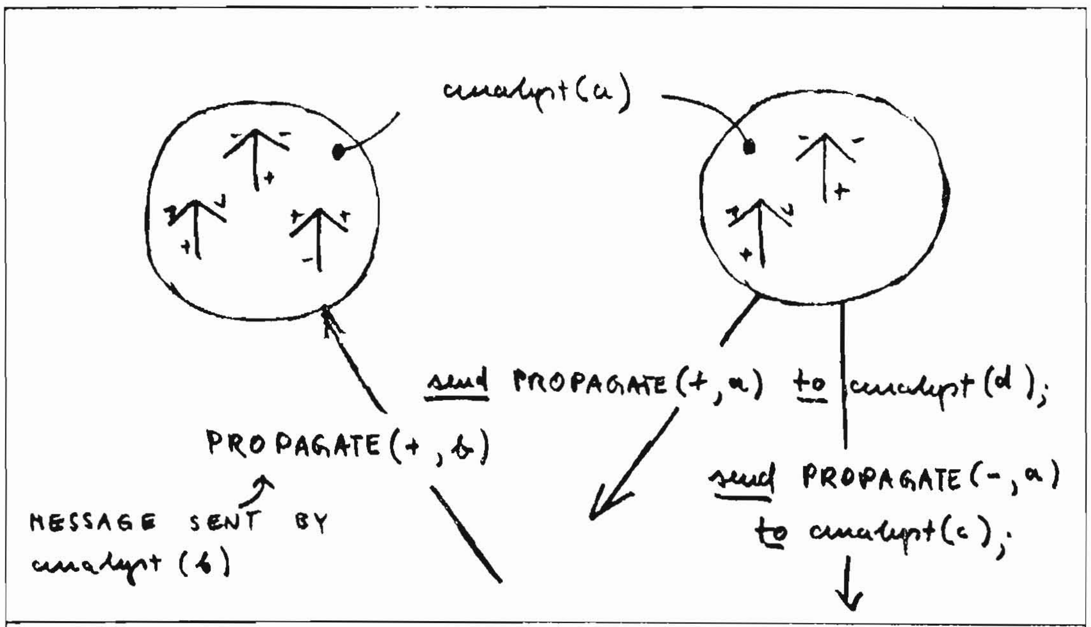

analyst(a)
send PROPAGATE(+,a) to analyst(d);
PROPAGATE(+,b)
MESSAGE SENT BY analyst(b)
send PROPAGATE(-,a) to analyst(c);

Figure 5.2: An agent evaluates and propagates constraints

5.2 Architecture and Interfaces of SCENELAB

The kernel constraint network is embedded in a computing system composed of several additional agents (figure 5.3). First of all, there is the SUPERVISOR being responsible for the overall control of the system. The SUPERVISOR communicates with the user of the system and generates and monitors the constraint network. The user specifies a picture description and a label dictionary in a special description language PDL. He is completely free in defining junction figures and labels of his own. This makes SCENELAB no general purpose constraint system, but it enables the user to define and manipulate arbitrary constraint problems that exhibit the same structure as the scene labelling problem, i.e. finite constraint relations where the compatibility between composite node labels reduces to a one-to-one compatibility between edge labels.

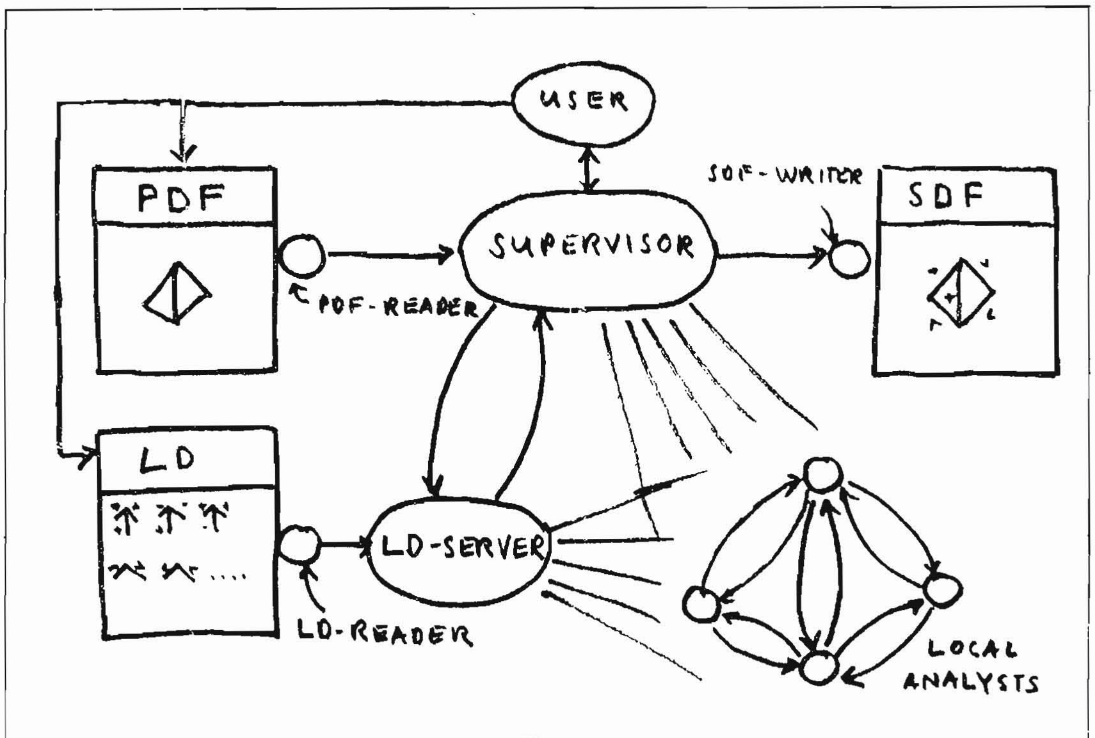


Figure 5.3: Architecture of SCENELAB

Here are the PDL-definitions of the junction figure ARROW, and of a 'figure' OR, representing a logical disjunction.

```
def-lab +;-;o>;>o.               { The basic segment labels }
spec-com +:+;-:-;o>:>o;>o:o>.  { Compatible segment labels }
def-fig ARROW.                   { A junction figure }
spec-deg ARROW:3.                { Degree of an ARROW }
spec-lab ARROW: +/-/+,+/+/+,o>/+/>o.
                                 { Admissible ARROW-labellings }


def-lab tt;ff.
spec-com tt:tt;ff:ff.
def-fig OR.
spec-deg OR:3.
spec-lab OR: ff/ff/ff,ff/tt/tt,tt/ff/tt,tt/tt/tt.
```

The following PDL-statements specify a part of the tetraeder shown in figure 5.1, and a related network representing a logical connection.

```
def-seg s;t;u;v;w.
def-jun a;b;c;d.
spec-fig a:ARROW; b:ELL; c:ARROW; d:ELL.
spec-gra a: s/b,t/c,u/d.

def-seg p;q;r;s;t.
```

```
def-jun a;b;c;d.
spec-fig  a:OR; b:2-IDENT;c: AND; d:NEG.
spec-gra a:p/b,q/c,r/d.
```

where 2-IDENT could be defined as

```
def-fig 2-IDENT.
spec-deg 2-IDENT:2.
spec-lab 2-IDENT:tt/tt,ff/ff.
```

Note that in the latter example, the segments play the role of propositional formulae. To overcome the restriction that every segment is constrained by exactly two junctions (here: every formulae is involved in exactly two other fomulae), an n-IDENT constraint can be used to replicate segments.

The picture description file PDF and the label dictionary LD are logical files which can be linked either to an interactive input device or to some predefined descriptions from a library. For scene labelling purposes, there typically will be a predefined standard label dictionary available, while the picture description is interactively supplied by the user. PDF and LD contain PDL-representations of line drawings and labels, which are incrementally translated into an internal representation. PDL as well as the internal structures are realizations of the abstract model to be discussed in the next part of the paper.

The LD-SERVER shown in figure 5.3 is a special agent managing a library of dictionaries and the dictionary currently in use. It supplies the agents in the constraint network with their initial labellings. The user controls the system by sending commands to the SUPERVISOR. The following sequence of commands would result in the construction of a constraint network to label a drawing being described on a PDF linked to a special reader agent PDF-READER, using a label dictionary residing on a file read by the LD-READER.

```
send INTERPRETE(PDF-READER) to SUPERVISOR;
send INTERPRETE(LD-READER) to LD-SERVER;'
 { Tell the SUPERVISOR and the LD-SERVER where to get their
 input data }

send INIT-NET to SUPERVISOR;
 { The SUPERVISOR will create a constraint network and
 supply the agents with their initial information as e.g.
 acquaintances and type of junction they have to work on. The
 agents then will send requests for appropriate initial labellin
 to the LD-SERVER }

send START-NET to SUPERVISOR;
 { Activate the constraint network }.
```

The user may also impose additional constraints by means of messages during the constraint process, excluding certain labellings, e.g. to enforce a unique labelling when the result is ambiguous.

```
send EXCLUDE(+,a,s) to SUPERVISOR;
 { The SUPERVISOR will sent appropriate messages to the
 agents working on the junctions joined by the segment s
 to rule out the label '+' for s, as seen from the junction a }
```

Furthermore, there are various inspection and control commands to display intermediate results, or to trace the constraint propagation process. An interactive error handling facility allows for correcting errors without restarting from the scratch. When an error occurs when reading from a file, the input stream is automaticallly switched to the interface agent, i.e. to the user, who is asked to correct the faulty input or to break the session. Appendix B gives a complete specification of PDL and lists the most important SCENELAB commands available to the user.

5.3 Correctness and Termination - Some Informal Arguments

Sequential and synchronized parallel versions of this algorithm exhibit simple criteria for correctness and termination [Waltz-72], [Rosenfeld|Hummel|Zucker-76]. Basically, the same arguments apply to the asynchronous case, too, but some technical overhead is needed to provide formal proofs. Before we turn to a somewhat more rigorous treatment of the problem, I would like to give some informal arguments that the system always terminates, providing the desired result.

First, notice that every inconsistency occuring at a junction is resolved by deleting the labels that are responsible for this inconsistency. Second, only necessary deletions are made. Deletions may only occur as the result of a message, and messages are not sent until some label must be excluded for local reasons. This shows that a label is removed if and only if it cannot consistently reside at a junction, hence the resulting labelling must be the greatest locally consistent labelling contained in the initial labelling.

Furthermore, both the constraint network and the label dictionary are finite, i.e. only a finite number of deletions may occur and, consequently, only a finite number of propagation messages will ever be sent. As message transmission times and transactions are finite, the system must eventually halt and come to a stable state.

However, although system termination seems to be obvious and, in fact, will be shown in a later chapter, how can we find out whether the network has terminated or not?

5.4 Distributed Termination Detection

The problem arises from the fact that, on one hand, termination is clearly a global feature. On the other hand, every agent only has a restricted local view of the entire state of affairs, and agents can only be inspected by message passing, i.e. only one agent is accessible at a time. Henceforth, although one can postulate a global system state, this state is not directly observable.

Another question hereby is, what does it mean that a system of asynchronously co-operating agents has terminated? This question seems odd first, but it turns out to be a non-trivial issue. Basically, there are two different views of that problem. Firstly, an event-oriented approach says, the system is terminated if no agent is currently active or will become active in the future. This means that every agent is in a stable state and that no message is currently on the way between two agents. Since no agent may become spontaneously active, only messages coming from outside the system can re-activate an agent. Notice that such messages relativize the condition 'no agent will become active in the future' to the system being considered as closed.

Another, state-oriented approach says, the system has terminated if it has reached a state providing the desired result. Here, some care must be taken that the termination condition on states is persistent in the sense that, whenever a state satisfying that condition is reached, no subsequent state may violate the condition, except those induced by external messages. Complete local consistency e.g. is such a persistent condition in our constraint network. And, as messages may be on the way that do not lead to further deletions, this condition does not imply that the system is terminated w.r.t. an event-oriented definition.

The trouble with any such termination condition, either state- or event-oriented, is that it usually does not subdivide into a conjunction of local conditions. In both cases, simply asking around every agent for his local termination, according to either condition, does not solve the problem. An agent saying, ok, I am done, will usually not be able to exclude that it could be re-activated by the receipt of a message that was on the way when the query was broad-casted (figure 5.4).

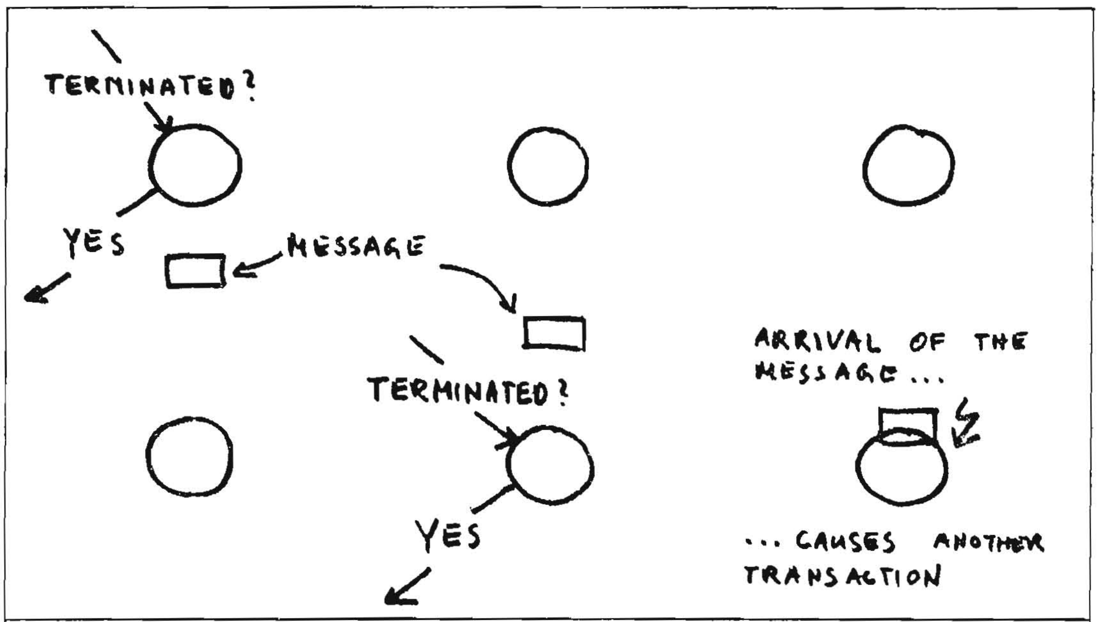


Figure 5.4: Simple broad-casting does not solve the distributed termination problem

Note that the condition 'will not become active except for messages coming from outside', considered for a single agent, translates to 'will not become spontaneously active'. Thus, from an event-oriented point of view, a single agent trivially fulfills a local termination condition whenever it is in a stable state.

In most reasonable cases, a more or less costly overhead will be needed to detect the termination of a distributed system working along the lines sketched in chapter 4. In [Beilken|Mattern|Reinfrank-85], we discuss the general termination detection problem and review some approaches thereto that have become known in the literature. In particular, several practical algorithms due to Christian Beilken and Friedemann Mattern are presented, which have been realized in several CSSA-programs running at Kaiserslautern, including a constraint propagation system for cryptarithmetic, as described in [Kornfeld-81]. I should point out that most of the vocabulary on distributed termination used in this section evolved from various discussions with Christian and Friedemann.

5.5 Freeze-and-Check

In SCENELAB, a brute-force method for termination detection has been chosen. Based upon the fact that complete local consistency is persistent, SCENELAB proceeds as follows. First the constraint system is frozen by sending appropriate STOP-messages to every LOCAL ANALYST, that break their constraint processing when receiving such a message. This is initiated by the command

```
     send CHECK-NET to SUPERVISOR;
```

The sequence in which these stops occur cannot be specified but, after a while, all of the LOCAL ANALYSTs will have interrupted constraint propagation. The local label sets - in spite of the fact that they have been evaluated within different intervals of time - determine a global labelling that is now checked for local consistency. To do this, every agent simply asks each of his neighbors, "is may labelling ok for you', i.e. for every segment label being possible from his point of view, he asks for a compatible label at every neighbor. Waiting for the answers requires some synchronization, which can be easily realized by using counters. The LOCAL ANALYSTs then tell the SUPERVISOR whether the labelling is locally consistent at their junctions. Depending on the answer, the SUPERVISOR resp. the user can decide e.g. to restart the constraint propagation, or to generate an output file containing the PDL-description of a labelled drawing. This is done resp. by the commands

```
     send RESTART-NET to SUPERVISOR;
   or
     send GENERATE-OUTPUT to SUPERVISOR;
```

The cautious reader may have noticed that the concept of 'freezing' does not neatly fit into our model of how CSSA realizes societies of agents. Freezing the constraint propagation means that the local agents temporarily do not accept for processing any PROPAGATE message but leave them unopened in the mailbox until a RESTART message is received. This involves the fact that a message with the same pattern once matches an operation of the adressee, and once does not. In fact, CSSA provides an additional facility, the facettes, a facette being an internal state of an agent where only a subset of all of its operations is available. A transaction then may also include the transition from one facette into another. A STOP-message simply makes an agent go into a facette where no PROPAGATE-operation is known. However, this significantly complicates our model of CSSA, and leads to substantial conceptual problems that lie considerably beyond the scope of this paper. Therefore, we should content ourselves with the informal notion that the LOCAL ANALYST leaves any incoming PROPAGATE message unopened until he has received a RESTART-message. This is less severe than could be assumed, because such a behavior could be simulated in our simple CSSA-model, too. To achieve this, let the STOP-message set a flag which can be reset by the RESTART-message. The PROPAGATE-operations then can be modified such that, as long as the flag is set, the agent does nothing but send a duplicate of this message to ıtself.

6 A Model of the Picture Domain

In this chapter, I am going to work out what knowledge we need about a line drawing as an input for a labelling procedure. Several levels of representation will be introduced, with various degrees of precision, a basic level, a topologic level, a geometric level, and a numeric level.

6.1 The Basic Level

At a basic level, I postulate a clear notion about what a line drawing is: a line drawing is composed of a number of straight line segments S, junctions J, and regions R, being arranged in some particular way. We will only consider drawings without isolated elements, especially without isolated or dead-end segments. Every segment joins exactly two junctions, and at every junction, two or three segments meet. This restriction partially reflects the requirement that the drawings be perfect drawings of trihedral scenes. However, perfectness and trihedrality qualify the scene and the relationship between the scene and the drawing, so it cannot be fully represented in the picture domain.

6.2 The Topologic Level

Most of the knowledge we need to label a line drawing is of topological nature, i.e. how are the primitives grouped together to configurate a line drawing? To capture this information, we define two kinds of picture graphs.

Notation [unordererd tuples]

Given a carrier set M, let $[x_1,x_2,...,x_n] \in M^n/\sim$ denote an un-ordered n-tuple of elements of M, with possibly multiple occurences of single elements.

■

The graph theoretical vocabulary used here is largely taken from [Bondy|Murty-76].

Definition 6.1 [junction picture graph]

For a given line drawing with segments S and junctions J, let an incidency function μ be defined as follows:

$$\begin{aligned} \mu:\ & S \rightarrow J^2/\sim \\ & s \mapsto \mu(s)=[a,b] \end{aligned}$$

where $\mu(s)=[a,b]$ if and only if the segment s joins the junctions a and b. The resulting undirected graph $G=(J,S,\mu)$ is called the junction picture graph of the drawing.

■

Notice that the junction picture graph of a perfect drawing of a trihedral scene is necessarily both simple and planar. Furthermore, every junction has a degree of 2 or 3.

Notation [degree - incidencts - adjacents]

> Given a graph G and a vertex v, let deg(v), inc(v), and adj(v) denote, respectively, the edge degree of v, the segments incident to v, and the junctions adjacent to v.
>
> ■

A related graph can be defined for the regions of a drawing.

Definition 6.2 [region picture graph]

> For a given line drawing with regions R and segments S, let an incidency function ν be defined as follows:
>
> $$\begin{aligned} \nu\colon\ S &\rightarrow R^2/\sim \\ s &\mapsto \nu(s)=[x,y] \end{aligned}$$
>
> where $\nu(s)=[x,y]$ if and only if the segment s separates the two regions x and y. The resulting undirected graph $H=(R,S,\nu)$ is called the region picture graph of the drawing.
>
> ■

Notice that a region picture graph is usually not simple (see figure 6.1). Since we do not consider region labellings in our example, we will focus on junction picture graphs.

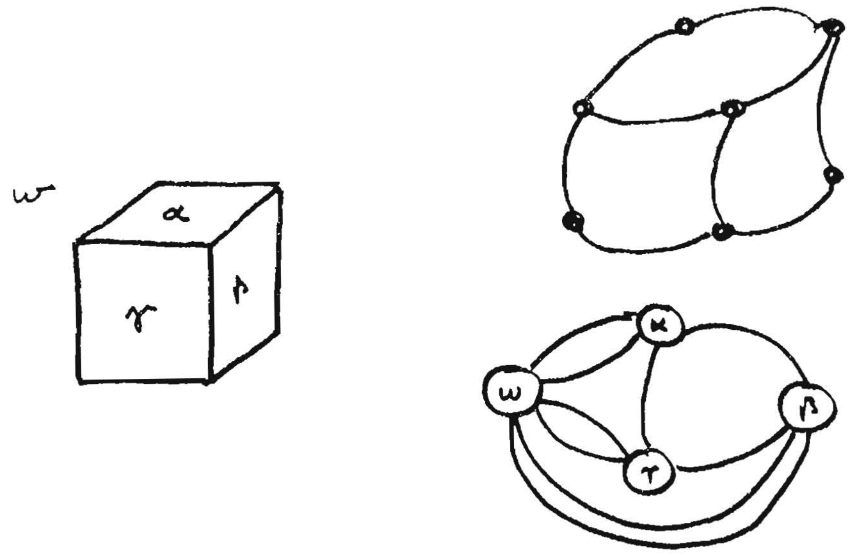


Figure 6.1: The junction picture graph, and the region picture graph of a drawing of a cube

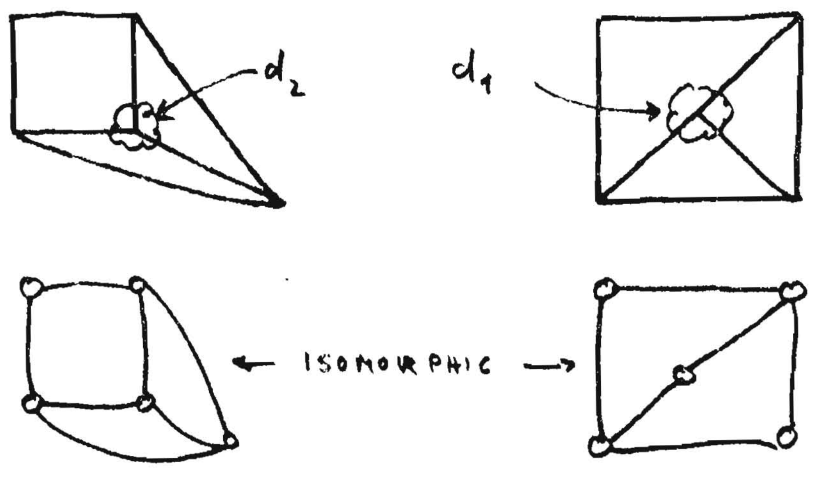


Figure 6.2: Two significantly different line drawings having isomorphic junction picture graphs

6.3 The Geometric Level

Unfortunately, line drawings exhibit some significant features that are not represented in the picture graphs but that we need to achieve proper labellings. Although e.g. the two line drawings shown in figure 6.2 have isomorphic picture graphs, they are quite differently to be labelled. These drawings have the same topological structure at the junctions d_1 resp. d_2, but the geometric shapes of d_1 and d_2 substantially differ in that d_1 has two segments co-linear while d_2 has not. In our vocabulary, d_1 is a TEE-typed junction and d_2 is an ARROW-typed junction. As we have already discussed in an earlier chapter, we call such typical intersection patterns of segments at junctions junction figures. We also have argued that such a figure has some physical meaning in that different figures result from the objects in consideration and from the particular viewpoint from which we look at these objects. However, this physical meaning cannot be directly represented in the picture domain. We therefore introduce figures independently from their physical interpretation.

Definition 6.3 [junction figure classification]

A junction figure classification is a partition of all possible junctions in the (plane) surface the drawing has been drawn within s.t. the following holds:

Junctions that belong to the same class have the same edge degree. This degree is referred to as to the degree of the figure.

Given a figure classification, the class of an arbitrary junction is effectively computable.

■

Notation [figure of a junction].

Given a figure classification $F=\{f_1,...f_n\}$, and a junction a, we denote the figure of a w.r.t. to F by $fig(a)=f_i$, for some i.

■

Notice that the equivalence relation induced by a equiv b if and only if fig(a)=fig(b) is a refinement of the equivalence relation induced by deg(a)=deg(b). For scene labelling purposes, such a figure classification is usually based on some geometric properties of the junctions. We use the classification F = { ARROW, ELL, FORK, TEE}, being defined as follows:

- ELL: any two line junction
- ARROW: any three line junction in which there are two segments such that for each of these segments, the other two segments lie on the same side of it

- TEE: any three line junction having two of its segments co-linear

- FORK: any other three line junction

Clearly, these descriptions provide a basis for an effective computation of the class any two or three line junction belongs two. To rigorously fulfill the requirement that a figure classification partition all possible junctions in a plane we would have to introduce additional figures, say DEG-4, DEG-5, etc., that subsume all non-trihedral junctions.

Frequently, junction figures allow for a distinction between uniquely determined segments of junctions of that class, as e.g. the 'shaft segment of an ARROW'. Such a distinction between particular segments is valuable for scene labelling, since different segments usually may be differently labelled, so we want to make this knowledge explicit in our representation. We introduce an ordering upon the segments of trihedral junctions according to the following convention. Firstly, ordering is always clockwise. Secondly, if there is one region bounded concavely by the segments of a junction of a particular class, we begin to count at the first segment to the right (in clockwise direction) of that region. Note that there is either no or exactly one such region. Junction figures having this property are called ordering figures, others are called semi ordering. In case of semi-ordering figures we start at an arbitrary but fixed segment and consider all cyclic permutations of the resulting order.

Classifying a junction, say x, according to its figure fig(x) thus induces an ordering resp. several orderings upon both inc(x) and adj(x). Notice that all but one of the Huffman Clowes figures (the FORK) are ordering (figure 6.3).

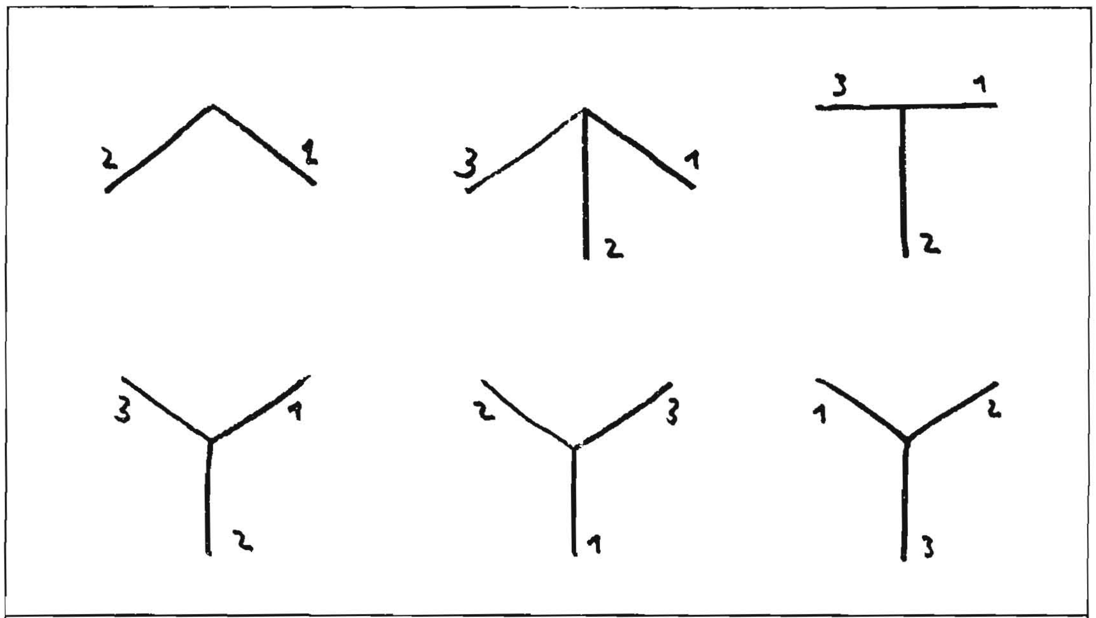


Figure 6.3: All but one of the junction figures allow for a unique ordering of their segments

We are now ready to define a structure that represents all the knowledge we need to label a line drawing. It is based on the junction picture graph, and on a classification of the junctions of that graph. Furthermore, we extend the incidence function by some explicit ordering information, as available through the figures:

Definition 6.4 [classified ordered picture graph]

Given the junction picture graph $G=(J,S,\mu)$ of a line drawing, and a figure classification fig: $J \rightarrow F$, we define its classified ordered picture graph , copg in short, as a quintuple

$H=(J,S,F,\text{fig},\text{graph})$
graph: $J \rightarrow (S \times J)^n/\sim$

where $\text{graph}(a)=((s_1,b_1),\ldots,(s_n,b_n))$ if and only if
$n=\deg(a)$
and for $i=1,2,\ldots,n$, $\mu(s_i)=[a,b_i]$,
$(b_1,\ldots,b_n)$ being an ordering induced by fig(a) upon adj(a).

If fig(a) is a semi-ordering junction, we take an arbitrary but fixed one of its orderings.

∎

Notice that n is variable in the definition above. A rigorously proper definition could be achieved e.g. by using the degree as an additional parameter of graph, or by defining separate functions graph-n for the junctions of degree n, graph being the union of all these functions. However, I do not want to be excessively formal and omit some formal rigor for the sake of simplicity.

6.4 A Note on Qualitative and Quantitative Numeric Descriptions

Although a copg provides a finer-grained representation of a line drawing than a pure picture graph, it is still a very crude level of abstraction. No distinction can be made e.g. between the two drawings shown in figure 6.4.

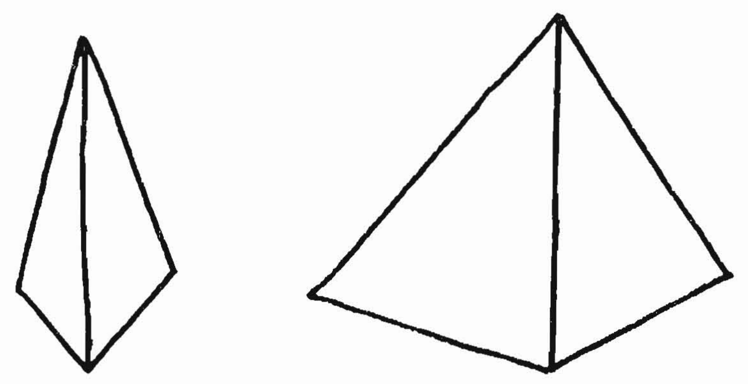


Figure 6.4: Two drawings that cannot be distinguished by Huffman/Clowes labellings

Another level of detail could be achieved by representing numeric coordinates w.r.t. to some underlying coordinate system. However, although such numeric values are usually available since the basic images are represented as pixel arrays, they are generally inadequate for generating semantic descriptions of a scene being represented by a drawing. Instead of saying, the angle of the peak of an ARROW measures 37,5 degrees, while that one of another measures 105, descriptions like the ARROW is peaked resp. very smooth are more expressive and adapted to a human user of the system. When working on such a qualitative level, appropriate constraints can also be defined and evaluated for picture processing. David Waltz [Waltz-72] has used e.g. a very crude qualitative distinction between shadowed and illuminated regions - instead of concrete illumination values, and his labelling procedure exploits constraints like 'if two regions are separated by a crack line, they must be either both shadowed or both illuminated'. The use of qualitative representation and related reasoning facilities is currently one of the hottest topics in AI [AI24-84].

7 Labellings

7.1 The Label Dictionary

Our final goal is to find a unambiguous assignment of labels to segments, labels standing for certain properties of the corresponding edges. As we have discussed in an earlier section, we always look at such a segment from one of its endjunctions, and that sometimes, the choice of viewpoint is significant. However, for every aspect of the segment that we consider in our application, the views from two opposite junctions are either identical or complementary.

Thus, we work with a finite set of segment labels

$$\Sigma=\{\sigma_1,...,\sigma_n\}$$

where there is a symmetric one-to-one compatibility relation $\equiv$ between labels. This compatibility induces a bijection

$$\text{match}: \Sigma \rightarrow \Sigma$$
$$\text{match}(\sigma)=\rho \text{ if and only if } \sigma\equiv\rho$$

Note that the symmetry of $\equiv$ means that match is equal to its inverse function match^{-1}, hence we have

$$\text{match}(\text{match}(\sigma))=\sigma$$

Different junction figures allow for different labellings, and the ordering of the segments usually, in the case of ordering figures, makes some difference. Thus, for every figure in consideration, there is a number of possible composite junction labels represented as ordered n-tuples of segment labels, n being the degree of the figure.

Definition 7.1 [dictionary page]

Let Σ be a set of segment labels, f be a figure from a given total F of figures. A dictionary page L(f) is a finite number of junction labels $\underline{\sigma}$ of the form

$$\underline{\sigma} = (\sigma_1,...,\sigma_n) \in \Sigma^n$$

where n is the degree of f. Notice that we use underlined greek letters for junction labels.

A label dictionary then is given by a number of dictionary pages, one for every figure in consideration.

∎

We do not require that the dictionary pages be disjoint. For such a dictionary, we write L(F) and omit F if it is clear what figure set is meant.

Notice that the only time a physical meaning of labels is involved is the time when the dictionary is generated. Once is it completed, a labelling procedure treats them as simple symbolic entities. In the case of semi-ordering figures, care must be taken to include every cyclic permutation of each of the labels.

The compatibility between segment labels extends to junction labels in the following canonical way:

Definition 7.2 [compatible junction labels]

Let $\underline{\sigma} = (\sigma_1,\ldots,\sigma_m)$ and $\underline{\rho} = (\rho_1,\ldots,\rho_n)$ be two junction labels, and let $k\in\{1,\ldots m\}$ and $l\in\{1,\ldots n\}$ be two indices. $\underline{\sigma}$ is said to be k/l-compatible with $\underline{\rho}$ if and only if

$$\sigma_k \equiv \rho_l$$

We write $\underline{\sigma}\ k{\equiv}l\ \underline{\rho}$

■

The definition of a set-valued junction labelling is straightforward:

Definition 7.3 [junction labelling]

Given the copg H=(J,S,F,fig,graph) of a line drawing, and a label dictionary L(F), a junction labelling is a function

$$\begin{aligned} I:\ & J \rightarrow \mathcal{P}(L(F)) \\ & a \mapsto I(a) \subseteq L(fig(a)) \end{aligned}$$

■

Defintion 7.4 [labelling problem]

A pair LP=(H,L), where H is a copg of a line drawing, and where L is an appropriate label dictionary, is called a labelling problem.

■

Notation [special labellings]

Given a labelling problem LP=(H,L), we denote the total of all possible labellings of H from L by LP-I.

The empty labelling I, where $\forall a\in J: I(a)=\{\}$, is called the null-labelling I-NULL.

The labelling I that assigns to every junction a its entire dictionary page L(fig(a)) is called the initial labelling I-INIT.

The mapping I, defined as $\forall a\in J: I(a)=\Sigma^n$, where n=deg(a), is called the label universe I-UNIV of a junction. Notice that I-UNIV is no labelling in the proper sense of the word.

■

7.2 Consistent Labellings

In this section, we rigorously re-define the notions of admissibility and consistency of labellings, that have been informally introduced in the preceding chapters. Based upon these definitions, we will be able to show that, in fact, every labelling problem has a well-defined result.

Definition 7.5 [admissible labellings]

Let LP be a labelling problem, and I∈LP-I a labelling. I is admissible if and only if

(1) $\forall a \in J$: $|I(a)|=1$
(2) Let u, v be two adjacent junctions, where u is the p'th neighbor of v and v is the q'th neighbor of u. Let furthermore $I(u)=\{\ \underline{\leqslant}\ \}$, and $I(v)=\{\ \underline{\varrho}\ \}$. Then $\underline{\leqslant}\ q{\equiv}p\ \underline{\varrho}$ holds.

■

Notation [matching labels]

In the situation described in the definition above, we say that the label $\underline{\leqslant}$ is matched by $\underline{\varrho}$.

■

In the context of Waltz-like filtering procedures, the concept of local consistency plays a central role.

Definition 7.6 [local consistency]

Let LP be a labelling problem, a∈J a junction, where

$$graph(a)=((s_1,b_1),\dots,(s_n,b_n)),\ n=deg(a)$$

We say that a labelling I∈LP-I is locally consistent at a if and only if for every $\underline{\leqslant} \in I(a)$ and for every neighbor b_i of a, there is at least one $\underline{\varrho} \in I(b_i)$ s.t. $\underline{\leqslant}$ is matched by $\underline{\varrho}$.

■

Definition 7.7 [complete local consistency]

A labelling I∈LP-I is called completetly locally consistent - clc in short - if and only if I is locally consistent at every junction a∈J.

■

From previous chapters, we know that Waltz-fitering procedures determine a maximal sublabelling contained in the initial labelling that enjoys this clc-property.

Notation [sub-labelling]

For two labellings I, I' ∈ LP-I, we say that I is a sub-labelling of I' if and only if

$\forall a \in J$: $I(a) \subseteq I'(a)$

We write $I \subseteq I'$.

■

Definition 7.8 [maximal clc sublabelling]

Let LP be a labelling problem, $I_0 \in$ LP-I. A labelling $I^* \in$ LP-I is said to be a maximally clc w.r.t. I_0 if and only if

(1) $I^* \subseteq I_0$
(2) I^* is clc
(3) $\forall\, I' \subseteq I_0$: I' is clc $\Rightarrow I' \subseteq I^*$

■

Azriel Rosenfeld and his colleagues [Rosenfeld|Hummel|Zucker-76] have shown that this maximal clc sublabelling is uniquely determined.

Theorem 7.1
[existence and uniqueness of maximal clc sublabellings]

Let LP be a labelling problem, $I_0 \in$LP-I. There is a unique labelling I^* that is maximally clc w.r.t. I_0.

Proof (Rosenfeld)

First, we show that the subset of all clc sublabellings of I_0 is closed under set union. To see this, let $I, I' \subseteq I_0$ be clc. Clearly, $(I \cup I') \subseteq I_0$. Furthermore, let for some $a \in J$, $\underline{\lessdot} \in (I(a) \cup I'(a))$. For every neighbor b of a, we can find a label $\underline{\varrho}$ in I(b) or in I'(b) that matches $\underline{\lessdot}$, by view of the consistency of both I and I'. Since $\underline{\varrho} \in (I(b) \cup I'(b))$, this is the desired result.

Second, note that the null labelling I-NULL is clc by definition, hence there is at least one clc sub-labelling of I_0. As LP-I is finite, we can show by induction that the union of all clc sublabellings of I_0 is clc, too. This union is the well-defined maximal clc sublabelling I^* of I_0.

■

Corollary 7.2 [maximal clc sublabelling of I-INIT]

Every labelling problem LP has a uniquely defined solution in terms of a labelling I^* that is maximally clc w.r.t. to the initial labelling I-INIT of LP.

■

In the sequel, we will speak of the maximal clc labelling of a labelling problem to denote this labelling I^* being maximally clc w.r.t. I-INIT. What we would like a labelling procedure to do is to produce a (possibly ambiguous) labelling such that we can start at every junction, choose an arbitrary label and find an unambiguously completed admissible labelling.

Definition 7.9
[global consistency and complete global consistency]

A labelling I∈LP-I is globally consistent at a junction a∈J if and only if for every $\underline{\leqslant} \in I(a)$, there is an admissible sublabelling I'⊆I of I such that $I'(a) = \underline{\leqslant}$. Here too, we extend the definition to complete global consistency as pointwise global consistency.

■

The following two lemmata summarize some of our previous results.

Lemma 7.3 [global consistency implies local consistency]

Let I∈LP-I be a labelling. If I is completely globally consistent then it is completely locally consistent.

<u>Proof</u> (indirect)

Let I be not clc. Then there must be a junction a s.t. I is not locally consistent at a. By definition 7.6, this means that there is some $\underline{\leqslant} \in I(a)$ and there is some neighbor b∈adj(a) s.t. no label $\underline{\varrho}$ of b matches $\underline{\leqslant}$. Thus, no admissible sublabelling I'⊆I with $I'(a) = \underline{\leqslant}$ can be found. I.e., I is not globally consistent at a and hence, I is not completely globally consistent.

■

Lemma 7.4
[local consistency is insufficient for global consistency]

There are labelling problems LP with labellings I∈LP-I such that I is clc but not completely globally consistent.

<u>Proof</u> (by counter-example)

The labelling shown in figure 7.1 is clc. However, no admissible labelling can be found for any of the labels of, say, the junction a.

■

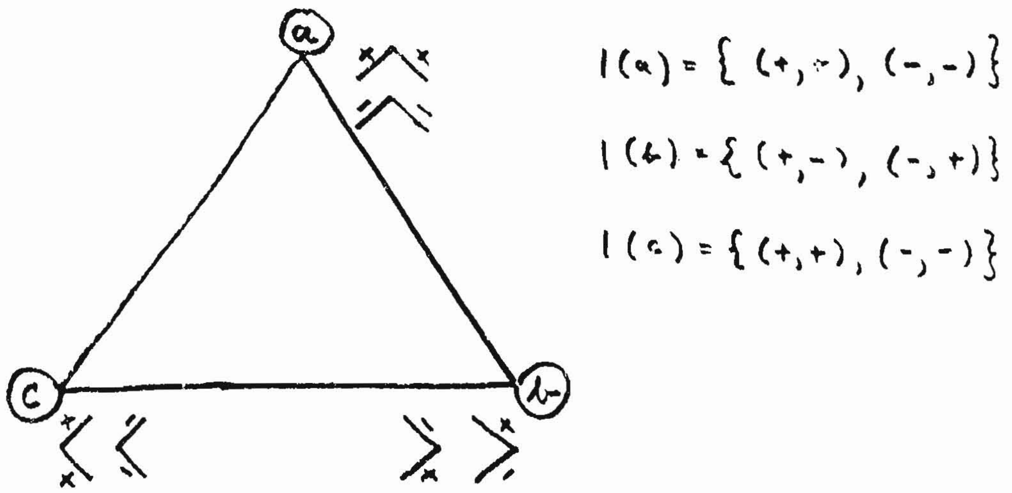


Figure 7.1: A locally consistent labelling without any admissible sub-labelling

The two lemmata above, together with the fact that, most usually, the initial labelling I-INIT is significantly larger than its maximal clc sublabelling I*, shed some light onto the relevance of filtering procedures. In the next chapter, then, we will show that Waltz-like filtering procedures are correct in that they, in fact, determine the maximal clc labelling of any given labelling problem.

7.3 Label Compatibilities Seen as Constraints

Labelling problems fit into the general constraint problem sketched in chapter 3 in two different ways. On the one hand, we can consider the segments as variables, taking values from the total of possible segment labels:

VARS = S
DOMS = {Σ}

The constraint relations then are given by the dictionary pages, as e.g.

ARROW ⊆ Σ X Σ X Σ
ARROW = { (+,−,+), (+,+,+), (o>,+,>o) }

Constraint instances relate to junctions, like e.g.

<a;ARROW;(s,t,u)>

However, this representation is only correct if the compatibility between segment labels reduces to simple identity. Otherwise, we must e.g. represent every segment by two variables that are constrained by a binary compatibility constraint being an instance of ≡ (figure 7.2). Besides that, every variable is constrained by exactly two constraints of various degrees.

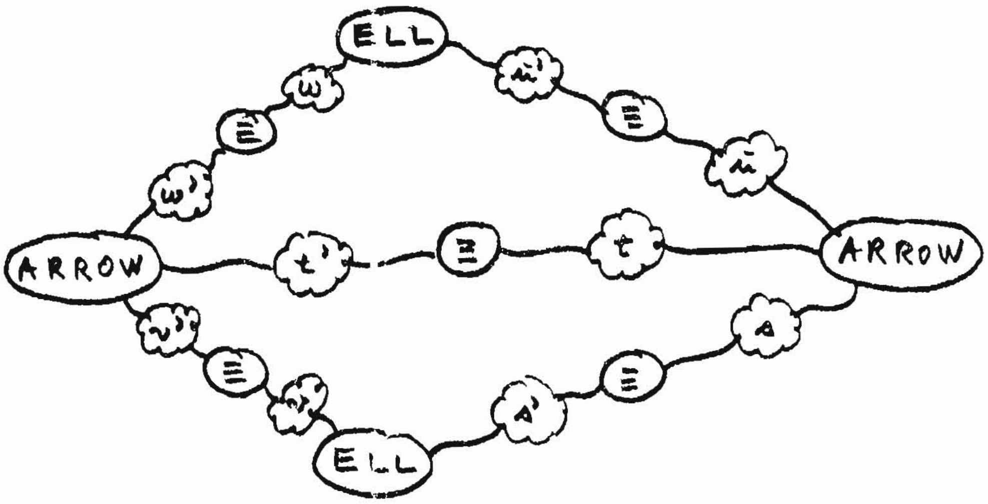


Figure 7.2: The labelling problem seen as an n-ary constraint problem. If ≡ is not identity then every segment must be represented by two variables whose values are related through a ≡-constraint.

An alternate way is to consider the junctions as variables taking values from appropriate dictionary pages. The constraints then correspond to the binary p≡q-compatibility relations. The labels of the junctions a and b in figure 7.3 e.g. are constrained by an instance of a relation

ARROW-1≡2-ELL $\subseteq$ L(ARROW) $\times$ L(ELL)
where
ARROW-1≡2-ELL = {((o>,+,>o),(o>,>o)),((o>,+,>o),(+,>o)),
((-,+,-),(o>,-)),((+,-,+),(>o,+))}

<CONNECT-a,b; ARROW-1≡2-ELL; (a,b)>

Alternatively, one could say that the labels are constrained by a relation $1{\equiv}2 \subseteq \Sigma^3 \times \Sigma^2$, the restriction to L(ARROW)$\times$L(ELL) resulting from an initial assignment. Or, this restriction could also be viewed as additional unary constraints instead of initial assignments.

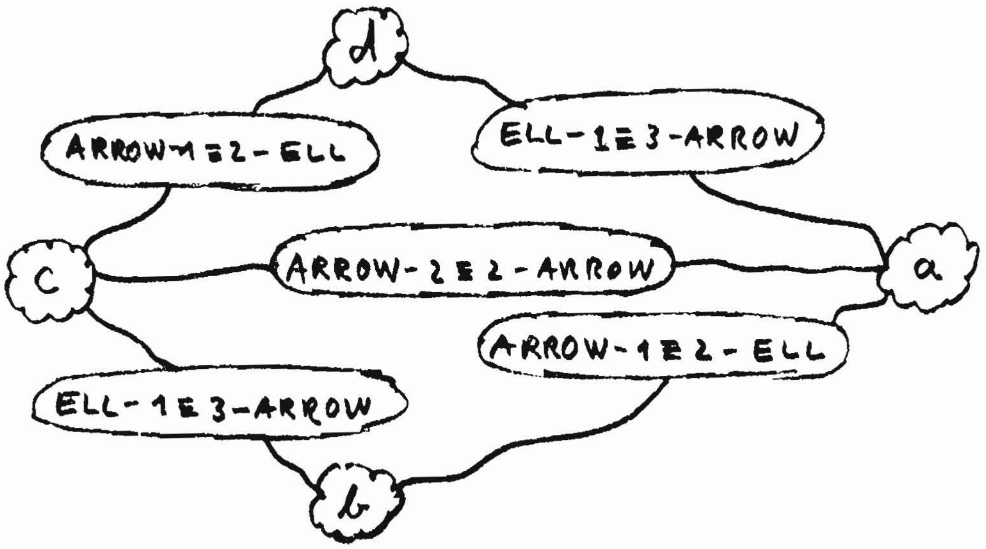


Figure 7.3: A labelling problem seen as a binary constraint problem

These binary constraints have a very simple, uniform structure that allows for an efficient representation.

Definition 7.10 [pseudo-transitive relations]

Let A, B be two disjoint sets. A relation $R \subseteq A \times B$ is pseudo-transitive if and only if for every $a_1, a_2 \in A$ and for every $b_1, b_2 \in B$, the following holds

if $\{(a_1,b_1), (a_1,b_2), (a_2,b_2)\} \subseteq R$ then $(a_2,b_1) \in R$.

■

The relation graph of a pseudo-transistive relation subdivides into complete bipartite componenents (figure 7.4).

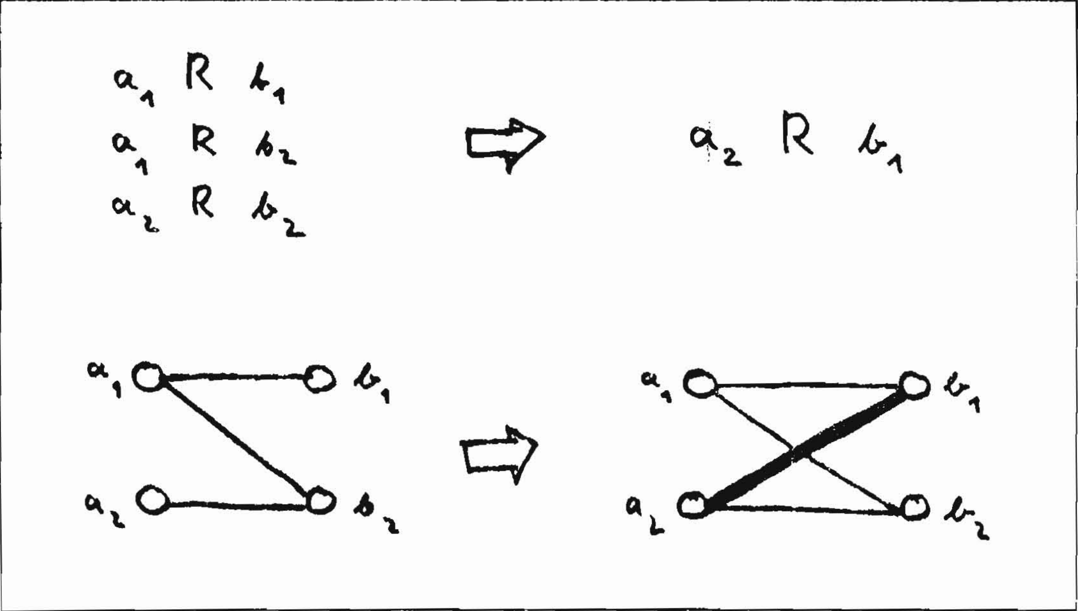


Figure 7.4: The relation graph of a pseudo-transitive relation subdivides into completely bipartite connectivity components

Lemma 7.5 [junction label compatibility is pseudo-transitive]

The relations p≡q are pseudo-transitive.

Proof

Pseudo-transitivity obviously follows from the fact that p≡q is defined in terms of ≡ which is a symmetric one-to-one relation. For, let $\underline{\sigma}$ p≡q $\underline{\rho}$, $\underline{\mu}$ p≡q $\underline{\rho}$, and $\underline{\mu}$ p≡q $\underline{\nu}$. By definition of p≡q, we then have $\sigma_p \equiv \rho_q$, $\mu_p \equiv \rho_q$, and $\mu_p \equiv \nu_q$. Since ≡ is one-to-one, it must be the case that $\sigma_p = \mu_p$, and hence $\sigma_p \equiv \nu_q$. This means that $\underline{\sigma}$ p≡q $\underline{\nu}$, what is the desired result.

∎

The structure of a completely bipartite graph can be fully represented by a related graph that is significantly less complex. All the nodes of A resp. B that belong to one common component are condensed into one single node. All the links of one such connectivity component then can be represented by one link between the resulting two super-nodes. Isolated nodes are all gathered in one super-node that is connected to a node representing the empty set (figure 7.5).

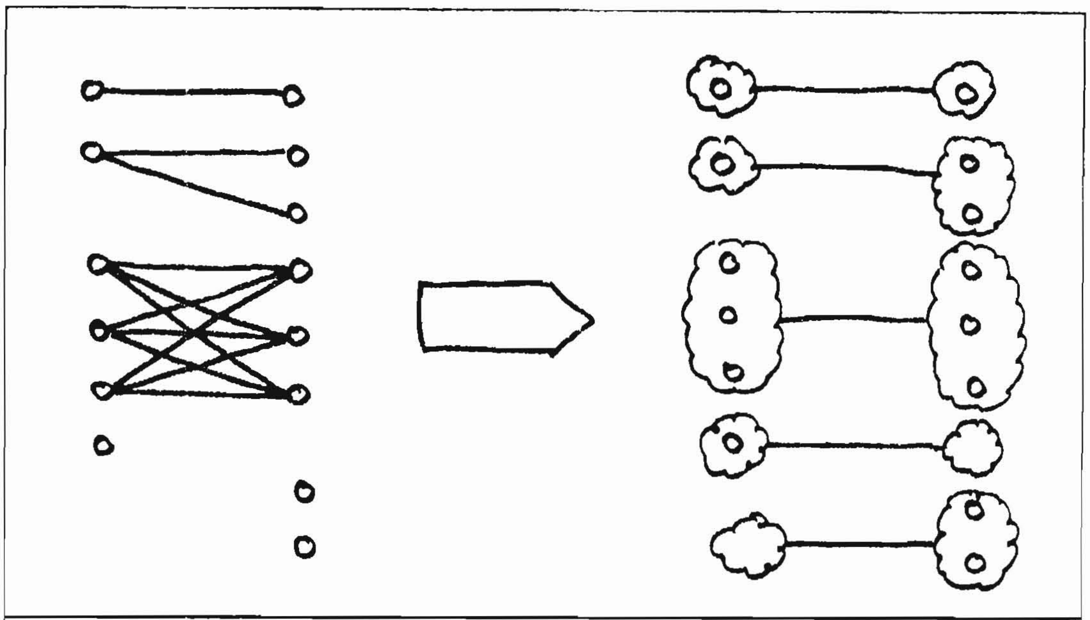


Figure 7.5: Completely bipartite graphs allow for a simplified representation

Figure 7.6 shows the relation graph and its simplified representation of the 1≡2-relation between ARROW and ELL labels.

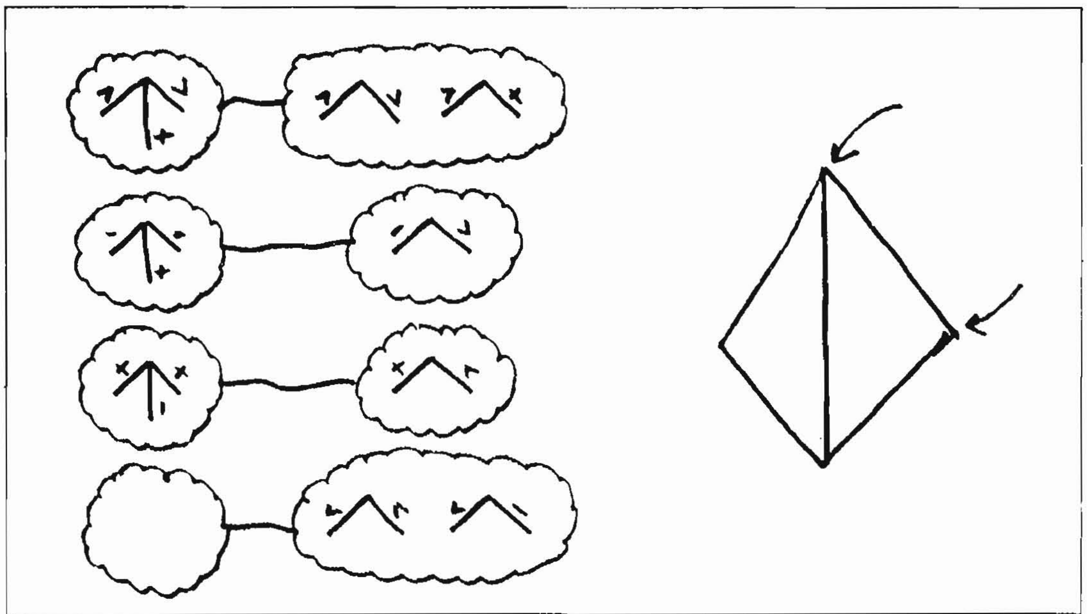


Figure 7.6: Relation graph of ARROW-1≡2-ELL

8 Filtering Algorithms Revisited

In this chapter, I am going to discuss several filtering algorithms using the vocabulary developed in the chapters six and seven. A basic filtering operator is introduced. After a review of the original Waltz-algorithm [Waltz-72], I briefly survey the synchronized parallel version developed by Azriel Rosenfeld [Rosenfeld|Hummel|Zucker-76]. In the next chapter, then, I will give a somewhat more rigorous specification of SCENELABs labelling procedure and prove that it is correct.

8.1 A Basic Operator

A basic operation that is, with some modifications, performed by every filtering procedure is to adjust the current labelling of a junction w.r.t. to the labelling of one of its neighbors.

Definition 8.1 [Γ-operators]

Let LP be a labelling problem. For two adjacent junctions $a,b \in J$, let the operator $\Gamma=\Gamma(a,b)$ be defined as follows:

$$\begin{aligned} &\Gamma: \text{LP-I} \rightarrow \text{LP-I} \\ &\Gamma I(a) = I(a) \setminus \{ \underline{\leqslant} \mid \neg\exists\ \underline{\wp} \in I(b) \text{ that matches } \underline{\leqslant} \} \\ &\forall x \neq a:\ \Gamma I(x) = I(x) \end{aligned}$$

■

Such a Γ-operator is 'correct' in the sense that it does not remove labels belonging to the solution I^* of LP for its own sake.

Lemma 8.1 [preservation of I^*-subsumption]

Let LP be a labelling problem, $I \in$ LP-I. Let $\Gamma=\Gamma(a,b)$ be as in definition 8.1.

If $I^* \subseteq I$ then $I^* \subseteq \Gamma I$.

<u>Proof</u> (indirect)

Suppose that $I^* \not\subseteq \Gamma I$. Then there is a junction x and a label $\underline{\leqslant} \in I^*(x) \setminus \Gamma I(x)$. If $x \neq a$ we are done, since $\Gamma I(x) = I(x)$, and hence $I^*(x) \not\subseteq I(x)$.
Therefore, assume $x=a$. By definition of Γ,
$\Gamma I(a) = I(a) \setminus \{ \underline{\mu} \mid \underline{\mu} \text{ is not matched by any } \underline{\nu} \in I(b)\}$.
If $I^*(a) \not\subseteq I(a)$ we are done. Otherwise, let $\underline{\leqslant} \in I^*(a)$ has been removed by Γ. This means that there is no $\underline{\wp} \in I(b)$ that matches $\underline{\leqslant}$. Since $\underline{\leqslant} \in I^*(a)$, there is at least one such $\underline{\wp} \in I^*(b)$. This shows that $I^*(b) \not\subseteq I(b)$. Hence, in any case, $I^* \not\subseteq I$.

■

Corollary 8.2 [iterated preservation of I^*-subsumption]

Let LP be a labelling problem, $I \in LP\text{-}I$. Let (a_i,b_i), $i=1,2,\ldots,n$, be a sequence of pairs of adjacent junctions. Set $\Gamma_i=\Gamma(a_i,b_i)$. If $I^* \subseteq I$ then $I^* \subseteq (\Gamma_n o \Gamma_{n-1} o \ldots o \Gamma_1)I$.

Proof

Obviously, induction on n provides the desired result, using lemma 8.1.

■

Γ-operators are also 'complete' in the sense that when a labelling is not altered by the application of any possible such operator then it is consistent.

Lemma 8.3 [stability implies consistency]

Let LP be a labelling problem. Let (a_i,b_i), $i=1,2,\ldots,n$, be an arbitrary enumeration of all ordered pairs of adjacent junctions. I.e., for every segment s where $\mu(s)=[a,b]$, both (a,b) and (b,a) are enumerated. Set, as above, $\Gamma_i=\Gamma(a_i,b_i)$. If $(\Gamma_n o \Gamma_{n-1} o \ldots o \Gamma_1)I = I$ then I is clc.

Proof

Clearly, if the precondition holds then, for every $k \in \{1,2,\ldots,n\}$, $(\Gamma_k o \Gamma_{k-1} o \ldots \Gamma_1)I = I$ holds, too. For an arbitrary junction a, we show that I is locally consistent at a. Let $adj(a)=\{b_1,\ldots,b_p\}$. For every such neighbor b_j, the operator $\Gamma(a,b_j)$ occurs in the enumeration. Thus, we have $\forall b \in adj(a)$: $\Gamma(a,b)I(a) = I(a)$. By the definition of Γ, this means that, for every neighbor b of a, the set $\{ \lesssim \in I(a) \mid \lesssim$ is not matched by a $\gtrsim \in I(b) \}$ is empty. Or, the other way round, every label $\lesssim$ at a is matched by at least one label $\gtrsim$ at each of its neighbors. I.e., I is locally consistent at a and, as the same arguments apply to arbitrary junctions, I is clc.

■

Corollary 8.2 and lemma 8.3, taken together, suggest the following overall structure of a filtering algorithm: First, assign to every junction the corresponding dictionary page. Second, for an enumeration of all pairs of adjacent junctions, iteratively apply the corresponding sequence of Γ-operators, until no more deletions occur (figure 8.1).

```
let $(a_i,b_i)$, i=1,2,...n, be an enumeration of
    all pairs of adjacent junctions.

begin

    for $a \in J$ do
        $I(a) := L(fig(a))$;

    repeat

       $I' := I$;
       for i:=1..n do
           $I := \Gamma(a_i,b_i)I$;

    until $I = I'$

end.
```

Figure 8.1: Basic version of a filtering algorithm

Theorem 8.4 [correctness of BASIC-FILTER]

When working on a labelling problem LP, BASIC-FILTER terminates after a finite number of iterations and provides the maximal clc labelling I^* of LP.

Proof

Clearly, after the initial setting I=I-INIT holds. Since Γ-operations perform only deletions, if any, we have $I \subseteq$ I-INIT, for every subsequently reached stage. Suppose that the procedure terminates with $I'=I$. By view of corollary 8.2, $I^* \subseteq I$. Furthermore, lemma 8.3 says, I is clc. Thus, we have:

$I^* \subseteq I \subseteq$ I-INIT, and I is clc

By view of the maximality of I^*, we conclude that $I=I^*$.

It remains to show that BASIC-FILTER terminates. To see this, note that I-INIT is finite, and that the resulting sequence of intermediate labellings is monotonically decreasing in that $I \subseteq I'$ holds after every iteration. Henceforth, the termination condition can only be violated finitely many times.

■

8.2 The Waltz-Procedure

As I have already informally introduced the original Waltz-algorithm in chapter 3, I just indicate one possible realization in some linear pseudo-code notation, without further explanation (figure 8.2). A call adjust(a,b) realizes a Γ-operator Γ(a,b), plus the corresponding propagative calls for the neighbors of a. Notice that the adjust calls here follow a queue-based scheduling, a stack-based organization is also possible.

The Waltz-procedure differs from our prototype algorithm mainly in two aspects. Firstly, there is some definite control regime that guides the application of Γ-like operators. Secondly, it works only with partial labellings, namely of those junctions being marked 'visited'. However, to require the labelling of a junction to be locally consistent w.r.t. the labellings of some but not necessarily all of its neighbors is equivalent to local consistency (w.r.t. all of the neighbors) where the neighbors not considered carry their entire label universe I-UNIV.

```
procedure WALTZ(labelling problem: LP);

 var labelling: I;
 var list-of-junction-pairs: CHECK-LIST;

 procedure ADJUST(junction: a,b)

  var set of labels: I';

  I' := { σ∈I(a) | no ρ∈I(b) matches σ };

  if I' ≠ {} then
     I(a) := I(a) \ I';

     for x ∈ adj(a)\{b} do
        if marked(x) then
           append(x,a) to CHECK-LIST;
        endif;
     endfor;
  endif;

 endprocedure ADJUST;

 for a ∈ J do

     I(a) := L(fig(a));
     mark(a);

     for b ∈ adj(a) do
         if marked(b) then
            I(a) := I(a) \ { σ| no ρ∈I(b) matches σ }
            append(b,a) to CHECK-LIST;
         endif;
     endfor;

     while CHECK-LIST ≠ empty-list do
           ADJUST(head(CHECK-LIST));
           CHECK-LIST := tail(CHECK-LIST);
     endwhile;

 endfor;

endprocedure WALTZ;
```

Figure 8.2: A possible realization of the Waltz-algorithm

Definition 8.2 [universal completion]

Let LP be a labelling problem, $K \subseteq J$. Given a partial labelling

$I: K \to \mathcal{P}(L)$
$x \mapsto I(x) \subseteq L(fig(x))$

we call the mapping ucI defined as below the universal completion of I.

$ucI: J \to \mathcal{P}(L)$
$\forall x \in K: ucI(x) = I(x)$
$\forall x \in (J \setminus K): ucI(x) = \Sigma^n, n = deg(x)$

Notice that ucI is no labelling in the proper sense of the word.

■

Definition 8.3 [K-partial local consistency]

Let LP be a labelling problem, $K \subseteq J$. A K-partial labelling I is called K-partially locally consistent, K-plc in short, if and only if the universal completion ucI of I is locally consistent at every junction $a \in K$.

■

Let $\{a_1, a_2, ..., a_n\}$ be the enumeration of J used in the Waltz-procedure. The outer for-loop results in a sequence of $\{a_1, ..., a_p\}$-partial labellings I_p, for p=1,2,...n.

Lemma 8.5 [partial consistency of I_p]

For every such p, I_p is $\{a_1, ... a_p\}$-plc.

<u>Proof</u> (by induction on p)

Let p=1.
$ucI_1(a_1) = L(fig(a_1))$, and $\forall x \neq a_1: ucI_1(x) = I\text{-}UNIV(x)$.
Hence, ucI_1 is locally consistent at a_1.

Now suppose that I_p is $\{a_1, ..., a_p\}$-plc, for some $p >= 1$.
ucI_{p+1} is made locally consistent at a_{p+1} by means of the inner for-loop. Local consistencey then may only be violated at some neighbor b of a_{p+1}, where the source for the inconsistency lies at a_{p+1}. These inconsistencies are removed by the corresponding adjust(b,a_{p+1})-calls. Possibly resulting inconsistencies due to these adjustments are resolved by the subsequent iterations of the while-loop.

■

Corollary 8.6 [local consistency of the final labelling]

I_n is clc.

Proof

Obviously, $ucI_n = I_n$.

■

Lemma 8.7 [iterated I^*-subsumption]

For every p, $I^* \subseteq ucI_p$.

Proof

Besides the initial assignments I(a) := L(fig(a)), the procedure performs only Γ-operations on the current labellings. Lemma 8.1 says, Γ-operations preserve I^*-subsumption.

■

Corollary 8.6 and lemma 8.7 suffice to show that the Waltz-procedure provides the desired result I^*.

Theorem 8.8 [correctness of the Waltz-procedure]

$I_n = I^*$.

Proof

By view of lemma 8.7, and as a consequence of the initial assignments, we have

$$I^* \subseteq I_n \subseteq I\text{-}INIT$$

Furthermore, I_n is clc (Corollary 8.6). As I^* is maximal, we conclude that $I_n \subseteq I^*$, and hence $I_n = I^*$.

■

By now, we left one little problem unaddressed: the Waltz-procedure contains a while-loop which might prevent it from termination. We argue that the check-list may increase only finitely many times within the while-loop, as a consequence of some label(s) being eliminated, while it decreases during every iteration of the loop. Additionally, when the loop is entered, the check-list will always have some finite length.

8.3 Rosenfelds Version

Azriel Rosenfeld and his colleagues from the University of Maryland [Rosenfeld|Hummel|Zucker-76] have pointed out that the very nature of a labelling problem favors a parallel solution which, in fact, turns out to be convincincly simple.

Given an actual labelling I, their algorithm simultaneously removes from every junction all those labels for which there is some neighbor having no matching label.

Definition 8.4 [Δ-operator]

Let LP be a labelling problem. The Δ-operator is defined as

$$\Delta: \text{LP-I} \rightarrow \text{LP-I}$$
$$I \mapsto \Delta I$$
where
$$\Delta I(a) = I(a) \setminus \{ \underline{\leqslant} \mid \exists\, b \in adj(a) \text{ s.t. } \neg\exists\, \underline{\varrho} \in I(b) \text{ s.t. } \underline{\varrho} \text{ matches } \underline{\leqslant} \}.$$

∎

Thus, for every pair (a,b) of adjacent junctions, Δ removes the labels at b being not matched by a label at a and vice versa. I.e., an evaluation of Δ corresponds to a simultaneous evaluation of the Γ-operators $\Gamma(a,b)$ for every (ordered) pair of adjacent junctions.

With arguments similar to those used in the lemmata 8.1 and 8.3, one can show that an iterated application of Δ to the initial labelling I-INIT until the resulting labelling becomes stable provides the solution I^* of a labelling problem.

Theorem 8.9 [correctness of Rosenfelds version]

Let LP be a labelling problem. Define a sequence I_n, $n \in N$ of labellings as follows:

$$I_0 = \text{I-INIT}$$
$$I_{i+1} = \Delta I_i$$

Then, for some finite n, $I_n = I_{n-1}$, and $I_n = I^*$.

<u>Proof</u> (Rosenfeld)

First, observe that for every n, $I_{n+1} \subseteq I_n$. As I-INIT is finite, the sequence monotonically converges to some finite subset, say I', of I-INIT.

Second, the definition of Δ assures that for every $n \geq 1$, and for every $a \in J$, the labelling I_n is locally consistent at a w.r.t. to the labelling I_{n-1} of all neighbors b of a. Once I_n equals I_{n-1}, I_n is locally consistent at every a, and hence it is clc. For a detailed proof, see [Rosenfeld|Hummel|Zucker-76].

∎

9 SCENELABs Asynchrounous Labelling Algorithm

Our solution, as discussed in Part B of this paper, is in some sense a non-compromising continuation of the development from the original sequential version through Rosendfelds synchronized parallel version. The motivation thereof is twofold.

- **Expected Speed-up**: Firstly, there is an expected speed-up. On one hand, in some regions of the drawing, restrictive constraints could be propagated relatively far but are retained until the next global, common step. On the other hand, there may be some regions being processed by any global step although no additional constraints can be evaluated. Thus, why not omit the synchronization overhead that is both costly and restrictive?

- **Ease of Representation**: Secondly, it is an issue of representational adequacy. Filtering algorithms, whether sequential or parallel, with or without a global control regime, can only make a labelling locally consistent. So, I find it more convenient to look at labelling problems through the paradigm of societies of agents, where every local processing unit is responsible for keeping its own, local labelling consistent w.r.t. to all of its neighbors.

Notice that the same arguments, expected speed-up and 'natural' view of the problem, also have motivated the development of a synchronized parallel version in comparison to a sequential version. The speed-up argument, however, remains somewhat hypothetical as long as there are no real, physically distributed realizations running on asynchronously coupled processors, or at least some appropriate performance measures that take the communication overhead into account.

9.1 The Operations of Local Analysts

Recall that the internal state of a CSSA-agent is given by variable/value pairs, and that state transactions are performed by CSSA-operations being executed after the receipt of an appropriate message. Both the total of possible states and (possibly parametrized) transactions are specified within the corresponding CSSA-scripts from which the concrete agents are instantiated by generator-expressions (see section 4.2).

The keywords used in the descriptions below form a small subset of a documentation language I have developed and used for an implementation of SCENELAB [Reinfrank-83]. I the sequel, we will refer to a LOCAL ANALYST that works on the labelling of a junction a by analyst(a). Every such agent has internal representations of the segment labels Σ, the matching function match, and the ordered set of adjacent junctions adj(a). Furthermore, it manipulates the representation of a set $I(a) \subseteq \Sigma^n$, n=deg(a), of junction labels, where the value of I(a) represents the local labelling of a as has been evaluated thus far by analyst(a). For the sake of simplicity, I do not distinguish between objects and their internal representations here.

Definition 9.1
[junction labels inducing a specific segment label]

For Σ, adj(a), and a set $I(a) \subseteq \Sigma^n$, n=deg(a), let

$$M(\sigma, b, I(a)) = \{ \underline{\sigma} \in I(a) \mid \begin{array}{l} \underline{\sigma} = (\sigma_1, \ldots, \sigma_{k-1}, \sigma, \sigma_{k+1}, \ldots, \sigma_n) \\ adj(a) = (b_1, \ldots, b_{k-1}, b, b_{k+1}, \ldots, b_n) \end{array} \}$$

I.e., $M(\sigma, b, I(a))$ consists of those labels in I(a) that induce the segment label σ for the segment joining a and b.

■

Basically, an agent analyst(a) can perform two different actions to manipulate labellings. Firstly, it can remove such a set $M(\sigma, b, I(a))$ from its current labelling. Secondly, it can tell a neighbor b about the fact that such a set has been eliminated and , hence, b should delete $M(match(\sigma), a, I(b))$.

These actions are performed by means of the CSSA-operations INIT, START, and PROPAGATE. The effects of these operations, when executed by analyst(a), are described below. The condition

```
    is-sent(OPERATION(PARAMETERS) to ADDRESSEE)
```

means that the corresponding message is issued during the operation.

```
operation INIT(Σ, adj(a), I-INIT(a), match, ...) is

 comment: initializes the variables of analyst(a). The
          additional parameters are irrelevant here.

 post-INIT: I(a) = I-INIT(a)
            Σ, adj(a),match, and I-INIT(a) are correct w.r.t.
            to the given labelling problem LP currently in work.

endoperation;


operation START is

 comment: propagates the initial constraints induced by the
          assignment I(a) := I-INIT(a), as done by INIT.

 pre-START: I(a) = I-INIT(a);

 post-START: ∀ σ∈Σ ∀ b∈adj(a):

                    is-sent(PROPAGATE(match(σ),a) to analyst(b))
                               if and only if
                           M(σ,b,I-INIT(a)) = {}
```

```
endoperation;

operation PROPAGATE($\sigma$,b) is

 comment: removes all labels from I(a) that induce $\sigma$ for the
          segment joining a with b, and immediately propagates
          resulting additional constraints, if any.

 let: i(I(a)) and o(I(a)) denote the value of I(a) when the
      operation is started resp. finished.

 post-PROPAGATE: $o(I(a)) = i(I(a)) \setminus M(\sigma,b,i(I(a))$

                 $\forall\rho\in\Sigma\ \forall c\in adj(a)$

                     is-sent(PROPAGATE(match($\rho$),a) to analyst(c))
                              if and only if
                     $M(\rho,c,i(I(a)))\neq\{\}$ and $M(\rho,c,o(I(a)))=\{\}$

endoperation;
```

Notice that the message send(PROPAGATE(match(ρ),a) to analyst(b)) is not necessary in PROPAGATE. Therefore, it is suppressed in the real implementation. Furthermore, to reduce the number of messages, one could pack all the messages sent during one START or PROPAGATE transaction to one agent b into a single message PROPAGATE($\{\sigma_1,...,\sigma_p\}$,a).

9.2 Snapshots Replace Global States

Clearly, the specifications above do not allow for a rigorous verification of SCENELAB. Notice e.g. that the post-conditions cannot be shown from the pre-conditions and the code alone. Even if we assume that all of these post-conditions, in fact, do hold, we need some additional assumptions about the global system structure and behavior. In particular, we will assume that

- The operations being executed by every agent analyst(a) satisfy the pathcondition INIT;START;(PROPAGATE)*.

- There are no PROPAGATE-messages besides those sent within START and PROPAGATE transactions. I.e. we do not consider PROPAGATE messages sent by the SUPERVISOR as a consequence of an EXCLUDE-command. In fact, such exclusions initiated by the user modify the underlying labelling problem and its result I^*.

The post-INIT condition says that the network is really isomorphic to the line drawing, and that the labellings are correctly initialized. Furthermore, recall our postulate that every message, especially the PROPAGATE-messages, once sent are eventually received and processed.

Based on these assumptions, we will show the following claim.

Claim 9.1 [correctness of SCENELABs labelling algorithm]

> When working on a labelling problem LP, the constraint network of SCENELAB eventually comes to a halt, and the local labellings I(a) then represent the solution I* of LP.

■

The versions of filtering procedures discussed in the preceding chapter could be adequately represented in terms of intermediate labellings and operators that modify these intermediate labellings. In the asynchronous case, there are no such well-defined intermediate labellings. Basically, the PROPAGATE-messages sent by analyst(a) to analyst(b) during one START or PROPAGATE transaction realize a Γ-operator Γ(b,a). However, the adjustments are performed after some finite delay. Asking around every analyst for its current local labelling may provide a chaotic result, since an unknown number of messages may be still on the way.

The basic problem hereby is the lack of an observable global state, as we have already discussed in chapter 4. Moreover, there is also no global system time observable by any of the agents. Such a global time could be used to assess past global states. To get things right: we can postulate a global time w.r.t. to, say, the users watch or so. However, as communication always involves some message transmission time, this clock cannot be simultaneously assessed by the agents. Hence, local clocks only can be synchronized with respect to each other up to some non-negligible fault-tolerance. Synchronization of real time clocks in distributed environments is discussed e.g. by Leslie Lamport [Lamport-78].

The inobservability of simultaneous global states leads us to the introduction of snapshots as a substitute for such states. Intuitively, a snapshot is a possible result of asking around every agent for its actual local state. Recall that there is a one-to-one correspondence between messages and transactions in that transactions are induced by the the receipt of messages and every message is eventually received. Given a network of LOCAL ANALYSTS working on a labelling problem, we restrict our considerations to transcations belonging to the kernel filtering algorithm, and to their effects upon those parts of the local states that represent the local labellings. Furthermore, we will condense the sequence of an INIT-transaction and a START-transaction into one single INIT;START-transaction. This does not make any difficulties, since the INIT-transactions only leads to a modification of the state but does not send any messages, while a START-transcation only sends messages, but does not modify the state. From a conceptual point of view, such a combined INIT;START transaction is very much like a PROPAGATE transaction: it "removes" labels from a hypothetical original labelling I-UNIV(a), and immediately propagates the resulting constraints.

Definition 9.2 [F-transactions and F-states]

The F-TRANSACTIONS performed by SCENELABs constraint network consist of

- for every analyst(a), an initial INIT;START-transaction
- exactly those PROPAGATE-transactions performed by an analyst(a) as a consequence of a message sent by another LOCAL ANALYST, say, analyst(b).

The F-STATES consist of

- for every analyst(a), an original state S_0 it is within after being created by the SUPERVISOR
- every state of an analyst(a) that results from a transaction $T \in$ F-TRANSACTIONS.

∎

Notation [state transitions]

Let T be a transaction performed by an agent analyst(a), leading to a state transition from S to S'. We then write

a:: S -T-> S'

Usually, if it is clear or unimportant which agent performs T, we omit a.

∎

Every LOCAL ANALYST, then, goes through a sequence of stable states and transactions of the form

a:: S_0 -INIT;START-> S_1 -PROPAGATE-> S_2 -PROPAGATE-> S_3 ...

Such a sequence induces a sequence of local labellings at a:

$I_0(a)$ = I-UNIV(a)
$I_1(a)$ = I-INIT(a)
$I_{n+1}(a)$ = $I_n(a) \setminus M(\triangleleft, b, I_n(a))$,

where T_{n+1} = PROPAGATE($\triangleleft$,b)

Definition 9.3 [snapshot]

Let $a_1, a_2, \ldots, a_n$ be an enumeration of J. A snapshot $\{I_{k1}(a_1), I_{k2}(a_2), \ldots, I_{kn}(a_n)\}$ is composed of local labellings $I_{kj}(a_j)$ being reached after the k_j's transaction of analyst(a_j).

∎

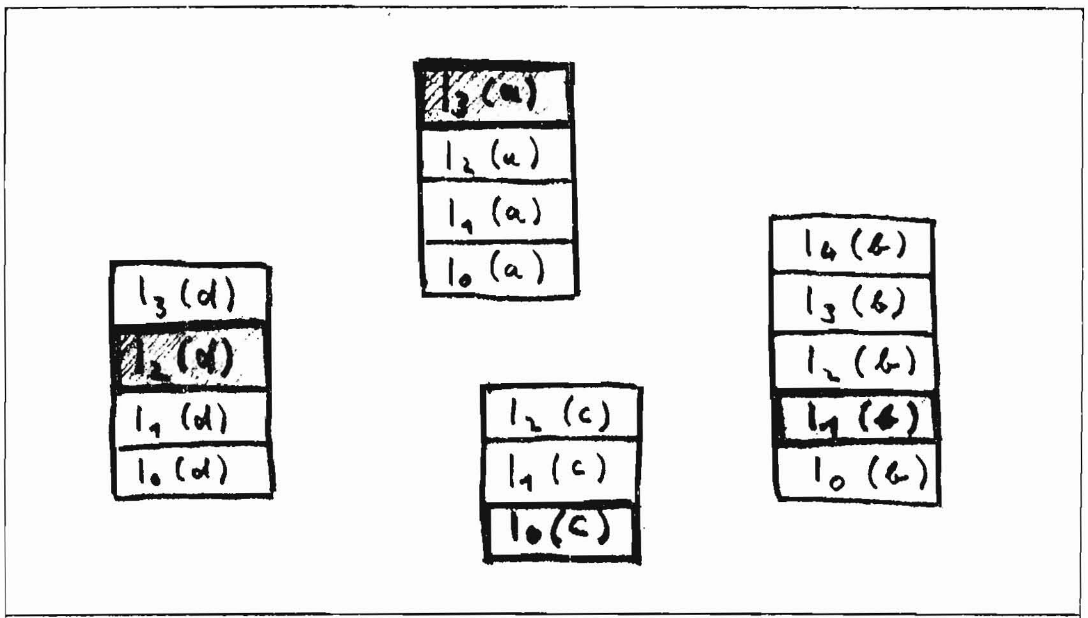


Figure 9.1: A snapshot $\{l_3(a), l_1(b), l_0(c), l_2(d)\}$

We now can define termination conditions, either event-oriented or state-oriented, in terms of properties of snapshots.

Definition 9.4 [satisfactory and final snapshots]

A snapshot is satisfactory if the corresponding labelling equals I^*.

A snapshot is final if there is no junction a where the k-th local labelling $I_k(a)$ is in the snapshot such that analyst(a) performs another transaction $T_p \in$ F-TRANSACTIONS, $p>k$.

■

Note that the definition of a final snapshot relativizes termination to F-TRANSACTIONS.

Independently from any global system clock, we can define a partial ordering upon transactions that reflects the cause/effect relationships between them.

Definition 9.5 [is-caused-by partial order of F-transactions]

Let T,T' $\in$ F-TRANSACTIONS, not necessarily being performed by two different agents. We say that T is caused by T' if and only if the message leading to T has been sent within T'. We write

T icb T'

■

Obviously, icb is a irreflexive partial ordering of F-TRANSACTIONS.

Lemma 9.1 [icb-decreasing chains]

For every T ∈ F-TRANSACTIONS, there is a unique icb-decreasing chain

$$T=T_n \text{ icb } T_{n-1} \text{ icb } \dots \text{ icb } T_1$$

downwards from T, and this chain is bound by a INIT;START-transaction T_1.

Proof

Obviously, every PROPAGATE-transaction is caused exactly by one other transaction, that is either a PROPAGATE-transaction,too, or an INIT;START-transaction. INIT;START-transactions are caused by transactions of the SUPERVISOR.

■

If we extend F-TRANSACTIONS by one addtional element that stands for the SUPERVISOR-transaction sending the INIT;START-messages, we get the following tree-structure of the partially ordered set (F-TRANSACTIONS, icb). Notice that the INIT;START-transctions are just the icb-minimal transactions (figure 9.2).

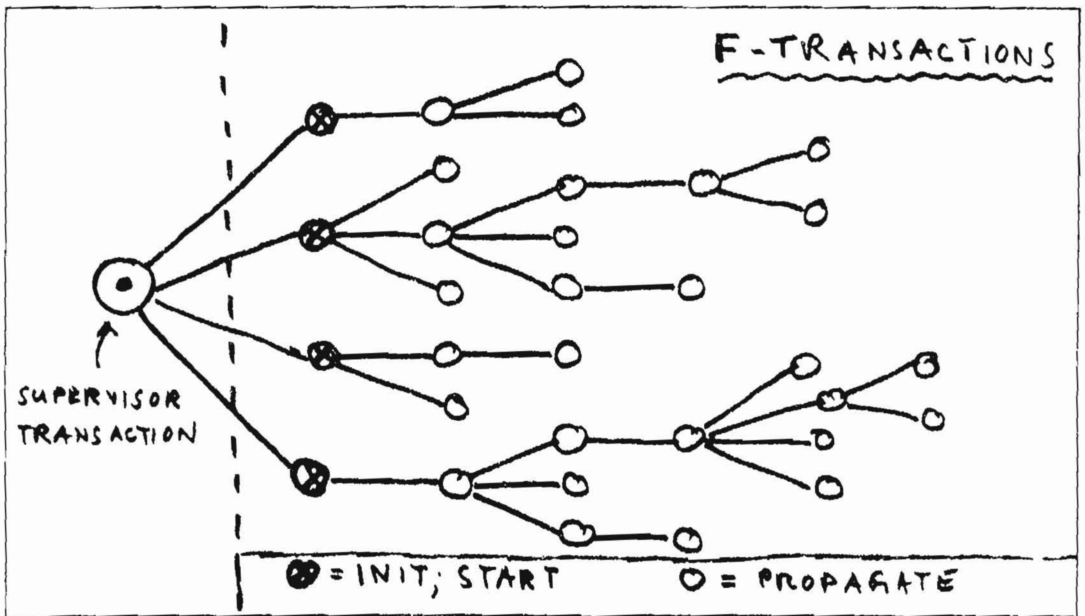


Figure 9.2: Structure of the icb partial order of F-TRANSACTIONS

We define yet another partial ordering for transactions, based on local features.

Definition 9.6 [is-locally-after partial order]

Let T,T' ∈ F-TRANSACTIONS. We say that T is locally after T', in signs

T ila T'

if and only if there is a junction a such that

$a:: S_k \text{ -T'-> } S_{k+1} \text{ -T-> } S_{k+2}$

■

Notice that both definitions are consistent with the flow of time, w.r.t. to a non-observable global time, in that T icb T' and T ila T' imply that T begins strictly after T'. This is guaranteed by the fact that message transmission is time-comsuming (for T icb T'), resp. by the fact that transactions are time-consuming and may not overlap (for T ila T'). To give a full proof of this intuitively correct notion, however, one would have to establish a correspondence between transactions and intervals of time on some time axis, or to associate every transaction with a pair [beg(T), end(T)] of points of time. Here again, we emphasize that we can postulate but not practically determine such a correspondence.

Definition 9.7 [partial ordering of F-transactions]

Let icb and ila be defined as above. We say that a transaction T is greater than T', in signs T >> T', if and only if there is a sequence

$T_1=T \ r_1 \ T_2 \ r_2 \ T_3 \ r_3 \ \ldots \ r_{n-1} \ T_n=T'$,

where r_i ∈ {icb, ila}

I.e., >> is the transitive closure of (icb ∪ ica)

■

Lemma 9.2 [<<-minimal transactions]

A transaction T ∈ F-TRANSACTIONS is <<-minimal in F-TRANSACTIONS if and only if T is an INIT;START-transaction.

Proof

Let T be an INIT;START-transaction. Then T has neither an icb-predecessor nor an ila-predecessor in F-TRANSACTIONS.

Conversely, if T is a <<-minimal transaction, it cannot be a PROPAGATE-transaction because every PROPAGATE-transaction has a unique icb predecessor in F-TRANSACTIONS and, hence, at least one <<-predecessor. Thus, if T is <<-minimal it must be an

INIT;START-transaction.

■

Leslie Lamport [Lamport-78] discussed orderings of events in distributed systems that are closely related to <<. A more detailed representational formalism for the temporal behavior of distributed systems has been developed by James F. Allen [Allen-83]. He distinguishes between 7 possible temporal relationships and their inverses between two intervals of time, such as meets, overlaps etc... Hans Voss [Voss-85] makes extensive use of an extension of Allen's technique to provide qualitatitve descriptions of the temporal and causal aspects of techno-physical systems, being represented by CSSA-like models. These issues, however, lie considerably beyond the scope of the present paper.

9.3 Towards a Correctness Proof for SCENELAB

The next lemma is central to our proof for SCENELAB's correctness.

Lemma 9.3 [cause-effect of PROPAGATE-messages]

Let $a \in J$, and let $I_n(a)$ be the n-th local labelling, evaluated by analyst(a).

$\forall \lessdot \in \Sigma \ \forall b \in adj(a)$:

$M(\lessdot,b,I_n(a)) = \{\}$
if and only if
$\exists\ k \leq n$: analyst(a) has sent a message PROPAGATE(match($\lessdot$),a) to analyst(b) during it's k-th transaction T_k.

Proof

Since transactions only remove but never add labels to I(a), and $I_0(a)$ = I-UNIV(a), it must be the case that

$M(\lessdot,b,I_n(a)) = \{\}$
if and only if
$\exists\ k \leq n$ s.t. $M(\lessdot,b,I_{k-1}(a)) \neq \{\}$ and
$M(\lessdot,b,I_k(a)) = \{\}$

Note that for every $p \geq k$, $M(\lessdot,b,I_p(a)) = \{\}$.

The specifications of PROPAGATE and INIT;START, together with the restrictions to F-TRANSACTIONS, say that exactly one message PROPAGATE(match($\lessdot$),a) has been sent during the transaction T_k to analyst(b) by analyst(a).

Conversely, let analyst(a) have sent this message during T_k. This is only done when the set $M(\sigma,b,I_k(a))$ has become empty during the transaction. Clearly, it remains empty afterwards. Hence, for any $n \geq k$, $M(\sigma,b,I_n(a))$ is empty.

■

Corollary 9.4 [SCENELABs kernel algorithm terminates]

When working on a labelling problem LP, the constraint network of SCENELAB sends and processes only a finite number of messages, and therefore reaches a final snapshot.

Proof

It obviously follows from lemma 9.3, by view of the finiteness of both J and L, that the number of PROPAGATE-transactions is finite. Moreover, there are also only a finite number of INIT;START-messages to be processed (exactly one per junction).

■

Lemma 9.5 [final snapshots and local consistency]

Every final snapshot is clc.

Proof (indirect)

Suppose that the labelling induced by a final snapshot is not clc. Then there are two adjacent junctions a,b such that for their current labellings $I_p(a)$, $I_q(b)$ in that particular snapshot the following holds

$M(\sigma,b,I_p(a)) = \{\}$
and
$M(match(\sigma),a,I_q(b)) \neq \{\}$

Lemma 9.3 now says that analyst(a) must have sent a message PROPAGATE(match(σ),a) to analyst(b). However, analyst(b) cannot yet have received this message, by view of post-PROPAGATE. We conclude that the snapshot is not final.

■

Lemma 9.6 [no spontaneous removals of I^*-labels]

Let a transaction T, where a:: S_m -T-> S_{m+1} remove a label $\underline{\sigma}$ $\in I^*$. Then there must be a neighbor b $\in$ adj(a) such that a transaction T'<<T, where b:: R_n -T'-> R_{n+1}, has removed another label $\underline{\wp} \in I^*$.

Proof

Clearly, T cannot be an INIT;START-transaction. Thus, let T be a transaction PROPAGATE($\leqslant$,b) of analyst(a). Lemma 9.3 says that the set M(match($\leqslant$),a,I(b)) has become empty exactly during the transaction, say T'', of analyst(b), in which this message has been sent. By definition of <<, T'' << T. Furthermore, there is at least one $\underline{\varrho}$ $\in$ I^*(b) matching $\underline{\leqslant}$. Hence, M(match($\leqslant$),a,I_0(b)) $\neq$ {}. This $\underline{\varrho}$ has been removed by analyst(b) either during T'', or by another, previous transaction T' << T''.

∎

Corollary 9.7 [preservation of I^*-subsumption]

No transaction T $\in$ F-TRANSACTIONS removes a label belonging to I^*.

<u>Proof</u>

The claim is clear for INIT;START transactions. If T is a PROPAGATE transaction, iterated application of lemma 9.6 requires the existence of a <<-decreasing chain

$T=T_n$ >> T_{n-1} >> .... >> T_1

where every T_i removes a label belonging to I^*. Every such chain has an INIT;START transaction as lower bound (lemma 9.2), and this is an obvious contradiction.

∎

Corollary 9.8 [I^*-subsumption]

Every snapshot subsumes I^*.

<u>Proof</u>

This is an obvious consequence of corollary 9.7.

∎

Lemma 9.9 [I-INIT-containment]

Every snapshot is subsumed by I-INIT.

<u>Proof</u>

Obvious, from post-INIT, post-START, and post-PROPAGATE. I.e., after an initial assignment I(a) := I-INIT(a), labels are only eliminated but never added.

∎

We are now ready to show that SCENELAB, in fact, works correctly.

Theorem 9.10 [SCENELAB Works Correctly]

Claim 9.1 holds.

Proof

Corollary **9.4 means that a final snapshot is reached.** Let it represent a labelling I. I is clc, as shown in lemma 9.5. Furthermore, I subsumes I^* (corollary 9.8), and itself is contained in I-INIT (lemma 9.9). Thus, the whole state of affairs is like that:

$I^* \subseteq I \subseteq$ I-INIT, and I is clc.

This shows that $I = I^*$.

■

10 Conclusive Remarks

SCENELAB evolved from the idea of realizing the Waltz-algorithm in CSSA, and it is felt that this goal has been succesfully achieved. SCENELAB can be used to specify and solve arbitrary labelling problems LP. In this last section, I will briefly address two further questions. Firstly, do the principal techniques applied in SCENELAB generalize to larger problem classes and, secondly, is CSSA an adequate tool to realize such systems?

The first question must be answered both no and yes. SCENELAB, as it has been implemented, makes extensive use of the simple structure of the constraint problems defined by labelling problems. Therefore, it does not support more complex constraint problems. However, we argue that the key idea of realizing constraints by agents, and constraint propagation by message passing mechanisms generalizes to arbitrary problems that fit into the scheme of a constraint-directed representation. I have discussed some of these issues elsewhere [Reinfrank-85].

Currently, a really physically distributed realization of CSSA is being developed, running on a network of about a dozen 32-bit CPU Charles River machines. CSSA-agents can make full use of the complete processing time of such a machine, when scheduled by an underlying distributed operation system. Such powerful units seem adequate to implement a high-level cooperation between e.g. the SUPERVISOR and the DB-SERVER, while they are 'overskilled' for such simple and largely uniform constraint processes like the LOCAL ANALYSTS. If the constraint network contains considerably more nodes than the computer network, the overhead introduced by the operating system may become too costly.

However, we should deliberately distinguish between the two questions raised here, can a certain class of problems suitably be attacked by a specific problem solving technique, and is a particular language or computing system an adequate tool to realize corresponding solutions.

Acknowledgements

Several people have contributed to the work presented in this paper. First of all, I would like to thank my friend Hans Voss. Without him, my thesis would never have been started nor completed. The insights on the distributed termination problem are largely due to several discussions with Christian Beilken and Friedemann Mattern. I am thankful to Peter Raulefs for encouraging me to work on constraints, and for commenting on an earlier draft of this paper. Finally, I am indepted to my thesis supervisor, Prof. Joerg Siekmann, who did not hesitate to help me go through a rather difficult situation that arose shortly before this thesis has been completed.

Appendix-A Bibliography

[AI17-1981] Bobrow, Daniel G. (ed.): Special Issue on Computer Vision. ARTIFICIAL INTELLIGENCE 17 (1981)

[Beilken|Mattern|Reinfrank-85]
Beilken, Christian; Mattern, Friedemann; Reinfrank, Michael: Verteilte Terminierung, ein wesentlicher Aspekt der Kontrolle in verteilten Systemen (In German). Forthcoming technical report, SFB VLSI und Parallelitaet, Univ. Kaiserslautern (Kaiserslautern, Automn 1985).

[Beilken|Mattern|Spenke-83]
Entwurf und Implementierung von CSSA - Beschreibung der Sprache, des Compilers und eines Mehrrechnersimulationssystems (In German). MEMO-SEKI 82-03, Univ. Kaiserslautern (Kaiserslautern, September 1982)

[Binford-82]
Binford, Thomas A.: Survey of Model-Based Image Analysis Systems. INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH 1(1) (1982)

[Bondy|Murty-76]
Bondy, J.A.: Murty, U.S.R. : Graph Theory with Applications. The Macmillan Press Ltd (London, 1976)

[Borning-79]
Borning, Alan: THINGLAB - A Constraint Oriented Simulation Laboratory. PhD dissertation, Stanford University (Stanford, 1979)

[Clowes-71] Clowes, M.B.: On Seeing Things. ARTIFICIAL INTELLIGENCE 2 (1971)

[Cohen|Feigenbaum-82]
Cohen, Paul R.; Feigenbaum, Edward A.: The Handbook of Artificial Intelligence, Volume III. Pitman Books Limited (London, 1982)

[Davis|Rosenfeld-80]
Davis, L.S.; Rosenfeld, Azriel: Cooperating Processes for Low Level Vision: A Survey. TR-851, CS-Center, Univ. of Maryland (College Park, January 1980)

[deKleer-79]
deKleer, Johan: **Causal and Teleogical Reasoning in Circuit Recognition.** TR 529, AI-LAB, Massachusetts Institute of Technology (Cambridge, 1979)

[Douglas-82]
Douglas, Robert J.: Computing Occlusion with Locally

Connected Networks of Parallel Processes. In Preston; Uhr (eds.): Multicomputers and Image Processing Algorithms and Programs. Academic Press (New York, 1982)

[Fox et al-83]
Fox, Mark S.; Allen, Bradley P.; Smith, Steven F.; Strohm, Gay .A.: ISIS - A Constraint Directed Reasoning Approach to Job Shop Scheduling. CMU-RI-83-3, The Robotics Institute, Carnegie-Mellon University (Pittsburgh, 1983)

[Freuder-82]
Freuder, Eugene: On the Knowledge Required to Label a Picture Graph. ARTIFICIAL INTELLIGENCE 15 (1982)

[Gaschnig-79]
Gaschnig, J.G.: Perfomance Measurement and Analysis of Certain Search Algorithms. PhD Thesis, Carnegie-Mellon University (Pittsburgh, 1979)

[Gosling-83]
Gosling, James: Algebraic Constraints. CMU-CS-TR-83-132, Carnegie-Mellon University (Pittsburgh, 1983).

[Haralick|Elliot-80]
Haralick, R.M.; Elliot, G.L.: Increasing Tree Search Efficiency for Constraint Satisfaction Problems. ARTIFICILAL INTELLIGENCE 14, p. 263 (1980)

[Haralick|Shapiro-79]
Haralick, R.M.; Shapiro, L.G.: The Consistent Labelling Problem, Part I. IEEE TRANSACTIONS ON PAMI 1(2) (1979)

[Haralick|Shapiro-80]
Haralick, R.M.; Shapiro, L.G.: The Consistent Labelling Problem, Part II. IEEE TRANSACTIONS ON PAMI 2(3) (1980)

[Hein-82] Hein, Uwe: Constraints and Event Sequences. LITH-MAT-R-82-02, Linkoeping University (Linkoeping, Sweden, 1982)

[Huffman-71]
Huffman D.A.: Impossible Objects as Nonsense Sentences. MACHINE INTELLIGENCE 6 (Edinburgh, 1971)

[Kanade-80] Kanade, T.: A Theory of Origami World. ARTIFICIAL INTELLIGENCE 13(3), p.279 (1980)

[Kornfeld-81]
Kornfeld, W.A.: The Use of Parallelism to Implement a Heuristic Search. Proc. 7th IJCAI (1981)

[Lamport-78]
Lamport, Leslie: Time, Clocks, and the Ordering of Events in a Distributed System. COMMUNICATIONS OF THE ACM 21(7) (July 1978)

[Mackworth-73]
Mackworth, Alan K.: Interpreting Pictures of Polyhedral Scenes. ARTIFICIAL INTELLIGENCE 4 (1973)

[Mackworth-77a]
Mackworth, Alan K.: How to See a Simple World - An Exegesis of Some Computer Programs for Scene Analysis. In Elcock/Michie (eds.): MACHINE INTELLIGENCE 8, pp. 510-537 (1977)

[Mackworth-77b]
Mackworth, Alan K.: Consistency in Networks of Relations. ARTIFICIAL INTELLIGENCE 8 pp. 99-118 (1977)

[Mackworth-83]
Mackworth, Alan K.: On Seeing Things Again. Proc. 8th IJCAI (1983)

[Mackworth|Freuder-85]
Mackworth, Alan K.; Freuder, Eugene: The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. ARTIFICIAL INTELLIGENCE 25, p. 65 (1985)

[Marr-82] Marr, David (posth.): VISION. W.H. Freeman (San Francisco, 1982)

[Mattern|Beilken-85]
Mattern, Friedemann; Beilken, Christian: The Distributed Programming Language CSSA - A Very Short Introduction. Interner Bericht 123/85, Univ. Kaiserslautern (Kaiserslautern, January 1985)

[McAllester-80]
McAllester, David A.: An Outlook on Truth Maintenance. AI-LAB Memo 551, AI-LAB, Massachusetts Institute of Technology (Cambridge, 1980)

[Nehmer et al-85]
Nehmer, J.; Beilken, C.; Haban, D.; Massar, R.; Mattern, F.; Rombach, D.; Stamen, F.-J.; Weitz, B.; Wybranietz, D.: The Multicomputer Project INCAS - Objectives and Basic Concepts. SFB 124-Report 11/85 Univ. Kaiserslautern (Kaiserslautern, 1985)

[Nudel-83] Nudel, Bernard: Consistent Labelling Problems and their Algorithms: Expected Complexities and Theory-Based Heuristics. ARTIFICIAL INTELLIGENCE 21 (1983)

[Reinfrank-83]
Reinfrank, Michael Th.: The Implementation of SCENELAB (an annotated CSSA-program). Projektarbeit, Universitaet Kaiserslautern, (Kaiserslautern, July 1983)

[Reinfrank-84]
Reinfrank, Michael Th.: Distributed Constraint Propagation - A Case Study. MEMO-SEKI-84-07, Universitaet Kaiserslautern (Kaiserslautern, September 1984)

[Reinfrank-85]
Reinfrank, Michael Th.: An Introduction to Non-Monotonic Reasoning. MEMO-SEKI-85-02, Univ. Kaiserslautern (Kaiserslautern, June 1985)

[Rosenfeld|Hummel|Zucker-76]
Rosenfeld, Azriel; Hummel, R.A.; Zucker, Stephen: Scene Labelling by Relaxation Operations. IEEE TRANSACTIONS ON SMC 6(6) (1976)

[Sandewall-83]
Sandewall, Erik: Partial Models, Attribute Propagation Systems, and Non-Monotonic Semantics. LITH-IDA-R-83-01, Dept. of CS, Linkoeping University (Linkoeping, Sweden, December 1983)

[Steele-80] Steele, Guy L.: The Definition and Implementation of a Computer Language Based on Constraints. TR 595, AI-LAB, Massachusetts Institute of Technology (Cambridge, 1980)

[Steels-82] Steels, Luc: Constraints as Consultants. Proc. ECAI-82, pp. 75-78 (1982)

[Stefik-81] Stefik, Mark: Planning with Constraints (MOLGEN: PART I). ARTIFICIAL INTELLIGENCE 16 (1981)

[Stallman|Sussman-77]
Stallman, Richard M.; Sussman, Gerald J.: Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Analysis. ARTIFICIAL INTELLIGENCE 9 (1977)

[Sugihara-82]
Sugihara, K.: Mathematical Structures of Line Drawings of Polyhedrons - Towards Man-Machine Communication by Means of Line Drawings. IEEE TRANSACTIONS ON PAMI 4(5) (1982)

[Sussman|Steele-80]
Sussman, Gerald; Steele, Guy: CONSTRAINTS - A Language for Expressing Almost-Hierarchical Descriptions. ARTIFICIAL INTELLIGENCE 14 pp. 1-40 (1980)

[Tenenbaum|Barrow-76]
Tenenbaum, J.M.; Barrow, H.G.: IGS: A Paradigm for Integrating Image Segmentation and Interpretation. Proc. 3rd International Joint Conference on Pattern Recognition, pp. 504-513 (1976)

[Voss-82] Voss, Hans: Programming in a Distributed Environment: A Collection of CSSA Examples. MEMO-SEKI-82-01, Univ. Kaiserslautern (Kaiserslautern, 1982)

[Voss-85] Voss, Hans: Representing and Analyzing Time and Causality in HIQUAL Models. Proc. of the 1985 German Workshop on Artificial Intelligence (to appear)

[Waltz-72] Waltz, David: Generating Semantic Descriptions from Drawings of Scenes with Shadows. MAC-TR 271, Massachusetts Institute of Technology (Cambridge, 1972). This paper has been reprinted in [Winston-75].

[Winston-75]
Winston, Patrick H. (ed.): The Psychology of Computer Vision. McGraw Hill (New York, 1975)

[Winston-84]
Winston, Patrick H.: Artificial Intelligence, 2nd edition. Addison-Wesley (1984)

Appendix-B Working with SCENELAB

When the CSSA-environment is loaded, an INTERFACE-agent is made available for the user. In order to work with SCENELAB, he then must create both a SUPERVISOR and an LD-SERVER

```
const agent: SUPERVISOR := new(SUPERVISOR-SCRIPT);
const agent: LD-SERVER := new(LD-SERVER-SCRIPT);
```

After that, he will mainly communicate with these two agents.

If a picture description or a label dictionary is to be read from a file, these files must be previously edited with the usual editing facilities, using the desciption language PDL. Additionally, special I/O-agents must be created to read these files:

```
const agent: LD-READER := new(LD-READER-SCRIPT(LD-FILE));
const agent: PDF-READER :=
   new(PDF-READER-SCRIPT(PDF-FILE));
```

where LD-FILE and PDF-FILE are supposed to be the corresponding files. For later output, a special writing agent must be created:

```
const agent: SDF-WRITER :=
   new(SDF-WRITER-SCRIPT(SDF-FILE));
```

SDF-FILE here is a file where the user wants the final output to be written (if any).

Translation of the PDL-input descriptions is initiated by means of

```
send INTERPRETE(<source-agent>) to SUPERVISOR|LD-SERVER;
```

where <source-agent> is either the INTERFACE, for interactive input, or a reader agent. The SUPERVISOR resp. the LD-SERVER then enters a translation cycle where it requests one PDL-statement after another from the corresponding source agent and incrementally generates an internal representation. Such a request is fulfilled by

```
send REPLY(PDL-statement) to SUPERVISOR|LD-READER;
```

Whenever an error is detected in such a statement, the source-agent is set to INTERFACE, and an approriate error message is sent. The user then may indicate that he is willing to continue and correct the fault by

```
REPLY(*cont)
```

or he may stop the translation cycle by

```
REPLY(*term)
```

(or by anything else different from *cont). After correcting a buggy PDL-statement being supplied from a file, the user may switch back to the corresponding reader-agent by

```
REPLY(*switch)
```

Here is a grammar of all possible PDL statements.

```
<PDL-description>  ->  (<stmt>.)*

<stmt>  ->  <comment> | <cntrlstmt> | <defstmt> | <specstmt>

<comment>  ->  &<text-line>

<cntrlstmt>  ->  <cntrlhdr>{<cntrllist>}

<cntrlhdr>  ->  *<cntrlopn>

<cntrlopn>  ->  genid | switch | term | cont

                genid: makes a number of predefined junction
                       and segment identifiers available
                switch: switches the source-agent
                term: terminates the translation cycle
                cont: initiates the correcture of a buggy
                      PDL-statement

<defstmt>  ->  <defhdr> <deflist>

<defhdr>  ->  def-<defopn>

<defopn>  ->  fig | jun | lab | seg

              fig: junction figure
              jun: junction
              lab: segment label
              seg: segment


<deflist>  ->  <defelem>(;<defelem>)*

<defelem>  ->  <identifier>

<identifier>  ->  <letter>(<letter>|<digit>)*

<specstmt>  ->  <spechdr> <speclist>

<spechdr>  ->  spec-<specopn>

<specopn>  ->  com | deg | fig | gra | lab | res

               com: compatibility between segment labels
               deg: degree of a figure
               gra: (junction picture) graph at a junction
               lab: junction labels of a figure
               res: resulting labelling of a junction
```

(output statement only)

```
<speclist>  ->  <specelem>(;<specelem>)*

<specelem>  ->  <arg>:<valtuplelist>

<arg>  ->  <identifier>

<valtuplelist>  ->  <valtuple>(,<valtuple>)*

<valtuple>  ->  <val>(/<val>)*

<val>  ->  <identifier>
```

Besides a check for syntactical correctness according to this grammar, a number of semantical checks are performed, e.g. a consistency check of the type of identifiers in sepcification statements.

When the translation phase is terminated, the user may initiate the generation of a corresponding constraint network and start the propagation mechanism. The most important SCENELAB commands are:

send INTERPRETE(<source-agent>) to LD-SERVER|INTERPRETER;

send INIT-NET to SUPERVISOR;
create the constraint network and initialize the LOCAL ANALYSTS

send START-NET to SUPERVISOR;
start the constraint propagation

send CHECK-NET to SUPERVISOR;
freeze the network and initiate a termination check

send RESTART-NET to SUPERVISOR;
restart the constraint propagation after a freeze

send GENERATE-OUTPUT to SUPERVISOR;
initiate the generation of a scene description file that contains the PDL description of the labelled line-drawing

send EXCLUDE(<label>,<junction>,<segment>) to SUPERVISOR;
exclude the label <label> for the segment <segment>, seen from the point of view of the junction <junction>

send DISPLAY-LABELS(<junction>) to SUPERVISOR;
the actual labelling of <junction> is sent to the INTERFACE

send SHOW-LABELS(<figure>) to LD-SERVER;
the dictionary page of <figure> is sent to the INTERFACE

Furthermore, the CSSA-environment provides a number of additional facilities that allow, among other things, to trace the operations of an agent, to inspect the mailbox of an agent, and to observe the message passing activities. The latter facility can be used to trace the entire constraint propagation process.