GEOMETRIC CONSTRAINTS FROM 2 1/2D SKETCH DATA AND OBJECT MODELS

R B Fisher
M J L Orr

DAI RESEARCH PAPER NO. 318

Paper presented at *Geometric Reasoning Conference*, Winchester, 1987.

GEOMETRIC CONSTRAINTS FROM 2 1/2D SKETCH DATA AND OBJECT MODELS

Robert B. Fisher

Mark J. L. Orr

Department of Artificial Intelligence

University of Edinburgh

Abstract

In this paper we describe geometric reasoning employed in a model-based vision system that uses surface data. Such reasoning is used to draw inferences about the spatial relationships between objects in a scene based on the fragmentary and uncertain geometric evidence provided by an image.

The paper reports on three aspects of our current work: understanding and properly specifying the tasks of a visual geometry reasoner, the types of geometric constraints that can be defined when three-dimensional data is paired to three-dimensional models, and a parallel network computation for evaluating the constraints.

Keywords: geometric reasoning, computer vision, surfaces

1. Introduction

A competent vision system needs to be able to locate objects in the environment as well as identify them. Hence, it is necessary to derive and integrate position information from data elements paired with model features. There are additional applications of geometric reasoning in the context of model-based vision:

- the prediction of image locations (or other properties) for additional evidence and
- testing of proposed model-to-data pairings by ensuring the geometric relationships embodied in the model are upheld by the data.

The first major section of this paper summarises a recent study [1] of several high-performance vision systems, classifying the tasks and operations used in their visual analysis. Analysis has shown that there are a few simple underlying geometric operations but that the types of models and data used drastically affect the machinery required to implement them.

Several recent vision projects here [2,3] have been concentrating on recognising and locating objects using surface based models and 3D surface data. The justification for this effort is two-fold. First, research into recognition based on edges has shown how fragile this approach is. Second, much low-level vision research is investigating delivery of some form of 3D scene descriptions, whose advantages are: (1) the interpretation of surface data is unambiguous (unlike intensity data, which can arise from several scene effects) and (2) 3D gives direct information about the scene (unlike projected intensity images). In consequence, our research has been investigating using surfaces as the primary data and model features.

To exploit the richer information in surface data, we have been investigating surface-based object representations [3]. Object primitives are represented using groups of rigidly connected surfaces, which have been segmented into regions of near constant shape character. Larger objects are hierarchically defined by connecting previously defined subcomponents, where the connections need not be rigid. Surface primitives promote direct pairing between model and data surfaces, easier extraction of object positions (using the 3D data) and easier prediction of feature visibility. Related work is also investigating the role of other geometric entities in the data and models (e.g. volumetric and boundary representations).

One important research question that has arisen from the conjunction of the model and data types is: what constraints on object position arise from the pairing of a model feature with a data feature? This is complicated, because of (1) the variety of data and model features, (2) the variety of constraints types obtainable from pairing such features and (3) the existence of constraints that are only partial. This paper reports our recent work in this area.

Finally, we have been investigating mechanisms for implementing the geometric reasoning functions needed for vision. Some of the difficulties in this area lie in:

- integrating partial constraints,

- coping with data errors,
- finding fast mechanisms.

The final main section of this paper summarises initial results in new research into a parallel network geometric reasoning engine. The network uses inequality constraints (the arcs) on individual geometric variables (the nodes). Integration of constraints occurs when the network converges to a consistent state. The structure of the network is predefined using some extensions of ACRONYM's [4] constraint manipulation system. This allows slow symbolic manipulation at network definition time, yet fast execution during recognition.

2. Geometric Reasoning for Model-Based Vision

The ability to reason about the geometry of a scene is an essential aspect of any sophisticated vision system. Geometric reasoning can be understood at three levels:

1) the various tasks that a vision system requires of a geometric reasoner (section 2.1),

2) the ideal data types and operations required for the tasks (section 2.2) and

3) the implementations of the operations (section 2.3). As we will see, it is easier to formulate the required operations than to find perfect implementations.

2.1. Geometric Reasoning Tasks

A geometric reasoner is characterised by the tasks that it is expected to carry out. Based on an analysis of existing vision systems, we have identified the following tasks.

Establishing Position Estimates

Every identified feature in an image can be used to form position constraints, first because the feature is visible, and second through its measurable properties (location, shape, dimensions and so on). In order to aggregate information from related features it must be possible to combine individual position constraints into a single position estimate. During such combination, the detection of inconsistent constraints is required to eliminate false hypotheses (formed, for example, from erroneous feature identifications).

It may be necessary to transform position estimates into another reference frame before other functions are applied. One example of this is when the position of a feature in an object's reference frame is desired, given only knowledge of the feature relative to a subcomponent and the subcomponent relative to the object.

If the modeled relation between parts of the same object involve degrees of freedom then there has to be some variable binding.

Image Prediction

Given a position estimate for an object it should be possible to predict the appearance and location of its features. This allows comparison between the predicted and observed features, and affords a basis for reasoning about occlusion effects. Additionally, image prediction can be used to search for features not already found.

Predicted features are not necessarily pixel type entities. They may also be more symbolic entities such as points, lines, surfaces, normals etc. Furthermore, real images are formed from objects with exact positions, but predicted images may involve objects whose positions are only roughly known, hence the predictions should be able to represent some kind of uncertainty.

2.2. Geometric Reasoning Functions

The second descriptive level of geometric reasoning concerns the abstract data types and operations required for the tasks described above.

Positions

Geometric reasoning requires a data type for representing positions. The traditional representation by three translational and three rotational degrees of freedom is not adequate for our purposes. Extensions are needed for:

- uncertainty present in image measurements,
- modeling objects with positional degrees of freedom, and
- modeling objects with size variation or tolerances.

Positions also transform points, vectors and other positions from one coordinate frame to another (e.g. the position of a subcomponent is equivalent to a transformation from the object's to the subcomponent's coordinate reference frame.) Hence, if we extend the notion of position from a point to a region of 6D parameter space, then these transformations are no longer one to one mappings, and we will have to deal with uncertain points and vectors.

In what follows we will be giving some simple data type specifications [5] using the operators FRAME and PLACED (capital letters will be used for all operators). Both operate on members of the set Position and return members of the set Model. The latter includes the special models World and Camera so that we can have world centered and viewer centered coordinate systems as well as relative positions between models. Thus we write the functionality of FRAME and PLACED as:

FRAME: Position -> Model

PLACED: Position -> Model

FRAME returns the model whose frame is the reference frame of a position and PLACED yields the model placed by a position.

Estimating Positions from Features

Each pairing of a model to a data feature produces constraints on the position of the model to which the feature belongs. We have then an operation, LOCATE, whose inputs are the model feature and the image feature and that yields a position estimate whose FRAME is the Camera and whose PLACED object is the model.

```
LOCATE: Image_feature, Model_feature -> Position u {undefined}

for all $f_i$ ε Image_Feature and $f_m$ ε Model_Feature:
        let p = LOCATE($f_i$, $f_m$)
        if p ≠ undefined then
                FRAME(p) = Camera &
                PLACED(p) = m

where:  m is the model to which $f_m$ belongs,
        u stands for set union, and
        {undefined} is a set with one member - the undefined object.
```

LOCATE returns 'undefined' to signal an invalid pairing between incompatible image and model features.

Merging Positions

In general models consist of more than just a single feature. If several features are identified in the data and produce constraints on the object position, then we need to verify geometric consistency and merge estimates.

The MERGE operation acts on sets of positions and returns a position. The result is only defined when all the input positions have the same coordinate frame and refer to the same object. The 'inconsistent' result means the input positions were inconsistent. If #(Position) is the power set of Position (the set of all possible subsets of Position):

```
MERGE: #(Position) -> Position u {undefined, inconsistent}

for all $m_1$, $m_2$ ε Model, S ε #(Position):

        let p = MERGE(S)
        if (for all q ε S: FRAME(q) = $m_1$ & PLACED(q) = $m_2$) then

                p = inconsistent or

                FRAME(p) = $m_1$ and PLACED(p) = $m_2$;

        else

                p = undefined
```

Transforming Position Constraints

Suppose we know the position of an object A relative to another object B. This may occur if:

- they are parts of the same larger model assembly,
- we have **a priori** knowledge about their relationship (e.g. the position of the camera in the world) or
- we observe the relationships between their features in the image (e.g. "face 1 of A is against face 2 of B").

Two geometric problems then arise. First, if we know the position of A in the frame of some other object C, what is the position of B in this frame. Second, if instead we know the position of C in A's frame, what is the position of C in B's frame. These problems require the operations TRANSFORM and INVERSE that obey the following rules in relation to the operators FRAME and PLACED.

```
TRANSFORM:      Position, Position -> Position ∪ {undefined}

INVERSE:        Position -> Position

for all p, q ε Position

        let r = TRANSFORM(p, q)

        if PLACED(p) = FRAME(q) then

                FRAME(r) = FRAME(p) &

                PLACED(r) = PLACED(q)

        else

                r = undefined

for all p ε Position

        FRAME(INVERSE(p)) = PLACED(p)

        PLACED(INVERSE(p)) = FRAME(p)
```

Now if we represent by X/Y a position whose FRAME is X and whose PLACED object is Y, our two problems can be written as:

```
C/B = TRANSFORM(C/A, A/B)                  [1st problem]

B/C = TRANSFORM(INVERSE(A/B), A/C)         [2nd problem]
```

Image Prediction

Image prediction involves several operations that differ only by what is being predicted. Typical image predictions are:

- feature visibility,
- feature orientation,
- feature appearance,
- feature location (image and scene),

- feature relative depth (e.g. surface ordering and occlusion relations),

The operation to be performed in any given prediction task depends on both the task and nature of the feature whose image is being predicted. For example, to determine whether a plane surface is front-facing requires projecting a predicted surface normal along the line of sight, but this operation would be insufficient if the surface were curved. We abstract them all into the PREDICT operator, which must have knowledge of the different feature types.

PREDICT: Model_feature, Position -> Pred_feature

where Pred_feature is a separate data type from Image_feature because it must incorporate variations due to uncertain positions.

2.3. Review of Current Implementations

Some existing AI programs, including RAPT [6], IMAGINE [2] and ACRONYM [4], have geometric reasoning capabilities. RAPT, a robot planning program, and ACRONYM, a model-based vision program, both represent and manipulate positions symbolically and exploit relationships between symbolic expressions (equalities or inequalities) for their deductive power. RAPT is less powerful as it can only deal with relationships that are equalities. In contrast to these IMAGINE uses a numerical representation scheme with two separate position data types. One type (homogeneous matrix) is designed for the TRANSFORM operator and the other (parameter bounds) for MERGE. A problem with this scheme is that the matrix type represents exact positions whereas the parameter bounds represent rough positions and so conversion between the two forms involves loss of information or approximation.

ACRONYM represents positions using a variable for each degree of freedom. Constraints on positions are formed by relating expressions in the variables to quantities measured from the image and manipulated symbolically.

The ideal constraint manipulation system (CMS) should be able to:

(a) decide if each variable has a value consistent with the constraint set and

(b) bound any expression in the variables over the constraint set.

The operations MERGE, TRANSFORM and PREDICT are achieved by unioning constraint sets, simplifying symbolic compositions of positions and bounding expressions in variables.

Three advantages of using an algebraic constraint representation are:

(1) - The representation is **incremental** in two senses. First, new constraints can be added when each new piece of evidence is discovered. Second, new classes of constraints can be added when new visual relationships are understood.

(2) - The representation is **uniform**, because it can represent a large range of visual relationships using the same mechanism.

(3) - **A priori** knowledge of scene relationships (e.g. the object must lie on the conveyor belt) is also expressible in algebraic form, allowing direct integration with observed and model relations.

We think that ACRONYM offers the best representation and machinery for geometric reasoning amongst those available.

3. Geometric Constraints from Surfaces

Most geometric reasoning operations (e.g. MERGE, TRANSFORM, INVERSE) can be defined abstractly, based only on the representation for positions. Hence, the key visual geometry problems are relating object positions to image-model feature pairings (i.e. LOCATE and PREDICT).

Since we have been investigating using surface patches as the primary model and data descriptive primitive, the natural questions are:

- what geometric constraints can be obtained using these primitives, and
- how can we represent them algebraically. This section attempts to answer these questions.

3.1. Surface Data

The raw data used here is three dimensional surface information including absolute surface depth and orientation, such as found in Marr's $2\frac{1}{2}$D sketch [7] or as used in Fisher [2]. The data is organised in a pointillist array aligned with a standard intensity image and segmenting the surface image into regions of nearly uniform shape, characterised by the two principal curvatures and the surface boundary. While there are no well developed data acquisition systems that deliver high quality surface descriptions of the scene, several promising data sources are under development, including laser striper range finders, optical flow and stereo.

Some advantages of using surface information are:

- surface information is explicitly 3D, so 3D properties can be calculated directly, rather than deduced from 2D image properties,
- surfaces can be segmented from both the model and the data according to the same criteria, hence producing directly corresponding features (disregarding scale questions),
- it makes explicit features found on non-blocks-world objects (e.g. curved surface patches),
- it makes explicit where occlusion occurs (depth discontinuity boundaries)

3.2. Surface-Based Modeling

Model-based object recognition requires geometric object models. Here, the models [2,3] are designed for object recognition, not image creation, so the model primitives are based on matchable data features.

The models use surface patches as the major primitive, because the surface is the primary data unit (though space curves and blob-like volumes are also represented). This allows more direct pairing of data with models, comparison of surface shapes and estimation of model to scene transformation parameters.

It is assumed that the surfaces of a large class of objects can be approximately segmented into patches of nearly constant shape. The surface patches are then described by their principal curvatures and boundary. Surfaces have zero, one or two directions of curvature (positive or negative). Size and shape descriptions can involve variable quantities. General algebraic constraints involving variables can be included.

Objects are recursively constructed from surfaces or subobjects using coordinate reference frame transformations. Each structure has its own local reference frame transformation and larger structures are constructed by placing the subcomponents in the reference frame of the aggregate. Variables occurring in the expressions that define the attachment relationship allow partially constrained relative placement.

One additional class of feature is the viewpoint dependent feature, such as the tangential boundary on a cylinder. The cylinder has no such boundaries in its definition, but we generally observe these because of the viewer-object relationship. While the existence of these can be deduced from the geometric model, we assume here that the model represents them explicitly.

Other features of the models not discussed here are: viewpoint dependent feature groupings and scale-based object representation simplifications.

3.3. Geometric constraints from model to data feature pairings

In ACRONYM [4] a single constraint is generated by first measuring upper and lower bounds for some scalar quantity, S, and then relating these values to a known algebraic expression, E, which involves a subset, Q, of the quantifiers or variables of interest. If S_u and S_l are the upper and lower bounds on the true value of S then the contribution from the constraint would be the inequalities:

$$E(Q) \leq S_u$$

$E(Q) \geq S_1$

ACRONYM, however, dealt only with constraints generated from pairing 2D edge features with a restricted vocabulary of model primitives. Here, we want to deal with richer 3D descriptions for both images and models, and have found it useful to classify the types of constraints generated by pairings of the various image and model features.

All measurements from 3D images take the form of vectors: point locations or directions in space. Constraints are then generated by pairing up the measured vectors with corresponding vectors in some model frame. When it is possible to make this pairing on a one-to-one basis we classify the constraint as type A. When there is ambiguity and we can only make the correspondence with some range of model vectors then the constraint is of type B.

Consider the example of a straight edge segment whose endpoints are obscured. The observed direction of the edge corresponds directly with its direction in the model frame and therefore generates a type A constraint on orientation. However, the visible length, l, is less than the true length l_0, and the visible endpoints do not correspond with the actual endpoints but with ranges of length $(l_0 - l)$. Consequently, the translational constraint generated from the endpoints is type B.

Constraints can be further divided according to the number, m, of point position vectors and number, n, of direction vectors involved. For type A constraints, if $m = 0$ or $(m + n) < 3$ the model position is not fully constrained. Type B constraints, because they always involve some ambiguity, are never fully constraining. However, in any model to image pairing there will normally be several feature pairings and their degrees degrees of freedom may well be orthogonal so that their combined effect is to fully constrain the model position.

For dealing with errors we have adopted a scheme whereby point vectors have associated error spheres and direction vectors have error cones. An estimated point position is therefore given by four numbers: three for the location and one for the radius of the sphere containing the true point. Similarly, an estimated direction is associated with three numbers: two for the nominal direction and one for the cone angle containing the true direction.

Suppose that a type A constraint involves the pairing of model point $\mathbf{p}_m$ with image point $\mathbf{p}_i$ and model direction $\mathbf{d}_m$ with image direction $\mathbf{d}_i$. If $\mathbf{p}_i$ and $\mathbf{d}_i$ are error free, then we can write:

$$\mathbf{R}(\mathbf{p}_m) + \mathbf{t} = \mathbf{p}_i$$

and

$$\mathbf{R}(\mathbf{d}_m) = \mathbf{d}_i$$

where $\mathbf{R}$ is the rotational and $\mathbf{t}$ the translational part of the model position. If $\mathbf{p}_i$ has error sphere radius ε and $\mathbf{d}_i$ error cone angle μ then the constraints would take the form:

$$| \mathbf{R}(\mathbf{p}_m) + \mathbf{t} - \mathbf{p}_i | \leq \varepsilon$$

and

$$\mathbf{R}(\mathbf{d}_m) \cdot \mathbf{d}_i \geq \cos(\mu)$$

These two equations exemplify the constraint equations generated by the measurement of, respectively, location and direction vectors. For each additional direction or point vector involved in a constraint another similar equation is generated.

The equational form of type B constraints is similar to that shown above except that $\mathbf{d}_m$ and $\mathbf{p}_m$ are parameterised by one or more variables. The variables are themselves subject to inequalities describing their allowed ranges. In the example of a straight edge with its ends obscured, the locus of model points corresponding to the visible end at $\mathbf{p}_i$ is a line which can be parameterised by a single variable, β. We have (ignoring errors):

$$R(\mathbf{p}_m + \beta\mathbf{d}_m) + \mathbf{t} = \mathbf{p}_i$$

and

$$0 \leq \beta \leq (l_0 - l)$$

We now illustrate constraint generation with a practical example. Figure 3-1a depicts a scene containing a cylindrical shaped surface patch and figure 3-1b depicts the features recorded in the model. The model patch lies on a notionally infinite cylinder of radius r lying along the x-axis. Four segments bound the patch, two straight and two circular, meeting at the points labelled A, B, C and D.

Figure 3-1: a) data surface patch, b) model patch, and c) obscured data patch.

We assume that the vision system has been able to identify the individual features of the patch so that the scene points A, B and C can be labelled as in figure 3-1a (the point D is not visible). There are then several constraints available from the data which together are more than enough to fully constrain the position of the patch. Each visible boundary vertex, A, B and C, together with the tangents to the boundaries defining them, give a type A constraint involving one point location vector (m=1) and two direction vectors (n=2). The circular arcs AFB and EC imply the position of axis points M_1 and M_2 as well as the direction of cylinder axis - two type A constraints with m=1 and n=1. The lines BC and AX and the surface normals along them give similar constraints as does the obscuring boundary FE since we know that the surface normal along FE is perpendicular to the line of sight.

When many of the type A features are obscured we have to resort to type B features. In figure 3-1c we see that most of the patch is hidden behind an obscuring object (shaded). None of the verticies are visible - instead we can only see where the boundary segments have been truncated at points A', B' and C'. As before the obscuring boundary FE and the arcs A'FB' and EC' can be used to estimate the point positions M_1 and M_2 and the cylinder axis direction. This still leaves a rotational degree of freedom - the angle ϕ around the cylinder axis. To constrain this angle we can use the arcs A'FB' and EC' in a similar fashion to the example above of a line with its endpoints obscured. We then obtain type B constraints which narrow down the range of possible values of ϕ.

Every feature in a modelling scheme is capable of producing constraints which can be exploited by the geometric reasoner. We are currently engaged in compiling a catalogue of the constraint types available from the various features of the surface based scheme described above. This will be used by the geometric reasoner to automatically relate any particular feature pairing to inequalities involving estimated quantities and free variables appropriate to that pairing. These inequalities will be the input to the constraint solving engine described in the next section.

4. A Network-Based Geometric Reasoning Engine

Since the geometric constraints defining object position are defined algebraically, following ACRONYM [4], one might use ACRONYM's constraint manipulation system (CMS). This CMS could bound expressions over non-linear inequalities and achieved respectable performance through a combination of case analysis and considerable symbolic algebra.

The advantage of using symbolic algebra instead of merely bounding each term is seen in the following example. Suppose we wish to bound the expression "A*B", where A and B are functions of X, which must lie in the range [1,2]. Suppose A = X and B = 2 / X. Bounding A*B by bounding and multiplying the individual terms gives the range [1,4]. However, using symbolic algebra to reduce A*B first, gives the tighter bound [2,2].

Unfortunately, symbolic algebra is expensive computationally and we aim to eventually analyze scenes in real time. Hence, we are adapting it for use in a parallel evaluation network. ACRONYM's CMS is used to derive symbolic bounds on non-observable values (e.g. object orientation) in terms of observable values (e.g. direction of a surface normal), which would initially be expressed as variables. These bounds define a network of nodes representing expressions linked by their algebraic relationships. The network can be pre-compiled before any observations, with sections activated only when appropriate evidence is obtained.

The network is evaluated in parallel when image evidence is obtained to (1) provide bounds on position and size parameters and (2) determine if an inconsistency is present (e.g. wrong model or model-data pairing). The parallel evaluation provides the necessary performance.

The methodology we are investigating is summarised in figure 4-1 below. At the top the constraint schemas discussed in section 3 are combined with models to define sets of algebraic constraints. Image observables are represented by variables at this stage. These constraints are then algebraically manipulated to produce a set of inequalities bounding each variable. An extended constraint manipulation system (CMS) based on ACRONYM's CMS is then used to form tighter bounds on each variable. The inequalities define the nodes and arcs in a network fragment, that are associated with object hypotheses to form complete networks (with inter-linkages through shared variables). When observable variables get bound to measured values the other variables (e.g. model or position variables) are forced into consistency.

Figure 4-1: Network Definition from Geometric Constraints

At present, the definition of the network method is largely only a paper exercise. However, the example in section 4.5 should show the feasibility of the approach, which we are continuing to investigate.

In the discussion below, the term **symbolic CMS** means our adaptation and improvement of ACRONYM. The term **network CMS** means the compiled network, which implements the same function as the symbolic CMS in a different manner.

4.1. Constraint Networks

A set of algebraic constraints can be viewed as a graph, where the nodes represent expressions and the arcs represent inequality relationships (binary) between them. Each node is identified with a processor and the arcs provide bounding input values.

We illustrate the network structure with the fragment for the inequality "$A \leq B - C$" in the given variables. (Whether the variables represent model, position or observable values is not important.) Two other equivalent relations are "$C \leq B - A$" and "$B \geq A + C$". (E.g. we want to bound all variables by expressions in other variables.) The symbolic CMS is then used to calculate supremum and infimum expressions for some of these variables. This gives:

$$\sup(A) \leq \sup(B - C) \leq \sup(B) - \inf(C)$$

$$\sup(C) \leq \sup(B - A) \leq \sup(B) - \inf(A)$$

$$\inf(B) \geq \inf(A + C) \geq \inf(A) + \inf(C)$$

This formulation suggests introducing new nodes to represent expressions, and new processing elements for combining bounds.

A value node is now associated with each value, whether a variable or an expression, and represents the current supremum and infimum of the value. Hence the nodes relate to: {A, B, C, B-C, B-A, A+C}.

The computation for the supremum (e.g. sup(A)) picks the minimum of each bounding expression on the supremum, including the current supremum, because the bounds can never get worse (assuming a bound is always valid). Hence, if:

$\sup(A) \leq X$

$\sup(A) \leq Y$

then:

$\sup(A_{t+1}) \leftarrow \min(\sup(A_t), X_t, Y_t)$

is the updating function for the supremum of A.

Since the bounding expression for "sup(B - A)" is a function of two expression bounds, we need to introduce arithmetic nodes into the network to compute the these functions. (Because some constraints are dependent on the sign of the expressions, gating nodes are also used in the networks, but are not exemplified here.)

Applying these ideas results in the network shown in figure 4-2. This fragment is just for the single inequality given, and other constraints will introduce new nodes and linkages.

Figure 4-2: Network Fragment for "A ≤ B - C"

In execution, some variables acquire explicit bounds from measurements. The network then iteratively computes bounds on each expression until the network converges. The bounds give the allowable range for the variables consistent with all measurements and constraints. This implements the LOCATE and MERGE functions. Inconsistency is detected when "sup(X) < inf(X)" for some expression X. The PREDICT function is implemented by examining the bounds on variables related to measurements, given boundings on other variables.

4.2. Extensions to ACRONYM's CMS

ACRONYM does not handle certain quadratic functions well. For example, with the following set of constraints, it fails to achieve the minimal upper bound. If:

$$0 \leq x \leq 2$$

$$0 \leq y \leq 2$$

$$x + y \leq 2$$

then ACRONYM derives 4 as an upper bound on xy, whereas 1 is the correct upper bound. This weakness stems from not treating quadratic expressions as special cases.

The symbolic CMS we use is based on ACRONYM, but has been extended to give better bounds on functions of the form:

$$A*x^2 + B*x + C$$

and

$$A*\cos(x) + B*\sin(x)$$

where we have numerical bounds on A, B, C, and x. The extensions are not included here for brevity.

These extensions are particularly important because of the presence of rotation expressions in visual geometry, which often contain terms of the above form. (The cos(x) and sin(x) terms can be represented as y and $\sqrt{(1 - y^2)}$.

4.3. Network Pre-Compilation

Section 3 showed that a series of algebraic constraints could be defined for different evidence types and model-data pairings. Though numerous, it is possible to enumerate all constraints on each structure before making any observations, except in the case where the number of data items is indeterminate (such as the number of surface normals observed for a given surface). Disallowing this case, the complete set of constraints on a model are represented with symbolic variables for each potential observed value. This constraint set would also include constraints defined as part of the model.

Whenever any measurements related to the object were made, then the related constraints could be enabled, or included in an active set. Alternatively, one could initialise all bounds on observed variables to be infinite.

Both defining the full constraint set and applying the symbolic CMS to produce symbolic bounding expressions can be done at model-compilation time. This may be slow, because there may be many variables in the expressions at this stage. However, this is unimportant, because this stage is done off-line. The only occasions for repeating the analysis are:

- new constraints types become understood and generated, and
- new CMS techniques are implemented.

There is no difference in competence between the symbolic CMS and the network, provided data variables are set only to numbers. If they acquired values containing variables, then the symbolic CMS might be able to improve the bounds on the other variables. Because the data variables are intended to represent measured quantities, the competence assumption is reasonable. Hence, the only real difference between the two is performance, where using the network has clear run-time advantages because it can be executed in parallel using simple arithmetic units.

Using the symbolic CMS with some measured values might produce a simpler constraint expression than that containing measurement variables. For example, it may reduce

$$\sup(x) = \min(5,\ 10*x)$$

to

$$\sup(x) = 5$$

when the constraint "$x \geq 1$" is added. However, since the network executes in parallel, no significant execution time differences should occur.

4.4. Example

The geometric reasoning promoted in this paper is illustrated by an example of constraining position variables. The problem is simplified to two dimensions for illustration.

Figure 4-3a shows a hypothetical unit line segment. It's location is defined by the translation of its central point and orientation (anti-clockwise) about the central point. The model segment is (redundantly) characterised by its position in a local coordinate frame:

$$p_m = (0,0)$$

its direction:

$$\mathbf{d}_m = (1,0)$$

and its normal:

$$\mathbf{n}_m = (0,-1)$$

Figure 4-3: Model and Data Line Segment

Suppose the model appears in the scene with the rotation θ and translation $\mathbf{t}_d = (t_x,t_y)$ (see figure 4-3b). Here, $\theta = -\pi/4$ and $\mathbf{t}_d = (0,0)$. Hence, the scene position, direction and normal are:

$$\mathbf{p}_d = R_\theta \mathbf{p}_m + \mathbf{t}_d = (t_x,t_y) = (0,0) \qquad (1)$$
$$\mathbf{d}_d = (0.7,-0.7)$$
$$\mathbf{n}_d = (-0.7,-0.7)$$

where $\mathbf{R}_\theta$ is the rotation matrix:

$$\begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix}$$

Suppose next that we observe an estimated normal $\hat{\mathbf{n}}_d = (\hat{n}_x,\hat{n}_y)$ at some unknown point $\mathbf{v}_d$ on the segment:

$$\mathbf{v}_d = \mathbf{p}_d + l*\mathbf{d}_d \qquad (2)$$

and hypothesise that $l \in [-0.4, 0.4]$. This is a type B constraint (see section 3) because we do not know which model point exactly corresponds with the point we measure.

Let the observed location be $\hat{\mathbf{v}} = (\hat{v}_x, \hat{v}_y)$. Since our observations must have some error, we know that:

$$|\ \mathbf{v}_d - \hat{\mathbf{v}}\ | \leq \varepsilon_1 \qquad (3)$$

$$1 - \hat{\mathbf{n}}\cdot\mathbf{n}_d \leq \varepsilon_2 \qquad (4)$$

Assume $\varepsilon_1 = 0.05$ $\varepsilon_2 = 0.005$.

Constraints (1) - (4) above define the position of the segment. The reasoning task is to combine the constraints to bound the possible range of its positions. Constraints (1) and (2) are part of the model, whereas (3) and (4) are returned by the LOCATE function. Representing all constraints in the network implements the MERGE function.

Hence, our problem is to bound the position variables θ, t_x, t_y and l given the observed variables $\hat{n}_x$, $\hat{n}_y$, $\hat{v}_x$ and $\hat{v}_y$. Assume that we observe the true normal at the true central point. Then:

$$\hat{\mathbf{n}} = (-0.7, -0.7)$$

$$\hat{\mathbf{v}} = (0,0)$$

The geometric constraints (1) - (4) reduce to a set of algebraic constraints on the variables, such as:

$$t_x \leq 0.05 - l*\cos(\theta)$$

$$t_x \geq -0.05 - l*\cos(\theta)$$

$$\cos(\theta) \leq -1.34 - \sin(\theta)$$

$$l \leq 0.4$$

plus a general equality:

$$| \cos(\theta) | = \sqrt{(1 - \sin^2(\theta))}$$

These together define the network solving for the position variables. Figure 4-4 shows the fragment for the "x" and "cos(θ)" nodes in schematic form and embodies the relations between $\hat{v}_x$, t_x, l and $\cos(\theta)$. From constraints (1), (2) and (3) we know that:

$$| \hat{v}_x - l*\cos(\theta) - t_x | \leq \varepsilon_1$$

The initial value ranges of the observed variables are $\inf(\hat{v}_x) = -\varepsilon_1$, $\sup(\hat{v}_x) = \varepsilon_1$, $\inf(l) = -0.4$ and $\sup(l) = 0.4$.

Figure 4-4: Network Fragment for "X" Linked Nodes

Letting this network converge, the following bounds on the position variables are obtained:

$$-0.4 \leq l \leq 0.4$$

$$-0.88 \leq \theta \leq -0.68$$

$$-0.36 \leq t_x \leq 0.36$$

$$-0.36 \leq t_y \leq 0.36$$

Since this is equal to the best bounds obtained by analytic solution, the network has been successful at bounding the variable values.

4.5. Network Behaviour

Three key computational questions are how does the network get evaluated, does it converge and does it converge to the best bound.

The network is intended to be executed in parallel, with each node in a separate processor. It could be evaluated synchronously or asynchronously in a MIMD processor with non-local connectivity, because expression relationships are not regular. Since the size of the expression relating two variables is typically less than 20 terms, and since it appears to only take a few iterations (e.g. 5-10) to evaluate the networks (at least the simple ones we've tried so far), we expect fast convergence.

Here, parallel evaluation is simulated serially using conventional programming (i.e. prolog). Keeping track of which dependent nodes need re-evaluating when one node changes eliminates recomputation in stable sections of the network.

It is easy to show that the networks must converge, that is, not oscillate. At any time the current bounds on a value V must be true. Then, if an upper bounding expression increases in value, the original bounds on V must still hold, as it then makes no sense to increase the range of potential values for V. Hence, the bounds can only get tighter, though perhaps asymptotically. As the bounds can only be equal (inconsistency is declared if they cross), each bound has a limit, so must converge.

ACRONYM's CMS was optimal when producing numerically bounded variables (i.e. not resulting in expressions in other variables) over sets of linear constraints, based on work by Bledsoe [8] and Shostak [9]. Since we reproduce all alternatives of the symbolic reasoning in the network, without substituting values for the data variables until later, it is likely to have the same performance on linear constraint sets, but we have not proved this result.

As most of the constraints derived in the previous subsection are non-linear, we cannot expect optimality. However, the inclusion of the quadratic and trigonometric bounding extensions should improve performance by accounting for most of the non-linear cases.

4.6. Continuing Research

Some areas where we are continuing research are:

trigonometric wrap-around

The example was set up so that we did not have to worry about cases where $\cos(\theta) > 0.9$ or $\cos(\theta) < -0.9$.

bounding X in A*X<B when the sign of A is unknown

If A can take values in the range [-r,s], then B/A can take any value. Hence, this constraint is not useful until the sign of A is known. Thereafter, the sense of the constraint depends on the sign. Consequently, the network structure should probably allow gating to switch on constraints when parity conditions hold.

automatic network compilation

The network should be automatically constructed from the constraints. Because the geometric constraints can be defined in standard form, isolating the variables, applying the symbolic CMS and compiling the network should be easy.

5. Conclusions

This paper has discussed three topics relating geometric reasoning to computer vision:

(1) What tasks must a visual geometric reasoner perform and what functions are needed to implement them?

(2) What classes of constraints exist from pairing 3D model features to 3D image features, and how they can be represented?

(3) How the constraints can be represented in a computational network to provide fast parallel bounding of variables and expressions, which is necessary for real time estimation of object position and model parameters?

Through analysing several high performance vision systems, the main reasoning tasks identified were: forming position constraints, combining constraints, detecting inconsistencies, transforming positions, variable binding and feature prediction. The main functions needed to implement the tasks were specified as abstract data types.

Sets of equality and inequality constraints were based on pairing point and vector features defined 3D model and data features. The constraints were represented in algebraic form. A network structure could be defined based on inequalities derived from the constraints. This network is suitable for parallel evaluation. An example of a network that successfully performing geometric reasoning was given.

This research is being pursued as part of a larger model-basec visual scene understanding project, which is still being implementec. Consequently, details of network reasoner performance on larger problems are not yet available.

Acknowledgements The work presented above was funded under Alvey grant GR/D/1740.3. Thanks go to J. Aylett and the Robot/Vision discussion group for helpful comments.

References

[1] Orr, M. J. L., and Fisher, R. B., "Geometric Reasoning for Computer Vision", presented at 1986 Alvey Computer Vision and Image Interpretation Meeting, also Dept. of Artificial Intelligence Research Paper 311, Univ. of Edinburgh, 1986.

[2] Fisher, R.B., "From surfaces to objects: recognising objects using surface information and object models", PhD. Thesis, University of Edinburgh, 1986.

[3] Fisher, R. B., "SMS: A Suggestive Modelling System for Object Recognition", presented at 1986 Alvey Computer Vision and Image Interpretation Meeting, also Dept. of Artificial Intelligence Research Paper 298, Univ. of Edinburgh, 1986.

[4] Brooks, R.A., "Symbolic reasoning among 3-D models and 2-D images", Artificial Intelligence, Vol 17, pp 285-348 , 1981.

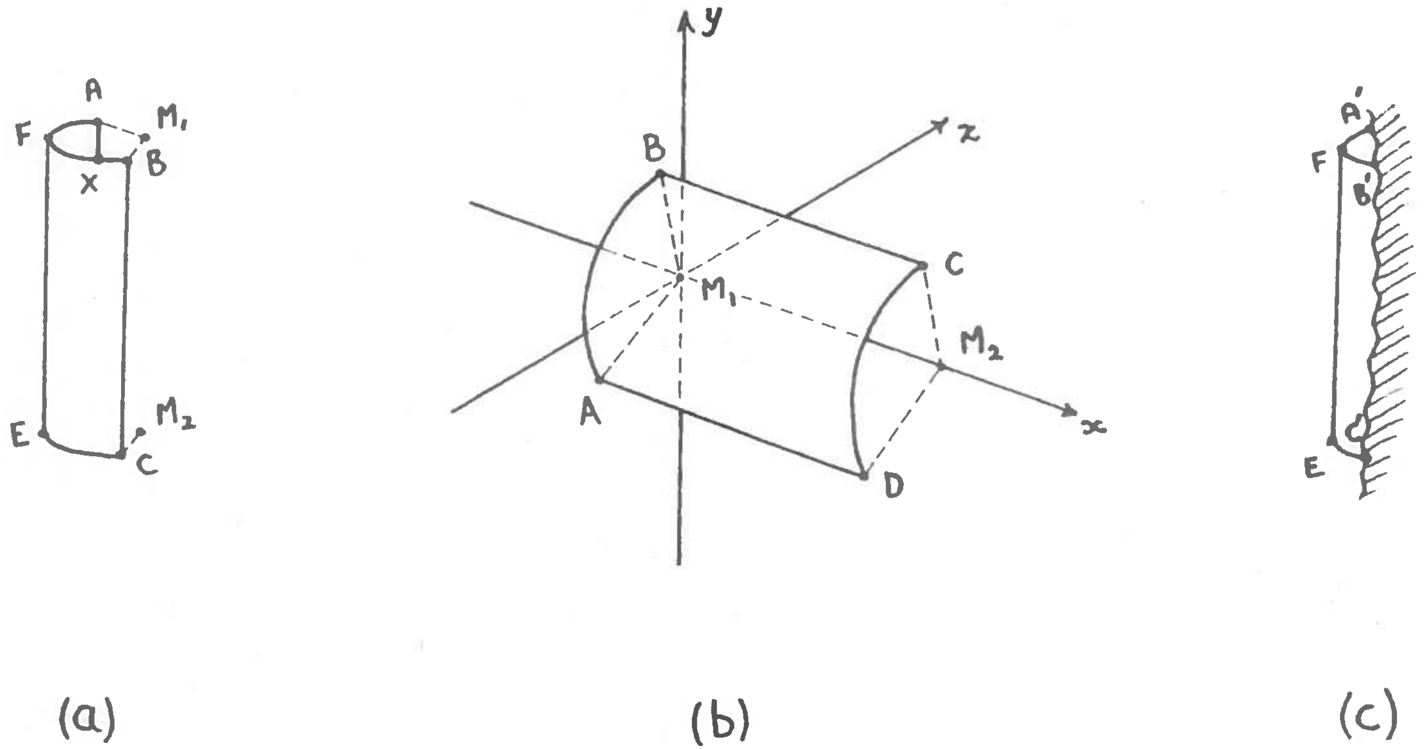
[5] Guttag, J.V., Horowitz, E. and Musser, D.R., "The design of data type specifications", in "Current trends in programming methodology", IV, (Ed. Yeh, R.), Prentice-Hall, 1978,

[6] Popplestone, R.J., Ambler, A.P. and Bellos, I.M., "An interpreter for a language describing assemblies", Artificial Intelligence, 14, 79, 1980.

[7] Marr, D., "Vision", W. H. Freeman, 1982.

[8] Bledsoe, W. W., "The sup-inf method in Presburger arithmetic", Memo ATP 18, Dept. of Math. and Comp. Sci., Univ. of Texas at Austin, 1974.

[9] Shostak, R. E., "On the sup-inf method for proving Presburger formulas", J. Assoc. Comput. Mach., 24, pp529-543, 1977.

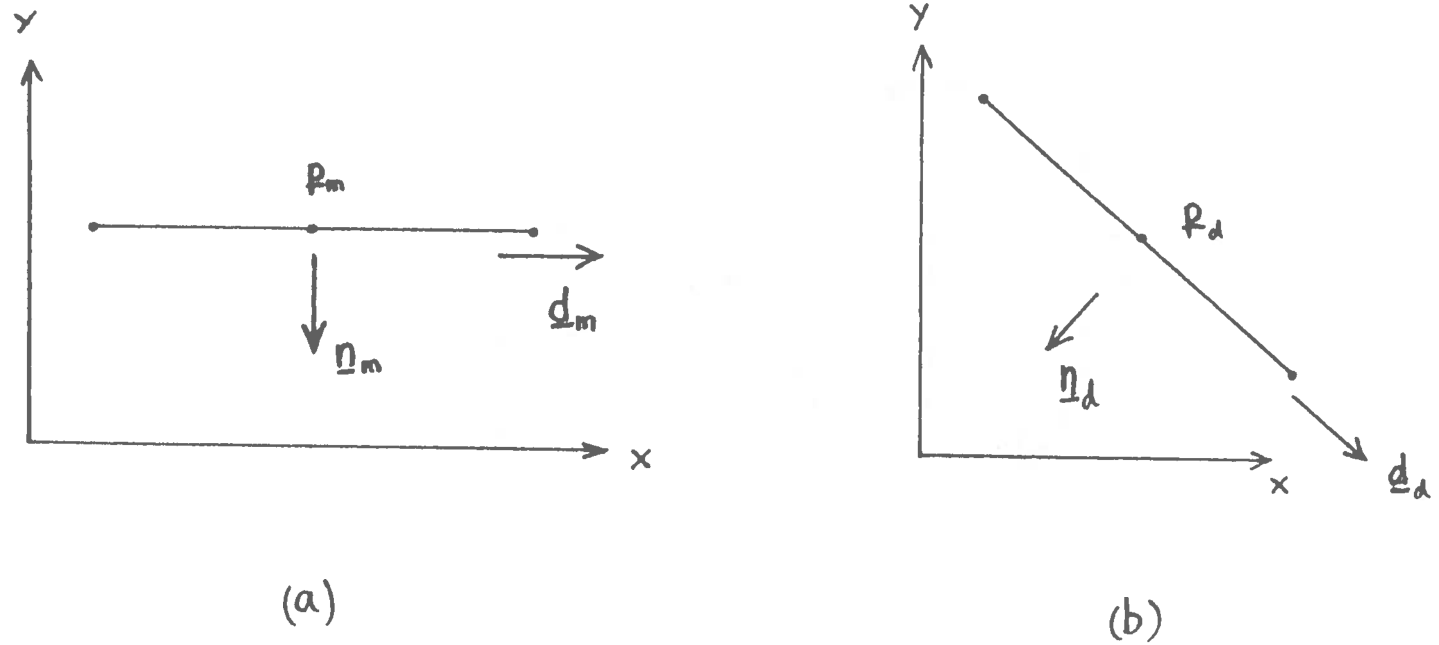


(a)

(b)

(c)

Figure 3-1


y
$\underline{p}_m$
$\underline{d}_m$
$\underline{n}_m$
x
y
$\underline{p}_d$
$\underline{n}_d$
x
$\underline{d}_d$

(a)

(b)

Figure 4-3

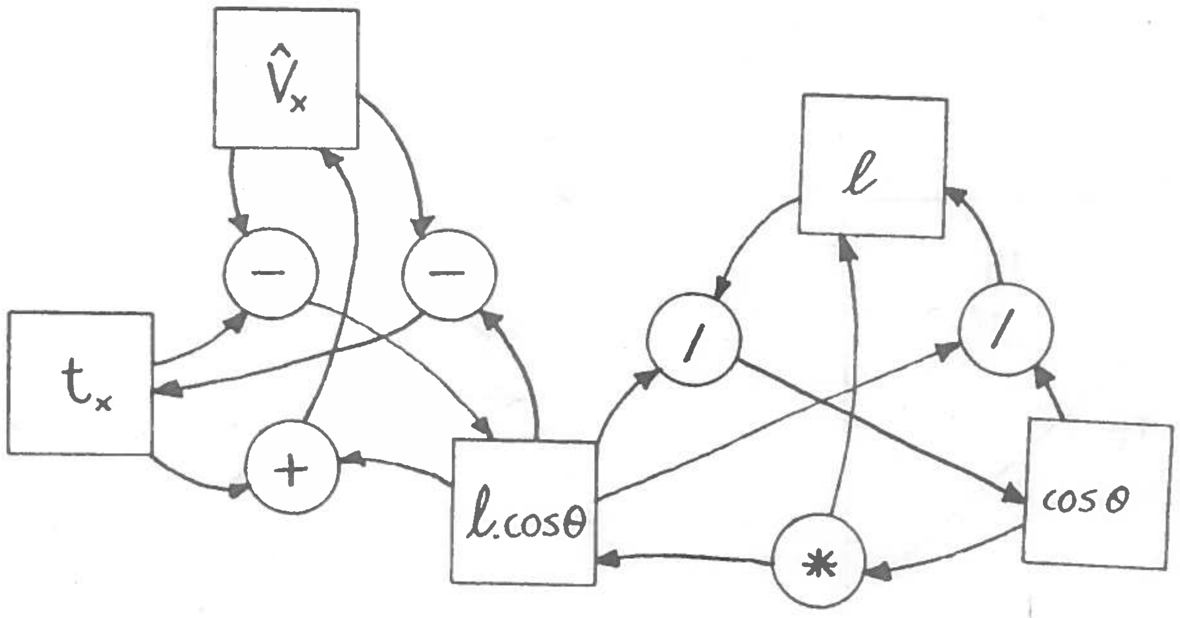


Figure 4-4

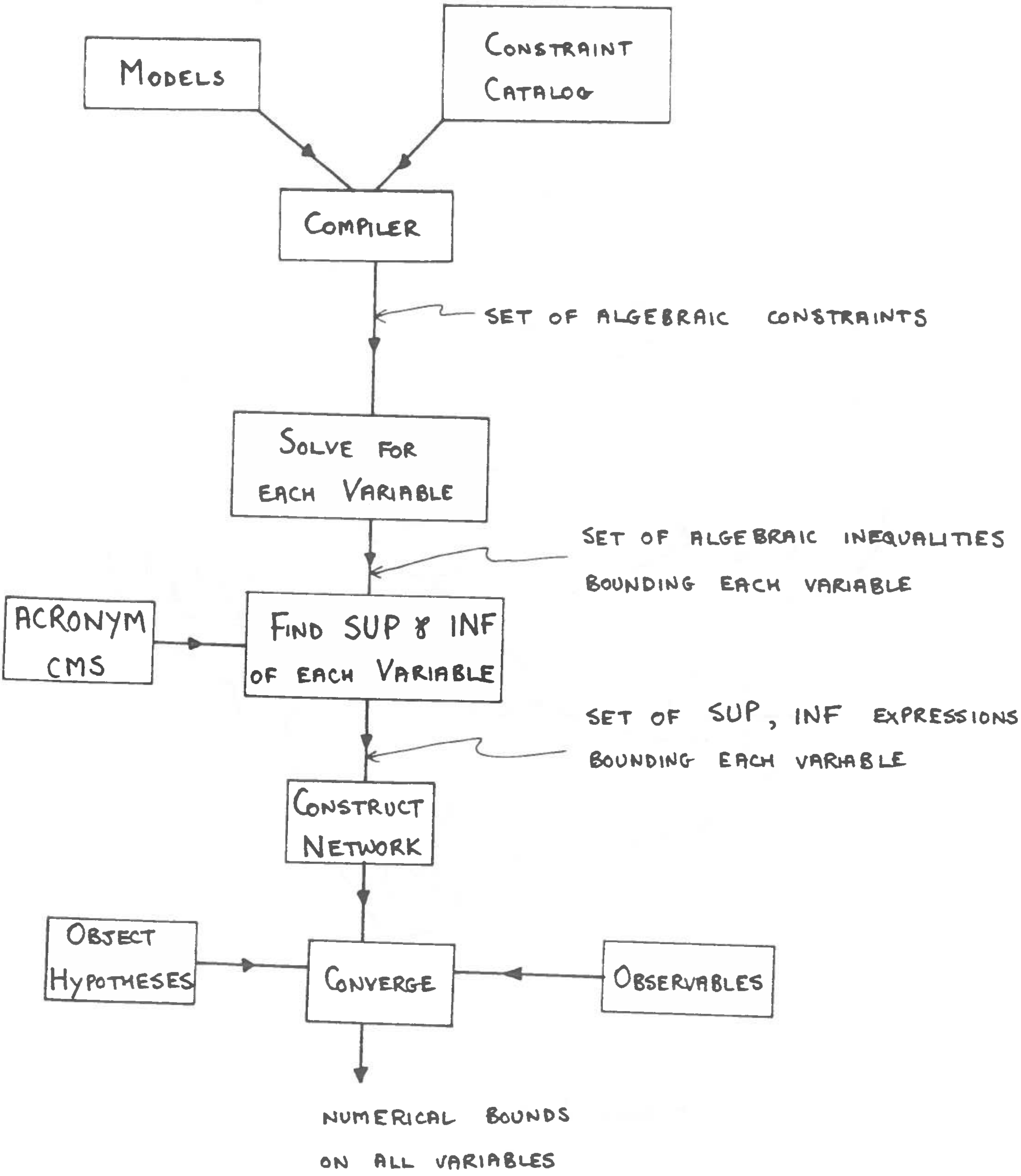

MODELS
CONSTRAINT CATALOG
COMPILER
SET OF ALGEBRAIC CONSTRAINTS
SOLVE FOR EACH VARIABLE
SET OF ALGEBRAIC INEQUALITIES BOUNDING EACH VARIABLE
ACRONYM CMS
FIND SUP & INF OF EACH VARIABLE
SET OF SUP, INF EXPRESSIONS BOUNDING EACH VARIABLE
CONSTRUCT NETWORK
OBJECT HYPOTHESES
CONVERGE
OBSERVABLES
NUMERICAL BOUNDS ON ALL VARIABLES

Figure 4-1

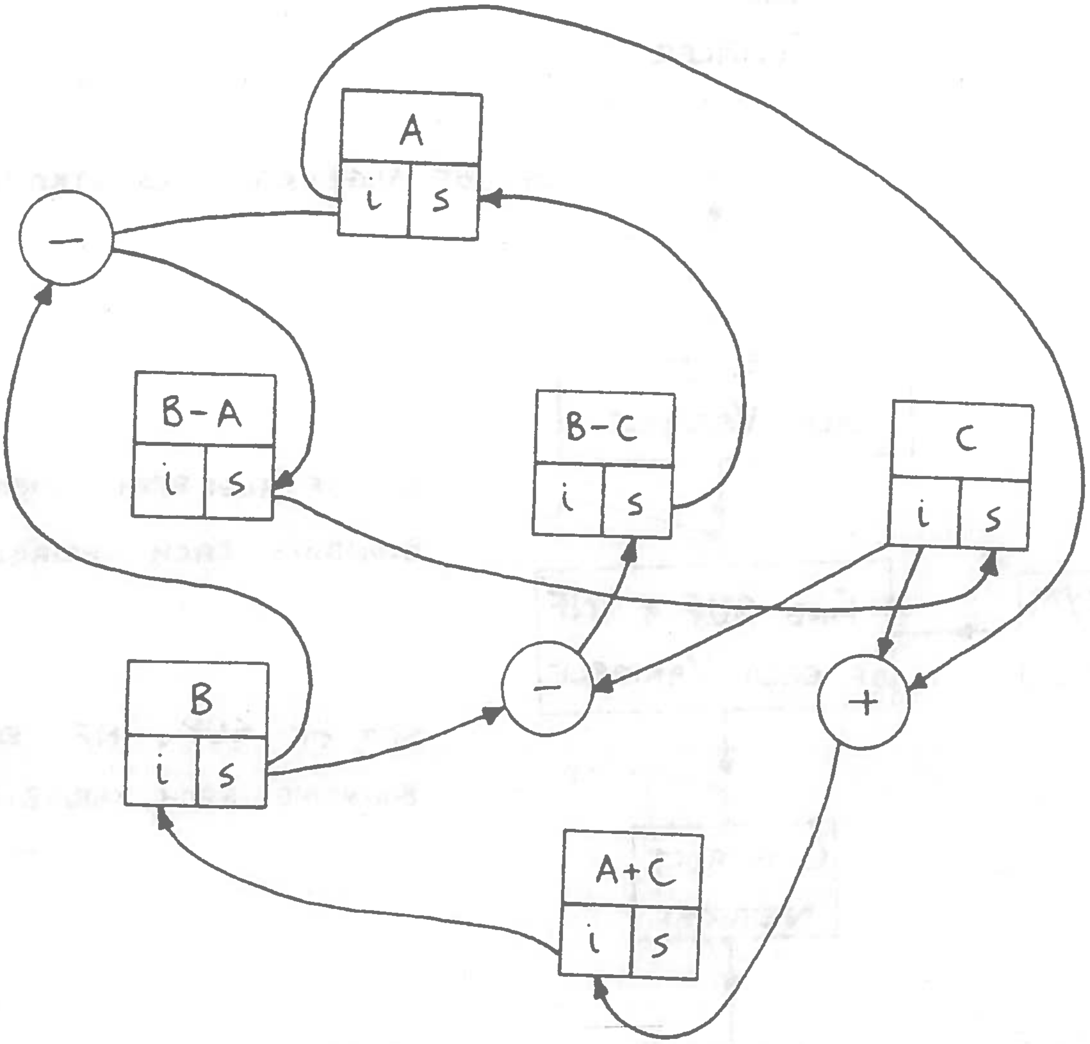


Figure 4-2