A Highly Expressive Language of Spatial Constraints

Ernest Davis*
Courant Institute
New York, New York

December 19, 1995

Abstract

AI applications require the representation and manipulation of partial spatial knowledge of many different kinds. This paper argues that a representation rich in primitives but fairly restricted in logical form will suffice for many of these purposes. We present and discuss one such representation language. We demonstrate that the language is expressive enough to capture exactly or closely approximate many of the representations that have been used in the AI literature. It also contains some original constructs for dealing with collections of regions of unknown cardinality.

1 Introduction

AI researchers working in spatial reasoning have developed many different kinds of representations that can express incomplete spatial information. However, for the most part, each of these representations have been developed in isolation and addresses a different problem. Little attention has been given to the question of how these different representations fit together. In this paper, we propose a representational framework capable of combining many different kinds of partial spatial information.

As is well known, reasoning with partial information, in spatial as well as other domains, is important for a number of reasons. First, full information may be impossible or impractical to obtain. Second, one may wish to reason about categories of scenarios, rather than a single scenario. Third, the object of reasoning may be a conjectural object not yet wholly defined, such as a mechanism under design. Fourth, complete and exact calculations may be computationally infeasible or fragile; reasoning with partial information may be either easier or more robust. (On the issue of robustness, see [Gelsey, 95].)

In developing a spatial reasoner, it is often tempting to require the use of a "picture-like"[1] representation, such as an occupancy array or a polygon with floating-point coordinates; that is, a representation that can easily be used to render a physical picture that depicts exactly the same spatial information, no more and no less. Clearly, however, if a task involves reasoning with partial information, then the use of picture-like representations will necessarily be indirect and awkward

*This research has been supported by NSF grant #IRI-9300446.

[1]More often called a "diagrammatic" representation. However, to judge by [Glasgow, Narayanan, and Chandrasekaran, 95] this latter term is used very broadly. It includes both external representations (physical pictures) and internal representations (data structures), and, in each of these categories, including representations of both spatial and non-spatial information. In what sense an data structure that expresses non-spatial information can be "diagrammatic" I confess I do not wholly understand. The term "analogical representation" is broader still; according to [Myers and Konolige, 95] it encompasses *any* data structure with a model-theoretic semantics.

[Hayes,95]. (The same is true, more generally, of the use of vivid representations for relational information [Davis, 91].) That is to say, the representational scheme used in a reasoning program should correspond to the information to be used — not, one would suppose, a particularly controversial claim.

Nonetheless, there persist surprisingly widespread misimpressions about the nature of non-picture-like spatial representations. In particular, Hayes' famous Naive Physics Manifesto [1979] and Ontology for Liquids [1985], and other studies along similar lines (e.g. [Davis, 88]) have been widely misread as advocating a spatial representation that consists purely of qualitative relations like "left-of(X, Y)" and "face-of(F, R)" combined in a predicate calculus syntax and manipulated using domain-independent logical inference rules. For example, David Waltz [1995] writes, in discussing the supposed[2] resistance of the AI community to diagrammatic representations, "It was widely believed [in the early eighties] that logic could successfully model images and scenes, even though the baroque improbability of that effort should have long been clear to everyone who read Pat Hayes' Naive Physics Manifesto." Since it is self-evident that practically any representation of spatial knowledge can be expressed in logical form, it is not quite clear what Waltz means here. I suspect that what he is denying is the claim that a spatial representation should consist exclusively of logical formulas similar in flavor to those found in the Naive Physics Manifesto. If so, this is a thoroughly straw man; neither Hayes nor, so far as I know, anyone else has ever claimed this. Nonetheless, this misconception reflects a real gap in this literature; namely, that Hayes does not say what a spatial representation for naive physical reasoning *should* look like. This paper here is a stab at that question.

For simplicity, this paper is confined to two-dimensional geometry. Three-dimensional representations are necessarily much more complicated, though I conjecture that the same flavor of representation will be suitable.

The development of our representation is guided by the following objectives:

1. The representation should have a clearly defined model-theoretic semantics. We will use the standard theory of the real plane, together with regions (sets of points), and finite collections of regions (section 2).
2. The representation should be able to express the many types of partial information that have been found particularly useful for commonsense spatial reasoning. I have not found any very well-defined principles for choosing categories of information to express; rather, I have been guided partly by the AI literature and partly by my own taste. Section 5 will illustrate, by example, some of the types of information expressible in our representation. Some of the gaps in expressive power will be discussed in section 6.1.
3. There should be a vocabulary of exact (picture-like) representations; clearly, when full information is available, it is often sensible to use it. We have chosen, somewhat arbitrarily, polygons and circles as primitives.
4. There should be a means of saying that the true shape of a scenario does not conform precisely to an idealized shape description, but that the description approximates it closely. This language provides as approximation measures the Hausdorff distance, the dual-Hausdorff distance, and the notion of approximation in tangent. (See section 3.3).
5. It should be possible to describe parts of a spatial scene with great precision and other parts much more vaguely. It should be possible to describe small-scale relations precisely and large-scale relations imprecisely.

[2]In view of the fact that other areas of computer science, such as graphics, CAD and even computational geometry, use almost exclusively picture-like representations for spatial information, and that, even in AI, spatial information has always been more often than not represented in picture-like form, it is hard to understand the defensive tone of this quotation and similar passages by other authors in [Glasgow, Narayanan, and Chandrasekaran, 95]

6. There should be a language to describe spatial patterns and sets of regions that does not require each component to be enumerated individually. For instance, it should be possible to describe spatially the legs on a centipede or the grains of sand in a bucket without enumerating each particular leg or each particular grain of sand. This aspect of our representation is, I believe, new in the AI literature.

7. The syntax of the language should be simple and systematic.

8. Subject to the above constraints, the representation should be as *inexpressive* as possible. That is, we wish to be able to represent all the types of information indicated in (2) and (6), and to achieve all the flexibility indicated in (3), (4), and (5), but we have no wish to be able to express spatial categories that fall outside these, and such superfluous expressive power will only hurt us in devising effective inference techniques. Our language will be mostly a system of constraints — atomic ground formulas, without disjunction, negation, or quantification — augmented with a limited ability to quantify over the regions in a collection. (Section 3)

One objective conspicuously missing from the above list is the requirement that the representation support effective inference techniques, except insofar as that requirement is reflected in (8). Section 7 will briefly discuss a possible architecture for an inference system over this language — basically, a large collection of forward inference rules — but we have not yet developed such a system. It is easily shown (section 6.2) that the problem of inference over our representation is not fully decidable. I believe that this is unavoidable; that any spatial representation expressive enough for flexible automated reasoning will be intractable in the worst case, and that any such spatial reasoner will have to make do with reasoning that is not always complete. This should not be a catastrophe.

2 Ontology

An entity in our ontology belongs to one of the following sorts:

- A *real number.* E.g. 2.67; 7; $\sqrt{2}+\sqrt{3}$. The class of *integers* is a subsort of the class of reals.
- A *point* in the plane.
- A *length.* E.g. 5 feet.
- An *angle.* E.g. 30°.
- A *direction.* E.g. North; 25° east of north.
- A measure of *area.* E.g. 4.1 square cm.
- A *vector.*
- A orthonormal *frame of reference*, a triple consisting of a point, a vector, and a boolean (the origin, the unit vector in the x direction, and a flag to distinguish right-handed from left-handed coordinate systems.)
- An entity of sort "*intervals*" is the union of finitely many intervals of length. E.g. the closed interval [1 in, 2.4in]; the union of two open intervals (3 foot, 5 foot) ∪ (8.1 foot, 8.2 foot).
- An entity of sort *arcs* is the union of finitely many arcs of directions. E.g. the open interval (N, NNW); the union of two intervals (N+5°, N+12°) ∪ (N+50.5°, N+92.7°).

- A *region* in the plane, defined to be a semi-algebraic set of points (see section 2.1). E.g. Relative to some particular frame of reference, the closed disc $X^2 + Y^2 \leq 25$; the open arc of parabola $Y = (3X - 8)^2 \wedge 3.0 < X < 5.0$; the union of these two regions. There are two non-exhaustive subsorts of region: A *regular* region is bounded and equal to the closure of its interior; a *curve* has an empty interior.
- A *directed curve*. See section 2.2.
- A *scale mapping*.
- A finite *collection* of regions.

2.1 Regions

Regions of the plane are our central ontological category; most of the descriptions that we will look at will be descriptions of some number of individuated regions. A region is a set of points in the plane; however, not all sets of points can have any possible physical significance. For example, the set of all points with rational coordinates in some fixed reference frame is not a set that is likely ever to arise in practical applications. Ideally, therefore, we would like to admit as "regions" exactly those point sets that could be significant, and no others. Our current approximation of this category is the set of semi-algebraic point sets. We will first define this set; then enumerate some of its features; then some of its weaknesses.

Definition: An *algebraic constraint* in the plane is an inequality either of the form $p(x, y) > 0$ or of the form $p(x, y) \geq 0$, where $p(x, y)$ is a polynomial in x and y with *real* coordinates. (Note that we are using real coordinates rather than integer coordinates.) A set of points $S \subset \Re^2$ is *semi-algebraic* if it can be expressed as the Boolean combination of algebraic constraints.

Features of the class of algebraic sets include the following [Mishra, 93]:

1. The class is closed under Boolean operations; under algebraic mappings, particularly projection; and under the operations of taking the closure, the interior, and the boundary of the set.
2. A semi-algebraic set has only finitely many connected components. In particular, a one-dimensional semi-algebraic set is the union of finitely many intervals. Thus, infinitary Zeno-like paradoxes [Davis, 92] can often be avoided by restricting all sets under consideration to be semi-algebraic.
3. The boundary of a semi-algebraic set is a curve (empty interior). Equivalently, there are no cases where a semi-algebraic set S and its complement are both dense over an open set U.
4. A semi-algebraic curve is infinitely differentiable everywhere except at finitely many points.

The class also has weaknesses:

1. Physical reasoning involves transcendental shapes and functions: the catenary, the helix, uniform rotational motion, harmonic oscillation, and so on. One possible way to include these would be to use instead the class of bounded *semi-analytic* regions; those defined by as the Boolean combination of inequalities of the form $f(x, y) > 0$ or $f(x, y) \geq 0$ where $f(x, y)$ is an analytic function. (An analytic function is one that is equal to the sum of a power series.) I conjecture that the class of bounded semi-analytic regions satisfies properties (1) – (4) above, but I have not found a proof of it.

2. The class of semi-algebraic regions admits closed sets, such as the closed disk $X^2 + Y^2 \leq 1$; open sets, such as the open disk $X^2 + Y^2 < 1$; and part closed / part open sets such as the semi-disk $X^2 + Y^2 \leq 1 \wedge Y > 0$. This is the classic example of a topological distinction without a physical difference that gives rise to indecorous snickering among those who oppose the use of the classical real line as an ontology for automated reasoning.

   I had originally hoped to do everything using exclusively closed sets, and thus avoid this anomaly. However, that approach breaks down with representations that use the sign calculus, such as those of NEWTON [de Kleer, 77] (see section 5.5). In the sign calculus, and similar calculi, the sign 0 is a closed set, while the signs + and − are open sets. Once one admits open sets in one place in the theory, it is very hard to exclude them anywhere else.

   Representationally, this redundancy should be at worst irritating. One can posit that all shapes of physical objects are regular; thus, the including of non-regular shapes need not affect the physical theory. How much it will complicate inference, I don't know. In the one-dimensional case, allowing open, closed, and half-open intervals in the representation costs only a perpetual tedious division of all computations into two or four cases. I should hope that something analogous holds in higher dimensions; but I can't be sure of that.

It should be kept in mind that using the set of semi-algebraic regions does not have any physical implications; we are not saying that nature, for some reason, prefers semi-algebraic regions. Physical objects of terrestrial kinds are made of atoms, so their shape, ultimately, is the union of spheres (?) of roughly one Angstrom diameter. (Our language, in fact, allows this as a description, if it is desired.) Rather, the class of semi-algebraic sets is the class of mathematical abstractions that we are using to approximate shape.

2.2 Directed Curves

Definition: A *directed curve* is $c(t)$ is a function from a real interval I to the plane such that

- $c(t)$ is continuous.
- $c(t)$ is piecewise C^∞.
- If U is an open subset of I, then $c(t)$ is not constant over U.

Thus, at any value t_0 in the interior of I, the derivative $\dot{c}(t)$ approaches a non-zero limit as t approaches t_0 from below, and (either the same or a different) non-zero limit as t approaches t_0 from above. The limit of the unit vector $\dot{c}(t)/ \mid \dot{c}(t) \mid$ as t approaches t_0 from below is the *incoming tangent* to c at t_0; the limit of the same unit vector as t approaches t_0 from above is the *outgoing* tangent.

Any C^∞ strictly monotonic reparametrization of a directed curve gives a directed curve with the same tangent field. Intuitively, we can consider these as all the "same" directed curve, though this identification will not be needed formally in our theory. Likewise, in the case of directed curves that are intuitively cyclic, a cyclic reparamerization gives intuitively the "same" directed curve.

Definition: The *reverse* of the directed curve $c(t)$ over interval I is the curve $c(-t)$ over interval $-I$.

Definition: Let $\mathbf{R}$ be a regular region. The *outside* of $\mathbf{R}$ is the (unique) unbounded connected component of the complement of $\mathbf{R}$. An *interior hole* of $\mathbf{R}$ is a bounded connected component of the complement of $\mathbf{R}$. The *inside* of $\mathbf{R}$ is the complement of the outside of $\mathbf{R}$; thus the union of $\mathbf{R}$ with all interior holes of $\mathbf{R}$. The *undirected outer boundary* of $\mathbf{R}$ is the intersection of $\mathbf{R}$ with the closure of the outside of $\mathbf{R}$. A directed curve $c(t)$ is an *outer directed boundary* of $\mathbf{R}$ if c lies in the outer undirected boundary and c is a maximal simple closed curve with a counter-clockwise orientiation. A directed curve $c(t)$ is an *inner directed boundary* of $\mathbf{R}$ if it is the reverse of an outer directed boundary of some interior hole of $\mathbf{R}$.

Clearly, every connected regular region has an outer directed boundary that is unique up to the cyclic and monotonic reparametrizations discussed above; and has an inner directed boundary for each interior hole that is unique in the same sense.

2.3 Frames of reference

We use orthonormal frames of reference, which may be either right-handed or left-handed. Frames of reference are handy so that one can use coordinates to name points in terms of real numbers. Multiple frames of reference are useful in cases where localized information is precise than global information. Finally, frames of reference are used to define scale mappings.

3 Language

As discussed in section 1, our goal in constructing a spatial language is be able to express the kinds of partial information that arise in practice, but, subject to that constraint, to keep the language as limited as possible. After all, we could achieve maximal expressivity by allowing arbitrary sentences about arbitrary set-theoretic constructions over sets of points. Such a language can (presumably) express any geometric concept capable of being given a model-theoretic semantics; but it is, we believe, completely impractical.

In order to limit the scope of the language, we have severely restricted the range of logical forms are allowed. This restriction is somewhat complicated, and is most naturally viewed in two parts: the language over sorts other than collections of regions, which we will call the "pure" language, and the language including collections of regions, which we call the "extended" language.

3.1 The pure language

The "pure" language is a pure constraint language, without quantifiers, negation, or disjunction. That is, it consists entirely of atomic ground formulas, assertions of the form "$p(t_1 \ldots t_n)$" where t_i are variable-free terms. For instance, the sentence

area(polygon(P1,P2,P3,P4)) < area(R)

which has predicate "<" and arguments "area(polygon(P1,P2,P3,P4))" and "area(R)", is an atomic constraint over the constants P1, P2, P3, P4 of sort *point* and R of sort *region.* Such a sentence is allowed in our language. However, our language does not permit sentences of the following kind:

L1 $<$ L2 $\vee$ L1 $>$ 2 $\cdot$ L2. (No disjunction allowed.)
$\neg$smooth(C). (No negation allowed.)
$\forall_{X \in R1} \exists_{Y \in R2}$ direction$(X - Y) = D$. (No quantifiers allowed.)

The representation consists of *descriptions* and *definitions*. Descriptions are sets of constraints over the geometric objects of interest. Definitions allow non-primitive constants, predicates, and functions to be defined in terms of primitive symbols. Predicate definitions consist of a name with formal parameters, local variables, constraints over the parameters and local variables. Function definitions consist of a name with formal paramters and an expresion over the parameters to be evaluated. Constant definitions consist of a name and a constant value.

To improve readability, we use a PASCAL-like syntax. However, it is important to keep in mind that this is a constraint language, not an imperative language. In particular, equality is true equality, not assignment and the order of constraints in a body is irrelevant. Thus, for example, the two statements

X = Y−1.
Y = 2X.

does not have the effect of changing the final value of Y to be twice its original value minus two. Rather, it is a pair of constraints that restricts Y to be 2 and X to be 1.

Here is a sample description of an isoceles triangle with a circular hole inside.

description example1
P1, P2, P3, PC : *point*; RAD : *length*; TRIANGLE, CIRCLE, DIFF: *regular*.
distance(P1,P2) = distance(P1,P3).
TRIANGLE = polygon(P1,P2,P3).
CIRCLE = circle(PC,RAD).
CIRCLE ⊂ TRIANGLE.
DIFF = TRIANGLE − CIRCLE.
return DIFF.

The "**return**" statement in the description does not affect the semantics; it merely serves to distinguish the key figure of the description from those figures just used for construction.

Here is the definition of the predicate "fits-in(R1,R2)" meaning that R1 could fit inside region R2.

predicate fits-in(R1,R2 : *region*).
M : *mapping*; R1A : *region*.
rigid-mapping(M).
R1A = apply(M,R1).
R1A ⊂ R2.
end.

Note that the local variables are, in effect, existentially quantified. This definition can be translated into the logical rule,

$$\text{fits-in(R1,R2)} \Leftrightarrow$$
$$\exists_{M,R1A}\ \text{rigid-mapping(M)} \wedge \text{R1A} = \text{apply(M,R1)} \wedge \text{R1A} \subset \text{R2}.$$

Functions and constants can be defined as abbreviations of complex terms. Here is the definition of function "midpoint(P1,P2)."

function midpoint(P1, P2 : *point*)
return P1 + 0.5 · (P2 − P1).

Predicate and function definitions are not allowed to be recursive, either directly or indirectly. Therefore, they are, in this part of the language, merely abbreviations; any description that uses defined symbols can be rewritten purely in terms of primitive symbols by expanding out the definitions and renaming variables to avoid conflict. Thus any description in the pure language is logically equivalent to a long conjuction of atomic formulas under the scope of an existential quantifier.

3.2 The extended language

The structure of the pure language is based on the hope that, given a suitably rich set of geometric primitives, one can get away without ever having to use universal quantification over points or regions explicitly. In dealing with collections of regions, however, most of the facts one wants to assert are statements like, "All of the regions in the collection are triangles," "Each of the regions will fit in a circle of radius 1," "No two of the regions overlap," and such. Adding primitives such as "all-triangles(C)" is clearly not a reasonable approach. One really needs explicit quantification over the regions in a collection.

To accommodate this need, we introduce the category of an *extended* predicate or description. An extended description may contain a statement of the form "$\forall_{R1 \ldots Rk \in C} \; \alpha(R1 \ldots Rk)$" where C is a term of sort *collection,* and α is a constraint in the pure language. Thus, for example, the following is a description of a collection of regions all of which fit in a circle of radius 1, and no two of which overlap.

extended description col1
C : *collection*; CIRCLE : *regular*; O : *point.*
C = circle(O,inch).
∀ (R ∈ C) fits-in(R,CIRCLE).
∀ (R1,R2 ∈ C) non-overlap(R1,R2).
end.

A definition that refers to an extended predicate or is likewise considered extended.

The restriction that the formula inside the scope of the quantifier be pure has the effect of limiting the depth of quantifier alternation. As remarked above, a defined pure predicate can be expanded into a formula with only primitive symbols and existentially quantified variables. If such a predicate is used within the scope of a universal quantifier in an extended definition, then this existential quantification takes place within the scope of a universal quantifier; but no deeper alternation is possible. Thus, for example, the above formula, "∀ (R ∈ C) fits-in(R,CIRCLE)" expands into the formula

$$\forall\ (\mathrm{R} \in \mathrm{C})\ \exists\ (\mathrm{M},\mathrm{R1A})\ \text{rigid-mapping}(\mathrm{M}) \wedge \mathrm{R1A} = \text{apply}(\mathrm{M},\mathrm{R}) \wedge \mathrm{R1A} \subset \mathrm{CIRCLE}.$$

Thus, any description in the extended language can be expanded into a conjunction of statements, each of which is either

a. an atomic ground formula over the language primitives; or

b. a statement of the form, "$\forall_{R1 \ldots Rk \in C} \exists_{V1 \ldots Vm} \alpha_1 \wedge \ldots \wedge \alpha_n$" where C is a term of sort *collection*, and each of $\alpha_1 \ldots \alpha_m$ is an atomic formula over the language primitives with free variables $R1 \ldots Rk, V1 \ldots Vm$.

3.3 Primitives

In order to attain the desired high level of expressivity within the above constraints on logical form, we need a rich collection of spatial primitives. These are enumerated below. No attempt has been made to reduce this to a minimal collection (for example, set intersection can be defined in terms of set difference.) All primitives are strictly sorted, though some overloading of common symbols (e.g. $\cup$, $-$) is allowed. Most primitives are prefix, but some common symbols are infix. The closed interval constructor [X,Y] is outfix. The meaning of a primitive is explained when it is not completely obvious from its name.

We categorize the primitives by the most "complex" sort of entity that they involve. The primitives on regions and directed curves, which are numerous, are further subdivided.

3.3.1 Set theoretic

(The sort *set* below ranges over *intervals, arcs,* and *region*; the sort *elt* ranges corresponding over *length, direction,* and *point.*)

Constant: $\emptyset$.

Functions:
$set \cup set \rightarrow set.$
$set \cap set \rightarrow set.$
$set - set \rightarrow set.$

Predicates:
$elt \in set.$
$set \subset set.$

3.3.2 Reals

Constants: Numerals and floating point numbers.

Functions:
floor(*real*) $\rightarrow$ *int.*
abs(*real*) $\rightarrow$ *real.*

Predicates:
$real < real.$
$real \leq real.$

3.3.3 Elements of $\Re^n$: Points, vectors, lengths, angles, directions, and frames

Constants:
π : *angle.*
up : *direction.*

Functions:

point + *vector* $\rightarrow$ *point*.
vector + *vector* $\rightarrow$ *vector*.
real $\cdot$ *vector* $\rightarrow$ *vector*.
length(*vector*) $\rightarrow$ *length*.
length + *length* $\rightarrow$ *length*
real $\cdot$ *length* $\rightarrow$ *length*.
area + *area* $\rightarrow$ *area*
real $\cdot$ *area* $\rightarrow$ *area*.
angle + *angle* $\rightarrow$ *angle*.
real $\cdot$ *angle* $\rightarrow$ *angle*
direction + *angle* $\rightarrow$ *direction*.
direction(*vector*) $\rightarrow$ *direction*.
distance(*point, point*) $\rightarrow$ *length*.
frame(*point, vector, integer*) $\rightarrow$ *frame*.
% frame(O,V,S) denotes the frame of reference where O is the origin,
% V is the unit x vector, and S is ± 1 for right/left-handedness.
coordinate(*frame, real, real*) $\rightarrow$ *point*.

Predicates:
length < *length*.
length $\leq$ *length*.
cyc-order(*direction, direction ... direction*).
% cyc-order(D1,D2, ... Dk) if these occur in strictly counter-clockwise order.
weak-cyc-order(*direction, direction, ... direction*).
% weak-cyc-order(D1,D2 ... Dk) if these occur in non-strictly counter-clockwise order.

We will also use X$-$Y as the inverse of X+Y and X/Y as the inverse of X$\cdot$Y.

3.3.4 Intervals and arcs

Constant: unit_circle : *arcs*;
% The set of all directions.
pos : *intervals*;
neg : *intervals*;
% The intervals of all positive and all negative lengths, respectively.
Functions:
sign(*length*) $\rightarrow$ intervals.
% sign(L) is either pos, neg, or [0,0], as L is positive, negative, or 0.
[*length, length*] $\rightarrow$ *intervals*.
[*direction, direction*] $\rightarrow$ *arcs*.
% [L,M] is the closed interval or arc from L to M.

3.3.5 Regions and directed curves

Primitive regions

Constant: space. % All of space.
Functions:
polygon(*point, point ... point*) $\rightarrow$ *regular*.

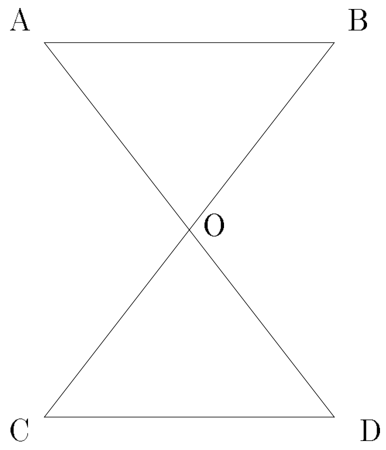

Figure 1: Self-intersecting polygon

% polygon(P1, P2, ... Pk) is the closure of the set of all points encircled at least once
% by the cyclic path P1 $\rightarrow$ P2 $\rightarrow \ldots \rightarrow$ Pk $\rightarrow$ P1,
% For instance, in figure 1, polygon(A,B,C,D) is the union of the two triangles ABO and DCO.

circle(*point, length*) $\rightarrow$ *regular*.
% circle(O,R) is the closed disk of center O and radius R.

Topological

Functions:
interior(*region*) $\rightarrow$ *region*.
undirect(*dcurve*) $\rightarrow$ *curve*.
% undirect(D) is the curve which is the image of directed curve D.
% (Intuitively, D with the arrows removed.)
reverse(*dcurve*) $\rightarrow$ *dcurve*.
% reverse(D) is D with the arrows reversed.

Predicates:
homeomorphic(*region, region*).
oboundary(*regular,dcurve*).
% oboundary(R,D) if directed curve D is a outer boundary of R with positive orientation.

Assembling directed curves

Functions:
polyline(*point, point ... point*) $\rightarrow$ *dcurve*.
% polyline(P1 ... Pk) is the directed curve that moves on lines
% from P1 to P2, from P2 to P3 ... up to Pk.
start-dcurve(*dcurve*) $\rightarrow$ *point*.
end-dcurve(*dcurve*) $\rightarrow$ *point*.
interior-dc(*dcurve*) $\rightarrow$ *dcurve*.
% The result of taking the end-points off a directed curve.

Predicates:

join(*dcurve, dcurve ... dcurve*).

% join(D1, D2 ... Dk, Djoin) holds if Djoin is defined over the union of the domains
% of D1 ... Dk, and Djoin(T) = Di(T) for any T in the domain of Di.

dcurve-seg(*dcurve, dcurve, point, point*).

% dcurve-seg(DCS, DC, A, B) if DCS is a segment of DC from point A to point B.

Derivatives

Functions:

in-tangent(*dcurve, point*) → *direction.*

% in-tangent(D,P) is the direction of the tangent to directed curve D coming into point P.

out-tangent(*dcurve, point*) → *direction.*

% out-tangent(D,P) is the direction of the tangent to directed curve D going out of P.

tangent-space(*dcurve*) → *darcs.*

% tangent-space(D) is the set of all tangent directions to directed curve D.

curvature(*dcurve, point*) → *length*

% curvature(D,P) is the signed radius of curvature of directed curve D at point P:
% positive if D is curving counterclockwise; negative if clockwise; undefined if D is not smooth at P.

curve-space(*dcurve*) → *interval*

% curve-space(D) is the set of all signed radii of curvature to D at points where D is smooth.
% undefined if D is not everywhere smooth.
% tangent-space and curve-space are rather kludgy ways around the fact that
% one is not allowed to quantify over the points on D.

Predicate:

smooth(*curve*).

Approximation

Since the shapes of real objects are rarely geometrically perfect, it is important to have measures of the accuracy with which a nominal shape approximates an actual shape. We give three such measures below, which we have found to be significant in physical reasoning [Davis, 95]. The Hausdorff distance from region A to region B is defined as the maximum of two quantities:

(i) the maximum over all points P in A of the distance from P to B;
(ii) the maximum over all points P in B of the distance from P to A.

The dual-Hausdorff distance from A to B is the maximum of the Hausdorff distance from A to B and the Hausdorff distance from the complement of A to the complement of B. The predicate "approx-in-tangent(A,B,δ, ϕ)" holds if there is a homeomorphism Γ from A to B such that (i) for all P in A, the distance from Γ(P) to P is less than δ; (ii) for all points P on the boundary of A, the normal to A at P is within angle ϕ of the normal to B at Γ(P). See [Davis,95] for further explanation.

Functions:

hausdoff(*region, region*) → *length.*

dual-hausdorff(*region, region*) → *length.*

Predicate:

approx-in-tangent(*region, region, length, angle*).

Other

Functions:

arc-length(*dcurve*) $\rightarrow$ *length.*

area(*region*) $\rightarrow$ *area.*

convex-hull(*region*) $\rightarrow$ *region.*

distance(*region, region*) $\rightarrow$ *length.*

3.4 Mappings

Functions

apply(*mapping, sort*) $\rightarrow$ *sort.*

% A scale mapping can be applied to any of our sorts.

linear(*frame, frame*) $\rightarrow$ *mapping.*

% linear(F1,F2) denotes the mapping M that maps the point coordinate(F1,X,Y) to
% coordinate(F2,X,Y).

3.5 Collections

Functions:

collection(*region, region ... region*).

% collection(R1 ... Rk) = { R1 ... Rk }.

connected_components(*region*) $\rightarrow$ *collection.*

union_over(*collection*) $\rightarrow$ *region.*

cardinality(*collection*) $\rightarrow$ *integer.*

collection-within(*collection, region*) $\rightarrow$ *collection.*

% collection-within(C1,R1) = { RC $\in$ C | RC $\subset$ R1 }.

collection-slice(*collection, region*) $\rightarrow$ *collection.*

% collection-slice(C,R1) = { RC $\cap$ R1 | RC $\in$ C }.

pattern(*region, mapping, integer, mapping, integer ...*) $\rightarrow$ *collection.*

% The pattern function generates regular textures of regions.

% pattern($R, M_1, N_1 \ldots M_k, N_k$) is the collection of all regions of the form
% $M_k^{P_k} \ldots M_2^{P_2} M_1^{P_1} R$, where $0 \leq P_j \leq N_j - 1$ for $j = 1 \ldots k$. See section 5.11 for an example.

4 Syntax and Semantics

The syntax of our language is defined by the BNF below. The primitive categories of symbols are "const", "func", "variable", "numeral" (a floating point or integer in standard format), "p-ident" (identifier for a "pure" entity), "e-ident" (identifier for an extended entity; that is, a collection which is being quantified over), "p-pred", "e-pred" (pure and extended predicates), and "sort". The top-level syntactic category is "definition". It is easily seen that this language is in fact regular.

identifier ::= p-ident | e-ident

term :: = const | identifier | variable | numeral | func(term ... term).

p-constraint ::= p-pred(term ... term). | term = term. | term $\neq$ term.

(Equality and inequality are allowed only on terms of the same sort.)

col-quant ::= $\forall$ (variable : *region* $\in$ e-ident) p-constraint.

constraint ::= p-constraint | e-pred(term ... term) | col-quant.

p-declaration ::= p-ident {, p-ident}* : sort

declaration ::= p-declaration | e-ident {, e-ident}* : *collection*;

p-declarations ::= ϵ | p-declaration {; p-declaration}*

declarations ::= ϵ | declaration {; declaration}*

constant_defn ::= **constant** const : sort; **return** term .

function_defn ::= **function** p-func(p-declarations) $\rightarrow$ sort; **return** term .

p-predicate_defn ::= **predicate** p-pred(p-declarations) p-declarations . p-constraint$^+$ **end**

e-predicate_defn ::= **extended predicate** e-pred(declarations) declarations . constraint$^+$ **end**

description ::= **description** name declarations . { constraints }* **return** term$^+$

definition ::= description | e-predicate_defn | p-predicate_defn | function_defn | constant_defn

Predicate, function, and constant definitions may not be indirectly or directly recursive.

The formal semantics is straightforward. All descriptions and definitions in the language can be translated into a set of first-order logic sentences, as sketched above (the formal definition is straightforward and uninteresting.) The first-order theory is then interpreted via a Tarskian semantics.

5 Examples

In this section, we will give examples to show how this language can be used to express many different kinds of partial spatial information.

5.1 Derived predicates

We begin by defining a number of useful functions and predicates. These are mostly self-explanatory.

function interior(R : *region*) $\rightarrow$ *region*.
return space $-$ closure(space $-$ R)

function boundary(R : *region*) $\rightarrow$ *curve*
return closure(R) $-$ interior(R).

predicate overlap(R1, R2 : *region*)
interior(R1) $\cap$ interior(R2) $\neq \emptyset$. **end**.

predicate non-overlap(R1, R2 : *region*)
interior(R1) $\cap$ interior(R2) $= \emptyset$. **end**

function normalize(R: *region*) $\rightarrow$ region.
return closure(interior(R)).

function norm-diff(R1,R2 : *region*) $\rightarrow$ region.
return normalize(R1 $-$ R2).

predicate connected(R : *region*)
cardinality(connected-components(R)) $= 1$. **end**

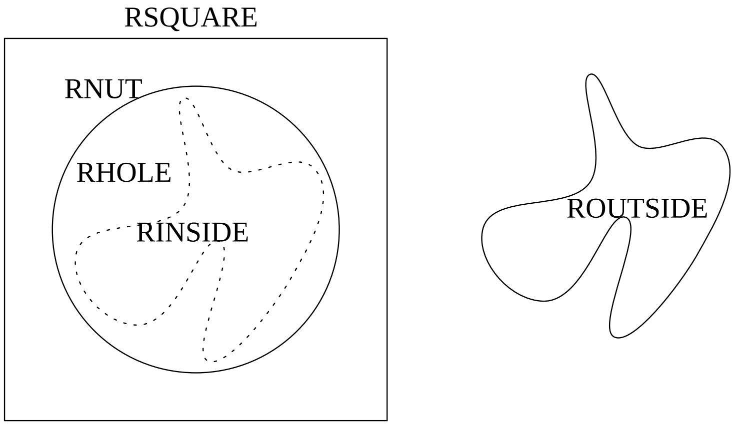

Figure 2: Nut and object: scene5.3

predicate rigid-mapping(M : mapping)
V1, V2 : *vector*; O1, O2: *point*.
length(V1) = length(V2).
M = linear-mapping(frame(O1,V1,1), frame(O2,V2,1)).
end

constant right : *direction*. **return** up $-$ $\pi/2$.

constant left : *direction*. **return** up + $\pi/2$.

constant down: *direction*. **return** up + π.

5.2

A square of size between 1 and 2 inches.

description scene5.2
F: *frame*; O : *point*; V : *vector*.
inch $\leq$ length(V) $\leq$ 2 $\cdot$ inch.
F = frame(O,V,1).
return polygon(coordinate(F, 0.0, 0.0), coordinate(F, 1.0, 0.0),
coordinate(F, 1.0, 1.0), coordinate(F, 0.0, 1.0)).

5.3

A nut consisting of a square of size between 1 and 2 inches minus a circular hole inside with a second connected object that would fit inside the hole but is currently outside the hole.

description: scene5.3
F: *frame*; OC : *point*;
RSQUARE, RHOLE, RNUT, RINSIDE, ROUTSIDE : *region*;
L2 : *length*; M : *mapping*.

inch $\leq$ length(V) $\leq$ 2 $\cdot$ inch.

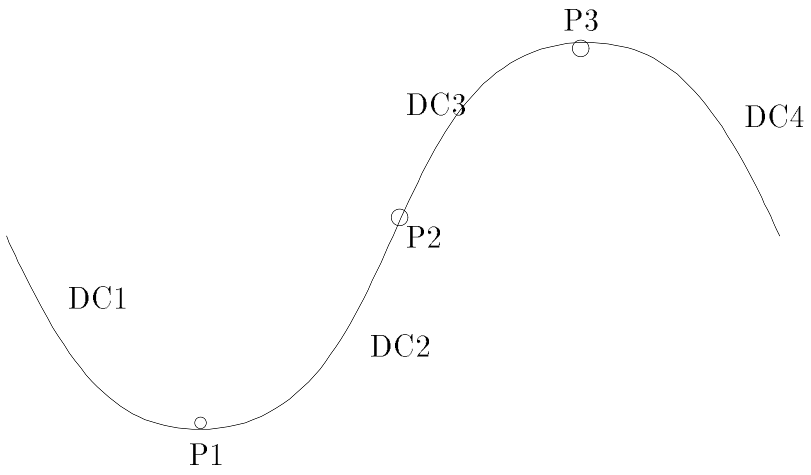

Figure 3: NEWTON representation: scene5.5

RSQUARE = polygon(coordinate(F, 0.0, 0.0), coordinate(F, 1.0, 0.0),
coordinate(F, 1.0, 1.0), coordinate(F, 0.0, 1.0)).
OC = coordinate(F, 0.5, 0.5).
L2 $<$ 0.5.
RHOLE = circle(OC,L2).
RNUT = norm-diff(RSQUARE,RHOLE).
RINSIDE $\subset$ RHOLE.
connected(RINSIDE).
rigid-mapping(M).
ROUTSIDE = apply(M,RINSIDE).
non-overlap(ROUTSIDE,RSQUARE).
return RNUT,ROUTSIDE.

5.4

A shape that is a square to within a tolerance of 0.01 inch.

description scene5.4. F: *frame*; V : *vector*; O : *point*; RSQUARE, R : *region*.

inch $\leq$ length(V) $\leq 2 \cdot$ inch.
F = frame(O,V).
RSQUARE = polygon(coordinate(F, 0.0, 0.0), coordinate(F, 1.0, 0.0),
coordinate(F, 1.0, 1.0), coordinate(F, 0.0, 1.0)).
hausdorff(R,RSQUARE) $\leq 0.1 \cdot$ inch.
return(R).

5.5

A NEWTON-like [deKleer, 1977] representation of a curve in terms of signs of tangents and curvature.

function open-darc(D1, D2 : direction) $\rightarrow$ darcs;
return [D1,D2] $-$ ([D1,D1] $\cup$ [D2,D2]).

constant upper-right : darcs;
return open-darc(right, up).

constant upper-left: darcs;
return open-darc(up,left).

constant lower-left: darcs;
return open-darc(left,down).

constant lower-right: darcs;
return open-darc(down,right).

% Directed curves DC1 and DC2 meet at P.
predicate meets-dc(DC1, DC2 : *dcurve*; P : *point*)
P = end-dcurve(DC1) = start-dcurve(DC2).
end

description scene5.5
DC, DC1, DC2, DC3, DC4 , DC1O, DC2O, DC3O, DC4O : *dcurve*;
P1, P2, P3 : *point.*
join(DC1,DC2,DC3,DC4,DC).
meets-dc(DC1, DC2, P1).
meets-dc(DC2, DC3, P2).
meets-dc(DC3, DC4, P3).

DC1O = interior-dc(DC1).
DC2O = interior-dc(DC2).
DC3O = interior-dc(DC3).
DC4O = interior-dc(DC4).

smooth(DC).

tangent-space(DC1O) $\subset$ lower-right.
tangent(DC,P1) = right.
tangent-space(DC2O) $\subset$ upper-right.
tangent(DC,P2) $\in$ upper-right.
tangent-space(DC3O) $\subset$ upper-right.
tangent(DC,P3) = right
tangent-space(DC4O) $\subset$ lower-right.

curvature-space(DC1O) $\subset$ pos.
curvature(DC,P1) $>$ 0.
curvature-space(DC2O) $\subset$ pos.
curvature(DC,P2) = 0.
curvature-space(DC3O) $\subset$ neg.
curvature(DC,P3) $<$ 0.
curvature-space(DC4O) $\subset$ neg.

return DC.

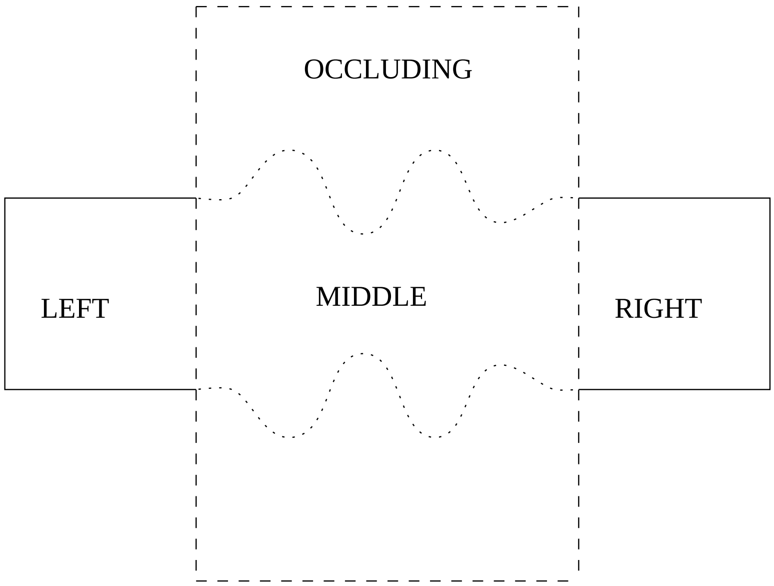

Figure 4: Object with middle occluded : scene5.6

5.6

An object with the middle "occluded" (i.e. unknown), but with two rectangular ends "visible" (known)

description scene5.6
SHAPE,RIGHT,LEFT,MIDDLE,OCCLUDING : *region*; L : *length*;
F : *frame*.

RIGHT = polygon(coordinate(F, 1.0, 0.0), coordinate(F, 2.0, 0.0),
coordinate(F, 2.0, 1.0), coordinate(F, 1.0, 1.0)).
LEFT = polygon(coordinate(F, −2.0, 0.0), coordinate(F, −1.0, 0.0),
coordinate(F, −1.0, 1.0), coordinate(F, −2.0, 1.0)).
L > 0.
OCCLUDING = polygon(coordinate(F, −1.0, −L), coordinate(F, 1.0, −L),
coordinate(F, 1.0, L), coordinate(F, −1.0, L)).
MIDDLE ⊂ OCCLUDING.
SHAPE = RIGHT ∪ MIDDLE ∪ LEFT.
connected(SHAPE).
return SHAPE.

5.7

Same visible scene, but now interpreted as two non-overlapping objects.

description scene5.7
SHAPE1,SHAPE2,RIGHT,LEFT,MID1,MID2,OCCLUDING : *region*;
L : *length*; F : *frame*.

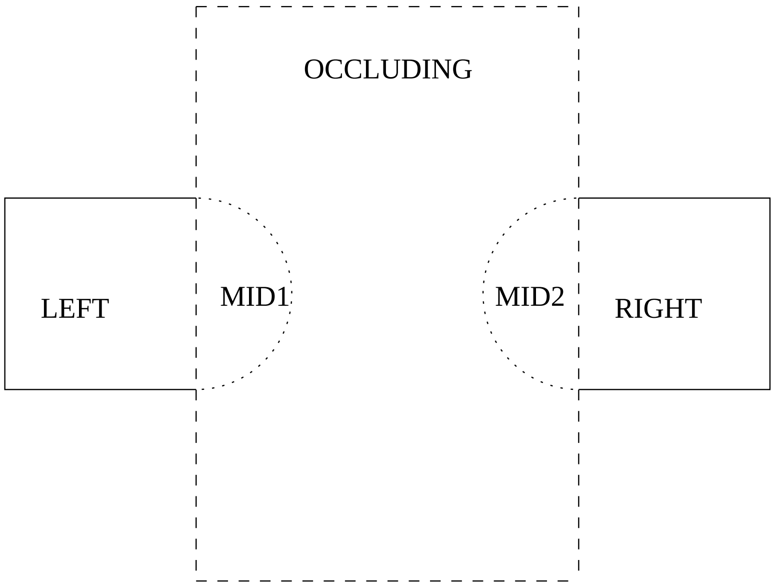

Figure 5: Two objects with middle occluded : scene5.7

RIGHT = polygon(coordinate(F, 1.0, 0.0), coordinate(F, 2.0, 0.0),
coordinate(F, 2.0, 1.0), coordinate(F, 1.0, 1.0)).
LEFT = polygon(coordinate(F, −2.0, 0.0), coordinate(F, −1.0, 0.0),
coordinate(F, −1.0, 1.0), coordinate(F, −2.0, 1.0)).
L > 0.
OCCLUDING = polygon(coordinate(F, −1.0, −L), coordinate(F, 1.0, −L),
coordinate(F, 1.0, L), coordinate(F, −1.0, L)).
MID1 ⊂ OCCLUDING.
MID2 ⊂ OCCLUDING.
SHAPE1 = LEFT ∪ MID1.
SHAPE2 = RIGHT ∪ MID2.
non-overlap(SHAPE1,SHAPE2).
connected(SHAPE1).
connected(SHAPE2).
return SHAPE1,SHAPE2.

5.8

Object inside a cavity with opening on top.

description scene5.8
CAVITY-BOUND, TOP, BOUND, REST: *dcurve*;
CONTAINER, CONTENT, CAVITY: *regular*.

BOUND = join(CAVITY,REST).
CAVITY-BOUND = join(dreverse(CAVITY),TOP).
tangent-space(TOP) = [left,left].
oboundary(CAVITY,CAVITY-BOUND).

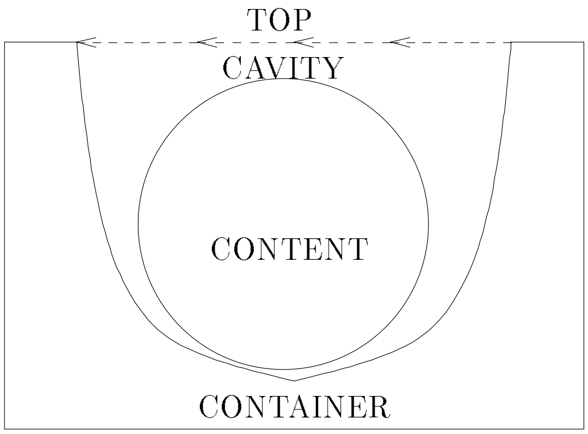

Figure 6: Object inside a cavity: scene5.7

oboundary(CONTAINER,BOUND).
CONTENT $\subset$ CAVITY.
return CONTAINER, CONTENT.

5.9

Topological predicates: those of [Randell, Cui, and Cohn, 92] over regular regions.

predicate not-subset(A,B : *set*)
A $-$ B $\neq \emptyset$.
end

predicate partial-overlap(A,B : *regular*)
overlap(A,B).
not-subset(A,B).
not-subset(B,A).
end

predicate proper-part(A,B : *regular*)
A $\subset$ B.
not-subset(B,A).
end

predicate tangential-proper-part(A,B : *regular*)
proper-part(A,B).
boundary(A) $\cap$ boundary(B) $\neq \emptyset$.
end

predicate non-tangential-proper-part(A,B : *regular*)
proper-part(A,B).
boundary(A) $\cap$ boundary(B) $= \emptyset$. **end**

predicate external-contact(A,B : *regular*)
non-overlap(A,B).
boundary(A) $\cap$ boundary(B) $\neq \emptyset$.

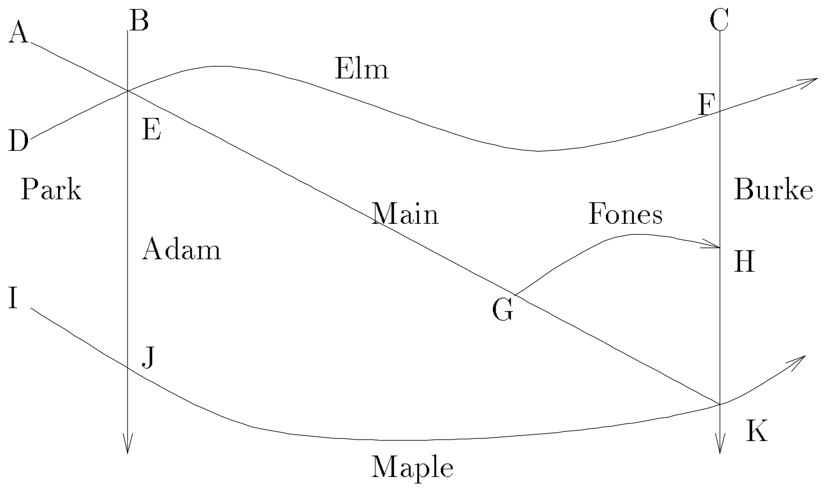

Figure 7: TOUR model of locations and paths: scene5.10

end

predicate disconnected(A,B : *regular*)
A $\cap$ B = $\emptyset$.
end.

The relation "discrete(A,B)" of [Randell, Cui, and Cohn, 92] is the same as our "non-overlap".

The definitions of predicates depending on convexity is the same in our theory as in [Randell, Cui, and Cohn, 92].

predicate top-inside(A,B : *regular*)
D: *dcurve*; R : *regular*. undirect(D) $\subset$ B.
oboundary(R,D).
A $\subset$ R.
end

5.10

Locations connected by paths: TOUR representation [Kuipers, 77].

% PATH goes from A to C through B.
predicate path3(A,B,C : *point*; PATH : *dcurve*)
P1, P2 : *dcurve*.

start-dcurve(P1) = A.
meets-dc(P1,P2,B).
end-dcurve(P2) = C.
join(P1,P2,PATH).
end.

% PATH goes from A to D through B and C.
predicate path4(A,B,C,D : *point*; PATH : *dcurve*)
P1, P2 : *dcurve*.

start-dcurve(P1) = A.
meets-dc(P1,P2,B).
meets-dc(P1,P2,C).
end-dcurve(P2) = D.
join(P1,P2,P3,PATH).
end.

description scene5.10
A, B, C, D, E, F, G, H, I, J, K : *point*;
MAPLE, ELM, ADAM, BURKE, JONES, PARK-BOUND, PBIJ, PBJE, PBED, PBID : *dcurve*;
PARK : *region*.

path4(A, E, G, K, MAIN).
path3(B, E, J, ADAM).
path4(C, F, H, K, BURKE).
path3(D, E, F, ELM).
G = start-dc(FONES).
H = end-dc(FONES).
path3(I, J, K, MAPLE).

cyc-order(out-tangent(ELM,E), −in-tangent(ADAM,E), −in-tangent(MAIN,E),
    −in-tangent(ELM,E), out-tangent(ADAM,E), out-tangent(MAIN,E)).
cyc-order(out-tangent(ELM,F), −in-tangent(BURKE,F),
    −in-tangent(ELM,F), out-tangent(BURKE,F)).
cyc-order(out-tangent(MAIN,G), out-tangent(FONES,G), −in-tangent(MAIN,G)).
cyc-order(out-tangent(BURKE,H), −in-tangent(BURKE,H), −in-tangent(FONES,H)).
cyc-order(out-tangent(MAPLE,J), −in-tangent(ADAM,J),
    −in-tangent(MAPLE,J), out-tangent(ADAM,J)).
cyc-order(out-tangent(MAPLE,K), −in-tangent(BURKE,K), −in-tangent(MAIN,K),
    −in-tangent(MAPLE,K), out-tangent(BURKE,K))

oboundary(PARK,PARK-BOUND).
dcurve-seg(PBIJ, MAPLE, I, J).
dcurve-seg(PBJE, dreverse(ADAM), J, E).
dcurve-seg(PBED, dreverse(ELM), E, D).
join(PBIJ, PBJE, PBED, PBID).
dcurve-seg(PBID, PARK-BOUND, I, D).
end.

5.11

Rectangular box, 1 unit high, top edge scalloped evenly with as many semicircles of at least 1 unit diameter as will fit.

function translation(VT : *vector*) → *mapping*;
V1 : *vector*; O : *point*.
return linear-mapping(frame(O,V1,1), frame(O+VT,V1,1))

description scene5.11
LENGTH, HEIGHT, DIAM, RADIUS : *length*; N : *integer*;
F : *frame*; V : *vector*; O : *point*;
RECT, CIRCLE, BOX: *region*; C : *collection*.

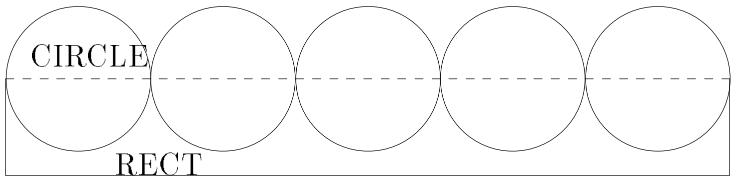

Figure 8: Box with circular scallops: scene5.10

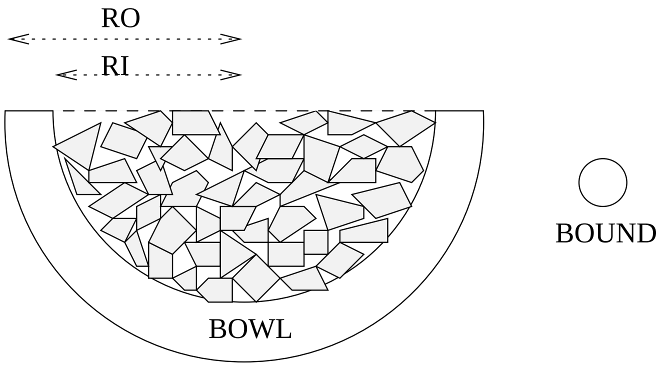

Figure 9: Grains of sand in a bowl: scene5.12

F = frame(O,V).
HEIGHT < LENGTH.
RECT = polygon(coordinate(F, 0.0,0.0), coordinate(F, LENGTH, 0.0),
coordinate(F, LENGTH, HEIGHT), coordinate(F,0.0,HEIGHT)).
N = fix(LENGTH/HEIGHT).
RADIUS = LENGTH / 2·N.
CIRCLE = circle(coordinate(F,RADIUS,HEIGHT),RADIUS).
C = pattern(CIRCLE, translation(RADIUS · V / length(V)),N).
BOX = RECT ∪ union-over(C).
return BOX

5.12

A collection of grains of sand filling the inside of a bowl.

extended description scene5.12 RI, RO, L : *length*; O : *point* F : *frame*; SQUARE, BOWL, INSIDE, BOUND : *regular*; SAND : *collection*.

RI < RO < L.
F = frame(O, L·right, 1).
SQUARE = polygon(coordinate(F, −L, 0), coordinate(F, −L,−L),
coordinate(F, L, −L), coordinate(F, L, 0)).
INSIDE = circle(O,RI) ∩ SQUARE.
BOWL = (circle(O,RO) ∩ SQUARE) − INSIDE.
BOUND = circle(O, centimeter).

$\forall$ (R : *region* $\in$ SAND) fits-in(R,BOUND).
$\forall$ (R : *region* $\in$ SAND) R $\subset$ INSIDE.
$\forall$ (R1,R2 : *region* $\in$ SAND) non-overlap(R1,R2).
connected(union-over(SAND)).
dhaus(union-over(SAND),INSIDE) $<$ centimeter.
    % The sand "fills" the inside, in the sense that every point of INSIDE
    % is close to one of the sand grains.
end.

5.13 Others

It is straightforward to express in this language the 2-D analogues of the representations used in SPAM [McDermott and Davis, 84], MERCATOR [Davis, 86], and ACRONYM [Brooks, 81].

6 Properties of the language

6.1 Omissions

6.1.1 The omission of negation

Our language does not contain a logical negation operator. This seems somewhat unnatural; the system, so to speak, can "understand" what it means for object O to *have* a particular property ϕ but not what it means for the object not to have property ϕ. This omission also leads to inelegancies such as the separate definition of "overlap" and "non-overlap." Unfortunately we are caught between three incompatible desires:

1. We want to have a negation operator.
2. We want to be able to define derived predicates, with their own, local existentially quantified variables.
3. We want our language to have less than the full logical power of first-order logic.

But it is easily seen that negation applied to the definition of a derived predicate gives a formula with universal quantification and disjunction. Hence, a language with (1) and (2) has all the power of first-order language. Something has to give, and we have judged that the least valuable of these is negation.

Other extensions to the logical power of the language, such as disjunction, the use of extended predicates in quantified formulas, and simultaneous quantification over two collections, were omitted because it was felt that the gain in useful expressivity would probably outweighed by the cost in complexity of inference.

6.1.2 Omission of a "locus" operator

Another notable omission in our language is a "locus" or "comprehension" operator that would allow us to take any property $\phi(P)$ of points P, and to construct the region of all points with property ϕ. Thus, for example, for any two foci F1, F2, and length L, we can define the property of a point P, "distance(P,F1) + distance(P,F2) $\leq$ L"; but this does not imply that we can define the ellipse, the locus of all such points, as a particular region in our language.

There are a number of reasons that this omission is necessary, and that it is less unnatural than it might at first appear:

- There are definable properties of points that do not correspond to regions in our ontology. For instance, one can define the property of a point having rational coefficients in a given frame of reference.

  predicate rational (P : *point*; F : *frame*)
  I,J,K : *integer*.
  P = coordinate(F, I/K, J/K).
  end.

  However the set of such points relative to a given frame is not a region by our definition.

- We want to have set-difference on regions to express CSG definitions. However, if we had a locus operator that could turn properties into regions, then set-difference on regions would be equivalent to negation on properties of points, which, as we have said, is undesirable (though there is a difference between negation on unary properties of points and unrestricted negation),

- Self-referential locus definitions (e.g. "R is the locus of all points X $\in$ R $\cup$ S;" "R is the locus of all points X such that X is in some circle C such that C has radius D and C$\subset$R") are very difficult to avoid in constraint languages, as the self-reference can be implicit through contraints involving other variables, and yield very hard fixed-point problems.

- It makes, I think, some intuitive sense that not every property definition can be turned into a region. The property of having rational coefficients is not a terribly unreasonable one, but the collection seems like a set-theorertic rather than a geometric construct. Even in the case of the ellipse, though the definition is obviously a intelligible one, it is not immediately obvious without some thought that the set of all such points is a region of a reasonable kind.

6.2 Limitations of scope

Many aspects of spatial representation that appear in the AI literature are omitted here.

There is no representation of degree of uncertainty here, so certainty grids [Moravec, 88] and other probabilistic spatial representations cannot be expressed. Adding uncertain information makes the problems of defining the model and defining correct inference very much more difficult, and is hopefully orthogonal to the object-level representation.

Many spatial properties of physical significance cannot be expressed (or at least have no direct expression — see section 6.2.) Some of these, no doubt, are inadvertant, properties that did not occur to me. Those that are deliberate are either properties that I felt did not belong to a "commonsense" level understanding, or they are properties that I felt properly belonged in a theory of higher dimensionality. For instance, the property of two objects being kinematically linked belongs to a theory that incorporates restrictions on motion, and therefore time as well as space.

6.3 Expressivity

We have gone out of our way, adopting a number of not wholly comfortable restrictions, in order to restrict the expressivity of our language. The question remains, however, whether all our restrictions accomplish anything. It is by no means inconceivable that our restricted language can in fact express every property that can be expressed in the full first-order language over our set of primitives. By way of analogy: the language of DNF formulas can express any formula in the propositional calculus.

This analogy also reveals that the restricting the language may not be a pointless exercise even if it does not change the expressive power, as it may change the computational properties. For example, it is trivial to determine that a DNF sentence is consistent, whereas it is co-NP-complete to determine that an arbitrary boolean formula is consistent. This, of course, relies on the fact that the translation of the arbitrary formula to DNF may involve an exponential increase in length. So if it turns out that any first-order property over our primitives can be expressed in our constraint language, the next question to ask would be how much expansion is involved in this translation.

I do not know the answers to either of these questions. My conjecture, however, would be that the following features, easily expressible in the first-order language, are not expressible in the constraint language. I list them in decreasing order of my own subjective confidence in this conjecture:

- Integrals. The area and arc-length are primitives in the language, but I doubt that other integrals, such as the center of mass or the moment of inertia, are expressible. (It may be desirable for physical reasoning to add a few such as further primitives.)
- Derivatives. Tangent and curvature are primitives, but I doubt that one can define the third derivative.
- Transcendental regions. If the ontology is extended to allow these as regions, I doubt that the curve $y = \sin x$ can be defined in the language.
- Transcendental properties, e.g. the property of lying on the curve $y = \sin x$.
- Higher-order algebraic curves, e.g. the curve $y = x^3$. Note that the *property* of lying on such a curve certainly is definable.
- Non-rigid linear transformation; the relation of one region being an affine or projective transformation of another. Note that this can be defined in 3-D as the intersection of a plane with a generalized cone.
- Conic sections: ellipses, parabolas, and hyperbolas. This is a special case of the last two.

6.4 Complexity

The general problem of inference in this language is uncomputable, though there are computable subsets of the language. This does not mean that the language is unusable, merely that we should not look for complete inference algorithms. Some simple results ("simple" given well-known results in the literature):

I. Consider the subset of the language containing only the sorts *integer* and *length* and only the functions I·L and L+L. Determining that a set of constraints over this sublanguage is consistent is at best semi-decidable (i.e. determining that it is inconsistent can be undecidable.) Proof: Any Diophantine equation can be expressed as a constraint in this language, and Diophantine equations, by the solution to Hilbert's 10th problem, are semi-decidable. Likewise, determining whether a statement ϕ is a consequence of a set of constraints is undecidable.

I.A. By the same token, even if the sort *integer* and the function "float" are eliminated from the language, if a sublanguage is rich enough to define the property of being an integer, then it is not fully ndecidable whether a set of constraints is consistent. For example, consider the following definition:

predicate int1 (N : *real*);
POLY : *regular*; PA, PB, PC : *point*;
SIDE1, SIDE2, SIDES12, OBOUND : *dcurve*;
ROT : *mapping*;

apply(ROT,POLY) = POLY. % 1.
oboundary(OBOUND,POLY). % 2.
PA $\neq$ PB. % 3.
dcurve-seg(polyline(PA,PB,PC),OBOUND,PA,PC). % 4.
apply(ROT,PA) = PB. % 5.
apply(ROT,PB) = PC. % 6.
N $\cdot$ distance(PA,PB) = length(OBOUND). % 7.
end.

Constraint (1) asserts that the regular region POLY is invariant under the scale mapping ROT. This very strong constraint means that one of the following four cases applies:

a. ROT is the identity.

b. ROT is a reflection and POLY is bilaterally symmetric.

c. ROT is a rotation about a center point by angle $2p\pi/q$, where p and q are integers and POLY is symmetric under rotations of $2\pi/q$.

d. ROT is a rotation by an angle that is not a rational multiple of π and POLY is radially symmetric.

Constraints (2–4) assert that the outer boundary of POLY contains two consecutive straight lines, PA-PB and PB-PC, thus ruling out (d). Constraints (5) and (6) assert that ROT maps PA into PB and PB into PC. This implies that case (c) must hold, and that POLY is, in fact, a regular polygon. Therefore (constraint 7) the length of the boundary is just the number of sides (an integer) times the length of side PA-PB.

Thus any sub-language that can express the above definition (or a similar definition) is not fully decidable.

I.B. There are many ways to define the set of integers in the extended language of collections. For instance:

predicate rectangle(R : *regular*; F : *frame*; L,H : *real*)
X,Y : *real*. % coordinates of lower-left corner.
R = polygon(coordinate(F, X, Y), coordinate(F, X+L, Y),
coordinate(F, X+L, Y+H), coordinate(F, X, Y+H).
end.

predicate int2(N : *real*)
F : *frame*; L,H *real*; R : *regular*; C : *collection*.
rectangle(R, F, L, H).
$\forall$ (RC $\in$ C) rectangle(RC, F, H, H).
$\forall$ (RA,RB $\in$ C) non-overlap(RA, RB).
R = union-over(C).
N $\cdot$ H = L.
end.

An L by H rectangle is tiled with non-overlapping H by H squares. Therefore, L/H is an integer.

II. Consider the subset of the language formed by deleting the functions "length" and "area", the multiplication of a real with an angle, and the predicate "approx-in-tangent". For the purposes of this paragraph, redefine "algebraic" to mean "defined by equations or inequalities on polygons with *integer* coefficients." Modify the semantics of the theory so that *real* and *length* are restricted to be algebraic numbers; *point*, *vector*, *direction*, *frame of reference*, and *mapping* are restricted to have algebraic coordinates; *angle* is restricted to angles whose sine is algebraic; *region, directed curve,* and *collection* are restriced to be semi-algebraic. Then finding a solution to a set of constraints is semi-decidable.

Proof: All possible values for all the variables in a given description can simply be enumerated in order. All of the functions and predicates in the remaining language are computable over the space of algebraic numbers and algebraic regions. Since all collections are by definition finite, the universal quantifications in the extended language can always be evaluated by enumeration.

III. Consider the language formed by deleting the sorts *region*, *dcurve*, and *collection*, with all their associated primitives, and also the multiplication of angles by reals. Then, if one replaces all references to angles by references to their sines and cosines, any definition in the remaining sublanguage is equivalent to an algebraic constraint with integer coefficients. Inference over this language is computable but very hard (the best current result is doubly-exponential) [Mishra, 93].

III.A. Remarkably, the language containing only the sorts *point* and *length* and only the function "distance(P1,P2)" and equality is essentially as hard as the language in (III). Proof: Addition and multiplication can be carried out through a standard compass construction, and an integer coefficient M can be built up through addition in log(M) steps. Therefore, any polynomial constraint of B bits can be expressed in this language using at most $B \log B$ bits (the additional factor of $\log B$ to accomodate the names of the extra variables we may need.)

IV. In many very small subsets of this language, the problem of finding a solution to a set of constraints is NP-hard. Examples:

a. The language containing only the sort *direction* and only the predicate "cyc-order". [Galil and Meggido, 77].

b. The language containing only the sorts *region* and *point*, and only the functions $S \cup R$, $S - R$, and the predicate $P \in R$. Proof: Straightfoward reduction from SAT.

7 The Inference Engine

Of course, this whole design stands or falls on the question of whether one can build an inference engine that can effectively carry out useful inferences. (We have seen that it is impossible to build an engine to carry out *all* inferences.) Since this project currently lacks such an engine, this paper must be considered as more a proposal for research than a report of completed research. However, let me sketch the architecture that I imagine for this inference engine.

There are three main top-level functions in the inference engine:

- infer(constraint,description) — Determine whether the given ground constraint can be inferred from the given description.

- bound(quantity,description) — Determine the bounds on some quantity that can be inferred from the given description.

- render(description) — Generate a "solution" to the constraints in the description; that is, a picture-like representation of a scenario that falls within the description.

Another spatial operators that might be useful, but which I do not intend to address soon, is

- match(description, description) — Find the maximal match between the shapes in the two descriptions.

Functionalities with physical significance, such as

- predict(description) — Given that the regions in the description represent objects, predict what will happen.
- plan-path(point,point,description) — Plan a path from a starting to an ending point, avoiding the regions in the description

are most naturally posed when this purely spatial theory is extended to include time and physical attributes.

The implementation of "infer" and "bound", both of which carry out sound inference, would be centered around pattern-directed inference rules, both forward and backward chaining, which can fall through to special-purpose algorithms. (Rules can be associated with defined symbols as well as primitive symbols.)

The implementation of "render" would begin by applying the same forward-inference room, to deduce as many useful additional constraints as it could. It would next use heuristics to choose a subset of these constraints that is both substantially constraining and tractable. Finally, for each unsatisfied constraint C, it would apply a modification heuristic associated with the form of C to push the current solution towards one that satisfies C.

Given that it is impossible to build a complete inference engine for this representation, our aim will be to automate commonsensically obvious, important inferences, and to ignore too-clever "puzzle-like" inferences [Levesque, 86]. For instance, we do NOT want our inference engine to be able to figure out the indirect representations of the integers in section 6.

8 Future Work

I should like to develop this work in the following directions, in decreasing order of importance:

- The implementation of an inference engine for comparatively small descriptions. This is "make-or-break" for the plausibility of this representation.
- The extension of this representation (a) to three spatial dimensions; (b) to two spatial dimensions times time; (c) to three spatial dimensions times time.
- The application of this representation in physical reasoning, in route planning, in high-level vision, and in other applications of physical reasoning.
- The fine-tuning of this representation: Can it be further restricted in some respects, without losing useful functionality? Need it be expanded for certain natural applications? Part of this will be trying to get a tighter grip on the line between reasonable "commonsensical" spatial inference, and "puzzle-like" inference.

- The extension of the representation and inference engine to handle large cognitive maps ($\geq$ 10,000 significant regions). Presumably, some strong hierarchical structure will be needed here.

References

Brooks, R. (1981). "Symbolic reasoning among 3-D models and 2-D images." *Artificial Intelligence,* vol. 15, pp. 285–348.

Davis, E. (1986). *Representing and Acquiring Geographic Knowledge.* Pitman Press, London.

Davis, E. (1988). "A Logical Framework for Commonsense Predictions of Solid Object Behavior," *Int. Journal of AI in Engineering,*, vol. 3, no. 3, pp. 125-140.

Davis, E. (1991) "Lucid Representations." Tech. Rep. #565, NYU Comp. Sci. Dept.

Davis, E. (1992) "Infinite Loops in Finite Time: Some Observations." *Third International Conference on Knowledge Representation and Reasoning.* pp. 47–58.

Davis, E. (1995) "Approximations of Shape and Configuration Space," NYU Computer Science Tech. Report #703.

de Kleer, J. (1977) "Multiple Representations of Knowledge in a Mechanics Problem-Solver." *Proc. IJCAI-77* pp. 299-304.

Galil, Z. and N. Megiddo (1977). "Cyclic Ordering is NP-complete." *Theoretical Computer Science,* vol. 5, pp. 179–182.

Gelsey, A. "Intelligent Automated Quality Control for Computational Simulation", *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing,* 1995, to appear.

Glasgow, J., N.H. Narayanan, and B. Chandrasekaran (1995). *Diagrammatic Reasoning,* MIT Press.

Hayes, P. (1979) "The Naive Physics Manifesto". In D. Michie (ed.) *Expert Systems in the Microelectronic Age."* Edinburgh University Press, Edinburgh.

Hayes, P. (1985) "Naive Physics I: Ontology for Liquids." In J. Hobbs and R. Moore (eds.) *Formal Theories of the Commonsense World.* ABLEX Pub., Norwood, NJ.

Hayes, P. (1995) "Theoretical Foundations: Section Introduction." In [Glasgow, Narayanan, and Chandrasekaran, 95]

Levesque, H. (1986). "Making Believers out of Computers." *Artificial Intelligence,* vol. 30, pp. 81–108,

McDermott, D. and E. Davis (1984). "Planning Routes through Uncertain Territory." *Artificial Intelligence.* vol. 22, pp. 107–156.

Mishra, B. (1993). *Algorithmic Algebra,* In Texts and Monographs in Computer Science Series, Springer-Verlag, New York.

Moravec, H. (1988). "Sensor Fusion in Certainty Grids for Mobile Robots." *AI Magazine,* vol. 9, pp. 61–74.

Myers, K. and K. Konolige (1995) "Reasoning with Analogical Representations." In [Glasgow, Narayanan, and Chandrasekaran, 95]

Randell, D., Z. Cui, and A.G. Cohn (1992). "A Spatial Logic Based on Regions and Connections." *Third International Conference on Knowledge Representation and Reasoning.* pp. 165–176.

Waltz, D. (1995) "Cognitive and Computations Models: Section Introduction." In [Glasgow, Narayanan, and Chandrasekaran, 95].