

A graph grammar approach to geographic data bases

Report

Author(s):

Meier, Andreas

Publication date:

1982-05

Permanent link:

<https://doi.org/https://doi.org/10.3929/ethz-a-000814171>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)

Originally published in:

D-INFK Technical Report 49

ETH

**Eidgenössische Technische Hochschule
Zürich**

Institut für Informatik

Andreas Meier

**A GRAPH GRAMMAR APPROACH
TO GEOGRAPHIC DATA BASES**

©

Institut für Informatik, ETH Zürich, 1982.

A GRAPH GRAMMAR APPROACH TO GEOGRAPHIC DATA BASES

Andreas Meier
Institut für Informatik
Swiss Federal Institute of Technology (ETH) Zurich
CH-8092 Zurich, Switzerland

Paper to be presented at the 2nd International Workshop on Graph Grammars and their Applications to Computer Science, Osnabrück, October 4-8, 1982.

Abstract:

This paper deals with the modeling of an important class of data bases, i.e. geographic data bases, with emphasis on both structural (data definition) and behavioral (data manipulation) aspects. Geometric objects such as polygons, line segments, and points may have different relations among each other (such as order, adjacency, connectivity) and can be represented in a uniform spatial data structure (structure graph). The dynamic behavior is defined by a finite set of consistency-preserving state transitions (productions) where coincidence problems as well as topological properties have to be solved. This graph grammar approach can be used to study the synchronization of several concurrent productions (Church-Rosser properties) and to specify the data and the manipulation schema of a geographic data base.

Keywords:

geographic data base, graph grammar, schema, structure graph, productions, maps, polygons.

This work was supported, in part, by the Zentenerfond of the ETH grant number 0.330.080.07/8. This is a revised version of a Ph.D. Thesis, Nr. 7043, ETH Zurich, Mai 1982.

Contents:

1.	<u>Representation of Geographic Data</u>	2
2.	<u>Basic Concepts of Graph Grammars</u>	3
	2.1. Relations and Graphs	3
	2.2. Description of the Structure Graph	4
	2.3. Consistency-Preserving Productions	5
3.	<u>Consistency Theorem Applied to a Map</u>	6
	3.1. Consistent States of a Map	7
	3.2. A Set of Map Productions	8
	3.3. Consistency Theorem	9
.	<u>Side Effects in Map Productions</u>	10
.	<u>Predicates over Polygons, Segments, and Points</u>	11
	5.1. Composition of Dimension Two	11
	5.2. Division of Dimension Two	12
6.	<u>Church-Rosser Properties</u>	13
7.	<u>Schema Specification of a Map</u>	14
	7.1. Data Schema	14
	7.2. Manipulation Schema	15
.	<u>Conclusions</u>	17
	<u>References</u>	17

1. Representation of Geographic Data

In recent years there has been a growing interest in the modeling of complex aspects of reality and dynamic systems. Data base modeling is concerned with the static and dynamic behavior of "real world" applications. As of today, most of the effort to represent reality in data bases has been invested in commercial applications, e.g. administration. For different reasons geographic information systems have not been discussed extensively in data base research.

We have studied a few applications concerning geographic data, e.g., triangulation networks, real estate parcel plans, earth resource and land use registers. The structure of these systems varies according to the specific requirements and constraints imposed by the data. In this paper we restrict ourselves to 2-dimensional objects. This category includes also some 3-dimensional objects in real world (such as landscapes) which can be modeled by projection with a sufficient degree of reality and accuracy in two dimensions.

Geographic entities are classified into POINT, LINE, and REGION according to their geometry. These three primitives will provide a convenient distinction for studying structure and behavior from a logical point of view. To design a digital data structure for storing geographic objects in a computer, two main forms (raster and vector) are suitable depending on how the continuous real space has been discretized. In raster form, geographic data are defined using a regular grid whereas in vector form x/y-coordinates describe points and line elements. In this paper we will discuss an example in vector form (real estate parcel plan).

The goal of this paper is to model structure and behavior of a geographic data base in a very consistent manner. It is our opinion, that up to the present time the behavioral properties have not found adequate attention and treatment in the data base literature. Our main objective is to show that by means of an appropriate modeling of manipulations, we can ensure that those preserve consistency. Therefore, it is not sufficient to describe the static components of data only because rather complex side-effects can be caused by operations especially in the field of geo-processing. As a consequence, data items and topological properties must be analyzed to decide whether or not certain modifications are allowed.

In section 2 we propose an approach to describe structural and behavioral aspects of geographic data bases using graph grammars. Section 3 illustrates the graph grammar concept and shows that all productions on a map of polygons preserve consistency. A discussion of merging and dividing productions shows wide side-effects in section 4 and their description by predicates in section 5. Synchronization questions are studied in section 6. Section 7 covers the translation of the structure of a map and its productions into the definition and manipulation schemas. Implementation aspects and concluding

remarks are found in section 8.

2. Basic Concepts of Graph Grammars

Data models can provide powerful abstractions to the design of data structures. However, a complete design of a data base application must include both structural and behavioral aspects. Until recently, few concepts have been studied for describing data as well as manipulations for data base applications in a uniform way [BRO 81]. In this section, we introduce a data model for the design of consistent states (data structures) and consistency-preserving state transitions (manipulation rules). This approach is based on ideas of the relational data model and concepts of graph grammars.

2.1. Relations and Graphs

In an application the objects of interest are grouped together in entity sets according to their common properties (attributes). In the wellknown relational model, these entity sets can easily be represented by relations. A relation is a subset of a Cartesian Product of (not necessarily distinct) domains (set of possible values of an attribute) and represents an entity set in terms of its intension (i.e., entity set name, attributes) and its extension (i.e., data values). However, it has been recognized in recent data base research that the relational model lacks semantics, mainly due to its uniform and sometimes even inconsistent treatment of attributes and relationships, and that it must be extended to capture more of the meaning of the data [COD 79].

Data models vary largely in the manner how they represent relationships among the data. Attributes and relationships between entity sets can be described by the edges of a graph, whereas entity sets or domains are represented by its nodes [ABR 74]. Applying results from graph grammar theory [CLA 79], all consistent states of a data base can be modeled by labeled graphs, and state transitions are given by graph productions. Traditionally, graph grammars are extensions of string grammars: each side of a production is a labeled graph. Unfortunately, most graph grammar approaches make no distinction between intension and extension of a given entity set and often lack concepts of abstraction (i.e., aggregation and generalization). On the other hand, graph grammar concepts are well suited for modeling both structure and manipulation of a data base.

Survey articles on pictorial information systems [CHA 81] show that the relational model is often used because of its tabular structure and the set oriented nature of relational query languages (relational algebra or calculus). This simple structure allows an easy representation of any type of data, but it does not cover much semantics. However, it is very

important to take into account data semantics because rather complex side-effects caused by a manipulation can be encountered. This is especially true in the case of geographic data processing, where geometric objects are embedded in a continuous space and a multitude of metric and topological properties have to be dealt with. This obvious deficiency of the relational model can be alleviated by combining its basic tabular structure with additional graph-theoretical ideas. In the following we introduce a graph-relational approach to model geographic data and manipulations.

2.2. Description of the Structure Graph

In the process of data base modeling, we first seek to identify the relevant entity sets and domains. In a second step, we analyze how entity sets and domains are interrelated. The result of this analysis can be described by a two-level labeled graph called the structure graph, where the nodes represent entity sets and domains, and the edges represent attributes and roles. Such "role" edges appear whenever an entity set includes references to other entity sets, and it links so-called primary domains [COD 79].

In figure 1 we describe a relationship between two entity sets E1 and E2.

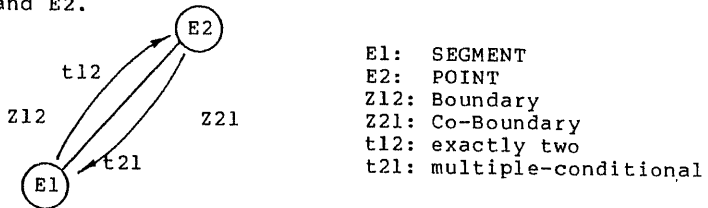


figure 1 Illustration of Abrials access functions.
 (Ei: entity sets, Zi: access functions,
 ti: association types)

The association t12 between entity sets E1 and E2 indicates the number of potential entities in E2 which can be associated with one entity in E1. This situation illustrates exactly the access functions given in [ABR 74]. For our purpose we distinguish four types t of associations between two entity sets [ZEH 80]:

- unique association: $t=1$ (exactly one).
- conditional association: $t=0$ or $t=1$ (at most one).
- multiple association: $t=m, m \geq 1$ (at least one).
- multiple conditional: $t=mc, mc > 0$ (none, one, or more).

Figure 1 also gives an example of the two associations between the entity sets SEGMENT and POINT: To each segment there must exist exactly two points (type $t12=2$), and, on the other hand, each point can be related to several segments (type $t21=mc$). The corresponding access functions Z12 and Z21 are the boundary of a segment and the co-boundary of the two points, respectively.

We now discuss a structure graph which consists of two entity sets and some relationship between them. As described above, each relationship must be written itself as a relation in the relational model. This is done by including identifying roles of the two relations R1 and R2 in relation R12. Therefore, the structure graph also illustrates the references to other relations.

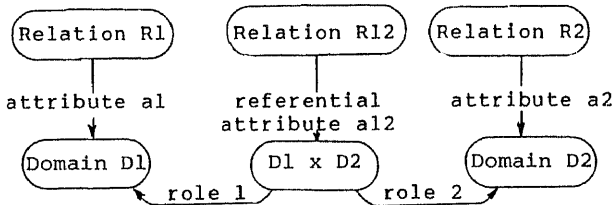


figure 2 Structure graph of a relationship.

As we can conclude from the above, more of the meaning of the data can be extracted from the structure graph than from its underlying relations. The graph illustrates the references more precisely and leads to considerations of data dependencies and semantics. Other approaches than the proposed one are given, e.g. in [DAT 81].

2.3. Consistency-Preserving Productions

Although there are many data models that can be used to describe complex data structures, there exist few tools for modeling behavioral aspects in a data base environment. The graph concept informally introduced above is well suited for describing dynamic aspects of a data base. We therefore present an attempt to model structure and behavior in a uniform way.

A production $p: B1 \Rightarrow B2$ consists of two graphs B1 and B2, each corresponding to the structure graph, and a set of glueing points [CLA 79]. Both sides of a production represent two-level graphs whose roots are labeled with the names of the appropriate relations and whose leaves bear the names of the suitable variables or constants of the referenced domains. The set of glueing points is given implicitly by denotation numbers and allows to identify lefthand and righthand side of a production.

While consistent states of a data base are described by the structure graph, their state transitions are given by a set of productions. Starting from an initial (consistent) state we are able to derive all consistent states of a data base by applying suitable productions. To apply a production $p: B1 \Rightarrow B2$ to a consistent state G, we have to identify the lefthand side B1 as a subgraph of G and replace it by the righthand side B2 of the production according to the glueing points. Locating the

lefthand graph in the data base at its momentary state (pattern matching) and the following replacement by the corresponding righthand graph transforms the data base into a new consistent state. Hence we have a basis for a data base description and manipulation language.

Next we discuss a sample production `INSERT_TUPLE` in relation `R12` (figure 3) which refers to the structure graph described in section 2.2.

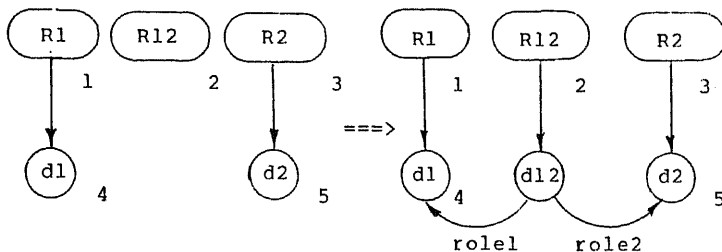


figure 3 shows the elementary side-effects of production `INSERT_TUPLE`.

Given a consistent state of the data base, the question arises which matching conditions must hold when applying an insertion. Since `INSERT_TUPLE` creates a new relationship in `R12`, both relations `R1` and `R2` must be referenced. Therefore, the lefthand side of the production consists of all three relations `R1`, `R2`, and `R12`. By studying the existence or absence of semantic edges in the lefthand side graph, we are able to check the conditions which lead to a possible insertion. These side-effects of the production are given explicitly in graph-notation and express referential constraints. The denotation numbers describe implicitly the glueing points which allow to identify lefthand and righthand side when applying a production to a consistent state. To insert a new tuple in relation `R12` there must be matching values in `R1` and `R2`, e.g., to insert a new segment the two points defining a segment must exist.

Using graph grammar concepts we are able to design consistent states and state transitions on a data base. An illustrative example in the next section will show that for a given set of productions the consistent states defined by the structure graph remain stable under all productions and that each consistent state can be derived from a starting graph by applying suitable manipulation rules.

3. Consistency Theorem Applied to a Map

For the sake of illustration, we describe a map of mutually disjoint polygons and a closed set of appropriate productions. In

[EHR 80] analogous concepts are applied to a data base system for a library. In contrast to this example from commercial data base processing, we must investigate geometric and topological properties defined on points, segments, and polygons. Therefore, composition and division of polygons or segments involve problems of computational geometry.

3.1. Consistent States of a Map

We now describe a map consisting of mutually disjoint polygonal regions. Each polygon is given by a set of directed line segments, and each segment is defined by its start and end point. A map (figure 4) is orientable because we can apply one orientation to each polygon such that each segment can be used in both possible directions. If we traverse the boundary of a given polygon, the polygon under consideration lies to the left or to the right of the enclosing line. It is important to note that neither the directions of two segments meeting at a point must be identical nor the entire boundary segments of a polygon has to be directed in the same sense as we travel around a polygon. Attaching a direction to a segment has been introduced only for the purpose of distinguishing the polygon to the left from the polygon to the right. It is also reasonable to demand that the polygons be singly connected regions. With no loss of generality multiply connected regions can be partitioned so as to result in singly connected regions.

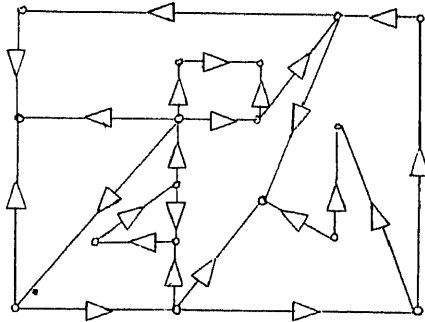


figure 4 Map of mutually disjoint polygons.

Let us discuss the corresponding structure graph which represents all consistent states of a map (figure 5).

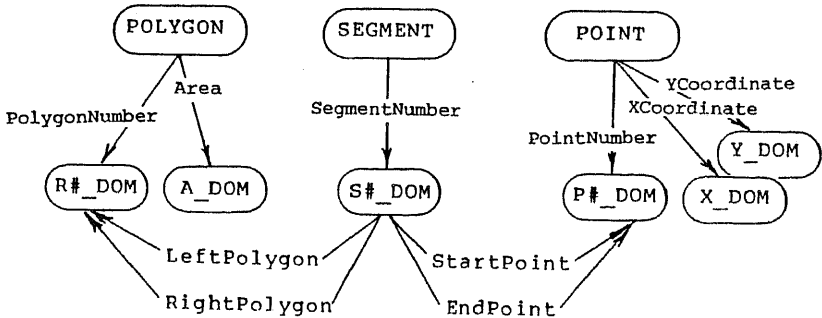


figure 5 Structure graph of a map of mutually disjoint polygons.

- The vertices of the starting-graph are represented by the relations POLYGON, SEGMENT and POINT.
- The relation POLYGON consists of the primary domain R#_DOM and the domain A_DOM.
- The relation POINT is defined by the primary domain P#_DOM and the domains X_DOM and Y_DOM. (Each coordinate pair must be unique with reference to the key)
- The relation SEGMENT consists of the primary domain S#_DOM and two references to the relation POINT by means of the primary domain P#_DOM and, similarly, two references to the relation POLYGON by R#_DOM.
- The vertices POINT and SEGMENT are double-linked by two edges which represent the relationship start point and end point, respectively.
- The two edges between POLYGON and SEGMENT express the relationship of left and right polygon of each segment.

As we can see, every consistent state of a map data base must hold the above constraints all the time. Therefore, each side of a production to be applied to a consistent state is restricted to the above rules of the spatial data structure.

3.2. A Set of Map Productions

The complete set of map productions is:

- INIT PLAN: The initial manipulation rule defines the boundary of a plan by existing points.
- MERGE_SEGMENT: This rule is only allowed for adjacent segments of the same or opposite direction. It is a composition of dimension one and merges two segments by deleting the "boundary" point in common.
- DIVIDE_SEGMENT: This rule is a subdivision of dimension one. A segment of a polygon is divided into two segments by

assigning to any inner point the meaning of an end point.

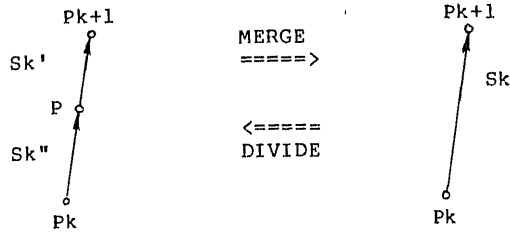


figure 6 illustrates the trivial case of composition and division of dimension one.

- MERGE_POLYGON: This composition of dimension two turns two polygons with a common boundary (segment or chain) into a new polygon by omitting this common boundary.
- DIVIDE_POLYGON: This is the reverse rule of merging two polygons and defines a segment or chain by which the polygon is parceled out (two neighboring polygons).

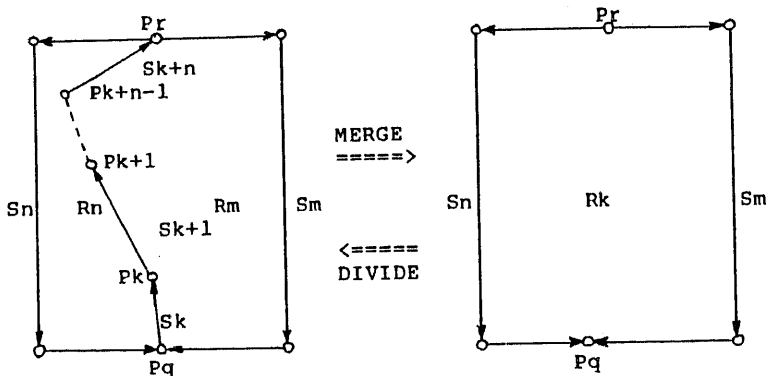


figure 7 illustrates the amazingly complex case of composition and division of dimension two.

3.3. Consistency Theorem

We now show that the consistent states of a map as defined by the rules of its structure graph remain stable under all productions and that each consistent state can be derived from the initial state by applying suitable productions:

Preservation of consistent states

We denote the number of points, segments, and polygons by a_0 ,

a_1 , and a_2 respectively and have to show that the alternating sum (Euler Characteristic)

$$X = a_2 - a_1 + a_0$$

equals 2 (normal form of the sphere) and does not change when applying one of the above productions.

In the case of division of dimension one the numbers a_0 and a_1 both increase by one and leave X constant. In the case of division of dimension two we create one new polygon and increase a_2 by one but, on the other hand, we create one more boundary segment than are inner points created and increase a_1 by n and a_0 by $n-1$, respectively. Therefore, X remains constant. The compositions of dimension one and two do not change X because their inverse productions do not. Thus we have shown that all productions have the same Euler Characteristic and leave all consistent states topologically invariant.

Generation of consistent states

We assume that a consistent state is given with more than one polygon, and we have to show that it can be derived by applying only the above productions.

Since we have a map which contains more than one polygon we can use the composition of dimension two such that we get a map with lesser degree of the parceling (number of polygons, segments, and points). By the induction hypothesis we conclude that our given map is also derivable.

4. Side-Effects in Map Productions

Let us discuss the production MERGE_POLYGON into some detail. Figure 8 shows the geometric and topological conditions of MERGE_POLYGON in an abstract notation by means of graphs.

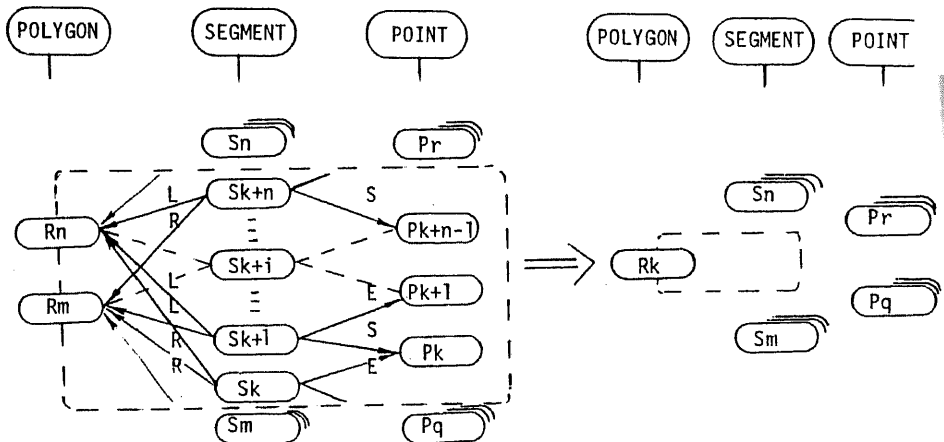


figure 8 Production MERGE_POLYGON with conditions encircled.

The allowed conditions are expressed by the existence or absence of semantic edges in the encircled area of the lefthand graph (figure 8). In this example they read:

- The two polygons R_n and R_m must exist.
- The polygons R_n and R_m must have at least one segment in common; i.e., there must be at least one node S_k with exactly two edges attaching the nodes R_n and R_m . Polygons sharing only one point are prohibited since the merging of such polygons would result in multiply connected regions which we have excluded from the beginning.
- The common boundary must be singly connected; i.e., there must be a path which starts from S_k and alternately visits a node among P_k, \dots, P_{k+n-1} or S_{k+1}, \dots, S_{k+n} .
- Each inner point of the common boundary must have exactly two polygons meeting there, namely R_n and R_m ; i.e., all nodes P_k, \dots, P_{k+n-1} must not have any semantic edges that intersect with the encircling line.

5. Predicates over Polygons, Segments, and Points

We express the conditions of the productions by breaking up each lefthand graph into a set of predicates. Since we investigate geometric and topological properties defined on points, segments, and polygons, the notion of predicate must be seen in a somewhat more general sense [MIN 69]. Let F be an abstract family of geometric objects being of specific interest to a topological property (e.g., family of singly connected line segments, family of convex polygons, etc.) and let X be a variable of basic geometric object type (e.g., point, segment, or polygon). Consequently, we can associate with any abstract family F of geometric objects a predicate $p[F](X)$ which is true if and only if X is in the family F . For example, $p[\text{convex}](X)$ is true or false according to whether X is or is not a convex set.

5.1. Composition of Dimension Two

The following predicates are relevant conditions for the production MERGE_POLYGON:

- $p[\text{existing polygons}](R_n, R_m)$
This predicate verifies the existence of polygons R_n and R_m .
- $p[\text{bordering}](R_n, R_m)$
This predicate is true or false depending on whether or not at least one common boundary segment of the polygons R_n and R_m exists.
- $p[\text{connected}](S_k, \dots, S_{k+n})$
This predicate indicates whether or not all common boundary segments S_k, \dots, S_{k+n} of the two polygons R_n, R_m are singly connected.
- $p[\text{coincident}](P_k, \dots, P_{k+n-1})$
This predicate determines whether or not all inner points

P_k, \dots, P_{k+n-1} are coincident with exactly the two polygons R_n and R_m , i.e. each inner point has exactly two adjacent segments.

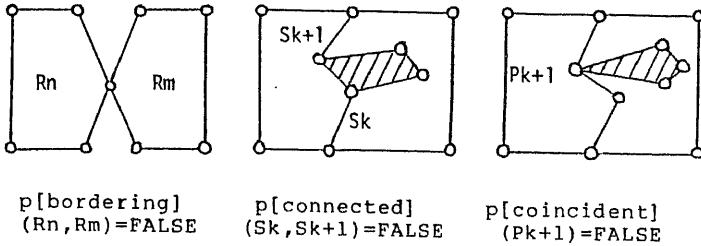


figure 9 Unacceptable conditions for MERGE_POLYGON.

If one of the predicates is false then the composition of dimension two would result in a previously excluded case. The following three examples (figure 9) illustrate what happens if one of the conditions does not hold.

5.2. Division of Dimension Two

Let us briefly consider the inverse production DIVIDE POLYGON. When dividing a polygon R_k into two non-overlapping polygons R_n and R_m , the new chain S_k, \dots, S_{k+n} must link two already existing points on the boundary of polygon R_k . This chain with inner points P_k, \dots, P_{k+n-1} is neither allowed to lie partly outside polygon R_k nor to coincide anywhere with the boundary of the polygon, except in the two border points P_q and P_r .

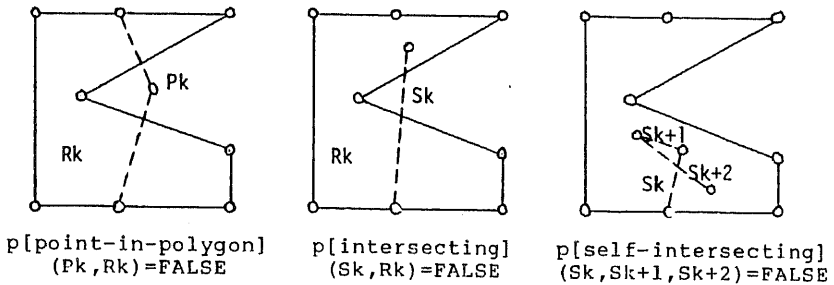


figure 10 Unacceptable conditions for DIVIDE_POLYGON.

Again, we can devise a set of predicates for transaction DIVIDE_POLYGON (see figure 10):

- $p[\text{point-in-polygon}](P_k, R_k)$

This predicate indicates whether or not a given inner point P_k is a point in the interior of polygon R_k .

- $p[\text{intersecting}](S_k, R_k)$
This predicate determines whether or not a chain segment S_k of the new boundary is intersecting with the boundary of polygon R_k in any point other than P_q or P_r .
- $p[\text{self-intersecting}](S_k, \dots, S_{k+n})$
This predicate is true or false according to whether or not any segments of the new boundary are intersecting one another.

Since designing efficient geometric algorithms is beyond the scope of this paper we do not go into further details.

6. Church-Rosser Properties

A subtle problem of mutual dependency arises when several polygons are to be manipulated at the same time (i.e., in parallel). This is not only of pure theoretical interest, but also has some practical consequences since modifications can last as long as a year. On the other hand, a user should not be able to gain exclusive access over the entire data base for his updating work. It is therefore desirable to find criteria for locking mechanisms involving only parts of a data base which could be engaged into a conflicting situation.

This difficulty of parallel independence has been treated in the literature [CLA 79] and condensed in the Church-Rosser theorem. It states that parallel independent productions can be executed concurrently or in any desired sequential order yielding the same result. The following example (figure 11) illustrates the basic idea:

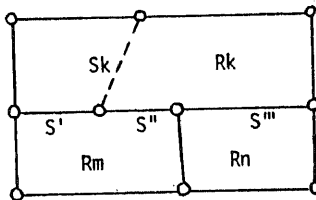


figure 11 Conflicting situation caused by concurrent productions.

Can we merge the two polygons R_m and R_n without complications concurrently with dividing R_k by segment S_k ? The common boundary segments S' , S'' , and S''' of the polygons R_k , R_m and R_n have common vertices and edges in the intersection of the leftside graphs of both productions MERGE POLYGON and DIVIDE POLYGON. This means that the requirements of the theorem are not met. As a consequence, we are not allowed to process the above in parallel. When merging R_m and R_n , the topological properties of the common boundary segments with R_k are being changed which cannot be detected by the process which concurrently divides region R_k . Undoubtedly, this will lead to a conflict.

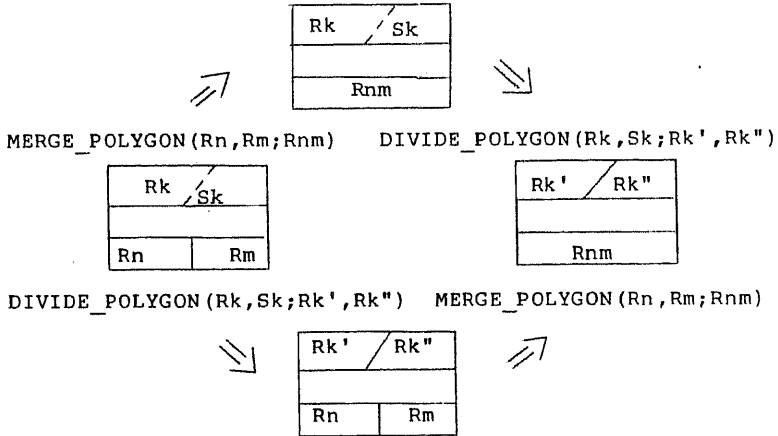


figure 12 Parallel independent productions.

Figure 12 illustrates how MERGE- and DIVIDE_POLYGON can be processed in parallel. Since the polygons involved by the map productions are separated widely enough, no conflicting situation can occur. This study of concurrent productions helps us to find reasonable criteria for locking mechanism from a logical point of view.

7. Schema Specification of a Map

In order to implement structure graph and productions, a transformation of both into a linear notation is required. The collection of these definitions for a specific application is called a schema [DBTG 71]. We now present an extension to this notion of a schema by describing the structure graph and appropriate productions.

7.1. Data Schema

The data schema contains the static aspects, i.e. the data structure. This structural definition basically lists domains, entity sets and properties of their respective elements. Thus the two-level structure of the structure graph introduced in section 2.2 is retained.

Also, the type of the association between two relations is included in the declaration of a role attribute, by specifying both cardinality and dependency of the association [ELM 80]. The cardinality (UNIQUE or MULTIPLE) places restrictions on the number of entities of one relation that may be related to an entity of the other relation, whereas the dependency defines whether an entity can exist on its own or not (CONDITIONAL).

We now present the data schema of the corresponding structure graph which describes all consistent states of a map of mutually disjoint polygons:

```
SCHEMA MapOfPolygons;

DOMAIN
  Numbers      = 10000..99999;

RELATION Point;
  ATTRIBUTE
    P#:          Numbers;
    XCoordinate: REAL;
    YCoordinate: REAL;
  IDENT
    P# PRIMARY DOMAIN PointNumbers;
    (XCoordinate,YCoordinate);
  END Point;

RELATION Polygon;
  ATTRIBUTE
    R#:          Numbers;
    Area:        REAL;
  IDENT
    R# PRIMARY DOMAIN PolygonNumbers;
  END Polygon;

RELATION Segment;
  ROLE
    LeftPolygon: Polygon ASSOCIATION MULTIPLE;
    RightPolygon: Polygon ASSOCIATION MULTIPLE;
    StartPoint:  Point ASSOCIATION MULTIPLE CONDITIONAL;
    EndPoint:    Point ASSOCIATION MULTIPLE CONDITIONAL;
  ATTRIBUTE
    S#:          Numbers;
  IDENT
    S#;
    (StartPoint,EndPoint);
  END Segment;
...

```

7.2. Manipulation Schema

After having transformed the structure graph into a data schema, we turn to the problem of specifying the productions of the graph grammar. This is accomplished by the notion of a transaction. We propose a three-part transaction specification as follows: The scope of the transaction describes the data affected by the transaction. The condition under which the production may be applied to this data is called the rule governing the transaction. Finally, the operations on the database which implement the production are termed the event of the transaction.

```
TRANSACTION MergePolygon;
  SCOPE (* left side of the production *)
    SELECT OldPolygons FROM Polygon;
    SELECT CommonEdgeSegments FROM Segment;
    SELECT OuterEdgeSegments FROM Segment;
    SELECT InnerPoints FROM Point;
    PREPARE NewPolygons FOR Polygon;
  RULE
    PolygonsExist(OldPolygons) AND
    bordering(CommonEdgeSegments) AND
    connected(CommonEdgeSegments) AND
    coincident(InnerPoints) AND
    NOT PolygonsExist(NewPolygons);
  EVENT (* -> right side of the Production *)
    DELETE OldPolygons FROM Polygon;
    INSERT NewPolygons IN Polygon;
    UPDATE OuterEdgeSegments IN Segment;
    DELETE CommonEdgeSegments FROM Segment;
    DELETE InnerPoints FROM Point;
END MergePolygon;
```

Next we list the specification of production DIVIDE_POLYGON:

```
TRANSACTION DividePolygon;
  SCOPE (* left side of the production *)
    SELECT OldPolygons FROM Polygon;
    SELECT OuterEdgeSegments FROM Segment;
    PREPARE NewPoints FOR Point;
    PREPARE NewSegments FOR Segment;
    PREPARE NewPolygons FOR Polygon;
  RULE
    PolygonsExist(OldPolygons) AND
    point-in-polygon(NewPoints) AND
    intersecting(NewSegments) AND
    self-intersecting(NewSegments) AND
    NOT PolygonsExist (NewPolygons);
  EVENT (* -> right side of the production *)
    DELETE OldPolygons FROM Polygon;
    INSERT NewPolygons IN Polygon;
    INSERT NewSegments IN Segment;
    UPDATE OuterEdgeSegments IN Segment;
    INSERT NewPoints IN Point;
END DividePolygon;
```

We have shown very briefly how the data structure and the productions can be specified. The specification language as presented in this paper is very close to the graph grammar concept and defines a set of rules both for structure and behavior of a data base. An interesting approach for process specification can be found in [GOL 80] which also is based on a cognitive model.

8. Conclusions

Investigations into the representation of data and logical schemes have, in the past, been carried out mainly in the field of AI. The principles involved however, are not only of interest to theoretical computer science. We have taken a geographic data base as an example to show the applicability of graph grammars to practical systems. Given a real estate parcel plan we can ask, which practical and frequent productions should be applied to it. As a result we find merging and dividing productions both in dimension one and two. The graph grammar concept outlined in this paper is well-suited to preserve consistency of such productions and general enough to investigate additional studies (e.g., Church-Rosser properties). An implementation [MEI 82] proved that the concept is reasonable and allowed to estimate the expenditure for design and implementation of such a consistency-preserving system.

Acknowledgments:

The author is grateful to J.Nievergelt and C.A.Zehnder for their support of this work leading to a doctoral degree in Computer Science and to H.Hinterberger, M.Ilg, and R.Marti for their critical comments on an earlier version of this paper.

References:

- [ABR 74] Abrial J.R.: Data Semantics. Klimbie J.W., Koffeman K.L. (Ed.): Data Base Management. North-Holland, Amsterdam 1974, pp. 1-59.
- [BRO 81] Brodie L.M.: On Modelling Behavioural Semantics of Databases. Proc. 1981 Conf. on Very Large Data Bases, Cannes 1981.
- [CHA 80] Chang S.K.: Pictorial Information Systems. IEEE Computer, Vol. 14, No. 11 (November 1981).
- [CLA 79] Claus V., Ehrig H., Rozenberg G.: Graph-Grammars and their Application to Computer Science and Biology. Lecture Notes in Comp. Science, No. 73, Springer-Verlag, Berlin 1979.
- [COD 79] Codd E.F.: Extending the Database Relational Model to Capture More Meaning. ACM Transactions on Data Base Systems, Vol. 4, No. 4, pp. 397-434.
- [DAT 81] Date C.J.: Referential Integrity. Proc. 1981 Conf. on Very Large Data Bases, Cannes 1981.
- [DBTG 71] Data Base Task Group of CODASYL Programming Language Committee. Report (April 1971), Available from ACM.

- [EHR 80] Ehrig H., Kreowski H.J.: Applications of Graph Grammar Theory to Consistency, Synchronization and Scheduling in Data Base Systems. Information Systems, Vol. 5, 1980, pp. 225-238.
- [ELM 80] El-Masri R., Wiederhold G.: Properties of Relationships and their Representation. AFIPS Conference Proc. 1980, Vol. 49, Anaheim, pp. 319-326.
- [GOL 80] Goldman N.W., Wile D.S.: A Relational Data Base Foundation for Process Specification. In: Chen P. P.-S.(Ed.): Entity-Relationship Approach to Systems Analysis and Design. North-Holland, Amsterdam 1980, pp. 413-432.
- [MEI 82] Meier A., Ilg M.: Consistent Operations on a Spatial Data Structure. IEEE Conf. on Pattern Recognition and Image Processing, Las Vegas 1982.
- [MIN 69] Minsky M., Papert S.: Perceptrons - An Introduction to Computational Geometry. The MIT Press, Cambridge, Mass. 1969.
- [ZEH 80] Zehnder C.A., Thurnherr B.: Dynamic Consistency Constraints in the Conceptual Schema and their Connections with the External Schema. Informatik-Fachberichte, Nr. 33, Springer Verlag, Berlin 1980, pp.181-195.

Berichte des Instituts für Informatik

- Nr.25 U. Ammann: Error Recovery in Recursive Descent Parsers and Run-time Storage Organization
- Nr.26 E. Zachos: Kombinatorische Logik und S-Terme
- *Nr.27 N. Wirth: MODULA-2
- *Nr.28 J. Nievergelt, J. Weydert: Sites, Modes and Trails: Telling the User of an Interactive System where he is, what he can do, and how to get to places
- *Nr.29 A.C. Shaw: On the Specification of Graphic Command Languages and their Processors
- *Nr.30 B. Thurnherr, C.A. Zehnder: Global Data Base Aspects, Consequences for the Relational Model and a Conceptual Schema Language
- *Nr.31 A.C. Shaw: Software Specification Languages based on regular Expressions
- Nr.32 E. Engeler: Algebras and Combinators
- *Nr.33 N. Wirth: A Collection of PASCAL Programs
- *Nr.34 R. Marti, J. Rebsamen, B. Thurnherr: Meta Data Base Design - Consistent Description of a Data Base Management System
- *Nr.35 H.H. Nägeli, R.Schoenberger: Preventing Storage Overflows in High-level Languages
- J. Hoppe: A Simple Nucleus written in Modula-2
- *Nr.36 N. Wirth: MODULA-2 (second edition)
- Nr.37 Hp. Bürkler, C.A. Zehnder: EDV-Projektentwicklung - Ein Arbeitsheft für Informatik-Studenten
- *Nr.38 H. Burkhart, J. Nievergelt: Structure-oriented editors
- *Nr.39 A. Meier, C.A. Zehnder: Flächenmodell-Register: Die Strukturen wichtiger geographischer Datensammlungen der Schweiz
- Nr.40 N. Wirth: The Personal Computer Lilith

- Nr.41 T.M. Fehlmann: Theorie und Anwendung des Graphmodells der Kombinatorischen Logik
- Nr.42 E. Graf: Probabilistische Algorithmen und Computer-unterstützte Untersuchungen von probabilistischen Primalitätstests
- Nr.43 H. Burkhart: Konzepte zur Systematisierung der Benutzerschnittstelle in interaktiven Systemen und ihre Anwendung auf den Entwurf von Editoren
- Nr.44 J. Nievergelt, F.P. Preparata: Plane-sweep Algorithms for Intersecting Geometric Figures
- Nr.45 M. Reimer, J.W. Schmidt: Transaction Procedures with Relational Parameters
- Nr.46 J. Nievergelt, H.Hinterberger, K.C. Sevcik: The Grid File: An adaptable, symmetric multi-key file structure
- Nr.47 J. Nievergelt: Errors in dialog design and how to avoid them
- Nr.48 P. Läubli: PG - Ein interaktives System für die Manipulation von Figuren der projektiven Geometrie
- Nr.49 A. Meier: A Graph Grammar Approach to Geographic Data Bases

* out of stock