Sortal Grammars for Urban Design

Rudi Stouffs

Introduction

Grammar formalisms have been around for over 50 years and have found application in a wide variety of disciplines and domains, to name a few, natural language, architectural design, mechanical design, and syntactic pattern recognition. Grammar formalisms come in a large variety (e.g., [1–5]), requiring different representations of the objects being generated, and different interpretative mechanisms for this generation. Altering the representation may necessitate a rewrite of the interpretative mechanism, resulting in a redevelopment of the entire system. At the same time, all grammars share certain definitions and characteristics. Grammars are defined over an algebra of objects, U, that is closed under the operations of addition, $+$, and subtraction, $-$, and a set of transformations, F. In other words, if u and v are members of U, so too are $u + f(v)$ and $u - f(v)$ where f is a member of F. In addition, a match relation, $\leq$, on the algebra governs when an object occurs in another object under some transformation, that is, $f(u) \leq v$ whenever u occurs in v for some member f of F, if u and v are members of U.

Building on these commonalities, we consider a component-based approach for building grammar systems, utilizing a uniform characterization of grammars, but allowing for a variety of algebras, and match relations (or interpretative mechanisms) [6]. *Sortal* representations constitute the components for this approach. They implement a model for representations, termed *sorts*, that defines formal operations on *sorts* and recognizes formal relationships between *sorts* [7]. Each *sort* defines an algebra over its elements; formal compositions of *sorts* derive their algebraic properties from their component *sorts*. This algebraic framework makes *sortal* representations particularly suited for defining grammar formalisms. Provided a large variety of primitive *sorts* are defined, *sortal* representations can be conceived and built corresponding to almost any grammar formalisms.

R. Stouffs (✉)
Delft University of Technology, Delft, The Netherlands
e-mail: r.m.f.stouffs@tudelft.nl

J. S. Gero (ed.), *Studying Visual and Spatial Reasoning for Design Creativity,*
DOI 10.1007/978-94-017-9297-4_4

The need for varying grammar formalisms using varying representations is quite apparent in urban design. CAD systems are very powerful drawing tools and fit for design practice, also in urban design. On the other hand, GIS systems are very powerful systems for accessing large-scale urban data; hence they play an important role in urban planning as analytical tools. However, these tools were conceived as interactive maps and so they lack capacities for designing. Therefore, in urban design, the linking of GIS to CAD tools and representations becomes an important goal to allow designing directly on the GIS data.

For urban design and simulation, *sortal* grammars may include, among others, descriptive grammars, GIS-based set grammars, shape grammars and any combination thereof.

Sortal Representations

Stouffs [7] describes a semi-constructive algebraic formalism for design representations, termed *sorts*, that provides support for varying grammar formalisms. It presents a uniform approach for dealing with and manipulating data constructs and enables representations to be compared with respect to scope and coverage, and data to be converted automatically, accordingly. *Sorts* can be considered as hierarchical structures of properties, where each property specifies a data type; properties can be collected and a collection of one or more properties can be assigned as an attribute to another property. *Sorts* can also be considered as class structures, specifying either a single data type or a composition of other class structures.

Each *sort* has a behavioral specification assigned, governing how data entities combine and intersect, and what the result is of subtracting one data entity from another or from a collection of entities from the same sort. This behavioral specification is a prerequisite for the uniform handling of different and a priori unknown data structures and the effective exchange of data between various representations. The behavioral specification of a *sort* is based on a part relationship on the entities of this *sort*, with the *sortal* operations of addition, subtraction, and product defined in accordance to this part relationship. As such, a behavioral specification explicates the match relation (or interpretative mechanism) underlying a *sortal* algebra and grammar. The behavioral specification of a primitive *sort* forms part of the predefined template of this *sort*; composite *sorts* derive their behavioral specification from the component *sorts* in conformity with the compositional operation. In addition, a functional *sort* allows the specification of data (analysis) functions that automatically apply to *sortal* structures through tree traversal.

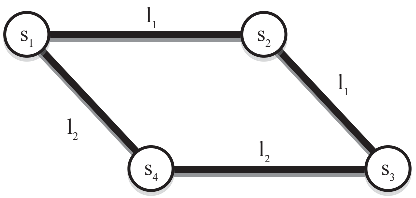

Fig. 1 A simple transportation network consisting of two lines and four stops

A Simple Example

Consider the following example: given a public transportation network, where the transportation nodes represent stations or stops and the edges represent transportation lines, how can we derive a transportation lines connectivity graph, where the nodes represent transportation lines and the edges exchanges between these lines? Basically, we are interested in knowing how many lines there are, which stations or stops are on which line, which lines connect to one another, etc., such that we can take into account the number of exchanges that might be necessary to get from one point to another. We assume that stations and stops have attribute information specifying the lines that stop here.

From a programming point of view, the derivation of a line connectivity graph from a transportation network or stop connectivity graph is not all that complex, but without proper programming knowledge, the task can still be very challenging. We show how one might approach this problem using *sortal* structures. First, we need to define the representational structure we will use as a starting point.

Figure 1 illustrates the data that may be present in the stop connectivity graph. We ignore the format in which the data may be provided, and instead consider the basic data entities that are required. Firstly, we need to represent the stops themselves, e.g., “s1”, “s2”, “s3” and “s4”. We can do so by their name, a string. We define a primitive *sort* with Label as *sortal* template:

```
sort stops : [Label];
form $stops = stops: { "s1", "s2", "s3", "s4"};
```

The first line defines the *sort* stops, with template Label. The second line defines an exemplary data form of *sort* stops and referenced by the *sortal* variable $stops. It defines a collection of stops or, more specifically, stop labels “s1” through “s4”. Similarly, we can represent the transportation lines that use a stop also as strings with *sortal* template Label:

```
sort lines : [Label];
form $lines = lines: { “l1”, “l2” };
```

Finally, we need to represent the connectivity relations. For this, we use the Property template. Unlike other templates, the Property template requires two primitive *sorts* as arguments, and defines not one but two new primitive *sorts*:

```
  sort   (connections,   rev_connections)   :   [Property]
(stops, stops);
```

The two arguments define the representational structure for the tails and heads of the connectivity relationship. Since the Property template applies to directional relationships, we consider two resulting sorts: connections and rev_connections (reverse connections). An example is given below.

A complex representational structure is defined as a composition of primitive representational structures. *Sortal* structures offer us two compositional operators: an attribute operator, ^, specifying a subordinate, conjunctive relationship between *sortal* data, and a sum operator, +, specifying a co-ordinate, disjunctive relationship. Considering the room adjacency graph, we can define a corresponding *sortal* structure as follows:

```
  sort   input   :   stops   ^   (lines   +   connections   +
rev_connections);
```

Stops have lines, connectivity relationships and reverse connectivity relationships as attributes. A corresponding data form would be defined as:

```
    form $input = input:
    { #me-stops-1 "s1"
      { (lines): { "l1" },
        (connections): { me-stops-2, me-stops-4 } },
      #me-rooms-2 "r2"
      { (lines): { "l1" },
        (connections): { me-stops-3 } },
      #me-rooms-3 "r3"
      { (lines): { "l1", "l2" },
        (connections): { me-stops-4 } },
      #me-rooms-4 "r4"
      { (lines): { "l2" } } };
```

Of course, this data form may be generated from the original data format, rather then specified in textual form. Especially, the connectivity relationships may be generated automatically from a sequentially-ordered list of stops on a transportation line. #me-stops-1 is a reference ID for “s1” that can be used later, in the form me-stops-1, to reference “s1” in an connectivity relationship from a different stop. The specification of reverse connectivity relationships is optional; the *sortal* interpreter will automatically generate these.

Similarly, we can define a representational structure for the output we need to produce. Consider the goal to group stops on the same line. For this, we can consider lines with stops as attributes; the stops themselves may still have (reverse) connectivity relationships as attributes:

```
  sort   output   :   lines   ^   stops   ^   (connections   +
rev_connections);
  form $output = output: $input;
```

The second line defines a variable of *sort* output with $input as data. Since $input is defined of sort input, the data must be converted to the new *sort*. This conversion is done automatically based on rules of semantic identity and syntactic similarity. The result is:

```
  form $output = output:
  { "l1"
    { #me-stops-1 "s1"
      { (connections): { me-stops-2, me-stops-4 } },
      #me-stops-2 "s2"
      { (connections): { me-stops-3 },
        (rev_connections): { me-stops-1 } } },
      #me-stops-3 "s3"
      { (connections): { me-stops-4 },
        (rev_connections): { me-stops-2 } } },
    "l2"
    { #me-stops-1 "s1"
      { (connections): { me-stops-2, me-stops-4 } },
      #me-stops-3 "s3"
      { (connections): { me-stops-4 },
        (rev_connections): { me-stops-2 } },
      #me-stops-4 "s4"
      { (rev_connections): { me-stops-1, me-stops-3 } }
} };
```

This is a collection of transportation lines, with for each line a list of stops (ordered alphabetically, rather than sequentially), with stop connectivity relationships (and reverse relationships). It does not yet constitute a transportation lines connectivity graph. For this, we need to define relationships (and reverse relationships) between transportation lines:

```
  sort (exchanges, rev_exchanges) : [Property] (lines,
lines);
```

We can now consider lines with exchange relationships (and reverse relationships); the lines may still have stops as attributes but for the automatic conversion of the relationships to take place, we must omit the stop connectivity relationships.

```
  sort   graph   :   lines   ^   (stops   +   exchanges   +
rev_exchanges);
  form $graph = graph: $ouput;
```

The result will be:

```
  form $graph = graph:
  { #me-lines-1 "l1"
    { (stops): { "s1", "s2", "s3" },
      (exchanges): { me-lines-1, me-lines-2 },
      (rev_exchanges): { me-lines-1 } },
    #me-lines-2 "l2"
    { (stops): { "s1", "s3", "s4" },
      (exchanges): { me-lines-2 },
      (rev_exchanges): { me-lines-1, me-lines-2 } } };
```

Using functional entities integrated in the representational structures, we can also calculate the number of lines, the number of stops per line, etc. For this, we define a new primitive *sort* with Function as *sortal* template, and define a representational structure of counting functions with lines as attribute, where the lines themselves have stops as attributes, though we ignore any relationships:

```
  sort counts : [Function];
  sort number_of_lines: counts ^ lines ^ stops;
  // func count(x) =  c : {c(0) = 0, c(+1) = c + 1};
  ind  $count  =  number_of_lines:  count(lines.length)
$output;
```

The last line defines a data form as an individual (a single data entity, not a collection of data entities or individuals) contained in the variable $count of *sort* number_of_lines. This individual consists of the count function applied to the length property of the *sort* lines. The function count is pre-defined in the *sortal* interpreter but, otherwise, could be specified as shown in the comment (preceded by '//'). A function always applies to the property of a *sort*. In this case, the exact property doesn't matter as its value is not actually used in the calculation of the result of the count function. The length property of a *sort* with Label as *sortal* template specifies the length—the number of characters—of the corresponding label. The result is:

```
  ind $count = number_of_lines: count(lines.length) =
2.0
  { "l1"
    { #me-stops-1 "s1",
      #me-stops-2 "s2",
      #me-stops-3 "s3" },
    "l2"
    { #me-stops-1 "s1",
      #me-stops-3 "s3",
      #me-stops-4 "s4" } };
```

Similarly, in order to calculate the number of stops per line, we can apply the function count to the length property of the *sort* stops. We reuse the *sort* number_of_lines for now.

```
  ind $count = number_of_lines: count(stops.length)
$output;
```

However, the result will be incorrect as stops belonging to multiple lines will be counted as many times. In order to correct the result, we need to alter the location of the count function in the representational structure to be an attribute of the *sort* lines. We can achieve this simply by creating a new *sort* and relying on the automatic conversion of one data form (or individual) into another.

```
    sort number_of_stops: lines ^ counts ^ stops;
    form $stops_per_line = number_of_stops: $count;
```

The result is:

```
form $stops_per_line = number_of_stops:
{ "l1"
  { 3.0
    { #me-stops-1 "s1",
      #me-stops-2 "s2",
      #me-stops-3 "s3" } },
  "l2"
  { 3.0
    { #me-stops-1 "s1",
      #me-stops-3 "s3",
      #me-stops-4 "s4" } } };
```

Sortal Grammars

Grammars are formal devices for specifying languages. A grammar defines a language as the set of all objects generated by the grammar, where each generation starts with an initial object and uses rules to achieve an object that contains only elements from a terminal vocabulary. A rewriting rule has the form *lhs* → *rhs*; *lhs* specifies the similar object to be recognized, *rhs* specifies the manipulation leading to the resulting object. A rule applies to a particular object if the *lhs* of the rule 'matches' a part of the object under some allowable transformation. Rule application consists of replacing the matching part by the *rhs* of the rule under the same transformation. In other words, when applying a rule $a \rightarrow b$ to an object s under a transformation f such that $f(a) \leq s$, rule application replaces $f(a)$ in s by $f(b)$ and produces the shape s—$f(a)+f(b)$. The set F of valid transformations is dependent on the object type. In the case of geometric entities, the set of valid transformations, commonly, is the set of all Euclidean transformations, which comprise translations, rotations and reflections, augmented with uniform scaling. In the case of textual entities, or labels, case transformations of the constituent letters may constitute valid transformations.

The central problem in implementing grammars is the matching problem, that of determining the transformation under which the match relation holds for the *lhs*. Clearly, this problem depends on the representation of the elements of the algebra. *Sorts* offer a representational flexibility where each *sort* additionally specifies its own match relation as a part of its behavior. For a given *sort*, a rule can be specified as a composition of two data forms, a *lhs* and a *rhs*. This rule applies to any particular data form if the *lhs* of a rule is a part of the data form under any applicable transformation f, corresponding to the behavioral specification of the data form's *sort*. Rule application results in the subtraction of $f(lhs)$ from the data form, followed by the addition of $f(rhs)$ to the result. Both operations are defined as part of the behavioral specification of a *sort*.

As composite *sorts* derive their behavior from their component *sorts*, the technical difficulties of implementing the matching problem only apply once for each primitive *sort*. As the part relationship can be applied to all kinds of data types, recognition algorithms can easily be extended to deal with arbitrary data representations, considering a proper definition of what constitutes a transformation. Correspondingly, primitive *sorts* can be developed, distributed, and adopted by users without any need for reconfiguring the system. At the same time, the appropriateness of a given grammar formalism for a given problem can easily be tested, the formalism correspondingly adapted, and existing grammar formalisms can be modified to cater for changing requirements or preferences.

The specification of spatial rules and grammars leads naturally to the generation and exploration of possible designs; spatial elements emerging under a part relation is highly enticing to design search [8, 9]. However, the concept of search is more fundamental to design than its generational form alone might imply. In fact, any mutation of an object into another, or parts thereof, can constitute an action of search. As such, a rule can be considered to specify a particular compound operation or mutation, that is, a composition of operations and/or transformations that is recognized as a new, single, operation and applied as such. Similarly, the creation of a grammar is merely a tool that allows a structuring of a collection of rules or operations that has proven its applicability to the creation of a certain set (or language) of designs.

Sortal Behaviors

The simplest specification of a part relationship corresponds to the subset relationship in mathematical sets. Such a part relationship applies to points and labels, e.g., a point is part of another point only if they are identical, and a label is a part of a collection of labels only if it is identical to one of the labels in the collection. Here, *sortal* operations of addition, subtraction, and product correspond to set union, difference, and intersection, respectively. In other words, if x and y denote two data forms of a *sort* of points (or labels), and X and Y denote the corresponding sets of data elements, i.e., sets of points (or labels), then (x: X specifies X as a representation of x)

$$\begin{aligned} x:X \wedge y:Y \Rightarrow x''y &\Leftrightarrow X \subseteq Y \\ x+y&:X \bigcup Y \\ x-y&:X\,/\,Y \\ x\cdot y&:X \bigcap Y \end{aligned}$$

An alternative behavior applies to weights (e.g., line thicknesses or surface tones) as is apparent from drawings on paper—a single line drawn multiple times, each time with a different thickness, appears as if it were drawn once with the largest thickness, even though it assumes the same line with other thicknesses (see also

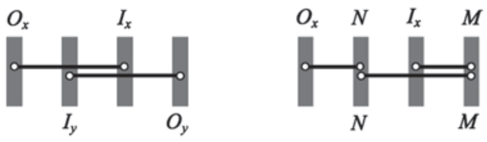

Fig. 2 The specification of the boundary collections I_x, O_x, I_y, O_y, M and N, given two data forms of intervals x (*above*) and y (*below*)

[11]). When using numeric values to represent weights, the part relation on weights corresponds to the less-than-or-equal relation on numeric values;

$$\begin{aligned} x:\{m\} \wedge y:\{n\} \Rightarrow x'' y \Leftrightarrow m'' n \\ x+y:\{\max(m,n)\} \\ x-y:\{\}\,\textit{if}\; m'' n, \text{else}\{m\} \\ x\cdot y:\{\min(m,n)\} \end{aligned}$$

Thus, weights combine into a single weight, with its value as the least upper bound of the respective individual weights, i.e., their maximum value. Similarly, the common value (intersection) of a collection of weights is the greatest lower bound of the individual weights, i.e., their minimum value. The result of subtracting one weight from another depends on their relative values and is either the first weight, if it is greater that the second weight, or zero (i.e., no weight).

Another kind of part relationship corresponds to interval behavior. Consider, for example, the specification of a part relationship on line segments. A line segment may be considered as an interval on an infinite line (or carrier); in general, one-dimensional quantities, such as time, can be treated as intervals. An interval is a part of another interval if it is embedded in the latter; intervals on the same carrier that are adjacent or overlap combine into a single interval. Specifically, a behavior for intervals can be expressed in terms of the behavior of the boundaries of intervals. Let $B[x]$ denote the boundary of a data form x of intervals and, given two data forms x and y let I_x denote the collection of boundaries of x that lie within y, O_x denote the collection of boundaries of x that lie outside of y, M the collection of boundaries of both x and y where the respective intervals lie on the same side of the boundary, and N the collection of boundaries of both x and y where the respective intervals lie on opposite sides of the boundary (Fig. 2) [12]. Then,

$$\begin{aligned} x:B[x] \wedge \mathrm{y}:B[y] \Rightarrow x'' y \Leftrightarrow I_x = 0 \wedge O_y = 0 \wedge N = 0 \\ x+y:B[x+y] = O_x + O_y + M \\ x-y:\mathrm{B[x-y]} = O_x + I_y + N \\ x\cdot y:B[\mathrm{x}\cdot y] = I_x + I_y + M \end{aligned}$$

This behavior applies to indefinite intervals too, providing that there is an appropriate representation of both (infinite) ends of its carrier. Likewise, behaviors can be specified for area intervals (plane segments) and volume intervals (polyhedral segments). The equations above still apply though the construction of I_x, O_x, I_y, O_y, M, and N is more complex [12].

Exemplar Grammar Systems

A uniform characterization for a variety of grammar systems is given in [1]. Krishnamurti and Stouffs [13] survey a variety of spatial grammar formalisms from an implementation standpoint. Here, we consider the specification of some of these examples using *sorts*.

Structure Grammars

Structure grammar is an example of a set grammar. "A structure is a symbolic representation of parts and their relationships in a configuration" [3]. A *structure* is represented as a set of pairs, each consisting of a symbol, e.g., a spatial icon, and a transformation. The resulting algebra corresponds to the Cartesian product of the respective algebras for the set of symbols and the group of transformations. Both symbols and transformations define *sorts* with discrete behavior, i.e., respective sets match under the subset relationship. These combine into a composite *sort* under the attribute relationship; each symbol in a set may have one or more transformations assigned as an attribute.

```
sort symbols : [ImageUrl];
sort transformations : [Transformation];
sort structures : symbols ^ transformations;
```

The *sort* symbols is specified to use the *sortal* template ImageUrl, a variant on the template Label that allows the label to be treated as a URL pointing towards an image that can be downloaded and displayed.

Tartan Worlds

Tartan Worlds [14] is a spatial grammar formalism that bestrides string and set grammars. We consider a simplified string grammar version of the *Tartan Worlds*: each symbol in a string corresponds to a geometrical entity represented as a graphical icon and located on a grid. A rule in these *simplified Tartan Worlds* [13] consists of one symbol on the *lhs* and symbols on the *rhs* given in their spatial relation. An equivalent *sortal* grammar may be defined over a *sort* composed over a grid of a *sort* of graphical icons. On a fixed-sized grid, the behavior of the composite *sort* breaks down into the behavior of the *sort* of graphical icons, e.g., ordinal or discrete, over each grid cell. The matching relation is defined in the same way.

```
        sort icons : [ImageUrl];
        sort tartan_worlds : icons {30, 20};
```

Again, the *sort* icons is specified to use the *sortal* template ImageUrl, The *sort* tartan_worlds is defined as a composition of the *sort* icons over a fixed-size grid, similar to a two-dimensional array, of 30 by 20.

Augmented Shape Grammars

A *shape* [1] is defined as a finite arrangement of spatial elements from among points, lines, planes, or volumes, of limited but non-zero measure. A shape is a part of another shape if it is embedded in the other shape as a smaller or equal element; shapes adhere to the maximal element representation [15, 16]. Shapes of the same dimensionality belong to the same algebra; these define a *sort*. A shape consisting of more than one type of spatial elements belongs to the algebra given by the Cartesian product of the algebras of its spatial element types. The respective *sorts* combine under the operation of sum, as a disjunctive composition.

A shape can be augmented by distinguishing spatial elements, e.g., by labeling, weighting, or coloring these elements. Augmented shapes also specify an algebra as a Cartesian product of the respective shape algebra and the algebra of the distinguishing attributes. However, the resulting behavior can better be expressed with a *sort* that is a subordinate composition of the respective *sorts*, i.e., combined under the attribute operator. A *sort* of labels may adhere to a discrete behavior, a *sort* of weights to an ordinal behavior; a weight matches another weight if it has a smaller or equal value.

Most shape grammars only allow for line segments and labeled points:

```
  sort line_segments : [LineSegment];
  sort labeled_points : (points : [Point]) ^ (labels :
[Label]);
  sort shapes : line_segments + labeled_points;
```

Sortal Rules

When considering a simple *sortal* grammar, the grammar rules can all be specified within the same *sort* as defined for the grammar formalism. In the case of more complex *sortal* grammars, or when the grammar formalism may change or develop over time, it may be worthwhile to consider grammar rules that are specified within a different *sort*, for example, a simpler *sort* or a previously adopted *sort*, without having to rewrite these to the *sortal* formalism currently adopted. *Sortal* grammar formalisms support this through the subsumption relationship over *sorts*. This subsumption relationship underlies the ability to compare *sortal* representations. and assess data loss when exchanging data from one *sort* to another. When a representation subsumes another, the entities represented by the latter can also be represented by the former representation, without any data loss.

Under the disjunctive operation of sum, any entity of the resulting *sort* is necessarily an entity of one of the constituent sorts. *Sortal* disjunction consequently defines a subsumption relationship on *sorts* (denoted '$\leq$'), as follows:

$$a \leq b \Leftrightarrow a + b = b;$$

a disjunctive *sort* subsumes each constituent *sort*.

Most logic-based formalisms link subsumption directly to information specificity, that is, a structure is subsumed by another, if this structure contains strictly more information than the other. The subsumption relationship on *sorts* can also be considered in terms of information specificity, however, there is a distinction to be drawn in the way in which subsumption is treated in *sorts* and in first-order logic based representational formalisms. First-order logic formalisms generally consider a relation of inclusion (hyponymy relation), commonly denoted as an is-a relationship. *Sorts*, on the other hand, consider a part-of relationship (meronymy relation).

Two simple examples illustrate this distinction. Consider a disjunction of a *sort* of points and a *sort* of line segments; this allows for the representation of both points and line segments. We can say that the *sort* of points forms part of the *sort* of points and line segments—note the part-of relationship. In first-order logic, this corresponds to the union of points and line segments. We can say that both are bounded geometrical entities of zero or one dimensions—note the is-a relationship.

This distinction becomes even more important when we consider an extension of *sortal* subsumption to the attribute operator. Consider a *sort* cost_types as a composition under the attribute relationship of a *sort* types with template Label and a sort costs with template Weight:

```
  sort  cost_types  :  (types  :  [Label])  ^  (costs  :
[Weight])
```

For example, these cost values may be specified per unit length or surface area for building components. If we lessen the conjunctive character of the attribute operator by making the cost attribute entity optional, then, we can consider a type label to be a cost type without an associated cost value or, preferably, a type label to be part of a cost type, that is, the sort of types is part of the sort of cost types. Vice versa, the sort of cost types subsumes the sort of types or, in general:

$$a \leq a \wedge b$$

In logic formalisms, a relational construct is used to represent such associations. For example, in description logic [17], roles are defined as binary relationships between concepts. Consider a concept Label and a concept Color; the concept of colored labels can then be represented as Label ∩ ∃ `hasAttribute.Color`, denoting those labels that have an attribute that is a color. Here, ∩ denotes intersection and ∃R.C denotes full existential quantification with respect to role R and concept C. It follows then that `Label` ∩ ∃ `hasAttribute.Color` ⊆ `Label`; that is, the concept of labels subsumes the concept of colored labels—this is quite the reverse of how it is considered in *sorts*.

As such, a shape rule specified for a *sort* of line segments and labeled points remains applicable if we extend the formalism to include plane segments or even volumes (if all considered in three dimensions). Similarly, the shape rule would still apply if we adapt the formalism to consider colored labels as attributes to the points, or line segments for that matter.

Another important distinction is that first order logic-based representations generally make for an open world assumption—that is, nothing is excluded unless it is done so explicitly. For example, shapes may have a color assigned. When looking for a yellow square, logically, every square is considered a potential solution—unless, it has an explicitly specified color, or it is otherwise known not to have the yellow color. The fact that a color is not specified does not exclude an object from potentially being yellow. As such, logic-based representations are automatically considered to be incomplete. *Sorts*, on the other hand, hold to a closed world assumption. That is, we work with just the data we have. A shape has a color only if one is explicitly assigned: when looking for a yellow square, any square will not do; it has to have the yellow color assigned. This restriction is used to constrain the application of grammar rules, as in the use of labeled points to constrain the application of shape grammar rules. Another way of looking at this distinction between the open or closed world assumptions is to consider their applicability to knowledge representation. To reiterate, logic-based representations essentially represent knowledge; *sorts*, on the other hand, are intended to represent data—any reasoning is based purely on present (or emergent) data.

Urban Design Grammars

Beirão, Duarte and Stouffs [18] present components of an urban design grammar inferred from an extension plan for the city of Praia in Cabo Verde (Fig. 3). The development of the urban grammar forms part of a large research project called City Induction aiming at integrating an urban program formulation model [19], a design generation model [20] and an evaluation model [21] in an 'urban design tool'. The central idea to the project is to read data from the site context on a GIS platform, generate program descriptions according to the context conditions, and from that program generate alternative design solutions guided by evaluation processes in order to obtain satisfactory design solutions. The generation part considers urban grammars in an extension of the discursive grammar schema developed by Duarte [22] and adapted for urban design. The generation of urban designs is led by the selection and application of urban patterns (denoted Urban Induction Patterns), each formalized as a discursive grammar. The application of an urban pattern—or discursive grammar—involves the application of urban design rules codified in a spatial grammar according to the requirements of the urban program codified in a description grammar [23]. The spatial grammar may manifest itself as an augmented shape grammar, allowing for various attribute data to be associated with the graphical elements, but may also include a raster-based string or set grammar, in order to allow rule-based operations on both vectorized and rasterized GIS data.

Acknowledgments The author wishes to thank Ramesh Krishnamurti for his contributions to the *sortal* research, and José Beirão and José Duarte for their collaboration on urban design grammars.

References

1. Stiny G (1980) Introduction to shape and shape grammars. Environ Plan B: Plan Des 7:343–351
2. Stiny G (1981) A note on the description of designs. Environ Plan B: Plan Des 8:257–267
3. Carlson C, McKelvey R, Woodbury RF (1991) An introduction to structure and structure grammars. Environ Plan B: Plan Des 18:417–426
4. Duarte JP, Correia R (2006) Implementing a description grammar: generating housing programs online. Constr Innov: Inf Process Manag 6(4):203–216
5. Duarte JP (2005) A discursive grammar for customizing mass housing: the case of Siza's houses at Malagueira. Autom Constr 14:265–275
6. Stouffs R, Krishnamurti R (2001) Sortal grammars as a framework for exploring grammar formalisms. In: Burry M, Datta S, Dawson A et al. (eds) Mathematics and design 2001. The School of Architecture & Building, Deakin University, Geelong, pp 261–269
7. Stouffs R (2008) Constructing design representations using a sortal approach. Adv Eng Informatics 22(1):71–89
8. Mitchell WJ (1993) A computational view of design creativity. In: Gero JS, Mahel ML (eds) Modeling creativity and knowledge-based creative design. Erlbaum, Hillsdale
9. Stiny G (1993) Emergence and continuity in shape grammars. In: Flemming U, Van Wyk S (eds) CAAD futures 1993. North-Holland, Amsterdam, pp 37–54

10. Gips J, Stiny G (1980) Production systems and grammars: a uniform characterization. Environ Plan B: Plan Des 7: 399–408
11. Stiny G (1992) Weights. Environ Plan B: Plan Des 19: 413–430
12. Krishnamurti R, Stouffs R (2004) The boundary of a shape and its classification. J Design Res 4(1)
13. Krishnamurti R, Stouffs R (1993) Spatial grammars: motivation, comparison and new results. In: Flemming U, Van Wyk S (eds) CAAD Futures '93. North-Holland, Amsterdam, pp 57–74
14. Woodbury RF, Radford AD, Taplin PN et al. (1992) Tartan worlds: a generative symbol grammar system. In: Noble D, Kensek K (eds) ACADIA '92
15. Krishnamurti R (1992) The maximal representation of a shape. Environ Plan B: Plan Des 19:585–603
16. Stouffs R (1994) The algebra of shapes. Ph.D. dissertation. Department of Architecture, Carnegie Mellon University, Pittsburgh
17. Baader F, Calvanese D, McGuinness D et al (2003) The description logic handbook: theory, implementation and applications. Cambridge University, Cambridge
18. Beirão J, Duarte J, Stouffs R (2009) An urban grammar for Praia: towards generic shape grammars for urban design. In: Computation: The new realm of architectural design. Istanbul Technical University, Istanbul, pp 575–584.
19. Montenegro NC, Duarte JP (2008) Towards a computational description of urban patterns. In: Muylle M (ed) Architecture 'in computro', integrating methods and techniques. eCAADe and Artesis Hogeschool Antwerpen, Antwerp, pp 239–248
20. Beirão J, Duarte J, Stouffs R (2008) Structuring a generative model for urban design: linking GIS to shape grammars. In: Muylle M (ed) Architecture 'in computro', integrating methods and techniques. eCAADe and Artesis Hogeschool Antwerpen, Antwerp, pp 929–938
21. Gil J, Duarte JP (2008) Towards an urban design evaluation framework. In: Muylle M (ed) Architecture 'in computro', integrating methods and techniques. eCAADe and Artesis Hogeschool Antwerpen, Antwerp, pp 257–264
22. Duarte JP (2005) A discursive grammar for customizing mass housing: the case of Siza's houses at Malagueira. Autom Constr 14(2):265–275
23. Stiny G (1981) A note on the description of designs. Environ Plan B: Plan Des 8(3):257–267