Knowledge and Information Systems (2023) 65:3995–4020
https://doi.org/10.1007/s10115-023-01879-6

REGULAR PAPER

Exploiting spatial relations for grammar-based specification of multidimensional languages

Giuseppe Della Penna[1] · **Sergio Orefice**[1] · **Andrea D'Angelo**[1]

Received: 24 January 2022 / Revised: 13 April 2023 / Accepted: 16 April 2023 /
Published online: 9 May 2023

Abstract
As opposed to textual programming languages, multidimensional languages compiler construction paradigms lack standardization. To this aim, in this paper we present the spatial grammar (SG) formalism, a grammar model for multidimensional languages which has string-like productions containing more general spatial relations other than string concatenation, and we provide mapping rules to translate an SG specification into a translation schema. In this way, the SG formalism inherits and extends to the multidimensional context concepts and techniques of standard compiler generation tools like YACC.

Keywords Spatial knowledge · Multidimensional languages · Spatial relations · Context-free grammars

1 Introduction

In the last decades, much research has been done to model spatial relations and a number of formalisms have been provided (see Sect. 2 for a brief survey) since, in general, spatial composition rules are fundamental in representing spatial knowledge and in designing visual systems, because a formal description of their structural characteristics is crucial to provide a systematic base and avoid ad hoc implementations. In [10], we have proposed a framework which uses a qualitative approach to represent spatial information and includes common qualitative spatial relations such as topological (e.g. overlapping, adjacency, containment) and directional (e.g. left, up) relations. Such formalism evolved in the PCT (*Position-Connection-Time*) framework [11] that now integrates *position*, *connection* and *spatio-temporal* relations. In particular, although most of the literature on spatial relations commonly relies on relations based on the position of the involved objects, PCT also includes spatial relations where the specific position of the objects is not relevant, as it happens in diagrammatic visual languages. This reflects the two basic modalities that can be used to compose graphical objects:*position-*

✉ Giuseppe Della Penna
Giuseppe.DellaPenna@univaq.it

[1] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy

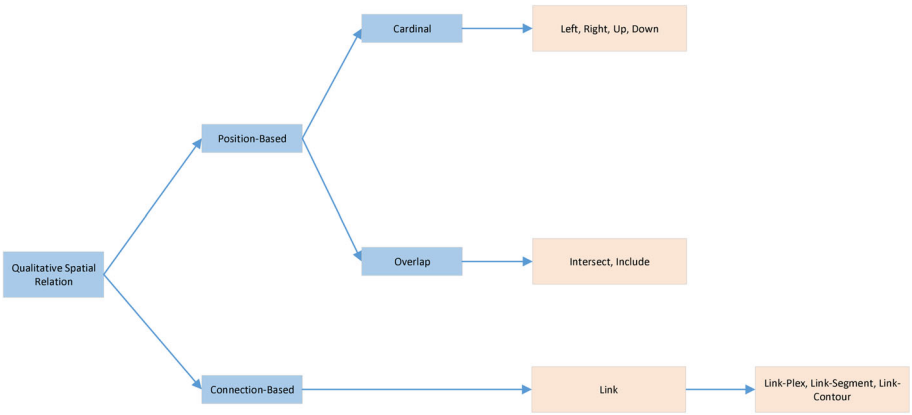


Fig. 1 A fragment of the PCT spatial relation hierarchy

based or*connection-based* that is by spatially arranging or by connecting them. So far, PCT relations have been profitably exploited in several domains, e.g. to support visual information extraction from visual documents (e.g. web pages, maps or biomedical images), [9], and to specify touch screen gestures [12].

In this paper, we apply PCT to the visual language theory and use its spatial relations as basis to define a grammar formalism, namely SG (*spatial grammar*), for the specification of the syntax of *multidimensional languages*, conceived as sets of *multidimensional sentences* composed of graphical objects arranged through spatial relations, as opposed to traditional formal languages which are defined as sets of strings (sentences) over an alphabet of textual symbols. SGs naturally extend context-free grammars for string languages by considering new relations in addition to string concatenation. In particular, Fig. 1 shows the fragment of the PCT spatial relation hierarchy that will be used within the SG formalism to describe the syntax of multidimensional languages.

Moreover, to avoid the efforts that must be accomplished in the recognition of multidimensional languages, we also present a methodology that allows to automatically translate SGs into equivalent context-free grammars with semantic actions (i.e. translation schemas, [1]), where such actions are calls to procedures that implement spatial relations. This allows to generate the corresponding parser through standard compiler-compiler techniques and tools like YACC [23].

In the practice, grammars for multidimensional languages, and in particular the proposed spatial grammars, can be effective in several fields spanning from *computer vision* to *image analysis* where, for example, they can assist or even perform better than machine learning. As an example, shape recognition is usually performed as a multi-step process including capture, pre-processing, feature extraction and recognition. Rule-based approaches (such as grammar-based ones) may help, for example, in the recognition phase when a machine learning approach has insufficient training data [5]. Indeed, in the case study presented in Sect. 6, spatial grammars are used as an highly formalized, human-readable and easily processable set of image (de)composition rules to assist traffic signs recognition, which is a widely studied problem (see, for example, the recent works in [33]).

Moreover, the image recognition performed by spatial grammars is *syntactic*, since they detect the image elements and their relations, and thus, the image *semantic* is derived through *syntactic reasoning* as it happens, i.e. for the code written in a specific programming language.

This may be also very important when an higher degree of precision is required and/or there is the need to further reason on the image structure after recognizing its overall shape as, for example, in biomedical applications [25].

On the other hand, most machine learning approaches perform image recognition "by similarity" with respect to a training set, usually without classification *explainability* [4], which can be very important in critical contexts, whereas the derivation of a spatial grammar represent a formal, detailed explanation of the recognition process and its results.

Despite these advantages, often grammar-based approaches are not widely employed due to their computational *complexity* (especially for highly expressive grammar formalisms, e.g. graph grammars), if compared to data-based (statistical, machine learning, etc.) and less powerful rule-based approaches based on simpler or *ad hoc* formalisms. For this reason in the present paper the main focus is the ability to *easily handle the grammars at application level*, possibly reusing *classical tools* for their recognition, making them actually usable in many contexts.

The paper is organized as follows. Section 2 contains a brief related work focused on the spatial relation literature. In Sect. 3, we recall the basic concepts from the PCT framework, in particular the definition of the spatial relations that will be used in the paper. Section 4 illustrates the SG formalism showing how these grammars extend common context-free grammars to multidimensional languages. Then, in Sect. 5 we describe how to convert SGs in equivalent translation schemas in order to use standard compiler generation tools for their recognition, and in Sect. 6 we apply this methodology to a simple but significant real-world case study. Finally, concluding remarks and further research issues are outlined in Sect. 7.

2 Related work

To begin, let us briefly describe the main qualitative spatial relation formalisms present in the literature. Further references to topological and direction spatial relation formalisms can be found, for example, in the extended survey presented in [6].

Commonly, such formalisms are classified in two main classes, i.e. *topological* and *direction*. Topological relations describe qualitative spatial relations that are invariant under topological transformations such as translation, rotation and scaling.

The *N-intersection* theory is one of the most widely recognized approach in this class. In this framework, a region x is associated with three related point sets: the region *interior* x°, its *boundary* ∂x and its *exterior* x^-. Then, a relationship between any two regions x and y can be characterized by a matrix defining the intersections between each pair of the sets x°, ∂x, x^- and y°, ∂y, y^-. Many formalisms derived from the N-intersection theory have been presented, but the most well-known is the *9-Intersection Model* (9-IM, see, for example, [7, 14, 15]), which can be used to define the relationships between all combinations of lines, points and regions.

Also the *Region Connection Calculus* (RCC, see, for example, [30]), is used to describe topological relations but, unlike the 9-intersection approach, takes regions rather than points as a fundamental notion. Relations are defined in the RCC formalism starting from the base *connected* relationship $C(x, y)$, which holds if and only if the regions x and y share a common point. A very large number of relations can be derived from $C(x, y)$, e.g. the eight relations *disconnected*, *equal*, *partially overlapping*, *externally connected*, *tangential proper part* (with its inverse) and *non-tangential proper part* (with its inverse), which form the well-known *RCC8* formalism.

On the other hand, direction relations still describe where an object is placed relative to another one, but without considering situations where the two objects overlap. Depending on the dimension of the objects involved, direction formalisms can be divided into *point-based*, where objects are simplified into points, or *extended-object*, where object have a shape.

Two well-known examples of point-based direction calculi are the *Oriented Point Relation Algebra* (OPRA, see, for example, [26, 28]), where the notion of point is extended to *oriented point* and the *Ternary Point Configuration Calculus* (TPCC, see, for example, [27]), which uses *ternary relations*, in contrast with the more common binary ones. Extended-object based calculi are more complex, due to the involved object shapes. Thus, to simplify the process, *minimal bounding rectangles* are often used as an approximation of the actual objects.

The *Cardinal Direction Calculus* [34] is the most well-known binary direction relation calculus. An arbitrary basic CDC relation is a binary relation involving a target object and a reference object, and a non-empty subset of the nine atomic relations N, NW, NE, W, O, E, S, SW and SE, corresponding to the possible intersections of sub regions of the target object with the 3×3 *direction matrix* [21] of the reference object.

Another well-known binary direction relation calculus is the *Rectangle Algebra* (RA) [3], an extension of the *Interval Algebra* [2]. Objects in this formalism are restricted to be *rational rectangles*, i.e. rectangles whose sides are parallel to the axes of some orthogonal basis in a 2-dimensional Euclidean space. Relations between these objects are the 13×13 pairs of atomic relations which can hold between two rational intervals and can be used to express directional relations but also topological relations such as disjoint and overlap. Spatial information is represented by *spatial constraint networks*, i.e. constraint satisfaction problems where variables represent rational rectangles and constraints are relations. This model, though restrictive, is sufficient for applications in domains like architecture or GIS.

Multidimensional languages have been addressed in various forms in the literature. Roughly speaking, such kind of languages introduce further relations between symbols other than the usual sequential concatenation. Actually, most of the works address two-dimensional languages, such as *picture languages* [19], whereas our proposal extends to a higher number of dimensions, since the PCT framework supports 2-D spatial relations as well as connection relations (see Sect. 3), which are recalled in this paper, and also 3-D spatial relations and temporal relations, although not yet addressed in our current grammar formalism.

One of the most general research works in this field is [8], where the authors define *relation grammars* for specifying the syntax of visual languages and, more generally, of multidimensional languages. The paper has some similarities with the approach presented in this paper, notably both have context-free grammars as the starting point, and however, the spatial grammar formalism defined here, being based on the already published PCT framework, has a more well-defined basis and results simpler in the syntax, closer to the common grammar format and thus easier to manipulate with standard tools.

On the other hand, several grammar formalisms such as graph grammars [16] have been proposed in the literature for the specification of multidimensional languages. However, studies on such formalisms have been focused on their expressive power, whereas their practical uses, when applied to real-world graph-like data, result in excessive complexity (see, for example, [17]). On the other hand, in this paper the main focus is the ability to easily handle the grammars at application level, possibly reusing classical tools for their recognition. For this reason, the grammar formalism presented in this paper is derived from the usual context-free string grammars, which are well known and supported by a variety of out-of-the-box software tools.

Finally, it is worth noting that the term spatial grammar has already been used in the literature. For example, [24] discusses spatial and shape grammars (not necessarily context-

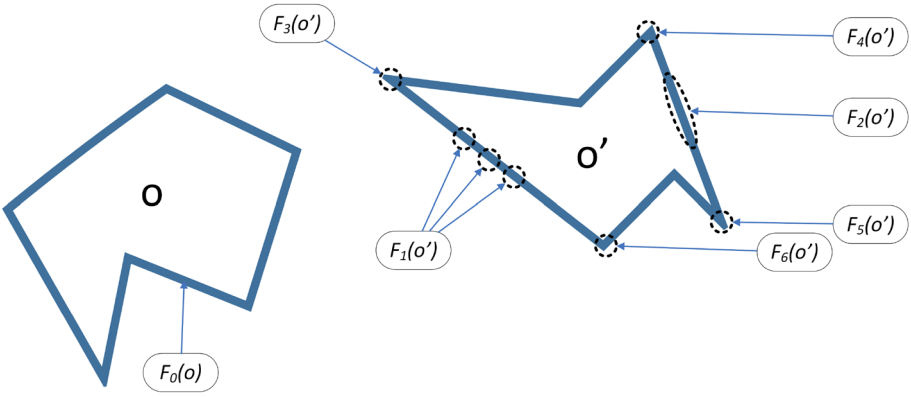


Fig. 2 Sample graphical objects

free) from an engineering point of view, but in a very generic context where the concept of spatial relation in not detailed as in the PCT and, again, no supporting software is presented.

3 Background: PCT

In this section, we recall the basic notions concerning the PCT formalism, i.e. graphical objects and spatial relations. An extensive characterization of the overall framework is given in [10, 11].

3.1 Graphical objects

In the PCT framework, a graphical object is formalized as follows.

Definition 3.1 (*Graphical object*) A *graphical object* is formally defined as pair $o = (C, A)$, where

- C denotes the set of the points $p \in \mathbb{R}^2$ forming the external *contour* of o (which is disjoint from its internal area), providing its shape.
- A is the set of the object *attributes*. Each attribute is a pair (n, v), where n is a property name and v its value.

The contour C is used essentially for the spatial manipulation of the object. In particular, C includes one or more *feasible regions*, namely $F_1(C), \ldots, F_k(C)$, defined as follows.

Definition 3.2 (*Contour feasible region*) Given a contour C, a *feasible region* of C, denoted by $F_i(C)$, is set $F_i(C) \subseteq C$.

Feasible regions represent subsets of points on the contour where the spatial relations are used in order to arrange graphical objects. As an abuse of notation, in the following we shall write $F_i(o)$ to indicate the region $F_i(C)$. For example, Fig. 2 shows a graphical object o with the overall contour as feasible region (conventionally denoted by $F_0(o)$) and a graphical object o' with particular points or parts of the contour as feasible regions.

Fig. 3 Disjoint cardinal spatial relations

$o\ UP\ o'$	$\Leftrightarrow$	$y_{UM(o')} \leqslant y_{DM(o)}$
$o\ LEFT\ o'$	$\Leftrightarrow$	$x_{RM(o)} \leqslant x_{LM(o')}$
$o\ DOWN\ o'$	$\Leftrightarrow$	$y_{UM(o)} \leqslant y_{DM(o')}$
$o\ RIGHT\ o'$	$\Leftrightarrow$	$x_{RM(o')} \leqslant x_{LM(o)}$

On the other hand, the attribute set A models non-spatial properties, i.e. visual or semantic aspects like *name*, *colour*, *font* or *latitude*. Actually, spatial properties (e.g. the object width, area) can be derived from the contour C and then are not included in A.

Note that PCT refers to graphical objects in $\mathbb{R}^2$ that are *regular*, in the sense that they do not contain *holes* and have a contour that can be modelled as a *closed curve* without self-loops (*simple curve*). More details on this issue are explained in [10]. However, this restriction defines a wide and significant domain, excluding only graphical objects with very irregular shapes, which would be hard to handle and, often, of little practical interest.

3.2 Spatial relations

In the PCT framework, spatial relations are defined according to the following classification.

3.2.1 Cardinal

The basic cardinal (disjoint) spatial relations, i.e. *LEFT*, *UP*, *RIGHT* and *DOWN*, are defined by means of four single-point feasible regions, namely $F_1(o) = UM(o)$, $F_2(o) = DM(o)$, $F_3(o) = LM(o)$ and $F_4(o) = RM(o)$ that represent, respectively, the set of upmost, downmost, leftmost and rightmost points of the contour of the graphical object o.

In particular, if C denotes the contour of a graphical object o, then $UM(o)$ can be defined as $\left\{ p \in C | \forall p' \in C,\, p'_y \leq p_y \right\}$. The other regions $DM(o)$, $LM(o)$ and $RM(o)$ can be formalized similarly. Then, given two graphical objects o and o', the cardinal spatial relations are defined as in Fig. 3.

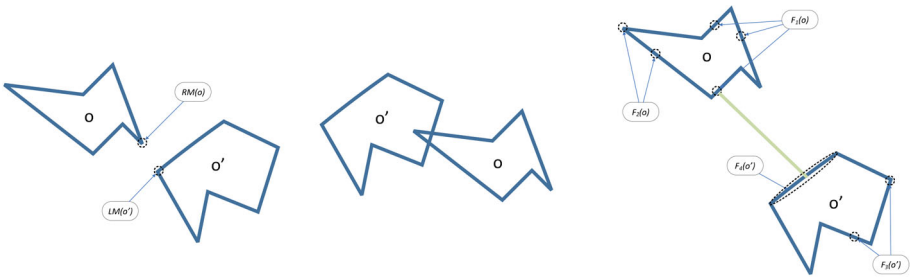
Note that these definitions refer to the canonical orientation of the Cartesian axes (i.e. the x coordinate increases rightwards, and the y one increases upwards). Moreover, they use the notation $y_{UM(o)}$ to denote the y coordinate common to all the upmost points in the feasible region $UM(o)$. The meaning of the other notations $x_{LM(o)}$, $y_{DM(o)}$ and $x_{RM(o)}$ can be defined analogously.

For example, Fig. 4a shows a spatial arrangement between o and o' (where the involved feasible regions have been outlined) corresponding to the spatial relation *LEFT*. Here, the *LEFT* relation models a spatial arrangement where the graphical object o is completely on the left of the graphical object o'.

3.2.2 Overlap

Overlap relations model spatial arrangements holding between graphical objects with intersecting contours or internal points (e.g. "full inclusion" or "partial intersection"). The main relations of this class are *INCLUDE* and *INTERSECT*, defined as in Fig. 5.

The first relation *INCLUDE* models a spatial arrangement between o and o' whenever all the internal points of o' are a subset of the internal points of o, i.e. o' is inside o. On the other



(a) *LEFT* (b) *INTERSECT* (c) *LINK*

Fig. 4 Examples of relations

o *INCLUDE* o'	$\Leftrightarrow$	$\text{Inside}(o') \subseteq \text{Inside}(o)$
o *INTERSECT* o'	$\Leftrightarrow$	$\text{Inside}(o) \cap \text{Inside}(o') \neq \varnothing$

Fig. 5 *INCLUDE* and *INTERSECT* spatial relations

o $LINK(F_i(o), F_j(o'))$ o'	$\Leftrightarrow$	$\exists k > 0$ $\lambda(F_i(o)) = \lambda(F_j(o')) = k$

Fig. 6 LINK connection spatial relation

hand, the *INTERSECT* relation holds whenever at least one of the internal points of o' is also an internal point of o, i.e. the two objects are (partially) overlapping.

For example, Fig. 4b shows a spatial arrangement between o and o' corresponding to the spatial relation *INTERSECT*.

Of course, the *INCLUDE* and *INTERSECT* relations, applied together with appropriate constraints, would be sufficient to express any other kind of spatial overlapping (see [10]).

3.2.3 Connection

Connection relations allow to relate graphical objects which are *logically* connected, regardless their position or spatial arrangement. Unlike cardinal or overlap relations, connection spatial relations are *explicit* relations, in the sense that they have a visual representation that conveys the link semantics. Conventionally, a link is represented by a polyline connecting two contour points of two graphical objects.

Formally speaking, the *LINK* relation is defined as in Fig. 6. This definition uses a value $\lambda(F_i(o)) \in \mathbb{N}$, which associates a number to each feasible region of a graphical object. This value is zero whether the region is not involved in any connection; otherwise, it is an integer greater than zero. Intuitively, two graphical objects o and o' are linked if two of their feasible regions share the same nonzero λ-value, i.e. $\exists i, j, k > 0, \lambda(F_i(o)) = \lambda(F_j(o')) = k$

An example of basic $LINK$ relation is shown in Fig. 4c, where the connection between two points of the regions $F_1(o)$ and $F_4(o')$ is explicitly depicted as a polyline between two arbitrary points of such regions.

In [10], the connection relations formalization includes further issues like the *link direction* or *constraints on the involved feasible regions* (i.e. single points, parts of the contour or

whole contours). In particular, three sub-types of connection spatial relations corresponding to the above constraints can be derived as shown in Fig. 7: in *LINK-PLEX*, the connected feasible regions are single points of the contour, in *LINK-SEGMENT*, they are contour segments (contiguous subsets of points), and in *LINK-CONTOUR*, they coincide with the overall contour of the objects.

4 Spatial grammars

Spatial grammars (SGs) extend context-free grammars including more general spatial relations other than the string concatenation, in order to specify multidimensional languages.

4.1 Definition of spatial grammar

Spatial grammars (SGs) preserve the classic components of common context-free grammars, in particular the string-like format of the right-hand side of the productions. Actually, in the paper, we conceive the term context-free as only referring to the form of grammar rules. A discussion on the meaning of context-freeness for high-dimensional languages is out of the scope of this work. This issue is addressed, for example, in [32].

As a novelty, spatial grammars are characterized by the following distinguishing elements:

- *grammar symbols* (terminals/non-terminals) become graphical objects as conceived in PCT. In particular, terminal graphical objects are the actual language objects forming the sentences, whereas non-terminal ones are fictitious objects representing fragments of the overall spatial arrangement modelled by a sentence;
- *spatial relations* are introduced in the right-hand side of the production to relate the grammar symbols;
- the *linearity* of the right-hand side of a production $X \rightarrow x_1 x_2 \dots x_n$ is broken by an appropriate use of spatial relations that are not forced to relate x_i with x_{i-1};
- *productions* can include *anchors* in order to define the properties of the left-hand non-terminal graphical object by suitably combining the properties of the right-hand symbols. In particular, anchors are used to derive the contour and the feasible regions of the left-hand graphical object from the ones of the right-hand objects.

In order to accomplish these issues, a spatial grammar has five components formalized as follows.

Definition 4.1 (*Spatial grammar*) A spatial grammar is a quintuple $SG = (N, T, S, P, R)$ where N is a finite set of *non-terminal graphical objects*, T is a finite set of *terminal graphical objects*, with $N \cap T = \emptyset$, $S \in N$ is the grammar *start symbol*, R is a finite set of *spatial relation* identifiers, and P is a finite set of *productions* of the form

$$X \rightarrow_I x_1 REL^{j_1}{}_1 x_2 \dots REL^{j_{n-1}}{}_{n-1} x_n$$

In each production,

- the head X is a non-terminal graphical object, and each x_i on the right-hand side is a graphical object belonging to $N \cup T$;
- each REL_i is a spatial relation identifier belonging to R, where the apex j_i means that the spatial relation REL_i holds between x_{i-j_i} and x_{i+1};
- the element I labelling the production arrow is the "anchor" of the production and has the form $I = (\{k_1, \dots, k_m\}, Fd_1, \dots, Fd_l)$, with $m \leq n$.

$o \;\; LINK\text{-}PLEX(F_i(o), F_j(o')) \;\; o'$	$\Leftrightarrow$	$o \;\; LINK(F_i(o), F_j(C')) \;\; o' \wedge$ $\exists a, a' \in [0,1],$ $F_i(o) = \{\gamma_o(a)\} \wedge$ $F_j(o') = \{\gamma_{o'}(a')\}$
$o \;\; LINK\text{-}SEGMENT(F_i(o), F_j(o')) \;\; o'$	$\Leftrightarrow$	$o \;\; LINK(F_i(o), F_j(C')) \;\; o' \wedge$ $\exists a, b, a', b' \in [0,1], a < b, a' < b',$ $F_i(o) = \{\gamma_o(x) \vert x \in [a,b]\} \wedge$ $F_j(o') = \{\gamma_{o'}(x) \vert x \in [a',b']\}$
$o \;\; LINK\text{-}CONTOUR \;\; o'$	$\Leftrightarrow$	$o \;\; LINK(F_0(o), F_0(o')) \;\; o'$

Fig. 7 Specialized connection spatial relations

The non-terminals of a spatial grammar are actually "fictitious" graphical objects, since they have no real visual representation, like it happens for terminal grammar symbols, but they are syntactically manipulated as formal graphical objects according to Definition 3.1. Therefore, we need a way to derive their contour and possibly their feasible regions: this is accomplished through the anchor I of a production, which describes how to relate its head X to the external.

In particular, in the anchor I, the expression $\{k_1, \ldots, k_m\}$ is a set of natural numbers pointing to the right-hand side grammar symbols and indicating that the contour of the non-terminal X is the convex hull of the objects $x_{k_1}, \ldots, x_{k_m}$ (that is, the minimum n-sided convex polygon that completely circumscribes such objects). Note that if this part of the anchor is omitted, we assume that it contains exactly all the indices of the right-hand grammar symbols.

On the other hand, each Fd_i is a *feasible region derivation* expression of the form $F_i(X) = F_j(x_{k_j})$. Such an expression means that X has a feasible region $F_i(X)$ which is the same as the feasible region $F_j(x_{k_j})$ of the grammar symbol x_{k_j}. Note that, for the sake of simplicity, if no feasible region derivation expression is present in the anchor, we assume that the feasible regions of X are exactly all the feasible regions of the right-hand side grammar symbols.

4.2 Language of a spatial grammar

To complete the characterization of the SG formalism, let us give some definitions extending the classic formal language theory, in order to get the notion of language generated by a spatial grammar $SG = (N, T, S, P, R)$.

Definition 4.2 (*One-step derivation*) Let $\alpha A \beta$ (with $\alpha, \beta \in (N \cup T \cup R)^*$, $A \in N$) be a string composed by terminal and non-terminal symbols and spatial relation identifiers, and $A \rightarrow_I \gamma$ a production. Then, $\alpha A \beta \Rightarrow \alpha[\gamma]_I \beta$ is a one-step derivation.

Definition 4.3 (*Derivation*) $\forall \alpha \in (N \cup T \cup R)^*$, let $\alpha \Rightarrow^* \alpha$. If $\alpha \Rightarrow^* \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \Rightarrow^* \gamma$.

Of course, during the derivation process, references to feasible regions of an expanded non-terminal symbol change, since they are mapped to those of the symbols deriving from that expansion, according to the feasible region derivation expressions in the corresponding production anchor. This aspect will be clarified in the examples at the end of this section.

Definition 4.4 (*Sentential form*) If $S \Rightarrow^* \gamma$, we call the string γ a *sentential form*.

Definition 4.5 (*Terminal sentential form*) Let γ be a sentential form. If each grammar symbol in γ is a terminal, we call γ a *terminal sentential form*.

A terminal sentential form models a family of spatial arrangements, and therefore of visual dispositions, satisfying the spatial relations included in the sentence. Indeed, a graphical object may be placed in a number of different ways with respect to another, still laying, for example, on its left.

Therefore, in order to map each terminal sentential form into a specific visual disposition, i.e. a *multidimensional sentence*, we need to introduce a further element, i.e. the *layout operator* $f_{R,p}$, which needs to know the specification of the spatial relations contained in R and the coordinates of a specific point p on the Cartesian plane that will be used as *starting point* for the object disposition.

Once applied to a terminal sentential form γ, $f_{R,p}(\gamma)$ produces a set $\{(o_i, (x_i, y_i))\}$ where $o_i \in T$ and (x_i, y_i) denotes a specific point on the Cartesian plane. This set includes exactly

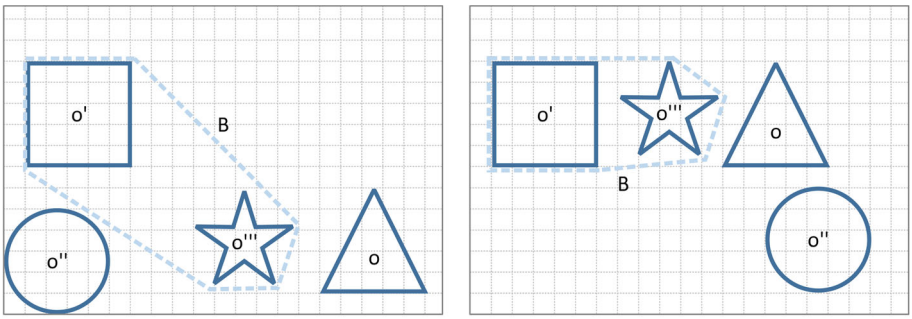


Fig. 8 Two possible visual dispositions for the terminal sentential form of Example 4.1

all the graphical objects o_i contained in γ, which are placed on the Cartesian plane so that $x_{LM(o_i)} = x_i$ and $y_{UM(o_i)} = y_i$ (i.e. roughly speaking, x_i and y_i represent the leftmost and uppermost coordinate of the object, respectively). The accomplished disposition of course satisfies the definition of the spatial relations in γ.

To conclude, the language generated by a spatial grammar SG, i.e. $L(SG)$, is the set of *multidimensional sentences* defined as follows.

Definition 4.6 (*Language of a SG*) Let $SG = (N, T, S, P, R)$ be a spatial grammar and $f_{R,p}$ a layout operator. The language generated by SG with respect to $f_{R,p}$ is $L(SG|f_{R,p}) = \{f_{R,p}(\gamma)|S \Rightarrow^* \gamma\}$

Note that, due to the presence of spatial relations into the spatial grammar productions, a derivation may lead to a terminal sentential form γ that the layout operator $f_{R,p}$ cannot convert into a multidimensional sentence since it is not possible to arrange the graphical objects of γ according to the involved spatial relations. We say that such derivation does not contribute to the language and, in that case, $f_{R,p}(\gamma)$ is undefined. Similarly to *useless* symbols in canonical context-free grammars, we call these derivations *useless derivations*. This issue will be better illustrated in Example 4.3.

Example 4.1 As a first example, let us consider the following spatial grammar productions:

$$S \rightarrow B\ LEFT\ o$$
$$B \rightarrow_{(\{1,3\})} o'\ UP\ o''\ LEFT^{-1}\ o'''$$

Applying two one-step derivations starting from the start symbol S, we obtain the following derivation:

$$\begin{aligned} S &\Rightarrow B\ LEFT\ o \\ &\Rightarrow \left[o'\ UP\ o''\ LEFT^{-1}\ o'''\right]_{(\{1,3\})}\ LEFT\ o \end{aligned}$$

Figure 8a, b shows two possible visual dispositions that may be represented by the final terminal sentential form $\gamma = \left[o'\ UP\ o''\ LEFT^{-1}\ o'''\right]_{(\{1,3\})}\ LEFT\ o$. Indeed, in each of them, o' is above o'', o' is on the left of o''', and both o', o''' (as required by the anchor $\{1, 3\}$) are on the left of o.

As explained above, applying a proper $f_{R,p}$, it is possible to obtain the unique corresponding multidimensional sentence contained in the language. For instance, in this case choosing a

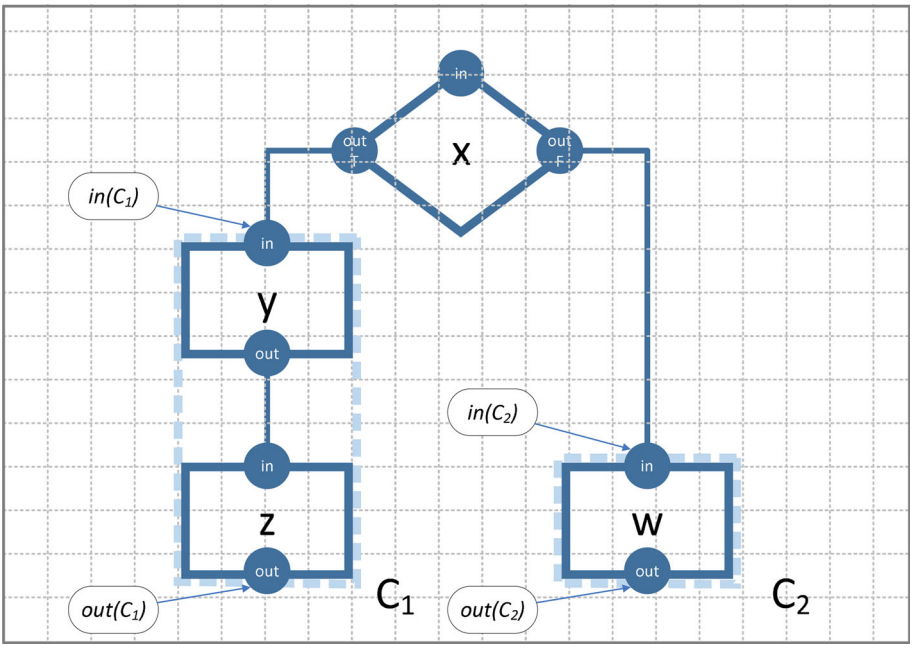


Fig. 9 Multidimensional sentence for Example 4.2

$f_{R,p}$ such that $f_{R,p}(\gamma) = \{(o, (12, 12)), (o', (1, 12)), (o'', (14, 6)), (o''', (8, 12))\}$ we obtain exactly the multidimensional sentence corresponding to the visual disposition shown in Fig. 8b.

Example 4.2 Let us consider another example including connection relations, referring to the following grammar and derivation:

$$
\begin{aligned}
S &\rightarrow x\ (UP \wedge LINK - PLEX(outT(x), in(C_1)))\ C_1 \\
&\quad\ (UP \wedge LINK - PLEX(outF(x), in(C_2)))^{-1}\ C_2 \\
C_1 &\rightarrow_I y\ (UP \wedge LINK - PLEX(out(y), in(z)))\ z \\
C_2 &\rightarrow w
\end{aligned}
$$

where $I = (in(C_1) = in(y), out(C_1) = out(z))$. Then,

$$
\begin{aligned}
S &\Rightarrow x\ (UP \wedge LINK - PLEX(outT(x), in(C_1)))\ C_1 \\
&\quad\ (UP \wedge LINK - PLEX(outF(x), in(C_2)))^{-1}\ C_2 \\
&\Rightarrow x\ (UP \wedge LINK - PLEX(outT(x), in(y))) \\
&\quad\ [y\ (UP \wedge LINK - PLEX(out(y), in(z)))\ z]_I \\
&\quad\ (UP \wedge LINK - PLEX(outF(x), in(C_2)))^{-1}\ C_2 \\
&\Rightarrow x\ (UP \wedge LINK - PLEX(outT(x), in(y))) \\
&\quad\ [y\ (UP \wedge LINK - PLEX(out(y), in(z)))\ z]_I \\
&\quad\ (UP \wedge LINK - PLEX(outF(x), in(w)))^{-1}\ w
\end{aligned}
$$

Applying a specific layout operator (which properly translates the link relations to common flowcharts-style polylines) to the final terminal sentential form of this derivation, we obtain the multidimensional sentence depicted in Fig. 9, which of course satisfies the constraint required by the involved spatial relations.

Example 4.3 To show an example of useless derivation, let us slightly modify the grammar fragment of Example 4.1 as follows.

$$S \rightarrow B\ LEFT\ o\ INCLUDE^{-1}h$$
$$B \rightarrow_{(\{1,3\})} o'\ UP\ o''\ LEFT^{-1}\ o'''$$

Again, starting from the start symbol S, we obtain the following derivation:

$$S \Rightarrow^* \left[o'\ UP\ o''\ LEFT^{-1}\ o'''\right]_{(\{1,3\})}\ LEFT\ o\ INCLUDE^{-1}h$$

Actually, this terminal sentential form does not model any semantically consistent spatial arrangement, since the object h cannot be included in the sub-expression $\left[o'\ UP\ o''\ LEFT^{-1}\ o'''\right]_{(\{1,3\})}$, whose contour includes two disjoint objects. Hence, the derivation that produced this terminal sentential form is useless and will not contribute to the language generated by the spatial grammar containing this fragment.

5 Automatic parser generation

Although multidimensional languages are clearly useful in the practice (to further clarify this point, we will introduce a real-world case study in Sect. 6), their recognition is, in general, computationally complex, and this limits the applications of these formalisms in the practice.

However, SGs have been defined to offer *a strong formal background to applications*, and therefore, their recognition is computationally affordable. Indeed, in this section we show a methodology to convert a spatial grammar into an equivalent *translation schema* that, once generated, may be fed to standard compiler–compiler tools such as YACC [23], Bison [20], AntLR [29] or JavaCC [22] to obtain a fully functional recognizer.

This methodology can be accomplished in two steps, as described in the following sections. Note that, currently, the methodology can not be applied to *inherently* recursive grammars. However, it can be applied to all the other spatial grammars, e.g. by removing possible left recursion through the standard techniques used for context-free grammars. In the conclusions section, we will discuss how we plan to handle also inherently recursive grammars in the next version of our methodology.

5.1 From spatial grammars to string grammars

In this section, we illustrate the algorithms and the concepts used to linearize the spatial formalism by merging the spatial relations into the grammar symbols. In this way, the resulting string grammars will contain terminal and non-terminal symbols of the form REL_x and REL_X that we shall call *spatial* grammar symbols from now on.

Before starting the conversion, it is necessary to apply a preliminary transformation of the source spatial grammar. The *Normalize* algorithm (1) ensures that each non-terminal symbol, with the exception of the start symbol, is present on the left-hand side of a single production. To accomplish this, it introduces new non-terminal symbols and duplicates some productions if necessary.

Of course, this normalization may strongly increase the size of the grammar. However, if the input grammar is not inherently recursive, it always terminates.

Algorithm 1 Normalize

Input: A spatial grammar $SG = (N, T, S, P, R)$.
Output: A spatial grammar SG'.
Procedure:

1. Consider the subsets of productions having the same left-hand side, excluding the start symbol S. For each subset $\{X \to \alpha_1, \ldots, X \to \alpha_n\}$:
   (a) introduce n new nonterminal symbols $\{X_1, \ldots, X_n\}$, one for each production, and rename the left-hand symbols accordingly, i.e., $\{X_1 \to \alpha_1, \ldots, X_n \to \alpha_n\}$
   (b) replace each production of SG where the non-terminal X appears in the right-hand side with a set of productions where the instances of X are replaced with all the possible combinations of $\{X_1, \ldots, X_n\}$. That is, if X appears k times, the production is replaced with n^k new productions.
2. Iterate this process until there are no new subsets of productions having the same left-hand side.

As an example, let us consider a spatial grammar containing the following three productions

$$\begin{aligned} X &\to \alpha_1 \\ X &\to \alpha_2 \\ Y &\to \alpha X \beta X \gamma \end{aligned}$$

Applying the Normalize algorithm, this fragment becomes

$$\begin{aligned} X_1 &\to \alpha_1 \\ X_2 &\to \alpha_2 \\ Y &\to \alpha X_1 \beta X_1 \gamma \\ Y &\to \alpha X_1 \beta X_2 \gamma \\ Y &\to \alpha X_2 \beta X_1 \gamma \\ Y &\to \alpha X_2 \beta X_2 \gamma \end{aligned}$$

In the next iteration, the resulting fragment will be

$$\begin{aligned} X_1 &\to \alpha_1 \\ X_2 &\to \alpha_2 \\ Y_1 &\to \alpha X_1 \beta X_1 \gamma \\ Y_2 &\to \alpha X_1 \beta X_2 \gamma \\ Y_3 &\to \alpha X_2 \beta X_1 \gamma \\ Y_4 &\to \alpha X_2 \beta X_2 \gamma \end{aligned}$$

At this point, the process stops since there are no more productions with the same left-hand side.

Now, we can start the transformation process with the *RelPrecede* algorithm (2) that, applied to a non-terminal A, calculates the set of spatial relations that can appear immediately to the left of A in some sentential form, that is, the set of spatial relations from which A can be reached.

Example 5.1 Let us consider the following spatial grammar:

$$\begin{aligned} S &\to A\ UP\ B \\ A &\to_2 x\ LEFT\ y \\ A &\to z \\ B &\to w\ UP\ h\ RIGHT^2\ f \end{aligned}$$

Algorithm 2 RelPrecede

Input: A normalized spatial grammar $SG = (N, T, S, P, R)$.
Output: For each non-terminal A of SG, the set of spatial relations $RelPrecede(A)$.
Procedure:

1. If the start symbol S of SG appears in the left-hand side of a single production, add $RELSP$ to $RelPrecede(S)$, where $RELSP$ is a fake spatial relation denoting the starting point of the multi-dimensional sentences generated by S. Otherwise, if S it appears k times, add the $RELSP_i$ with $i = 1 \ldots k$ to $RelPrecede(S)$.
2. For all productions $A \rightarrow_j \alpha\ REL^i\ B\ \beta$, add REL to $RelPrecede(B)$.
3. For all productions $A \rightarrow_j\ B\ \alpha$, add the content of $RelPrecede(A)$ to $RelPrecede(B)$.

Once normalized, the grammar becomes

$$\begin{aligned} S &\rightarrow A_1\ UP\ B \\ S &\rightarrow A_2\ UP\ B \\ A_1 &\rightarrow_2 x\ LEFT\ y \\ A_2 &\rightarrow z \\ B &\rightarrow w\ UP\ h\ RIGHT^2\ f \end{aligned}$$

Now, the RelPrecede algorithm calculates the following sets

$$\begin{aligned} RelPrecede(S) &= \{RELSP_1, RELSP_2\} \\ RelPrecede(A_1) &= \{RELSP_1, RELSP_2\} \\ RelPrecede(A_2) &= \{RELSP_1, RELSP_2\} \\ RelPrecede(B) &= \{UP\} \end{aligned}$$

Before presenting the next algorithm *Spatial_to_String* (3) that implements the translation of the multidimensional spatial formalism into a string one, we need to define a function that joins spatial relations and grammar symbols into new merged grammar symbols, i.e. spatial terminals and spatial non-terminals. This function is defined as follows.

Definition 5.1 Let $\alpha = REL_1\ a_1 \ldots REL_k\ a_k$ be a sequence of spatial relations and grammar symbols of a spatial grammar. Then, we define $\Sigma(\alpha)$ as the sequence $REL_1\_a_1 \ldots REL_k\_a_k$.

As an example, if α is the sequence $UP\ x\ RIGHT\ y\ DOWN\ z$, then $\Sigma(\alpha) = UP\_x\ RIGHT\_y\ DOWN\_z$.

Algorithm 3 Spatial_to_String

Input: A normalized spatial grammar $SG = (N, T, S, P, R)$.
Output: A context-free grammar G.
Procedure:

1. For each production $A \rightarrow_j x\ \alpha \in P$, with $x \in N \cup T$, $A \neq S$, insert in G the context-free productions $REL_i\_A \rightarrow_j REL_i\_x\ \Sigma(\alpha)$, for each $REL_i \in RelPrecede(A)$.
2. If there are k productions of the start symbol S, i.e., $S \rightarrow_{j_i} x_i\ \alpha_i$, $i = 1..k$, insert in G the context-free productions $RELSP_i\_S \rightarrow_{j_i} RELSP_i\_x\ \Sigma(\alpha_i)$.

Example 5.2 The spatial grammar of Example 5.1 is transformed by the Spatial_to_String algorithm into the following context-free grammar.

$$\begin{aligned}
RELSP_1_S &\rightarrow RELSP_1_A_1\ UP_B \\
RELSP_2_S &\rightarrow RELSP_2_A_2\ UP_B \\
RELSP_1_A_1 &\rightarrow_2 RELSP_1_x\ LEFT_y \\
RELSP_2_A_2 &\rightarrow RELSP_2_z \\
UP_B &\rightarrow UP_w\ UP_h\ RIGHT^2_f
\end{aligned}$$

Note that the grammar was simplified by removing the useless productions having $RELSP_1\_A_2$ and $RELSP_2\_A_1$ on the right-hand side.

Clearly, after this first step of the methodology, the current, intermediate string grammar, is no more equivalent to the original spatial grammar. Indeed, the terminal symbols of the source language have been embedded within spatial terminals. Moreover, the original start symbol of the spatial grammar has been split into multiple different spatial non-terminals. In the next step of the methodology, both terminals and start symbol of the original spatial grammar will be properly restored.

5.2 From string grammars to translation schemas

In this section, we illustrate the algorithms and the concepts used to convert a string grammar G as obtained by the above step into a canonical translation schema TS, which is equivalent to the original spatial grammar SG. This is accomplished by including in the translation schema suitable semantic actions which simulate the spatial relations in SG.

The *RelInside* algorithm (4), applied to a generic spatial non-terminal REL_x, calculates the set of terminals that can be generated by the grammar productions having REL_x on the left-hand side, and that are pointed by the corresponding anchors. As an extension, *RelInside*, applied to a spatial terminal, returns the terminal itself.

Algorithm 4 RelInside

Input: A context-free grammar G obtained from Algorithm 3.
Output: For each spatial terminal or non-terminal REL_x in G, the set of spatial relations $RelInside(x)$.
Procedure:

1. for each spatial terminal symbol REL_x, $RELINSIDE(REL\_x) = \{x\}$
2. for each production $REL\_A \rightarrow_j REL_1\_x_1 \ldots REL_n\_x_n$, for each spatial symbol $REL_i\_x_i$ appearing on its right-hand side and pointed by the anchor j (remember that if the anchor is not present, we suppose it to contain all the indices $1 \ldots n$), put in $RELINSIDE(REL\_A)$ all the contents of $RELINSIDE(REL_i\_x_i)$

Example 5.3 The algorithm above, applied to the spatial non-terminals of the string grammar in Example 5.2, produces the following sets:

$$\begin{aligned}
RelInside(RELSP_1_S) &= \{y, w, h, f\} \\
RelInside(RELSP_2_S) &= \{z, w, h, f\} \\
RelInside(RELSP_1_A_1) &= \{y\} \\
RelInside(RELSP_2_A_2) &= \{z\} \\
RelInside(UP_B) &= \{w, h, f\}
\end{aligned}$$

Finally, algorithm Grammar_to_TranslationSchema (5) creates a translation schema where the right-hand symbols are interleaved by *semantic actions* that, in our case, are function calls implementing the spatial relations. In particular, such functions may check that the given relations hold between the parameter objects: otherwise, they may stop the parsing or instruct the parser to backtrack (e.g. with constructs like the *semantic lookahead* [22]).

Algorithm 5 Grammar_to_TranslationSchema

Input: A context-free grammar $G = (N', T', S', P')$ obtained from the spatial grammar $SG = (N, T, S, P, R)$ through Algorithm 3, and the $RelInside$ sets generated by Algorithm 4.
Output: A translation schema $TS = (N'', T'', S'', P'')$ equivalent to SG.
Procedure:

1. Let $TS = (N' \cup \{S\}, T, S, P'')$ be a translation schema, i.e., a context-free grammar where the productions may contain (sets of) actions mixed with the grammar symbols.
2. For all productions $p' = REL\_A \rightarrow REL_1^{j_1}\_x_1 \ldots REL_k^{j_k}\_x_k \in P'$, insert a corresponding production p'' in P'' with the same left-hand symbol (REL_A) and a right-hand side built as follows. For all $i \in [1, \ldots, k]$,
   (a) if $i > 1$, append to the right side of p'' the set of actions $A = \left\{REL_i(y, z) | y \in RelInside(x_{i-j_i}), z \in RelInside(x_i)\right\}$
   (b) if $x_i \in T$, append it to the right side of p'', otherwise append $REL_i^{j_i}\_x_i$.
3. Finally, for all the non-terminals $RELSP_i\_S \in N'$, add the production $S \rightarrow RELSP_i\_S$ to P''.

Example 5.4 The algorithm above, applied to the spatial grammar obtained in Example 5.2 and the sets of relations RelInside from Example 5.3, produces the following grammar:

$$
\begin{array}{lcl}
S & \rightarrow & RELSP_1\_S \\
S & \rightarrow & RELSP_2\_S \\
RELSP_1\_S & \rightarrow & RELSP_1\_A_1 \; \{UP(y, w)\} \; \{UP(y, h)\} \; \{UP(y, f)\} \; UP\_B \\
RELSP_2\_S & \rightarrow & RELSP_2\_A_2 \; \{UP(z, w)\} \; \{UP(z, h)\} \; \{UP(z, f)\} \; UP\_B \\
RELSP_1\_A_1 & \rightarrow_2 & x \; \{LEFT(x, y)\} \; y \\
RELSP_2\_A_2 & \rightarrow & z \\
UP\_B & \rightarrow & w \; \{UP(w, h)\} \; h \; \{RIGHT(w, f)\} \; f
\end{array}
$$

which is equivalent to the original spatial grammar in Example 5.1 since it produces an equivalent language.

Indeed, the spatial grammar produces the following terminal sentential forms:

1. $z \; UP \; \left[w \; UP \; h \; RIGHT^2 \; f\right]$
2. $[x \; LEFT \; y]_{\{2\}} \; UP \; \left[w \; UP \; h \; RIGHT^2 \; f\right]$

On the other hand, the translation schema above has the following terminal productions:

1. $z \; \{UP(z, w)\} \; \{UP(z, h)\} \; \{UP(z, f)\} \; w \; \{UP(w, h)\} \; h \; \{RIGHT^2(w, f)\} \; f$
2. $x \; \{LEFT(x, y)\} \; y \; \{UP(y, w)\} \; \{UP(y, h)\} \; \{UP(y, f)\} \; w \; \{UP(w, h)\} \; h \; \{RIGHT(w, f)\} \; f$

Note that, in the translation schema actions, the spatial relations have no apex, since their actual arguments have been resolved by the algorithm. If we suppose to embed in the spatial actions of the translation schema the same semantics of the layout operator $f_{R,p}$, then

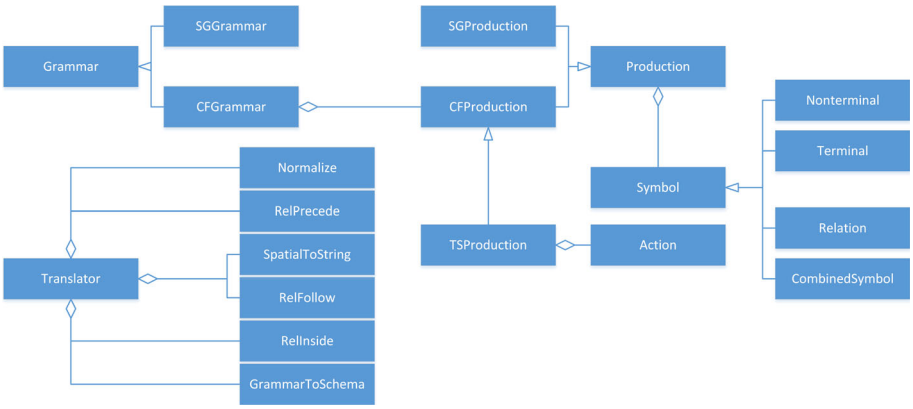


Fig. 10 A simplified class diagram for the algorithm implementation and the grammar modelling structures

the language $L(G|f_{R,p})$ will be the same accepted by the translation schema productions. Indeed, it is easy to see that the semantic actions in the translation schema check that all the spatial relations in the corresponding terminal sentential form hold.

5.3 Implementation

In order to extensively test the complete spatial grammar to translation schema process, all the algorithms described have been implemented in Java, trying to maintain the closest resemblance between the pseudo-code given in the paper and the actual code, as a further way to check the implementation validity. Moreover, the library contains a set of classes to model context-free grammars, spatial grammars and translation schemas. All the classes have pretty-printing routines that can be used to output the corresponding grammar structures in a human-readable format that is very similar to the canonical grammar notation. Figure 10 shows a simplified view of the overall class library, whose full source code is publicly available in the author's GitHub repository [13].

The user can define a source spatial grammar using a set of helper functions, pass such grammar to the algorithm and then analyse the resulting translation schema via code or simply print it to the console. However, the context-free and spatial grammar classes are also able to load the grammar definition from a file. To this aim, we defined a simple JSON-based grammar format that can be both easily written by a human and read by the machine.

As an example, the following file defines the same grammar of Example 5.1:

```
{
     "terminals": ["h", "f", "x", "y","z","w"],
     "nonterminals": ["A", "B", "S"],
     "start": ["S"],
     "productions": [
     {"S": [[],"A",{"UP":1},"B"]},
     {"A": [[2],"x", {"LEFT":1},"y"]},
     {"A": [[],"z"]},
     {"B": [[],"w",{"UP":1},"h",{"RIGHT":2},"f"]}
     ]
}
```

If we run the translator on the input above, we obtain the following diagnostic output showing all the intermediate results that can be easily compared with the examples of the corresponding algorithms included in the previous section.

```
Input Spatial Grammar:
S -> A UP B
A ->[2] x LEFT y
A -> z
B -> w UP h RIGHT^2 f

_________________Normalize_________________
S -> A1 UP B
S -> A2 UP B
A1 ->[2] x LEFT y
A2 -> z
B -> w UP h RIGHT^2 f

** RelPrecede:
{A1=[RELSP1, RELSP2], A2=[RELSP1, RELSP2], B=[UP], S=[RELSP1
    , RELSP2]}

_____________Spatial to String Grammar_____________
RELSP1_S -> RELSP1_A1 UP_B
RELSP2_S -> RELSP2_A2 UP_B
UP_B -> UP_w UP_h RIGHT^2_f
RELSP2_A2 -> RELSP2_z
RELSP1_A1 ->[2] RELSP1_x LEFT_y

** RelFollow:
{UP_B=[REL$], RELSP2_A2=[UP], RELSP1_S=[REL$], RELSP2_S=[
    REL$], RELSP1_A1=[UP]}
** RelInside:
{UP_B=[w, h, f], RELSP2_A2=[z], RELSP1_S=[y, w, h, f],
    RELSP2_S=[z, w, h, f], RELSP1_A1=[y]}

_________String Grammar to Translation Schema_________
```

Finally, the last step outputs the following final translation schema, which is identical to one of Example 5.4:

```
StartSymbol -> RELSP1_S
StartSymbol -> RELSP2_S
UP_B ->  w {UP(w,h)} h {RIGHT(w,f)} f
RELSP2_A2 ->  z
RELSP1_S ->  RELSP1_A1 {UP(y,w)} {UP(y,h)} {UP(y,f)} UP_B
RELSP2_S ->  RELSP2_A2 {UP(z,w)} {UP(z,h)} {UP(z,f)} UP_B
RELSP1_A1 -> [2] x {LEFT(x,y)} y
```

Note that the implementation actually removes all the useless productions from the output grammar, in particular after the first step (spatial grammar to string grammar). On the other hand, the algorithm described in Sect. 5 has not been optimized in this way, to make it as simple as possible to describe.

6 A case study

In this section, we show a simple real-world example where the spatial grammars and the ability to transform them in YACC-compatible translation schemas can be useful to accomplish concrete tasks. Indeed, fast, standard compilers or interpreters, as the ones that can be generated through YACC or similar tools, can be useful in many applications.

Understanding traffic signs is a very important task for self-driving cars and, in general, for software assistants embedded in modern cars. Therefore, researchers have been studying this task for a long time (see, for example, [18, 33] for two recent surveys in this argument), which is often achieved using ML-assisted image recognition algorithms trained using the sign images. However, traffic signs are not generic, complex pictures, but rather have a very simple structure, composed by basic figures (e.g. triangles for warnings, circles for regulatory signs, etc.) combined with a predefined set of standard symbols. Indeed, traffic signs are intentionally designed with particular colours and shapes which make them easy to recognize, so the most reliable way to detect a sign should be using the colour and shape information [18]. This suggests that (context-free) grammars may be a more suitable way both to compactly specify a wide range of traffic signs and to implement lightweight traffic signs recognizers. Clearly, signs are images, so we need a spatial grammar to define them.

In particular, the grammar exploits the structure recalled above to define each sign by means of basic elements such as geometric shapes, numbers, icons and arrows composed using spatial relations. Such basic, simple elements would be easy to visually recognize using traditional visual matching algorithms: as an example, ML can be exploited in this task, but with much lower complexity and higher precision, since the recognizer must be trained to understand only a small number of simple shapes.

Once the basic shapes and their position have been detected, the spatial grammar composition rules can be matched against them (through the feasible regions that are easily derivable from the objects' position and contour) in order to quickly decode the sign meaning.

The grammar derivation for a sign constitutes a formal proof of its recognition process making it more *explainable*. Moreover, grammars are very robust, and their use may drastically decrease the number of classification errors that may arise from the use of neural networks in the overall process [35].

A spatial grammar fragment which models a set of common traffic signs is shown in Fig. 11. In the grammar, the non-terminal W produces warning signs, F regulatory signs indicating prohibited actions and R mandatory actions. The figure uses visual symbols to represent the grammar terminals in order to make it easier to understand.

Fig. 11 A fragment of the traffic signs spatial grammar

$$
\begin{array}{lcl}
S & \rightarrow & W \\
S & \rightarrow & F \\
S & \rightarrow & R \\
W & \rightarrow & C_W \ INSIDE \ \triangle \\
F & \rightarrow & C_F \ INSIDE \ \bigcirc \\
F & \rightarrow & \bigcirc \\
R & \rightarrow & C_R \ INSIDE \ \bullet \\
C_W & \rightarrow & \uparrow\downarrow \\
C_F & \rightarrow & C_{FB} \\
C_F & \rightarrow & \diagup \ INTERSECT \ C_{FB} \\
C_F & \rightarrow & C_{FP} \\
C_{FB} & \rightarrow & N \\
C_{FB} & \rightarrow & N \ INSIDE \ \blacktriangleright\blacktriangleleft \\
N & \rightarrow & numbers \\
C_{FP} & \rightarrow & \diagup \ INTERSECT \ \bullet \\
C_R & \rightarrow & A \\
A & \rightarrow & \blacksquare
\end{array}
$$

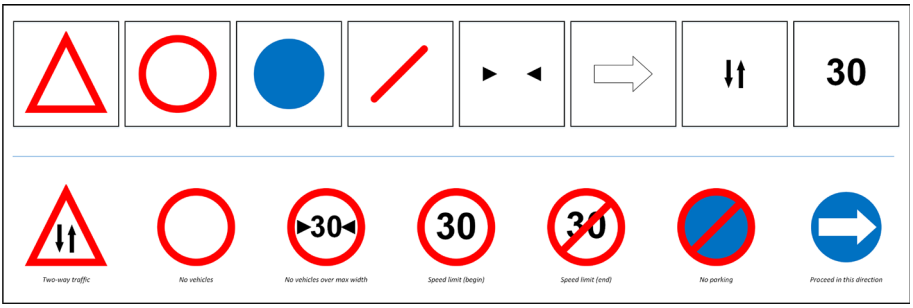


Fig. 12 Terminal tokens (up) and some productions (down) of the traffic signs grammar

Examples of productions (signs) of this grammar are the following:

1. **two-way traffic:** $S \Rightarrow W \Rightarrow C_W \ INSIDE \ \triangle \Rightarrow \uparrow\downarrow \ INSIDE \ \triangle$
2. **no vehicles:** $S \Rightarrow F \Rightarrow \bigcirc$
3. **speed limit (begin):** $S \Rightarrow F \Rightarrow C_F \ INSIDE \ \bigcirc \Rightarrow C_{FB} \ INSIDE \ \bigcirc \Rightarrow N \ INSIDE \ \bigcirc \Rightarrow 30 \ INSIDE \ \bigcirc$
4. **speed limit (end):** $S \Rightarrow F \Rightarrow C_F \ INSIDE \ \bigcirc \Rightarrow [\diagup \ INTERSECT \ C_{FB}] \ INSIDE \ \bigcirc \Rightarrow [\diagup \ INTERSECT \ N] \ INSIDE \ \bigcirc \Rightarrow [\diagup INTERSECT \ 30] \ INSIDE \ \bigcirc$
5. **no vehicles over maximum width:** $S \Rightarrow F \Rightarrow C_F \ INSIDE \ \bigcirc \Rightarrow C_{FB} \ INSIDE \ \bigcirc \Rightarrow [N \ INSIDE \ \blacktriangleright\blacktriangleleft] \ INSIDE \ \bigcirc \Rightarrow [30 \ INSIDE \ \blacktriangleright\blacktriangleleft] \ INSIDE \ \bigcirc$
6. **no parking:** $S \Rightarrow F \Rightarrow C_F \ INSIDE \ \bigcirc \Rightarrow C_{FP} \ INSIDE \ \bigcirc \Rightarrow [\diagup \ INTERSECT \ \bullet] \ INSIDE \ \bigcirc$
7. **proceed in this direction:** $S \Rightarrow R \Rightarrow C_R \ INSIDE \ \bullet \Rightarrow A \ INSIDE \ \bullet \Rightarrow \blacksquare \ INSIDE \ \bullet$

Figure 12 visually shows the grammar terminals (upper portion of the image) and the results of the example productions above (lower portion).

```
StartSymbol  -> RELSP1_S
StartSymbol  -> RELSP2_S
StartSymbol  -> RELSP3_S
StartSymbol  -> RELSP4_S
StartSymbol  -> RELSP5_S
StartSymbol  -> RELSP6_S
StartSymbol  -> RELSP7_S
StartSymbol  -> RELSP8_S

INTERSECT_CFB1 -> INTERSECT_N
INTERSECT_CFB2 -> INTERSECT_N {INSIDE(30,rlarr)} rlarr
INTERSECT_N -> 30
RELSP1_CW -> udarr
RELSP1_S -> RELSP1_W
RELSP1_W -> RELSP1_CW {INSIDE(udarr,triangle)} triangle
RELSP2_CF1 -> RELSP2_CFB1
RELSP2_CFB1 -> RELSP2_N
RELSP2_F1 -> RELSP2_CF1 {INSIDE(30,rcircle)} rcircle
RELSP2_N -> 30
RELSP2_S -> RELSP2_F1
RELSP3_CF2 -> RELSP3_CFB2
RELSP3_CFB2 -> RELSP3_N {INSIDE(30,rlarr)} rlarr
RELSP3_F2 -> RELSP3_CF2 {INSIDE(30,rcircle)} {INSIDE(rlarr,rcircle)} rcircle
RELSP3_N -> 30
RELSP3_S -> RELSP3_F2
RELSP4_CF3 -> slash {INTERSECT(slash,30)} INTERSECT_CFB1
RELSP4_F3 -> RELSP4_CF3 {INSIDE(slash,rcircle)} {INSIDE(30,rcircle)} rcircle
RELSP4_S -> RELSP4_F3
RELSP5_CF4 -> slash {INTERSECT(slash,30)} {INTERSECT(slash,rlarr)}
    INTERSECT_CFB2
RELSP5_F4 ->  RELSP5_CF4 {INSIDE(slash,rcircle)} {INSIDE(30,rcircle)} {INSIDE
    (rlarr,rcircle)} rcircle
RELSP5_S -> RELSP5_F4
RELSP6_CF5 -> RELSP6_CFP
RELSP6_CFP -> slash {INTERSECT(slash,bcircle)} bcircle
RELSP6_F5 -> RELSP6_CF5 {INSIDE(slash,rcircle)} {INSIDE(bcircle,rcircle)}
    rcircle
RELSP6_S -> RELSP6_F5
RELSP7_F6 ->  rcircle
RELSP7_S -> RELSP7_F6
RELSP8_A ->  rarr
RELSP8_CR -> RELSP8_A
RELSP8_R -> RELSP8_CR {INSIDE(rarr,bcircle)} bcircle
RELSP8_S -> RELSP8_R
```

Fig. 13 Translation schema generated from the traffic signs grammar in Fig. 11

The algorithm described in Sect. 5.3, applied to the grammar above, produces the output in Fig. 13. In the translation schema, the terminals should be easy to connect to the symbols used in Fig. 11, e.g. ◯ is denoted by *rcircle*, ● by *bcircle*, ►◄ by *rlarr*, etc.

As an example, the generated grammar above produces sentence 1. with the context-free production

$$StartSymbol \Rightarrow RELSP1\_S \Rightarrow RELSP1\_W \Rightarrow RELSP1\_CW\ \{INSIDE(udarr, triangle)\}\ triangle \Rightarrow udarr\{INSIDE(udarr, triangle)\}\ triangle$$

and sentence 3. with the production

$$StartSymbol \Rightarrow RELSP2\_S \Rightarrow RELSP2\_F1 \Rightarrow RELSP2\_CF1\ \{INSIDE(30, rcircle)\}\ rcircle \Rightarrow RELSP2\_CFB1\ \{INSIDE(30, rcircle)\}\ rcircle \Rightarrow RELSP2\_N\ \{INSIDE(30, rcircle)\}\ rcircle \Rightarrow\ 30\ \{INSIDE(30, rcircle)\}\ rcircle.$$

It is easy to check that the semantic actions in the produced sentences actually contain the relations that must hold for the sign to be drawn correctly, if we assign to the spatial actions their "common" semantics.

7 Conclusions

In this paper, we presented a grammar formalism for multidimensional languages, namely the spatial grammars, which combine common string-like productions with the spatial relations taken from the PCT framework. Grammars written with such formalism can be automatically translated in equivalent translation schemas, which can then be handled by any standard compiler generation tool. Thus, spatial grammars are both easy to understand for humans and easy to manipulate for the machine.

The presented case study shows how spatial grammars can be a valuable aid in computer vision tasks like image recognition, where they can be used to support, e.g. ML techniques in order to make their results more robust and explainable.

However, the applicability of spatial grammars extends beyond such computer vision tasks. In our future work, we will continue working on the overall algorithm, testing it with different case studies taken from other real-world contexts in order to explore new application fields. As an example, we plan to apply spatial grammars in the computer programming context as a tool to check properties of visual code representations like flowcharts. (Example 4.2 gives a first idea on how spatial grammars can be used to model flowcharts.) In general, spatial grammars could be easily applied as a specification/verification/recognition tool where diagrams are involved, for example, also on vector/CAD drawings, where the use of grammars has already been proposed (see, for example, [31]).

From the theoretical point of view, as further research in this field, we intend to analyse the width of the multidimensional language class that can be described by SGs, i.e. their expressive power, and compare them with well-established formalisms for multidimensional languages such as graph grammars.

We are also studying how to include the full power of the PCT framework in the SG formalism, in particular enhancing the algorithm of Sect. 5 to support also the temporal part of PCT. In this way, spatial grammars may be also used to encode dynamic spatial arrangements, for example, gestures, where PCT has been already successfully applied [12].

Finally, we are working to remove the non-recursiveness limitation from the grammars that can be handled by the algorithm presented in Sect. 5. Actually, given the kind of languages targeted by this formalism, it may be sufficient to set a reasonable recursion limit in the normalization algorithm (1) to make the overall transformation work on inherently recursive grammars in real-world applications.

Acknowledgements We thank the anonymous reviewers and the journal editor for their useful comments and suggestions.

References

1. Aho A, Sethi R, Ullman J (1986) Compilers, principles, techniques, and tools, Addison-Wesley series in computer science and information processing. Addison-Wesley, Boston
2. Allen JF (1983) Maintaining knowledge about temporal intervals. Commun ACM 26(11):832–843

3. Balbiani P, Condotta J, del Cerro LF (1998) A model for reasoning about bidemsional temporal relations. In: Proceedings of the sixth international conference on principles of knowledge representation and reasoning (KR'98), Trento, Italy, June 2–5, 1998, pp 124–130
4. Barredo Arrieta A, Díaz-Rodríguez N, Del Ser J, Bennetot A, Tabik S, Barbado A, Garcia S, Gil-Lopez S, Molina D, Benjamins R, Chatila R, Herrera F (2020) Explainable artificial intelligence (xai): concepts, taxonomies, opportunities and challenges toward responsible ai. Inf Fusion 58:82–115
5. Becker G (2007) Combining rule-based and machine learning approaches for shape recognition. In: 36th Applied imagery pattern recognition workshop (aipr 2007), pp 65–70
6. Chen J, Cohn AG, Liu D, Wang S, Ouyang J, Yu Q (2015) A survey of qualitative spatial representations. Knowl Eng Rev 30:106–136
7. Clementini E, Felice PD, Oosterom P (1993) A small set of formal topological relationships suitable for end-user interaction. In: Proceedings of the third international symposium on advances in spatial databases (SSD'93). Springer, London, pp 277–295
8. Crimi C, Guercio A, Nota G, Pacini G, Tortora G, Tucci M (1991) Relation grammars and their application to multi-dimensional languages. J Vis Lang Comput 2(4):333–346
9. Della Penna G, Magazzeni D, Orefice S (2016) Extending visual information extraction to biomedical applications. Comput Syst Sci Eng 31(5):371–383
10. Della Penna G, Magazzeni D, Orefice S (2017) A formal framework to represent spatial knowledge. Knowl Inf Syst 51(1):311–338
11. Della Penna G, Orefice S (2018) Qualitative representation of spatio-temporal knowledge. J Vis Lang Comput 49:1–16
12. Della Penna G, Orefice S (2019) Using spatial relations for qualitative specification of gestures. Comput Syst Sci Eng 34(6):325–338
13. Della Penna G, D'Angelo A (2022) PCT_SGtoTS algorithm repository. https://github.com/gdellapenna/PCT_SGtoTS
14. Egenhofer MJ, Mark DM, Herring J (1991) Categorizing binary topological relationships between regions, lines and points in geographic databases, Technical report, Department of Surveying Engineering, University of Maine
15. Egenhofer MJ, Mark DM, Herring J (1994) The 9-intersection: formalism and its use for natural-language spatial predicates, Technical Report 94-1, National Center for Geographic Information and Analysis
16. Engelfriet J (1997) Context-free graph grammars. In: Rozenberg G, Salomaa A (eds) Handbook of formal languages: volume 3 beyond words. Springer, Berlin, pp 125–214
17. Flesca S, Furfaro F, Greco S (2006) A graph grammars based framework for querying graph-like data. Data Knowl Eng 59(3):652–680
18. Fu M-Y, Huang Y-S (2010) A survey of traffic sign recognition. In: 2010 International conference on wavelet analysis and pattern recognition, pp 119–124
19. Giammarresi D, Restivo A (1997) Two-dimensional languages. In: Rozenberg G, Salomaa A (eds) Handbook of formal languages: volume 3 beyond words. Springer, Berlin, pp 215–267
20. GNU (2021) Gnu bison—the yacc-compatible parser generator. https://www.gnu.org/software/bison
21. Goyal R, Egenhofer M (2001) Similarity of cardinal directions. In: Jensen C, Schneider M, Seeger B, Tsotras V (eds) Advances in spatial and temporal databases, vol 2121. Lecture notes in computer science. Springer, Berlin, pp 36–55
22. JavaCC Community (2021) Java compiler compiler (javacc). https://javacc.github.io/javacc
23. Johnson SC, Sethi R (1990) Yacc: a parser generator. In: UNIX Vol. II: research system, 10th edn. W. B. Saunders Company, USA, pp 347–374
24. Krishnamurti R, Stouffs R (1993) Spatial grammars: motivation, comparison, and new results. In: Proceedings of the fifth international conference on computer-aided architectural design futures. North-Holland, NLD, pp 57–74
25. Meyer-Baese A, Schmid V (2014) Chapter 6—statistical and syntactic pattern recognition. In: Meyer-Baese A, Schmid V (eds) Pattern recognition and signal analysis in medical imaging, 2nd edn. Academic Press, Oxford, pp 151–196
26. Moratz R (2006) Representing relative direction as a binary relation of oriented points. In: Proceedings of the ECAI 2006, 17th European conference on artificial intelligence, August 29—September 1, 2006, Riva del Garda, Italy, including prestigious applications of intelligent systems (PAIS 2006), pp 407–411
27. Moratz R, Ragni M (2008) Qualitative spatial reasoning about relative point position. J Vis Lang Comput 19(1):75–98
28. Mossakowski T, Moratz R (2012) Qualitative reasoning about relative direction of oriented points. Artif Intell 180–181:34–45
29. Parr T (2021) Antlr (another tool for language recognition). https://www.antlr.org

30. Randell DA, Cui Z, Cohn AG (1992) A spatial logic based on regions and connection. In: Proceedings of the 3rd international conference on principles of knowledge representation and reasoning (KR'92). Cambridge, MA, October 25–29, 1992, pp 165–176
31. Rowe PDG, Reed C (2006) Cad grammars. In: Gero JS (ed) Design computing and cognition '06. Springer, Dordrecht, pp 503–520
32. Rozenberg G, Salomaa A (eds) (1997) Handbook of formal languages, vol 3: beyond words. Springer, Berlin
33. Sanyal B, Mohapatra RK, Dash R (2020) Traffic sign recognition: a survey. In: 2020 International conference on artificial intelligence and signal processing (AISP), pp 1–6
34. Skiadopoulos S, Koubarakis M (2004) Composing cardinal direction relations. Artif Intell 152(2):143–171
35. Zhu Z, Liang D, Zhang S, Huang X, Li B, Hu S (2016) Traffic-sign detection and classification in the wild. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), pp 2110–2118

Giuseppe Della Penna obtained the master degree in Computer Science in 1998 at the University of L'Aquila, Italy, and the PhD in Computer Science in 2002 at the University of Rome "La Sapienza". He is currently an Associate Professor in the Department of Information Engineering, Computer Science and Mathematics of the University of L'Aquila. His research interests include formal methods applied to web engineering, software engineering and visual languages. In these contexts, he published several peer-reviewed articles in international journals and conferences.

Sergio Orefice received the master degree in Computer Science from the University of Salerno, Italy, and a PhD degree in applied mathematics and computer science from the University of Naples, Italy. From 2000 to 2020, he has been an Associate Professor in the Department of Information Engineering, Computer Science and Mathematics ats University of L'Aquila, Italy, where he worked in the group of Software Engineering and Programming Languages. In 2020, he quit the Academia for other interests. His main research topics include logic programming, visual languages and parsing technologies for multidimensional formal languages. On these topics, he has published several papers in international journals, books and proceedings of refereed international conferences.

Andrea D'Angelo is a PhD student at the University of L'Aquila, Italy, where he previously earned his Bachelor's and Master's degrees in Computer Science, in 2019 and 2022, respectively. His research interests revolve around information retrieval, formal language theory and machine learning models for natural language processing, such as transformer neural networks.