Constraint Logic Programming over Sets of Spatial Objects

Jesús M. Almendros-Jiménez
Dpto. de Lenguajes y Computacion. Universidad de Almeria
jalmen@ual.es

Abstract

Constraint Logic Programming (CLP) is a framework integrating Constraint Programming (CP) and Logic Programming (LP). CLP is described as a general schema of combination of logic-based languages and constraint solvers. Recently, a constraint system for the handling of constraints over sets of spatial objects has been presented. In this paper we study how to consider a CLP instance for this kind of constraints. In particular, we study the fixed-point and operational semantics of such instance. With respect to the operational semantics it is described how the constraint solver interacts with the mechanism of resolution, in particular how to detect failure branches by means of a consistence constraint checker and how to achieve constraint propagation and compute solved forms.

Categories and Subject Descriptors F.3.2 [*Logics and meanings of programs*]: semantics of programming languages

General Terms Languages, Theory

Keywords CLP, Logic Programming

1. Introduction

Constraint Logic Programming (CLP) [10] is a theoretical framework whose aim is to combine *Constraint Programming* (CP) and *Logic Programming* (LP). The motivation of this integration is to take advantage from the declarative and rule-based style of logic programming for the incorporation of constraint problems which can specified in a declarative way and easily combined with the operational mechanism of logic programming. This paradigm can be considered as the best escenario for testing constraint solvers. In fact, most of constraint solvers has been primarily integrated to current implementations of logic-based systems. In order to make this integration one can formulate a query solving in logic programming as a constraint problem by considering term unification as a simple constraint system. Once we have made this identification, constraint programming can be combined with logic programming by adding new kind of constraints to the base constraint system consisting on term unification.

On the other hand, in the field of *Constraint Programming* [3], a wide research has been done in order to define specialized constraint systems for particular problems. For instance, *linear equations and inequations over reals* [13] for spatial objects handling, *boolean constraints* [8] for circuit structure reasoning, *linear constraints on integer intervals* [11] for combinatorial discrete problems, and *temporal constraints* [16] for time and scheduling reasoning. For each one of these constraint systems both generic and particular constraint solver techniques have been studied in order to improve the *searching of solutions*. They typically are based on *heuristics*, *backtracking* and *branch and bound algorithms*. Several techniques like *constraint propagation* achieving *local*, *arc* and *path* consistency, among others, ensure a good performance of *constraint solvers*. Most of these techniques are closely related with the basic operational mechanisms of logic-based languages, in fact, some of them can be found in term unification and solution searching in logic programming: backtracking for the traversal of the search space of solutions of a logic goal, and constraint propagation and consistence checking in the propagation of bindings in the unification process.

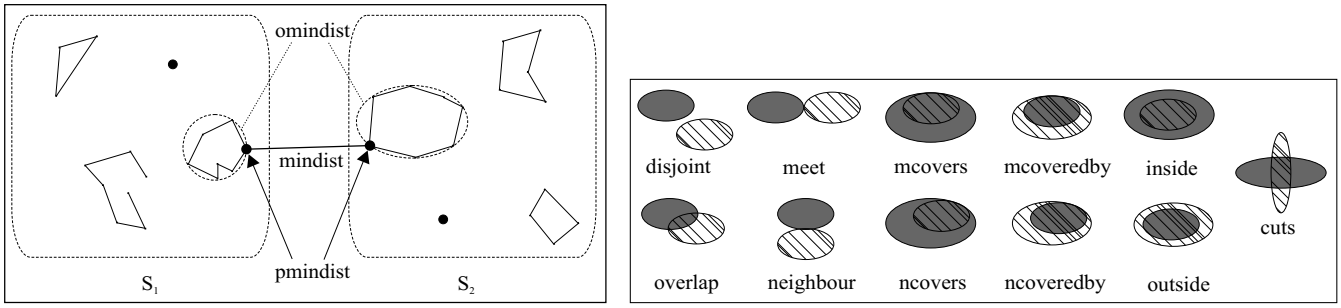
The framework of *constraint databases* [12] is also a application field of constraint programming. In this case there also exists an interest of capturing the declarative nature of database query languages, whose main exponent is SQL. In this case, this integration is based on the simple idea that a constraint can be seen as an extension of a relational tuple, called *generalized tuple*, or, vice versa, that a relational tuple can be interpreted as a conjunction of equality constraints. Each generalized tuple finitely represents a possibly infinite set of relational tuples, called *extension* of the generalized tuple, one for each solution of the constraint. In this framework, relations are now composed by generalized tuples, and operations on relations like cross products, selections and projections can be now generalized [15, 14, 5].

In this paper we focus on an special kind of problems which has been studied in both frameworks. This is how to use constraints for solving *spatial problems*. So far this kind of problems has been solved by using *linear* and *non-linear constraints* which seem to be suitable for modelling infinite and continuous spatial objects. In spatial constraint databases there have been studied *linear constraints* handling [15, 14, 5] in which spatial reasoning is based on linear equations and inequations. The basic idea is to see spatial objects as infinite sets of points and provide a finite representation of these infinite sets, with constraints over a dense domain. Most known spatial data types like *point*, *line* and *region* can be modeled with these frameworks, and *distance-based operations* can be included on it [4]. With respect to constraint logic programming constraint solvers for linear and non-linear constraints has been implemented and tested [7, 6].

However, in our opinion, an interesting direction could be the study of other kinds of constraint systems, *specialized for any class of spatial data types*, but suitable for practical applications. In particular, we are interested in modeling problems on *sets of spatial objects*, where each spatial object can be a *point*, *line*, *polygon*, or

WCFLP'05 September 29, 2005, Tallinn, Estonia.

Figure 1. Metric and Topological Operations



a *region*. In addition, the *query language* can handle *metric* and *topological queries* on these object sets.

Recently, a constraint system based on these ideas has been presented in [2]. In the quoted work, a *constraint solver* for *constraints on sets of spatial objects* (points, lines, polygons and regions) has been studied. This constraint system handles these *spatial data types* and constraints on them (*equalities and inequalities, memberships, metric, topological and structural constraints*). For this constraint system a suitable theory and constraint solving method have been presented.

The natural extension of the quoted work is the integration of the constraint system with logic programming, which gives rise to a new instance of constraint logic programing for sets of spatial objects, named as CLP($\mathcal{SSO}$). With respect to our previous work, we will present how to integrate our constraint system with logic programming. In particular, we will define the *syntax and (fixed point and operational) semantics* of this new instance, showing examples of CLP programs for *modeling spatial problems* and a set of *queries* requesting spatial answers. With respect to the semantics we will provide the corresponding *instantiation of the general CLP schema*. In addition, we will discuss some issues about the operational semantics, such as, how *unification* is adapted for unifying spatial objects, and how the constraint solver is integrated with the resolution mechanism. The constraint solver presented in [2] makes a distinction between a *consistence checker* which is used for *discarding search branches* in the constraint solving process and a *constraint store* which contains constraints over the so-called *candidate objects*, which represents the solution partially computed. This particular structure of the constraint solver is here combined with the resolution process, in order to *prune search branches* in the search tree of a CLP program. In addition, we have included some of the results presented in our previous paper in order to show a complete version of our framework.

The structure of the paper is as follows. Section 2 present how to introduce constraints on sets of spatial objects in a CLP program. Section 3 formally defines the framework of CLP($\mathcal{SSO}$) and present the fixed-point semantics. Section 4 presents the operational semantics and the results of soundness and completeness. Finally, section 5 concludes and presents future work.

2. Preliminaries

In this section we present the basis of the new instance of CLP for handling sets of spatial objects. Firstly, we will review the constraint system of sets of spatial objects presented in [2], and secondly, we will present some examples of CLP programs and queries handling this kind of constraints.

2.1 Constraints over Sets of Spatial Objects

Basically, we handle sets of spatial objects where a spatial object can be a point, line, polygon or region. Points will be represented by means of pairs of real and variable coordinates: $(1, 2)$ and $(\mathtt{x}, 1)$; lines by means of an ordered sequence of points: $\mathtt{line}[(1, 2), (4, 5), (9, 10)]$; and the same can be said for polygons and regions, where the points define a closed path (i.e. last point is connected to the first one). Lines, polygons and regions are not intended to contain more vertices.

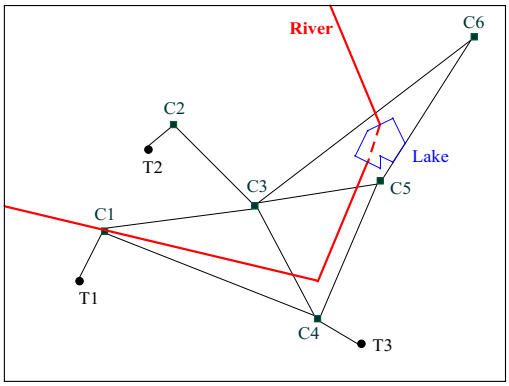
With this representation we can describe sets of *closed point-based geometric figures*: $\{(8, 1), \mathtt{line}[(1, 2), (4, 5), (9, 10)]\}$. On these spatial data types, a set of binary spatial operations is defined. We can classify them in five groups: equality and membership relations, set operations, metric operations, topological relations, and finally, structural operations. We are not interested in a complete set of operations, but enough for most usual spatial queries.

With respect to equality relations, spatial objects can be compared, for instance $(1, 2) = (2, 3)$, $(1, \mathtt{x}) = (1, 9)$, $\mathtt{line}[(1, 2), (10, \mathtt{y})] = \mathtt{line}[(\mathtt{z}, 2), (10, 12)]$. The first equality relation is false, but the other two equality relations define a relation on $\mathtt{x}$, $\mathtt{y}$ and $\mathtt{z}$, which can be seen as a constraint, that is, a requirement that states which combinations of values from the variable domains are admitted. In this case, $\mathtt{x}$ equal to 9, $\mathtt{y}$ equal to 12 and $\mathtt{z}$ equal to 1. This can be expressed, at the same time, as constraints of the form $\mathtt{x} = 9, \mathtt{y} = 12, \mathtt{z} = 1$.

With respect to the membership relations, we can express membership of points to a spatial object, for instance $(1, 6) \in \mathtt{line}[(1, 2), (1, 9)]$, which is trivially true, but we can also add variables. However, we can also use the membership relation for expressing that a spatial object belongs to a set of spatial objects. For instance, $\mathtt{A} \in \{\mathtt{line}[(1, 2), (1, 9)], (9, 10)\}$. In this case, $\mathtt{A}$ ranges on two spatial objects, $\mathtt{line}\ [(1, 2), (1, 9)]$ and $(9, 10)$. Combining membership relations we can build our first CSP using spatial data. For instance, $(1, \mathtt{x}) \in \mathtt{A}, \mathtt{A} \in \{\mathtt{line}[(1, 2), (1, 9)], (1, 10)\}$. The allowed set of values for $\mathtt{x}$ is then $[2, 9]$ and 10. This can be expressed by means of two constraints $\mathtt{x} \in [2, 9]$ and $\mathtt{x} = 10$.

With respect to the set operations, we consider *union*, *intersection*, and *difference*. We have two cases, for spatial objects and sets of spatial objects. For instance, $(1, 2) \cup \mathtt{line}[(1, 2), (9, 10)]$ represents the set of points belonging to both spatial objects. Therefore, the constraint $(1, \mathtt{x}) \in (1, 2) \cup \mathtt{line}[(1, 2), (9, 10)]$ has a solved form $\mathtt{x} = 2$. An accurate treatment needs the *difference*. When the difference involves two spatial objects, the result of performing such operation can be a spatial object with holes. For this reason, we allow solved forms in our framework, including geometric figures like $\mathtt{line}[(1, 2), (9, 10)] - (1, 2)$ representing a *closed*

Table 1. Spatial Problem and CLP program




```
(1) city(c1,(16, 24)).          (5) city(c5,(60, 32)).
(2) city(c2,(27, 41)).          (6) city(c6,(75, 55)).
(3) city(c3,(40, 28)).          (7) tourist(t1,(12, 16)).
(4) city(c4,(50, 10)).          (8) tourist(t2,(23, 37)).
                                (9) tourist(t3,(57, 6)).
(10) road(road311,Road):- Road ∈ {line[(75,55),(60,32),(40,28)],line[(75,55),
      (40,28),(16,24),(12,16)],line[(60,32),(50,10),(16,24)],
      line[(57,6),(50,10),(40,28),(27,41),(23,37)]}.
(11) river(smallriver,River):-River=line[(52,60),(60,41),(50, 16),(0,28)].
(12) lake(biglake,Lake):- Lake=polygon[(64, 38),(62,35),(60, 36),(60, 34),
      (56, 36),(58, 40),(62, 42)].
(13) bridge(Bridge):-Bridge=Road ∩ River |
      road(Roadname,Road),river(Rivername,River).
(14) cities(C):-C ∈ {C1, C2, C3, C4, C5, C6} |
      city(c1,C1),city(c2,C2),city(c3,C3),city(c4,C4),city(c5,C5),city(c6,C6).
```

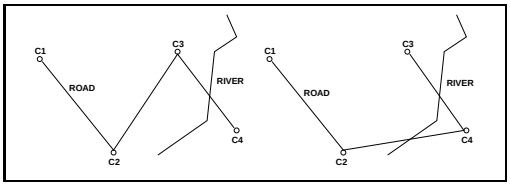
Table 2. Queries and Answers

	Query	CSP	Answer
(1)	List the part of the river that passes through the lake	$: -\texttt{Parts} \in \texttt{setsegments}(\texttt{River}),$ $\texttt{Through} = \texttt{Parts} \cap \texttt{Lake},$ $\mid \texttt{river}(\texttt{smallriver}, \texttt{River}), \texttt{lake}(\texttt{biglake}, \texttt{Lake}).$	$\texttt{Through} = \texttt{line}[(60, 41), (56, 35)],$ $\texttt{Parts} = \texttt{line}[(60, 41), (50, 16)]$
(2)	Report the part of the road that touches the lake and its length	$\texttt{Parts} \in \texttt{setsegments}(\texttt{Road}),$ $\texttt{Touches} = \texttt{Parts} \cap \texttt{Lake},$ $\texttt{Parts neighbor Lake},$ $\texttt{L} = \texttt{length}(\texttt{Touches}), \texttt{L} \neq 0$ $\mid \texttt{road}(\texttt{road311}, \texttt{Road}), \texttt{lake}(\texttt{biglake}, \texttt{Lake}).$	$\texttt{Touches} = \texttt{line}[(64, 36), (62, 35)],$ $\texttt{Parts} = \texttt{line}[(17, 55), (60, 32)], \texttt{L} = 3.5$
(3)	Report all the cities that can use water from the river (at most within 10 km)	$\texttt{D} = \texttt{mindist}(\texttt{City}, \texttt{River}),$ $\texttt{D} \in [0, 10]$ $\mid \texttt{cities}(\texttt{City}),$ $\texttt{river}(\texttt{smallriver}, \texttt{River}).$	$\texttt{D} = 0, \texttt{City} = (16, 24);$ $\texttt{D} = 9, \texttt{City} = (40, 28);$ $\texttt{D} = 6, \texttt{City} = (50, 10);$ $\texttt{D} = 3, \texttt{City} = (60, 32)$
(4)	Find tourist resorts that are within 7.5 km of a city	$\texttt{Near} \in \{\texttt{T1}, \texttt{T2}, \texttt{T3}\}$ $\texttt{rangeq circle}(\texttt{City}, 7.5),$ $\mid \texttt{tourist}(\texttt{t1}, \texttt{T1}), \texttt{tourist}(\texttt{t2}, \texttt{T2}), \texttt{tourist}(\texttt{t3}, \texttt{T3}),$ $\texttt{cities}(\texttt{City}).$	$\texttt{Near} = (23, 37), \texttt{City} = (27, 41)$
(5)	Which city is the closest to any tourist resort?	$\texttt{Closest} \in \texttt{pmindist}(\{\texttt{C1}, \texttt{C2}, \texttt{C3}, \texttt{C4}, \texttt{C5},$ $\texttt{C6}\}, \{\texttt{T1}, \texttt{T2}, \texttt{T3}\})$ $\mid \texttt{tourist}(\texttt{t1}, \texttt{T1}), \texttt{tourist}(\texttt{t2}, \texttt{T2}), \texttt{tourist}(\texttt{t3}, \texttt{T3}),$ $\texttt{city}(\texttt{c1}, \texttt{C1}), \texttt{city}(\texttt{c2}, \texttt{C2}), \texttt{city}(\texttt{c3}, \texttt{C3}),$ $\texttt{city}(\texttt{c4}, \texttt{C4}), \texttt{city}(\texttt{c5}, \texttt{C5}), \texttt{city}(\texttt{c6}, \texttt{C6}).$	$\texttt{Closest} = < (27, 41), (23, 37) >$
(6)	List the cities to the north of C3	$(\texttt{X}, \texttt{Y}) = \texttt{City}, (\texttt{U}, \texttt{V}) = (40, 28), \texttt{Y} \in [\texttt{U}, 60],$ $\mid \texttt{cities}(\texttt{City}).$	$(\texttt{X}, \texttt{Y}) = (27, 41), (\texttt{U}, \texttt{V}) = (40, 28);$ $(\texttt{X}, \texttt{Y}) = (60, 32), (\texttt{U}, \texttt{V}) = (40, 28);$ $(\texttt{X}, \texttt{Y}) = (75, 55), (\texttt{U}, \texttt{V}) = (40, 28)$
(7)	Report bridges which are not in a city	$\texttt{Bridge} \notin \{\texttt{C1}, \texttt{C2}, \texttt{C3}, \texttt{C4}, \texttt{C5}, \texttt{C6}\}$ $\mid \texttt{city}(\texttt{c1}, \texttt{C1}), \texttt{city}(\texttt{c2}, \texttt{C2}), \texttt{city}(\texttt{c3}, \texttt{C3}),$ $\texttt{city}(\texttt{c4}, \texttt{C4}), \texttt{city}(\texttt{c5}, \texttt{C5}), \texttt{city}(\texttt{c6}, \texttt{C6}),$ $\texttt{bridge}(\texttt{Bridge}).$	$\texttt{Bridge} = (59, 43);$ $\texttt{Bridge} = (56, 31);$ $\texttt{Bridge} = (46, 27)$

point-based geometric figure with holes. These holes are an arbitrary union of closed point-based geometric figures.

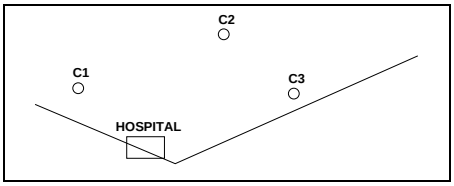
The third kind of operations are the metric ones (see figure 1). They are based on distances, and also handle spatial objects and sets of spatial objects. We can suppose a fixed distance $d(\_, \_)$ for pairs of points: Euclidean, Manhattan, etc. The fixed distance is not relevant, and we can suppose any of them. In this case we have `mindist`, `maxdist`, `pmindist`, `pmaxdist`, `omindist` and `omaxdist`. The first two represent the minimum and maximum distance between two spatial objects. For instance $\texttt{x} = \texttt{mindist}(\texttt{line}[(1, 2), (1, 10)], \texttt{line}[(3, 2), (12, 15)])$, is a constraint with solved form $\texttt{x} = 3$. Distance-based queries have a particularity. They are *rigid* operations, that is, objects for which a minimum (maximum) distance is computed must be a fully defined object, avoiding the use of ranged variables in their description. For instance, it is forbidden to formulate a constraint satisfaction problem of the form $\texttt{x} = \texttt{mindist}((1, \texttt{y}), (1, 1)), \texttt{y} \in [1, 3]$. The operations `pmindist` and `pmaxdist` (resp. `omindist` and `omaxdist`) get the set of pairs of points (resp. objects) with the minimum (resp. maximum) distance. For instance, $\texttt{P} \in \texttt{pmindist}(\texttt{line}[(1, 2), (1, 10)], \texttt{line}[(3, 2), (12, 15)])$, requires the variable `P`, a new kind of variable, to be a member of a set of pairs of spatial objects. This set represents the set of pairs of points which are at the minimum distance. The solved form will be $\texttt{P} \in \{\{< (1, 2), (3, 2) >$

Table 3. CLP Program for the Bridge Problem




```
(1) cities(C):- C ∈ { C1,C2,C3,C4 }
    | city(c1,C1),city(c2,C2),city(c3,C3),city(c4,C4).
(2) road(line[A,B,C,D]):- A≠B,A≠C, A≠D,B≠C,
    B≠D,C≠D |  cities(A),cities(B),cities(C),cities(D).
Query:
:-P ∈ Road ∩ River, Road disjoint (River-P)
   | river(Rivername,River),road(Road).
```

Table 4. CLP Program for the Hospital Problem




```
Query:
:- P ∈ pmindist({Road},{C1,C2,C3})  |  city(c1,C1),city(c2,C2),city(c3,C3) .
```

}}. Therefore, we have to handle in addition to set of objects, set of pairs of objects, using a couple of brackets as notation.

With respect to topological relations (see figure 1), they are usual binary topological relations on spatial objects and set of spatial objects. For instance, $\mathtt{A} \in \{(1,4),(4,5)\}$, $\mathtt{B} \in \{\mathtt{line}[(1,4),(1,10)],(4,5)\}$, `A inside B` has as solved form $\mathtt{A} = (1,4), \mathtt{B} = \mathtt{line}[\,(1,4),(1,10)] \vee \mathtt{A} = (4,5), \mathtt{B} = (4,5)$. Similarly, the constraint $\mathtt{P} \in \{(1,4),(4,5)\}$ `inside` $\{\mathtt{line}[\,(1,4),(1,10)],(4,5)\}$ has as solved form $\mathtt{P} \in \{\{< (1,4), \mathtt{line}[\,(1,4),(1,10)] >, < (4,5),(4,5) >\}\}$. A special case is `rangeq`, which is a topological operation for range queries, such as the occurrence of a spatial object or a set of spatial objects that fall on a certain distance from a point or on a window. With this aim, we consider as special case of spatial data types, the circle defined by a point and a given radius (`circle`) and a window given by two points (`window`). For instance, $\mathtt{A} \in \{(1,2),(9,10)\}$ `rangeq` $\mathtt{circle}((0,0),3)$ has as solved form $\mathtt{A} = (1,2)$.

In addition, we have considered a set of structural operations in order to compute the size of spatial objects, and the set of points and line segments which conform the frontier of a spatial object (and a set of spatial objects). They are `setpoints`, `setsegments`, `area`, `length` and `perimeter`.

Finally, our constraint system can handle *negation*, but with restricted use. Equality and membership relations can be used in negative form $\neq$ and $\notin$, but variables used in them (which can be referred to distances, coordinates of points, spatial objects and pairs of spatial objects) must be ranged in *at least one positive constraint* in order to formulate a constraint satisfaction problem. This restricted version of negation prevents anomalies in the use of negation such as infinite answers for a constraint satisfaction problem. For instance, $\mathtt{C} = \mathtt{A} \cap \mathtt{B}, \mathtt{A} \notin \{(1,2),(3,4)\}, \mathtt{B} \notin \{(1,2)\}$ cannot be represented in solved form as a finite disjunction of equality and membership relations for `C`, `A` and `B`. The problem is that there is a *infinite and not finitely representable set of solutions*.

2.2 Examples of CLP Programs

In table 1 we show an example of CLP program handling this kind of constraints. In this example, we have spatial information about Cities (set of points), Tourist Resorts (set of points), Roads (set of lines), River (line) and Lake (polygon). The left-hand side figure illustrates the spatial data, and assuming the work area in the window (0, 0) for lower-left corner and (80, 60) for the upper-right corner, the CLP program of the right-hand side represents this information.

Facts are supposed to be either atoms or constrained atoms. This is the case of rules (1) to (12), where the shape of each spatial object is declared. *Rules* can be used for defining some spatial operations. For instance, in the example rule (13) computes the Bridges and rule (14) collects the positions of the Cities. Note that the Road is expressed as a set of line segments due to we cannot handle graphs in our representation. With respect to the previous CLP program, we can request the spatial queries (*topological*, *distance-based* and *directional*) of table 2.

In our instance, a CLP program and query can be used for solving *combinatorial problems* as usual in constraint programming. For instance, we can suppose the following problem. We have four Cities for which a Road is planned to be build to connect them. In addition, we can suppose that a River crosses the area in which these Cities are located. The spatial problem is how to connect the Cities by means of the Road in such a way that a unique Bridge is needed. The left-hand side of the table 3 show two feasible solutions, one of them violates the condition a unique Bridge has to be build. This spatial problem can be model by means of CLP as is shown in the right-hand side of table 3 (assuming the same representation as before for cities and the river).

Finally, our framework can be used for modelling *optimization problems*. For instance, we can suppose that now we would like to build a new Hospital near three Cities but in some point of the main Road. In addition, in order to provide service to each City, the distance should be minimized, that is, distance is optimized. Now, we can describe and formulate the problem as in the table 4.

3. CLP($\mathcal{SSO}$)

In this section we formally present the constraint system and the instance CLP($\mathcal{SSO}$). Firstly we present our constraint system.

3.1 Constraint System

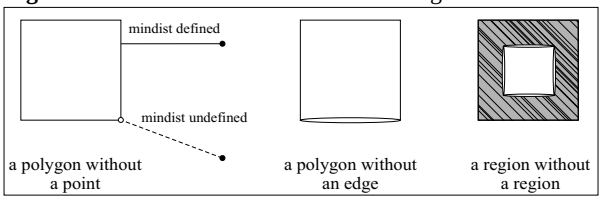
Let `NVar` be a set of numerical variables $x, y, \ldots$, `OVar` a set of variables for objects $A, B, \ldots$, `PVar` a set of variables for pairs of objects $P, Q, \ldots$, and $\mathbb{R}$ the set of real numbers.

Now, we have a data constructor for pairs of coordinates $(\_, \_)$: $\mathtt{NType} \times \mathtt{NType} \rightarrow \mathtt{Point}$, where $\mathtt{NType} = \mathtt{NVar} \cup \mathbb{R}$, a set of data constructors for spatial data types: $\emptyset :\rightarrow \mathtt{SDT}$, $\mathtt{line} : [\mathtt{Point}] \rightarrow \mathtt{Line}$, $\mathtt{polygon} : [\mathtt{Point}] \rightarrow \mathtt{Polygon}$, and $\mathtt{region} : [\mathtt{Point}] \rightarrow \mathtt{Region}$ where $\mathtt{SDT} = \mathtt{OVar} \cup \mathtt{Point} \cup \mathtt{Line} \cup \mathtt{Polygon} \cup \mathtt{Region}$.

Table 5. Semantics of Operations

$\cup, \cap, -$: SDT $\times$ SDT $\rightarrow$ SetSDT, SetSDT $\times$SetSDT $\rightarrow$ SetSDT
$[\![O_1 \Theta O_2]\!]_{(\mu,\Delta,\Omega)} = [\![O_1]\!]_{(\mu,\Delta,\Omega)} \Theta [\![O_2]\!]_{(\mu,\Delta,\Omega)}$
$[\![S_1 \Theta\ S_2]\!]_{(\mu,\Delta,\Omega)} = [\![S_1]\!]_{(\mu,\Delta,\Omega)} \Theta\ [\![S_2]\!]_{(\mu,\Delta,\Omega)}$, where $\Theta \in \{\cup, \cap, -\}$
$=, \neq$: SDT $\times$ SDT $\rightarrow$ Boolean, PSDT $\times$ PSDT $\rightarrow$ Boolean
$\in, \notin$: Point $\times$ SDT $\rightarrow$ Boolean, SDT $\times$ SetSDT $\rightarrow$ Boolean, PSDT $\times$ SetPSDT $\rightarrow$ Boolean
$[\![O_1 = O_2]\!]_{(\mu,\Delta,\Omega)} =_{def} true$ if $\Delta\mu(O_1) = \Delta\mu(O_2)$
$[\![(p_1, p_2) \in O]\!]_{(\mu,\Delta,\Omega)} =_{def} true$ if $\mu((p_1, p_2)) \in \Delta\mu(O)$
mindist, maxdist : SDT $\times$ SDT $\rightarrow$ $\mathbb{R}$, SetSDT $\times$ SetSDT $\rightarrow$ $\mathbb{R}$
pmindist, pmaxdist : SDT $\times$ SDT $\rightarrow$ PSDT, SetSDT $\times$ SetSDT $\rightarrow$ PSDT
omindist, omaxdist : SetSDT $\times$ SetSDT $\rightarrow$ PSDT
$[\![mindist(O_1, O_2)]\!]_{(\mu,\Delta,\Omega)} =_{def}$
$min\{d((p_1, p_2), (q_1, q_2)) \mid (p_1, p_2) \in [\![O_1]\!]_{(\mu,\Delta,\Omega)}, (q_1, q_2) \in [\![O_2]\!]_{(\mu,\Delta,\Omega)}\}$
if $mindist([\![O_1]\!]_{(\mu,\Delta,\Omega)}, [\![O_2]\!]_{(\mu,\Delta,\Omega)}) \neq \perp$, and $\perp$, otherwise
$[\![mindist(S_1, S_2)]\!]_{(\mu,\Delta,\Omega)} =_{def}$
$min\{mindist(\mathcal{G}_1, \mathcal{G}_2) \mid \mathcal{G}_1 \in [\![S_1]\!]_{(\mu,\Delta,\Omega)}, \mathcal{G}_2 \in [\![S_2]\!]_{(\mu,\Delta,\Omega)}, mindist(\mathcal{G}_1, \mathcal{G}_2) \neq \perp\}$
inside, outside, overlap, disjoint, meet, cuts, neighbor,
mcovers, ncovers, mcoveredby, ncoveredby : SDT $\times$ SDT $\rightarrow$ Boolean, SetSDT $\times$ SetSDT $\rightarrow$ SetPSDT
rangeq : SetSDT $\times$ Range $\rightarrow$ SetSDT
$[\![O_1\ inside\ O_2]\!]_{(\mu,\Delta,\Omega)} =_{def} true$ if $[\![O_1]\!]_{(\mu,\Delta,\Omega)} \subseteq [\![O_2]\!]_{(\mu,\Delta,\Omega)}$
$[\![S_1\ inside\ S_2]\!]_{(\mu,\Delta,\Omega)} =_{def} \{(\mathcal{G}_1, \mathcal{G}_2) \mid \mathcal{G}_1 \in [\![S_1]\!]_{(\mu,\Delta,\Omega)}, \mathcal{G}_2 \in [\![S_2]\!]_{(\mu,\Delta,\Omega)}$, and $\mathcal{G}_1 \subseteq \mathcal{G}_2\}$
area : Region $\rightarrow$ $\mathbb{R}$, length : Line $\rightarrow$ $\mathbb{R}$, perimeter : Polygon $\cup$ Region $\rightarrow$ $\mathbb{R}$,
area, length, perimeter : SetSDT $\rightarrow$ $\mathbb{R}$ setpoints, setsegments : SDT $\rightarrow$ SetSDT, SetSDT $\rightarrow$ SetSDT
$[\![area(S)]\!]_{(\mu,\Delta,\Omega)} =_{def} \sum_{\mathcal{G} \in [\![S]\!]_{(\mu,\Delta,\Omega)}, \mathcal{G}\ is\ a\ region} \mathcal{A}(\mathcal{G})$

Figure 2. Closed Point-Based Geometric Figures with Holes



We have also a data constructor for pairs of *spatial data types*: $< \_, \_ >$: SDT $\times$ SDT $\rightarrow$ PairSDT, and PSDT = PVar $\cup$ PairSDT. With respect to sets of spatial objects and pairs of spatial objects, we have a set of data constructors for sets of (pairs of) spatial objects: $\emptyset :\rightarrow$ SetSDT, $\{\_|\_\}$: SDT $\times$ SetSDT $\rightarrow$ SetSDT, $\emptyset :\rightarrow$ SetPSDT, and $\{\{\_|\_\}\}$: PSDT $\times$ SetPSDT $\rightarrow$ SetPSDT.

Finally, we have data constructors for a special kind of spatial data types, called Window, Circle such that window : Point $\times$ Point $\rightarrow$ Window, circle : Point $\times$ NType $\rightarrow$ Circle and Range = Window $\cup$ Circle. In addition, Interval builds closed real intervals: $[\_, \_]$: NType $\times$ NType $\rightarrow$ Interval.

Now we present a set of *operations* over the defined types. Some operations are *partial*, that is, are not defined for some arguments. Assuming the symbol $\perp$ representing a special value called *undefined*, we can consider the following types. For the type NType, we have the binary operations $=, \neq$: NType $\times$ NType $\rightarrow$ Boolean, and $\in, \notin$: NType $\times$ Interval $\rightarrow$ Boolean. The formal semantics is as follows.

Given a valuation μ of numerical variables into $\mathbb{R} \cup \{\perp\}$, then we denote by $\mu(\mathtt{n})$, where $\mathtt{n} \in$ NType, the value of n under the valuation μ as the real number (or undefined), defined as $\mu(n) = \mu(x)$ if $n \equiv x$ is a variable and $\mu(n) = n$, otherwise.

Now, given $n_1, n_2, n_3 \in$ NType, and a valuation μ, we define $\perp = \perp, \perp \neq \perp$ are both false; $n_1 = n_2$ iff $\mu(n_1) = \mu(n_2)$; $n_1 \neq n_2$ iff $\mu(n_1) \neq \mu(n_2)$; $n_1 \in [n_2, n_3]$ iff $\mu(n_2) \leq \mu(n_1) \leq \mu(n_3)$; and finally, $n_1 \notin [n_2, n_3]$ iff $\mu(n_2) > \mu(n_1)$ or $\mu(n_1) > \mu(n_3)$. $\perp$ represents undefined and therefore an incomparable value.

We denote by $Val(\mathtt{NType})$ the set of valuations of numerical variables into $\mathbb{R} \cup \{\perp\}$. Now, the value of a spatial object O under a valuation $\mu \in Val(\mathtt{NType})$, denoted by $\mu(O)$, is a *closed point-based geometric figure* ($\mathcal{FIG}$), and it is defined as follows:

- $\mu((p_1, p_2)) =_{def} \{(\mu(p_1), \mu(p_2))\}$ if none of the $\mu(p_i)$ are $\perp$; and $\emptyset$, otherwise.
- $\mu(line[(p_1, p_2), (p_3, p_4), \ldots, (p_{n-1}, p_n)]) =_{def} \{(r, s) \mid r = \alpha \times \mu(p_{2k+1}) + (1-\alpha) \times \mu(p_{2k+3}), s = \alpha \times \mu(p_{2k+2}) + (1-\alpha) \times \mu(p_{2k+4}), \alpha \in [0, 1], 0 \leq k \leq n-4/2, k \in \mathbb{N}, \alpha \in \mathbb{R}\}$, if none of the $\mu(p_i)$ is $\perp$; and $\emptyset$, otherwise, where $n \geq 4$, $n \in \mathbb{N}$, and each $p_i \in$ NType.
- $\mu(polygon[(p_1, p_2), (p_3, p_4), \ldots, (p_{n-1}, p_n)]) =_{def}$ $\{(r, s) \mid r = \alpha \times \mu(p_{2k+1}) + (1-\alpha) \times \mu(p_{2k+3}), s = \alpha \times \mu(p_{2k+2}) + (1-\alpha) \times \mu(p_{2k+4}), \alpha \in [0, 1], 0 \leq k \leq n-4/2, k \in \mathbb{N}, \alpha \in \mathbb{R}\}$ $\cup \{(r, s) \mid r = \alpha \times \mu(p_{n-1}) + (1-\alpha) \times \mu(p_1), s = \alpha \times \mu(p_n) + (1-\alpha) \times \mu(p_2), \alpha \in [0, 1], \alpha \in \mathbb{R}\}$, if neither of the $\mu(p_i)$ is $\perp$; and $\emptyset$, otherwise, where $n \geq 4$, $\mathtt{n} \in \mathbb{N}$
- $\mu(region[(p_1, p_2), (p_3, p_4), \ldots, (p_{n-1}, p_n)]) =_{def}$ $\{(r, s) \mid r = \alpha \times \mu(p_{2i+1}) + \beta \times \mu(p_{2j+1}), s = \alpha \times \mu(p_{2i+2}) + \beta \times \mu(p_{2j+2}), \alpha + \beta = 1, \alpha, \beta \geq 0, \alpha, \beta \in \mathbb{R}, 0 \leq i \leq n - 2/2, i \in \mathbb{N}, 0 \leq j \leq n - 2/2, j \in \mathbb{N}\}$ if neither of the $\mu(p_i)$ is $\perp$; and $\emptyset$, otherwise, where $n \geq 4$, $n \in \mathbb{N}$

Spatial objects can be grouped in both sets of objects and sets of pairs of objects. The operations on sets of (pairs of) spatial objects are interpreted into *closed point-based geometric figures with holes* ($\mathcal{FH}$).

DEFINITION 3.1. **Figures with Holes.** *Given $\mathcal{FIG}$'s: F, H_i, a figure with holes $\mathcal{G}$ has the form $F - \cup_{k \geq i \geq 1} H_i$ where $k \geq 0$, $H_i \cap H_j = \emptyset$ if $i \neq j$ and $\cup_{k \geq i \geq 1} H_i \subseteq F$. The set $\cup_{k \geq i \geq 1} H_i$ is called the holes of $\mathcal{G}$. The set of figures with holes is denoted by $\mathcal{FH}$. In addition:*

(i) $\bar{\mathcal{G}}$ *denotes the figure obtained from* $\mathcal{G}$*, adding the topological frontier;*

(ii) $mindist(\mathcal{G},\mathcal{G}')$ *(resp.* $maxdist(\mathcal{G},\mathcal{G}')$*) denotes the minimum (resp. maximum) distance from* $\mathcal{G}$ *to* $\mathcal{G}'$*, defined as follows:* $mindist(\mathcal{G},\mathcal{G}') =_{def} \perp$ *if* $min\{d((p,q),(r,s)) \mid (p,q) \in \mathcal{G}, (r,s) \in \mathcal{G}'\} < min\{d((p,q),(r,s)) \mid (p,q) \in \bar{\mathcal{G}}, (r,s) \in \bar{\mathcal{G}}'\}$ *and otherwise* $mindist(\mathcal{G},\mathcal{G}') =_{def} min\{d((p,q),(r,s)) \mid (p,q) \in \mathcal{G}, (r,s) \in \mathcal{G}'\}$ *and analogously for* $maxdist$*;*

(iii) $\mathcal{L}(\mathcal{G})$ *denotes the length of a line* $\mathcal{G}$*;*

(iv) $\mathcal{P}(\mathcal{G})$ *denotes the perimeter of a polygon or region* $\mathcal{G}$*;*

(v) $\mathcal{A}(\mathcal{G})$ *denotes the area of a region* $\mathcal{G}$*. The three are defined as* $\perp$ *whenever the figure has a hole.*

(vi) $\mathcal{PS}(\mathcal{G})$ *(resp.* $\mathcal{LS}(\mathcal{G})$*) denotes the set of points (resp. line segments of the frontier) which define* $\mathcal{G}$*.*

An element of $\mathcal{FH}$ can have holes, which are in the topological frontier of the object (see figure 2). When the distance (minimum or maximum) from an element of $\mathcal{FH}$ to another element of $\mathcal{FH}$ is computed, it could be that the nearest points are in the holes. In this case, we consider both distances are undefined (see figure 2). The elements of $\mathcal{FH}$ can be grouped into sets, denoted by $\mathcal{SFH}$, and sets of pairs of elements of $\mathcal{FH}$, denoted by $\mathcal{SPFH}$. Grouping $\mathcal{FH}$ into sets we can handle more complex spatial objects like graphs, among others. Sets of (pairs of) spatial objects can have variables representing spatial objects. Given a valuation Δ of variables of spatial objects into $\mathcal{FH}$ then:

- $\Delta(\mu(\{O_1,\ldots,O_n\})) =_{def} \{\Delta(\mu(O_1)),\ldots,\Delta(\mu(O_n))\}$ and
- $\Delta(\mu(\{\{< O_1, O_2 >,\ldots,< O_{n-1}, O_n >\}})) =_{def}$ $\{(\Delta(\mu(O_1)), \Delta(\mu(O_2))),\ldots,(\Delta(\mu(\ O_{n-1})), \Delta(\mu(O_n)))\}$

where each $O_i \in$ SDT. We can also consider valuations of variables representing pairs of spatial objects into pairs of $\mathcal{FH}$'s, named as $\mathcal{PFH}$. We denote by $Val($SDT$)$ the set of valuations into $\mathcal{FH}$'s and $Val($PSDT$)$ the set of valuations into $\mathcal{PFH}$.

In the table 5, we present the main operations on (sets of) spatial objects. They are defined with regard to $\mu \in Val($NType$)$, $\Delta \in Val$ (SDT), and $\Omega \in Val($PSDT$)$. The definitions of such operations are not very complex, and they have only to take into account that elements of $\mathcal{FH}$ can have holes, and therefore some operations can be not defined, that is, they are defined as $\perp$, mainly when the operations involves (minimum and maximum) distances. We have included the main cases. A full version can be found in [1].

3.2 Spatial Constraint Satisfaction Problem

In this subsection we define what is a spatial constraint satisfaction problem, by presenting its general form, a special class which represents the solved forms, and a set of conditions in order to ensure that each spatial constraint satisfaction problem is equivalent to a solved form.

DEFINITION 3.2. **Spatial CSP.**
A spatial constraint satisfaction problem (SCSP) Γ *is a conjunction of typed boolean operations over the types* NType, SDT, PSDT, SetSDT, SetPSDT, Range *and* Interval *of the form* $\Gamma \equiv \varphi_1,\ldots,\varphi_n$.

DEFINITION 3.3. **Solution.** *A triple* (μ,Δ,Ω)*, where* $\mu \in Val($NType$)$, $\Delta \in Val($SDT$)$ *and* $\Omega \in Val($PSDT$)$ *is a* solution *of a SCSP* $\Gamma \equiv \varphi_1,\ldots,\varphi_n$ *if each* $[\![\varphi_i]\!]_{(\mu,\Delta,\Omega)}$ *is equal to* $true$*. A SCSP is called* satisfiable *whenever has at least a solution.*

DEFINITION 3.4. **Solved SCSP.** *A* solved SCSP Π *is a disjunction of spatial constraint satisfaction problems of the form* $\Pi \equiv \bigvee_{i\geq 1} \Gamma_i$ *where each* Γ_i *is of the form:* $\Gamma_i \equiv \varphi_1,\ldots,\varphi_n$*, and each* φ_j *is a solved constraint of the form:*

1. $x = n$ *where* $x \in$ NVar *and* $n \in \mathbb{R} \cup Dom(\Gamma_i)$
2. $x \in [n_1,n_2] - \cup_{k\geq j\geq 1}[h_j,h_{j+1}]$ *where* $x \in$ NVar*,* n_1,n_2*,* $h_j,h_{j+1} \in \mathbb{R} \cup Dom(\Gamma_i)$ *and* $k \geq 0$*; in addition,* $\Gamma_i(n_1) \leq \Gamma_i(h_j) \leq \Gamma_i(h_{j+1}) \leq \Gamma_i(n_2)$*, for all* $k \geq j \geq 1$*, and* $[\Gamma_i(h_j),\Gamma_i(h_{j+1})] \cap [\Gamma_i(h_l),\Gamma_i(h_{l+1})] = \emptyset$ *if* $j \neq l$*.*
3. $(x,y) \in O$*, where* $x,y \in$ NVar *and* $O \in \mathcal{FH} \cup Dom(\Gamma_i)$
4. $A = O - \cup_{k\geq j\geq 1} H_j$ *where* $A \in$ OVar*,* $O, H_j \in \mathcal{FH} \cup Dom(\Gamma_i)$ *and* $k \geq 0$*; in addition,* $\cup_{k\geq j\geq 1}\Gamma_i(H_j) \subseteq \Gamma_i(O)$*, and* $\Gamma_i(H_j) \cap \Gamma_i(H_l) = \emptyset$ *if* $j \neq l$*.*
5. $A \in S$ *where* $A \in$ OVar *and* $\Gamma_i(S) \in \mathcal{SFH}$*.*
6. $P = Q$ *where* $P \in$ PVar*, where* $Q \in Dom(\Gamma_i)$ *or* $Q \equiv < Q_1, Q_2 >$*,* $Q_1, Q_2 \in \mathcal{FH} \cup Dom(\Gamma_i)$
7. $P \in SP$ *where* $P \in$ PVar *and* $\Gamma_i(SP) \in \mathcal{SPFH}$*.*

where there exists at most one solved constraint for each variable x*,* A *and* P *in each* Γ_i*. Variables* x*,* A *and* P *in the definition represent the domain of* Γ_i*, denoted by* $Dom(\Gamma_i)$*.* $\Gamma_i(x)$ *(resp.* $\Gamma_i(A), \Gamma_i(P)$*) denote the set of solutions represented by the solved constraints in which occurs* x *(resp.* A *and* P*), and* $\Gamma_i(S), \Gamma_i(SP)$ *the set of* $\mathcal{SFH}$*'s, respectively* $\mathcal{SPFH}$*'s obtained from them.*

DEFINITION 3.5. **Solutions of a Solved SCSP.**
A solved SCSP $\Pi \equiv \bigvee_{i\geq 1} \Gamma_i$ *defines a set of solutions, denoted by* $Sol(\Pi)$*, and defined as* $Sol(\Pi) = \cup_{\Gamma_i} Sol(\Gamma_i)$*, where each* Γ_i *defines a set of solutions which consists on triples* (μ,Δ,Ω)*, recursively defined as follows:*

$$
\begin{aligned}
\mu =_{def}\ & \\
& \cup_{x=n\in\Gamma_i}\{x/\mu(n)\} \\
& \cup_{\substack{(x\in[n_1,n_2]-\cup_{k\geq j\geq 1}[h_j,h_{j+1}])\in\Gamma_i,\\ r\in[\mu(n_1),\mu(n_2)], r\notin\cup_{k\geq j\geq 1}[\mu(h_j),\mu(h_{j+1})]}}\{x/r\} \\
& \cup_{((x,y)\in O)\in\Gamma_i,(p,q)\in\Delta\mu(O))}\{x/p, y/q\} \\
\\
\Delta =_{def}\ & \cup_{(A=O-\cup_{k\geq j\geq 1}H_j)\in\Gamma_i}\{A/\Delta\mu(O) - \cup_{k\geq j\geq 1}\Delta\mu(H_j)\} \\
& \cup_{(A\in S)\in\Gamma_i,\mathcal{G}\in\Delta\mu(S)}\{A/\mathcal{G}\} \\
\\
\Omega =_{def}\ & \cup_{P=Q\in\Gamma_i}\{P/\Omega\Delta\mu(Q)\} \\
& \cup_{(P\in SP)\in\Gamma_i,(\mathcal{G}_1,\mathcal{G}_2)\in\Omega\Delta\mu(SP)}\{P/(\mathcal{G}_1,\mathcal{G}_2)\}
\end{aligned}
$$

Finally, $\Gamma_i(x) =_{def} \mu(x), \Gamma_i(A) =_{def} \Delta(A)$ *and* $\Gamma_i(P) =_{def} \Omega(P)$*,* $\Gamma_i(n) =_{def} n$ *if* $n \in \mathbb{R}$*,* $\Gamma_i(O) =_{def} O$ *if* $O \in \mathcal{FH}$*,* $\Gamma_i(P) =_{def} P$ *if* $P = (\mathcal{G}_1,\mathcal{G}_2)$ *and* $\mathcal{G}_1,\mathcal{G}_2 \in \mathcal{FH}$ *for each* Γ_i*.*

In order to ensure that any spatial constraint satisfaction problem is equivalent to a solved SCSP we have to require that the SCSP follows some syntactic conditions. We call *safe SCSP* to SCSP ensuring these conditions.

DEFINITION 3.6. **Range Restricted Variables.** *We say that a variable* $\alpha \in$ NVar $\cup$ OVar $\cup$ PVar *occurring on a SCSP* Γ *is* range restricted *if there exist (i) an equality constraint* $\alpha = E$ *or (ii) membership constraint* $\alpha \in E$*, such that* E *is ground (without variables) or all the variables of* E *are range restricted.*

We call that a variable is *strongly range restricted* considering only the case (i) in the above definition. Finally, we require that distance-based and structural operations are *rigid*.

DEFINITION 3.7. **Rigid Constraints.** *Given a set of variables* $\mathcal{V}$*, a spatial constraint is* $\mathcal{V}$-rigid *if whenever an operation involving* $\Theta \in \{$mindist, maxdist, pmindist, pmaxdist, omindist, omaxdist, area, length, perimeter$\}$ *is included in the form* $\Theta(O_1,O_2)$ *(or* $\Theta(O)$*) (resp.* $\Theta(S_1,S_2)$ *(or* $\Theta(S)$*)) then* O_1 *and*

O_2 (or O) (resp. S_1 and S_2 (or S)) only contain variables of $\mathcal{V}$ strongly range restricted.

DEFINITION 3.8. **Safe SCSP.** *Given a set of variables $\mathcal{V}$, a $\mathcal{V}$-safe SCSP is a SCSP where all the variables of $\mathcal{V}$ are range restricted, and all the spatial constraints are $\mathcal{V}$-rigid.*

THEOREM 3.1. **Safety.** *Each satisfiable $var(\Gamma)$-safe SCSP Γ is equivalent to a solved SCSP.*

3.3 CLP Programs

Now, we present the framework of CLP($\mathcal{SSO}$). Firstly, we define a CLP($\mathcal{SSO}$) program, and after the fixed-point operational semantics.

Now we have to assume a *signature* Σ which consists on a set of predicate symbols $p_1, \ldots, p_n$, and a set of function symbols $f_1, \ldots, f_m$ and constants $c_1, \ldots, c_k$, and in addition the special typed function symbols $[\_,\_], (\_,\_), <\_,\_>, \cup, \cap, -$ `line`, `polygon`, `region`, $\emptyset$, $\{\_\}$, $\{\{\_\}\}$, `window` and `circle`, the operations on sets of spatial objects, and finally, $\mathbb{R}$. We denote by Σ^T the *signature for (logic) terms*, which consists on $f_1, \ldots, f_m$, $c_1, \ldots, c_k$, $[\_,\_], (\_,\_), <\_,\_>$, `line`, `polygon`, `region` and $\mathbb{R}$. We denote by Σ^C the *signature for constraint terms*, which consists on $[\_,\_], (\_,\_), <\_,\_>, \cup, \cap, -$ `line`, `polygon`,`region`, $\emptyset$, $\{\_\}$, $\{\{\_\}\}$, `window` and `circle`.

From Σ^T and the sets of variables `Var` (for logic values), `NVar` (for real values) `OVar` (for spatial objects) and `PVar` (for pairs of spatial objects), we can build the set of *CLP terms*, denoted by `Terms`. In addition, we have to consider the Herbrand base obtained from Σ^T, denoted by $\mathcal{T}$. Analogously, from Σ^C and the same sets of variables, we can build the set of *CLP constraint terms*, denoted by `Constrs`, and we can consider the Herbrand base obtained from Σ^C, denoted by $\mathcal{C}$, which consists on the sets $\mathcal{FH}$, $\mathcal{PFH}$, $\mathcal{SFH}$ and $\mathcal{SPFH}$.

DEFINITION 3.9. **Program.** *A CLP($\mathcal{SSO}$)* program $\mathcal{P}$ *under a signature Σ consist on rules of the form: $p : -\Gamma | q_1, \ldots, q_n$, where p, q_i are atoms using predicate symbols of Σ and terms of the signature Σ^T; and Γ is a SCSP under the signature Σ^C. In addition, Γ is $\mathcal{V}$-safe where $\mathcal{V} = var(\Gamma) \setminus (var(p) \cup_{1 \leq i \leq n} var(q_i))$. In the particular case of $p : -\Gamma$, we call it a fact.*

DEFINITION 3.10. **Goal.** *A CLP($\mathcal{SSO}$)* goal $\mathcal{G}$ *under a signature Σ is of the form: $: -\Gamma | q_1, \ldots, q_n$, where q_i are atoms using predicate symbols of Σ and terms of the signature Σ^T; and Γ is a SCSP under the signature Σ^C. In addition Γ is $\mathcal{V}$-safe where $\mathcal{V} = var(\Gamma) \setminus \cup_{1 \leq i \leq n} var(q_i)$. In the particular case of a goal $\mathcal{G} \equiv: -$ we write $\square$.*

Let remark that terms in predicates can be neither sets nor figures with holes. It is a syntactic condition which forces to carry out them in the constraint side. In addition, programs and goals are required to accomplish a *safety condition* in order to guarantee each satisfiable safe goal w.r.t. a safe program can be represented in solved form. This safety condition is adapted to CLP programs and goals as follows. The variables of Γ should be ranged restricted and operations occurring in Γ are rigid, but it is only required for extra variables, that is, not occurring on the clause head and body.

Now, we can assume the domain $\mathcal{D}$ of our CLP-program which consists on the sets $\mathcal{T}$ and $\mathcal{C}$. A *$\mathcal{D}$-valuation* $\psi = (\theta, \mu, \Delta, \Omega)$ consist is a replacement of variables of `Var`, `NVar`, `OVar` and `PVar`, by elements of $\mathcal{T}$ where $\mu(\alpha) \in \mathbb{R}$ if $\alpha \in$ `NType`, $\Delta(\alpha) \in \mathcal{FH}$ if $\alpha \in$ `OVar`, $\Omega(\alpha) \in \mathcal{PFH}$, if $\alpha \in$ `PVar`, and $\theta(\alpha) \in \mathcal{T}$ otherwise. When applying a $\mathcal{D}$-valuation ψ to a variable α we will write $\psi(\alpha)$ or $\alpha\psi$.

In addition, we can consider *$\mathcal{D}$-substitutions* which have the form $\psi = (\theta, \mu, \Delta, \Omega)$ and map variables into elements of `Terms` and `Constrs`. Finally, elements of `Terms` can be *unified* by means of $\mathcal{D}$-substitutions. In the case of usual logic terms it corresponds with the usual unification of logic programming. In the case of $\mathbb{R}$, it basically consists on identification of real variables and values. The case of objects and pairs of objects consists on identification of figures. Figures are equals with they are defined with same sets of points, therefore unification consists on unification of real variables and values. The same can be said for intervals. Therefore, we can still consider *most general unifiers* for our term domain.

Now, an *interpretation* $\mathcal{I}$ under a signature Σ, consists on sets of sequences $\mathcal{I}_p = \{v_1 \ldots v_n \mid v_i \in \mathcal{T} \cup \mathcal{C}\}$ for each $p \in \Sigma$. In an interpretation, each $\mathcal{I}_p$ corresponds with a subset of the Herbrand base, that is, the Herbrand base element $p(\bar{v})$ corresponds with the sequence $\bar{v} \in \mathcal{I}_p$. The set of interpretations under a signature Σ forms a *complete lattice* under the ordering defined as follows. $\mathcal{I} \subseteq \mathcal{I}'$ iff $\mathcal{I}_p \subseteq \mathcal{I}'_p$ for each $p \in \Sigma$.

DEFINITION 3.11. **Fixed Point Operator.** *Given a CLP($\mathcal{SSO}$) program $\mathcal{P}$, and an interpretation $\mathcal{I}$ under Σ, we define the fixed point operator $T_{\mathcal{P}}(\mathcal{I})$, for each $p \in \Sigma$, as follows:*

$$T_{\mathcal{P}}(\mathcal{I})_p = \{v_1 \ldots v_n \mid \textit{there exists a variant } p(\bar{t}) : -\Gamma | q_1(\bar{s_1}), \ldots, q_n(\bar{s_n}) \in \mathcal{P}, \textit{ and there exists a } \mathcal{D}-\textit{valuation } \psi = (\theta, \mu, \Delta, \Omega) \textit{ such that } t_i\psi \equiv v_i, \mathcal{I}_{q_i} \supseteq \bar{s_i}\psi, \textit{ and } [\![\Gamma]\!]_{(\mu,\Delta,\Omega)} \textit{ is true}\}$$

$T_{\mathcal{P}}(\mathcal{I})_p$ defines a new interpretation $T_{\mathcal{P}}(\mathcal{I})$. The application of the operator computes the least Herbrand model of a CLP-program.

DEFINITION 3.12. **Herbrand Model** *An interpretation $\mathcal{I}$ under a signature Σ is a model of $\mathcal{P}$ iff $\mathcal{I}_p \supseteq \bar{t}\psi$ for every $\mathcal{D}$-valuation $\psi = (\theta, \mu, \Delta, \Omega)$ such that $\mathcal{I}_{q_i} \supseteq \bar{s_i}\psi$, $[\![\Gamma]\!]_{(\mu,\Delta,\Omega)}$ is true, for every $p(\bar{t}) : -\Gamma | q_1(\bar{s_1}), \ldots, q_n(\bar{s_n}) \in \mathcal{P}$. In this case we write $\mathcal{I} \models \mathcal{P}$.*

An interpretation $\mathcal{I}$ under a signature Σ satisfies a goal $\mathcal{G} \equiv : -\Gamma | q_1(\bar{s_1}), \ldots, q_n(\bar{s_n})$ iff there exists a $\mathcal{D}$-valuation $\psi = (\theta, \mu, \Delta, \Omega)$ such that $\mathcal{I}_{q_i} \supseteq \bar{s_i}\psi$ and $[\![\Gamma]\!]_{(\mu,\Delta,\Omega)}$ is true. In the case, we write $\mathcal{I} \models \mathcal{G}\psi$.

THEOREM 3.2. **Least Herbrand Model.** *The fixed point operator computes the least model of a program $\mathcal{P}$, that is:*

(a) for every Herbrand model $\mathcal{M}$ of $\mathcal{P}$ there exists k such that $T^k_{\mathcal{P}}(\emptyset) \subseteq \mathcal{M}$;

and (b) $T^{\infty}_{\mathcal{P}}(\emptyset)$ is a Herbrand model of $\mathcal{P}$.

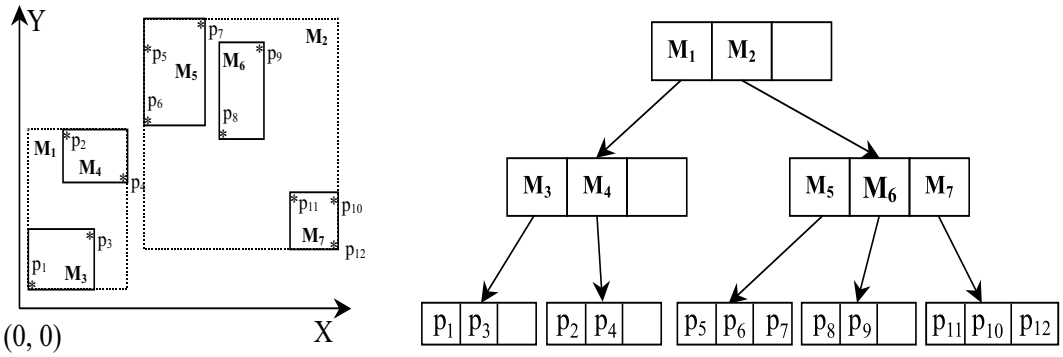
4. Operational Semantics of CLP($\mathcal{SSO}$)

In this section we will describe the operational semantics of our constraint logic language. The operational semantics is based on the combination of term unification and constraint solving. One of the main aspects of the operational semantics in constraint logic programming, and in particular, in our case are the *efficiency of the constraint solving*. A constraint solver for the constraint system presented should solve constraints on sets of spatial objects.

4.1 Minimum Bounded Rectangles and R-trees

With this aim, the constraint solver handles a *constraint store* in which sets of spatial objects are represented by means of *Minimum Bounded Rectangles* (MBR's) to enclose objects and set of objects. In the case of set of objects the MBR's are organized in data structures called *R-trees*. These spatial access methods are a similar structure to the well-known data structures *B-trees* used for file indexing. In the case of R-trees, each object is enclosed in its MBR and stored in the *leaves*. Each *internal node* of the tree stores the set of MBR's enclosing its children and has as *searching key* the MBR covering the children.

Figure 3. MBRs and R-trees



This structure allows optimizations in the form of branch and bound algorithms which takes into account the coordinates of the MBR's (upper-left corner and lower-right corner of the MBR). As an example of this data structure, the figure 3 shows how to store in an R-tree the points $\mathtt{p}_1, \ldots, \mathtt{p}_{12}$ in which the MBR $\mathtt{M}_1$ encloses the points $\mathtt{p}_1, \ldots, \mathtt{p}_4$ and $\mathtt{M}_2$ the rest of points. At the same time $\mathtt{M}_1$ is subdivided into $\mathtt{M}_3$ and $\mathtt{M}_4$, and so on. It gives us the R-tree described in the right chart of the figure. For more details about R-trees see [9].

We adopt the cited structure but adapting MBR's and R-trees to the context of constrained objects in the following sense. Each spatial object is enclosed into an MBR but the MBR is also constrained given that the spatial object can be described by means of constrained coordinates. That is, a spatial object can be $\mathtt{line}[(\mathtt{x}, 8), (10, 12)]$, and $\mathtt{x}$ can be constrained to belong to an interval (note that due to safe condition, each coordinate must be constrained by an equality or a membership to an interval). It forces to consider constrained MBRs which have the upper-left corner and lower-right corner also constrained. For instance in the case $\mathtt{line}[(\mathtt{x}, 8), (10, 12)]$, and supposing $\mathtt{x} \in [7, 22]$ it is enclosed by two MBRs with corners $(\mathtt{x}, 12)$ and $(10, 8)$ if $\mathtt{x} \in [7, 10]$, and $(10, 12)$ and $(\mathtt{x}, 8)$ whenever $\mathtt{x} \in [10, 22]$ following the criteria for building MBRs. Now, the building of the R-tree enclosing both MBRs follows the usual criteria. It is supposed to be achieved in the *constraint store* when the CLP-program is compiled.

4.2 Resolution

Now, we will define the resolution rule for our CLP programs. This rule calls the constraints solver which will be defined by means of transformation rules.

DEFINITION 4.1. **Resolution** *Given a goal* $\mathcal{G} \equiv: -\Gamma | p_1(\bar{s_1}), \ldots, p_n(\bar{s_n})$ *we define an step of resolution* $\mathcal{G} \Rightarrow_\psi \mathcal{G}'$ *as follows.* $\mathcal{G}'$ *consists on*

$$: -\Gamma' | (p_1(\bar{s_1}), \ldots, p_{i-1}(\bar{s_{i-1}}), q_1(\bar{t_1}), \ldots, q_r(\bar{t_r}), p_{i+1}(\bar{s_{i+1}}), \ldots, p_n(\bar{s_n}))\psi$$

whenever there exist a variant $p_i(\bar{v}) : -\Gamma_0 | q_1(\bar{t_1}), \ldots, q_r(\bar{t_r}) \in \mathcal{P}$*, and a* $\mathcal{D}$*-substitution* $\psi = (\theta, \mu, \Delta, \Omega)$ *such that* θ *is the m.g.u. of* $\bar{v}$ *and* $\bar{s_i}$*, and there exists a sequence of transformation steps* $\Gamma \wedge \Gamma_0 \hookrightarrow^*_{(\mu,\Delta,\Omega)} \Pi \bigvee \Gamma'$*; and* $\mathcal{G}'$ *consists on* `fail` *whenever*

- *there not exists* $p_i(\bar{v}) : -\Gamma_0 | q_1(\bar{t_1}), \ldots, q_r(\bar{t_r}) \in \mathcal{P}$ *such that* $\bar{s_i}$ *and* $\bar{v}$ *unify or*
- *there exists* $p_i(\bar{v}) : -\Gamma_0 | q_1(\bar{t_1}), \ldots, q_r(\bar{t_r}) \in \mathcal{P}$ *such that* $\bar{s_i}$ *and* $\bar{v}$ *unify with m.g.u.* ψ *and there exists a sequence of transformation steps* $(\Gamma \wedge \Gamma_0)\psi \hookrightarrow^*_{(\mu,\Delta,\Omega)}$ `fail`

This resolution rule is defined in a general way, that is, here we are not interested in the definition of *any concrete strategy* of combination of constraint solving and resolution. The computations $\Gamma \hookrightarrow_{(\mu,\Delta,\Omega)} \Gamma'$ and $\Gamma \hookrightarrow_{(\mu,\Delta,\Omega)}$ `fail` are calls to the *constraint solver*. These calls consists at the same time on *consistence checking* and *constraint propagation* process. If *consistence checking fails* then the step of resolution fails. Otherwise, the constraint propagation achieves the *adding of new constraints in the constraint store*. Let us remark that each rule application can involve several steps of resolution (i.e. several branches of the search tree) once our constraint solver can generate *disjunctions of constraints*.

DEFINITION 4.2 (Computed Answer). *We say that a* $\mathcal{D}$*-substitution* ψ *is a computed answer from a goal* $\mathcal{G}$ *iff there exists a sequence of steps of resolution* $\mathcal{G} \Rightarrow_{\psi_0} \mathcal{G}_1 \Rightarrow_{\psi_1} \mathcal{G}_2 \cdots \Rightarrow_{\psi_n} \square$ *such that* $\psi = \psi_0 \cdot \psi_1 \cdot \cdots \cdot \psi_n$.

4.3 Constraint Solver

Our constraint solver will be described by means of a set of transformation rules of the form $\Pi \bigvee CH \odot \Gamma \hookrightarrow \Pi \bigvee CH' \odot \Gamma'$. Basically, these transformation rules achieve the *consistence checking* and *constraint propagation* process.

The set CH is a *conjunction of simple constraints*, called *consistence constraints*, which should be checked for *consistency* in each step of resolution. Initially, CH contains the set of constraints on `NType` of the Γ occurring in the goal $\mathcal{G}$. Therefore, initially is supposed to have $CH \odot \Gamma \backslash CH$. Whenever new constraints are introduced in the application of a rule, the transformation rules allow the consistence checking of the *new CH's*, and achieve constraint propagation by generating either *simpler constraints*, or adding *new kind of constraints* or *solving previously delayed constraints*. There are some constraints which have to be delayed due to *non bounded variables*. The transformation process may generate disjunctions of constraints whenever there are several alternatives.

There are some special kind of constraints which we call *constraints on candidate objects* which can be delayed to be solved at the end of predicate solving in order to obtain the solved form. The solving of these constraints on candidate objects, which we call *refinement step*, consists on checking the operations defined in table 5 for pairs of spatial objects. The consistence checking is able to *prune tree branches*, but constraint solving on candidate objects *can still fail*. We can say that consistence checking definitively discards branches or fails by using the MBR structure as an *approximation to the shape* of an spatial object. However, the refinement step has to *complete the solving process* applying the operations over the spatial objects for which there was not found any inconsis-

tence. The efficiency of our constraint solver should be evaluated in the presence of a *massive quantity of constraints*.

Finally, and from a practical point of view, our CLP language could be implemented, and this is our aim, using as host language a CLP system with a *constraint solver over intervals of real numbers*. The reason for that is that the consistence checking process consists on consistence checking over intervals of real numbers. It will be explained later.

4.4 Consistence Constraints

The basis of the optimization of our constraint solver is the consistence checking of a conjunction of simple constraints. The consistence constraints have the following form:

- $\mathtt{n}_1 = \mathtt{n}_2$, $\mathtt{n}_1 \neq \mathtt{n}_2$, $(\mathtt{p}_1, \mathtt{p}_2) = (\mathtt{p}'_1, \mathtt{p}'_2)$, $(\mathtt{p}_1, \mathtt{p}_2) \neq (\mathtt{p}'_1, \mathtt{p}'_2)$, $\mathtt{n}_1 \in [\mathtt{n}_2, \mathtt{n}_3]$, $\mathtt{n}_1 \notin [\mathtt{n}_2, \mathtt{n}_3]$, which are constraints on NType;
- $(\mathtt{p}_1, \mathtt{p}_2) \in \mathtt{R}$ and $(\mathtt{p}_1, \mathtt{p}_2) \notin \mathtt{R}$ where $\mathtt{p}_1, \mathtt{p}_2 \in$ NType, and R is an MBR; which are also equivalent to real interval constraints once $(\mathtt{p}_1, \mathtt{p}_2) \in \mathtt{R}$ is equivalent to $\mathtt{p}_1 \in [\mathtt{R.up.x}, \mathtt{R.low.x}] \wedge \mathtt{p}_2 \in [\mathtt{R.up.y}, \mathtt{R.low.y}]$, and analogously for $(\mathtt{p}_1, \mathtt{p}_2) \notin \mathtt{R}$; where the suffixes up and low denote the upper-left corner and lower-right corner of the MBR R, respectively, and the suffixes x and y denote the coordinates of such corners in the axis X and Y, respectively;
- mbrbound(S, R) where S is from SetSDT, and R is an MBR; and distbound(PS, $\mathtt{n}_1$, $\mathtt{n}_2$) where $\mathtt{n}_1$ and $\mathtt{n}_2$ are from NType and PS is from SetPSDT. The first one requires the elements of the set S to be included in R; and the second one requires the distance for each pair of objects of PS is bounded in $[\mathtt{n}_1, \mathtt{n}_2]$.

The consistence checker for the two last kind of constraints should follows the next rules of *constraint propagation*:

$$(\mathbf{P1}) \quad \Pi \bigvee mbrbound(S, R_1) \wedge mbrbound(S, R_2) \odot \Gamma \hookrightarrow \Pi \bigvee mbrbound(S, R_3) \wedge R_3 = intermbr(R_1, R_2) \odot \Gamma$$

$$(\mathbf{P2}) \quad \Pi \bigvee distbound(PS, n_1, n_2) \wedge distbound(PS, n_3, n_4) \odot \Gamma \hookrightarrow \Pi \bigvee distbound(PS, n_5, n_6) \wedge n_5 = max(n_1, n_3) \wedge n_6 = min(n_2, n_4) \odot \Gamma$$

where intermbr($\mathtt{R}_1, \mathtt{R}_2$) denotes the intersection of two MBRs which is trivially an MBR, and max($\mathtt{n}_1, \mathtt{n}_2$) (resp. min($\mathtt{n}_3, \mathtt{n}_4$)) denotes the maximum (respectively the minimum) of two real numbers.

In addition, the constraint solver should be able to propagate membership constraints to these special constraints in order to ensure consistence, and it should follow the next rules:

$$(\mathbf{P3}) \quad \Pi \bigvee mbrbound(S, R) \odot O \in S \wedge \Gamma \hookrightarrow \Pi \bigvee mbrbound(S, R) \wedge O \in R \odot O \in S \wedge \Gamma$$

$$(\mathbf{P4}) \quad \Pi \bigvee distbound(PS, n_1, n_2) \odot < O_1, O_2 > \in PS \wedge \Gamma \hookrightarrow \Pi \bigvee distbound(PS, n_1, n_2) \wedge mindmbr(O_1.mbr, O_2.mbr) = n_3 \wedge n_3 \in [n_1, n_2] \wedge maxdmbr(O_1.mbr, O_2.mbr) = n_4 \wedge n_4 \in [n_1, n_2] \odot < O_1, O_2 > \in PS \wedge \Gamma$$

where mindmbr($\mathtt{R}_1, \mathtt{R}_2$) (resp. maxdmbr($\mathtt{R}_1, \mathtt{R}_2$)) denotes the minimum (resp. maximum) distance of two MBR's, and the suffix O.mbr denotes the MBR enclosing the object O.

In summary, the consistence checker of these simple constraints should always check interval constraints.

The *failure rule* for the consistence checker is as follows:

$$(\mathbf{FAILURE}) \quad \Pi \bigvee CH \odot \Gamma \hookrightarrow \Pi \quad if\ \Pi\ is\ not\ empty\ and\ CH\ is\ inconsistent$$

$$\Pi \bigvee CH \odot \Gamma \hookrightarrow \mathtt{fail} \quad if\ \Pi\ is\ empty\ and\ CH\ is\ inconsistent$$

The first case corresponds with the prune of the search tree, and the second with a failure branch.

4.5 Transformation Rules

In this subsection we will review the transformation rules, showing the main cases. First of all, we will summarize the notation used in the rules:

Notation

1. For objects, we use the suffixes mbr, up and low with the meaning of the previous section, and using the suffix obj to refer to the object itself.
2. For sets of objects, the suffix root denotes the root of the R-tree storing the set of objects, and we use induces $\mathtt{i}_1, \ldots, \mathtt{i}_n$ for denoting the child of index $\mathtt{i}_j$ in a internal node of an R-tree.
3. For set of pairs of objects, the suffixes first and second refer to the set of objects in the first (resp. second) component of each pair.
4. For MBRs, we use functions unionmbr, intermbr and diffmbr for computing the same operations on MBRs.
5. We introduce *a new kind of constraints* for each operation obtaining a *set of pairs of objects*, of the form: mindtree(m, PS, $\mathtt{R}_1$, $\mathtt{R}_2$), maxdtree(m, PS, $\mathtt{R}_1$, $\mathtt{R}_2$), insidetree(PS, $\mathtt{R}_1$, $\mathtt{R}_2$), ..., etc, where PS is from SetPSDT, and $\mathtt{R}_1$ and $\mathtt{R}_2$ are MBRs. The meaning of the new kind of constraints is a bound (in the form of MBRs) for the search space for each pair of PS.
6. Finally, we introduce *a new kind of constraints* for each operation obtaining a *set of objects*, of the form: rangetree(S, R, Win) and rangetree(S, R, Circle), and so on, where R is an MBR, S is from SetSDT. The meaning of such constraints is a bound in the form of an MBR for the search space of S.

The transformation rules are shown in tables 6,7 and 8. We have included the main cases of the transformation rules, the full version can be found in [1]. With respect to the transformation rules of **equality constraints**, they use the suffixes up and low to refer to the corners of the MBRs enclosing the object. As an example, the rule **(E1)** introduces *consistence constraints* for comparing the two MBRs of the compared objects. In addition, it uses the suffix obj to refer to the object itself (in this case the compared objects are trivially candidates).

With respect to the **membership constraints**, basically they introduce consistence constraints for handling the MBRs (resp. R-trees) enclosing an object (resp. a set of objects) (rule **(M1)** (resp. **(M2)** to **(M4)**)). The most interesting rules of this block are the rules from **(M2)** to **(M4)**. They handle the R-tree enclosing a set of objects. For instance, the rule **(M2)** starts the search in the tree root, and the rule **(M3)** discards the children which do not contain the MBR enclosing the searched object. It should be checked by means of the **(FAILURE)** rule. Finally, rule **(M4)** adds constraints for the candidate objects. As an example, we can consider w.r.t the figure 3:

$$\emptyset \odot p_5 \in \{p_1, \ldots, p_{12}\} \hookrightarrow \emptyset \odot p_5 \in \{p_1, \ldots, p_{12}\}.root \hookrightarrow p_5 \in \{p_1, \ldots, p_{12}\}.root.up \wedge p_5 \in \{p_1, \ldots, p_{12}\}.root.low \odot p_5 \in \{p_1, \ldots, p_{12}\}.root.M_1 \bigvee p_5 \in \{p_1, \ldots, p_{12}\}.root.up \wedge p_5 \in \{p_1, \ldots, p_{12}\}.root.low \odot p_5 \in \{p_1, \ldots, p_{12}\}.root.M_2 \hookrightarrow \ldots \emptyset \odot p_5 \in \{p_1, \ldots, p_{12}\}.root.M_2.M_5 \hookrightarrow \ldots$$

With respect to the transformation rules for **set constraints**, they use the cited operations for MBRs (as an example see rules **(SE1)** and **(SE2)**).

With respect to **metric constraints**, they add consistence constraints of the form $\mathtt{m} \in [\mathtt{a}, \mathtt{b}]$ and distbound(PS, a, b), for each minimum and maximum distance to be computed. m is a variable used for computing the minimum (resp. maximum) distance of two objects (or set of objects) and PS is a variable for storing pairs of objects at the minimum (resp. maximum) distance. In the refine-

Table 6. Equality and Membership Transformation Rules

(**E1**)	$\Pi \bigvee CH \odot (O_1 = O_2 \wedge \Gamma) \hookrightarrow \quad \Pi \bigvee (O_1.up = O_2.up \wedge O_1.low = O_2.low \wedge CH) \odot (O_1.obj = O_2.obj \wedge \Gamma)$
(**M1**)	$\Pi \bigvee CH \odot ((p_1, p_2) \in O \wedge \Gamma) \hookrightarrow \quad \Pi \bigvee ((p_1, p_2) \in O.mbr \wedge CH) \odot ((p_1, p_2) \in O.obj \wedge \Gamma)$
(**M2**)	$\Pi \bigvee CH \odot (O \in S \wedge \Gamma) \hookrightarrow \quad \Pi \bigvee CH \odot (O \in S.root \wedge \Gamma)$
(**M3**)	$\Pi \bigvee CH \odot (O \in R \wedge \Gamma) \hookrightarrow \quad \Pi \bigvee_{j=s_1,\ldots,s_k} ((O.up \in R \wedge O.low \in R \wedge CH) \odot (O \in R.j \wedge \Gamma))$ $if\ R\ has\ subtrees\ s_1, \ldots, s_k$
(**M4**)	$\Pi \bigvee CH \odot (O \in R \wedge \Gamma) \hookrightarrow \quad \Pi \bigvee (O.up \in R \wedge O.low \in R \wedge CH) \odot (O.obj = R.obj \wedge \Gamma)$ $if\ R\ is\ a\ leaf$

Table 7. Set and Metric Transformation Rules

(**SE1**)	$\Pi \bigvee CH \odot \Gamma[O_1 \cup O_2] \hookrightarrow \Pi \bigvee mbrbound(S, unionmbr(O_1.mbr, O_2.mbr)) \wedge CH \odot \Gamma[S] \ \wedge S = O_1.obj \cup O_2.obj$
(**SE2**)	$\Pi \bigvee CH \odot \Gamma[S_1 \cup S_2] \hookrightarrow \Pi \bigvee mbrbound(S, unionmbr(S_1.root, S_2.root)) \wedge CH \odot \Gamma[S] \ \wedge S = S_1.root \cup S_2.root$
(**ME1**)	$\Pi \bigvee CH \odot \Gamma[mindist(O_1, O_2)] \hookrightarrow$ $\Pi \bigvee m \in [mindmbr(O_1.mbr, O_2.mbr), maxdmbr(O_1.mbr, O_2.mbr)] \wedge CH \odot \Gamma[m] \wedge m = mindist(O_1.obj, O_2.obj)$
(**ME2**)	$\Pi \bigvee CH \odot \Gamma[pmindist(O_1, O_2)] \hookrightarrow$ $\Pi \bigvee distbound(PS, mindmbr(O_1.mbr, O_2.mbr), maxdmbr(O_1.mbr, O_2.mbr)) \wedge CH \odot \Gamma[PS] \wedge PS = pmindist(O_1.obj, O_2.obj)$
(**ME3**)	$\Pi \bigvee CH \odot \Gamma[mindist(S_1, S_2)] \hookrightarrow$ $\Pi \bigvee m \in [mindmbr(S_1.root, S_2.root), maxdmbr(S_1.root, S_2.root)] \wedge$ $CH \odot (\Gamma[m] \wedge mindtree(m, PS, S_1.root, S_2.root))$
(**ME4**)	$\Pi \bigvee m \in [a, b] \wedge CH \odot mindtree(m, PS, R_1, R_2) \wedge \Gamma \hookrightarrow \Pi \bigvee m \in [mindmbr(R_1, R_2), b]$ $\wedge CH \odot \wedge_{j=i_1,\ldots,i_n,k=l_1,\ldots,l_t} mindtree(m, PS, R_1.j, R_2.k) \wedge \Gamma$ $if\ R_1\ has\ subtrees\ i_j\ and\ R_2\ has\ subtrees\ l_k\ and\ a >= mindmbr(R_1, R_2)$
(**ME5**)	$\Pi \bigvee m \in [a, b] \wedge CH \odot mindtree(m, PS, R_1, R_2) \wedge \Gamma \hookrightarrow \Pi \bigvee m \in [mindmbr(R_1, R_2), b]$ $\wedge CH \odot m = mindist(PS) \wedge PS = \{\{< R_1.obj, R_2.obj >\}\} \cup PS_1 \wedge \Gamma[PS/PS_1]$ $if\ R_1\ and\ R_2\ are\ leaves\ and\ a >= mindmbr(R_1, R_2)\ and\ PS\ occurs\ in\ \Gamma$
(**ME6**)	$\Pi \bigvee m \in [a, b] \wedge CH \odot mindtree(m, PS, R_1, R_2) \wedge \Gamma \hookrightarrow \Pi \bigvee m \in [a, b] \wedge CH \odot \Gamma$ $if\ a < mindmbr(R_1, R_2)$

Table 8. Topological and Structural Transformation Rules

(**T1**)	$\Pi \bigvee CH \odot \Gamma[O_1\ inside\ O_2] \hookrightarrow$ $\Pi \bigvee O_1.up \in O_2.mbr \wedge O_1.low \in O_2.mbr \wedge CH \odot \Gamma[O_1.obj\ inside\ O_2.obj]$
(**T2**)	$\Pi \bigvee CH \odot \Gamma[S_1\ inside\ S_2] \hookrightarrow \Pi \bigvee (mbrbound(PS.first, S_1.root) \wedge$ $mbrbound(PS.second, S_2.root) \wedge CH) \odot (\Gamma[PS] \wedge insidetree(PS, S_1.root, S_2.root))$
(**T3**)	$\Pi \bigvee mbrbound(PS.first, R_1) \wedge mbrbound(PS.second, R_2) \wedge CH \odot insidetree(PS, R_1, R_2)$ $\wedge \Gamma \hookrightarrow \Pi \bigvee \wedge_{j=i_1,\ldots,i_n,k=l_1,\ldots,l_t} mrbbound(PS.first, R_1.j) \wedge mbrbound(PS.second, R_2.k)$ $\wedge CH \odot \wedge_{j=i_1,\ldots,i_n,k=l_1,\ldots,l_t} insidetree(PS, R_1.j, R_2.k) \wedge \Gamma)$ $if\ R_1\ has\ subtrees\ i_1, \ldots, i_n\ and\ R_2\ has\ subtrees\ l_1, \ldots, l_t, and\ R_1, R_2\ intersect$
(**T4**)	$\Pi \bigvee CH \odot insidetree(PS, R_1, R_2) \wedge \Gamma \hookrightarrow$ $\Pi \bigvee CH \odot PS = R_1.obj\ inside\ R_2.obj \cup PS_1 \wedge \Gamma[PS/PS_1]$ $if\ R_1\ and\ R_2\ are\ leaves\ and\ R_1, R_2\ intersect$
(**T5**)	$\Pi \bigvee CH \odot insidetree(PS, R_1, R_2) \wedge \Gamma \hookrightarrow \Pi \bigvee CH \odot \Gamma$ $if\ R_1, R_2\ do\ not\ intersect\ and\ PS\ occurs\ in\ \Gamma$
(**S1**)	$\Pi \bigvee CH \odot \Gamma[area(O)] \hookrightarrow$ $\Pi \bigvee m \in [0, (O.up.x - O.low.x) * (O.up.y - O.low.y)] \wedge CH \odot \Gamma[m] \wedge m = area(O.obj)$

ment step the bounds `a` and `b` should be updated for candidate objects. These constraints represent the *lower* and *upper bounds* of the distance of two objects and sets of pairs of points or objects, respectively. In such a way, they are used for the pruning of the search for the minimum distance (and pairs of points or objects at the minimum distance). Similarly for maximum distances. In addition, the use of the new kind of constraints `mindtree`, `pmindtree`, etc, allows the handling of each R-tree, and enables the decomposition of an MBR into its children. For instance, rules **(ME1)** and **(ME2)** compute these bounds for a couple of objects, and the rule **(ME3)** starts the search in the tree root, and the rules from **(ME4)** to **(ME6)** update the lower bounds, for the case of minimum distance of two sets of objects. Finally, **(ME5)** obtains the candidate objects. The rules for pairs of points and objects at the minimum distance are similar. The case of maximum distance is also similar, updating the upper bound.

With respect to the **topological constraints**, we will show the case of `inside`, and the rest of cases are similar. The technique for solving such constraints is based on the use of the consistence constraint $\texttt{mbrbound}(\texttt{S}, \texttt{R})$, which keeps the bound in the form of an MBR for the elements of `S`. In this case, a new constraint `insidetree` allows the handling of the R-tree (rules **(T1)** to **(T5)**). Finally, the **structural constraints** take into account the *lower* and *upper bounds* of area, length and perimeter operations of an object and a set of objects, using also MBRs and R-trees (as an example see rule **(S1)**).

4.6 Delayed Constraints

As was commented some constraints have to be delayed up to the variables are bounded. Variables can be bounded when a new rule is applied. Basically, the operations on (sets of) (pairs) of spatial objects defined in the paragraph **Notation** in the items (1) to (5), such as `mbr`, `up`, `low`, and so on need to be refereed to an spatial object, however whenever they are applied to a variable, the constraint that uses them should be delayed up to the variable is bound to the spatial object. And the same can be said for sets of spatial objects and sets of pairs of spatial objects.

4.7 Refinement Step

The refinement step consists of the solving of the constraints over candidate objects of Γ. For solving these constraints now we should take into account the candidate objects (i.e. figures with holes) stored in each MBR computed with the transformation rules, and apply the operations on table 5. This refinement step is supposed to be applied in any step of resolution, and it allows to obtain partially solved forms, in particular, $\mathcal{D}$-valuations representing solutions of the solved forms. In practice, these solutions may not be computed and the answer can be shown as a solved form. The refinement step consists on a basic rule:

$$(\mathbf{REFINEMENT})\ CH \odot \Gamma \wedge \mathbf{Op}(O_1.obj, \ldots, O_n.obj) \hookrightarrow_{(\mu,\Delta,\Omega)} (CH \odot \Gamma)\Omega\Delta\mu \quad if\ [\![\mathbf{Op}(O_1.obj, \ldots, O_n.obj)]\!]_{(\mu,\Delta,\Omega)}\ is\ true$$

Finally, we conclude with a result of soundness and completeness of our operational semantics.

THEOREM 4.1. **Soundness and Completeness.** *For every goal $\mathcal{G}$ we have that $T_{\mathcal{P}}^{\infty}(\emptyset) \models G\psi'$ for a $\mathcal{D}$-valuation ψ' iff there exists a computed answer ψ such $\psi' = \psi \cdot \lambda$.*

5. Conclusions and Future Work

In this paper we have presented an instance of CLP for sets of spatial objects. We have defined its fixed point and operational semantics. The operational semantics is based on a constraint solver for which a consistence checking and constraint propagation techniques have been presented. As future work we plan to implement this instance in a CLP system like CIAO o SICSTUS. In the presented form there is many work to do in order to make an efficient implementation of our constraint solving process. We hope the prototype would allow to introduce and test improvements.

Acknowledgments

This work has been partially supported by the Spanish project of the Ministry of Science and Technology, "INDALOG" TIC2002-03968 under FEDER funds.

References

[1] J. M. Almendros-Jimenez and A. Corral. Solving constraints on sets of spatial objects, available in `http://www.ual.es/~jalmen/padl05tr.ps`. Technical report, Dpto. de Lenguajes y Computaci´on, Universidad de Almer´ıa, 2004.

[2] J. M. Almendros-Jimenez and A. Corral. Solving constraints on sets of spatial objects. In *Proc. of Practical Aspects of Declarative Languages*, volume 3350, pages 158–173. LNCS Springer, 2005.

[3] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[4] A. Belussi, E. Bertino, and B. Catania. Manipulating Spatial Data in Constraint Databases. In *Proceeding of SSD'97 Conference*, LNCS 1262, pages 115–141. Springer, 1997.

[5] A. Belussi, E. Bertino, and B. Catania. An Extended Algebra for Constraint Databases. *IEEE Transactions on Knowledge and Data Engineering, TKDE*, 10(5):686–705, 1998.

[6] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. Lopez-Garcia, and G. Puebla. The ciao system. reference manual (v1.10). num. clip3/97.1.10(04). Technical report, School of Computer Science, Technical University of Madrid (UPM), 2004.

[7] A. M. Cheadle, W. Harvey, A. J. Sadler, J. Schimpf, K. Shen, and M. G. Wallace. Eclipse: An introduction. Technical report, IC-Parc, Imperial College London, Technical Report IC-Parc-03-1, 2003.

[8] P. Codognet and D. Diaz. A Simple and Effi cient Boolean Solver for Constraint Logic Programming. *Journal of Automated Reasoning*, 17(1):97–129, 1996.

[9] A. Guttman. Rtrees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD Conference*, pages 47–57, 1984.

[10] J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming, JLP*, 19,20:503–582, 1994.

[11] J. Jaffar, M. J. Maher, P. J. Stuckey, and R. H. C. Yap. Beyond Finite Domains. In *Proceedings of Principles and Practice on Constraint Programming*, pages 86–94, 1994.

[12] G. M. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases*. Springer, 2000.

[13] K. Marriot and P. J. Stuckey. *Programming with Constraints: an Introduction*. MIT Press, 1998.

[14] P. Z. Revesz. Safe Query Languages for Constraint Databases. *ACM TODS*, 23(1):58–99, 1998.

[15] P. Rigaux, M. Scholl, L. Segoufi n, and S. Grumbach. Building a Constraint-based Spatial Database System: Model, Languages, and Implementation. *Information Systems*, 28(6):563–595, 2003.

[16] E. Schwalb and L. Vila. Temporal Constraints: A Survey. *Constraints*, 3(2/3):129–149, 1998.