AN IMPLEMENTATION OF SPATIO-TEMPORAL ANNOTATED CONSTRAINT LOGIC PROGRAMMING (STACLP)

Claudia ZEPEDA, David SOL
{sc098382, sol}@mail.udlap.mx
Universidad de las Américas-Puebla
San Andrés Cholula, Puebla, México, MEXICO

ABSTRACT

In this paper we present important aspects throughout the implementation of Spatio-Temporal Annotated Constraint Logic Programming (STACLP), in which temporal and spatial data are represented by means of annotations that label atomic first order formulae. The reason to perform the implementation is due to that there is not software available for STACLP. At the same time, we explore the implementation of STACLP to obtain experience and to learn from it. This implementation uses XSB system to verify it as a good alternative of knowledge representation.

KEY WORDS

Constraint Logic Programming, Spatio-Temporal Reasoning, Knowledge Representation.

1. INTRODUCTION

One of the most promising directions of the current research in GIS field focuses on the development of reasoning formalisms that merge contributions from both Artificial Intelligence (AI) and mathematical research areas in order to express spatial qualitative reasoning. In recent literature [1] [2] [3] [4], there has been a great interest in studying spatial concepts from a cognitive point of view, giving rise to Qualitative Spatial Reasoning as a new research area between AI and GIS. Qualitative Spatial Representation addresses different aspects of space including topology, orientation, shape, size, and distance. Qualitative Spatial Reasoning also has been recognized as a major point in the future developments of GIS [1] [2] [3] [4]. It has been claimed that today GIS technology is capable of efficiently storing terabytes of data, but the key point is to abstract away from the huge amount of numerical data and to define formalisms that allow the user to specify qualitative queries. In [1] [2] [3] [4] is showed how the approach of expressing Spatio-Temporal Reasoning on geographical data is useful and they introduce STACLP Spatio-Temporal Annotated Constraint Logic Programming (STACLP) where temporal and spatial data are represented by means of annotations that label atomic first order formulae. For example, if we assume that a transport is described by its name, the kind of things it transports and its spatial position(s) in certain time interval, using STACLP we can model information like *Bus1 transports people, it is parked from 10am to 12am, and it is working from 12am to 5pm in some places of Calpan town but it is loading people between 1pm to 2pm.*

In this paper are presented some important aspects throughout the implementation, using XSB system, of STACLP. The reason to perform the implementation is because up to this time there is not a software available for STACLP and using XSB looks to be a very interesting and useful way to explore STACLP and to learn from it. And at the same time, we can verify XSB system as an alternative of knowledge representation.

The paper is organized as follows. In Section 2, it is presented STACLP which combines spatial and temporal annotations and its semantics, and in Section 3, it is presented some features of XSB considered as necessary for the implementation of the meta-interpreter of STACLP. In Section 4 are showed some important aspects throughout the implementation, using XSB, of the meta-interpreter for STACLP and in this same section some examples of queries involving spatial and temporal knowledge are formulated in our implemented meta-interpreter. Finally, in Section 5, the conclusions are discussed and possible future work.

2. SPATIO-TEMPORAL ANNOTATED CONSTRAINT LOGIC PROGRAMMING

In [4] was introduced an extension to Temporal Annotated Constraint Logic Programming (TACLP) where both temporal and spatial information can be dealt with and reasoned about. The resulting framework was called Spatio-Temporal Annotated Constraint Logic Programming (STACLP), where temporal and spatial data are represented by means of annotations that label atomic first order formulae.

Time can be discrete or dense. Time points are totally ordered by the relation ≤. We denote by τ the set of time points and we suppose to have a set of operations to manage such points. A *time period* is an interval *[r,s]* with *r, s* ∈ τ and 0 ≤ r ≤ s ≤ ∞ which represents the convex and non-empty set of time points $\{ t \mid r \leq t \leq s \}$. Analogously, space can be discrete or dense and we consider as spatial regions *rectangles* represented as *[(x1,x2), (y1,y2)]* which are intended to model the region $\{(x,y) \mid x1 \leq x \leq x2\}$, $\{y1 \leq y \leq y2\}$ [4].

In [1] is defined an *annotated formula* as a formula with the form $A\alpha$, where A is an atomic formula and α is an annotation. Also are defined three kinds of temporal and spatial annotations:

at *T* and **atp** *(X,Y)* are used to express that a formula holds in a time point or in a spatial point.
th *I*, **thr** *R* are used to express that a formula holds *throughout*, i.e. at *every* point, in the temporal interval or the spatial region, respectively.
in *I*, **inr** *R* are used to express that a formula holds at *some* point(s), in the temporal interval or the spatial region, respectively.

On the other hand, in [1] is showed that the set of annotations is endowed with a partial order relation $\sqsubseteq$ which turns it into a lattice. Given two annotations α and β, the intuition is that α ⊑ β if α is "less informative" than β in the sense that for all formulae A, $A\beta \Rightarrow A\alpha$. This partial order is used in the definition of new inference rules. In addition to Modus Ponens, STACLP has the two inference rules below:

$$\frac{A\alpha \qquad \gamma \sqsubseteq \alpha}{A\gamma} \; rule(\sqsubseteq)$$

$$\frac{A\alpha \qquad A\beta \qquad \gamma = \alpha \sqcup \beta}{A\gamma} \; rule(\sqcup)$$

The rule $(\sqsubseteq)$ states that if a formula holds with some annotation, then it also holds with all annotations that are smaller according to the lattice ordering. The rule $(\sqcup)$ says that if a formula holds with some annotation and the same formula holds with another annotation β then it holds with the least upper bound α ⊔ β of the two annotations. We can find the *constraint theory for temporal and spatial annotations* in [1].

The class of annotations which combines spatial and temporal annotations was introduced in [4].

Definition (Spatio-Temporal Annotations). The class of Spatio-Temporal annotations is the pairing of the spatial annotations **Spat** built from **atp, thr** and **inr** and of the temporal annotations **Temp**, built from **at, th** and **in**, i.e. **Spat*Temp**.

For technical reasons related to the properties of annotations, in [4] is restricted the rule ⊔ only to least upper bounds that produce regions which are rectangles and the temporal components are time periods. Thus there are six cases considered. Here, only present the first case where last upper bound is considered and the other five cases can be found in [4].

$$\begin{array}{l} \mathtt{thr}[(x_1,x_2),(y_1,y_2)]\mathtt{th}[t_1,t_2] \sqcup \\ \mathtt{thr}[(x_1,x_2),(x_1,x_2)]\mathtt{th}[t_1,t_2] = \\ \mathtt{thr}[(x_1,x_2),(y_1,x_2)]\mathtt{th}[t_1,t_2] \Leftrightarrow \\ \quad y_1 \leq x_1,\ x_1 \leq y_2,\ y_2 \leq x_2 \end{array}$$

This axiom allows one to enlarge the region in which a property holds in a certain interval. If a property A holds both throughout a region *R1* and throughout a region *R2* in every point of the time period I then it holds throughout the region which is the union of *R1* and *R2*, throughout I. Notice that the constraints on the spatial variables ensure that the resulting region is still a rectangle.

The clausal fragment of STACLP, which can be used as an efficient Spatio-Temporal Programming Language, consists of clauses of the following form [1]:

$$A\alpha\beta \leftarrow C_1, \ldots, C_n, B_1\alpha_1\beta_1, \ldots, B_m\alpha_m\beta_m$$

where n, m ≥ 0, A is an atom (not a constraint), α, αi, β, βi are (optional) temporal and spatial annotations, the *C*j's are constraints and the *B*i's are atomic formulae. Constraints *Cj* cannot be annotated. A STACLP *program* is a finite set of STACLP clauses.

Semantics of STACLP
The definition of the semantics for STACLP is as follows [1]:

- It is assumed that all atoms are annotated with **th, in, thr** or **inr** labels. In fact, **at t** and **atp(x,y)** annotations can be replaced with **th[t,t]** and **thr [(x,x),(y,y)]** respectively by exploiting the **(at th)** and **(atp thr)** axioms.
- Each atom in the object level program which is not two-annotated, i.e., which is labeled by at most one kind of annotation, is intended to be true throughout the whole lacking dimension(s). For instance an atom **A thr R** is transformed into the two-annotated atom **A thr R th [0,∞].**
- Constraints remain unchanged.
- The meta-interpreter for STACLP is defined by the following clauses:

$$
\begin{array}{l}
demo(empty) \\
demo((B_1, B_2)) \leftarrow demo(B_1), demo(B_2) \\
demo(A\alpha\beta) \leftarrow \alpha \sqsubseteq \delta, \beta \sqsubseteq \gamma, \\
\qquad\qquad clause(A\,\delta\gamma, B), demo(B) \\
demo(A\alpha'\beta') \leftarrow \alpha_1\beta_1 \sqcup \alpha_2\beta_2 = \alpha\beta, \\
\qquad \alpha' \sqsubseteq \alpha, \beta' \sqsubseteq \beta,\ clause(A\,\alpha_1\beta_1, B), \\
\qquad\qquad demo(B), demo(A\alpha_2\beta_2) \\
demo((C)) \leftarrow constraint(C), C
\end{array}
$$

- A clause Aαβ←B of a STACLP program is represented at the meta-level by

$$clause(A\alpha\beta, B) \leftarrow valid(\alpha), valid(\beta)$$

where *valid* is a predicate that checks whether the interval or the region in the annotation is not empty.

The resolution rule, third clause, implements both the Modus Ponens rule and the rule $(\sqsubseteq)$. It contains two relational constraints on annotations, which are processed by the constraint solver using the constraint theory for temporal and spatial annotations mentioned in Section 2.2. Fourth clause implements the rule $(\sqcup)$ combined with Modus Ponens and rule $(\sqsubseteq)$. The constraint $\alpha_1\beta_1 \sqcup \alpha_2\beta_2 = \alpha\beta$ in such a clause is solved by means of the axioms defining the least upper bound mentioned in Section 2.3. Fifth clause manages constraints by passing them directly to the constraint solver.

3. SYSTEM XSB

We used XSB system to implement the meta-interpreter for STACLP defined in Section 2.5. In this Section we present some features of XSB considered as necessary for the implementation of the meta-interpreter.

XSB is a Logic Programming and Deductive Database system for Unix and Windows. In addition to providing all the functionality of Prolog, XSB contains several features not usually found in Logic Programming systems, including [6]:

- Constraint handling for tabled programs based on an engine-level implementation of annotated variables and a package, **clpqr**, for handling real constraints.
- A number of interfaces to other software systems, such as C, Java, Perl, ODBC, SModels, and Oracle.
- Preprocessors and Interpreters so that XSB can be used to evaluate programs that are based on advanced formalisms, such as extended logic programs (according to the Well-Founded Semantics [9]); Generalized Annotated Programs [7]; and F-Logic.

Prolog is based on a depth-first search through trees that are built using program clause resolution (SLD) [5]. As such, Prolog is susceptible to getting lost in an infinite branch of a search tree, where it may loop infinitely. SLG evaluation, available in XSB, can correctly evaluate many such logic programs if they are compiled as a **tabled** predicate. The user can declare that SLG resolution is to be used for a predicate by using table declarations. Alternately, an **auto_table** compiler directive can be used to direct the system to invoke a simple static analysis to decide what predicates to table.

4. IMPLEMENTATION OF THE META-INTERPRETER

In this Section we present some important aspects throughout the implementation using XSB of the meta-interpreter for STACLP, defined in Section 2.5, and some examples of queries involving spatial and temporal knowledge.

Implementation using XSB

Below we show the main code of the meta-interpreter for STACLP, the total number of code lines is about 300.

```
:- auto_table.
 demo(true).
demo(((B1,Alfa1,Beta1),(B2,Alfa2,Beta2))) :-
        demo((B1,Alfa1,Beta1)),demo((B2,Alfa2,Beta2)).
 demo((A,Alfa,Beta)) :-
          contR(Alfa,Delta),contT(Beta,Gama),
          clause1((A,Delta,Gama),B),demo(B).
 demo((A,Alfap,Betap)) :-
          une(Alfa1,Beta1,Alfa2,Beta2,Alfa,Beta),
          contR(Alfap,Alfa),contT(Betap,Beta),
          clause1((A,Alfa1,Beta1),B),demo(B),
          demo((A,Alfa2,Beta2)).

clause1((A,Alfa,Beta),B) :-
        claus((A,Alfa,Beta),B),
         validaR(Alfa), validaT(Beta).
```

The clauses *contR* and *contT* correspond to the rule $(\sqsubseteq)$ to Space and Time respectively mentioned in Section 2.2.

Clause *une* implements the rule $(\sqcup)$ of Section 2.3. In Section 2.5 we described that **at** *t* and **atp**(*x,y*) annotations can be replaced with **th** *[t,t]* and **thr** [(*x,x*),(*y,y*)] respectively by exploiting the (**at th**) and (**atp thr**) axioms. However, it is necessary highlight that a consequence of this replacement is that (**at th**), (**atp thr**), (**at in**) and (**atp inr**) axioms can be rewritten as the two next axioms

(**th in**) **th** [*t,t*] = **in** [**t,t**]
(**thr inr**) **thr**[(*x,x*),(y,y)] = **in**[(x,x),(y,y)]

because are the *link* between (**th**) annotation and (**in**) annotation or between (**thr**) annotation and (**inr**)

respectively. So, these two axioms were implemented instead of the other four axioms.

On the other hand, we can see as the first clause **auto_table**, because without it there would be some cases where XSB may loop infinitely.

Examples

In this section we present some examples which illustrate how spatial data are modeled by annotations and integrated with temporal information using the implemented meta-interpreter.

Volcano Zone

This example describes spatial and temporal information in the context of Popocatépetl volcano. Decision making is a very important activity in this context. About 200,000 people distributed in 50 towns are in danger when the volcano starts its activity. "Plan Operativo Popocatépetl" office in Puebla has the responsibility of coordinating the actions to keep the integrity of the people. This office uses printed maps and printed reports to decide the best sequence of actions in case of danger. Usually it is difficult to justify the decisions because they do not have enough information. Our example describes the activity of buses. We assume that a transport is described by its name, the kind of things it transports and its spatial position(s) in certain time interval. For instance, Bus1 transports people and it is parked from 10am to 12am, and it is working from 12am to 5pm in some places of Calpan town but it is loading people between 1pm to 2pm. This can be expressed by means of the following clauses in our implemented meta-interpreter.

claus(transport(bus1,people),true).
claus((does(bus1,parking), inr(3,3,4,4),th(10,12),true).
claus((does(bus1,working), inr(1,2,2,3),in(12,17)),true).
claus((does(bus1,loading),thr(1,1,2,2),th(13,14)),true).

claus(transport(bus2,people),true).
claus((does(bus2,working), inr(3,4,4,7),th(10,12)),true).
claus((does(bus2,loading), thr(1,1,2,2),th(14,15)),true).

Furthermore, a *town* can be described by its name and its area represented by **thr** annotation. The temporal information for a town is represented by **th**(0,24) annotation because, as we know, a town is all the time in the same spatial position.

claus((town(huejotzingo), thr(3,4,4,7),th(0,24)),true).
claus((town(calpan), thr(1,2,2,3),th(0,24)),true).

Now we show how some queries, involving the spatial and temporal knowledge, can be formulated:

- Which buses did load at Calpan between 12am and 3pm?

demo(((does(X,loading),inr(R),in(12,15))town(calpan), thr(R), th(_))))

X = bus1
X = bus2

The answer to this query consists of all the buses that were loading at Calpan during that time period. The region of Calpan is assigned to the variable **R** and then is solved which buses were loading in some place of that region. We use **in** annotation because we want to know all the different positions of every bus between 12am and 3pm while the **inr** annotation allows one to know the region every bus is in during that time period, even if its exact position is unknown.
If we asked for

demo(((does(X,loading), atp(R), th(12,15)),
(town(calpan), thr(R), th(_)))).

then we would have constrained buses to stay in only one place, in this example Calpan town, for the whole time period.
The query

demo(((does(X,loading),atp(R), in(12,15)),
(town(calpan), thr(R), th(_)))).

asks buses which are in definite positions sometime in (12, 15).

- Which buses were working or parked in Huejotzingo before 12am?

demo(((does(X,working), inr(R), in(0,12)),
(does(X,parking),inr(R),in(0,12)),
(town(huejotzingo),thr(R), th(_)))).
X = bus1
X = bus2

The result are all the buses that were working or parked in the region, assigned to variable R, that corresponds to Huejotzingo town, before 12am.

In Section 4.1, we saw that in the implementation of the meta-interpreter the first clause is *auto_table*, and it was necessary because without it there would be some cases where XSB may loop infinitely. An example of this situation occurs if the query to resolve is about some inexistent information. If we use the same information of the last example and we formulate the query *Is Bus2 parked at Atlixco?* in this case there is not information about where is Atlixco town so it leads to an infinity loop. The reason of the infinity loop, in this example, is because the meta-interpreter tries to resolve the clauses **contR** or **contT** corresponding to the rule ⊑ to Space or Time mentioned in Section 2.2., and when it fails, it tries to resolve the query once and again.

5. CONCLUSION

Thanks to the implementation of STACLP we could realize that the axioms about **(at th)**, **(atp thr)**, **(at in)** and **(tp inr)** can be replaced for the next two: **(th in)**, **(thr inr)**. We used XSB system, because we consider that it has the necessary features for the implementation of the meta-interpreter. Finally, it would be interesting to cope with an interface using Natural Language to specify queries. More ideas about the evolution of STACLP are in [1] [2] [3] [4].

4. ACKNOWLEDGEMENT

This work has been supported by the Mexican Council of Science and Technology (CONACYT) W 35804- A.

REFERENCES

[1] A. Raffaetá, C. Renso, F. Turini, Qualitative Reasoning in a Spatio-Temporal Language, *CRGD Workshop 2001*, http://www-kdd.cnuce.cnr.it/crgd/

[2] C. Renso, A. Raffaetá. Temporal Reasoning in Geographical Information Systems. *DEXA Workshop, 2000*, 899-905.

[3] P. Mancarella, G. Nerbini, A.Raffaetá, F.Turini, MuTACLP: A Language for Declarative GIS Analysis, *Computational Logic*, 2000, 1002-1016

[4] A.Raffaetá and T. Frühwirth, Spatio-Temporal Annotated Constraint Logic Programming, *PADL 2001*, 2001, 1990, 259-273.

[5] W. Chen and D. S. Warren, Tabled Evaluation with Delaying for General Logic Programs, *Journal of the ACM,43*(1), 1996, 20-74

[6]D. S. *Warren, Applications of logic databases. Programming the PTQ Grammar in* XSB (Ed.: Raghu Ramakrishnan, Kluwer Academic Publishers, USA. 1995).

[7] M. Kifer, V. S. Subrahmanian, Theory of generalized annotated logic programming and its applications, *J. Logic Programming, 12*(4), 1992, 335-368.

[9] J. Alferes, C. Damasio, and L. Pereira, SLX: a top-down derivation procedure for programs with explicit negation, *International Logic Programming Symp.*, 1994, 424-439.