Tabled CLP for Reasoning over Stream Data*

Joaquín Arias[1,2]

1 **IMDEA Software Institute**
`joaquin.arias@imdea.org`
2 **Technical University of Madrid**
`joaquin.arias.herrero@alumnos.upm.es`

Abstract

The interest in reasoning over stream data is growing as quickly as the amount of data generated. Our intention is to change the way stream data is analyzed. This is an important problem because we constantly have new sensors collecting information, new events from electronic devices and/or from customers and we want to reason about this information. For example, information about traffic jams and costumer order could be used to define a deliverer route. When there is a new order or a new traffic jam, we usually restart from scratch in order to recompute the route. However, if we have several deliveries and we analyze the information from thousands of sensors, we would like to reduce the computation requirements, e.g. reusing results from the previous computation. Nowadays, most of the applications that analyze stream data are specialized for specific problems (using complex algorithms and heuristics) and combine a computation language with a query language. As a result, when the problems become more complex (in e.g. reasoning requirements), in order to modify the application complex and error prone coding is required.

We propose a framework based on a high-level language rooted in logic and constraints that will be able to provide customized services to different problems. The framework will discard wrong solutions in early stages and will reuse previous results that are still consistent with the current data set. The use of a constraint logic programming language will make it easier to translate the problem requirements into the code and will minimize the amount of re-engineering needed to comply with the requirements when they change.

1998 ACM Subject Classification D.3.2 Constraint and logic languages, I.2.8 Graph and tree search strategies, H.2.8 Data mining

Keywords and phrases logic, languages, tabling, constraints, graph, analysis, reasoning

Digital Object Identifier 10.4230/OASIcs.ICLP.2016.17

1 Introduction and Problem Description

In recent years, wired and wireless sensors, social media and the Internet of Things generate data (stream data) which is expanding in three fronts: velocity (speed of data generation), variety (types of data) and volume (amount of data). As a result the demand for analysis and reasoning over stream data (stream data mining) has exploded [17].

The main property of stream data is that the sets of data change due to insertion, modification and/or deletion of data. In most cases, the subset of changed data is substantially smaller than the complete amount of data which is analyzed. The objective of stream data mining is to find relations and associations between the values of categorical variables in big sets of data (millions of items or more), which are dynamically updated.

* Work partially funded by Comunidad de Madrid project S2013/ICE-2731 *N-Greens Software* and MINECO Projects TIN2012-39391-C04-03 *StrongSoft* and TIN2015-67522-C3-1-R *TRACES*.

Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016).
Editors: Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos; Article No. 17; pp. 17:1–17:8

Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Datalog, a high level language based on logic, has demonstrated its efficiency in stream reasoning (systems like Deductive Applications Language System (DeALS) [25] developed in UCLA, StreamLog [33] and Yedalog [7] by Google, are based on Datalog), and in machine learning where the queries are executed in parallel over big databases distributed in different clusters.

Our system is based on Prolog because it is more expressive (Datalog is semantically a subset of Prolog) and its native search strategy is top-down instead of bottom-up. This means that the search is guided by the query reducing the search tree. Our system also provides two extensions: constraints logic programming, which discards search options in early stages reducing the search tree, and tabling, an execution strategy which avoids entering loops in some cases and reuses previous results. As a result these extensions not only increase the performance of Prolog but also its expressiveness as we show in Sec. 4.

When data changes, most current approaches have to recompute the analysis over the complete data set. With our system, the combination of tabling and constraints will minimize or override the recomputation overhead by computing only the subset of data affected by the modified data, because:

- The tabling engine will invalidate results that are inconsistent with the current data set in order to reuse previous results in such a way that we can ensure they are correct.
- New constraints solvers will make it possible to define restriction to prune the search tree during the data analysis. A pruned search tree reduces the number of accesses to databases and/or tables.

2 Background and Overview of the existing literature

In this section we will describe the framework (TCLP) which make it possible to integrate constraints solvers in the tabling engine; the data model that will be used to represent data; the constraints needed to deal with the dynamic nature of the data; and a brief state of the art.

2.1 TCLP: Tabling + Constraints

Constraint Logic Programming (CLP) [13] extends Logic Programming (LP) with variables which can belong to arbitrary constraint domains and the ability to incrementally solve the equations involving these variables. CLP brings additional expressive power to LP, since constraints can very concisely capture complex relationships between variables. Also, shifting from "generate-and-test" to "constrain-and-generate" patterns reduces the search tree and therefore improves performance, even if constraint solving is in general more expensive than unification.

Tabling [26, 30] is an execution strategy for logic programs which suspends repeated calls which would cause infinite loops. Answers from other, non-looping branches, are used to resume suspended calls which can in turn generate more answers. Only new answers are saved, and evaluation finishes when no new answers can be generated. Tabled evaluation always terminates for calls / programs with the bounded term depth property and can improve efficiency for programs which repeat computations, as it automatically implements a variant of dynamic programming. Tabling has been successfully applied in a variety of contexts, including deductive databases, program analysis, semantic Web reasoning, and model checking [31, 9, 34, 19].

The combination of CLP and tabling [28, 22, 8, 5], called TCLP, brings several advantages. It improves termination properties and increases speed in a range of programs. It has been

```
person {                          triple(s01, type,  person)
  name: "John Doe"                triple(s01, name,  "John Doe")
  email: "jdoe@gmail.com"         triple(s01, email, "jdoe@gmail.com")
}
```

Figure 1 A person model in Protocol Buffers (left) and Prolog syntax (right)

applied in several areas, including constraint databases [14, 28], verification of timed automata and infinite systems [4], and abstract interpretation [27].

2.2 Graph Databases

Graph Databases are increasingly used to store data and most of the current data, such as linked data on the Web and social network data, are graph-structured [32]. A graph database is essentially a collection of nodes and edges. There are different graph data models but we will limit our research to the directed labeled graphs where the edges are directed and identified with a label.

The Resource Description Framework (RDF) [20] is a standard model for data interchange on the Web. RDF referees an edge as a "triple" <subject> <predicate> <object> and allows structured and semi-structured data to be mixed, exposed, and shared across different applications. As a result, it facilitates the integration of data from different sources. The RDF model theory also formalizes the notion of inference in RDF and provides a basis for computing deductive closure of RDF graphs.

The OWL Web Ontology Language [12], based on the RDF framework, was designed to represent rich and complex knowledge about things, groups of things, and relations between things. OWL documents, known as ontologies, define concept type hierarchies in such a way that a property defined for a more general concept is also defined for the concept subsumed by the more general concept. It is also possible to define various hierarchical relations.

OWL is a computational logic-based language that can be exploited to verify the consistency of the database knowledge or to make explicit an implicit knowledge. As a result, since Prolog is also a logic-based language, there are several RDF-APIs in Prolog which provide an interface to RDF databases and engine interfaces based on Prolog like F-OWL [34] to reasoning over OWL ontologies. The RDF triples can be easily translated into Prolog i.e. using facts of the form triple(Subject, Predicate, Object).

Other languages, like Protocol Buffers [3] based on name-value pairs, which Google uses as a common representation of data, can also be modelled as a directed labeled graph. Fig 1 shows the model of a person with a name and an email in protocol buffer test format (left) and in Prolog syntax (right).

The data model based on directed labeled graphs combined with the unification of Prolog, makes it easy to read, write, match and transform the data. Additionally, Sec. 4 shows that TCLP will increase the performance and termination properties of Prolog in most of the reasoning problems over graph databases because they can be solved in terms of reachability, connectivity, and distance in graphs.

2.3 Stream Time Constraints

The analysis of stream data has to deal with the unbounded nature of the data. First, it is not possible to store all the generated data, therefore several techniques have been developed to process the data and to store only the relevant information. Second, the queries have to

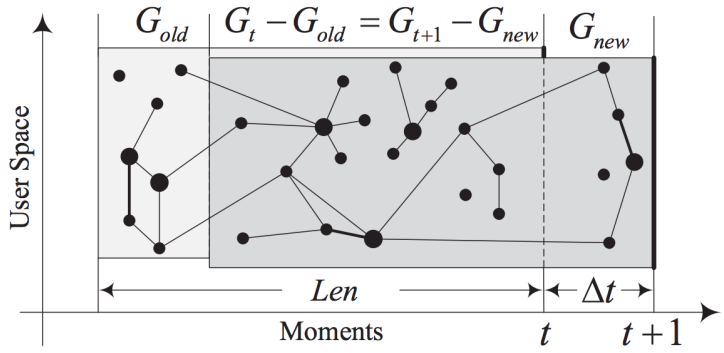

Figure 2 Sliding time window from time t to t+1. G_t will be updated by deleting subgraph G_{old} and adding subgraph G_{new}. Example from [16].

be re-evaluated periodically and in one pass because the source data is not stored for further evaluation.

Usually the reasoning is performed over a snapshot of a finite amount of data (a window). A window is defined by its size (a fixed time interval or a number of data items) and, since the queries are repeated in the time, it is also defined by a slide distance (the time between two consecutive queries). In many applications, the time interval size of the window is larger than the slide distance, so the set of data that is modified (due to addition or deletion) is smaller than the set of data contained in the window. Our intention is to design a system that updates the results recomputing the part of the modified data instead of recomputing the query over the complete data set. Similar work presented in [16] applied an incremental tracking framework (see Fig 2) to the event evolution tracking task in social streams, showing much better efficiency than other approaches.

Constraint logic programming provides arithmetical constraint solvers that can deal with the window definition in a natural manner (i.e. interval constraint solvers), and the operations required to deal with temporal reasoning [1] can be evaluated by the constraint solver.

2.4 State of the Art

In recent years several new logic languages, most of which are based on Datalog, have been developed to reason over stream data. Two of them are: Yedalog [7] developed by Google, an extension of Datalog that seamlessly mixes data-parallel pipelines and computation in a single language, and adds features for working with data structured as nested records; and LogiQL [11] developed by LogicBlox, a unified and declarative language based on Datalog with advanced incremental maintenance (changes are computed in an incremental fashion) and live programming facilities (changes to application code are quickly compiled and "hot-swapped" into the running program). There is more research done in this direction and some of its results are described in the surveys [17, 32].

3 Goal of the Research

Our goal is to extend the functionality of Prolog (logic programming language) to provide a full high level programming language which can be used to reason over stream data, reusing previous results instead of recomputing them from scratch when new data arrives.

We intend to make the stream analysis a native capability of our system by using the monotonicity of logic programming and by introducing the revision of previous inferences when facts are removed, which is a form of non-monotonicity.

We envision advantages in several fronts: complex queries and non-trivial reasoning will be easier to express thanks to the higher-level of logic programming and constraints; fewer computations will be necessary thanks to the automatic reuse of previous inferences brought by tabling (which in a certain sense performs dynamic programming in an automatic way); queries and associated actions (if any) can be programmed using the same syntax.

4 Current Status of the Research and Results Accomplished

During my first year of PhD I have been designing and implementing the TCLP framework which eases the integration of additional constraint solvers in an existing tabling module in Ciao Prolog[1].

The main goal of the TCLP framework is to make the addition of constraint solvers easier. In order to achieve this goal, we determined the services that a constraint solver should provide to the tabling engine. The constraint solver can freely implement them and has been designed to cover many different implementations.

To validate our design we have interfaced: one solver for difference constraints, previously written in C, existing classical solvers (CLP(Q/R)), and a new solver for constraints over finite lattices. We have found the integration to be easy — certainly easier than with other designs, given the capabilities that our system provides. We evaluate the performance of our framework in several benchmarks using the aforementioned constraint solvers. All the development work and evaluation was done in Ciao Prolog and is described in [2].

In order to highlight some of the advantages of TCLP versus Prolog, CLP and tabling with respect to declarativeness and logical reading, we compare the behavior of these paradigms and strategies using different versions of a program to compute distances between nodes in a graph. Each version is adapted to a different paradigm, but trying to stay as close as possible to the original code, so that the additional expressiveness can ultimately be attributed to the semantics of the programming language and not to differences in the code itself.

```
dist(X, Y, D) :-
    dist(X, Z, D1),
    edge(Z, Y, D2),
    D is D1 + D2.
dist(X, Y, D) :-
    edge(X, Y, D).
```

```
dist(X, Y, D) :-
    D1 #> 0, D2 #> 0,
    D  #= D1 + D2,
    dist(X, Z, D1),
    edge(Z, Y, D2).
dist(X, Y, D) :-
    edge(X, Y, D).
```

Figure 3 Versions of distance in a graph: Prolog / tabling (left) and CLP / TCLP (right). The symbols #> and #= are (in)equalities in CLP.

The code in Fig. 3, left, is the Prolog / tabling version of the program dist/3 to find nodes in a graph within a distance K from each other. Fig. 3, right, is the CLP / TCLP version

[1] A robust, mature, next-generation Prolog system. Stable versions of Ciao Prolog are available at `http://www.ciao-lang.org`.

	Prolog	CLP	Tabling	TCLP	
Left recursion	–	–	144	45	Without
Right recursion	1917	200	291	184	cycles
Left recursion	–	–	–	420	With
Right recursion	–	4261	–	1027	cycles

Table 1 Run time (ms) for dist/3. A '–' means no termination.

of the same code. In order to find the nodes X and Y within a maximum distance K from each other we use the queries ?- dist(X,Y,D), D < K. and ?- D #< K, dist(X,Y,D). in Prolog / tabling and CLP / TCLP, respectively. To evaluate the performance, we use a graph of 25 nodes without cycles (with 584 edges) or with cycles (with 785 edges).

Table 1 shows the termination properties and speed of dist/3 in the four paradigms. It highlights that TCLP terminates in all the cases and it is also the fastest one. Additionally, it shows, in line with the experience on tabling, that left-recursive implementations are usually faster and preferable.

These results are relevant because most of the reasoning problems over graph databases are solved in terms of reachability, connectivity and distances in graphs. In fact, this example is a typical query for the analysis of social networks [24].

5 Open Issues and Expected Achievements

Constraint solver over ontologies The idea of answer subsumption (which only stores an answer if it is more general than the previous answers according to a defined partial order) was presented in [24]. The paper also analyzes its application in social network analysis. From our point of view, the TCLP framework will increase this performance because it can be used not only to check answer subsumption, but also to avoid the execution of queries where the concepts are more particular (they are entailed in terms of the ontology hierarchy) than the concepts of a previous query. Moreover, the constraint solver can be used to state the relationships defined in the ontology as constraint before the analysis starts. These relations can propagate and prune the search space reducing the computation and eventually avoiding accesses to databases.

Temporal constraint solver The analysis should be done over a finite window of time, therefore a constraint solver is needed to deal with the operation required by the temporal reasoning tasks [1]. Moreover, the integration of the solver with the TCLP framework will increase its benefits because some of its operations will explode the stored results stored.

Stream-TCLP In order to apply our framework to stream data, the answers must be returned as soon as they are available. Instead of the local scheduling which tries to find all the answers before returning them, the tabling engine should use an incremental answering strategy similar to batch scheduling [10], JET mechanism [21] or swapping evaluation [6].

Dynamic tabling A more complex technique - similar to incremental tabling [23] - has to be defined in order to: invalidate knowledge inferred by data which is updated / removed; update the knowledge when the temporal window slides; and remove previous tabled results to make place for more recent results.

Stream recursive aggregates Some research has been done in the field of aggregates (see [15, 18, 29]) regarding the Prolog program semantic in tabled execution and with recursive

queries. And since most of the queries are defined in terms of aggregates as min, sum or count, it is relevant to take into consideration this research problem which is unclear and related with non-monotonic properties.

References

1 James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 J. Arias and M. Carro. Description and Evaluation of a Generic Design to Integrate CLP and Tabled Execution. In *18th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'16)*. ACM Press, September 2016.

3 Protocol Buffers. `https://developers.google.com/protocol-buffers/`.

4 Witold Charatonik, Supratik Mukhopadhyay, and Andreas Podelski. Constraint-based infinite model checking and tabulation for stratified clp. In Peter J. Stuckey, editor, *ICLP*, volume 2401 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 2002.

5 P. Chico de Guzmán, M. Carro, M. Hermenegildo, and P. Stuckey. A General Implementation Framework for Tabled CLP. In Tom Schrijvers and Peter Thiemann, editors, *FLOPS'12*, number 7294 in LNCS, pages 104–119. Springer Verlag, May 2012.

6 P. Chico de Guzmán, M. Carro, and David S. Warren. Swapping Evaluation: A Memory-Scalable Solution for Answer-On-Demand Tabling. *Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP'10) Special Issue*, 10 (4–6):401–416, July 2010.

7 Brian Chin, Daniel von Dincklage, Vuk Ercegovac, Peter Hawkins, Mark S Miller, Franz Och, Christopher Olston, and Fernando Pereira. Yedalog: Exploring knowledge at scale. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 32. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

8 Baoqiu Cui and David Scott Warren. A System for Tabled Constraint Logic Programming. In *Computational Logic*, pages 478–492, 2000.

9 S. Dawson, C. R. Ramakrishnan, and D. S. Warren. Practical Program Analysis Using General Purpose Logic Programming Systems – A Case Study. In *Proceedings of the ACM SIGPLAN'96 Conference on Programming Language Design and Implementation*, pages 117–126, New York, USA, 1996. ACM Press.

10 Juliana Freire, Terrance Swift, and David Scott Warren. Beyond Depth-First Strategies: Improving Tabled Logic Programs through Alternative Scheduling. *Journal of Functional and Logic Programming*, 1998(3), 1998.

11 Todd J Green, Dan Olteanu, and Geoffrey Washburn. Live programming in the LogicBlox system: a MetaLogiQL approach. *Proceedings of the VLDB Endowment*, 8(12):1782–1791, 2015.

12 OWL Web Ontology Language Guide. `http://www.w3.org/TR/owl-guide/`.

13 J. Jaffar and M.J. Maher. Constraint LP: A Survey. *JLP*, 19/20:503–581, 1994.

14 Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint Query Languages. *J. Comput. Syst. Sci.*, 51(1):26–52, 1995.

15 David B Kemp and Peter J Stuckey. Semantics of logic programs with aggregates. In *ISLP*, volume 91, pages 387–401. Citeseer, 1991.

16 Pei Lee, Laks VS Lakshmanan, and Evangelos E Milios. Incremental cluster evolution tracking from highly dynamic network data. In *2014 IEEE 30th International Conference on Data Engineering*, pages 3–14. IEEE, 2014.

17 Emanuele Panigati, Fabio A Schreiber, and Carlo Zaniolo. Data streams and data stream management systems and languages. In *Data Management in Pervasive Systems*, pages 93–111. Springer International Publishing, 2015.

18 Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe. Well-Founded and Stable Semantics of Logic Programs with Aggregates. *TPLP*, 7(3):301–353, 2007.

19 Y.S. Ramakrishna, C.R. Ramakrishnan, I.V. Ramakrishnan, S.A. Smolka, T. Swift, and D.S. Warren. Efficient Model Checking Using Tabled Resolution. In *CAV*, volume 1254 of *LNCS*, pages 143–154. Springer Verlag, 1997.

20 Resource Description Framework (RDF). `https://www.w3.org/RDF/`.

21 Konstantinos F. Sagonas and Peter J. Stuckey. Just Enough Tabling. In *Principles and Practice of Declarative Programming*, pages 78–89. ACM, August 2004.

22 Tom Schrijvers, Bart Demoen, and David Scott Warren. TCHR: a Framework for Tabled CLP. *TPLP*, 8(4):491–526, 2008.

23 Terrance Swift. Incremental tabling in support of knowledge representation and reasoning. *Theory and Practice of Logic Programming*, 14(4-5):553–567, 2014.

24 Terrance Swift and David Scott Warren. Tabling with answer subsumption: Implementation, applications and performance. In Tomi Janhunen and Ilkka Niemelä, editors, *JELIA*, volume 6341 of *Lecture Notes in Computer Science*, pages 300–312. Springer, 2010.

25 Deductive Application Language System. `http://wis.cs.ucla.edu/deals/`.

26 H. Tamaki and M. Sato. OLD Resol. with Tabulation. In *ICLP*, pages 84–98. LNCS, 1986.

27 David Toman. Constraint Databases and Program Analysis Using Abstract Interpretation. In *CDTA*, volume 1191 of *LNCS*, pages 246–262, 1997.

28 David Toman. Memoing Evaluation for Constraint Extensions of Datalog. *Constraints*, 2(3/4):337–359, 1997.

29 Alexander Vandenbroucke, Maciej Pirog, Benoit Desouter, and Tom Schrijvers. Tabling with Sound Answer Subsumption. *Theory and Practice of Logic Programming, 32th Int'l. Conference on Logic Programming (ICLP'16)*, 16, October 2016.

30 D. S. Warren. Memoing for Logic Programs. *CACM*, 35(3):93–111, 1992.

31 R. Warren, M. Hermenegildo, and S. K. Debray. On the Practicality of Global Flow Analysis of Logic Programs. In *JICSLP*, pages 684–699. MIT Press, August 1988.

32 Peter T Wood. Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60, 2012.

33 Carlo Zaniolo. A logic-based language for data streams. In *SEBD*, pages 59–66, 2012.

34 Youyong Zou, Tim Finin, and Harry Chen. F-OWL: An Inference Engine for Semantic Web. In *Formal Approaches to Agent-Based Systems*, volume 3228 of *Lecture Notes in Computer Science*, pages 238–248. Springer Verlag, January 2005.