Declarative Algorithms in Datalog with Extrema: Their Formal Semantics Simplified

Carlo Zaniolo
University of California, Los Angeles, USA
zaniolo@cs.ucla.edu

Mohan Yang
Google, USA
yang@cs.ucla.edu

Matteo Interlandi
Microsoft Corporation, USA
matteo.interlandi@microsoft.com

Ariyam Das
University of California, Los Angeles, USA
ariyamo@cs.ucla.edu

Alexander Shkapsky
University of California, Los Angeles, USA
shkapsky@gmail.com

Tyson Condie
University of California, Los Angeles, USA
tcondie@cs.ucla.edu

Abstract

Recent advances are making possible the use of aggregates in recursive queries thus enabling the declarative expression classic algorithms and their efficient and scalable implementation. These advances rely the notion of Pre-Mappability (*PreM*) of constraints that, along with the seminaive-fixpoint operational semantics, guarantees formal non-monotonic semantics for recursive programs with min and max constraints. In this *extended abstract*, we introduce basic templates to simplify and automate task of proving *PreM*.

2012 ACM Subject Classification Information systems → Query languages

Keywords and phrases Recursive Queries

Digital Object Identifier 10.4230/OASIcs.ICLP.2018.9

1 Pre-Mappable Extrema constraints in Recursive Rules

Pre-mappable (*PreM*) extrema constraints in recursive Datalog programs enable concise declarative formulations for classical algorithms [3]. The programs expressing these algorithms have formal non-monotonic semantics [1, 2]. For instance, a classical recursive application for traditional databases is Bill of Materials (BOM), where we have a Directed Acyclic Graph (DAG) of parts-subparts, `assbl(Part, Subpart, Qty)` describing how a given part is assembled using various subparts, each in a given quantity. Not all subparts are assembled, since basic parts are instead supplied by external suppliers in a given number of days, as per the facts `basic(Part, Days)`. Simple assemblies, such as bicycles, can be put together the very same day in which the last basic part arrives. Thus, the time needed to produce the

Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018).
Editors: Alessandro Dal Palu', Paul Tarau, Neda Saeedloei, and Paul Fodor; Article No. 9; pp. 9:1–9:3

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

assembly is the maximum number of days required by the basic parts it uses. This can be computed by the following stratified program:

▶ **Example 1** (How many days till all the required parts arrive)**.**

```
deliv(Part, Days) ←          basic(Part, Days), is_max(Part, Days).
deliv(Part, Days) ←          deliv(Sub, Days), assbl(Part, Sub).
actualDays(Part, Days) ←     deliv(Part, Days), is_max(Part, Days).
```

But the iterated fixpoint computation of the perfect model of this program can be very inefficient. This problem is solved by transferring `is_max((Part), Days)`, to the rules defining `deliv`, whereby the rule defining `actualDays` now becomes a redundant copy rule. Thus from the previous exo-max version of our program we obtain its endo-max version as follows:

```
deliv(Part, Days) ←          basic(Part, Days), is_max((Part), Days).
deliv(Part, Days) ←          deliv(Sub, Days), assbl(Part, Sub), is_max((Part), Days).
actualDays(Part, Days) ←     deliv(Part, Days).
```

The questions we need to answer about our `exo2endo` transformation are the following two:

(i) is this a valid optimization inasmuch as the fixpoint computation of the endo-max program delivers the same model as the iterated fixpoint of the original exo-max program (and still allows recursive optimizations such as seminaive-fixpoint and magic-sets)?

(ii) once we re-express `is_max` using negation, does the transformed program has a unique stable model semantics, efficiently computed as described jn (i).

The notion of *PreM* [3] provides provides a formal answer to both these questions?

▶ **Definition 2** (The *PreM* Property)**.** In a given Datalog program, let P be the rules defining a (set of mutually) recursive predicate(s). Also let T be the ICO defined by P. Then, the constraint γ will be said to be *PreM* to T (and to P) when, for every interpretation I of P, we have that: $\gamma(T(I)) = \gamma(T(\gamma(I)))$.

The importance of this property follows from the fact that if $I = T(I)$ is a fixpoint for T, then we also have that $\gamma(I) = \gamma(T(I))$, and when γ is *PreM* to T then: $\gamma(I) = \gamma(T(I)) = \gamma(T(\gamma(I)))$. Now, let T_γ denote the application of T followed by γ, i.e., $T_\gamma(I) = \gamma(T(I))$. If I is a fixpoint for T and $I' = \gamma(I)$, then the above equality can be rewritten as: $I' = \gamma(I) = \gamma(T(\gamma(I))) = T_\gamma(I')$. Thus, when γ is *PreM* , the fact that I is a fixpoint for T implies that $I' = \gamma(I)$ is a fixpoint for $T_\gamma(I)$. In many programs of practical interest, the transfer of constraints under *PreM* produces optimized programs for the naive fixpoint computation that are safe and terminating even when the original programs were not. Thus we focus on programs where, for some integer n, $T_\gamma^{\uparrow n}(\emptyset) = T_\gamma^{\uparrow n+1}(\emptyset)$, i.e., the fixpoint iteration converges after a finite number of steps n. As proven in [3], the fixpoint $T_\gamma^{\uparrow n}(\emptyset)$ so obtained is in fact a minimal fixpoint for T_γ, where γ denotes a min or max constraint:

▶ **Theorem 3.** *If γ is PreM to a positive program P with ICO T and, for some integer n, $T_\gamma^{\uparrow n}(\emptyset) = T_\gamma^{\uparrow n+1}(\emptyset)$, then:*

(i) *$T_\gamma^{\uparrow n}(\emptyset) = T_\gamma^{\uparrow n+1}(\emptyset)$ is a minimal fixpoint for T_γ, and*

(ii) *$T_\gamma^{\uparrow n}(\emptyset) = \gamma(T^{\uparrow\omega}(\emptyset))$.*

Therefore, when the *PreM* holds, declarative exo-min (or exo-max) programs are transformed into endo-min (or endo-max) programs having highly optimized operational semantics that computes the perfect model of the former and the unique stable modelof the latter.

2 Proving Premappability

The application of a min or max constraint to the ICO of a rule r can be expressed by the addition of a min or max goal to r, whereby *PreM* holds if this insertion of a new goal does not change the mapping defined by the rule. For the example at hand, we have:

$$\texttt{deliv(Part, Days)} \leftarrow \texttt{deliv(Sub, Days)}, \backslash^{\texttt{is\_max((Sub),Days)}}/, \texttt{assbl(Part, Sub)}, \texttt{is\_max((Part), Days)}.$$

Thus, we must prove that the insertion $\backslash^{\texttt{is\_max((Sub),Days)}}/$ does not change the mapping defined by our rule–a property that is guaranteed to hold if can prove that the original mapping already satisfies this constraint. We next define the concept of min- and max-constraints for individual tuples:

▶ **Definition 4.** We will say that a tuple $t \in R$ satisfies the min-constraint `is_min((X), A)` and write $X \xrightarrow{min} A$ when R contains no tuple having the same X-value and a smaller A-value. Symmetrically, we say that the tuple $t \in R$ satisfies the max-constraint `is_max((X), A)` and write $X \xrightarrow{max} A$ when `R` contains no tuple with the same X-value and a larger A-value.

Thus in our example we have $\texttt{Part} \xrightarrow{\texttt{max}} \texttt{Days}$ and must prove that $\texttt{Sub} \xrightarrow{\texttt{max}} \texttt{Days}$. Toward that goal, we observe that $X \xrightarrow{min} A$ and $X \xrightarrow{max} A$ can be informally viewed as "half functional dependencies (FDs)", since both must hold before we can conclude that $X \rightarrow A$. In fact, although min- and max-constraints on single tuples are much weaker than regular FDs, they preserve some of their important formal properties including those involving multivalued dependencies (MVDs) that result from the natural joins in the recursive rules – e.g. $\texttt{Sub} \twoheadrightarrow \texttt{Days}$ and $\texttt{Sub} \twoheadrightarrow \texttt{Part}$, in our example.

Therefore, the following properties, proven in [4], hold for tuple for min-constraints, max-constraints, and MVDs, and also illustrate the appeal of the arrow-based notation:

Min/Max Augmentation:	If $X \xrightarrow{min} A$ and $Z \subseteq \Omega$, then $X \cup Z \xrightarrow{min} A$.
	If $X \xrightarrow{max} A$ and $Z \subseteq \Omega$, then $X \cup Z \xrightarrow{max} A$.
MVD Augmentation:	If $X \twoheadrightarrow Y$, $Z \subseteq \Omega$ and $Z \subseteq W$, then $X \cup W \twoheadrightarrow Y \cup Z$.
Mixed Transitivity:	If $Y \twoheadrightarrow Z$ and $Z \xrightarrow{min} A$, with $A \notin Z$, then $Y \xrightarrow{min} A$.
	If $Y \twoheadrightarrow Z$ and $Z \xrightarrow{max} A$, with $A \notin Z$, then $Y \xrightarrow{max} A$.

For the example at hand, $\texttt{Sub} \twoheadrightarrow \texttt{Part}$ and $\texttt{Part} \xrightarrow{\texttt{max}} \texttt{Days}$ implies $\texttt{Sub} \xrightarrow{\texttt{max}} \texttt{Days}$, by mixed transitivity. Since this constraint holds, the additional goal $\backslash^{is\_max((Sub),Days)}/$ enforcing thus max constraint does not change it. Q.E.D.

References

1 Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. Big Data Analytics with Datalog Queries on Spark. In *SIGMOD*, pages 1135–1149. ACM, 2016.

2 Mohan Yang, Alexander Shkapsky, and Carlo Zaniolo. Scaling up the performance of more powerful Datalog systems on multicore machines. *The VLDB Journal*, 26(2):229–248, 2017.

3 Carlo Zaniolo, Mohan Yang, Ariyam Das, Alexander Shkapsky, Tyson Condie, and Matteo Interlandi. Fixpoint semantics and optimization of recursive Datalog programs with aggregates. *TPLP*, 17(5-6):1048–1065, 2017.

4 Carlo Zaniolo, Mohan Yang, Matteo Interlandi, Ariyam Das, Alexander Shkapsky, and Tyson Condie. Declarative Algorithms by Aggregates in Recursive Queries: their Formal Semantics Simplified. Report no. 180001, Computer Science Department, UCLA, April, 2018.