Verification of Prev-Free Communicating Datalog Processes

Francesco Di Cosmo

Free University of Bozen-Bolzano, Italy

Abstract

Communicating Datalog Programs (CDPs) are a distributed computing model grounded on logic programming, where networks of nodes perform Datalog-like computations, leveraging also on information coming from incoming messages and databases received from interactive external services. In previous works, the decidability and complexity border of verification for different variants of CDPs was charted. While the problem is undecidable in general, model-checking FO-CTL formulae (specialized to the distributed setting) is PSPACE-complete in data-complexity for CDPs where all data-sources, except the external inputs, are bounded. While an intuitive explanation for decidability is that "a bounded state is unable to fully take advantage of an unbounded input", a formal justification is missing. At closer inspection, we have noticed that CDPs have a limited capability of handling external inputs, i.e., they cannot compare two successive instances. Thus, an alternative explanation is that an unbounded data-source does per se not cause undecidability, as long as the CDP cannot compare two successive instances. In this short paper, we report about our work in progress on this problem.

Keywords

Logic Programming, Formal Verification, Distributed Systems, Decidability

1. Introduction

Declarative languages are appreciated for their capability of specifying queries and algorithms in an elegant and succinct way. For that reason, in the last decades, Datalog-like languages have been proposed as a programming language for distributed systems. Some examples are Webdamlog [1], NDlog [2], and Dedalus [3]. We loosely call this kind of model *Declarative Distributed Systems* (DDSs). The data-centricity of Datalog-like languages makes the DDS approach especially interesting also for modelling data-aware Business Processes [4, 5, 6]. In fact, DDSs can be considered as a natural model for interdependent business processes in which the identity of the data items is crucial and cannot be abstracted away.

Since DDSs are formalized in logic, one can directly apply formal methods for verification. However, the verification of data-aware systems is hard, since they can manipulate fresh data provided by input databases (DBs) from external services, resulting in infinite state systems. Unsurprisingly, verification of problems like control-state reachability is undecidable.

Nevertheless, previous works [7] showed that decidability is gained when the active domains of all *data-sources*, i.e., the DBs representing the inputs, the internal memory of the nodes,

CILC'23: 38th Italian Conference on Computational Logic, June 21–23, 2023, Udine, Italy
fdicosmo@unibz.it (F. Di Cosmo)
0000-0002-5692-5681 (F. Di Cosmo)

and the channel configurations, are assumed bounded. A remarkable exception is the data-source representing the interaction with dynamic external services. In fact, leaving this source unbounded does not, per se, cause undecidability or a change in complexity. An intuitive explanation is that an otherwise bounded DDS cannot make proper use of unbounded external inputs.

However, this fact could also depend on additional reasons, based on the peculiarity of the interaction of DDSs with external sources, e.g., lack of queries over previously provided external data. In fact, remarkably, the traditional DDS model assumes that nodes can query the previous configuration of their internal memory but not the external inputs. Thus, it may be the case that combinations of boundedness/unboundedness conditions and the ability/inability to query previous instances of the data-sources may have an important impact on the decidability and complexity boundary of verification problems for DDSs.

In this paper, we report about our work in progress towards the study of this problem. To that end, first, we consider a type of DDS, called *Communicating Datalog Programs* (CDP), already studied by Calvanese et al. [7]. We consider all the fragments concerning addition/removal of the boundedness condition and the ability/inability to query the previous configurations on some data-sources. Finally, we sketch proofs to completely chart the decidability boundary for the termination problem of all such fragments.

2. CDPs

We assume that the reader is familiar with the notion of Database (DB), under the logic programming perspective, and with variants of the Datalog language (see [8] for an introduction), specifically Datalog with stratified negation and inequality constraints. Here, we introduce CDPs informally (the interested reader can refer to [7] for a formal definition).

A CDP is a network of nodes running a program, in the Datalog-like language D2C, sharing messages containing single facts. We assume that the network is a connected, reflexive, and symmetric directed graph, where each edge represents a channel. Each node has access to a DB containing its own name and the names of all its neighbors.

Messages are sent on a channel (v, u) from the source node v to the destination node u. They are single relational facts over a dedicated transport signature $\mathcal{T}$. Messages are received according to the *asynchronous policy*, i.e., at each computation step only one sent message is delivered. A configuration for a channel represents the messages sent on the channel that have not yet been received. In CDPs, channels are assumed to be unordered, i.e., messages are never lost but the sending order may be different from the delivery one.[1] Thus, channel configurations are assumed to be bag-DBs, i.e., a finite bag of facts.

Next to channels, CDP nodes receive information from external services. This is modeled by the availability to each node of an *input DB* over an input signature $\mathcal{I}$. We assume that the input DB can freely change over time, i.e., at each computation step, the input DB may be updated to a different (possibly unrelated) DB over $\mathcal{I}$. On top of that, nodes have a local memory, which

[1]This choice is in line with the fact that, in general, Datalog rules (responsible for the production of messages) do not have to be computed in a specific order.

contains auxiliary information deduced by the node during the previous computation step. The latter is stored in a *state DB* over a state signature $\mathcal{S}$.

The reception of a message m at a node v triggers a new computation step.[2] First, m is removed from its channel. Then, while all other nodes remain inactive, i.e., their state DB is not updated, the node v combines the input DB, its previously computed state DB, and the message m labeled by the sender name w (i.e., a labeled transport fact $m@w$) in a single DB, used as extensional data on which a D2C program is computed. The program returns a new state DB and a set of outgoing messages labeled with destinations. Finally, the CDP configuration is upadated by *(1)* updating the state DB of v with the new one and *(2)* adding the outgoing messages (without addresses) to the respective channel configurations.

A *D2C program* is a finite set of Datalog-like rules with special features to handle the reception and sending of messages and the query of the previous state DB. Communication is handled by labeling transport literals T with addresses t, resulting in the formulas $T@t$: in the body of a rule, the address represents the node sending the incoming message; in the head, the address represents the recipient of the outgoing message. To query the previous state DB, a special flag $\mathbf{prev}$ is used: all predicates in its scope should be considered as ranging over the (extensional) previous input, state, and locally received message, while all other predicates should be considered as ranging over the (intensional) current state DB. Traditionally, it is required that only state literals can occur in the scope of $\mathbf{prev}$. In this paper, we allow all literals, irrespective of their signature, to occur in the scope of $\mathbf{prev}$. D2C rules can also make use of inequality constraints and choice operators $\texttt{choice}(\texttt{X}, \texttt{Y})$ and $\texttt{choice}(\texttt{Y})$ a la Saccà and Zaniolo [9]. These are used to enforce functional dependencies $X \mapsto Y$ and $\{\} \mapsto Y$ (which is used to select a single value for Y) in the tuples deduced by the rule.

Finally, the D2C program must be *stratified*, i.e., stratified a la Datalog when the transport predicates in the body and the predicates in the scope of $\mathbf{prev}$ are interpreted as extensional predicates. Both stratification and formal semantics of the program can be formalized via an encoding in traditional Datalog rules. Moreover, we assume that the payload of outgoing messages is retrieved from the state DBs, so that it is not possible to directly transfer the active domain of the input DB into the channels.

We now define constraints on the various CDP data-sources, i.e., state, input, and channel. A configuration is reachable if it can be obtained from the initial configuration in a finite number of computation steps. Given a $b \in \mathbb{N}$, we say that a CDP is b-state bounded (or that the state data-source is b-bounded) if, for each reachable configuration σ, the active domains of all state DBs in σ contain at most b constant. We say that it is b-channel bounded (or that the channel data-source is b-bounded) if all the channel bag-DBs contain at most b facts (considering multiplicities). A similar constraint can be put on the input. However, in this case, we have to change the semantics of CDPs. Specifically, a b-input bounded CDP is a CDP with the provision that the available input DBs have active domain bounded by b. A data-source is bounded if it is b-bounded for some b. A CDP is bounded if all its data-sources are bounded.

Given a CDP D with program Π, a data-source C is *prev-free* if there is no literal in the signature of C that occurs in the scope of $\mathbf{prev}$ in some rule of Π. Otherwise, it is *prev-aware*.

[2] At startup, each self-loop channel contains a special activation message, used to trigger the first computation step.

Input	PF	BPF	BPF	PF	⊤	B	BPF	BPF	PF	B	B
State	B	⊤	B	PF	BPF	B	PF	BPF	BPF	BPF	PF
Channel	B	BPF	PF	BPF	BPF	B	PF	⊤	PF	PF	B
Status	D	U	U	U	U	D	D	U	D	U	D

Table 1
The CDP fragments we consider in this paper. B indicates that the data-source is bounded; PF indicates that the data-source is prev-free; ⊤ indicates that the data-source is not constrained by neither boundedness nor prev-freeness. The first three fragments were studied by Calvanese et al. [7].

3. Problem

Previous results on the verification of CDPs showed that CTL_{DDS} model checking is decidable only if all the data-sources are bounded [7],[3] with the only exception of the input data-source, whose unboundedness is irrelevant. In all other cases, problems like control-state reachability, termination, and convergence (expressible in CTL_{DDS}) are undecidable. The decidability for unbounded input can be explained by the fact that a bounded state cannot make proper use of an unbounded input.

However, the input data-source is unique in its kind because it is the only one on which it is not possible to simulate a **prev** by exploiting the **prev** on the state, without violating state boundedness assumptions. In other words, the absence, by definition, of **prev** over the input data-source is not without loss of generality and may *(1)* motivate the unique impact of input-unboundedness on decidability of verification and *(2)* prevent the formulation of interesting CDP fragments, with decidable verification, based on combinations of boundedness conditions and non-availability of the **prev** operator over the various data-sources.

We name fragments by means of a triple of constraints on the various data-sources. Specifically, $C_i - C_s - C_c$ denotes the family of all CDPs whose input data-source is constrained by C_i, whose state data-source is constrained by C_s, and whose channel data-source is constrained by C_c. We consider four types of constraints: BPF enforces both boundedness and prev-freeness, B enforces boundedness but not necessarily prev-freeness, PF enforces prev-freeness but not necessarily boundedness, and ⊤ does not enforce any constraint. Thus, we are dealing with 64 fragments. We call them *prev-based* fragments.

As verification task, we focus on the problem of *termination*[4] because *(1)* this is one of the problems that characterises the decidability status of model checking problems against CTL_{DDS} and *(2)* it is closely related with other problems, such as *control-state reachability* and *convergence*.[5]

The decidability of termination of some of these fragments is an immediate consequence of the results by Calvanese et al. [7]. These are depicted in the first three columns of Table 1. We sketch proofs to extend the known cases to the full picture in the table. It is easy to see that

[3] CTL_{DDS} is a specialization of CTL-FO to the distributed setting of CDPs: labeled FO formulas are used to query the node state DBs and channel bag-DBs, while CTL is used to analyze the temporal evolution.
[4] Termination asks whether there is a run that reaches a configuration where no node can be activated anymore.
[5] Since termination occurs when the channels are empty, termination over bounded states can be viewed as control-state reachability. Moreover, convergence, i.e., the reachability of a configuration from which the state DBs do not change anymore, is a generalization of termination.

these cases are sufficient to completely classify the decidability status of all 64 fragments, since each other case is a fragment of a decidable one or an extension of an undecidable one.

Finally, we focus on single-node networks. This comes without loss of generality, since any arbitrary network can be encoded in a single-node network, at the cost of considering a sort of disjoint union of the CDP signatures and program.

4. Proof Sketches

PF-PF-BPF This fragment has undecidable termination, since, given a 2-counter machine (2CM) C, we can build a PF-PF-BPF CDP D that terminates iff C terminates, which, in turn, is an undecidable problem. This can be done by encoding finite runs of C in input DBs. By exploiting the unbounded state DB, we can write a program Π that does not make use of $\mathbf{prev}$ such that, as soon as an input DB I is received, it produces a `terminate` flag iff I encodes a terminating run of C. In this case, no message is sent on the channel (making it empty). Otherwise, `foo` messages will be sent on the channel forever. The program Π exists because it can make use of an unbounded state DB, which can be used to compute, most notably, transitive closures of unbounded graphs in input DBs.

$\top$-BFP-BPF Also this fragment has undecidable termination. In fact, we can modify the previous technique by scattering the provision of a 2CM run along many steps. Specifically, we can encode the next configuration of the (unbounded) counters of C in the input DB and use $\mathbf{prev}$ on the input to check whether the new configuration can be reached via a transition of C. If this test fails, non-termination is triggered as above. Otherwise, termination is triggered only if the final state of C is reached.

B-B-B This fragment has decidable termination. In fact, since all data-sources are bounded, we can use the same technique as for BPF-B-BPF CDPs by Calvanese et al. [7] (Theorem 1). The idea is that, because of boundedness, it is possible to finitely represent CDP configurations up to isomorphisms. The possible prev-awareness of all data-sources simply results in an extension of the abstracted representation of the data-sources, but does not affect the applicability of the method.

BPF-PF-PF This fragment has decidable termination. Since the state is prev-free, the new state contains only constants from the incoming message (which are in bounded number because the transport signature has a maximal finite arity) and from the input DB, which is bounded. Thus, also the state is bounded. Notice that, since all sources are prev-free, the name of constants in the messages on the channel are irrelevant. Thus, we can abstract away the messages up to isomorphisms. Since each message contains a bounded number of constants, they result in a finite family of messages. We interpret them as non-terminal symbols of a context free grammar (CFG). The reception of one of these symbols results in the production of a (possibly empty) set of non-terminal symbols, by sending outgoing (abstracted) messages. Thus, the program can be encoded in a finite number of production rules, defining a CFG. Hence, termination of BPF-PF-PF CDPs reduces to the emptyness problem of CFGs, which is decidable.

BPF-BPF-$\top$ This fragment has undecidable termination. The idea is that we can, at each step, encode in the unbounded channel an arbitrary natural number. This can be achieved via a binary predicate `succ`, maintaining a structure of the form: $\texttt{succ}(\texttt{min}, \texttt{n}_0), \ldots, \texttt{succ}(\texttt{n}_\texttt{m}, \texttt{max})$ where `min` and `max` are special constants mentioned in the program and the $\texttt{n}_\texttt{i}$s are pairwise different constants retrieved from the input. By exploiting the prev-awareness of the channel and exploiting a technique similar to the one from Theorem 2 in [7], the node is able to scan the full content of the channel in order to understand whether a constant c retrieved from the input is actually fresh or not. To maintain the constant c across multiple steps, we have to incorporate it in the last sent message and expect to receive it in the next one. This could be done by exploiting a ternary version of `succ` and the prev-awareness of the channel. Similarly, we can further extend the arity of `succ` in order to encode and maintain also the state and counters configuration of a 2CM. As in the previous case, only when reaching the terminal state of the 2CM, CDP termination is triggered.

PF-BPF-PF This fragment has decidable termination. In fact, we can extend the technique from Theorem 4 in [7] to abstract away the unbounded input DB in front of a bounded state DB. The idea is that a bounded state can only make use of an unbounded input to *(1)* retrieve a bounded number of constants and *(2)* answer a finite set of Boolean queries, which behave as guards to rules of the D2C program. By abstracting away the input with the value of those constants and queries, the fragment boils down to BPF-BPF-PF, whose decidability directly follows from that of the BPF-PF-PF fragment.

B-BPF-PF This fragment has undecidable termination. In fact, we can use the **prev** on the input to simulate a **prev** on the channels. It suffices to require that the current received message is also mentioned in the current input DB. If that is not the case, an error is deduced and non-termination is triggered. Thus, this fragment can encode an extension of BPF-BPF-$\top$, which was argued to be undecidable.

B-PF-B This fragment has decidable termination. In fact as in a previous case, the state turns out to be bounded, thus boiling down to the decidable fragment B-BPF-B.

5. Conclusions

We have sketched proofs to establish the decidability status of termination for 8 prev-based fragments. Together with the previous results in [7], these can be generalized to categorize all 64 fragments. In our opinion, the most surprising result is the one about BPF-BPF-$\top$, since prev-awareness on the channel data-source, on which the node has a very limited control, returns undecidability. The results above enable a fine-grained discussion on the interaction of boundedness and prev-freeness. As a preliminary remark, we can conclude that the principle that "unboundedness causes undecidability only in front of **prev**" is not completely correct, because of the undecidability of the fragment B-BPF-PF, which has an unbounded but prev-free channel. However, in that case, we obtained undecidability by simulating **prev** on the channel by using **prev** on the input. Notice that the opposite is not true, in the sense that it is not

possible to use **prev** on the channel to simulate **prev** on the input (PF-BPF-B is decidable). This indicates that the principle regulating the interaction between boundedness and prev-freenes is sensitive to the nature of the constrained data-sources.

References

[1] É. Antoine, Distributed data management with a declarative rule-based language Webdamlog. (Gestion des données distribuées avec le langage de règles Webdamlog), Ph.D. thesis, University of Paris-Sud, Orsay, France, 2013.

[2] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, I. Stoica, Implementing declarative overlays, in: A. Herbert, K. P. Birman (Eds.), Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP), Brighton, UK, October 23-26, 2005, ACM, 2005, pp. 75–90.

[3] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier, R. Sears, Dedalus: Datalog in time and space, in: O. de Moor, G. Gottlob, T. Furche, A. J. Sellers (Eds.), Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers, volume 6702 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 262–281.

[4] A. Deutsch, L. Sui, V. Vianu, Specification and verification of data-driven web applications, J. Comput. Syst. Sci. 73 (2007) 442–474.

[5] F. Belardinelli, A. Lomuscio, F. Patrizi, An abstraction technique for the verification of artifact-centric systems, in: G. Brewka, T. Eiter, S. A. McIlraith (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012, AAAI Press, 2012.

[6] B. B. Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, M. Montali, Verification of relational data-centric dynamic systems with external services, in: R. Hull, W. Fan (Eds.), Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), New York, NY, USA - June 22 - 27, 2013, ACM, 2013, pp. 163–174.

[7] D. Calvanese, F. Di Cosmo, J. Lobo, M. Montali, Convergence verification of declarative distributed systems, in: S. Monica, F. Bergenti (Eds.), Proceedings of the 36th Italian Conference on Computational Logic, Parma, Italy, September 7-9, 2021, volume 3002 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 62–76.

[8] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

[9] D. Saccà, C. Zaniolo, Stable models and non-determinism in logic programs with negation, in: D. J. Rosenkrantz, Y. Sagiv (Eds.), Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA, ACM Press, 1990, pp. 205–217.