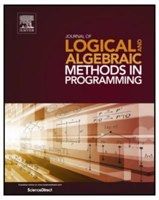
Journal of Logical and Algebraic Methods in Programming 100 (2018) 98–112

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



Finding models through graph saturation

Sebastiaan J.C. Joosten

Formal Methods and Tools group, University of Twente, the Netherlands

A R T I C L E I N F O

Article history:
Received 29 October 2017
Received in revised form 21 April 2018
Accepted 16 June 2018
Available online xxxx

A B S T R A C T

We give a procedure that can be used to automatically satisfy invariants of a certain shape. These invariants may be written with the operations intersection, composition and converse over binary relations, and equality over these operations. We call these invariants sentences that we interpret over graphs. For questions stated through sets of these sentences, this paper gives a semi-decision procedure we call graph saturation. It decides entailment over these sentences, inspired on graph rewriting. We prove correctness of the procedure. Moreover, we show the corresponding decision problem to be undecidable. This confirms a conjecture previously stated by the author [7].

1. Introduction

The question ‘what models does a set of formulas $\mathcal{T}$ have’ has practical relevance, as it is an abstraction of an information system: We interpret the data set stored in an information system at a certain point in time as a model, and each invariant of the system corresponds to a formula in $\mathcal{T}$. This correspondence is the core idea behind languages such as Ampersand [8], that define an information system this way. Users of an information system try to change the data set continually. These changes might violate the constraints. While Ampersand responds to such violations by rejecting the change, it would be convenient to automatically add data items such that all constraints are satisfied. The question then becomes: what data items should be added? We solve this question partially by means of a graph saturation procedure.

The question ‘does a set of formulas $\mathcal{T}$ have a model satisfying all formulas’ essentially asks whether $\mathcal{T}$ is free of contradictions. So far, we did not discuss the language in which we can write the formulas in $\mathcal{T}$. Several interesting problems arise when restricting the language in which we can write formulas: the satisfiability problem is obtained by restricting to disjunctions of positive and negative literals. Restricting to linear integer equalities, we obtain the linear programming problem. In this paper, we restrict those formulas to equalities over terms, in which terms are expressions of relations combined through the allegorical operations.[1] We define *sentence* to be a formula over the restricted language considered in this paper (Definition 3).

Our interest in this language stems from experience in describing systems in Ampersand. All operations from relation-algebra are part of the Ampersand language. The operations considered here include only the most frequently used subset of those operations. Therefore, many of the formulas used in Ampersand will be sentences as considered in this work. We therefore consider this work a step towards an Ampersand system that helps the user find models.

E-mail address: Sebastiaan.Joosten@utwente.nl.

[1] These are $⨟$, $\sqcap$, $\breve{\ }$ and $\mathbb{1}$. See the book by Freyd and Scedrov for details on allegories [4].

https://doi.org/10.1016/j.jlamp.2018.06.005
2352-2208/© 2018 Elsevier Inc. All rights reserved.

1.1. Approach

We give a short summary of the basic algorithm presented here, so we can better relate our approach to other literature, describe our contributions, and give the outline of this paper. Italicized words in the next paragraph are defined later.

The algorithm aims to determine whether there is a particular *model* for a set of *sentences*, say $\mathcal{T}$, and is guaranteed to terminate if no such *model* exists. It proceeds to construct a (possibly infinite) *model* otherwise. The procedure has two phases: first, we translate the *sentences* in $\mathcal{T}$ into a set of *graph rules*. We then apply a saturation procedure on the *graph rules*. This procedure creates a *chain* of *graphs*, whose limit is a *least consequence graph*. A *graph* contains a *conflict* if it has an edge with the label $\perp$. If a *least consequence graph* contains a *conflict*, then there is no *model* for $\mathcal{T}$. Otherwise, the *least consequence graph* corresponds to a *model* of $\mathcal{T}$, if the *graph rules* correspond to $\mathcal{T}$ according to a straightforward *translation*. We abort the procedure as soon as a *conflict* arises, because we can be sure that no *models* for $\mathcal{T}$ exist in this case. A second question we can answer through the same algorithm is that of *entailment*: *entailment* is the question whether a *sentence* ϕ follows from a set of *sentences* $\mathcal{T}$.

In an information system, a least consequence graph is a well suited to determine which data items to add: If conflict free, it corresponds to a graph that maintains the invariants. At the same, it only contains necessary consequences: it will not cause data items to be added that have nothing to do with the change the user made.

1.2. Related work

We compare the work in this paper to existing work in two ways: work it is similar to in motivation, and work it is similar to in implementation from an abstract perspective. In motivation, our research is closely related to the Alcoa tool, which we'll discuss first. In approach, our methods are related to description logics and to graph rewriting, which we'll discuss second.

The Alcoa tool Our search for a reasoner for Ampersand is related to Alcoa [6], which is the analyzer for Alloy [5], a language based on Z [13]. Like Ampersand, the languages Z and Alloy are based on relations. Alloy is a simplification of Z: it reduces the supported operations to a set that is small yet powerful. This paper differs from Alloy in the expressivity of its operations, however: Alloy allows writing full first order formula's plus the Kleene-star, making it a language that is even more expressive than Ampersand. We compare to Alcoa because this work is similar in purpose.

In Alloy, a user may write assertions, which are formulas that the user believes follow from the specification. Alcoa tries to find counterexamples to those assertions, as well as a finite model for the entire specification. Unfortunately, several properties of the Alcoa tool hinder our purposes in Ampersand: Alcoa requires an upper bound on the size of (or number of elements in) the model. It does not perform well if this bound is too large. In a typical information system, the amount of data is well above what can be considered 'too large'. As an additional complication, we cannot adequately predict the size of the model we might require. This is why we look at other methods for achieving similar goals.

Description logics We can regard our procedure as a way to derive facts from previously stated facts: this is what happens in terms of sentences between subsequent graphs in the chain we create. So called description logics are languages used in conjunction with an engine, that gives a procedure to learn new facts from previously learned facts, using declarative statements (or rules) in the corresponding description logic. For a good overview of description logics, see the book on that topic by Baader [1].

A set of derivation rules is consistent if it has a model. For a highly expressive description logic such as OWL DL, determining consistency is undecidable. Still, a rule engine for OWL DL will happily try to learn new facts until a model is found. Users of OWL DL typically need to ensure that the stated derivation rules together with the rule engine give a terminating procedure. For many description logics, termination of its rule engine is syntactically guaranteed, and these logics are consequentially decidable.

The description logic for which the language and implementation is closest to our language is the logic $\mathcal{EL}$ and its extensions proposed by Baader et al. [2,3]. Instead of using tableau-based procedures, as most description logics, it uses a saturation-based reasoner. Syntax of the derivation rules is limited to ensure termination of any saturation procedure: $\mathcal{EL}$ allows statements about unary relations using top, bottom, individual elements called 'nominal', and conjunction. Statements about binary relations use a different syntax, that can be translated into sentences using composition, converse and the identity relation (but not necessarily vice-versa). By modeling $\mathcal{EL}$'s unary relations as binary relations that are a subset of the identity relation, all of $\mathcal{EL}$ and its extensions can be expressed through the sentences described in this paper. In particular, the syntax of $\mathcal{EL}$ does not have disjunctions, thus eliminating the need for backtracking. In fact, $\mathcal{EL}$ is designed such that its consistency can be decided deterministically in polynomial time. Its extensions have different complexity bounds, but preserve polynomial runtime for the fragment that falls within $\mathcal{EL}$.

In our work, we do not work under the assumption of termination: neither the user or the syntax guarantees it. This allows us to use a richer language than one that is syntactically guaranteed to terminate. Despite this lack of termination, we do ensure termination in case of conflicts: a conflict will be found if our sentences imply it. This allows the user to approach certain problems through any set of rules within the grammar, rather than just those sets for which the