proceedings

MDPI

Proceedings

Minish HAT: A Tool for the Minimization of Here-and-There Logic Programs and Theories in Answer Set Programming †

Rodrigo Martin * and Pedro Cabalar

CITIC, University of Corunna, 15001 A Coruña, Spain; pedro.cabalar@udc.es

* Correspondence: r.martin1@udc.es

† Presented at the 2nd XoveTIC Conference, A Coruña, Spain, 5–6 September 2019.

Published: 31 July 2019

Abstract: When it comes to the writing of a new logic program or theory, it is of great importance to obtain a concise and minimal representation, for simplicity and ease of interpretation reasons. There are already a few methods and many tools, such as Karnaugh Maps or the Quine-McCluskey method, as well as their numerous software implementations, that solve this minimization problem in Boolean logic. This is not the case for *Here-and-There logic*, also called *three-valued logic*. Even though there are theoretical minimization methods for logic theories and programs, there aren't any published tools that are able to obtain a minimal equivalent logic program. In this paper we present the first version of a tool called that is able to efficiently obtain minimal and equivalent representations for any logic program in Here-and-There. The described tool uses an hybrid method both leveraging a modified version of the Quine-McCluskey algorithm and Answer Set Programming techniques to minimize fairly complex logic programs in a reduced time.

Keywords: logic minimization; knowledge representation; answer set programming; here-and-there logic

1. Introduction

In the field of *logic programming* it has always been f great importance to reduce the formulae and expressions to their minimal conjunctive or disjunctive normal forms (*CNF* and *DNF* respectively), as this reduces the number of logic OR and AND gates needed to implement the function as a circuit. This topic has been broadly studied for *Boolean Logic* and well known methods such as the Karnaugh Maps [1] and Quine-McCluskey algorithm [2,3] have served as base for powerful tools such as ESPRESSO [4] and BOOM [5]. Despite this being the case for Boolean Logic, it is not for Here-and-There logic (i.e., three-valued logic). There is a modified version of the Quine-McCluskey algorithm that takes in account the particularities of this specific logic [6] but there aren't any available tools implementing it. In this paper we describe a first approach to a tool that is able to minimize logic programs and theories in Here-and-There logic leveraging both the Quine-McCluskey algorithm alongside the power of Answer Set Programming.

2. Materials and Methods

The Quine-McCluskey algorithm aims to obtain a minimal normal form equivalent to any propositional theory. It can obtain either a minimal *DNF* from the set of models of the theory or a minimal *CNF* from its countermodels. The algorithm computes the set of prime implicates of a given theory given its countermodels. To obtain the minimal CNF we also need a coverage algorithm (Usually *Petrick's Method* [7]) to select the elements of the minimal subset of prime implicates that cover all of the initial countermodels of the theory.

In the logic of *here-and-there* (HT), a *formula* is defined in the usual way as a well-formed combination of the operators $\bot$,$\wedge$,$\vee$,$\rightarrow$ with atoms in a propositional signature At. We also define $\varphi \stackrel{\text{def}}{=} \varphi \rightarrow \bot$, and $\top \stackrel{\text{def}}{=} \bot$. A theory is a set of formulas.

An HT-interpretation $\langle H, T\rangle$ is a model of a given theory if satisfies all formulas in that theory. A formula true in all models is said to be valid or a tautology, while an *Equilibrium Model* of a theory is any total model ($H = T$) $\langle T, T\rangle$ of that theory such that no $\langle H, T\rangle$ with $H \subset T$ is model of the theory. *Equilibrium Logic* is the logic induced by equilibrium models.

As shown in [8], logic programs constitute a CNF for HT. A logic program is a conjunction of *clauses*, called *rules* with a positive and negative bodies and positive and negative heads in the form:

$$B_r^+ \wedge B_r^- \rightarrow Hd_r^+ \vee Hd_r^- \tag{1}$$

2.1. Implementation

Fundamental rules are rules in which all pairwise intersections of the head and body sets are empty with the possible exception of $Hd_r^+ \cap Hd_r^-$. Said rules can be translated to a minterm-like notation. Such rules can be transformed into minterm-like labels using a set of six symbols $\{0, 1, 2, \bar{2}, \bar{0}, -)\}$

These rules are then codified to octal and compared in a pairwise manner by using bit-by-bit operations, which allows us to perform the prime implicate generation taking in account the set of adjacent values for HT. This octal codification uses each one of the three bits for each symbol to describe which values are used (e.g., the symbol $\bar{2}$, meaning "not 2" having all bit positions set to 1 except the one corresponding to 2).

Value	**2**	**1**	**0**	**Value**	**2**	**1**	**0**
0	0	0	1	$\bar{2}$	0	1	1
1	0	1	0	$\bar{0}$	1	1	0
2	1	0	0	-	1	1	1

On top of the modified version of the Quine-McCluskey method, we add a previous step that allows us to work with the direct translation of the rules, instead of having to expand all of the possible minterms that each aggregated rule contains. Instead of comparing which labels are totally adjacent, we check that all positions are *compatible* except the adjacent one. This means that every position should subsum each other, or in the best case, be equal. Since we only expand the partially adjacent labels, in the worst case scenario it will be equal to expanding all of the labels into the possible minterms and in the average case will be logarithmically smaller. Finally, for the minimal coverage step, since this is a well studied case in ASP we leverage the ASP solver *clingo* to obtain the set of minimal versions of the original program, thus avoiding the implementation of the Petrick's method.

2.2. Tests

For a minimal program to be correct, it has to verify three conditions:

- **Size**: The minimal program has to be smaller, or at least equal in number of rules to the original program.
- **Subsumption**: The rules of the minimal program subsume all of the rules of the original program.
- **Strong Equivalence**: Both the original program and the minimal program have exactly the same models.

The size condition check is straightforward, only in the case that the number of rules remain the same (i.e., the original program is already minimal) we have to check that any rewriting of the original rules is valid regarding the other two conditions.

The subsumption condition is further detailed in [6] but to summarize, every rule of the minimal program has to subsume at least a rule of the original program and there can't be any rule of the original program that is not subsumed by the rules of the minimal program. This condition is checked with a program in ASP that compares both the original and the minimal programs to verify the condition by checking which parts of each rules of the latter are a subset of the ones of the original one.

For the strong equivalence condition, the objective is to transform both the original and minimal programs to classical logic by applying a set of transformations. By comparing the classical models obtained for the transformations of the original program and the minimal version we can ensure the strong equivalence property.

3. Conclusions

We have developed a novel tool that implements the modified version of Quine-McCluskey's method for HT, while also using the capabilities of Answer Set Programming to perform the minimal coverage of the initial countermodels, as well as using it to test and validate the results.

This first version of the tool is able to minimize the samples used in [6] in better times than the proof-of-concept Prolog script provided by the authors, since it doesn't rely on the calculation of all of the countermodels for a logic program, being able to work directly with the rules of the program as input by checking which rules can be potentially subsumed and only expanding those into minterms.

As for future work, we are already studying the *splitting properties* of logic programs in terms of minimization, which would allow us to separately minimize parts of a given logic program, greatly minimizing the number of atoms and rules that have to be dealt with at a single time in the current approach.

The currently *in-development* version of the scripts can be found at https://github.com/Trigork/minish-hat alongside examples and usage guidelines.

Funding: This research received no external funding.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Karnaugh, M. The map method for synthesis of combinational logic circuits. *Trans. Am. Inst. Electr. Eng. Part I* **1953**, *72*, 593–599.
2. Quine, W.V.O. The problem of simplifying truth functions. *Am. Math. Mon.* **1952**, *59*, 521–531.
3. McCluskey, E.J. Minimization of boolean functions. *Bell Syst. Tech. J.* **1956**, *35*, 1417–1444.
4. McGeer, P.; Sanghavi, J.V.; Brayton, R.K.; Sangiovanni-Vincentelli, A. Espresso-Signature: A New Exact Minimizer for Logic Functions. *IEEE Trans. VLSI Syst.* **1993**, *1*, 618–624.
5. Fiser, P.; Hlavička, J. BOOM, A Heuristic Boolean Minimizer. *Comput. Inform.* **2003**, *22*, 19–51.
6. Cabalar, P.; Pearce, D.; Valverde, A. Minimal Logic Programs. In *Logic Programming*; Dahl, V., Niemelä, I., Eds.; Springer Berlin Heidelberg: Berlin/Heidelberg, Germany, 2007; pp. 104–118.
7. Petrick, S.R. *A Direct Termination of the Irredundant Forms of a Boolean Functionfrom the Set of Prime Implicants*; Technical Report AFCRC-TR-56-110; Air ForceCambridge Res. Center: Cambridge, MA, USA, 1956.
8. Cabalar, P.; Ferraris, P. Propositional theories are strongly equivalent to logic programs. *Theory Pract. Logic Program.* **2007**, *7*, 745–759.