Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 213 (2008) 75–94
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

An Algorithm for Approximating the Satisfiability Problem of High-level Conditions

Karl-Heinz Pennemann[1,2]

Department of Computing Science
University of Oldenburg
D-26111 Oldenburg, Germany

Abstract

The satisfiability problem is the fundamental problem in proving the conflict-freeness of specifications, or in finding a counterexample for an invalid statement. In this paper, we present a non-deterministic, monotone algorithm for this undecidable problem on graphical conditions that is both correct and complete, but in general not guaranteed to terminate. For a fragment of high-level conditions, the algorithm terminates, hence it is able to decide. Instead of enumerating all possible objects of a category to approach the problem, the algorithm uses the input condition in a constructive way to progress towards a solution. To this aim, programs over transformation rules with external interfaces are considered. We use the framework of weak adhesive HLR categories. Consequently, the algorithm is applicable to a number of replacement capable structures, such as Petri-Nets, graphs or hypergraphs.

Keywords: first-order satisfiability problem, high-level conditions, high-level programs, graph transformation, weak adhesive HLR categories.

[1] This work is supported by the German Research Foundation (DFG), grants GRK 1076/1 (Graduate School on Trustworthy Software Systems) and HA 2936/2 (Development of Correct Graph Transformation Systems). Thanks to Arend Rensink for the visit to the University of Twente, during which the preliminary ideas for this work were discussed, to Annegret Habel for constructive remarks concerning this paper, and to the referees for their thorough reviews.

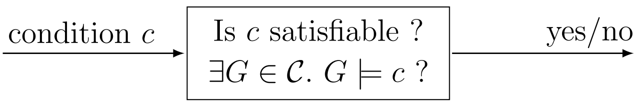
[2] Email: Pennemann@Informatik.Uni-Oldenburg.de

doi:10.1016/j.entcs.2008.04.075

1 Introduction

(High-level) Conditions are a graphical formalism to specify valid objects as well as morphisms, i.e., they can be used to describe system or program states as well as specify matches for transformation rules. They provide an intuitive formalism for structural properties and are well suited for reasoning about the behavior of transformation systems.

For a given category $\mathcal{C}$ of objects, the satisfiability problem is the problem to decide for any given condition c, whether or not $\exists G \in \mathcal{C}.\ G \models c$.



The satisfiability problem can be used to show that a specification is conflict free or to prove that a statement is invalid, i.e. if the negated statement is satisfiable. If some object G is provided along with a positive answer, one yields a counterexample for the latter case, illustrating an invalid system state. In this sense, a satisfiability algorithm complements a first-order theorem prover, with the prover searching for proof and the satisfiability algorithm looking for a counterexample. For the category **Graph** of finite, directed, labeled graphs, conditions are expressively equivalent to first order logic on graphs [20,11]. Therefore the satisfiability problem for arbitrary conditions over arbitrary categories is not decidable, i.e., there does not exist an algorithm that decides the satisfiability of arbitrary conditions over arbitrary categories. Still, an approximation of this undecidable problem is possible, but necessarily either unsound, incomplete or not guaranteed to terminate.

In this paper, we present a sound and complete algorithm that works for conditions over a class of replacement capable categories. Instead of enumerating all possible objects of a category to approach the problem, the presented algorithm uses the input condition in a constructive way. Starting from the initial object, e.g. the empty graph, elements of positive statements are added if necessary, while the absence of forbidden patterns is checked. The result is a monotone (non-deleting) algorithm which non-deterministically progresses towards a satisfiable object. Technically, we generate for each condition a program `SeekSat`. As we need to handover information between computation steps, `SeekSat` works on morphisms of the considered category. To this aim,