Artificial Intelligence 248 (2017) 123–157

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

From model checking to equilibrium checking: Reactive modules for rational verification

Julian Gutierrez, Paul Harrenstein, Michael Wooldridge *

Department of Computer Science, University of Oxford, United Kingdom

ARTICLE INFO

Article history:
Received 2 July 2015
Received in revised form 2 April 2017
Accepted 8 April 2017
Available online 12 April 2017

Keywords:
Complexity of equilibria
Reactive modules
Temporal logic

ABSTRACT

Model checking is the best-known and most successful approach to formally verifying that systems satisfy specifications, expressed as temporal logic formulae. In this article, we develop the theory of *equilibrium checking*, a related but distinct problem. Equilibrium checking is relevant for multi-agent systems in which system components (agents) are assumed to be acting rationally in pursuit of delegated goals, and is concerned with understanding what temporal properties hold of such systems under the assumption that agents select strategies in equilibrium. The formal framework we use to study this problem assumes agents are modelled using Reactive Modules, a system modelling language that is used in a range of practical model checking systems. Each agent (or *player*) in a Reactive Modules *game* is specified as a nondeterministic guarded command program, and each player's goal is specified with a temporal logic formula that the player desires to see satisfied. A strategy for a player in a Reactive Modules game defines how that player selects enabled guarded commands for execution over successive rounds of the game. For this general setting, we investigate games in which players have goals specified in Linear Temporal Logic (in which case it is assumed that players choose deterministic strategies) and in Computation Tree Logic (in which case players select nondeterministic strategies). For each of these cases, after formally defining the game setting, we characterise the complexity of a range of problems relating to Nash equilibria (e.g., the computation or the verification of existence of a Nash equilibrium or checking whether a given temporal formula is satisfied on some Nash equilibrium). We then go on to show how the model we present can be used to encode, for example, games in which the choices available to players are specified using STRIPS planning operators.

1. Introduction

Our main interest in this paper is in the analysis of concurrent systems composed of multiple non-deterministic computer programs, in which at run-time each program resolves its non-determinism rationally and strategically in pursuit of an individual goal, specified as a formula of temporal logic. Since the programs are assumed to be acting strategically, game theory provides a natural collection of analytical concepts for such systems [53]. If we apply game-theoretic analysis to such systems, then the main questions to be answered about such systems are not just "what computations might the system produce?", but rather, "what computations might the system produce if the constituent programs *act rationally*?" If we interpret acting rationally to mean choosing strategies (for resolving non-determinism) that are in Nash equilibrium, then

* Corresponding author.
E-mail address: mjw@cs.ox.ac.uk (M. Wooldridge).

http://dx.doi.org/10.1016/j.artint.2017.04.003
0004-3702/

this question amounts to asking "*which of the possible computations of the system will be produced in equilibrium*?" Further, if we use temporal logic as the language for expressing properties of our multi-agent and concurrent system (as is standard in the computer aided verification community [20]), then we can also interpret this question as "*which temporal logic formulae are satisfied by computations arising from the selection of strategies in equilibrium*?" We refer to this general problem as *equilibrium checking* [75].

Related questions have previously been considered within computer science and artificial intelligence – see e.g., [14,23, 29,30,51,13,8]. However, a common feature in this previous work is that the computational models used as the basis for analysis are highly abstract, and in particular are not directly based on real-world programming models or languages. For example, in [29] the authors define and investigate iterated Boolean games (iBG), a generalisation of Boolean games [36,37], in which each agent exercises unique control over a set of Boolean variables, and system execution proceeds in an infinite sequence of rounds, with each agent selecting a valuation for the variables under their control in each round. Each player has a goal, specified as a formula of Linear Temporal Logic (LTL), which it desires to see achieved. The iterated Boolean games model is simple and natural, and provides a compelling framework with which to pose questions relating to strategic multi-agent interaction in settings where agents have goals specified as logical formulae. However, this model is arguably rather abstract, and is some distance from realistic programming languages and system modelling languages; we discuss such work in more detail in the related work section towards the end of this article.

In brief, our main aim is to study a framework without these limitations. Specifically, we study game-like systems in which players are specified using (a subset of) the Reactive Modules language [2], which is widely used as a system modelling language in practical model checking systems such as mocha [4] and Prism [43]. Reactive Modules is intended to support the succinct, high-level specification of concurrent and multi-agent systems. As we will see, Reactive Modules can readily be used to encode other frameworks for modelling multi-agent systems (such as multi-agent STRIPS planning systems [10]).

The remainder of the article is structured as follows:

- We begin in the following section by motivating our work in detail, in particular by arguing that the classical notion of system correctness is of limited value in multi-agent systems, and introducing the idea of equilibrium checking as representing a more appropriate framework through which to understand the behaviour of such systems.
- We then survey the logics LTL and CTL, and their semantic basis on Kripke structures, present srml – a sublanguage of Reactive Modules that we use throughout the article – and then develop a formal semantics for it.
- We then introduce Reactive Modules games, in which the structure of the game (what we call the "arena") is specified using Reactive Modules, and the preferences of players are specified by associating a temporal (LTL or CTL) goal formula with each player, which defines runs or computation trees that would satisfy the player's goal.
- We then investigate the complexity of various game-theoretic questions in Reactive Modules games, for both the LTL and the CTL settings, and conclude by discussing the complexity and expressiveness of our new framework against the most relevant related work. Table 2 at the end of the paper summarises our findings.
- Finally, to demonstrate the wider applicability of our framework, we show how it can be used to capture propositional STRIPS games (cf. [22,12,25]), such as the MA-STRIPS model of Brafman and Domshlak [10].

Although largely self-contained, our technical presentation is necessarily terse, and readers may find it useful to have some familiarity with temporal logics [20,18], model checking [16], complexity theory [54], and basic concepts of non-cooperative game theory [53].

2. Motivation

Our aim in this section is to motivate and introduce the idea of equilibrium checking as a multi-agent systems counterpart to the standard notion of verification and model checking. (Many readers will be familiar with much of this material – we beg their indulgence so that we can tell the story in its entirety.)

Correctness and formal verification The *correctness problem* has been one of the most widely studied problems in computer science over the past fifty years, and remains a topic of fundamental concern to the present day [9]. Broadly speaking, the correctness problem is concerned with checking that computer systems behave as their designer intends. Probably the most important problem studied within the correctness domain is that of *formal verification*. Formal verification is the problem of checking that a given computer program or system P is correct with respect to a given formal (i.e., mathematical) specification φ. We understand φ as a description of system behaviours that the designer judges to be acceptable – a program that guarantees to generate a behaviour as described in φ is deemed to correctly implement the specification φ.

A key insight, due to Amir Pnueli, is that *temporal logic* can be a useful language with which to express formal specifications of system behaviour [56]. Pnueli proposed the use of Linear Temporal Logic (LTL) for expressing desirable properties of computations. LTL extends classical logic with tense operators **X** ("in the next state…"), **F** ("eventually…"), **G** ("always…"), and **U** ("…until …") [20]. For example, the requirement that a system never enters a "crash" state can naturally be expressed in LTL by a formula $\mathbf{G}\neg crash$. If we let $[\![P]\!]$ denote the set of all possible computations that may be produced by the program P, and let $[\![\varphi]\!]$ denote the set of state sequences that satisfy the LTL formula φ, then verification of LTL properties