Introduction to INFOMLSAI

Natasha Alechina Brian Logan

Utrecht University

n.a.alechina@uu.nl b.s.logan@uu.nl

Outline of this lecture

- introduction
- overview
- structure of the course

Introduction

Safety of AI systems

- focus of the course is on AI systems that make decisions
- we will view such systems as agents
- what is an agent ...

Characteristics of agents

Agents are often defined in terms of properties such as

- autonomy
- situatedness
- reactivity
- proactivity

Characterising agents

- many complex computational systems exhibit some degree of autonomy, situatedness, reactivity, and proactivity
- how to decide what's 'really' an agent?
- an alternative way to think of agents is not in terms of their behaviour but in terms of: how they are characterised

Agents as intentional systems

For the purposes of this course, an agent can be defined as:

Definition (Agent)

- a computational system whose behaviour can be usefully **characterised** in terms of propositional attitudes such as beliefs, goals and intentions

BDI agents

- this is sometimes termed the Belief-Desire-Intention (BDI) model of agency
- *beliefs* represent the agent's information about itself, the environment and other agents
- *goals* represent the state of the environment the agent is trying to bring about
- *intentions* are the future courses of action the agent is committed to carrying out

See Bratman et al Plans and resource-bounded practical reasoning. Computational Intelligence, 4(4), 1988, 349–355.

Applications of (multi-)agent systems

- autonomous vehicles, e.g., Google car, UAVs etc.
- workflow & logistics, e.g., warehouse management, maintenance scheduling
- real-time control & decision support (process monitoring, disaster management)
- healthcare, e.g., care coordination, patient monitoring
- power engineering, e.g., smart grids, VPPs
- information integration, e.g., sensor networks

Verification

- many applications of agents and multi-agent systems are safety critical
- for agent technology to be adopted in such areas, the public and regulators must be convinced that the (multi-)agent programs controlling autonomous systems are safe and reliable
- (formal) verification is a way of proving that a system meets its design requirements, e.g., safety properties
- verification differs from testing
  - testing usually considers only a *subset* of possible system behaviours (those induced by the tests)
  - verification considers *all possible* system behaviours

What is verification?

- in formal verification, we are given a property that should hold of a system, and a model of the system
- aim is to ensure that the system satisfies the property
- there are various ways of doing this—we will focus on:
  - **model checking:** the desired property is specified in logic and a model checking algorithm is used to check whether it holds for a model of the system
  - **synthesis:** automatically synthesise (provably) correct behaviours from a logical task specification

Verification and synthesis of AI systems

- verification is widely used in computer science for the automatic verification of hardware and software
- properties are expressed in a temporal logic such as LTL or CTL, and e.g., checked using a model checker such as SPIN
- AI systems can be verified and synthesised in this way
- however the particular characteristics of agents (focus on propositional attitudes) means that temporal epistemic logics and/or logics of strategic ability are often a better choice
- these logics are the focus of this course

For those who have seen verification before

- some of you may be familiar with automatic verification of hardware and software using temporal logic
- we will briefly recapitulate some of this material as it forms the foundation of the logics we consider
- however the focus of INFOMLSAI is logics and techniques specifically developed for the verification and synthesis of AI systems
- we will provide some optional, more advanced, exercises for those already familiar with LTL and CTL

Example: verification of the Autosub6000

- verification of the correctness and fault tolerance of the NOC Autosub6000 autonomous underwater vehicle using the MCMAS model checker

Image from ECO magazine

See, Ezekiel et al. Verifying Fault Tolerance and Self-Diagnosability of an Autonomous Underwater Vehicle. IJCAI 2011: 1659-1664

Example: verification of vehicle platooning

- verification of the correctness of autonomous vehicle platooning

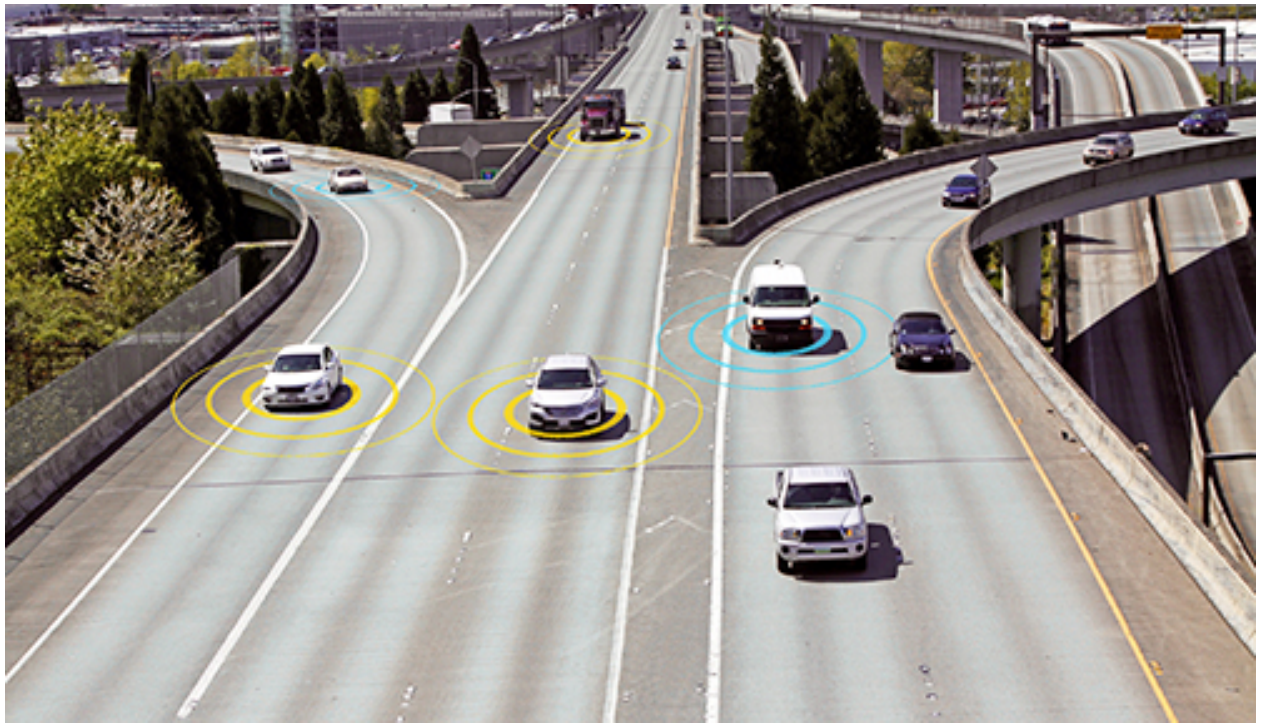

Image from US Department of Transportation Volpe Center

See, Kamali et al. Formal verification of autonomous vehicle platooning. Science of Computer Programming 2017: Vol 148, 88–106

Example: synthesis of plans

- synthesis of a plan to achieve a goal specified in temporal logic
- e.g., a goal for a healthcare robot might be ‘walk Mrs Smith to the bathroom, then get her a coffee, and if you meet the nurse on the way, tell her that I need her’

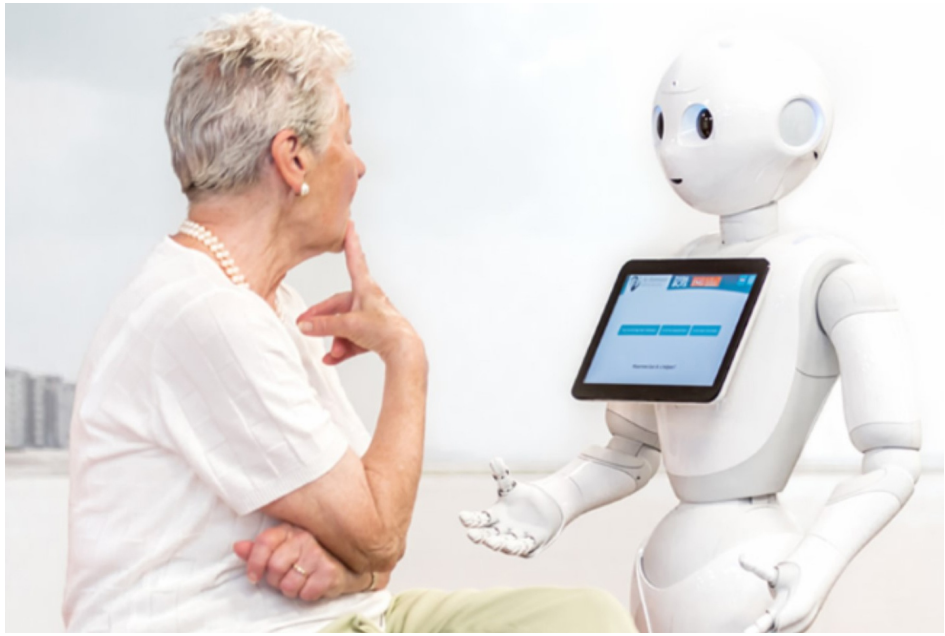

Image from AIJRF

See, Camacho et al. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. AAAI 2017: 3716–3724

Example: synthesis of process plans in manufacturing

- synthesis of a controller for a team of manufacturing robots from a *production recipe* specified in a formal language

See, e.g., Felli et al. Process Plan Controllers for Non-Deterministic Manufacturing Systems. IJCAI 2017: 1023-1030

Other approaches

- temporal logics are also useful in other approaches to safe AI
- for example, properties specified in temporal logic can be used
  - to constrain the behaviours learnt by a machine learning algorithm to be safe
  - to specify complex temporally-dependent rewards in reinforcement learning
- these techniques are not the main focus of the course, but will be covered in the exercises

Example: specifying reward functions

- generate a reward function for an RL agent from a temporal logic specification
- e.g., ‘bring gems to the shed, always avoid zombies’

See, e.g., Camacho et al. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. IJCAI 2019: 6065-6073

Overview

Overview

Week 1 Introduction and motivation. Temporal logic. What can be expressed in temporal logic.

Week 2 Temporal logic continued. Model checking for temporal logics. Introduction to MCMAS.

Week 3 Epistemic logic. Interpreted systems.

Week 4 Combined temporal and epistemic logic. Mock exam on weeks 1-4.

Overview continued

Week 5 Strategies in multi-agent systems. Coalition Logic.

Week 6 Alternating Time Temporal Logic (ATL).

Week 7 ATL model checking. Combining ATL with epistemic logic (ATEL).

Week 8 ATL synthesis.

Week 9 Belief, Desire, Intention (BDI) logics that combine temporal and epistemic logics and add intentions.

Structure of the course

Lectures

- due to the COVID-19 pandemic, lectures, practical sessions and assessments will be online
- first (this) lecture given live on MS Teams
- other lecture slots (Mondays 11:00–12:00 and Wednesdays 10:00-10:45) will be used for live Q&A sessions on MS Teams
- lectures for these sessions will be provided on Blackboard as recordings prior to the Q&A sessions
- please watch the recordings before the Q&A session!

Practicals

- practical sessions (Wednesdays 11:00–12:45) will be used to discuss coursework assignments, and will be live on MS Teams
- coursework assignments will be issued bi-weekly, and will cover both theoretical and practical aspects
- students are expected to work on coursework in (self-organised) groups of two to three people
- make sure at least one member of your group is comfortable with installing the model checker!

Learning materials

- the textbook for the course is *Logical Methods for the Specification and Verification of Multiagent Systems* by Wojtek Jamroga
- available as a free download at:

  https://home.ipipan.waw.pl/w.jamroga/papers/jamroga15specifmas-20200411.pdf

- for specific topics, we will also provide references to open source papers and software tools
- some of the slides are adapted from a course by Wojtek (thanks!)

MCMAS model checker

- the practical courseworks will use the MCMAS model checker
- MCMAS is specifically developed for the verification and synthesis of multi-agent systems
- stable, open source, and runs on Windows, MacOS and Linux
- a guide to installing MCMAS can be found on Blackboard
- please download MCMAS before the first practical!