Administrative Stuff

Even though the science is the most interesting part of this course, it is also a module that counts towards a degree at Swansea University. Therefore, some administrative issues arise. We'll try to keep it short and hopefully settle them once and for all in today's lecture.

Contact and Lecture Times

My name is Klaus Aehlig and, given that I'm teaching this module, I will obviously attend every lecture. So the best way to contact me is to meet me immediately after any lecture.

> Klaus Aehlig
> `k.t.aehlig@swan.ac.uk`
>
> Talbot 58

If you want to contact me between lectures (you're attending all of them anyway, aren't you?), feel free to come to my office which is room 58 in Talbot building.

Normally, I'm in my office every work day, not so early in the morning though, but quite late in the evening.

Outside office hours, the fastest way to reach me, is by sending an email. My email address in `k.t.aehlig@swan.ac.uk`.

As you have found out already, lectures are Tuesdays and Thursdays.

Tue	11	Far-H
Thu	12	Far-M

Assessment

Although you should attend the lectures because you're interested in the subject, experience shows that the first question of the students usually is that about examination. So here's the answer.

	CS-316	CS-M16
Coursework	20%	30%
Exam	80%	70%

Coursework:

- two assignments for all
- an extra assignment for CS-M16

The extra assignment for the CS-M16 students will be around the middle of the term. Whereas the regular assignments will consist of several questions in the same stile as typical exam questions, the extra assignment for CS-M16 will, most probably, ask to write a small essay or similar.

Work Attitude

Even though the main part of the assessment is the final exam, this does *not* mean that you can be lazy till a few weeks before the exam. Theoretical course are quite easy if *and only if* they are followed lecture by lecture. If, however, you fall behind, try to catch up as soon as possible. Once you've fallen behind too far, you won't be able to follow the lectures any more—and trying to learn a term's worth of lecture in two weeks before the exam turned out to be impossible for nearly all students.

In other words, revise what has been said after *each single lecture.* If there is anything you have not understood ask at the beginning of the next lecture! Chances are, you are not the only student with the same question.

Questions

As mentioned, it is important to keep up-to-date with every single lecture. So do ask questions (about the course, that is) as soon as they arise. There is no point in me rushing over the material with the audience unable to follow.

Course Material

This module has first been taught by Martin Otto in 2003 and the syllabus has not much changed since then; so his old lecture notes are still a good reference. Various text books and original articles will be pointed to during the lectures, where appropriate.

For those *and only those* lectures where I made written notes while preparing them, these notes will be handed out and made available on the course web site.

0 Introduction

Computer scientists often express certain properties. A specification is nothing but a (more or less) formal statement of some desirable property of a program. When working with specifications, one can ask several questions.

CS Questions Around Formalisms

$$\left.\begin{array}{l}\text{Does } \underline{\text{my}} \text{ program satisfy the spec?}\\ \text{Is there } \underline{\text{any}} \text{ program satisfying it?}\end{array}\right\}\text{How difficult to check?}$$

Can I express the property at all?
...and how complicated?

Of course, the answers to these questions depends on the formalism used to for the specification. More expressive formalisms tend to have harder model checking properties. Therefore it is reasonable to have several such formalisms available.

Moreover, non-expressibility can also be a feature. If we know that all properties expressible in our logic are invariant under certain kinds of changes of the model, we can allow for various changes of the implementation without having to do the verification again. For example, if our temporal logic never mentions "at the next time" all expressible properties are invariant under "stuttering", i.e., repeating states. In other words, efficiency doesn't matter for correctness.

$$\text{Expressiveness} \quad \longleftrightarrow \quad \begin{array}{l}\text{Easy to check}\\ \text{Invariant under changes}\end{array}$$

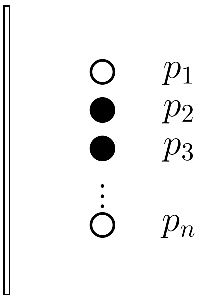
In this course, we will study several such logics. We will address some of the above questions for each logic. The emphasis will be on the particular properties and limitations of each logic itself, not on practically specifying programs.

To keep things simple, we will not specify properties of programs but of simple mathematical structures, like bit-vectors, (finite) strings of symbols, and directed graphs. We nevertheless should think of these structures as related to programs. Think of the states of a program (described by a finite set of bits!), the evolvement of states over time (modelled as discrete steps), and the possible transitions a program can make. With this intuition one can consider properties like "It is not the case that both processes are in the critical section", "*At no point in time* both processes are in the critical section", and "There is always a state the program can make a transition to".

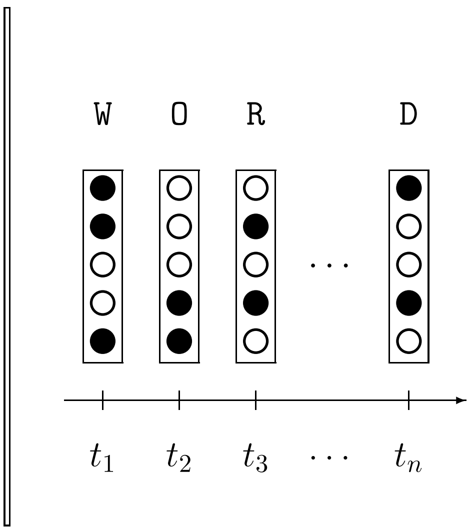
All the structures we're considering will be finite (in some way). This puts us in the realm of "finite model theory".

Let's finish this introduction with an overview over the logics we will consider in this course.

Propositional logic is the logic of individual states.



First-order logic and monadic second-order logic are used to describe sequences of states evolving over time.



Modal logic is used, if more then one possible evolution of a system is considered (under-specified behaviour, non-determinism, etc).