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	KH-430

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

Does my program satisfy the spec?
Is there any program satisfying it?
} 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.

Expressiveness $\longleftrightarrow$ Easy to check
Invariant under changes

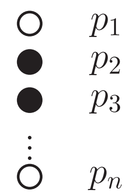
In this course, we will study several such logics. We will address some of the above questions for each logic. The emphasis will be on the particular properties and limitations of each logic itself, not on practically specifying programs.

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".

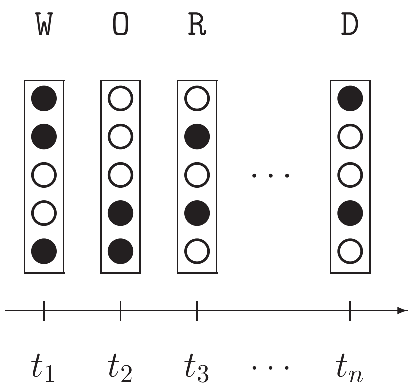
All the structures we're considering will be finite (in some way). This puts us in the realm of "finite model theory".

Let's finish this introduction with an overview over some of the most relevant logics from a specification point of view.

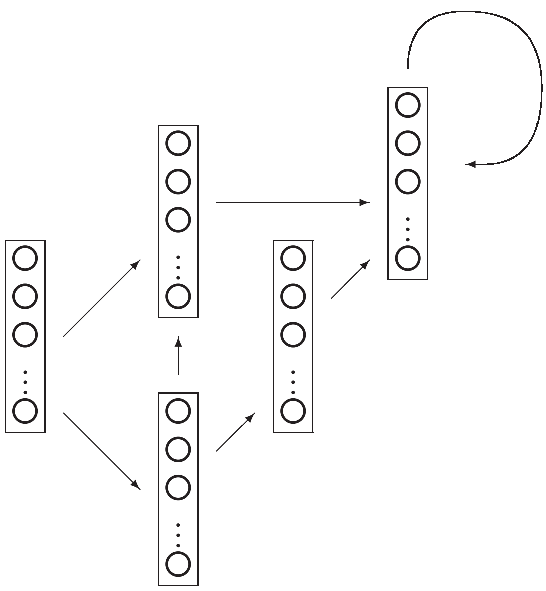
Propositional logic is the logic of individual states.



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

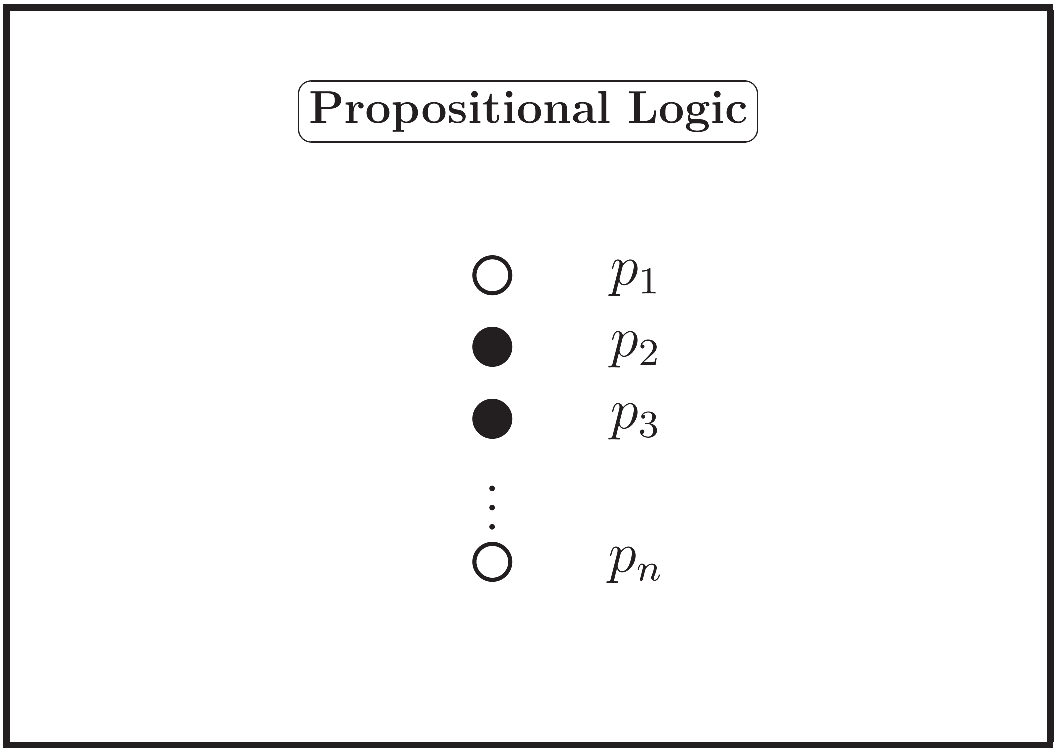


Modal logic is used, if more then one possible evolution of a system is considered (under-specified behaviour, non-determinism, etc).



1 Propositional Logic

Propositional logic is the logic of individual bits.



1.1 Syntax and Semantics

So, how do we describe such a situation? We can have a statement for every bit, saying that it is set. The rest is just composing statements, say, by "and" and "or"; we also allow $\top$ and $\bot$ as "logical constants", that is, as statements that are always true or always false, respectively.

In the following definition note the difference the language we use to speak *about formulae* and the actual syntax of the *formulae themselves*. The symbols "φ" and "ψ" are used to *denote* arbitrary formulae, but are not part of the syntax themselves (our formulae do not contain Greek letters). The symbols "(", ")", "$\wedge$", "$\vee$", "$\bot$", and "$\top$", however, *are* part of the syntax.

Syntax of Propositional Logic

The set $\mathsf{PL}[p_1, \ldots, p_n]$ of *propositional formulae over* $p_1, \ldots, p_n$ is freely generated as follows.

- $\top$, $\bot$, and all $p_i \in \{p_1, \ldots, p_n\}$ are propositional formulae *(so called "atomic formulae")*.
- If φ is a propositional formula, then so is $\neg\varphi$.
- If φ and ψ are propositional formulae, then so are $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$.

Here the "freely generated" (sometimes also called "inductively defined") means that the set $\mathsf{PL}[p_1, \ldots, p_n]$ is the *smallest* set with the given closure properties and, moreover, for every term we can read off—in a unique way!—in which way it is generated from these rules.

Note that we added enough parentheses so that for every string there is at most one way how it could have been generated by the above rules. This allows us to take these strings as formulae and still consider them freely generated.

Such definitions are quite common in many areas of logic and theoretical computer science. This definition principle gives us immediately a principle to define or proof something for *all* formulae. Note that to prove something for all formulae, it is sufficient to know that all formulae can be obtained by one of the rules; for defining a function on formulae, however, we also need that formulae are generated in a unique way.

Let's show this principle at a simple example. Our language $\mathsf{PL}[p_1, \ldots, p_n]$ of propositional logic is parametrised by the set $\{p_1, \ldots, p_n\}$ of allowed propositional variables. These symbols may be used in formulae of $\mathsf{PL}[p_1, \ldots, p_n]$. However, they do not have to be used. For $\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$, the set $\mathsf{fv}(\varphi)$ contains the variables that actually *are* used. We will see that only the value of those bit matter for a formula to be true or false.

$$
\begin{array}{lcl}
\mathsf{fv}(\top) = \mathsf{fv}(\bot) & = & \emptyset \\
\mathsf{fv}(p_i) & = & \{p_i\} \\
\mathsf{fv}(\neg\varphi) & = & \mathsf{fv}(\varphi) \\
\mathsf{fv}((\varphi \wedge \psi)) = \mathsf{fv}((\varphi \vee \psi)) & = & \mathsf{fv}(\varphi) \cup \mathsf{fv}(\psi)
\end{array}
$$

As mentioned, we regard propositional logic as the logic of bit strings. So our structures for $\mathsf{PL}[p_1, \ldots, p_n]$ are n-tuples $\underline{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$. The variable p_i is interpreted as a_i. We extend this (again, using the free generation of the formulae) to a value $\varphi[\underline{a}] \in \{0,1\}$ by the following truth table.

Evaluation of a Boolean Formula

For $\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$ and $\underline{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$ we define $\varphi[\underline{a}] \in \{0,1\}$ as follows.

- $\top[\underline{a}] = 1$, $\bot[\underline{a}] = 0$, $p_i[\underline{a}] = a_i$
- $(\neg\varphi)[\underline{a}] = \neg\!\!\neg(\varphi[\underline{a}])$
- $(\varphi \wedge \psi)[\underline{a}] = (\varphi[\underline{a}]) \mathbin{\wedge\!\!\!\wedge} (\psi[\underline{a}])$, $(\varphi \vee \psi)[\underline{a}] = (\varphi[\underline{a}]) \mathbin{\vee\!\!\!\vee} (\psi[\underline{a}])$

The functions $\neg\!\!\neg, \wedge\!\!\!\wedge, \vee\!\!\!\vee$ are given by

x	y	$\neg\!\!\neg y$	$x \wedge\!\!\!\wedge y$	$x \vee\!\!\!\vee y$
0	0	1	0	0
0	1	0	0	1
1	0		0	1
1	1		1	1

Note the difference between the *symbols* $\neg, \wedge, \vee$ and the *functions* $\neg\!\!\neg\colon \{0,1\} \to \{0,1\}$, $\wedge\!\!\!\wedge, \vee\!\!\!\vee\colon \{0,1\} \times \{0,1\} \to \{0,1\}$. However, from the context is usually clear whether a function or a name of a function is needed in some particular place. So we follow the usual laziness and often write $\neg, \wedge, \vee$ as a shorthand for $\neg\!\!\neg, \wedge\!\!\!\wedge, \vee\!\!\!\vee$. Nevertheless, you should always be able to tell, whether a symbol is meant, or whether that shorthand is used in a particular place.

Of course, when evaluating a formula, only those variables mentioned in the formula matter. The lemma expressing this fact is usually known as "Coincidence Lemma". We'll prove it as an exercise to see how one can argue on the inductive structure of an inductively defined object.

Lemma ("Coincidence Lemma").
$\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$, $\underline{\mathrm{a}} = (a_1, \ldots, a_n) \in \{0,1\}^n$, $\underline{\mathrm{b}} = (b_1, \ldots, b_n) \in \{0,1\}^n$.
Assume $a_i = b_i$ for all i s.t. $p_i \in \mathsf{fv}(\varphi)$.
Then $\varphi[a_1, \ldots, a_n] = \varphi[b_1, \ldots, b_n]$.

Proof.
If $\varphi = p_i$, then $p_i \in \mathsf{fv}(\varphi)$. Hence $\varphi[a_1, \ldots, a_n] = p_i[a_1, \ldots, a_n] = a_i = b_i = \varphi[b_1, \ldots, b_n]$.
If $\varphi = \psi \wedge \psi'$ then $\varphi[a_1, \ldots, a_n] = (\psi \wedge \psi')[a_1, \ldots, a_n] = \psi[a_1, \ldots, a_n] \wedge \wedge\psi[a_1, \ldots, a_n]' \stackrel{IH}{=} \psi[b_1, \ldots, b_n] \wedge \wedge\psi[b_1, \ldots, b_n]' = \varphi[b_1, \ldots, b_n]$.
$\ldots$

Logic is usually formulated in terms of the model relation $\models$ that says whether a formula holds in a particular structure. Even though our interpretation is formulated in terms of 0 and 1, it is obvious that we think of 1 as being true and of 0 as being false. This motivates writing $\underline{\mathrm{a}} \models \varphi$ for $\varphi[\underline{\mathrm{a}}] = 1$.

Model Relation for Propositional Logic

Writing $\underline{\mathrm{a}} \models \varphi$ for $\varphi[\underline{\mathrm{a}}] = 1$ we obtain the following.

- $\underline{\mathrm{a}} \models \top$ always holds and $\underline{\mathrm{a}} \models \bot$ never holds
- $\underline{\mathrm{a}} \models \neg\varphi$ holds iff $\underline{\mathrm{a}} \models \varphi$ does *not* hold
- $\underline{\mathrm{a}} \models \varphi \wedge \psi$ holds if $\underline{\mathrm{a}} \models \varphi$ *and* $\underline{\mathrm{a}} \models \psi$ both hold.
  $\underline{\mathrm{a}} \models \varphi \vee \psi$ holds if $\underline{\mathrm{a}} \models \varphi$ holds *or* $\underline{\mathrm{a}} \models \psi$ holds.

1.2 Boolean Functions

We have seen that propositional logic can be used to define functions $\{0,1\}^n \to \{0,1\}$. An obvious question arising is "Can we define all functions in that way?".

The answer is "Yes" and we will work towards the following Theorem.

Expressive Completeness

Theorem. For every $f\colon \{0,1\}^n \to \{0,1\}$ there is some $\varphi \in \mathsf{PL}[p_1,\dots,p_n]$ such that for all $\underline{\mathrm{a}} \in \{0,1\}^n$ we have $f(\underline{\mathrm{a}}) = \varphi[\underline{\mathrm{a}}]$.

To start our task, we build up things up from simpler tasks. How to represent that a variable has a particular value? How to build a function that is 1 at precisely one point $\underline{\mathrm{a}}$? Once this is achieved, we use that there are only finitely many $\underline{\mathrm{a}} \in \{0,1\}^n$; so we can just enumerate the positions where f is 1.

Proof.
Write x^1 for x and write x^0 for $\neg x$.
Then $\underline{a} \models x_i^\delta$ iff $a_i = \delta$.

Define
$$\chi_{\underline{a}} = \bigwedge_i x_i^{a_i}$$
Then $\underline{b} \models \chi_{\underline{a}}$ iff $\underline{b} = \underline{a}$.

Finally set
$$\varphi = \bigvee_{f(\underline{a})=1} \chi_{\underline{a}}$$

Carefully inspecting the proof, we note that we have proven even more.

Expressive Completeness

Theorem. For every $f\colon \{0,1\}^n \to \{0,1\}$ there is some $\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$ such that for all $\underline{a} \in \{0,1\}^n$ we have $f(\underline{a}) = \varphi[\underline{a}]$.

In fact, φ can be chosen to be of the form
$$\bigvee_j \bigwedge_i \xi_{ij} \qquad \text{with } \xi_{ij} \in \{x_i, \neg x_i\}$$

(in "disjunctive normal form")

In a similar way, we can show that every boolean function can be represented in conjunctive normal form. Working out the details will be (part of) your homework.

This completeness result also justifies why we didn't include more connectives in our language: having "and", "or" and "not" is enough. Nevertheless, there are quite a few abbreviations commonly used; the most important ones are "$\to$" and "$\leftrightarrow$".

$$\begin{aligned} \phi \to \psi &\equiv \neg\phi \vee \psi \\ \phi \leftrightarrow \psi &\equiv (\phi \to \psi) \wedge (\psi \to \phi) \end{aligned}$$

In fact, our choice of connectives is not minimal. A smaller number of connectives suffices to define all boolean functions.

Other Complete Sets of Connectives

- $\wedge$, $\neg$. *Indeed,* $x \vee y = \neg((\neg x) \wedge (\neg y))$.
- $\vee$, $\neg$
- `nand` where

x	y	x `nand` y
0	0	1
0	1	1
1	0	1
1	1	0

Indeed, $\neg x = x \texttt{ nand } x$ *and* $x \wedge y = \neg(x \texttt{ nand } y)$.

There are more complete set of connectives; one will be shown in your homework.

We have seen that we can express every boolean function by a propositional formula. Our construction yielded formulae in disjunctive normal form of size $\mathcal{O}(n \cdot 2^n)$. We now can ask, whether we can do better by using formulae not in disjunctive normal form. The answer is, that we cannot get rid of the exponential growths. More precisely, we have the following theorem.

On Succinctness

Theorem. Let $\varepsilon > 0$. For large n, the fraction of functions

$$\{0,1\}^n \to \{0,1\}$$

that can be represented by formulae of size up to

$$2^{(1-\varepsilon)\cdot n}$$

tends to zero.

In other words, almost all function require exponentially large formulae.

The reason is quite simple: there just not enough short formulae!

> **Proof.** Functions $\{0,1\}^n \to \{0,1\}$ have 2^n possible different inputs; hence
>
> $$2^{2^n}$$
>
> of them.
>
> ...

Note that for each input, we can independently set the output to 0 or 1. This gives

$$\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{2^n} = 2^{2^n}$$

possibilities.

> ...
>
> Formulae with n variables: $n + C$ symbols.
>
> ...

Note that, besides the variables $p_1, \ldots, p_n$, the only symbols we're using are $\wedge, \vee, \neg, (,)$. The result will not change (just take a bigger C) if we allowed a finite set of more connectives (like $\rightarrow, \leftrightarrow$).

> ...
> Hence $(n + C)^s$ formulae of size s. If $s = 2^{(1-\varepsilon)n}$ we get
>
> $$(n + C)^{2^{(1-\varepsilon)n}}$$
>
> of them.
> The fraction is
>
> $$\frac{(n + C)^{2^{(1-\varepsilon)n}}}{2^{2^n}} = 2^{2^{(1-\varepsilon)n} \cdot \log(n+C) - 2^n}$$
>
> which tends to 0, as
>
> $$2^n \gg 2^{(1-\varepsilon)n} \cdot \log(n + C)$$
>
> for large n.

1.3 Model-Checking, Satisfiability

Model checking propositional logic is easy; by the very definition of the semantics, the truth of a formula only depends on the truth of its subformulae.

Model Checking in Propositional Logic

Given: Propositional formula $\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$ and $\underline{\mathrm{a}} \in \{0, 1\}^n$

Question: $\underline{\mathrm{a}} \models \varphi$?

Solvable in polynomial time (essentially $\mathcal{O}(|\varphi|)$): just compute truth value following the buildup of φ.

Note the argument: the truth can be defined merely in terms of the truth values of smaller formulae—and there are only a small number of them. We will see similar "dynamic programming" arguments later in the course as well.

The satisfiability problem is (believed to be) much harder.

Satisfiability in Propositional Logic

Given: Propositional formula $\varphi \in \mathsf{PL}[p_1, \ldots, p_n]$.

Question: Is there some $\underline{a} \in \{0,1\}^n$ such that $\underline{a} \models \varphi$?

This problem is NP-complete.

Before we're going to prove this fact, let's explain first, what NP-complete means.

$\boxed{\mathsf{NP}}$

Definition. A problem is said to be in NP iff it can be solved by a non-deterministic Turing machine in polynomial time.
"NP is verifying proofs"

Conjecture. $\mathsf{P} \neq \mathsf{NP}$.

The conjecture $\mathsf{P} \neq \mathsf{NP}$ is believed by most computer scientists. Nevertheless, it is an open problem (in fact, one of the millennium problems) to prove or disprove it.

$\boxed{\mathsf{NP}\textbf{-completeness}}$

Definition. A problem L is NP-complete iff

- it belongs to NP
- for any problem L' in NP there is an easy (say, in polynomial time) function f such that

$$x \in L' \text{ iff } f(x) \in L$$

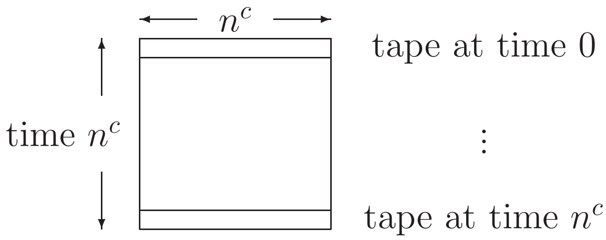
In other words, a problem is NP-complete if it is "the hardest" problem in NP. In particular, if one NP-complete problem can be solved in polynomial time, then $\mathsf{P} = \mathsf{NP}$.

Now, let's prove the stated complexity-result about the satisfiability problem.

Theorem. The satisfiability problem for propositional logic is NP-complete.

Proof Sketch.
"in NP": just guess $\underline{a}$. Then verify as in the model-checking problem.

"NP-hard": given a TM deciding L in time n^c, it uses at most that space. So its runs can be described by an $n^c \times n^c$ table.

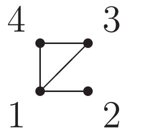


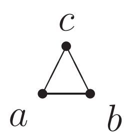
Introducing variables for "at time t the symbol at position i is a" and "at time t the head is at position i" leads to local conditions on this scheme, expressible in propositional logic.

Examples related to Propositional Logic

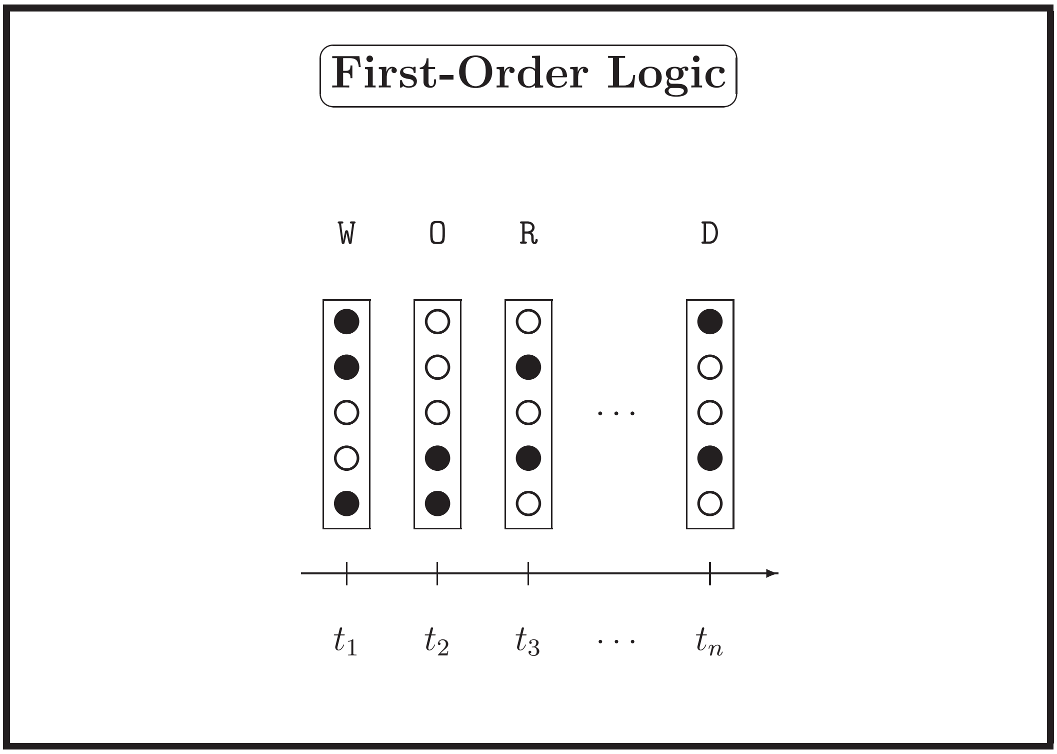
Example

Let $G = (\{1, 2, 3, 4\}, E)$ be an undirected graph. It can be described by formulae in $\mathsf{PL}[p_{12}, p_{13}, p_{14}, p_{23}, p_{24}, p_{34}]$ where p_{ij} expresses the fact that there is an edge from i to j.

(a) Express: The graph is 

(b) Express: The graph contains some triangle. 

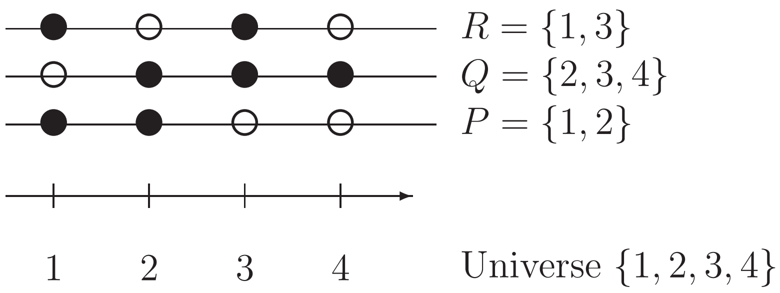
2 First-Order and Second-Order Logics of Strings



2.1 Trace Structures and Word Structures

So far, we considered propositional logic as the logic of individual bits. This was a completely static view. Now imagine a scenario where this changes over time.

Finite Trace Structures



$$< = \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\}$$

So, relations immediately come into play, once we consider a time component. The structures usually associated with such a setting are first-order structures.

First-Order Structures

Definition. A first-order structure $\mathcal{A} = (A, R_1, R_2, \ldots)$ is given by

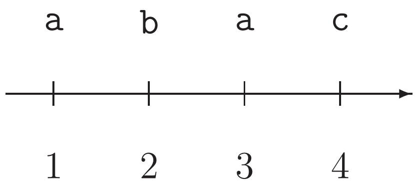
- A non-empty set A, called the "universe" of the structure
- Relations $R_1, R_2, \ldots$ on A.
  I.e., each $R_i \subseteq A^{n_i}$ for some n_i, called the arity.

In this course, we will mainly be looking at finite structures, that is, structures where the universe is a finite set. Nevertheless, infinite first-order structures play an important role in mathematics; in fact, most parts of logic are mainly concerned with infinite structures.

For the moment, we will not use that framework of first-order logic in its full generality. Instead, for the time being, we restrict ourself to finite trace and words.

Word Structures

$P_{\mathtt{a}} = \{1, 3\}$, $P_{\mathtt{b}} = \{2\}$, $P_{\mathtt{c}} = \{4\}$



Universe $\{1, 2, 3, 4\}$

$< = \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\}$

Trace Structures and Word Structures

The universe is an initial segment of natural numbers, i.e., $A = \{1, 2, \ldots, \ell\}$ and $<$ the usual order on them.

- finite trace structures are given by arbitrary unary relations P_1, P_2, ...
- word structures are given by unary relations P_a for $a \in \Sigma$ *partitioning* the univserse

Note.

Trace structures are in 1-1, onto, correspondence with word structures over $\{0, 1\}^n$

Word structures are in 1-1, onto, correspondence with words.

The last observation allows to consider whatever logic we invent for a word-structures from a point of view of formal languages: "What are the languages, i.e., sets of strings, that can be specified by this or that logic?"

2.2 Linear Temporal Logic

Even though, technically, the structures we are considering are first-order structures, we start by considering a language specially tailored to express the temporal aspect.

Syntax of Linear Temporal Logic

The set $\mathsf{LTL}[p_1, \ldots, p_n]$ of *LTL-formulae* is freely generated

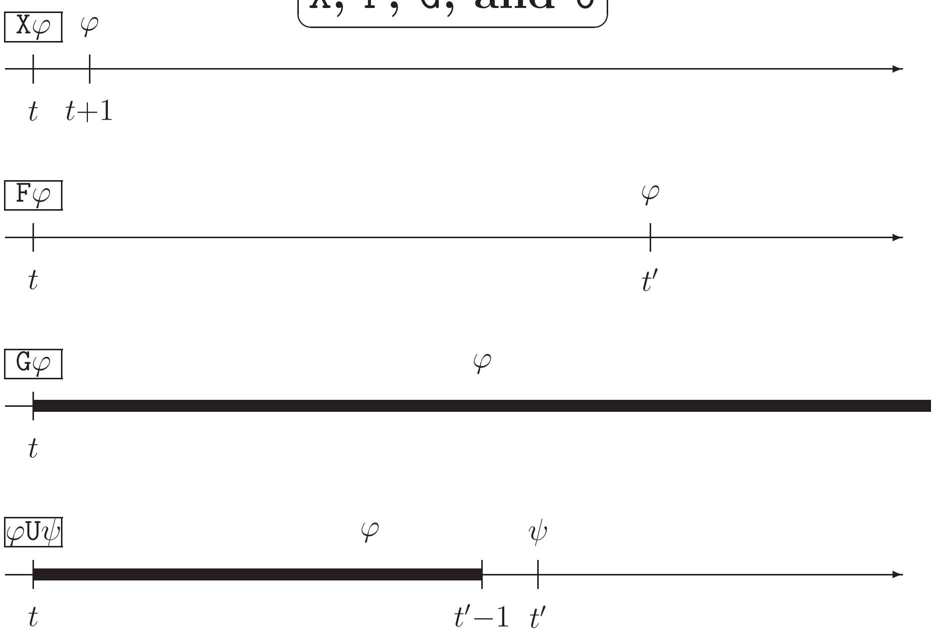
- $\top$, $\bot$, and all $p_i \in \{p_1, \ldots, p_n\}$
- If $\varphi, \psi \in \mathsf{LTL}[p_1, \ldots, p_n]$, then also $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$.
- ... and also
  - $\mathtt{X}\varphi$ "next"
  - $\mathtt{F}\varphi$ "finally"
  - $\mathtt{G}\varphi$ "globally"
  - $(\varphi\mathtt{U}\psi)$ "until"

Again, all the same comments about freely generated sets apply as for propositional logic. We just abbreviated this statement slightly on the slide. As usual, we omit unneeded parentheses when writing down LTL-formulae.

The formula $\mathtt{F}\varphi$ is often also pronounced as "eventually φ" and the formula $\mathtt{G}\varphi$ as "always φ".

Note that $\mathsf{LTL}[p_1, \ldots, p_n]$ properly extends the syntax of propositional logic.

X, F, G, and U



Semantics of LTL

Let $\mathcal{A} = (\{1, \ldots, \ell\}, <, P_1, P_2, \ldots, P_n)$ and $t \in \{1, \ldots, \ell\}$. Define $\mathcal{A}, t \models \varphi$ for $\varphi \in \mathsf{LTL}[p_1, \ldots, p_2]$ inductively.

- $\mathcal{A}, t \models \top$; $\mathcal{A}, t \not\models \bot$; $\mathcal{A}, t \models p_i$ iff $t \in P_i$
- $\mathcal{A}, t \models \varphi \wedge \psi$ iff ...
- $\mathcal{A}, t \models \mathtt{X}\varphi$ iff $t < \ell$ and $\mathcal{A}, t{+}1 \models \varphi$
- $\mathcal{A}, t \models \mathtt{F}\varphi$ iff $\mathcal{A}, t' \models \varphi$ for some $t' \geq t$
- $\mathcal{A}, t \models \mathtt{G}\varphi$ iff $\mathcal{A}, t' \models \varphi$ for all $t' \geq t$
- $\mathcal{A}, t \models \varphi\mathtt{U}\psi$ iff there is some $t' \geq t$ s.t.
  $\mathcal{A}, t' \models \psi$ and $\mathcal{A}, t'' \models \varphi$ for all $t \leq t'' < t'$

In the previous slide we just abbreviated the model relation for the propositional connectives, as they are unchanged. Recall the slide from an earlier lecture.

Model Relation for Propositional Logic

Writing $\underline{a} \models \varphi$ for $\varphi[\underline{a}] = 1$ we obtain the following.

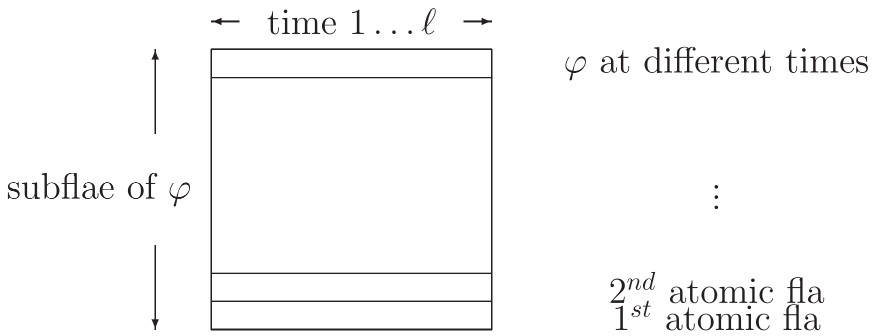
- $\underline{a} \models \top$ always holds and $\underline{a} \models \bot$ never holds
- $\underline{a} \models \neg\varphi$ holds iff $\underline{a} \models \varphi$ does *not* hold
- $\underline{a} \models \varphi \wedge \psi$ holds if $\underline{a} \models \varphi$ *and* $\underline{a} \models \psi$ both hold.
  $\underline{a} \models \varphi \vee \psi$ holds if $\underline{a} \models \varphi$ holds *or* $\underline{a} \models \psi$ holds.

Recall the model-checking problem, which, for LTL reads as follows.

> **Given:** $\varphi \in \mathsf{LTL}[p_1, \ldots, p_2]$ and $\mathcal{A} = (\{1, \ldots, \ell\}, <, P_1, P_2, \ldots, P_n)$ and $1 \leq t \leq \ell$
>
> **Question:** Does $\mathcal{A}, t \models \varphi$?

For propositional logic, we observed that the truth of $\underline{a} \models \varphi$ only depends on the truth of $\underline{a} \models \varphi'$ for subformulae φ' of φ. Here we can do a similar observation: the truth of $\mathcal{A}, t \models \varphi$ only depends on that of $\mathcal{A}, t' \models \varphi'$ for φ' a subformulae of φ.

Hence, a similar dynamic programming approach will work.

> **Answer:** Polynomial in $|\varphi|$ and $|\mathcal{A}|$.
>
> 
>

We can do even better than that! Looking again at the definition of the $\mathcal{A}, t \models \varphi$ relation, we note that we can reformulate it in such a way, that truth only depends on the truth of proper subformulae at the same time, or a (non-proper) subformula at the next point in time.

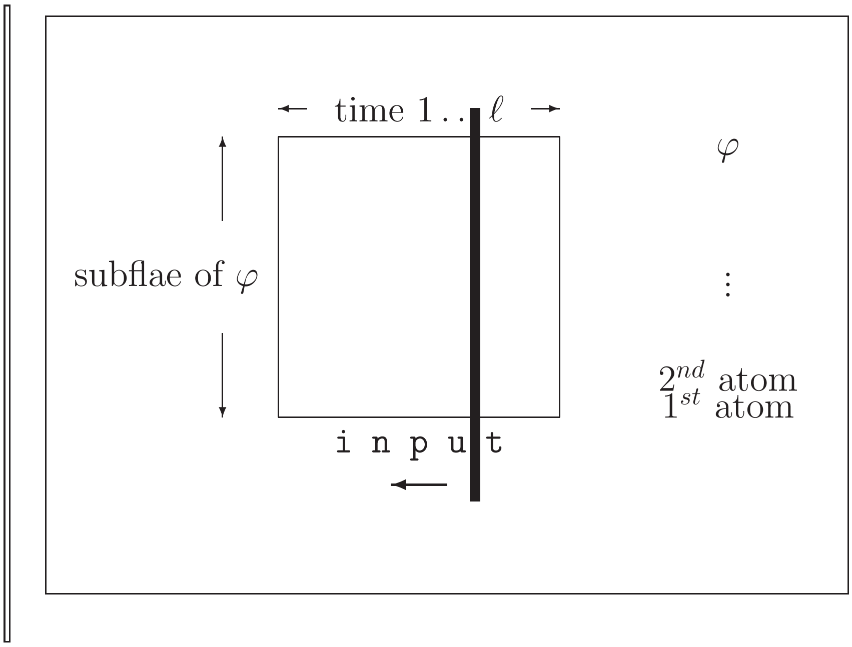
LTL Model Checking

t		t		$t{+}1$
$\mathcal{A}, t \models \mathtt{X}\varphi$	iff			$\mathcal{A}, t{+}1 \models \varphi$
$\mathcal{A}, t \models \mathtt{F}\varphi$	iff	$\mathcal{A}, t \models \varphi$	or	$\mathcal{A}, t{+}1 \models \mathtt{F}\varphi$
$\mathcal{A}, t \models \mathtt{G}\varphi$	iff	$\mathcal{A}, t \models \varphi$	and	$\mathcal{A}, t{+}1 \models \mathtt{G}\varphi$
				in case $t + 1 \leq \ell$
$\mathcal{A}, t \models \varphi \mathtt{U} \psi$	iff	$\mathcal{A}, t \models \psi$	or	
		$\mathcal{A}, t \models \varphi$	and	$\mathcal{A}, t{+}1 \models \varphi \mathtt{U} \psi$

On the one hand, the above proof now gives the better time complexity, since the amount of lookup for any particular filed is now constant.

Time $\mathcal{O}(|\mathcal{A}| \cdot |\varphi|)$

Moreover, the amount of information we have to store only depends on the size of the formula. We can just sweep through the word/trace and update the information about all subformulae.



Let's see this form of model checking in an example.

Example for LTL Model Checking

Formula $\mathtt{G}(y \rightarrow y\mathtt{U}r)$. *That this* $\mathtt{G}(\neg y \vee y\mathtt{U}r)$.

$\mathcal{A}$

r	1	1	1	0	0	0	0	0	1
y	0	0	1	0	0	0	1	1	0
g	0	0	0	1	1	1	0	0	0
$\mathtt{G}(y \rightarrow y\mathtt{U}r)$									
$y \rightarrow y\mathtt{U}r$									
$y\mathtt{U}r$									
$\neg y$									

Note that, when working our way from the end of the trace to its beginning, we could, at any point, forget any future columns after the earliest one we filled.

This is precisely the characteristic behaviour of an automaton walking (backwards) over the word.

LTL and Automata

For φ construct DFA $\mathcal{M}_\varphi$ with $L(\mathcal{M}_\varphi) = \{w | w^{-1} \models \varphi\}$.

States: sets of sub-formulae of φ.
indicating with formulae hold at a given position

Transitions: Given

- previous state $\{\psi \mid \mathcal{A}, t{+}1 \models \psi\}$
- t'th letter of w, i.e., local properties of $\mathcal{A}$ at time t

determine $\{\psi \mid \mathcal{A}, t \models \psi\}$ by the rules seen.

This description by an automata has several important consequence. First of all, we get decidability of the satisfiability problem. This was not a priory clear. As opposed to propositional logic, we didn't have an a priory bound on the candidate structures (as words can have arbitrary length!).

Theorem. Satisfiability for an LTL-formula φ over finite words can be decided in *space* polynomial $|\varphi|$.

Proof. Decide the emptiness-problem for the automaton associated with φ.
(Since a state is described by $|\varphi|$ many bits, we are in non-deterministic space $\mathcal{O}(|\varphi|)$, which is in deterministic space $\mathcal{O}(|\varphi|^2)$.)

Moreover, the fact that the number of states of the automaton has "only" $2^{|\varphi|}$ states gives us a bound on the size of the smallest structure satisfying an LTL property.

Corollary. If an LTL-formula φ can be satisfied by a finite structure, then it can also be satisfied by a structure of size at most $2^{|\varphi|}$.

Also note that from a DFA for w^{-1} we can get directly an NFA for w (and therefore also an DFA at the price of an exponential increase in the number of states).

All LTL-definable properties are regular.

In particular, we have found a bound on the expressibility of LTL. Just take your favourite non-regular language and LTL cannot express it.

There is no LTL-formula expressing "the word looks like $\underbrace{\mathtt{aa}\ldots\mathtt{a}}_{n}\underbrace{\mathtt{bb}\ldots\mathtt{b}}_{n}$"

As it turns out, this bound is not sharp. There are regular languages that *cannot* be expressed using an LTL formula. However, proving this fact is beyond the scope of this course.

Examples related to Linear Temporal Logic

Example

For each of the following formulae, decide whether they hold at the first letter of the given words!

- a and $\mathtt{X}a$

  `baab` `abc` `aaa` `a`

- $\mathtt{F}a$

  `bbbbbba` `ba` `a`

- $\mathtt{G}a$

  `baaaaaaaaaaaaa` `aaa` `a` `aaaaaab`

- $a\mathtt{U}b$

  `aaaaab` `abc` `bcaaab` `b`

Example

Consider the language with the predicates r_1, r_2, g_1, g_2 with the interpretation that r_1 and r_2 express that process 1 and 2, respectively, are requesting access to a shared resource, and g_1 and g_2 express that access to the shared resource is granted for process 1 and 2.

Formalise the following statements.

- "No two requests are granted at the same time."
- "Every request will eventually be granted."
- "Every request by process 1 will be granted in the next round."

Example

Consider a traffic light. In our formalisation, we will use the variables r, y, g for the events the red/yellow/green light is on.

Formalise the following events.

- "There is always at least one light on"
- "It is always the case, that you will get a green light sometimes"
- "Whenever there's a yellow light, it will stay till a red light shows up"

2.3 First-Order Logic

As mentioned, the structures we have been looking at are first-order structures. The language usually used to describe them is first-order logic. We'll see that first-order logic can, without a significant overhead, express everything we could express in linear temporal logic. However, the complexity of model checking and the satisfiability problem will turn out to be much harder for first-order logic.

Recall the general setting of the structures we're talking about.

First-Order Structures

Definition. A first-order structure $\mathcal{A} = (A, R_1, R_2, \ldots)$ is given by

- A non-empty set A, called the "universe" of the structure
- Relations $R_1, R_2, \ldots$ on A.
  I.e., each $R_i \subseteq A^{n_i}$ for some n_i, called the arity.

So far, the elements of the universe were always handled implicitly. So, we didn't have to have names for them. Now we assume some fixed infinite set

$$V = \{x_1, x_2, \ldots\}$$

of first order variables. Our notion of (first-order) formula will implicitly refer to this set.

We will use $x, y, z, \ldots$ to denote variables.

Syntax of First-Order Logic

The set $\mathsf{FOL}[R_1, \ldots, R_n]$ of first-order formulae over the relation symbols $R_1, \ldots, R_n$ is freely generated as follows.

- $(x = y)$ for variables x, y
- $R_j x_{i_1} \ldots x_{i_{n_j}}$ if R_j is of arity n_j
- $\top$, $\bot$, $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$ for formulae ϕ, ψ
- $\forall x \varphi$ and $\exists x \varphi$ for φ a formula and x a variable

Intuitively, we consider the variable x to be bound in $\forall x \varphi$. This intuition is reflected in our definition of the set of "free variables" of a formula.

$$\begin{array}{lcl}
\mathsf{fv}(x = y) & = & \{x, y\} \\
\mathsf{fv}(R y_1 \ldots y_k) & = & \{y_1, \ldots, y_k\} \\
\mathsf{fv}(\top) = \mathsf{fv}(\bot) & = & \emptyset \\
\mathsf{fv}(\neg\varphi) & = & \mathsf{fv}(\varphi) \\
\mathsf{fv}((\varphi \wedge \psi)) = \mathsf{fv}((\varphi \vee \psi)) & = & \mathsf{fv}(\varphi) \cup \mathsf{fv}(\psi) \\
\mathsf{fv}(\forall x \varphi) = \mathsf{fv}(\exists x \varphi) & = & \mathsf{fv}(\varphi) \setminus \{x\}
\end{array}$$

To assign a meaning to first-order formulae, we do not only have to know, what the relation symbols stand for, but we also need to know, how the variables are to be interpreted—just as we had to know the meaning of "now" in LTL.

To do so, we use a mapping η from the variables to the universe. We also need the notion of a mapping modified at one particular place

> For $\eta \colon V \to A$, $x \in V$, $a \in A$ we set
>
> $$\eta_x^a(y) = \begin{cases} \eta(y) & x \not\equiv y \\ a & x \equiv y \end{cases}$$

Semantics of First-Order Logic

Let $\mathcal{A} = (A, R_1^{\mathcal{A}}, \ldots)$ and $\eta \colon V \to A$.
Define $\mathcal{A}, \eta \models \varphi$ for $\varphi \in \mathsf{FOL}[R_1, \ldots]$ inductively.

- $\mathcal{A}, \eta \models x = y$ iff $\eta(x) = \eta(y)$
- $\mathcal{A}, \eta \models R_1 y_1 \ldots y_n$ iff $(\eta(y_1), \ldots, \eta(y_n)) \in R_i^{\mathcal{A}}$
- $\mathcal{A}, \eta \models \varphi \wedge \psi$ iff ...
- $\mathcal{A}, \eta \models \forall x \varphi$ iff for all $a \in A$ we have $\mathcal{A}, \eta_x^a \models \varphi$
- $\mathcal{A}, \eta \models \exists x \varphi$ iff for some $a \in A$ we have $\mathcal{A}, \eta_x^a \models \varphi$

Again, we have a coincidence Lemma.

> **Lemma.**
> η, η' such that $\eta(x) = \eta'(x)$ for all $x \in \mathsf{fv}(\varphi)$.
> Then $\mathcal{A}, \eta \models \varphi$ iff $\mathcal{A}, \eta' \models \varphi$.
>
> **Proof.** Induction on φ.
>
> ...

Most cases are straight forward. The only non-trivial case is that of quantifiers.

> ...
>
> Case $\varphi \equiv \forall x \psi$. Then $\mathsf{fv}(\psi) \subseteq \mathsf{fv}(\forall x \psi) \cup \{x\}$. Moreover, for all $a \in A$ and $y \in \mathsf{fv}(\forall x \psi) \cup \{x\}$ we have $\eta^a_x(y) = {\eta'}^a_x(y)$.
> Hence, by IH,
> $\mathcal{A}, \eta \models \forall x \psi$ iff
> for all $a \in A$, $\mathcal{A}\eta^a_x \models \psi$ iff
> for all $a \in A$, $\mathcal{A}{\eta'}^a_x \models \psi$ iff
> $\mathcal{A}, \eta' \models \forall x \psi$
>
> ...

The case of $\exists x \phi$ is handled similarly.
The Coincidence Lemma justifies the following

Notation. If $\mathsf{fv}(\varphi) = \{x_1, \ldots, x_n\}$ with $x_1, \ldots, x_n$ understood, we write

$$\mathcal{A}, a_1, \ldots, a_n \models \varphi$$

if for some (and hence any) η with $\eta(x_i) = a_i$ for all i,

$$\mathcal{A}, \eta \models \varphi$$

In particular, we just write

$$\mathcal{A} \models \varphi$$

if $\mathsf{fv}(\varphi) = \emptyset$.

As usual, we use some abbreviations. Especially when one relation symbol is $<$ and is thought of as some linear order, quite a few abbreviations are commonly in use.

$<$ is usually written infix, i.e. $x < y$ denotes $< xy$.

Moreover,

$$\begin{array}{lcl} x \leq y & \equiv & (x < y \vee x = y) \\ x > y & \equiv & y < x \\ x \geq y & \equiv & y \leq x \\ \ldots & & \end{array}$$

Over discrete linear orders, we can do even more. Remember that LTL's `X`-modality required us to define "the next point in time". This concept can be described as "in the future, but between now and that point there is nothing else".

$$(\{1, \ldots, t\}, <), t, t' \models (x < x' \wedge \neg \exists y (x < y \wedge y < x'))$$

iff

$$t' = t + 1$$

This observation motivates us to abbreviate the above formula by $\chi_{\text{next}}(x, y)$.

We can express the semantics of LTL as a first-order formula speaking about a particular point in time. So we translate a LTL formulae φ into first-order formulae $\tilde{\varphi}(t)$ with one distinguished variable t.

LTL and First-Order Logic

$\varphi \in \mathsf{LTL}[p_1, \ldots, p_n]$	$\tilde{\varphi}(t) \in \mathsf{FOL}[<, P_1, \ldots, P_n]$
p_i	$P_i(t)$
$\mathtt{X}\varphi$	$\exists t'(\chi_{\text{next}}(t, t') \wedge \tilde{\varphi}(t'))$
$\mathtt{F}\varphi$	$\exists t'(t \leq t' \wedge \tilde{\varphi}(t'))$
$\mathtt{G}\varphi$	$\forall t'(t \leq t' \wedge \tilde{\varphi}(t'))$
$\varphi \mathtt{U} \psi$	$\exists t'(\ t \leq t' \wedge \tilde{\psi}(t') \wedge$
	$\forall t''(((t \leq t'') \wedge (t'' < t')) \rightarrow \tilde{\varphi}(t))\ )$

With this translation we get that

$$\mathcal{A}, t \models \varphi \text{ iff } \mathcal{A}, t \models \tilde{\varphi}$$

Note, however, that we couldn't have chosen the trivial translation $\tilde{\varphi} \equiv \varphi$, was we wanted a translation from LTL to first-order logic. Note the different meanings of the $\models$ symbols above!

In propositional logic, we have seen that every property can be expressed by a formula where the negation sign $\neg$ is only used in front of propositional atoms ($\top$,$\bot$,p). A similar observation holds true for first-order logic. This time we're not arguing semantically (we don't have an overview over the expressible properties yet!) but syntactically. A similar syntactical argument would have also been possible for propositional logic.

Negation Normal Form

Lemma.
$\mathcal{A}, \eta \models \neg\forall x\varphi$ iff $\mathcal{A}, \eta \models \exists x\neg\varphi$
$\mathcal{A}, \eta \models \neg\exists x\varphi$ iff $\mathcal{A}, \eta \models \forall x\neg\varphi$

Recall from propositional logic.
$\mathcal{A}, \eta \models \neg(\varphi \wedge \psi)$ iff $\mathcal{A}, \eta \models (\neg\varphi) \vee (\neg\psi)$
$\mathcal{A}, \eta \models \neg(\varphi \vee \psi)$ iff $\mathcal{A}, \eta \models (\neg\varphi) \wedge (\neg\psi)$
$\mathcal{A}, \eta \models \neg\neg\varphi$ iff $\mathcal{A}, \eta \models \varphi$

So we can push all the $\neg$ symbols inwards, till they're only in front of atomic formulae. In fact, inspecting this transformation in more detail, we note that the size of the formula at most doubles; in the worst case, every atomic formula gets a negation in front.

The new concept in first-order logic are quantifiers. Also for them, we can find a canonical place in the formula: the beginning. For our argument, we can already assume the formula to be in negation normal form.

Prenex Normal Form

Lemma. Assume $x \notin \mathsf{fv}(\psi)$.
$\mathcal{A}, \eta \models (\forall x \varphi) \land \psi$ iff $\mathcal{A}, \eta \models \forall x(\varphi \land \psi)$
$\mathcal{A}, \eta \models (\forall x \varphi) \lor \psi$ iff $\mathcal{A}, \eta \models \forall x(\varphi \lor \psi)$
$\mathcal{A}, \eta \models (\exists x \varphi) \land \psi$ iff $\mathcal{A}, \eta \models \exists x(\varphi \land \psi)$
$\mathcal{A}, \eta \models (\exists x \varphi) \lor \psi$ iff $\mathcal{A}, \eta \models \exists x(\varphi \lor \psi)$

For the proof we use that, by the coincidence lemma, the truth value of ψ is not affected by the assignment to x.

The lemma allows us to "pull out" all quantifiers to the front. Not that we may always rename the bound variables in such a way that no conflict arises. Also note, that bringing a formula to prenex normal form does not increase its size.

2.4 Ehrenfeucht-Fraïssé Games

For LTL the simple observation that we can describe the model-checking problem by a finite automaton immediately lead to a result that certain properties are not expressible by LTL-formulae. To show get concrete examples of properties not expressible in first-order logic, we need slightly more elaborate techniques.

So, what does "inexpressibility" mean? To show that a certain property (like "the universe has even size") *cannot* be expressed by a certain class $\mathcal{L}$ of formulae, it certainly suffices to exhibit two structures $\mathcal{A}$ and $\mathcal{B}$ such that

> $\mathcal{A}$ has the property, $\mathcal{B}$ doesn't
>
> For all $\varphi \in \mathcal{L}$
>
> $$\mathcal{A} \models \varphi \quad \text{iff} \quad \mathcal{B} \models \varphi$$

So, the problem we're looking at is how to construct different structures that are hard to distinguish. Of course, the smaller $\mathcal{L}$ is the easier it is to come up with examples. Since we will need the last property several times, we introduce a notation for it.

> **Definition.** $\mathcal{A}, \vec{a} \equiv_{\mathcal{L}} \mathcal{B}, \vec{b}$ iff for all $\varphi \in \mathcal{L}$, either both $\mathcal{A}, \vec{a} \models \varphi$ and $\mathcal{B}, \vec{b} \models \varphi$ or neither.
>
> Here $\vec{a} = a_1, \ldots, a_\ell$ and $\vec{b} = b_1, \ldots, b_\ell$.

So, how can first order structures look different? Let's look at some examples.

Words—Spot the difference!

a a b a a *vs* a a a a a

---

a a b a a *vs* a a a a b

---

a b a c a *vs* a a b c a

In the first example, the left-hand structure has a letter b, which the right-hand structure doesn't.

In the second example, both structures have a letter b, but on the left-hand structure we can find an to the right of it.

In the third example both structures have a b and the same set of letters (a, c) to the right of it. But only in the left-hand structure we can find a b and a c with an a between them.

Generalising this sequence of more and more complicated differences between structures, we define a game that mimics this step by step discovery of a difference in the structure.

The game, called Ehrenfeucht-Fraïssé game, is a game for two players.

- **Spoiler** tries to prove that the two structures are different.
- **Duplicator** tries to pretend as long as possible, that the structures are the same.

The game is organised in rounds. In each round, Spoiler tries to highlight a difference between the structures by putting an additional pebble in one of the structure. Duplicator tries to match this move in the other structure. The game ends when either the fixed number of rounds is over, or the substructure given by the pebbles differ.

Ehrenfeucht-Fraïssé Games

The game is played on configurations

$$\underbrace{(A, R_1^{\mathcal{A}}, R_2^{\mathcal{A}}, \dots)}_{\mathcal{A}}, a_1, \dots, a_k \;\mid\; \underbrace{(B, R_1^{\mathcal{B}}, R_2^{\mathcal{B}}, \dots)}_{\mathcal{B}}, b_1, \dots, b_k$$

where $a_1, \dots, a_k \in A$ and $b_1, \dots, b_k \in B$.

In each round, Spoiler picks either $a_{k+1} \in A$ or $b_{k+1} \in B$. Then Duplicator picks the other.

Duplicator needs to keep the invariants

- $a_i = a_j$ iff $b_i = b_j$
- $R_i^{\mathcal{A}}(a_{i_1}, \dots, a_{i_\ell})$ iff $R_i^{\mathcal{B}}(b_{i_1}, \dots, b_{i_\ell})$

Back to our examples, we see that in the first structure, Duplicator cannot survive a single round. In the second structure, Duplicator can survive the first round, but not the second. And in the third structure, Duplicator can survive two rounds, but not the third.

As mentioned, we're interested in structures $\mathcal{A}$ and $\mathcal{B}$ with $\mathcal{A} \equiv_{\mathcal{L}} \mathcal{B}$ for appropriate $\mathcal{L} \subseteq \mathsf{FOL}[\ldots]$. We cannot take $\mathcal{L}$ to be the full of first-order logic, as every finite structure $\mathcal{A} = (\{a_1, \ldots, a_n\}, R_1^{\mathcal{A}}, \ldots)$ can be completely (up to isomorphism) described by a first-order formula.

$$\begin{aligned}
&\exists x_1 \exists x_2 \ldots \exists x_n \\
&\qquad ((\neg) R_1(x_1) \wedge (\neg) R_1(x_2) \wedge \ldots \wedge \ldots \\
&\qquad \wedge \forall y (y = x_1 \vee \ldots \vee y = x_n))
\end{aligned}$$

So we have to restrict the language in some useful way. Also note that it is really the absolute number of quantifiers we have to restrict; just restricting the alternations of $\forall$ and $\exists$, that is restricting the "logical complexity" does not help for finite structures.

We therefore define the "quantifier-rank" as the number of nested (not necessarily alternating) quantifiers a formula contains.

Quantifier-Rank

Definition.

$\mathsf{qr}(\top) = \mathsf{qr}(\bot) = \mathsf{qr}(x = y) = \mathsf{qr}(Rx \ldots z) = 0$

$\mathsf{qr}(\neg\varphi) = \mathsf{qr}(\varphi)$

$\mathsf{qr}(\varphi \wedge \psi) = \mathsf{qr}(\varphi \vee \psi) = \max\{\mathsf{qr}(\varphi), \mathsf{qr}(\psi)\}$

$\mathsf{qr}(\forall x \varphi) = \mathsf{qr}(\exists x \varphi) = \mathsf{qr}(\varphi) + 1$

$\mathsf{FOL}_k[R_1, \ldots, R_n] = \{\varphi \in \mathsf{FOL}[R_1, \ldots, R_n] \mid \mathsf{qr}(\varphi) \leq k\}$

$\mathcal{A}, \vec{a} \equiv_k \mathcal{B}\vec{b}$ iff $\mathcal{A}, \vec{a} \equiv_{\mathsf{FOL}_k[\ldots]} \mathcal{B}, \vec{b}$

The notion of the quantifier-rank turns out to be just the right concept for Ehrenfeucht-Fraïssé Games. An increase of the quantifier rank by one corresponds to another round in the game.

Ehrenfeucht-Fraïssé Theorem

Theorem. For any configuration $\mathcal{A}, \vec{a} \mid \mathcal{B}, \vec{b}$ in an EF-game over finite structures, the following are equivalent.

- Duplicator can survive m more rounds.
- $\mathcal{A}, \vec{a} \equiv_m \mathcal{B}, \vec{b}$

Proof. "$\Rightarrow$"

We actually show the contrapositive: if there is a formula distinguishing $\mathcal{A}, \vec{a}$ and $\mathcal{B}, \vec{B}$ then Spoiler can use this to with the game.

Assume $\mathcal{A}, \vec{a} \models \varphi$, $\mathcal{B}, \vec{b} \not\models \varphi$, $\mathsf{qr}(\varphi) \leq m$.

We argue by induction on m.

If $m = 0$, then Spoiler has won already.

If the two structures differ on an atomic formula, the rules of the game say that Spoiler has won. If they differ on a conjunction or disjunction, then they also differ on one conjunct or disjunct.

So let $m \geq 1$.
W.l.o.g $\varphi \equiv \exists x \psi$.

Concerning the "without loss of generality": if φ is a conjunction, one of the conjuncts would already separate the structures (and this is a simpler formula!); similar for a disjunction. For a negation, just swap the roles of $\mathcal{A}$ and $\mathcal{B}$. If φ is a $\forall x\varphi'$ then $\neg\varphi$ can equivalently written as $\exists x\neg\varphi'$ and this formula obviously has the same quantifier rank.

> Since $\mathcal{A}, \vec{a} \models \exists x\psi$ we can pick a c such that $\mathcal{A}, \vec{a}, c \models \psi$. Spoilers strategy is to place a pebble on c. No matter which d Duplicator picks, it will always be the case that $\mathcal{B}, \vec{b}, d \not\models \psi$. Since $\mathsf{qr}(\psi) + 1 = \mathsf{qr}(\varphi)$ the claim follows by IH.

For the other direction we essentially formalise the rules of the game.

> "$\Leftarrow$"

Winning Condition for $\mathcal{A}, \vec{a}$

Define $\chi_{m,\mathcal{A},\vec{a}}$ with $\mathsf{qr}(\chi_{m,\mathcal{A},\vec{a}}) \leq m$ s.t. $\mathcal{B}, \vec{b} \models \chi_{m,\mathcal{A},\vec{a}}(\vec{x})$ iff Duplicator has a strategy for m round in $\mathcal{A}, \vec{a} \mid \mathcal{B}, \vec{b}$.

$$\chi_{m+1,\mathcal{A},\vec{a}}(\vec{x}) = (\bigwedge_{a \in A} \exists y \chi_{m,\mathcal{A},\vec{a},a}) \wedge (\forall y \bigvee_{a \in A} \chi_{m,\mathcal{A},\vec{a},a})$$

$$\chi_{0,\mathcal{A},\vec{a}}(\vec{x}) = \bigwedge_{\substack{\mathcal{A}, \vec{a} \models \varphi \\ \varphi \text{ atomic}}} \varphi \;\wedge \bigwedge_{\substack{\mathcal{A}, \vec{a} \not\models \varphi \\ \varphi \text{ atomic}}} \neg\varphi$$

Note that there are only finitely many $a \in A$ and only finitely many atomic formulae. So we can completely unfold the "big" propositional connectives. Moreover

> $$\mathsf{qr}(\chi_{m,\mathcal{A},\vec{a}}) = m$$

Now, assume $\mathcal{A}, \vec{a} \equiv_m \mathcal{B}, \vec{b}$. Obviously, $\mathcal{A}, \vec{a} \models \chi_{m,\mathcal{A},\vec{a}}$. But then $\mathcal{B}, \vec{b} \models \chi_{m,\mathcal{A},\vec{a}}$. Therefore, Duplicator has a strategy for m rounds.

Example

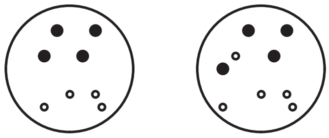
Let $\mathcal{A} = (\{1, 2, 3\}, <, P_a, P_b)$ be the structure for the word `aab`, i.e. $P_a = \{1, 2\}, P_b = \{3\}$.

- Write down $\chi_{1,\mathcal{A},1}$.
- Does $\texttt{aaab}, 1 \models \chi_{1,\mathcal{A},1}$?

Example: Unstructured Sets

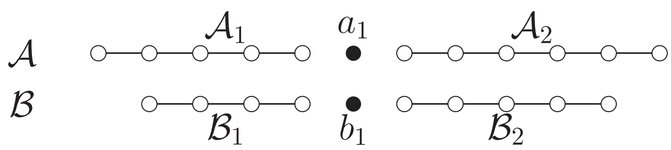
If $\mathcal{A} = (A)$ and $\mathcal{B} = (B)$ are structures over the empty signature, then

$$\mathcal{A} \equiv_m \mathcal{B} \text{ iff } (|A| = |B| \text{ or } |A|, |B| \geq m).$$



We now study Ehrenfeucht-Fraïssé games over linear order, that is, words over the one letter alphabet. Our results will, in particular, imply that the (regular!) property "the length is even" is *not* first-order definable.

The crucial observation is that a pair of pebbles completely separates the game into two independent ones.



For every move of Spoiler in one part, Duplicator has to play in the same part to avoid losing immediately due to order. But, if he loses following this strategy, he has also lost one of the part games, as all the order can say is, that one part game is to the one side of the initial pebble pair and the other part game to the other side. So, the best Spoiler can do is to chose one of the part games and try and win there.

If $\mathcal{A}_1 \equiv_m \mathcal{B}_1$ and $\mathcal{A}_2 \equiv_m \mathcal{B}_2$ then
$\mathcal{A}_1 \oplus \{a_1\} \oplus \mathcal{A}_2 \equiv_m \mathcal{B}_1 \oplus \{b_1\} \oplus \mathcal{B}_2$.

And vice versa!

EF-Games over Linear Orders

Let $m \geq 1$ be a natural number, and $\mathcal{A} = (A, <^{\mathcal{A}})$ and $\mathcal{B} = (B, <^{\mathcal{B}})$ be linear orderings of lengths ℓ_A and ℓ_B, respectively.

Then $\mathcal{A} \equiv_m \mathcal{B}$ iff ($\ell_A = \ell_B$ or $\ell_A, \ell_B \geq 2^m - 1$).

Let's first show how Spoiler can distinguish structures of length less then $2^m - 1$. This is the implication from left to right (or, more precisely, the contrapositive thereof).

“$\Rightarrow$”
Assume $\ell_A \neq \ell_B$, say $\ell_A < \ell_B$, and one of ℓ_A or ℓ_B strictly smaller than $2^m - 1$. So $\ell_A < 2^m - 1$.

By Induction on m we show that Spoiler can win in m rounds.
For $m = 2$ we get $\ell_A < 2^2 - 1 = 4 - 1 = 3$, so $\ell_A \leq 2$. If either structure has strictly less than 2 elements, it's easy to win in 2 rounds. So the case remaining is a 2 element order versus an order with at least 3 Elements. Place a pebble in the larger(!) structure so that there is at least one element to either side. Duplicator has to play an “end-node” in the smaller structure.

In the degenerate case $m = 1$, we have that $\ell_A = 0$. Formally these are no legal first-order structures, as we require the universe to be non-empty. Nevertheless, the argument perfectly extends and, in fact, gives the more natural induction start. Spoiler just puts a pebble on an element of $\mathcal{B}$ and Duplicator can't match by a move in $\mathcal{A}$.

So let $m \geq 3$.
In case $\ell_B \geq 2^m - 1$ Spoiler puts a pebble "in the middle" of $\mathcal{B}$, such that on either side there are at least there are at least $2^{m-1} - 1$ pebbles left. Duplicator necessarily has to split $\mathcal{A}$ such that at least one part has length strictly less than $2^{m-1} - 1$. IH applies.
Otherwise $\ell_B < 2^m - 1$, so $\ell_B \leq 2^{m-1} - 2$, so $\ell_A \leq 2^{m-1} - 3$. In this case, Spoiler puts a pebble "in the middle" of $\mathcal{A}$, i.e., such that on either side there are at strictly less than $2^{m-1} - 1$ nodes left. Note that $2^{m-1} - 2 + 1 + 2^{m-1} - 2 = 2^{m-1} - 3$.
No matter how Duplicator plays, one of the parts will have different lengths, with the smaller part of size strictly less than $2^{m-1} - 1$.
IH applies.

For the other direction we have to provide a strategy for Duplicator to last m rounds. This is trivial in the case that $\ell_A = \ell_B$.

"$\Leftarrow$"
Assume $\ell_A, \ell_B \geq 2^m - 1$.

Roughly speaking, if Spoiler plays close to a border, we take the same distance from that border; otherwise, the structures will remain large, and hence indistinguishable.

Spoiler's move in, say, $\mathcal{A}$ will split up $\mathcal{A}$ in orders of length $\ell_{A,1}$ and $\ell_{A,2}$ with $\ell_{A,1} + \ell_{A,2} + 1 \ell_A \geq 2^m - 1$. So, at most one $\ell_{A,1}, \ell_{A,2}$ is smaller than $2^{m-1} - 1$, since $2^{m-1} - 1 + 2^{m-1} - 1 + 1 = 2^m - 1$.

If one, say $\ell_{A,1}$ is small, then split $\mathcal{B}$ into parts of size $\ell_{B,1} = \ell_{A,1}$ and $\ell_{B,2} \geq 2^{m-1} - 1$. Otherwise split $\mathcal{B}$ into parts both of size at least $2^{m-1} - 1$.

By Induction hypothesis, Duplicator can last another $m - 1$ rounds.

An important consequence of this observation is, that we cannot express the parity of the length of a word.

Corollary. The property "the word has even length" is not first-order definable.

The argument is simply that we can find arbitrary long orders of different parity.

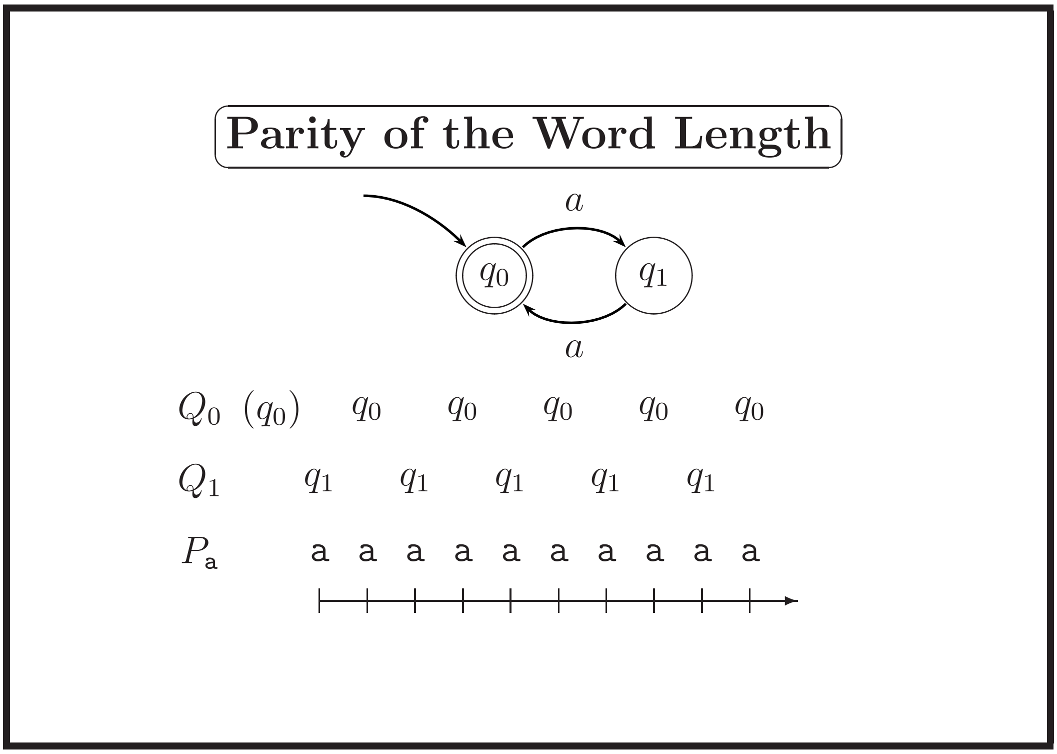
Proof.
Assume $\varphi \in \mathsf{FOL}[<]$ would define the property "even length". Let $m = \mathsf{qr}(\varphi)$. Then φ cannot distinguish $(\{1, 2, \ldots, 2^m\}, <)$ and $(\{1, 2, \ldots, 2^m + 1\}, <)$.

In particular, there are regular properties that are not first-order definable. We will later see, that all first-order definable properties in deed *are* regular.

2.5 Monadic Second-Order Logic

We have seen, that first-order logic cannot represent all regular properties; for example the property "has even length" cannot be expressed. Now let's try and add as little as possible to get all the regular properties.

Regular properties are given by finite automata. So, let's have a look at the one for "has even length".



This is almost a trace structure! The only new thing added are the two "custom predicates". So, as the new feature of first-order logic was, that we could talk freely about elements of the universe, we will now study a new logic where we can talk freely about unary properties.

As in the case of first-order logic, we presuppose an unbounded set of unary predicate variables

$$V^{(1)} = \{X_1, X_2, \ldots\}$$

Syntax of Monadic Second-Order Logic

The set $\mathsf{MSO}[R_1, \ldots, R_n]$ of first-order formulae over the relation symbols $R_1, \ldots, R_n$ is freely generated as follows.

- $(x = y)$ for variables x, y
- $R_j x_{i_1} \ldots x_{i_{n_j}}$ if R_j is of arity n_j
- $\boxed{Xx \text{ for } X \text{ predicate variable}}$
- $\top$, $\bot$, $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$ for formulae ϕ, ψ
- $\forall x\varphi$ and $\exists x\varphi$ for φ a formula and x a variable
- $\boxed{\forall X\varphi \text{ and } \exists X\varphi \text{ for } \varphi \text{ a fla and } X \text{ predicate var}}$

Again, we think of the second-order variables X as bound in $\forall X\varphi$ and $\exists X\varphi$. Correspondingly, we extend our definition of the set of free variables by adding two new clauses.

$$\begin{array}{lcl} \mathsf{fv}(Xx) & = & \{X, x\} \\ \mathsf{fv}(\forall X\varphi) = \mathsf{fv}(\exists X\varphi) & = & \mathsf{fv}(\varphi) \setminus \{X\} \end{array}$$

Just as we had an assignment $\eta\colon V \to A$ to have a meaning for the free first-order variables, we now also need a function $H\colon V^{(1)} \to \mathfrak{P}(A)$ to explain the meaning of the free second-order variables.

Semantics of Monadic Second-Order Logic

Let $\mathcal{A} = (A, R_1^{\mathcal{A}}, \ldots)$ and $\eta\colon V \to A$, and $H\colon V^{(1)} \to \mathfrak{P}(A)$. Define $\mathcal{A}, H, \eta \models \varphi$ for $\varphi \in \mathsf{MSO}[R_1, \ldots]$ inductively.

- $\mathcal{A}, H, \eta \models x = y$ iff $\eta(x) = \eta(y)$
  $\mathcal{A}, H, \eta \models R_1 y_1 \ldots y_n$ iff $(\eta(y_1), \ldots, \eta(y_n)) \in R_i^{\mathcal{A}}$
  $\boxed{\mathcal{A}, H, \eta \models Xx \text{ iff } \eta(x) \in H(X)}$
- $\mathcal{A}, H, \eta \models \varphi \wedge \psi$ iff $\ldots$
  $\mathcal{A}, H, \eta \models \forall x \varphi$ iff $\mathcal{A}, H, \eta_x^a \models \varphi$ for all $a \in A$
- $\boxed{\begin{array}{l}\mathcal{A}, H, \eta \models \forall X \varphi \text{ iff } \mathcal{A}, H_X^U, \eta \models \varphi \text{ for all } U \in \mathfrak{P}(A)\\ \mathcal{A}, H, \eta \models \exists X \varphi \text{ iff } \mathcal{A}, H_X^U, \eta \models \varphi \text{ for some } U \in \mathfrak{P}(A)\end{array}}$

Again, we have a coincidence Lemma.

> **Lemma.**
> η, η', H, H' such that $\eta(x) = \eta'(x)$ for all $x \in \mathsf{fv}(\varphi)$, and $H(X) = H'(X)$ for all $X \in \mathsf{fv}(\varphi)$. Then $\mathcal{A}, \eta, H \models \varphi$ iff $\mathcal{A}, \eta', H' \models \varphi$.

This observation justifies to use the notation

$$\mathcal{A}, U_1, \ldots, U_k, a_1, \ldots, a_\ell \models \varphi$$

If $\mathsf{fv}(\varphi) \subset \{X_1, \ldots, X_k, x_1, \ldots, x_k\}$ with the order understood.

Now, we show that this new logic really is as expressive as we hoped. The proof will, indeed, closely follow our initial intuition.

Representing Automata Runs in MSO

Theorem. Let $\mathcal{L} \subset \Sigma^+$ be regular. Then there is an $\mathsf{MSO}[<, P_a, \ldots]$-formula φ such that

$$w \in \mathcal{L} \text{ iff } w \models \varphi$$

The proof goes by formalising a run of an automaton. The atomic formula $X_i(t)$ can be understood as

After reading the t'th symbol, $\mathfrak{A}$ can be in state q_i.

Run of an Automaton

Let $\mathfrak{A} = (Q, I, \Delta, F)$ be an NFA, $Q = \{q_0, \ldots, q_n\}$.
"$\mathfrak{A}$ has an accepting run": $\exists X_0 \ldots \exists X_n(\varphi_i \wedge \varphi_s \wedge \varphi_f)$

$$\varphi_i \equiv \forall x(\text{“}x \text{ first”} \rightarrow \bigwedge_j (X_j(x) \rightarrow \bigvee_{q_i \in I, (q_i, a, q_j) \in \Delta} P_a(x)))$$

$$\varphi_s \equiv \forall x \forall y(\chi_{\mathsf{next}}(x, y) \rightarrow \bigwedge_j (X_j(y) \rightarrow \bigvee_{(q_i, a, q_j \in \Delta)} (X_i(x) \wedge P_a(y))))$$

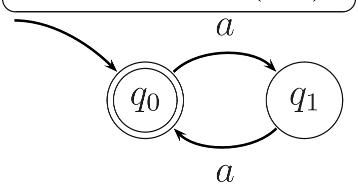
$$\varphi_f \equiv \forall x(\text{“}x \text{ last”} \rightarrow \bigvee_{q_j \in F} X_j(x))$$

Remember, that we can first-order define the used abbreviations.

"x first"	$\forall z. \neg(z < x)$
$\chi_{\mathsf{next}}(x, y)$	$x < y \wedge \forall z(\neg(x < z \wedge z < y))$
"x last"	$\neg \exists y(x < y)$

Intuitively, the formula φ_i says "whatever state $\mathfrak{A}$ is in after reading the first symbol, it must be reachable by one step from an initial state". φ_s says "for every pair x, y of adjacent nodes, if after reading y we can be in some state, this must be due to a legal transition". Finally, φ_f says "at the end, we can be in some accepting state".

Example: $(aa)^*$



$$\begin{aligned}\exists X_0 \exists X_1 [\ & \forall x((\forall y. x \leq y) \rightarrow (X_1(x) \wedge P_a(x)))\wedge \\ & \forall x \forall y(((x < y) \wedge \neg \exists z(x < z \wedge z < y)) \rightarrow \\ & \quad ((X_0(y) \rightarrow (X_1(x) \wedge P_a(y)))\wedge \\ & \quad (X_1(y) \rightarrow (X_0(x) \wedge P_a(y)))))\wedge \\ & \forall x((\forall y. y \leq x) \rightarrow X_0(y))]\end{aligned}$$

Note that this way of describing a run one the one hand works with non-deterministic automata, and on the other hand enforces positive knowledge about the existence of a run. The given formula, for example, would not hold on the word `aabbaa`.

Also, as we have seen earlier, the property "even length" cannot be expressed in first-order logic. So, one consequence of the automaton we have just constructed is, the MSO strictly extends first-order logic.

> **Corollary.** There are MSO-definable word-languages that are not first-order definable.

We have seen that monadic second-order logic strictly extends first-order logic. What we don't have yet is an upper bound on how expressive monadic second-order is. For first-order logic, Ehrenfeucht-Fraïssé games turned out to be a useful tool. We now define a similar type of games.

MSO Games

$$\mathcal{A}, U_1, \ldots, U_k, a_1, \ldots, a_\ell \mid \mathcal{B}, V_1, \ldots, V_k, b_1, \ldots, b_\ell$$

where $U_1, \ldots, U_k \subset A$, $V_1, \ldots, V_k \subset B$.

Spoiler can choose between two types of moves.

- choose $a_{\ell+1} \in A$, or $b_{\ell+1} \in B$
- choose $U_{k+1} \subset A$, or $V_{k+1} \subset B$

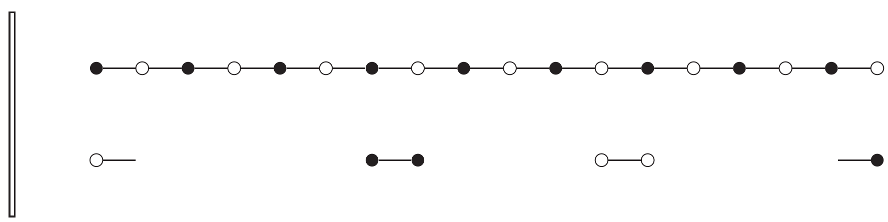
Duplicator needs to keep the invariants

- $a_i = a_j$ iff $b_i = b_j$; $R_i^{\mathcal{A}}(a_{i_1}, \ldots, a_{i_\ell})$ iff $R_i^{\mathcal{B}}(b_{i_1}, \ldots, b_{i_\ell})$
- $a_i \in U_j$ iff $b_i \in V_j$

In other words, the game is extended by the possibility of colouring a set of nodes.

Returning back to the example "even length", we immediately see that this extension of the game gives Spoiler a lot more power. He can now distinguish a linear order of odd length from one of even length by a fixed number of moves.

First, he colours every other node of the structure of even length.



No matter how Duplicator tries to match this colouring, Spoiler can either point out that the first or last node is coloured wrongly, or that there are two adjacent nodes of the same colour.

An analogous theorem as with first-order Ehrenfeucht-Fraïssé games holds.

MSO Games—The Theorem

For any configuration $\mathcal{A}, \vec{U}, \vec{a} \mid \mathcal{B}, \vec{V}, \vec{b}$ in an MSO-game over finite structures, the following are equivalent.

- Duplicator can survive m more rounds.
- $\mathcal{A}, \vec{U}, \vec{a} \equiv_m^{MSO} \mathcal{B}, \vec{V}, \vec{b}$

Here $\equiv_m^{MSO}$ denotes the relation of indistinguishability by MSO-formulae of quantifier rank up to m. The notion of quantifier rank is extended in a straight-forward way, by setting

$$\mathsf{qr}(\forall X \varphi) = \mathsf{qr}(\exists X \varphi) = 1 + \mathsf{qr}(\varphi)$$

The proof is quite analogous to the proof of the Ehrenfeucht-Fraïssé theorem.

Proof. "$\Rightarrow$"
Assume $\mathcal{A}, \vec{U}, \vec{a} \models \varphi$, $\mathcal{B}, \vec{V}, \vec{b} \not\models \varphi$, $\mathsf{qr}(\varphi) \leq m$.

Again, we have to describe a winning strategy for Spoiler. The only new case is handled by the additional move that Spoiler can do in an MSO-game.

Spoiler than can win in m rounds, using a strategy as in the EF-game. In the only new case $\varphi \equiv \exists X \psi$, there is some $U \subset A$ satisfying ψ and Spoiler will colour this set.

"$\Leftarrow$"

For the other direction we, again, just formalise the rules of the game.

Winning Condition for $\mathcal{A}, \vec{U}, \vec{a}$

Define $\chi_{m,\mathcal{A},\vec{U},\vec{a}}$ with $\mathsf{qr}(\chi_{m,\mathcal{A},\vec{U},\vec{a}}) \leq m$ s.t. $\mathcal{B}, \vec{V}, \vec{b} \models \chi_{m,\mathcal{A},\vec{U},\vec{a}}(\vec{x})$ iff Duplicator has a strategy for m rounds in $\mathcal{A}, \vec{U}, \vec{a} \mid \mathcal{B}, \vec{V}, \vec{b}$.

$$\begin{aligned}
\chi_{m+1,\mathcal{A},\vec{U},\vec{a}}(\vec{X}, \vec{x}) = \; & (\bigwedge_{a \in A} \exists y \chi_{m,\mathcal{A},\vec{U},\vec{a},a}) \wedge (\forall y \bigvee_{a \in A} \chi_{m,\mathcal{A},\vec{U},\vec{a},a}) \\
& \wedge (\bigwedge_{U \subset A} \exists Y \chi_{m,\mathcal{A},\vec{U},U,\vec{a}}) \wedge (\forall Y \bigvee_{U \subset A} \exists Y \chi_{m,\mathcal{A},\vec{U},U,\vec{a}}) \\
\chi_{0,\mathcal{A},\vec{U},\vec{a}}(\vec{x}) = \; & \bigwedge_{\substack{\mathcal{A}, \vec{a} \models \varphi \\ \varphi \text{ atomic}}} \varphi \quad \wedge \quad \bigwedge_{\substack{\mathcal{A}, \vec{a} \not\models \varphi \\ \varphi \text{ atomic}}} \neg\varphi
\end{aligned}$$

Looking back to our formulae with bounded quantifier rank, we have seen that the set of structures are divided into equivalence classes by the relation $\equiv_m^{MSO}$, that can also be well described by appropriate games.

An important observation is that there are only finitely many of them.

Proposition. For every m and every finite $X \subset V \cup V^{(1)}$ there are, up to logical equivalence, only finitely many $\varphi \in \mathsf{MSO}[R_1, \ldots, R_n]$ with $\mathsf{qr}(\varphi) \leq m$ and $\mathsf{fv}(\varphi) \subset X$.

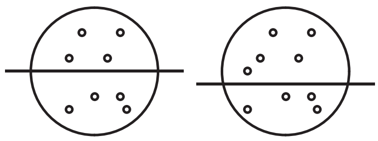
Proof. Induction on m. For $m = 0$ there are only finitely many atoms, hence only finitely many boolean combinations.
All φ with $\mathsf{qr}(\varphi) = m + 1$ are boolean combinations of formulae $\exists y \psi$ or $\forall y \psi$ with $\mathsf{qr}(\psi) = m$, of which there are only finitely many.

Now that we have developed a tool for obtaining bounds on the expressibility of Monadic Second Order Logic, let's use them and go back to the counting example. Remember, that first-order logic could count linearly far on unstructured sets and exponentially for on linear orders. MSO doesn't need an order to get exponentially far.

Example: Unstructured Sets

If $\mathcal{A} = (A)$ and $\mathcal{B} = (B)$ are structures over the empty signature, then

$$\mathcal{A} \equiv_m^{MSO} \mathcal{B} \text{ iff } (|A| = |B| \text{ or } |A|, |B| \geq 2^{m-1}).$$



Note that this observation in particular shows that, even though the word language $(\mathtt{aa})^+$ is MSO-definable, the property "The universe has even size" is not.

Proof. "$\Rightarrow$"
Let $|A| < |B|$ and $|A| < 2^{m-1}$. We describe a strategy for Spoiler to win in at most m moves.
The case $m = 1$ is trivial.
Let $m \geq 2$. If $|A| \leq 2^{m-1}-2$ Spoiler "splits" (using a set move) it into two small sets, i.e., of size $< 2^{m-2}$. No matter how Duplicator splits B, the induction hypothesis applies to one part. Otherwise, $|B| \geq 2^{m-1}$ and can be spilt into two big parts (of size at least 2^{m-2}). No matter how Duplicator splits A, one part will be small and the induction hypothesis applies.

"$\Leftarrow$" Let $|A|, |B| \geq 2^{m-1}$. We describe a strategy for Duplicator to play m rounds.
The case $m = 1$ is easy.
Let $m \geq 2$. Point moves of Spoiler are handled trivially; set-moves split one set, say A. Since $|A| \geq 2^{m-1}$, at most one part can be small. This part is then matched exactly by Duplicator.

The observation that $\equiv_m^{MSO}$ partitions, for every m, the class of all first-order structures into *finitely* many equivalence classes is the crucial tool towards obtaining an upper bound on the definable properties of monadic second-order logic. As usual, finitely many classes of indistinguishable objects leads to regular languages.

MSO-Definability and Regular Language

For $m \geq 0$ we note that

- there are only finitely many $\equiv_m^{MSO}$ classes $[\![\mathcal{A}_w]\!]$, and
- the $\equiv_m^{MSO}$ class of wu only depends on that of w and u.

So

$$\begin{aligned} Q &= \{[\![\mathcal{A}_w]\!] \mid w \text{ a word}\} \\ \delta\colon\ & Q \times \Sigma \quad \to \quad Q \\ & ([\![\mathcal{A}_w]\!], a) \quad \mapsto \quad [\![\mathcal{A}_{wa}]\!] \end{aligned}$$

defines a finite automaton.

Recall that the composition property can be easily seen by looking at games. If Duplicator has a strategy for m rounds on $w \mid w'$ and a strategy for m rounds on $u \mid u'$, he can combine them to a strategy for m rounds on $wu \mid w'u'$. Also note that this property is essential for the above "definition" to make sense!

Summing up, we get Büchi's Theorem.

Büchi's Theorem

The following are equivalent for word languages $\mathcal{L} \subset \Sigma^+$.

- $\mathcal{L}$ is regular
- $\mathcal{L}$ is MSO definable, i.e., there is an MSO formula φ and

$$\mathcal{L} = \{w \mid w \models \varphi\}$$

Proof. "⇒" Code up the run of an automaton
"⇐" Given φ, construct the automaton with $\equiv_m^{MSO}$ classes as states, where $m = \mathsf{qr}(\varphi)$.

Büchi's Theorem and its proofs have some interesting consequences.

> **Corollary.** There is $s\colon \mathbb{N} \to \mathbb{N}$ such that if φ has a word model, it has a word model of size at most $s(\mathsf{qr}(\varphi))$.

We know, by the pumping lemma, that if an automaton has an accepting run, then it has an accepting run on a word with at most as many letters as the automaton has states. But we used the same automaton for all the formulae with a given quantifier rank m.

> **Corollary.** If a language of words is MSO-definable, then it is MSO-definable by a formula of the shape
>
> $$\exists X_1 \exists X_2 \dots \exists X_n \varphi$$
>
> with φ first-order.

In other words, there is no need to alternate existential and universal quantification of second-order quantification. This is a completely different situation then in second-order arithmetic!

The reason is quite simple. If a property is MSO-definable, then the property, by Büchi's Theorem is regular. Hence there is a finite automaton defining it. But we can translate the property "$\mathfrak{A}$ has a successful run" into a formula of the given shape.

The "semantic" proof of Büchi's Theorem provides a lot of insight into the nature of monadic second-order logic. However, it does not give us a way to construct the automaton we know it has to exist. Therefore it's worth knowing an alternative proof of Büchi's Theorem, which gives additional insights. We will sketch it here.

Alternative Proof

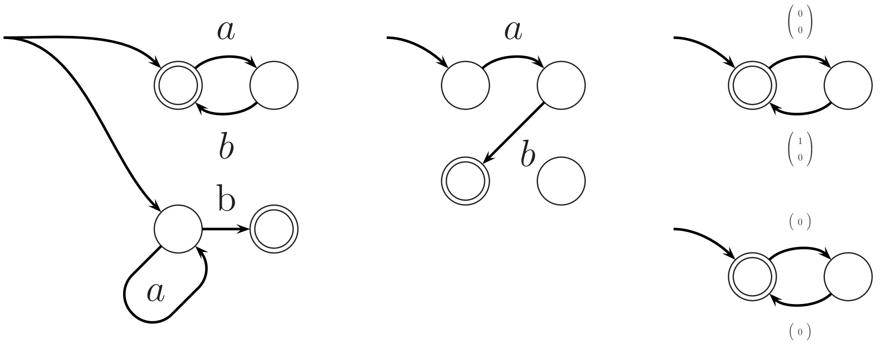
NFAs are closed under

- intersection $\mathcal{L} \cap \mathcal{L}'$
- union $\mathcal{L} \cup \mathcal{L}'$
- complement
  e.g., via power-set construction to get a DFA
- projection of the alphabet
  $\{\pi(a_1) \ldots \pi(a_n) \mid a_1 \ldots a_n \in \mathcal{L}\}$

of their languages.

Moreover, this closure is effective.

NFA Closure Properties



$Q = Q_0 \cup Q_1$ $\qquad$ $Q = Q_0 \times Q_1$ $\qquad$ $Q = Q_0$

Once we have this observation, our strategy is, to first eliminate first-order(!!) quantification. $\exists x \varphi$ is replaced by

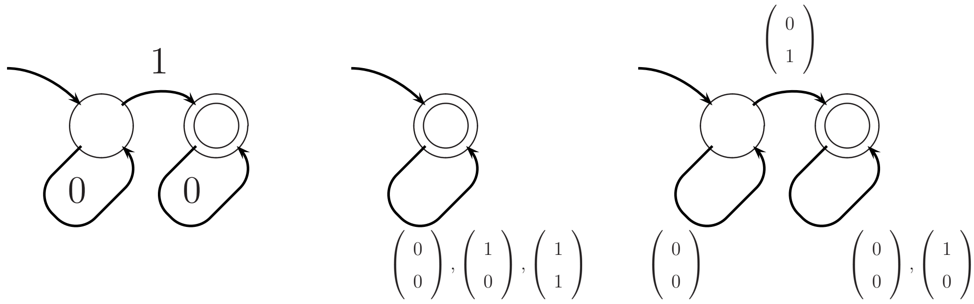
$$\exists X.(\text{“}X \text{ singleton”} \wedge \varphi[X/x])$$

The formula $x \in Y$ then becomes

$$X \subset Y$$

Note that both primitives (“singleton”, inclusion, order) can be easily defined by automata working on the corresponding trace structures.

Primitives



"X is a singleton" $\qquad$ $X \subset Y$ $\qquad$ "$x < y$"

Then the above observations suffice, as intersection, union, and complement correspond to the boolean connectives. Projection of the language is existential set quantification.

This new has additional consequences. Since the construction of the automaton is constructive, we can decide satisfiability. For a given formula just construct the automaton and decide non-emptiness.

> **Corollary.** Satisfiability for MSO on words is decidable.

Another interesting consequence of this decidability result is the decidability of Presburger arithmetic, that is, the first-order arithmetic of the natural numbers with addition as only operation.

The idea is, that a predicate in a trace structure can be thought of as a natural number with the X corresponding to

$$\sum_{i \geq 1} 2^{i-1} \cdot X(i)$$

where the finiteness of the sum is guaranteed by the fact, that we consider only finite words.

Presburger Arithmetic

$X = 13$	1	0	1	1	0	0	0	0	0	0
$X = 30$	0	1	1	1	1	0	0	0	0	0
$Z = 43$	1	1	0	1	0	1	0	0	0	0

In this presentation, addition is MSO definable. Note that $X + Y = Z$ iff

$$\begin{aligned} \exists C.(\ & (\forall i.\text{“}Z(i) = X(i) \oplus Y(i) \oplus C(i)\text{”}) \\ & \wedge \forall i(C(i) \leftrightarrow \exists j(j \leq i \wedge X(j) \wedge Y(j) \wedge \forall k((j < k \wedge k < i) \rightarrow (X(k) \vee Y(k)))))) \end{aligned}$$

provided the universe is big enough, to hold the result, which can be expressed, e.g., by asserting $\forall i((X(i) \vee Y(i)) \rightarrow \exists j(i < j))$.

Corollary. Presburger Arithmetic is decidable.