

CONCURRENT SYSTEMS

LECTURE 15

Prof. Daniele Gorla



Logics for System's Properties

(In)equivalences between systems hold because of different properties of the systems themselves

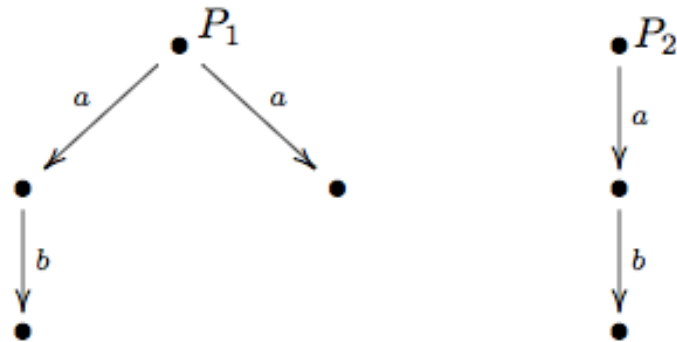
Logics = a formal way to express these properties

Satisfiability relation states when a process satisfies a property

Enjoying the same properties coincides with being bisimilar



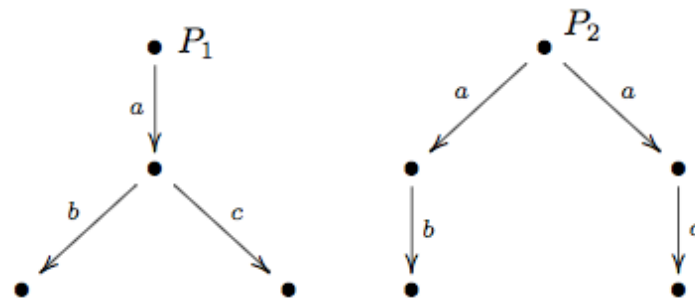
Example:



These two proc's are NOT bisimilar because

- P1 can perform an action a not followed by any b
- P2, after every a, can always perform an action b

Example:



These two proc's are NOT bisimilar because

- P1 can perform an action a and then choose between b and c
- P2 can perform an a not followed by any b and an a not followed by any c

$$\varphi := \text{TT} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \diamond a\varphi \text{ where } a \in \text{Action}$$

The language generated by this grammar will be denoted by Form; every element of this set will be called formula

Let $\models \subseteq \text{Proc} \times \text{Form}$ be a relation between processes and formulae.

process P *satisfies* formula φ , and write $P \models \varphi$, if and only if $(P, \varphi) \in \models$.

relation \models can be inductively defined

$$P \models \text{TT}$$

$$P \models \neg\varphi \text{ iff } P \not\models \varphi$$

$$P \models \varphi_1 \wedge \varphi_2 \text{ iff } P \models \varphi_1 \wedge P \models \varphi_2$$

$$P \models \diamond a\varphi \text{ iff } \exists P' : P \xrightarrow{a} P' \wedge P' \models \varphi$$

To simplify the proofs, we consider a more general form of conjunction: $\bigwedge_{i \in I} \varphi_i$ where I is any indexing set (possibly, also infinite).

Satisfiability for this operator is similar to that for the binary operator.





Of course, we can use the boolean constant FALSE (written FF) and classical logical operators like disjunction \vee and implication \Rightarrow (they can all be derived in the usual way from TT, \neg and \wedge).

Another very useful logical operator is ‘box’: let us define $\neg \diamond a\varphi$ as $\Box a\neg\varphi$

$$\begin{aligned} P \models \Box a\varphi \text{ iff } P \models \neg \diamond a\neg\varphi \text{ iff } P \not\models \diamond a\neg\varphi \text{ iff} \\ \nexists P' : P \xrightarrow{a} P' \wedge P' \models \neg\varphi \text{ iff } \forall P' : P \xrightarrow{a} P' \vee P' \models \varphi \end{aligned}$$

From the last condition, we have that $P \models \Box a\varphi$ if and only if

$$\forall P' : P \xrightarrow{a} P' \Rightarrow P' \models \varphi$$

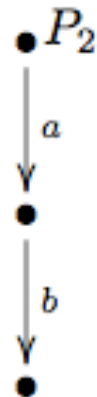
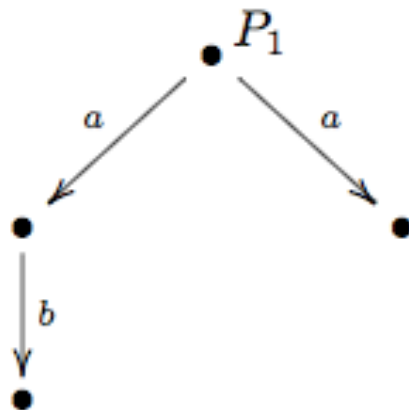
EXAMPLE: Let us now consider formula $\Box a\text{FF}$

By definition, this happens only if, for every P' resulting from P after action a , it holds $P' \models \text{FF}$

However, this can never happen, whatever P' be;

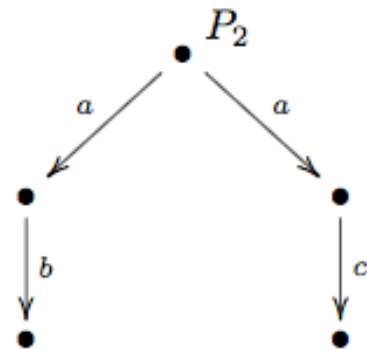
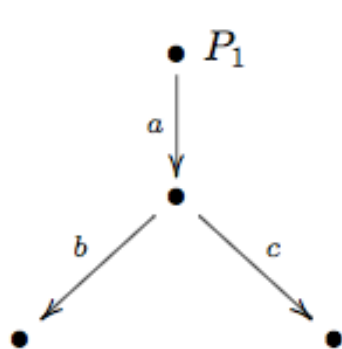
hence, $P \models \Box a\text{FF}$ holds true only if P *cannot* perform any action a





$$\varphi \triangleq \diamond a \Box b \text{FF}$$

We have that $P_1 \models \varphi$ whereas $P_2 \not\models \varphi$.



$$\Box a \diamond b \text{TT}$$

$$\diamond a (\diamond b \text{TT} \wedge \diamond c \text{TT})$$

These formulae are both satisfied by P_1 but not by P_2 .



Let us now define the set of formulae satisfied by a process as

$$L(P) = \{\varphi \in \text{Form} : P \models \varphi\}$$

To simplify the proof, let us modify the set of formulae by allowing conjunctions over a numerable set of formulae

Theorem: $P \sim Q$ if and only if $L(P) = L(Q)$

(\Rightarrow) By induction on the syntax tree of the formula, let us show that

$$\varphi \in L(P) \text{ iff } \varphi \in L(Q)$$

Base case: The only possible case is with $\varphi = \text{TT}$.

- By the satisfiability relation, φ belongs to the set of formulae of every process
- so also to $L(P)$ and $L(Q)$



Inductive step: Let us assume the thesis for every tree of height at most h . Let $h + 1$ be the height of φ . Let us distinguish on the outmost operator in φ

1. $\varphi \triangleq \bigwedge_{i \in I} \varphi_i$. Let us assume that $\varphi \in L(P)$; by definition,

$$\varphi \in L(P) \iff P \models \varphi \iff \forall i \in I P \models \varphi_i \iff \forall i \in I \varphi_i \in L(P)$$

By definition, every formula φ_i has a syntactic tree of height at most h ; hence, by inductive hypothesis, $\varphi_i \in L(P) \iff \varphi_i \in L(Q)$. From this,

$$\forall i \in I \varphi_i \in L(Q) \iff \forall i \in I Q \models \varphi_i \iff \bigwedge_{i \in I} \varphi_i \in L(Q) \iff \varphi \in L(Q)$$

2. $\varphi \triangleq \neg \varphi'$. In this case, $\varphi \in L(P)$ iff $\neg \varphi' \in L(P)$ iff $\varphi' \notin L(P)$. The height of the syntax tree for φ' is h ; by inductive hypothesis,

$$\varphi' \notin L(P) \iff \varphi' \notin L(Q) \iff \neg \varphi' \in L(Q)$$



3. $\varphi \triangleq \diamond a\varphi'$. We now have $\varphi \in L(P)$ iff $\exists P' : P \xrightarrow{a} P' \wedge P' \models \varphi'$; the last condition is equivalent to $\varphi' \in L(P')$.

By hypothesis, $P \sim Q$; hence, $\exists Q' : Q \xrightarrow{a} Q' \wedge P' \sim Q'$.

Because the height of the syntax tree for φ' is h , by inductive hypothesis we have that $\varphi' \in L(Q')$.

By definition, this entails that $Q' \models \varphi'$ and so $Q \models \diamond a\varphi' = \varphi$.

Up to now, we have proved that $\varphi \in L(P) \Rightarrow \varphi \in L(Q)$. To complete the proof, we have to prove the converse implication; this can be easily done by swapping P and Q .



(\Leftarrow) We prove that $R \triangleq \{(P, Q) : L(P) = L(Q)\}$ is a simulation; this suffices, since the relation just defined is trivially symmetric.

Let $(P, Q) \in R$ and $P \xrightarrow{a} P'$. Let us consider

$$\varphi \triangleq \Diamond a \varphi' \text{ where } \varphi' \triangleq \bigwedge_{\varphi'' \in L(P')} \varphi''$$

By construction, $P \models \varphi$, hence $Q \models \varphi$, because $L(Q) = L(P)$; then, $\exists Q' : Q \xrightarrow{a} Q'$ such that $Q' \models \varphi'$.

Hence, $\forall \varphi'' \in L(P'). \varphi'' \in L(Q')$, i.e. $L(P') \subseteq L(Q')$.

By contradiction, let us assume that the inclusion is proper, i.e. $L(P') \subset L(Q')$. Thus, $\exists \hat{\varphi} : \hat{\varphi} \in L(Q') \wedge \hat{\varphi} \notin L(P')$.

Then, $\neg \hat{\varphi} \in L(P')$ and this would imply that $\neg \hat{\varphi} \in L(Q')$. This is a contradiction because $L(Q')$ cannot contain a formula and its negation. \square

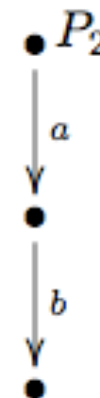


The Logic approach presented so far is very natural for proving inequivalences:

- Show one formula that is satisfied by a proc but not by the other

It is not very effective for concretely proving equivalences:

- E.g., to show that $P \sim Q$, we should check that every formula in $L(P)$ belongs to $L(Q)$ and conversely.
- The problem is that $L(P)$ is infinite, for every P : it contains TT , $TT \wedge TT$, $TT \wedge TT \wedge TT$, ...
- Even if we restrict to logical equivalence class, the situation does not change.
- EXAMPLE: consider process P_2 :
 - it satisfies $\Box bFF$, $\Box cFF$, $\Box dFF$, ...
 - so $L(P_2)$ is infinite because so is the action set





Sub-Logics

A negation-free logic Let us first consider the sub-logic without negation:

$$\varphi := \text{TT} \mid \varphi \wedge \varphi \mid \diamond a\varphi \text{ where } a \in \text{Action}$$

Remark: the formula $\Box a\text{FF}$ is not expressible anymore.

→ Hence, we can only express through formulae what a process is able to do.

Let us call $L_-(P)$ the set of negation-free formulae satisfied by process P

Theorem: P simulates Q if and only if $L_-(Q) \subseteq L_-(P)$.

Proof. The proof is similar to the one for the previous Theorem.

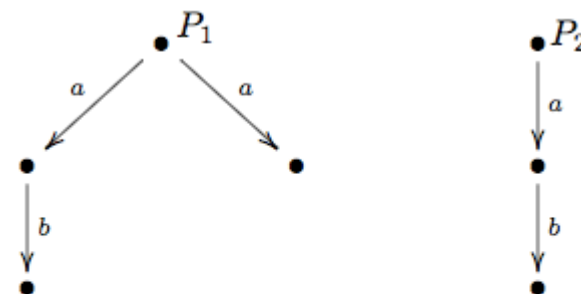
The main difference is in the (\Leftarrow) implication, when $\varphi = \diamond a\varphi'$, because here we do not prove that the inclusion cannot be proper (indeed, in general it is not).

An easy corollary of this result is that there is a double simulation between P and Q if and only if $L_-(P) = L_-(Q)$.

Example: it can be checked that

$$L_-(P_2) = L_-(P_1)$$

indeed, P_1 can simulate P_2 and viceversa,
but the simulations are not bisimulations.





A negation- and conjunction-free logic To conclude, let us consider the sub-logic obtained by also removing conjunction:

$$\varphi := \text{TT} \mid \diamond a\varphi \text{ where } a \in \text{Action}$$

Let us call $L_{\wedge, -}(P)$ the set of formulae of this sub-logic satisfied by process P

Let us also call $\text{Lang}(P)$ the set of strings accepted by the automaton isomorphic to the LTS of P in which every state is final

Theorem: $L_{\wedge, -}(P) = L_{\wedge, -}(Q)$ if and only if $\text{Lang}(Q) = \text{Lang}(P)$.

Proof. Let $\varphi = \diamond a_1, \dots, \diamond a_n \text{TT} \in L_{\wedge, -}(P)$.

By definition, this holds if and only if $\exists P_1, \dots, P_n$ such that $P \xrightarrow{a_1} P_1 \dots \xrightarrow{a_n} P_n$

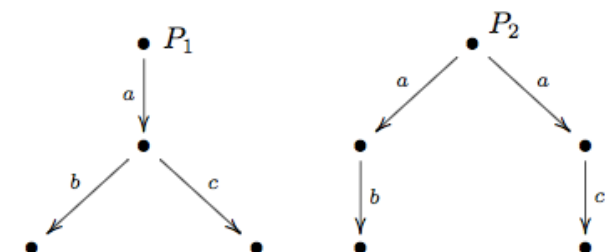
hence, if and only if $a_1, \dots, a_n \in \text{Lang}(P)$.

In concurrency theory, when two processes have the same language, they are called *trace equivalent*, where a trace is any sequence of actions performed by the process.

It is easy to see that the set of traces of a process P is $\text{Lang}(P)$

EXAMPLE: P_1 and P_2 are trace equivalent

they satisfy the same set of formulae of the sub-logic without negation and conjunction.





A Logic for Weak Bisimilarity

The logic that characterizes weak bisimilarity is very similar to the logic for strong bisimilarity:

$$\varphi := \text{TT} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle\!\rangle\!\rangle a\varphi \text{ where } a \in \text{Action}$$

The only difference is in the diamond operator $\diamond a$, that here is turned into $\langle\!\langle\!\rangle\!\rangle a$.

This is very similar to the difference between the definition of weak and strong simulation, where the former is obtained from the latter by replacing “ $\xrightarrow{\alpha}$ ” with “ $\xRightarrow{\hat{\alpha}}$ ”. Satisfiability of this new operator is expectable:

$$P \models \langle\!\langle\!\rangle\!\rangle a\varphi \text{ iff } \exists P' : P \xRightarrow{\hat{a}} P' \wedge P' \models \varphi$$