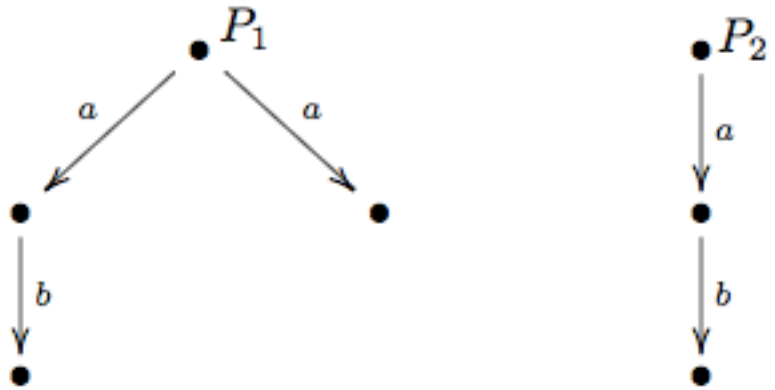
Logics for System's Properties

- (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

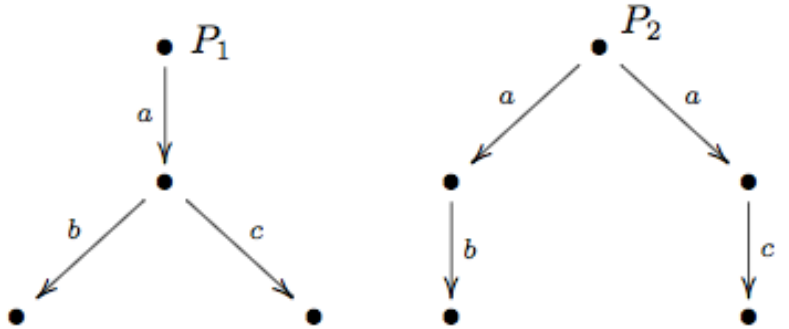
- Example:



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

Syntax and Satisfiability

$$\varphi \;:=\; \mathrm{TT} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond a\varphi \quad \textit{where } a \in \mathsf{Action}$$

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

Let $\models \subseteq \mathsf{Proc} \times \mathsf{Form}$ 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

$$P \models \mathrm{TT}$$

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

$$P \models \varphi_1 \wedge \varphi_2 \textit{ iff } P \models \varphi_1 \;\wedge\; P \models \varphi_2$$

$$P \models \Diamond a\varphi \textit{ iff } \exists P' : P \xrightarrow{a} P' \;\wedge\; P' \models \varphi$$

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 $\Box a \neg \varphi$

$$P \models \Box a\varphi \text{ iff } P \models \neg \Diamond a \neg \varphi \text{ iff } P \not\models \Diamond a \neg \varphi \text{ iff}$$
$$\nexists P' : P \xrightarrow{a} P' \wedge P' \models \neg\varphi \text{ iff } \forall P' : P \overset{a}{\not\to} P' \vee P' \models \varphi$$

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

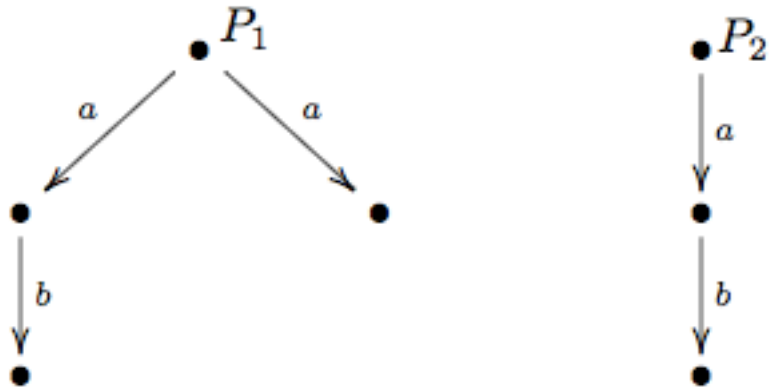
$$\forall P' : P \xrightarrow{a} P' \Rightarrow P' \models \varphi$$

EXAMPLE: Let us now consider formula □aFF

By definition, this happens only if, for every P' resulting from P after action a, it holds P' |= FF

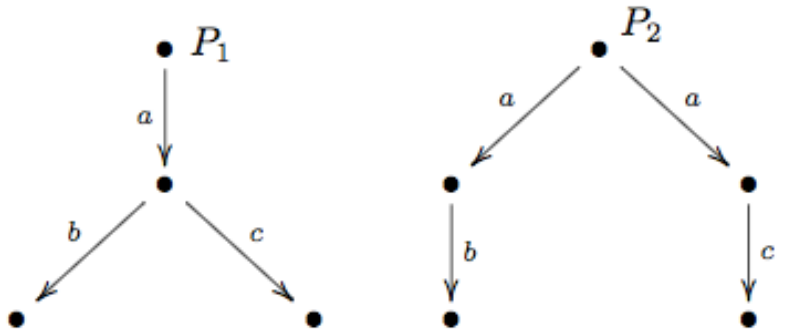
However, this can never happen, whatever P' be;

hence, P |= □aFF holds true only if P ***cannot*** perform any action a



$$\varphi \triangleq \Diamond a \Box b \mathtt{FF}$$

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



$$\Box a \Diamond b \mathtt{TT} \qquad \Diamond a(\Diamond b \mathtt{TT} \wedge \Diamond c \mathtt{TT})$$

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

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

- 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 $L(P) = L(Q)$

(=>) By induction on the syntax tree of the formula, let us show that

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

- Base case: The only possible case is with $\phi = TT$.
  - 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 $\exists P' : P \xrightarrow{a} P' \ \wedge \ P' \vDash \varphi'$; 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 $Q' \vDash \varphi'$ and so $Q \vDash \diamond a\varphi' = \varphi$.

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

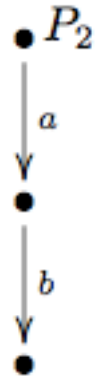
Hence, $\forall \varphi'' \in L(P').\varphi'' \in L(Q')$, i.e. $L(P') \subseteq L(Q')$

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 ~ 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 ⋀ TT, TT ⋀ TT ⋀ TT, …
  - Even if we restrict to logical equivalence class, the situation does not change.
  - EXAMPLE: consider process P2:
    - it satisfies □bFF, □cFF, □dFF, ...
    - so L(P2) is infinite because so is the action set



Sub-Logics

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

$$\varphi \;:=\; \mathrm{TT} \mid \varphi \wedge \varphi \mid \Diamond a\varphi \quad \textit{where } a \in \mathsf{Action}$$

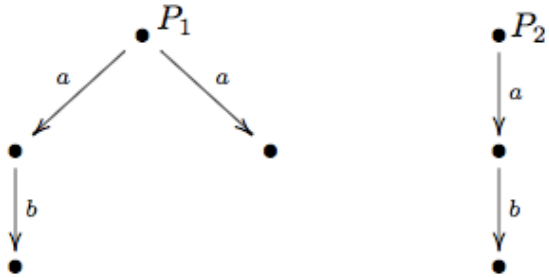
- Remark: the formula □aFF is not expressible anymore.
  → Hence, we can only express through formulae what a process is able to do.
- Let us call $L_{\neg}(P)$ the set of negation-free formulae satisfied by process P

Theorem: P simulates Q if and only if $L_{\neg}(Q) \subseteq L_{\neg}(P)$.

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

The main difference is in the (⇐) implication, when $\phi = \Diamond a\phi'$, 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 $L_{\neg}(P) = L_{\neg}(Q)$.
- Example: it can be checked that
  $L_{\neg}(P2) = L_{\neg}(P1)$
  indeed, P1 can simulate P2 and viceversa,
  but the simulations are not bisimulations.



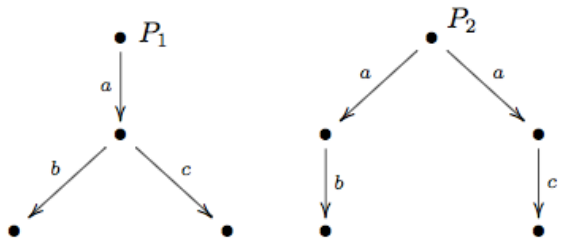
A negation- and conjunction-free logic To conclude, let us consider the sub-logic obtained by also removing conjunction:

$$\varphi \;:=\; \mathrm{TT} \mid \diamond a\varphi \quad \textit{where } a \in \mathsf{Action}$$

- Let us call $L_{\wedge,\neg}(P)$ the set of formulae of this sub-logic satisfied by process P
- Let us also call Lang(P) the set of strings accepted by the automaton isomorphic to the LTS of P in which every state is final

Theorem: $L_{\wedge,\neg}(P) = L_{\wedge,\neg}(Q)$ if and only if Lang(Q) = Lang(P).
Proof. Let $\phi = \Diamond a1, \ldots, \Diamond an\mathrm{TT} \in L_{\wedge,\neg}(P)$.
By definition, this holds if and only if $\exists$ P1,...,Pn such that P −a1−> P1 ... −an−> Pn
hence, if and only if a1,...,an $\in$ 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 exactly Lang(P)
- EXAMPLE: P1 and P2 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:

$$\varphi \;:=\; \mathtt{TT} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \ll\!\gg a\varphi \quad \text{where } a \in \mathit{Action}$$

The only difference is in the diamond operator $\Diamond a$, that here is turned into $\ll\!\gg a$.

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 "$\stackrel{\widehat{\alpha}}{\Longrightarrow}$". Satisfiability of this new operator is expectable:

$$P \vDash \ll\!\gg a\varphi \;\mathit{iff}\; \exists P' : P \stackrel{\widehat{a}}{\Longrightarrow} P' \;\wedge\; P' \vDash \varphi$$