

CIS 673: Lecture 12: Oct 16  
Assume Guarantee Reasoning

- $Q_1$  is a simplified version of  $P_1$ ,  $Q_2$  is a simplified version of  $P_2$ , and we wish to establish  $P_1 \parallel P_2$  implements  $Q_1 \parallel Q_2$
- $P_1$  may not implement  $Q_1$ , but only under the assumption that its environment behaves like  $Q_2$ , and vice versa
- Assume guarantee strategy:
  - Establish  $P_1 \parallel Q_2$  implements  $Q_1$
  - Establish  $Q_1 \parallel P_2$  implements  $Q_2$
- Advantage: No need to consider detailed modules  $P_1$  and  $P_2$  together
- Problem: Is this circular reasoning sound?
  - $Q_1$  is established assuming  $Q_2$
  - $Q_2$  is established assuming  $Q_1$

## Assume Guarantee Theorem

- Let  $P_1$  and  $P_2$  be two compatible modules
- Let  $Q_1$  and  $Q_2$  be two compatible modules
- Suppose  $P_1 \parallel Q_2 \preceq^L Q_1$
- Suppose  $Q_1 \parallel P_2 \preceq^L Q_2$
- Every external variable of  $Q_1 \parallel Q_2$  is an observable variable of  $P_1 \parallel P_2$
- Then,  $P_1 \parallel P_2 \preceq^L Q_1 \parallel Q_2$

## Soundness of Assume Guarantee

- It is not sufficient for implementation to be like implication and parallel composition to be like conjunction
- Crucial aspect of our model: a module interacts with its environment in a non-blocking way
- In every subround, there is a way to choose new values of controlled variables, no matter what has happened so far
- Example: send-receive alternating-bit protocol

**module** *ABPSender* **is**  
**interface**  $trans_S: \mathbb{E}$ ;  $abp: \mathbb{B}$ ;  $msg: \mathbb{M}$   
**external**  $trans_R: \mathbb{E}$ ;  $ack: \mathbb{B}$   
**private**  $cons: \mathbb{E}$ ;  $pc: \{send, wait\}$ ;  $x: \mathbb{B}$ ;  $z$ : **queue of**  $\mathbb{B}$   
**passive atom controls**  $z$  **reads**  $cons, trans_R$  **awaits**  $cons, trans_R$   
**init**  
 $\parallel true \rightarrow z' := \emptyset$   
**update**  
 $\parallel cons? \wedge trans_R? \rightarrow z' := Enqueue(ack', Dequeue(z))$   
 $\parallel cons? \wedge \neg trans_R? \rightarrow z' := Dequeue(z)$   
 $\parallel \neg cons? \wedge trans_R? \rightarrow z' := Enqueue(ack', z)$   
**lazy atom controls**  $cons, x$  **reads**  $pc, z$   
**init**  
 $\parallel true \rightarrow x' := 0$   
**update**  
 $\parallel pc = wait \wedge z \neq \emptyset \wedge x = Front(z) \rightarrow cons!$   
 $\parallel pc = wait \wedge z \neq \emptyset \wedge x \neq Front(z) \rightarrow cons!; x' := \neg x$   
**passive atom controls**  $pc$  **reads**  $pc, cons, trans_S$  **awaits**  $cons'$   
**init**  
 $\parallel true \rightarrow pc' := send$   
**update**  
 $\parallel pc = send \wedge trans_S? \rightarrow pc' := wait$   
 $\parallel pc = wait \wedge cons? \rightarrow pc' := send$   
**lazy atom controls**  $trans_S, msg, abp$  **reads**  $pc, trans_S, x$   
**update**  
 $\parallel pc = send \rightarrow trans_S!; abp' := x; msg' := \mathbb{M}$

```

module ABPReceiver is
  external  $trans_S: \mathbb{E}; abp: \mathbb{B}; msg: \mathbb{M}$ 
  interface  $trans_R: \mathbb{E}; ack: \mathbb{B}$ 
  private  $cons: \mathbb{E}; pc: \{send, wait\}; x: \mathbb{B}; z: \mathbf{queue\ of\ } \mathbb{B}$ 
  passive atom controls  $z$  reads  $cons, trans_S$  awaits  $cons, trans_S$ 
  init
     $\parallel true \rightarrow z' := \emptyset$ 
  update
     $\parallel cons? \wedge trans_S? \rightarrow z' := Enqueue(abp', Dequeue(z))$ 
     $\parallel cons? \wedge \neg trans_S? \rightarrow z' := Dequeue(z)$ 
     $\parallel \neg cons? \wedge trans_S? \rightarrow z' := Enqueue(abp', z)$ 
  lazy atom controls  $cons, x$  reads  $pc, z$ 
  update
     $\parallel pc = wait \wedge z \neq \emptyset \rightarrow cons!; x' := Front(z)$ 
  passive atom controls  $pc$  reads  $pc, cons, trans_R$  awaits  $cons, trans_R$ 
  init
     $\parallel true \rightarrow pc' := wait$ 
  update
     $\parallel pc = send \wedge trans_R? \rightarrow pc' := wait$ 
     $\parallel pc = wait \wedge cons? \rightarrow pc' := send$ 
  lazy atom controls  $trans_R, ack$  reads  $pc, trans_R, x$ 
  update
     $\parallel pc = send \rightarrow trans_R!; ack' := x$ 

```

```

module AbstractSender is
  interface  $trans_S: \mathbb{E}; abp: \mathbb{B}$ 
  external  $trans_R: \mathbb{E}$ 
  private  $pc: \{send, wait\}; x: \mathbb{B}$ 
  passive atom controls  $pc$  reads  $pc, trans_S, trans_R$ 
  awaits  $trans_R, trans_S$ 
  init
     $\parallel true \rightarrow pc' := send$ 
  update
     $\parallel pc = send \wedge trans_S? \rightarrow pc' := wait$ 
     $\parallel pc = wait \wedge trans_R? \rightarrow pc' := send$ 
  lazy atom controls  $x, trans_S, abp$  reads  $pc, x, trans_S$ 
  init
     $\parallel true \rightarrow x' := 0$ 
  update
     $\parallel pc = send \rightarrow trans_S!; abp' := x; x' := \neg x$ 

```

- ABPSender does not implement AbstractSender
- $ABPSender \parallel AbstractReceiver \preceq^L AbstractSender$

```

module AbstractReceiver is
  external  $trans_S: \mathbb{E}$ 
  interface  $trans_R: \mathbb{E}; ack: \mathbb{B}$ 
  private  $pc: \{send, wait\}; x: \mathbb{B}$ 
  passive atom controls  $pc$  reads  $pc, trans_S, trans_R$ 
    awaits  $trans_R, trans_S$ 
  init
     $\parallel true \rightarrow pc' := wait$ 
  update
     $\parallel pc = send \wedge trans_R? \rightarrow pc' := wait$ 
     $\parallel pc = wait \wedge trans_S? \rightarrow pc' := send$ 
  lazy atom controls  $x, trans_R, ack$  reads  $pc, x, trans_R$ 
  init
     $\parallel true \rightarrow x' := 0$ 
  update
     $\parallel pc = send \rightarrow trans_R!; ack' := x; x' := \neg x$ 

```

- ABPReceiver does not implement AbstractReceiver
- $ABPReceiver \parallel AbstractSender \preceq^L AbstractReceiver$

## From Bisimilarity to Implementation

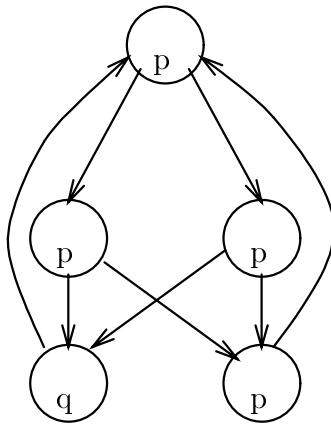
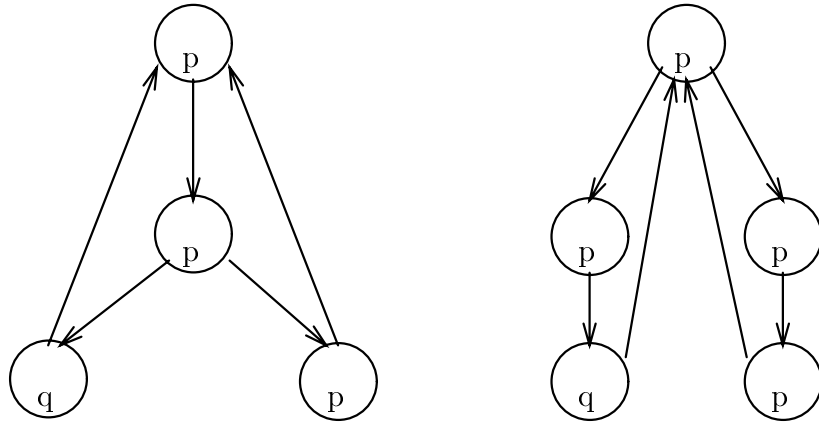
- Bisimilarity is an equivalence on states. How do we use it to compare structures?
- Let  $K_1$  and  $K_2$  be two disjoint observation structures, consider  $K_1 + K_2$
- $K_1 \preceq^B K_2$  if for all states  $s \in \sigma_1^I$ , there is a state  $t \in \sigma_2^I$  such that  $s \preceq_{K_1+K_2}^B t$ .
- How do we lift bisimilarity to compare modules?
- For two modules  $P$  and  $Q$ ,  $P \preceq^B Q$  if
  - every interface variable of  $Q$  is an interface variable of  $P$
  - every external variable of  $Q$  is an observable variable of  $P$ ,
  - for all variables  $x$  in  $\mathbf{obs}X_Q$  and  $y$  in  $\mathbf{intf}X_Q$ , if  $y \prec_Q x$  then  $y \prec_P x$
  - $K_{P'} \preceq^B K_Q$  for  $P' = \mathbf{hide} (\mathbf{obs}X_P \setminus \mathbf{obs}X_Q) \mathbf{in} P$ .

- In general, every state preorder  $\preceq$  gives a preorder on modules
- Theorem: If  $P \preceq^B Q$  then  $P \preceq^L Q$ .
- Heuristic: Instead of establishing  $P$  implements  $Q$ , try to establish  $P \preceq^B Q$ .
- Obstacle: bisimilarity is too strong, does not allow implementation to be more constrained than the specification
- Examples:
  - Sync3BitCounter and Sync3BitCounterSpec are bisimilar
  - Scheduler  $\not\preceq^B$  NonDetScheduler does not hold

## Simulation

- State preorder between bisimilarity and trace
- A simulation  $\preceq \subseteq \Sigma^2$  of  $K$  is a binary relation on the state space such that for all states  $s$  and  $t$  of  $K$ , if  $s \preceq t$  then
  - $s$  and  $t$  have same observation
  - if  $s \rightarrow s'$ , then there is a state  $t'$  such that  $t \rightarrow t'$  and  $s' \preceq t'$ .
- For two simulation relations  $\preceq_1$  and  $\preceq_2$ , their union  $\preceq_1 \cup \preceq_2$  is a simulation relation.
- The union of all simulation relations is the maximal simulation relation  $\preceq_K^S$
- $s \preceq_K^S t$  means  $t$  simulates  $s$
- The maximal simulation  $\preceq^S$  is reflexive-transitive
- Simulation game: adversary cannot switch sides

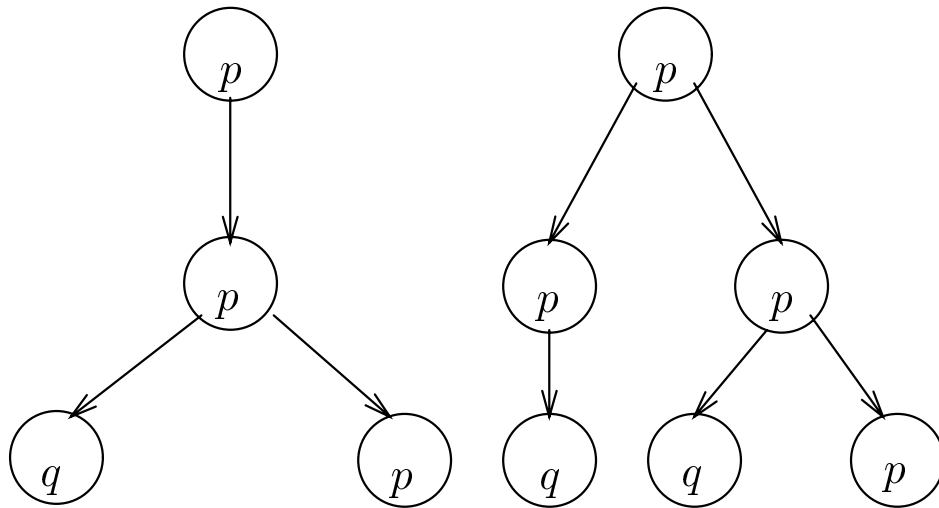
# Simulation Game



## Simulation Continued

- Simulation implies language inclusion:
  - If  $s \preceq^S t$ , then  $s \preceq^L t$ .
  - If  $K_1 \preceq^S K_2$ , then  $K_1 \preceq^L K_2$ .
- $s \preceq^L t$  does not imply  $s \preceq^S t$
- Similarity: equivalence induced by simulation.  
 $s \simeq^S t$  if  $s$  simulates  $t$  and  $t$  simulates  $s$
- Does similarity imply bisimilarity?
- NO! In similarity game adversary has to choose a side at the beginning, and stick to it
- Similarity is
  - Less distinguishing than bisimilarity
  - More distinguishing than trace equivalence
- If  $K$  is deterministic then all three (trace, similarity, bisimilarity) equivalences coincide

# Similarity Game



## Simulation Preorder over Modules

- Simulation preorder  $\preceq^S$  gives a way of comparing modules
- If  $P \preceq^S Q$  then  $P \preceq^L Q$
- Is simulation a good heuristic to establish implementation?
- Yes. It still has the essence of refinement: whatever  $P$  does is allowed by  $Q$
- Scheduler  $\preceq^S$  NonDetScheduler. Simulation relation: projection (omit the value of *prior*)
- Advantage: easier to check than implementation

## Logic and Simulation

- Does simulation equivalence have a logical characterization?
- Consider STL without switching of universal and existential quantifiers
- The formulas of  $\forall\text{Stl}$  are generated by the grammar

$$\phi ::= p \mid \phi \vee \phi \mid \phi \wedge \phi \mid \forall \bigcirc \phi \mid \phi \forall \mathcal{W} \phi.$$

- Thm. If  $s \preceq^S t$  and  $t \models \phi$  for a formula  $\phi$  of  $\forall\text{Stl}$  then  $s \models \phi$
- Thm. If two states are not similar, then they can be distinguished by a formula of  $\forall\text{Stl}$  (using next's suffices).
- $\forall\text{Stl}$  is a fully abstract semantics for similarity
- If  $P \preceq^S Q$  then  $P$  inherits  $\forall\text{Stl}$  properties of  $Q$

- Analogous results hold for the existential fragment  $\exists\text{Stl}$
- Checking simulation requires quadratic time
- Summary:

State Equivalence	Complexity	Logic
Trace equivalence $\simeq^L$	$O(m \cdot 2^n)$ /Pspace	Sal
Similarity $\simeq^S$	$O(m \cdot n)$	$\forall\text{Stl}, \exists\text{Stl}$
Bisimilarity $\simeq^B$	$O(m \cdot \log n)$	Stl