

CIS 673: Lecture 12: Oct 18
Logic and Simulation

- Does simulation equivalence have a logical characterization?
- Consider STL without switching of universal and existential quantifiers
- The formulas of \forall 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 ϕ of \forall STL then $s \models \phi$
- Thm. If two states are not similar, then they can be distinguished by a formula of \forall STL (using next's suffices).
- \forall STL is a fully abstract semantics for similarity

- If $P \preceq^S Q$ then P inherits \forall STL properties of Q
- Analogous results hold for the existential fragment \exists 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 STL, \exists STL
Bisimilarity \simeq^B	$O(m \cdot \log n)$	STL

Computing Similarity

- For a state s , the simulator set $sim(s)$ of s is the set of states that simulate s .
- Similarity Checking problem: Given K , compute the simulator sets for all reachable states
- Enumerative algorithms: If n is number of states and m is number of transitions, then best-known algorithm is $O(mn)$.
- We consider symbolic algorithms

Symbolic Similarity

- Two similar states have identical simulator sets
- Simulator set of a state is a block of the similarity partition
- Hence, instead of mapping states to simulator sets, map equivalence classes of a partition to blocks of the partition
- Symbolic simulator structure
 - Similarity partition \simeq^S
 - Function Sim that maps a region in \simeq^S to a block of \simeq^S

Schematic Algorithm1

Input: a finite observation structure K

Output: for each state s , the simulator set $sim(s)$.

foreach $s \in \Sigma$ do

$sim(s) := \{t \in \Sigma \mid \langle\langle t \rangle\rangle = \langle\langle s \rangle\rangle\}$

while \exists three states t , s , and u such that

$s \in post(t)$, $u \in sim(t)$, and $post(u) \cap$

$sim(s) = \emptyset$ do

$sim(t) := Delete(u, sim(t))$

{I0: for all $s, t \in \Sigma$, if t simulates s then
 $t \in sim(s)$ }

od.

Schematic Algorithm2

foreach $s \in \Sigma$ do
 $sim(s) := \{t \in \Sigma \mid \langle\langle t \rangle\rangle = \langle\langle s \rangle\rangle\}$
while \exists three states s, t , and u such that
 $post(t) \cap sim(s) \neq \emptyset$, $u \in sim(t)$, and
 $post(u) \cap sim(s) = \emptyset$ do
 {I4: for all $s \in \Sigma$, $s \in sim(s)$ }
 {I5: for all $s, t, u \in \Sigma$, if $t \preceq^S u$ and
 $t \in sim(s)$, then $u \in sim(s)$ }
 $sim(t) := Delete(u, sim(t))$
od.

Symbolic Algorithm

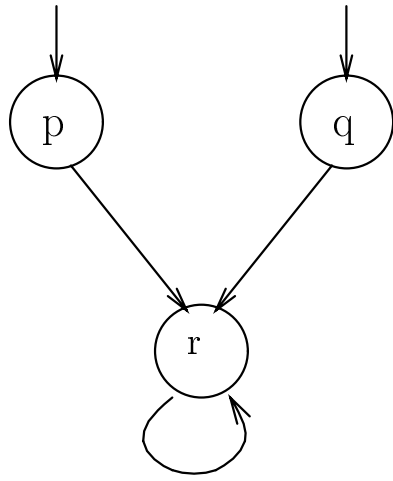
```

 $\cong := \{\Sigma_a \mid a \in A \text{ and } \Sigma_a \neq \emptyset\};$ 
foreach  $\sigma \in \cong$  do  $Sim(\sigma) := \sigma$  od;
while  $\exists$  regions  $\sigma, \tau \in \cong$  such that
     $\sigma \cap pre(Sim(\tau)) \neq \emptyset$  and
     $Sim(\sigma) \setminus pre(Sim(\tau)) \neq \emptyset$  do
    {I6: for  $\sigma \in \cong$  and  $s \in \sigma$ ,  $sim(s) =$ 
         $Sim(\sigma)$ }
    {I7: for  $\sigma \in \cong$ , both  $\sigma$  and  $Sim(\sigma)$  are
        blocks of  $\simeq^S$ }
     $\sigma' := \sigma \cap pre(Sim(\tau))$ 
     $\sigma'' := \sigma \setminus pre(Sim(\tau));$ 
     $\cong := Insert(\sigma', Delete(\sigma, \cong));$ 
     $Sim(\sigma') := Sim(\sigma) \cap pre(Sim(\tau));$ 
    if  $\sigma'' \neq \emptyset$  then
         $\cong := Insert(\sigma'', \cong)$ 
         $Sim(\sigma'') := Sim(\sigma)$  od.
    
```

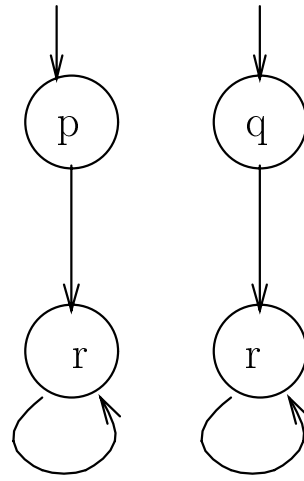
Refinement mappings

- Refinement maps (or homomorphisms) are special types of simulation relations
- A refinement mapping hom from K_1 to K_2 is a function from the reachable region σ_1^R of K_1 to Σ_2 such that
 1. for each reachable state s , $\langle\langle \text{hom}(s) \rangle\rangle_2 = \langle\langle s \rangle\rangle_1$,
 2. for every reachable transition $s \rightarrow_1 t$ of K_1 ,
 $\text{hom}(s) \rightarrow_2 \text{hom}(t)$
 3. for all $s \in \sigma_1^I$, $\text{hom}(s) \in \sigma_2^I$
- If there exists a refinement mapping from the observation structure K_1 to K_2 then $K_1 \preceq^S K_2$.
- Existence of refinement map is a sufficient, but not necessary, condition for simulation
- For Scheduler \preceq^S NonDetScheduler, there is a refinement map (project out *prior*)

Refinement vs. Simulation



K_1



K_2

Using Refinement Maps

- We wish to establish P implements Q
- Specify a mapping hom from states of P to states of Q
- Mapping specified component-wise: define the value of every specification variable in terms of implementation variables
- Task for the model checker: establish that hom is a homomorphism
- Can be done reachability analysis on the product $K_P \times K_Q$
- Current tool support: COSPAN, SMV, MOCHA

Stutter closed relations

- Stutter closure K^S of a structure K is obtained by adding extra transitions
- Stutter-closed trace preorder: $s \underline{\preceq}^L t$ if trace-language of s in K^S is contained in trace-language of t in K^S
- Alternatively, $s \underline{\preceq}^L t$, for every source- s trace \bar{a} there exists a source- t trace \bar{b} such that \bar{a} and \bar{b} have a common stutter extension
- Stutter extension of a word is obtained by repeating its symbols
- The stutter-closed trace preorder leads to stutter-closed (weak) implementation relation over modules
- Weak implementation is less distinguishing than implementation
- Modules SyncMsg and AsyncMsg are weak trace-equivalent

Asynchronous modules

- Asynchronous module can stutter in each round
- If \bar{a} is a trace of P then every stutter-extension of \bar{a} is also a trace of P
- Weak implementation is compositional wrt to \parallel as long as we consider only asynchronous modules:
 - For asynchronous modules P , Q , and R , if $P \underline{\preceq}^L Q$ and R is compatible with P , then R is compatible with Q and $P \parallel R \underline{\preceq}^L Q \parallel R$.
- Assume guarantee rule holds for weak-implementation if we consider only asynchronous modules

Weak similarity

- Stutter-closed version of similarity
- Corresponding logic: $\forall\text{STL}^{\mathcal{W}}$ (no next)

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

Weak Trace Equivalence \cong^L	\sqsubseteq	Trace Equivalence \simeq^L
\sqcap		\sqcap
Weak Similarity \cong^S	\sqsubseteq	Similarity \simeq^S
\sqcap		\sqcap
Weak Bisimilarity \cong^B	\sqsubseteq	Bisimilarity \simeq^B

Course Logistics

- No lectures on 10/23,25,30
- Homework 2: Due Nov 1
- Course project: presentation 12/11