Equivalence checking

Istvan Majzik
majzik@mit.bme.hu

Budapest University of Technology and Economics
Dept. of Measurement and Information Systems

Intro: Equivalence and refinement checking in model based design

Refining statechart models

Properties expected from refinement relations

Introduction: Relations between models

- Equivalence between models:

  Reference model ↔ Modified model

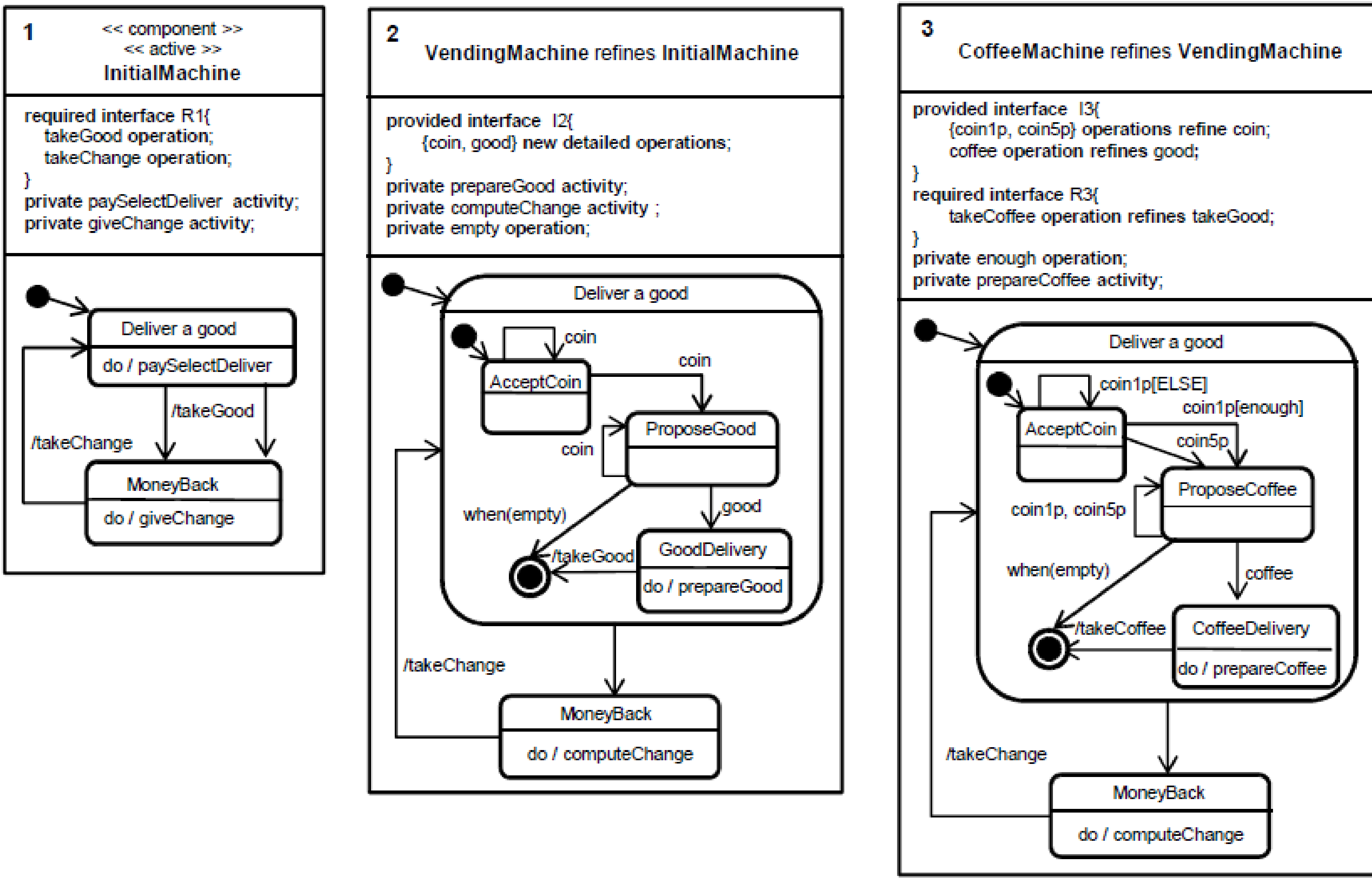
  Specification (abstract) ↔ Implementation (concrete, more detailed)

  Expected behavior ↔ Provided behavior (e.g., protocol)

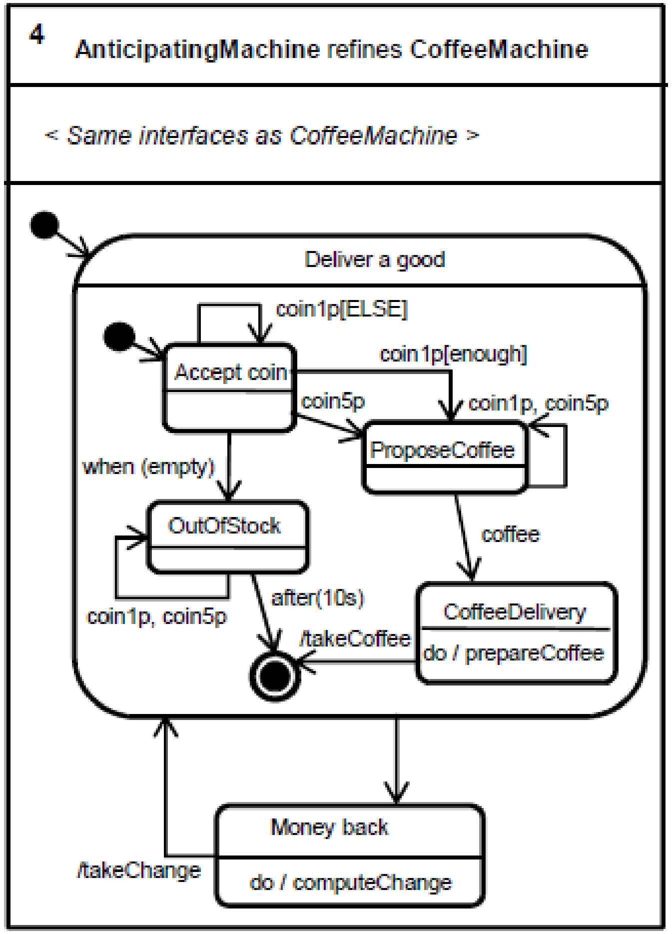
  Fault-free “perfect” system ↔ Fault tolerant system in case of specific faults

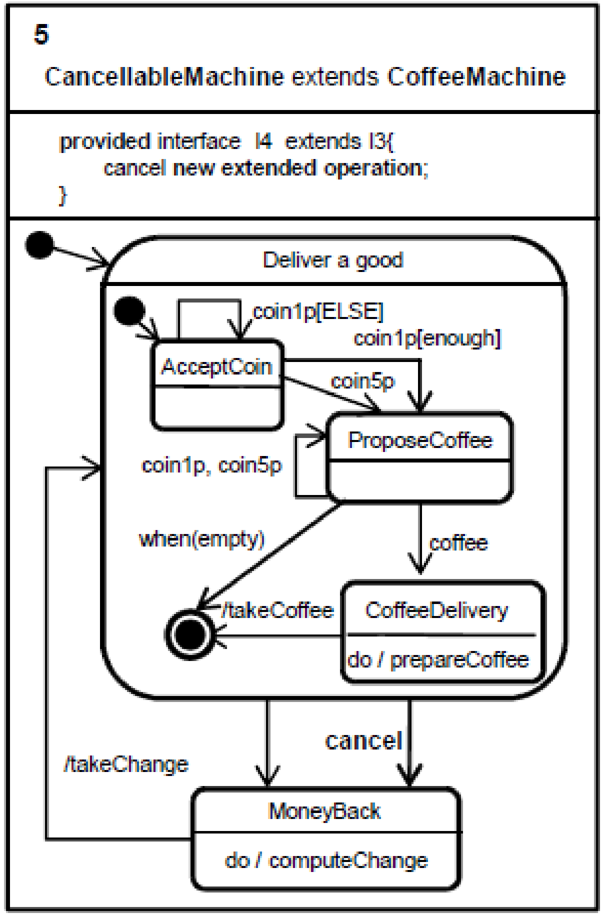
- Refinement between models:
  - Preserving original behavior and extending it in an allowed way
  - Reducing nondeterminism in the model (with concrete conditions)

Example: Refinement in statechart models (1)

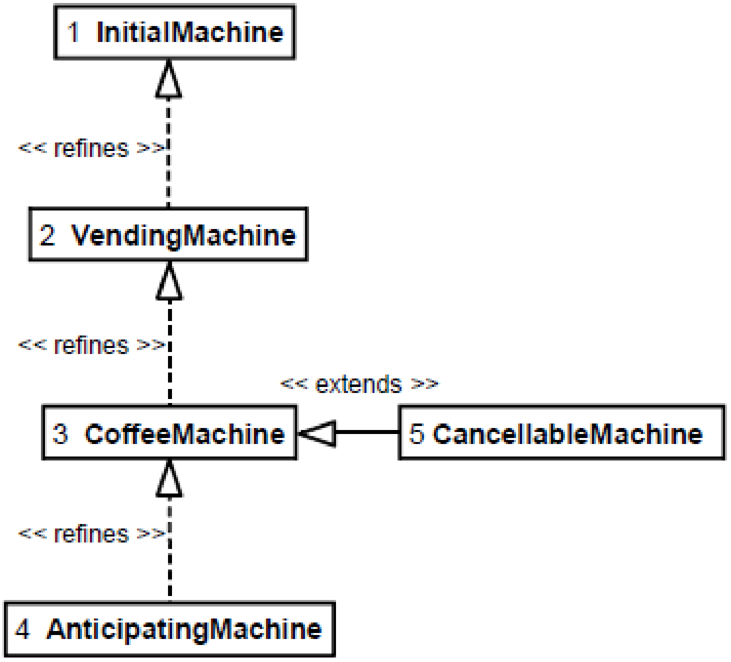


Example: Refinement in statechart models (2)





What is expected: Checking well-defined relations



- "Refines" relation to keep existing behavior (with proper mapping of events and actions) with refinements
- "Extends" relation to allow controlled changes in existing behavior

What do we expect from a refinement relation?

- Reflexive and transitive
- Not symmetric
- Keeping liveness property: The refined model shall be able to provide the behavior that the original model is able to provide
  - With proper mapping of events and actions of the refined model
  - Assuming fairness: Keeping the liveness property in case of fair behavior (i.e., in case of choices, all potential behaviors will eventually occur)
- Composability:
  - Subsequent refinements result in refinement
  - Refinement and extension result in extension
- …

Precise definitions of the relations are required!

Definition of the relations

- Relations are defined on low-level models, typically on LTS
- Recap: LTS (Labeled Transition System)

$$LTS = (S, Act, \rightarrow)$$

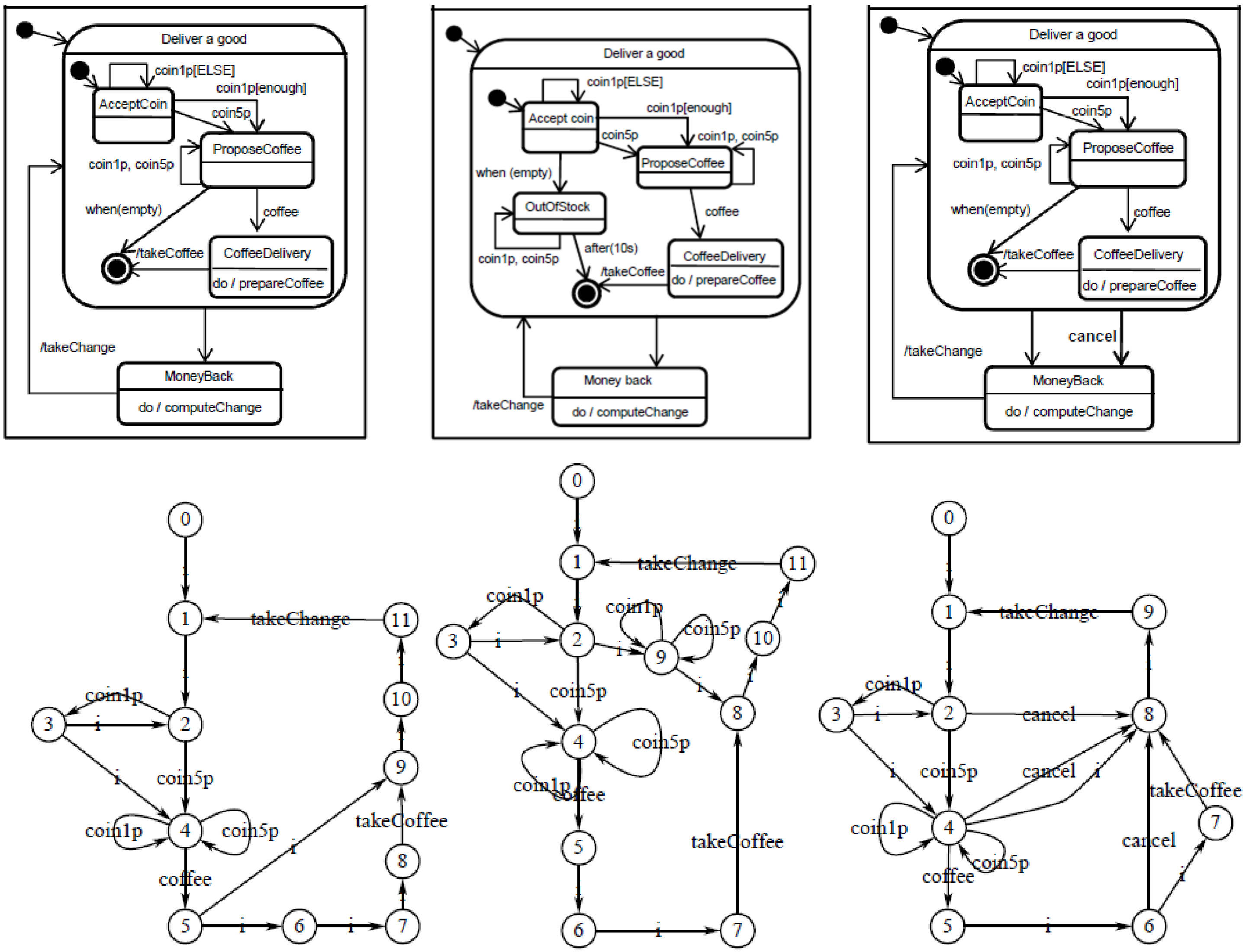
S set of states

Act set of actions

$\rightarrow \subseteq S \times Act \times S$ state transition relation

- LTS may be derived from higher-level formalisms (using operational semantics)
  - E.g., statecharts, Petri-nets, process algebra, …

Example: Mapping LTS from statechart



a. LTS_3: CoffeeMachine

b. LTS_4: AnticipatingMachine

c. LTS_5: CancellableMachine

Equivalence relations

Trace equivalence

Strong bisimulation equivalence

Weak bisimulation (observational) equivalence

Classification of relations

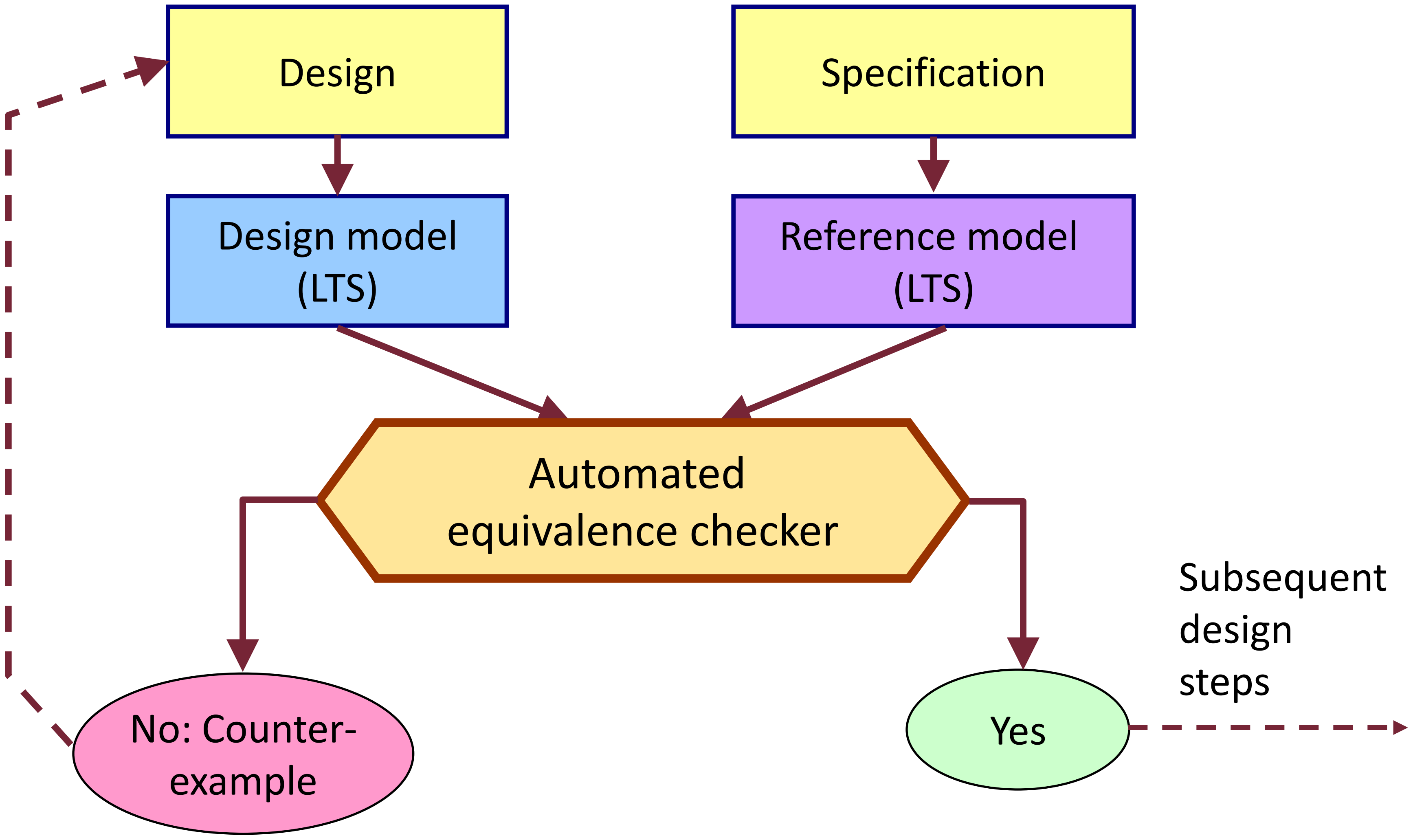
- Equivalence relations, denoted in general by =
  - Reflexive, transitive, symmetric

  Some equivalence relations are congruence:
  - If T1=T2, then for all C[ ] context C[T1]=C[T2]
  - The same context preserves the equivalence
  - Dependent on the formalism: how to embed in C[ ]

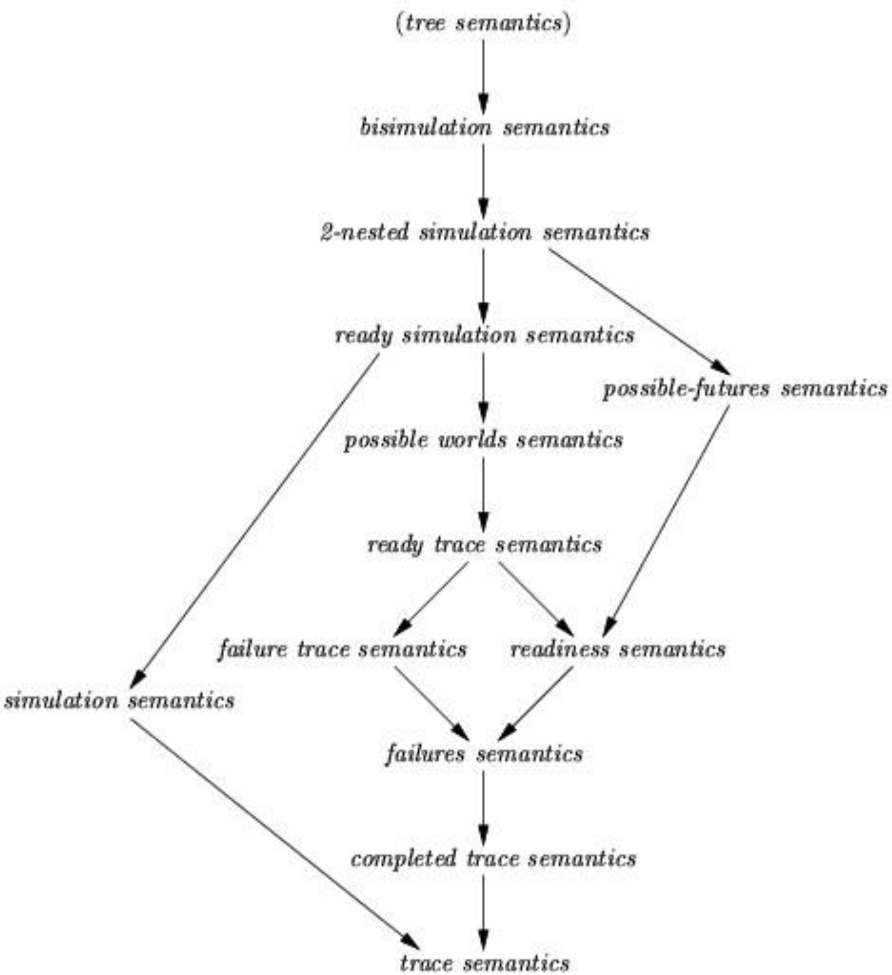
- Refinement relations, denoted by ≤
  - Reflexive, transitive, anti-symmetric (→ partial order)

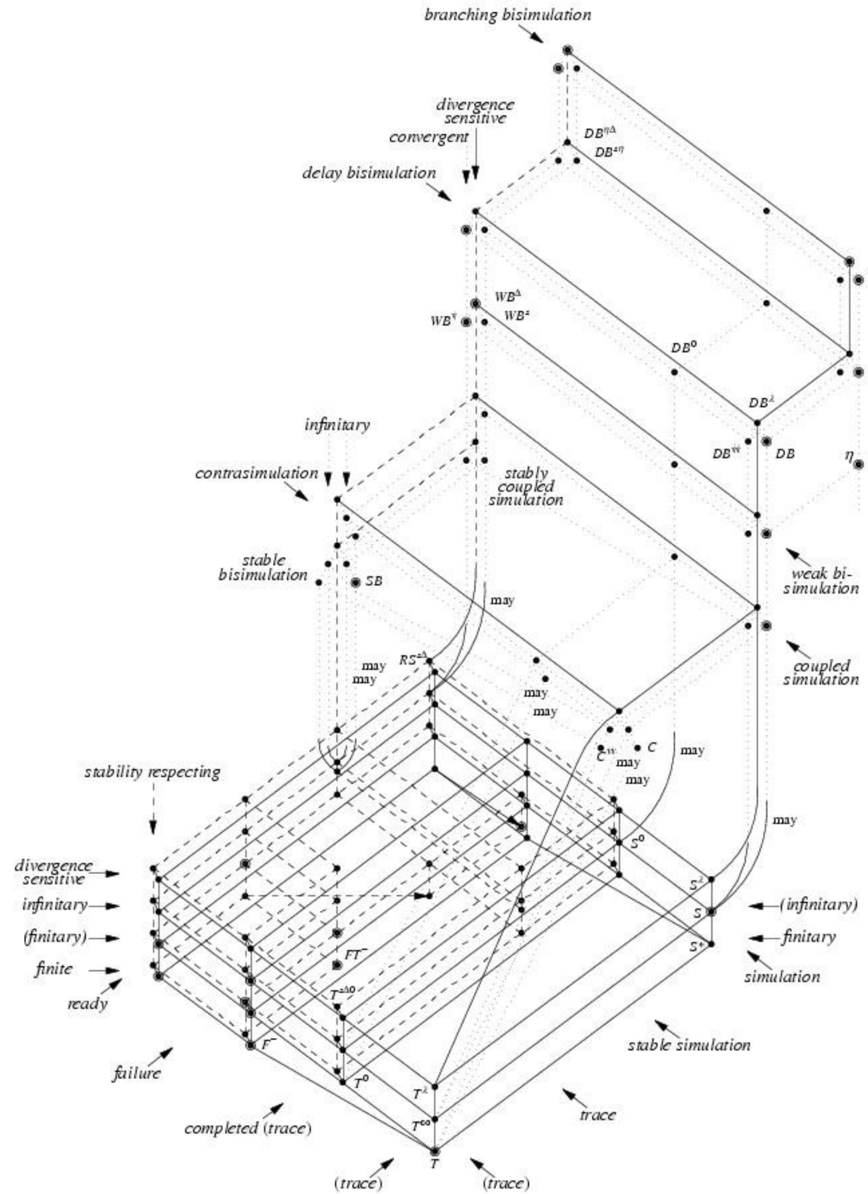
  Precongruence relation:
  - If T1≤T2, then for all C[ ] context C[T1] ≤ C[T2]
  - The same context preserves the refinement

Equivalence checking using an equivalence relation



Hierarchy of relations proposed in the literature



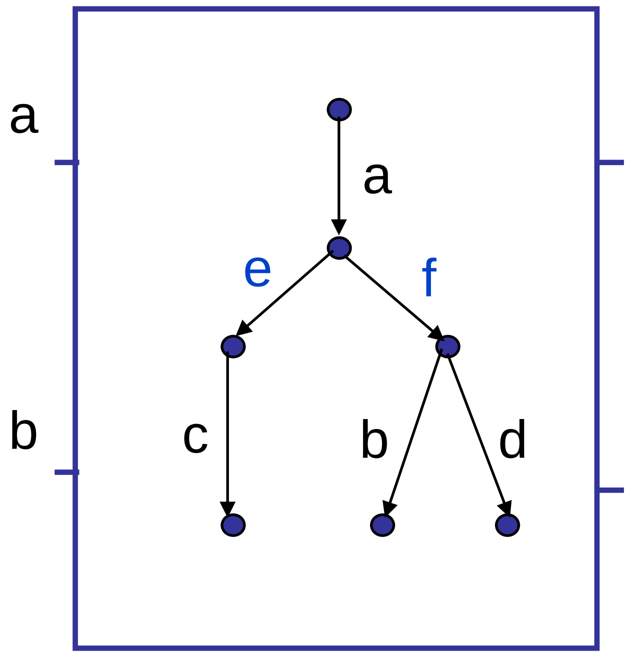


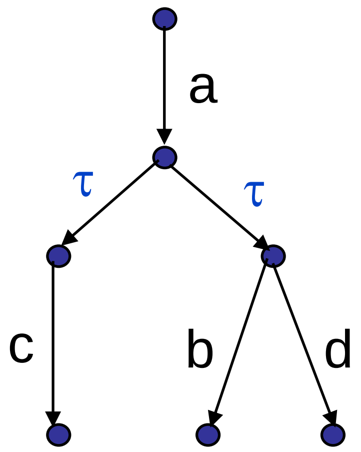
Why do exist so many relations?

Properties that characterize the relations

- Distinguishing observable and internal actions:
  - Observable actions: Appear on the external interface (“ports”) of the modeled component, relevant for the environment
    - Representing: method call, sent or received message, provided service etc.
  - Unobservable (internal) actions: Do not appear on the external interface (“ports”) of the modeled component, not relevant for the environment
    - Representing: internal activities, calls, messages etc.
    - Their effects can be observed only through the subsequent actions
    - Notation: τ (or sometimes i)

Example: Internal actions





Internal behavior
of the component:
e and f are internal actions

Observable behavior
of the component:
e and f are mapped to τ

Properties that characterize the relations

- Distinguishing observable and internal actions:
  - Observable actions: Appear on the external interface (“ports”) of the modeled component, relevant for the environment
    - Representing: method call, sent or received message, provided service etc.
  - Unobservable (internal) actions: Do not appear on the external interface (“ports”) of the modeled component, not relevant for the environment
    - Representing: internal activities, calls, messages etc.
    - Their effects can be observed through the subsequent actions
    - Notation: τ (or sometimes i)
- Allowing nondeterminism:
  - From a state, many transitions are labeled with the same action
    - „Image finite system”: their number is finite
  - Typically used in abstract models, resolved during refinement
- Semantics of concurrent component models:
  - Interleaving (one action at a time)
  - True concurrency (several actions at a time)

The notion of “test” and “deadlock”

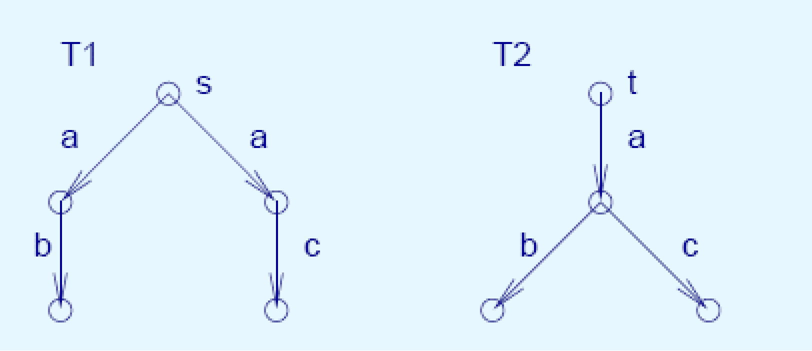
- “Test” in LTS based behavior checking:
  - Test: A sequence of actions that is expected (from the initial state)
    - Analogy: interactions on ports during testing
    - Test steps are actions that may represent: sending or receiving messages, raising or processing events etc.
- “Deadlock” in LTS based behavior checking:
  - A given action cannot be provided by the system in an expected action sequence (test)
    - Analogy: no interaction is possible on a port
    - The deadlock is given by the action that is not possible; it may represent that is not possible to send or receive message, process an event etc.
  - Failure of a test: The action that cannot be provided (deadlock)

  - Example: Piano with keys that can be dynamically locked/unlocked
    - Behavior: unlocking is determined by the actions of the LTS
    - Successful test is a tune that can be played

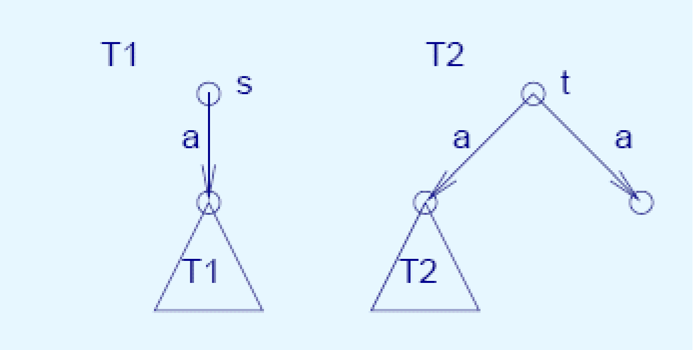
Examples for deadlocks

- What is deadlock after action a?

Act={a, b, c}

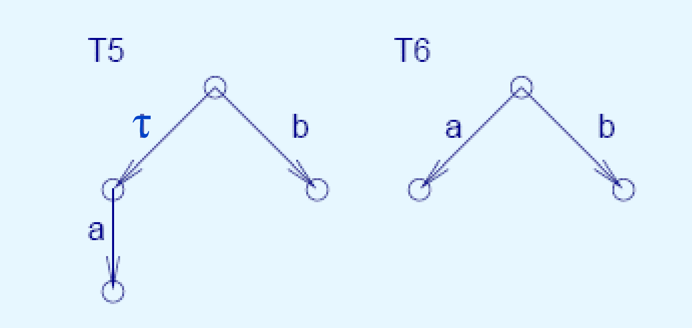


„Recursive” LTS model, Act={a}

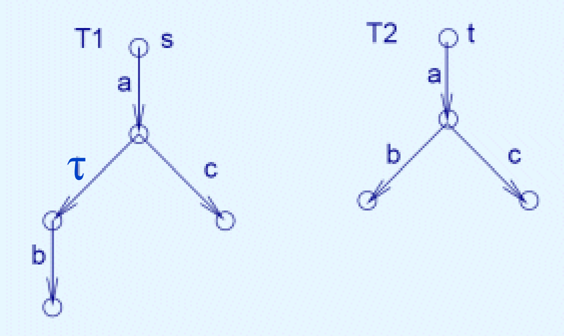


- How internal actions influence deadlock?

Act={a, b, c}



Act={a, b, c}



Trace equivalence: Notation

- Analogy: Automata on finite words

$$A_1 = A_2 \ \text{ if } \ L(A_1) = L(A_2)$$

  - Applying this analogy in case of LTS:
    - Each state is an “accepting state”
    - ”Language”: Each possible action sequence (trace)

- Notation:

$$\alpha = a_1 a_2 a_3 a_4 \ldots a_n \in Act^* \ \text{ finite action sequence } (\varepsilon \text{ is empty})$$

$$s \xrightarrow{\alpha} s' \ \text{ if } \exists \mathrm{s}_0 \mathrm{s}_1 \ldots \mathrm{s}_\mathrm{n} \ \text{ state sequence where } \mathrm{s}_0 = s,\ \mathrm{s}_\mathrm{n} = s',\ s_i \xrightarrow{a_{i+1}} s_{i+1}$$

$$\alpha(s) \text{ is a trace from s, if } \exists s' : s \xrightarrow{\alpha} s'$$

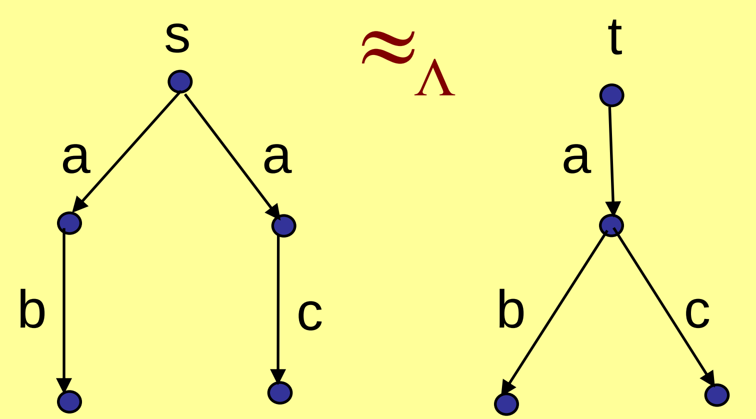
$$\Lambda(s) \text{ is the set of traces from s: } \ \Lambda(s) = \left\{ \alpha \mid \exists s' : s \xrightarrow{\alpha} s' \right\}$$

Trace equivalence: Definition and examples

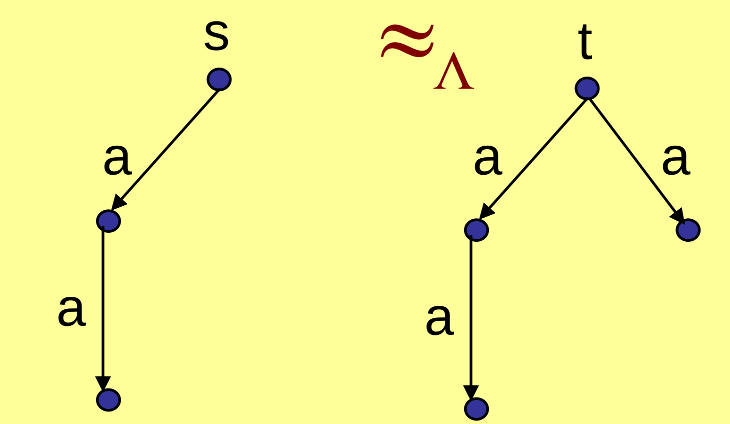
- Let T_1 and T_2 two LTS, s_1 and s_2 their initial states
- Definition of trace equivalence $\approx_\Lambda$:

$$T_1 \approx_\Lambda T_2 \ \text{ iff. } \ \Lambda(s_1)=\Lambda(s_2)$$

- Examples:



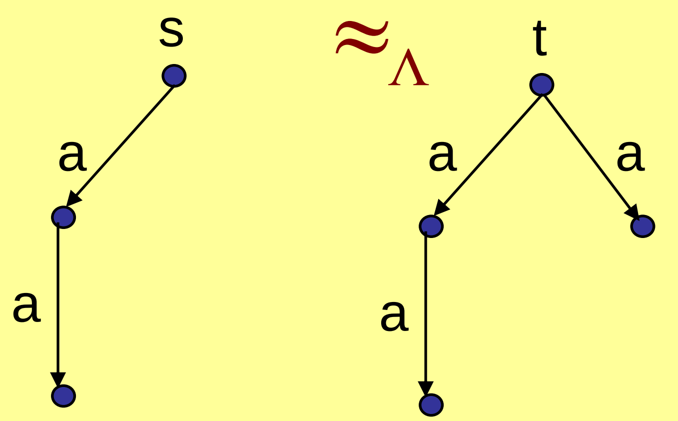
$$\Lambda(s) = \Lambda(t) = \{\varepsilon, a, ab, ac\}$$



$$\Lambda(s) = \Lambda(t) = \{\varepsilon, a, aa\}$$

Trace equivalence: Disadvantages

- Sensitivity to deadlock
  - Equivalent LTSs may have different deadlock behavior
  - Caused by nondeterminism or internal actions



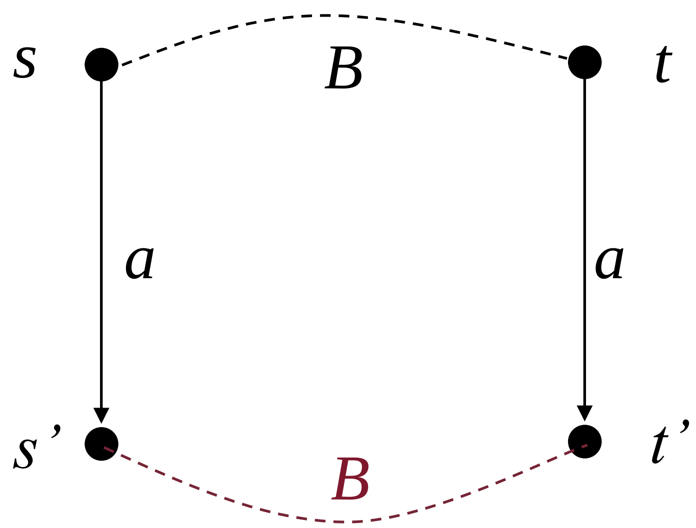
$$\Lambda(s) = \Lambda(t) = \{\varepsilon, a, aa\}$$

- Solution:
  - It has to be checked whether the states reached by the same trace allow the same continuation of the trace

- Definition of the strong bisimulation relation:

$B \subseteq S \times S$ is a bisimulation, if for all $(s,t) \in B$ and any $a \in Act,\ s',t' \in S$ it holds:

- if $s \xrightarrow{a} s'$ then $\exists t' : t \xrightarrow{a} t'$ and $(s',t') \in B$
- if $t \xrightarrow{a} t'$ then $\exists s' : s \xrightarrow{a} s'$ and $(s',t') \in B$

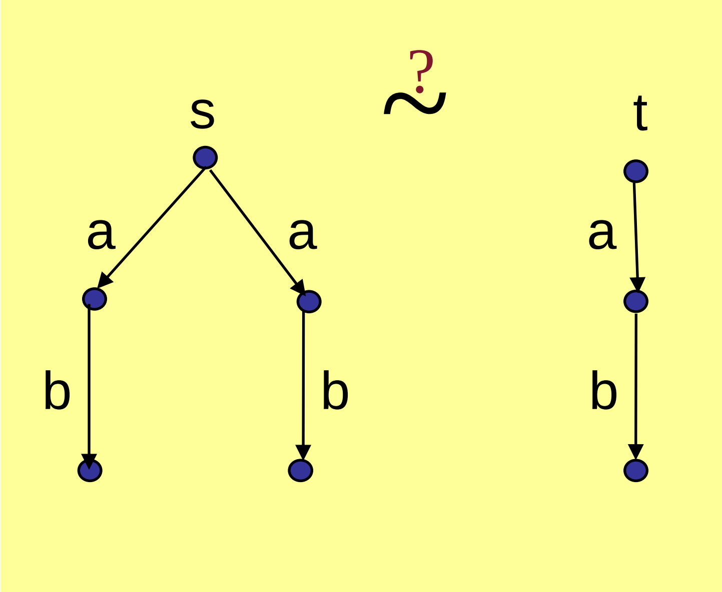


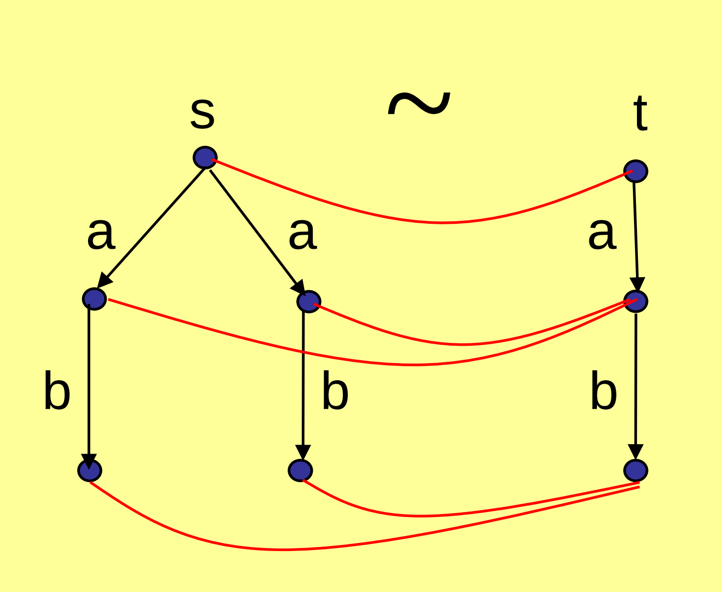
Strong bisimulation equivalence: Definition

- Strong bisimulation equivalence $\sim$:

$$T_1 \sim T_2 \;\; \text{iff} \;\; \exists B : (s_1, s_2) \in B, \;\; \text{also denoted as } s_1 \sim s_2$$

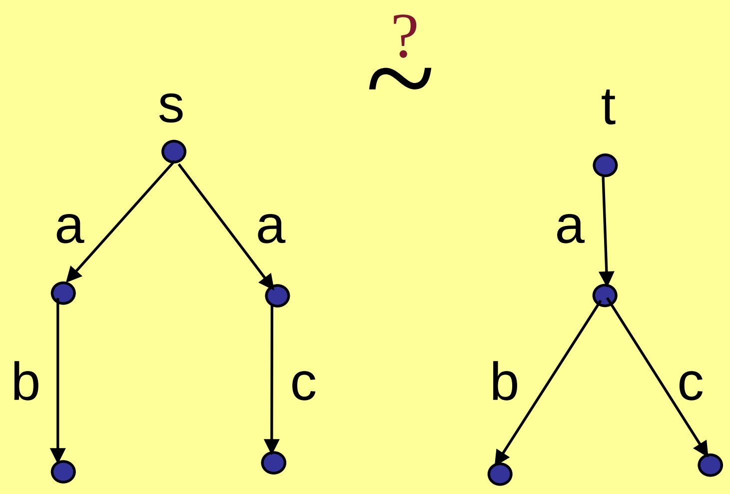
- Intuition: Equivalent systems can “simulate” each other
  - Matching transitions with actions in equivalent states
  - The same traces are possible through equivalent states
- Example (1):



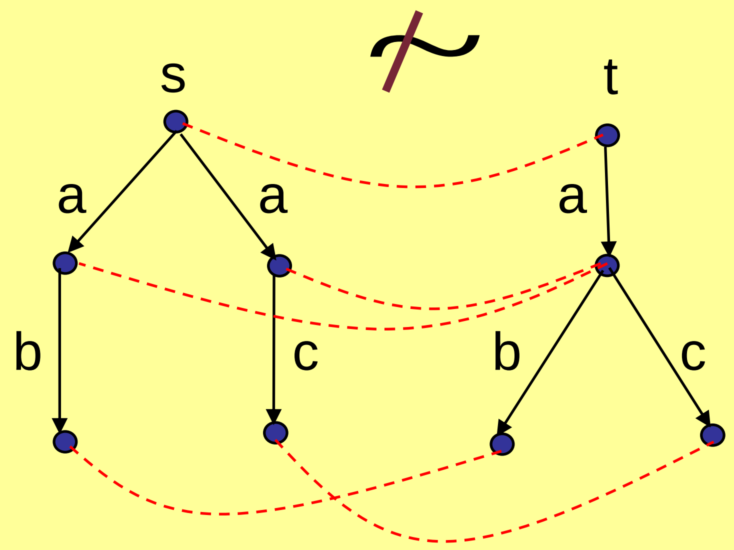


Strong bisimulation equivalence: Example (2)

- Strong bisimulation equivalence between LTSs:



$$\Lambda(s) = \Lambda(t) = \{\varepsilon, a, ab, ac\}$$



$$\Lambda(s) = \Lambda(t) = \{\varepsilon, a, ab, ac\}$$

MŰEGYETEM 1782

Strong bisimulation equivalence: Advantages

- Strong bisimulation implies trace equivalence
- Congruence for specific “CCS-like” LTS
  - Recap: An equivalence relation is congruence if the same context preserves the equivalence:
    - Here in case of T1~T2, for all C[ ] context C[T1] ~ C[T2]
  - “CCS-like” LTS:
    - LTS has a tree structure
    - Embedding an LTS: merging initial state of the embedded LTS T_i with any state of the context LTS C[] to get $C[T_i]$
- Strong bisimulation equivalent systems provide the same deadlock behavior
  - If T1~T2 then if deadlock is possible in LTS T_1 then the same deadlock is possible in LTS T_2

Strong bisimulation equivalence: Formalizing deadlock

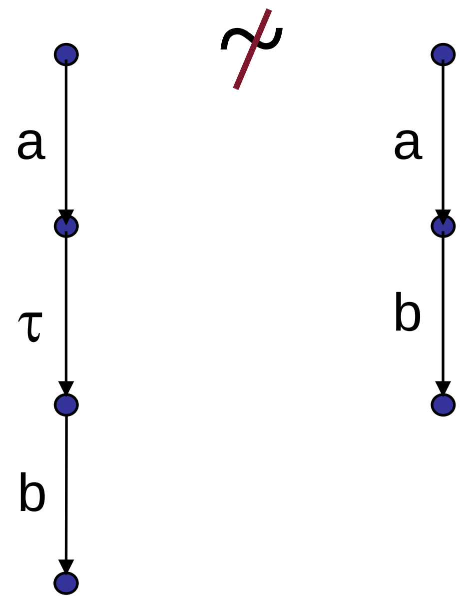
- Deadlock possibilities can be expressed using the Hennessy-Milner logic:
  - Deadlock for action a is expressed as $[a]\text{false}$
    - It holds only if there is no transition labeled with a, i.e., a is deadlock
  - Deadlock for a set of actions $\{a_1, a_2, \ldots a_n\}$:

    $$\{[a_1]\text{false} \wedge [a_2]\text{false} \wedge \ldots \wedge [a_n]\text{false}\}$$
  - Deadlock for a set of actions in a state reachable by $\langle b_1\rangle\langle b_2\rangle\ldots\langle b_n\rangle$:

    $$\langle b_1\rangle\langle b_2\rangle\ldots\langle b_n\rangle\ \{[a_1]\text{false} \wedge [a_2]\text{false} \wedge \ldots \wedge [a_n]\text{false}\}$$
- Theorem:
  In case of LTSs, $T_1 \sim T_2$ iff for any HML expression p:
  - either $T_1, s_1 \models p$ and $T_2, s_2 \models p$,
  - or $T_1, s_1 \not\models p$ and $T_2, s_2 \not\models p$

Strong bisimulation equivalence: Disadvantages

- Sensitivity to unobservable actions:
  - In some cases there is no observable effect of an internal action, but the relation makes a difference
  - Simple example:



Weak bisimulation equivalence: Notation

- The "weak" variant of strong bisimulation
  - It is not sensitive to internal actions without observable effect
  - Rationale: Have the possibility of the same observable traces through equivalent states
- Notation:

$\alpha \in Act^*$ finite action sequence (ε is empty)

$\hat{\alpha} \in (Act - \tau)^*$ observable action sequence (τ deleted)

here $\hat{\alpha} = \varepsilon$ if $\alpha = \tau$

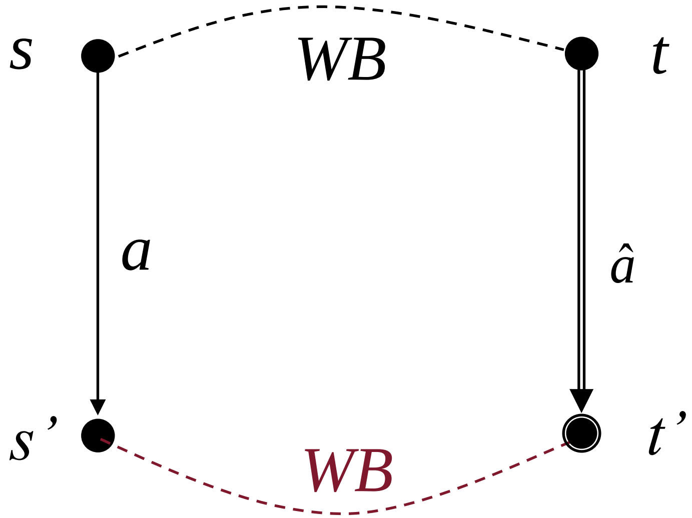
$s \overset{\beta}{\Rightarrow} s'$ if $\exists \alpha: s \overset{\alpha}{\rightarrow} s'$ and $\beta = \hat{\alpha}$

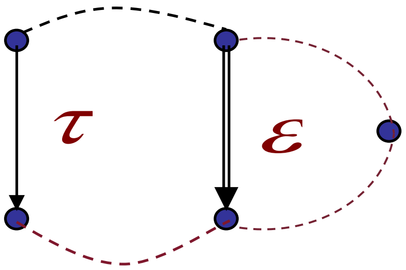
Weak bisimulation relation: Definition

- Definition of weak bisimulation relation:

$WB \subseteq S \times S$ weak bisimulation, if for all $(s,t) \in WB$ and any $a \in Act,\ s',t' \in S$ it holds:

- if $s \xrightarrow{a} s'$ then $\exists t' : t \xRightarrow{\hat{a}} t'$ and $(s',t') \in WB$
- if $t \xrightarrow{a} t'$ then $\exists s' : s \xRightarrow{\hat{a}} s'$ and $(s',t') \in WB$



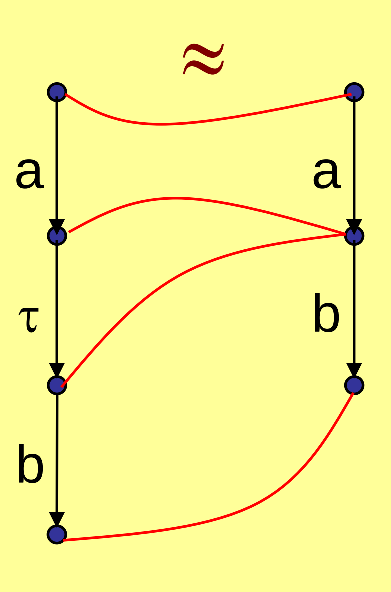


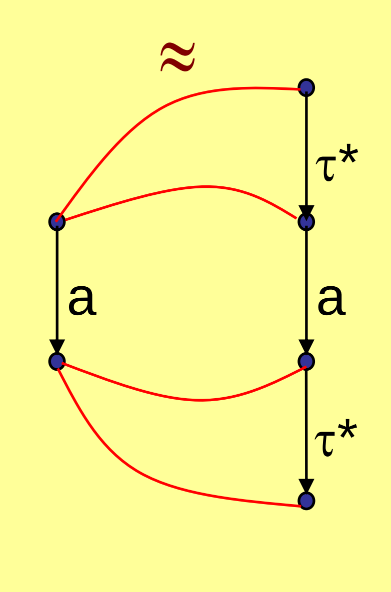
Weak bisimulation equivalence: Definition

- Weak bisimulation equivalence $\approx$ (also called as Observation equivalence)

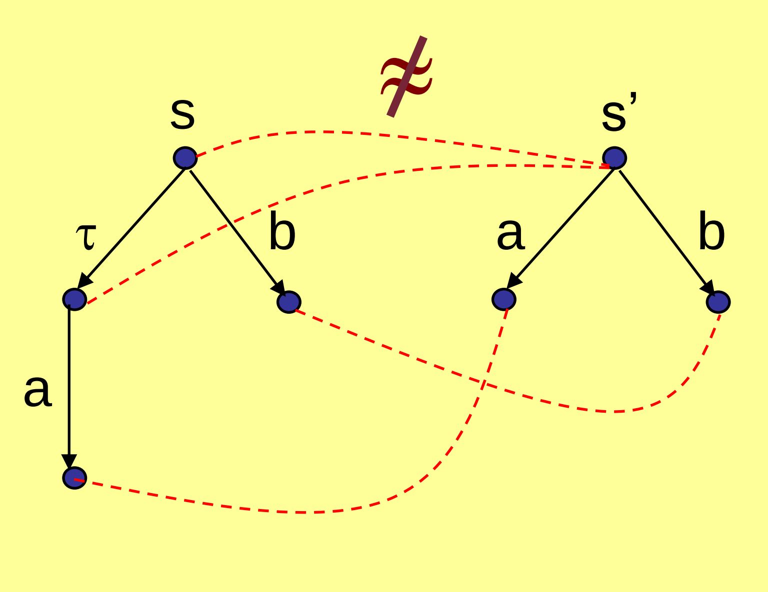
$$T_1 \approx T_2 \ \text{ iff } \ \exists WB : (s_1, s_2) \in WB, \ \text{also denoted as } s_1 \approx s_2$$

- Examples:





Internal action with effect:

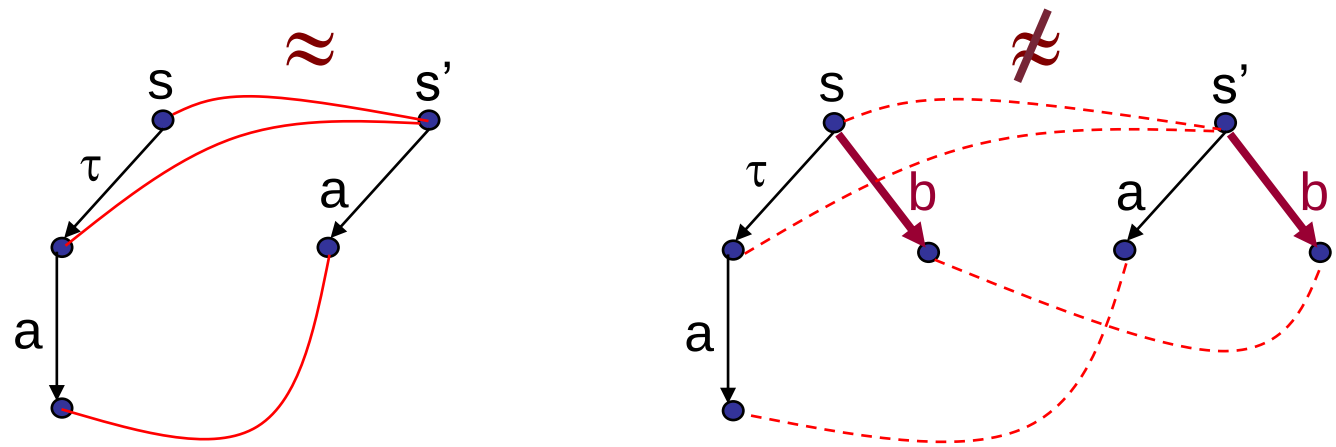


Weak bisimulation equivalence: Formalizing deadlock

- HML variant for observable actions:
  HML* ::= true | false | $p \wedge q$ | $p \vee q$ | [[a]] p | <<a>> p
- Semantics:
  - **H3***: $T,s \models [[a]]p$ iff $\forall s'$ where $s \Rightarrow^a s'$: $s' \models p$
  - **H4***: $T,s \models \langle\langle a \rangle\rangle p$ iff $\exists s'$: $s \Rightarrow^a s'$ and $s' \models p$
- Theorem:
  In case of LTSs, $T_1 \approx T_2$ iff for any HML* expression p:
  - either $T_1,s_1 \models p$ and $T_2,s_2 \models p$
  - or $T_1,s_1 \not\models p$ and $T_2,s_2 \not\models p$

Weak bisimulation equivalence: Properties

- It is not congruence for CCS-like LTSs (there is a counterexample):



- Interesting: The most permissive congruence relation, that implies weak bisimulation equivalence:

$$s \approx^c t, \text{ if for any } a \in Act,\ s', t' \in S \text{ it holds:}$$

- if $s \xrightarrow{a} s'$ then $\exists t' : t \overset{a}{\Rightarrow} t'$ and $s' \approx t'$
- if $t \xrightarrow{a} t'$ then $\exists s' : s \overset{a}{\Rightarrow} s'$ and $s' \approx t'$

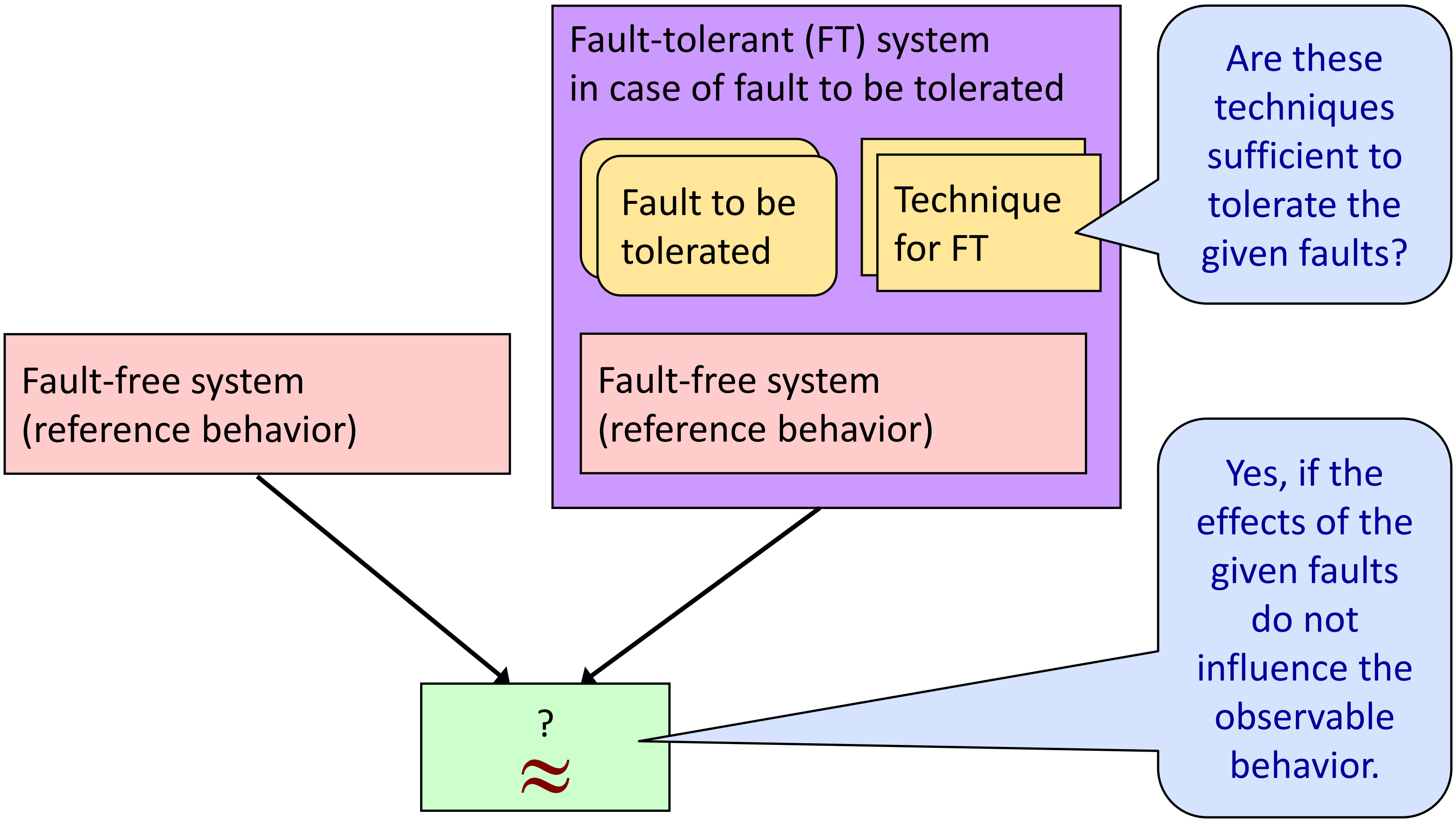
Computing equivalence relations: Basic idea

Partition refinement algorithm

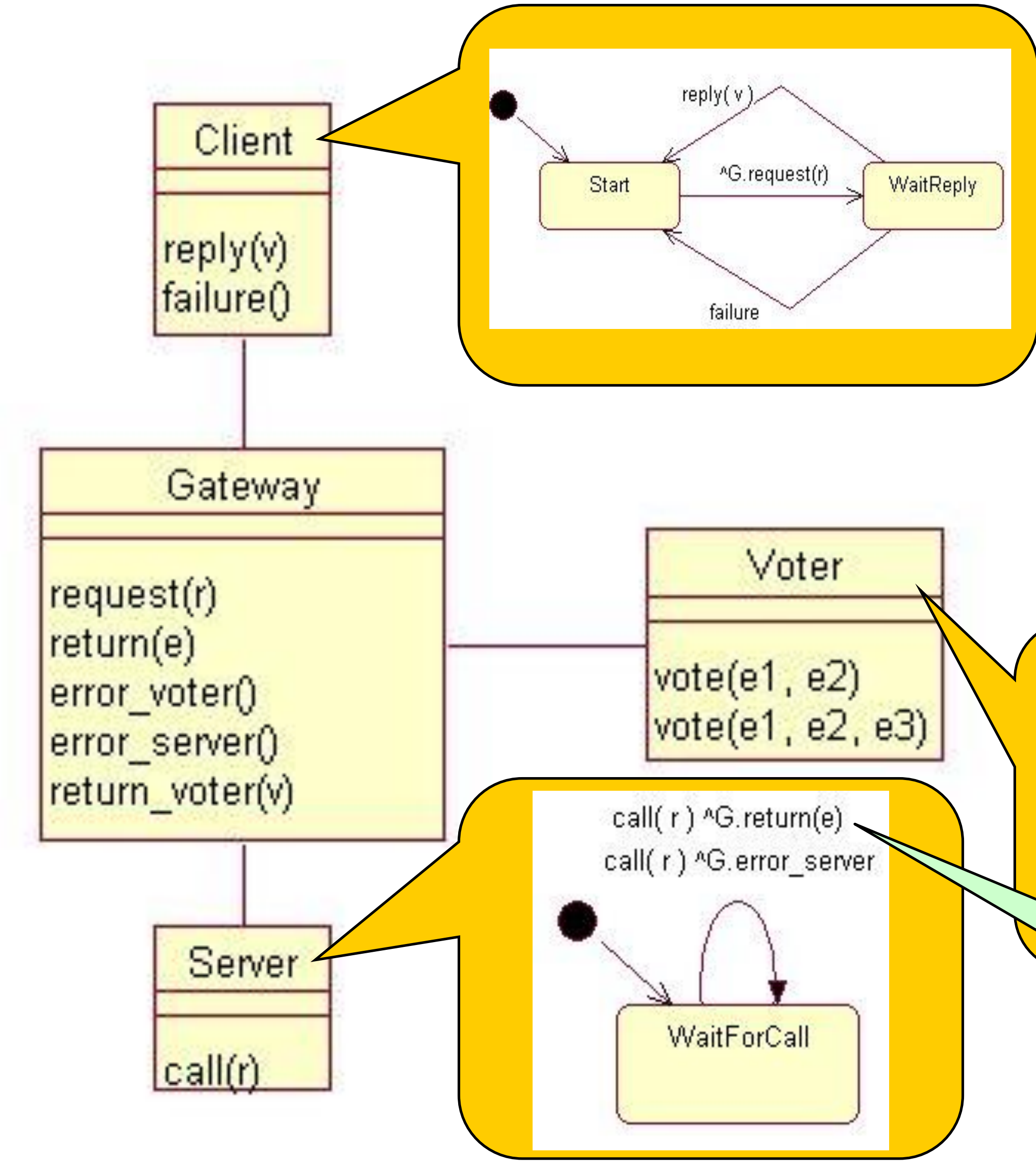
1. Initially, each pair of states is included in the relation
   They form a single partition (equivalence class)
2. For each pair of states, to be checked:
   If there is a labeled transition starting from one of the states
   that cannot be matched by a labeled transition from the other, then
   - Remove that state pair from the partition,
   - Apply the consequences for the state pairs at the sources of incoming matching transitions
     - Since these are not equivalent if the matching transitions lead to nonequivalent states
3. If there are no changes (fixed point is reached):
   The equivalence classes are found
   - If the initial states are in the same equivalence class then the LTSs are equivalent

Case study: Verification of fault tolerance using observation equivalence

Case study: Verification of fault tolerance



System architecture



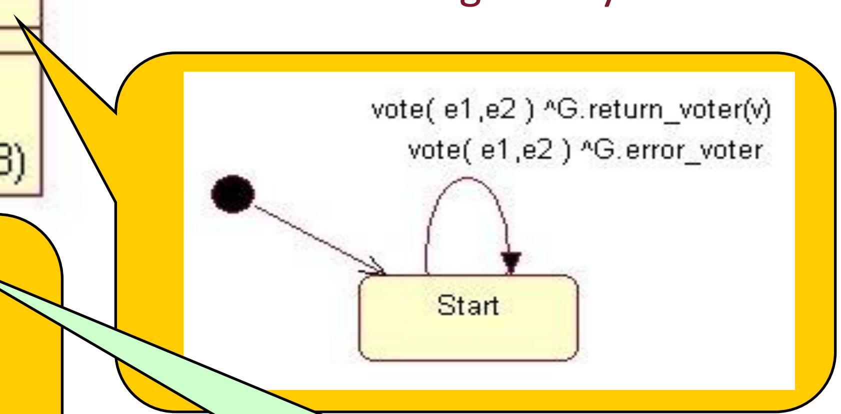
Fault to be tolerated:

Erroneous data in the reply from the server

FT technique: Redundant servers and voting

Behavior from the point of view of the client:

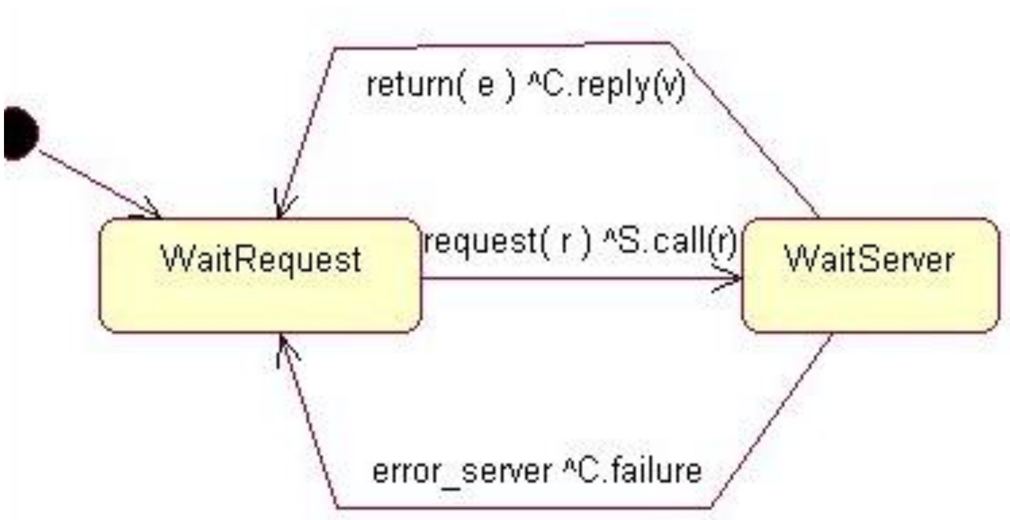
The observable behavior of the gateway



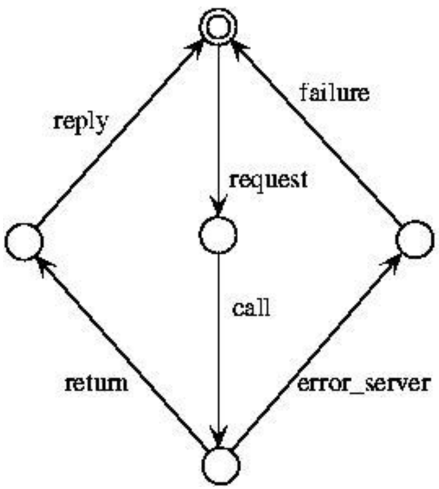
Here e may be erroneous

The Gateway component without fault tolerance

- Statechart diagram (reference behavior):

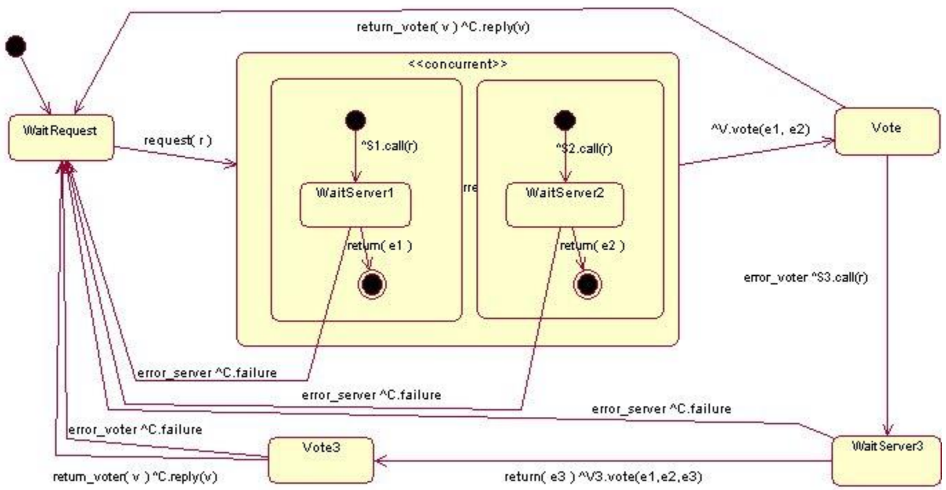


- LTS representation (reference behavior):

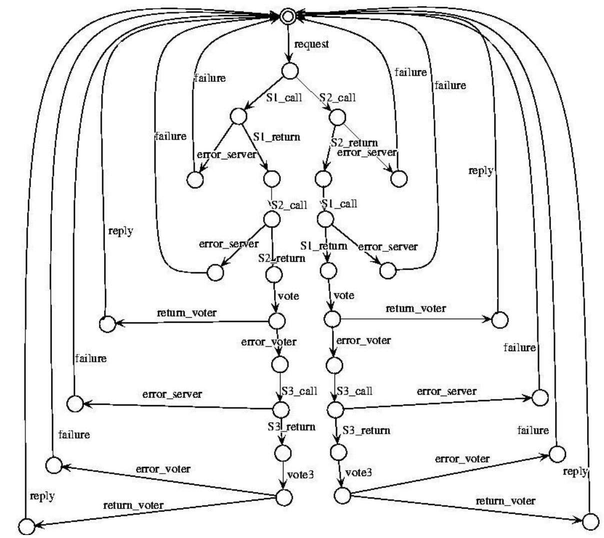


The Gateway component with fault tolerance

- Statechart diagram:

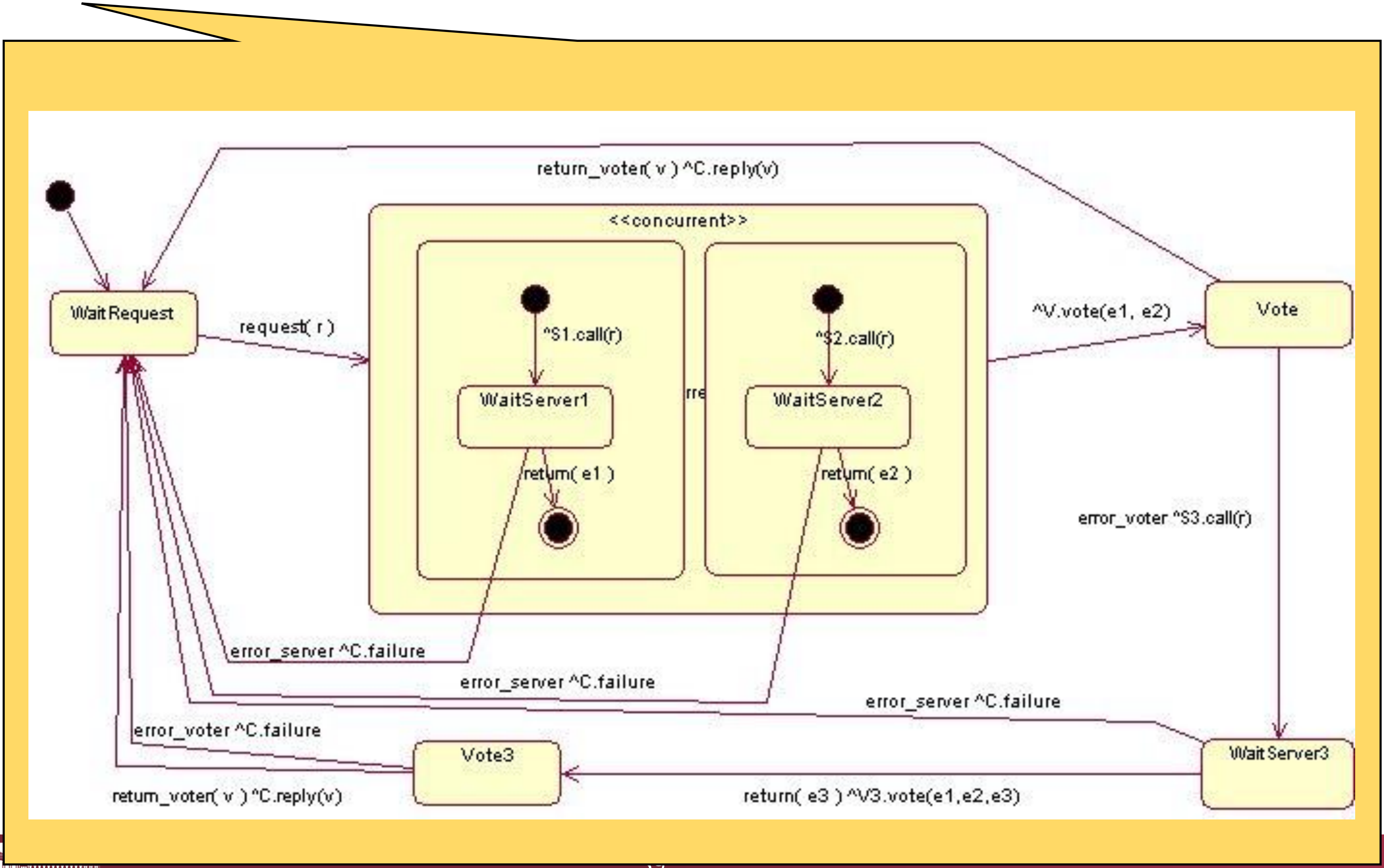


- LTS representation:



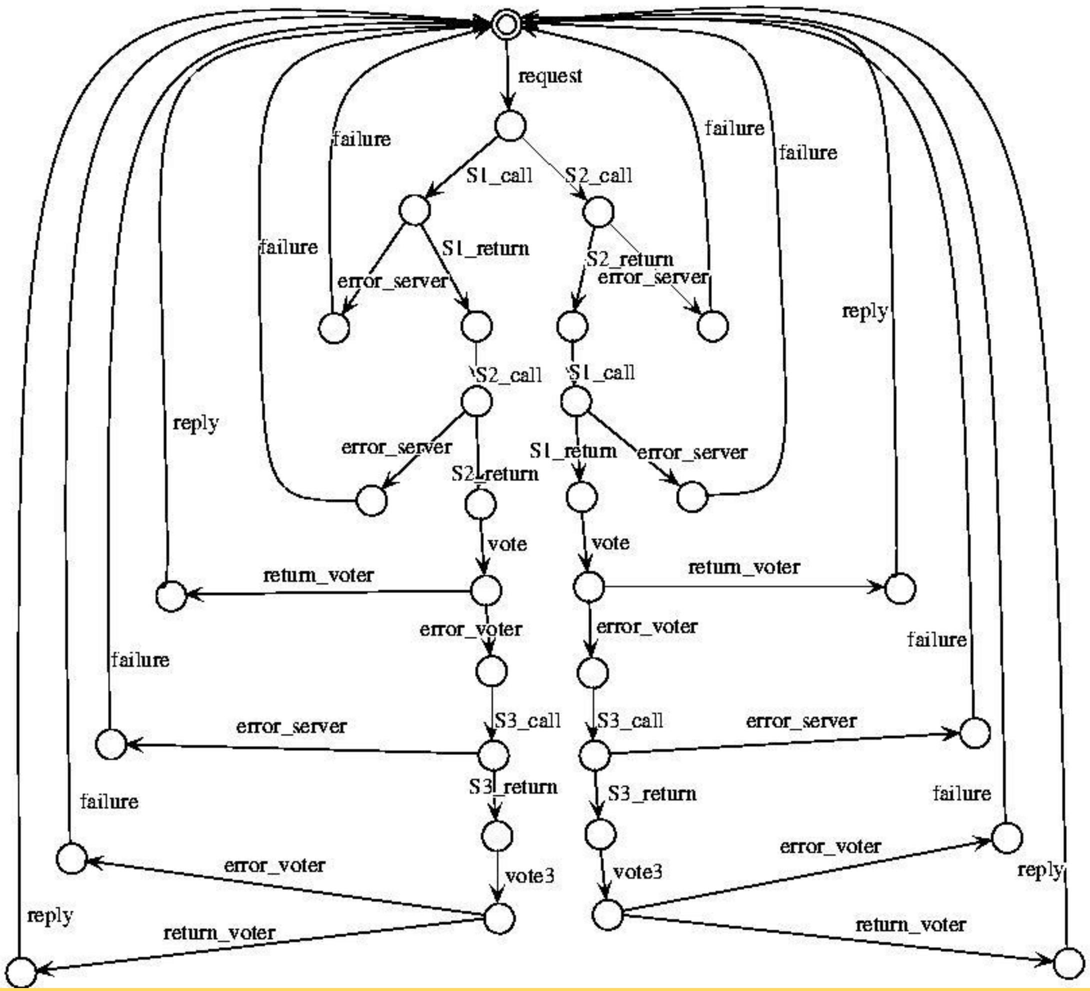
The Gateway component with fault tolerance

- Statechart diagram:
- LTS representation:

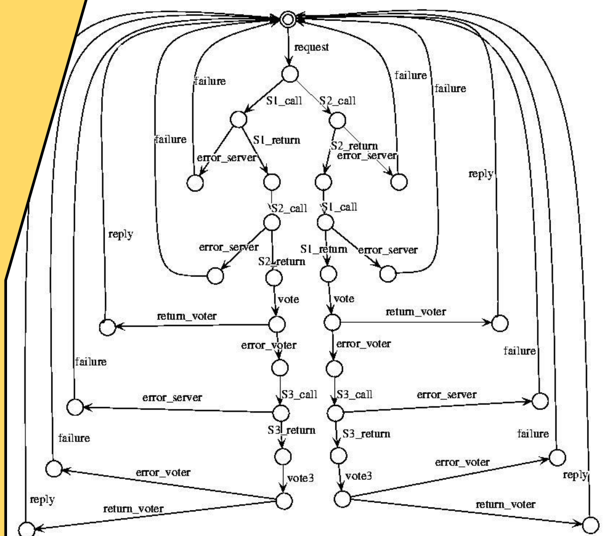


MŰEGYETEM 1782

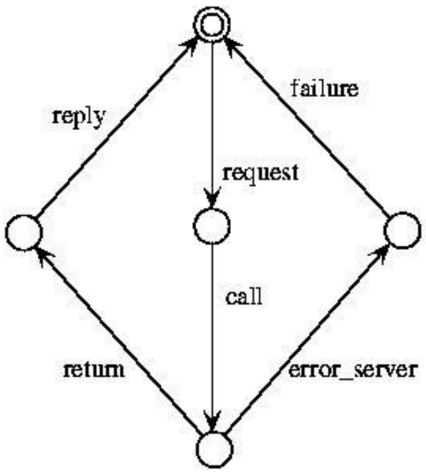
The Gateway component with fault tolerance



LTS representation:



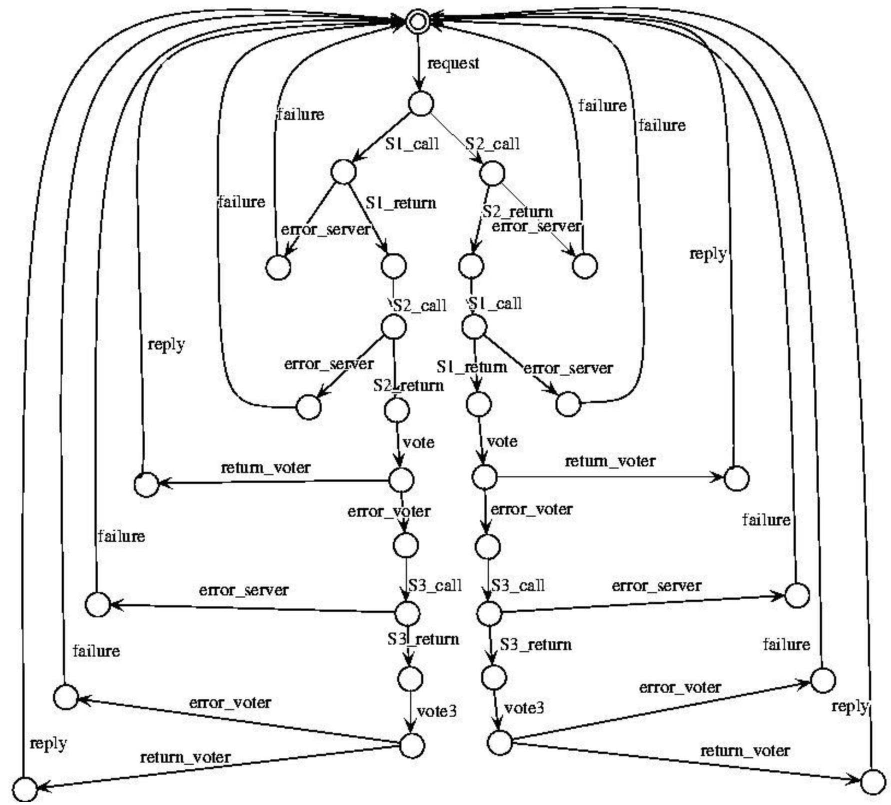
Checking observation equivalence



Reference behavior of the Gateway (without fault tolerance)

≈

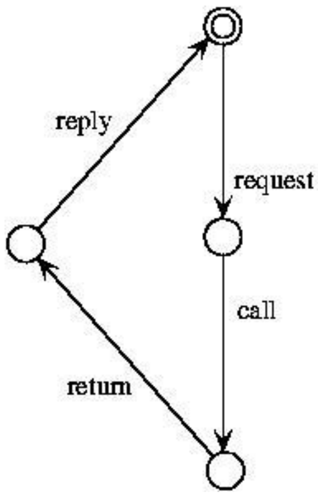
This way it is shown that for the client
the fault tolerance technique is transparent.



Behavior of the FT Gateway;
here each internal action that is not observable by the client becomes τ

MŰEGYETEM 1782

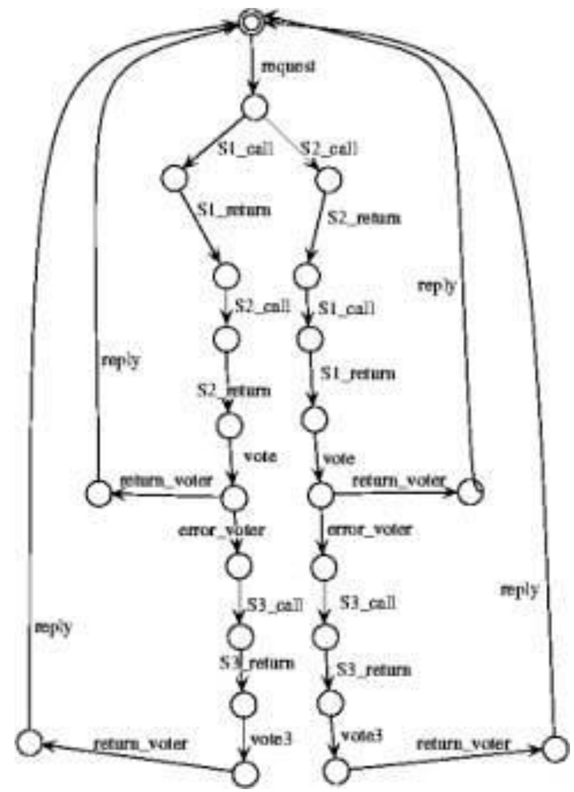
Checking fault tolerance in case of error from S1



Behavior of the fault-free gateway

≈

This way it is shown that for the client fault tolerance holds.



Behavior of the FT Gateway in case of error from S1 (voting and call of S3); here each internal action that is not observable by the client becomes τ

Summary

- Motivation and basic ideas
  - The role of behavioral equivalence and refinement
  - Observable and unobservable behavior
  - The notion of testing and deadlock
- Equivalence relations
  - Trace equivalence
  - Strong bisimulation equivalence
  - Weak bisimulation equivalence (observation equivalence)
- Case study
  - Verifying fault tolerance using observation equivalence
- (Refinement relations: See later)