State-Based Object Models Are More Abstract Than Trace-Based Models

Towards a Unified Specification Framework

Ilham W. Kurnia, Arnd Poetzsch-Heffter, Yannick Welsch

FoVeOOS 2010
Paris, France

June 29, 2010
Background

- State-based specification techniques
  - Close to implementation
  - Examples: Eiffel, JML, Spec#
1. Introduction

Background

- State-based specification techniques
  - Close to implementation
  - Examples: Eiffel, JML, Spec#
- Trace-based specification techniques
  - No need to think about states
  - Examples: Jass (trace assertions), Java Jr. (program equivalence)
This Talk is About ...

This talk is **not** about:

- Comparing specifications

![Diagram showing comparison between State-based spec and Trace-based spec](image-url)
This Talk is About ...

This talk is **not** about:
- Comparing specifications

This talk is about:
- Comparing models via abstractions
- Applying simulation theory to this context
This Talk is About ...

This talk is **not** about:

- Comparing specifications, however ...

This talk is about:

- Comparing models via abstractions
- Applying simulation theory to this context
1. Introduction

Our settings

- Object-based component
  A set of runtime objects.
- Sequential
  All method calls must fulfill call stack property
- Deterministic
  Given a state and a particular call, a component will always produce the same behavior.
Subject-Observer Pattern [GOF]

```java
interface Observer {
    void notify(State s);
}

class Subject {
    Observer o1, o2;

    Subject(Observer o1, Observer o2) {
        this.o1 = o1; this.o2 = o2;
    }

    void update(State s) {
        o1.notify(s);
        o2.notify(s);
    }
}
```
Subject-Observer Pattern [GOF]

```java
interface Observer {
    void notify(State s);
}

class Subject {
    Observer o1, o2;

    Subject(Observer o1, Observer o2) {
        this.o1 = o1; this.o2 = o2;
    }

    void update(State s) {
        o1.notify(s);
        o2.notify(s);
    }
}
```
Subject-Observer Pattern [GOF]

```java
interface Observer {
    void notify(State s);
}

class Subject {
    Observer o1, o2;

    Subject(Observer o1, Observer o2) {
        this.o1 = o1; this.o2 = o2;
    }

    void update(State s) {
        o1.notify(s);
        o2.notify(s);
    }
}
```
Subject-Observer Pattern [GOF]

```java
interface Observer {
    void notify(State s);
}

class Subject {
    Observer o1, o2;  // o1 ≠ o2 ≠ null
    Subject(Observer o1, Observer o2) {
        this.o1 = o1; this.o2 = o2;
    }
    void update(State s) {
        o1.notify(s);
        o2.notify(s);
    }
}
```


Subject-Observer Pattern [GOF]

```java
interface Observer {
    void notify(State s);
}

class Subject {
    Observer o1, o2;

    Subject(Observer o1, Observer o2) {
        this.o1 = o1; this.o2 = o2;
    }

    void update(State s) {
        o1.notify(s);
        o2.notify(s);
    }
}
```
Subject Component Behavior

\[
\begin{array}{l}
\text{u: User} \\
\text{sbj: S} \\
o1: O \\
o2: O
\end{array}
\]
Subject Component Behavior

2. Trace-Based vs. State-Based Models

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component Behavior

\[ u : \text{User} \rightarrow \text{update}(\text{state}) \rightarrow \text{notify}(\text{state}) \rightarrow \text{subj} : S \rightarrow o1 : O \rightarrow o2 : O \]
2. Trace-Based vs. State-Based Models

Subject Component Behavior

\[ u : \text{User} \]
\[ \text{subj} : S \]
\[ o1 : O \]
\[ o2 : O \]

\[ \text{update(state)} \] → \[ \text{notify(state)} \] → \[ \text{return} \]
Subject Component Behavior

\[ \text{u: User} \rightarrow \text{subj: S} \rightarrow \text{o1: O} \rightarrow \text{o2: O} \]

- update(state)
- notify(state)
- return
- notify(state)
Subject Component Behavior

\[ u : \text{User} \quad \text{subj} : \text{S} \quad o1 : 0 \quad o2 : 0 \]

- \text{update(state)}
- \text{notify(state)}
- \text{return}
- \text{notify(state)}
- \text{return}
2. Trace-Based vs. State-Based Models

Subject Component Behavior

u: User

update(state)

subj: S

notify(state)

return

o1: 0

notify(state)

return

o2: 0

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component Behavior

\begin{itemize}
\item \textbf{u: User}
\item \textbf{sbj: S}
\item \textbf{o1: O}
\item \textbf{o2: O}
\end{itemize}

- \texttt{update(state)}
- \texttt{notify(state)}
- \texttt{return}

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
2. Trace-Based vs. State-Based Models

Subject Component Behavior (Callback)

\[ u: \text{User} \rightarrow \text{subj: S} \rightarrow o1: O \rightarrow o2: O \]

- update(st)
- notify(st)
2. Trace-Based vs. State-Based Models

Subject Component Behavior (Callback)

\[ u : \text{User} \rightarrow \text{sbj} : S \rightarrow o1 : O \rightarrow o2 : O \]

- `update(st)`
- `notify(st)`
- `update(newSt)`
Subject Component Behavior (Callback)

- **u**: User
- **subj**: S
- **o1**: O
- **o2**: O

- `update(st)`
- `notify(st)`
- `update(newSt)`
- `notify(newSt)`
Subject Component Behavior (Callback)

\[ u: \text{User} \quad \text{subj: S} \quad o1: O \quad o2: O \]

- update(st)
- notify(st)
- update(newSt)
- notify(newSt)
- return
Subject Component Behavior (Callback)

\[ \text{update(st)} \]

\[ \text{notify(st)} \]

\[ \text{update(newSt)} \]

\[ \text{notify(newSt)} \]

\[ \text{return} \]

\[ \text{notify(newSt)} \]
Subject Component Behavior (Callback)

\[ u : \text{User} \]

\[ \text{update}(st) \rightarrow \text{notify}(st) \rightarrow \text{update}(\text{newSt}) \rightarrow \text{notify}(\text{newSt}) \rightarrow \text{return} \]

\[ \text{obj} : O \]

\[ \text{obj} : O \]

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component Behavior (Callback)

\[
\text{update(st)} \rightarrow \text{notify(st)} \rightarrow \text{update(newSt)} \rightarrow \text{notify(newSt)} \rightarrow \text{return} \rightarrow \text{return} \rightarrow \text{return}
\]
Subject Component Behavior (Callback)

- **Subject**: `S
- **Object 1**: `O1`
- **Object 2**: `O2`

**User** (`u`):
- `update(st)`

**Subject** (`S`):
- `notify(st)`
- `update(newSt)`
- `notify(newSt)`
- `return`

**Object 1** (`O1`):
- `return`

**Object 2** (`O2`):
- `return`

**IK, APH, YW** – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component Behavior (Callback)

\[ u: \text{User} \]
\[ \text{subj: S} \]
\[ o1: 0 \]
\[ o2: 0 \]

\[ \text{update(st)} \]
\[ \text{notify(st)} \]
\[ \text{update(newSt)} \]
\[ \text{notify(newSt)} \]
\[ \text{return} \]
\[ \text{return} \]
\[ \text{return} \]
\[ \text{return} \]

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
2. Trace-Based vs. State-Based Models

Subject Component Behavior (Callback)

- **u: User**
- **subj: S**
- **o1: O**
- **o2: O**

```plaintext
update(st)
notify(st)
update(newSt)
notify(newSt)
return
```

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
2. Trace-Based vs. State-Based Models

Subject Component Behavior (Callback)

\[ u: \text{User} \]

\[ \text{subj}: S \]

\[ o1: 0 \]

\[ o2: 0 \]

\[ \text{update}(st) \]

\[ \text{notify}(st) \]

\[ \text{update}(\text{newSt}) \]

\[ \text{notify}(\text{newSt}) \]

\[ \text{return} \]

\[ \text{return} \]

\[ \text{return} \]

\[ \text{return} \]

\[ \text{return} \]

\[ \text{return} \]

\[ \text{return} \]

\[ \text{return} \]
Subject Component Behavior (Callback)

**u**: User

**subj**: S

**o1**: O

**o2**: O

- `update(st)`
- `notify(st)`
- `update(newSt)`
- `notify(newSt)`
- `return`
- `notify(newSt)`
- `return`
- `return`
- `return`
- `return`
- `notify(st)`
- `return`
2. Trace-Based vs. State-Based Models

Model ingredients

- **Object universe:**

\[ O = \{ sbj, o_1, o_2, st_1, st_2, st_3, \ldots \} \]

- **A component = a set of objects (subset of \( O \))**

\[ C = \{ sbj \} \]
Model ingredients

- Object universe:

\[ O = \{ \text{subj}, o_1, o_2, st_1, st_2, st_3, \ldots \} \]

- A component = a set of objects (subset of \( O \))

\[ C = \{ \text{subj} \} \]

- A set of messages.

\[
Msg = \{ \rightarrow \text{subj}.\text{Subject}(o_1, o_2), \leftarrow \text{subj}.\text{Subject}() \} \\
\cup \{ \rightarrow \text{subj}.\text{update}(s), \leftarrow \text{subj}.\text{update}() \} \\
\cup \{ \rightarrow o.\text{notify}(s), \leftarrow o.\text{notify}() \}
\]
Model ingredients

- Object universe:

\[ O = \{ \text{subj}, o_1, o_2, st_1, st_2, st_3, \ldots \} \]

- A component = a set of objects (subset of \( O \))

\[ C = \{ \text{subj} \} \]

- A set of messages.

\[ \text{Direction} \]

\[ Msg = \{ \text{subj}.\text{Subject}(o_1, o_2), \text{subj}.\text{Subject}() \} \]

\[ \cup \{ \text{subj}.\text{update}(s), \text{subj}.\text{update}() \} \]

\[ \cup \{ o.\text{notify}(s), o.\text{notify}() \} \]
Model ingredients

- **Object universe:**

  \[ O = \{ \text{subj}, o_1, o_2, st_1, st_2, st_3, \ldots \} \]

- **A component** = a set of objects (subset of \( O \))

  \[ C = \{ \text{subj} \} \]

- **A set of messages.**

  \[ \text{Callee} \]

  \[ \text{Msg} = \{ \rightarrow \text{subj}.\text{Subject}(o_1, o_2), \leftarrow \text{subj}.\text{Subject}() \} \]
  \[ \cup \{ \rightarrow \text{subj}.\text{update}(s), \leftarrow \text{subj}.\text{update}() \} \]
  \[ \cup \{ \rightarrow \text{o}.\text{notify}(s), \leftarrow \text{o}.\text{notify}() \} \]
Model ingredients

- **Object universe:**
  \[ O = \{sbj, o_1, o_2, st_1, st_2, st_3, \ldots\} \]

- **A component** = a set of objects (subset of \( O \))
  \[ C = \{sbj\} \]

- **A set of messages.**
  \[ Msg = \{→ sbj.\textbf{Subject}(o_1, o_2), ← sbj.\textbf{Subject}()\} \]
  \[ \cup \{→ sbj.\textbf{update}(s), ← sbj.\textbf{update}()\} \]
  \[ \cup \{→ o.\textbf{notify}(s), ← o.\textbf{notify}()\} \]
2. Trace-Based vs. State-Based Models

Model ingredients

▶ Object universe:

\[ O = \{ \text{subj}, o_1, o_2, st_1, st_2, st_3, \ldots \} \]

▶ A component = a set of objects (subset of \( O \))

\[ C = \{ \text{subj} \} \]

▶ A set of messages.

\[ Msg = \{ \rightarrow \text{subj}.\text{Subject}(o_1, o_2), \leftarrow \text{subj}.\text{Subject}() \} \]
\[ \cup \{ \rightarrow \text{subj}.\text{update}(s), \leftarrow \text{subj}.\text{update}() \} \]
\[ \cup \{ \rightarrow o.\text{notify}(s), \leftarrow o.\text{notify}() \} \]
Model ingredients

- Object universe:
  \[ O = \{ sbj, o_1, o_2, st_1, st_2, st_3, \ldots \} \]

- A component = a set of objects (subset of \( O \))
  \[ C = \{ sbj \} \]

- A set of messages.
  \[ \text{Callee} + \text{Method} = \text{Header} \]
  \[ \text{Msg} = \{ \rightarrow \text{sbj}.\text{Subject}(o_1, o_2), \leftarrow \text{sbj}.\text{Subject}() \} \]
  \[ \cup \{ \rightarrow \text{sbj}.\text{update}(s), \leftarrow \text{sbj}.\text{update}() \} \]
  \[ \cup \{ \rightarrow \text{o.notify}(s), \leftarrow \text{o.notify}() \} \]
Model Representation
Model Representation
2. Trace-Based vs. State-Based Models

Model Representation

$\text{IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models}$
Model Representation

\[ t = \text{IK, APH, YW} \]

State-Based Object Models Are More Abstract Than Trace-Based Models
Model Representation

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Model Representation
Model Representation

2. Trace-Based vs. State-Based Models

State-Based Object Models Are More Abstract Than Trace-Based Models
Model Representation

State-Based Models

Transition system of (abstract) states

\[ \mathcal{M}(\text{Msg}, O) = \langle S, \Theta, s_0 \rangle \]
Model Representation
Model Representation

\[ t = m_1 \]
Model Representation

\[ t = (m_1, m_2) \]
Model Representation

\[ t = (m_1, m_2) \]
Model Representation

\[ t = (m_1, m_2), (m_1, m_3) \]
Model Representation

A trace-based model = a set of traces

\[ T(Msg, O) = \text{Traces}(O) \]

A trace is a sequence of pairs of messages

\[ t = (m_1, m_2), (m_1, m_3) \]
2. Trace-Based vs. State-Based Models

Subject Component Behavior

\[ u : \text{User} \]
\[ \text{update(state)} \]
\[ \text{return} \]

\[ \text{subj} : S \]
\[ \text{notify(state)} \]
\[ \text{return} \]

\[ o1 : O \]
\[ \text{notify(state)} \]
\[ \text{return} \]

\[ o2 : O \]
Subject Component State-Based Model

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component State-Based Model

\[ \text{subj} \to \text{subj}.update(s), \]

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component State-Based Model

\[ \text{s} \rightarrow \text{sbj} \]
\[ \text{sbj}.\text{update}(\text{s}) \]
\[ \rightarrow \text{o1.notify}(\text{s}) \]

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component State-Based Model

\[ o_1.\text{notify}(s) \]
\[ \text{sbj.update}(s), \quad o_1.\text{notify}(s) \]

\[ \text{← } o_1.\text{notify}(), \quad \text{← } o_1.\text{notify}() \]

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
2. Trace-Based vs. State-Based Models

Subject Component State-Based Model

\[ \text{subj} \rightarrow \text{subj}.\text{update}(s) \]

\[ \text{o1} \rightarrow \text{o2}.\text{notify}(s) \]

\[ \text{o2} \rightarrow \text{subj}.\text{update}(s), \text{o1}.\text{notify}(s) \]

\[ \text{o1} \leftarrow \text{o1}.\text{notify}(), \text{o2}.\text{notify}() \]

\[ \text{s} \rightarrow \text{o2}.\text{notify}() \]
2. Trace-Based vs. State-Based Models

Subject Component State-Based Model

IKE, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component State-Based Model

IK, APH, YW – State-Based Object Models Are More Abstract Than Trace-Based Models
Subject Component Trace-Based Model

\[ T = \{[...,(\rightarrow \text{subj.update}(s), \rightarrow \text{o1.notify}(s))], \ldots \} \]
Subject Component Trace-Based Model

\[ T = \{ \ldots, (\rightarrow \text{subj} \cdot \text{update}(s), \rightarrow \text{o1} \cdot \text{notify}(s)) \}, \]
\[ \ldots, (\rightarrow \text{subj} \cdot \text{update}(s), \rightarrow \text{o1} \cdot \text{notify}(s)), \]
\[ (\leftarrow \text{o1} \cdot \text{notify}(), \rightarrow \text{o2} \cdot \text{notify}(s)) \}, \]
Subject Component Trace-Based Model

\[ \mathcal{T} = \{[\ldots, (\rightarrow \text{subj.update}(s), \rightarrow \text{o1.notify}(s))], \\
[\ldots, (\rightarrow \text{subj.update}(s), \rightarrow \text{o1.notify}(s)), \\
(\leftarrow \text{o1.notify}(), \rightarrow \text{o2.notify}(s))], \\
[\ldots, (\rightarrow \text{subj.update}(s), \rightarrow \text{o1.notify}(s)), \\
(\leftarrow \text{o1.notify}(), \rightarrow \text{o2.notify}(s))], \\
(\leftarrow \text{o2.notify}(), \leftarrow \text{subj.update}())], \\
\ldots \} \]
Well-Formedness Condition

All traces must follow the call stack property.

e.g. $t = \ldots, \rightarrow \textit{subj}.\text{update}(s), \rightarrow o1.\text{notify}(s), \leftarrow o2.\text{notify}()$ is disallowed.
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \rightarrow subj.update(s), \rightarrow o1.notify(s), \leftarrow o2.notify() \)
is disallowed.

Trace-based models

Description: mutually recursive functions, *match* and *partition*.

\[ t = \rightarrow a.m, \rightarrow b.n, \rightarrow c.o, \rightarrow d.p, \leftarrow d.p, \rightarrow e.q, \leftarrow e.q, \leftarrow c.o \]
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \to \text{subj}.\text{update}(s), \to o1.\text{notify}(s), \leftarrow o2.\text{notify()} \)
is disallowed.

## Trace-based models

Description: mutually recursive functions, *match* and *partition*.

\[
t = \to a.m, \to b.n, \to c.o, \to d.p, \leftarrow d.p, \to e.q, \leftarrow e.q, \leftarrow c.o
\]
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \rightarrow \text{subj.update}(s), \rightarrow o1.\text{notify}(s), \leftarrow o2.\text{notify}() \) is disallowed.

Trace-based models

Description: mutually recursive functions,  \textit{match} and \textit{partition}.

\[
\begin{align*}
  t &= \rightarrow a.m, \rightarrow b.n, \rightarrow \text{c.o}, \rightarrow d.p, \leftarrow d.p, \rightarrow e.q, \leftarrow e.q, \leftarrow \text{c.o}
\end{align*}
\]
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \rightarrow \text{subj}.\text{update}(s), \rightarrow \text{o1.notify}(s), \leftarrow \text{o2.notify}() \)

is disallowed.

Trace-based models

Description: mutually recursive functions, \textit{match} and \textit{partition}.

\[
\begin{align*}
  t = & \rightarrow a.m, \rightarrow b.n, \rightarrow c.o, \rightarrow d.p, \leftarrow d.p, \rightarrow e.q, \leftarrow e.q, \leftarrow c.o
\end{align*}
\]
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \rightarrow subj.update(s), \rightarrow o1.notify(s), \leftarrow o2.notify() \)
is disallowed.

State-based models

Create a call stack restrictor model.
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \rightarrow subj.update(s), \rightarrow o1.notify(s), \leftarrow o2.notify() \)

is disallowed.

State-based models

Create a call stack restrictor model.
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \rightarrow \text{subj}.\text{update}(s), \rightarrow \text{o1.notify}(s), \leftarrow \text{o2.notify}() \)
is disallowed.

**State-based models**

Create a call stack restrictor model.

![Diagram showing call stack restrictor model]
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \rightarrow \text{subj}.\text{update}(s), \rightarrow \text{o1}.\text{notify}(s), \leftarrow \text{o2}.\text{notify()} \) is disallowed.

State-based models

Create a call stack restrictor model.
Well-Formedness Condition

All traces must follow the call stack property.

e.g. \( t = \ldots, \rightarrow \text{subj}\.update(s), \rightarrow o1\.notify(s), \leftarrow o2\.notify() \) is disallowed.

State-based models

Create a call stack restrictor model.

- extend this to pairs of messages
- take the synchronous product
Trace-Based Models as State-Based Models

Lemma

Every trace-based model $\mathcal{T}$ can be canonically represented as a state-based model $\mathcal{M} = \langle S, \Theta, s_0 \rangle$. 
**Trace-Based Models as State-Based Models**

**Lemma**

Every trace-based model $\mathcal{T}$ can be canonically represented as a state-based model $\mathcal{M} = \langle S, \Theta, s_0 \rangle$.

$t = (m_1, m_2), (m_3, m_4), \ldots$
2. Trace-Based vs. State-Based Models

Trace-Based Models as State-Based Models

**Lemma**

Every trace-based model $\mathcal{T}$ can be canonically represented as a state-based model $\mathcal{M} = \langle S, \Theta, s_0 \rangle$. 

$t = (m_1, m_2), (m_3, m_4), \ldots$
2. Trace-Based vs. State-Based Models

Trace-Based Models as State-Based Models

Lemma

Every trace-based model \( T \) can be canonically represented as a state-based model \( M = \langle S, \Theta, s_0 \rangle \).

Comparable

\[ t = (m_1, m_2), (m_3, m_4), \ldots \]
Model Abstraction

Given: $\mathcal{M}_1 = \langle S_1, \Theta_1, s_{0,1} \rangle$ and $\mathcal{M}_2 = \langle S_2, \Theta_2, s_{0,2} \rangle$

$\mathcal{M}_2$ is more abstract than $\mathcal{M}_1$ iff

there is $\alpha : S_1 \rightarrow S_2$ such that

- $s_{0,2} = \alpha(s_{0,1})$, and
- if $s_1 \xrightarrow{m_a,m_b} s'_1$, then $\alpha(s_1) \xrightarrow{m_a,m_b} \alpha(s'_1)$. 
Abstraction Example

\[ t = t', \rightarrow \text{sbj.u}(s), \rightarrow o_1.n(s), \leftarrow o_1.n(), \rightarrow o_2.n(s), \leftarrow o_2.n(), \leftarrow \text{sbj.u()} \]
Abstraction Example

\[ t = t', \rightarrow sbj.u(s), \rightarrow o_1.n(s), \leftarrow o_1.n(), \rightarrow o_2.n(s), \leftarrow o_2.n(), \leftarrow sbj.u() \]
Abstraction Example

\[ t = t', \rightarrow sbj.u(s), \rightarrow o_1.n(s), \leftarrow o_1.n(), \rightarrow o_2.n(s), \leftarrow o_2.n(), \leftarrow sbj.u() \]
Abstraction Example

\[ t = t', \rightarrow \text{subj.} \text{u}(s), \rightarrow o_1.\text{n}(s), \leftarrow o_1.\text{n}(), \rightarrow o_2.\text{n}(s), \leftarrow o_2.\text{n}(), \leftarrow \text{subj.} \text{u}() \]
Abstraction Example

\[ t = t', \rightarrow sbj.u(s), \rightarrow o_1.n(s), \leftarrow o_1.n(), \rightarrow o_2.n(s), \leftarrow o_2.n(), \leftarrow sbj.u() \]
2. Trace-Based vs. State-Based Models

Model Comparison

**Theorem**
For any trace-based model $\mathcal{T}$, there is state-based model $\mathcal{M}$ which is more abstract than $\mathcal{T}$. The converse does not hold.

**Intuition:**
$\Rightarrow$ Use previous lemma and take $\alpha$ as the identity function.
$\Leftarrow$ Counter example: subject component.
Main cause: sequential setting
Simulation [Milner 1971]

Given: $\mathcal{M}_1 = \langle S_1, \Theta_1, s_{0,1} \rangle$ and $\mathcal{M}_2 = \langle S_2, \Theta_2, s_{0,2} \rangle$

A *simulation* for $(\mathcal{M}_1, \mathcal{M}_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

1. $(s_{0,1}, s_{0,2}) \in \mathcal{R}$,
2. $\ldots$ \hspace{1cm} $\in \mathcal{R}$

If such $\mathcal{R}$ exists for $(\mathcal{M}_1, \mathcal{M}_2)$, $\mathcal{M}_1 \preceq \mathcal{M}_2$. 
Simulation [Milner 1971]

Given: $\mathcal{M}_1 = \langle S_1, \Theta_1, s_{0,1} \rangle$ and $\mathcal{M}_2 = \langle S_2, \Theta_2, s_{0,2} \rangle$

A *simulation* for $(\mathcal{M}_1, \mathcal{M}_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

1. $(s_{0,1}, s_{0,2}) \in \mathcal{R},$

2. $m_a, m_b$

If such $\mathcal{R}$ exists for $(\mathcal{M}_1, \mathcal{M}_2)$, $\mathcal{M}_1 \preceq \mathcal{M}_2$. 
Simulation [Milner 1971]

Given: \( M_1 = \langle S_1, \Theta_1, s_{0,1} \rangle \) and \( M_2 = \langle S_2, \Theta_2, s_{0,2} \rangle \)

A simulation for \((M_1, M_2)\) is a binary relation \( R \subseteq S_1 \times S_2 \) such that

1. \((s_{0,1}, s_{0,2}) \in R\),
2. \( \text{If such } R \text{ exists for } (M_1, M_2), M_1 \preceq M_2.\)
Simulation [Milner 1971]

Given: \( M_1 = \langle S_1, \Theta_1, s_{0,1} \rangle \) and \( M_2 = \langle S_2, \Theta_2, s_{0,2} \rangle \)

A simulation for \((M_1, M_2)\) is a binary relation \( R \subseteq S_1 \times S_2 \) such that

1. \((s_{0,1}, s_{0,2}) \in R\),

2. \(m_a, m_b \in R\)

If such \( R \) exists for \((M_1, M_2)\), \( M_1 \preceq M_2 \).
Simulation [Milner 1971]

Given: $\mathcal{M}_1 = \langle S_1, \Theta_1, s_{0,1} \rangle$ and $\mathcal{M}_2 = \langle S_2, \Theta_2, s_{0,2} \rangle$

A *simulation* for $(\mathcal{M}_1, \mathcal{M}_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

1. $(s_{0,1}, s_{0,2}) \in \mathcal{R}$,

2. 

If such $\mathcal{R}$ exists for $(\mathcal{M}_1, \mathcal{M}_2)$, $\mathcal{M}_1 \preceq \mathcal{M}_2$.

Abstraction function is an instance of simulation relation.
Simulation [Milner 1971]

Given: $\mathcal{M}_1 = \langle S_1, \Theta_1, s_{0,1} \rangle$ and $\mathcal{M}_2 = \langle S_2, \Theta_2, s_{0,2} \rangle$

A simulation for $(\mathcal{M}_1, \mathcal{M}_2)$ is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that

1. $(s_{0,1}, s_{0,2}) \in \mathcal{R}$,
2. $m_a, m_b \in \mathcal{R}$

If such $\mathcal{R}$ exists for $(\mathcal{M}_1, \mathcal{M}_2)$, $\mathcal{M}_1 \preceq \mathcal{M}_2$.

Abstraction function is an instance of simulation relation.

Simulation retains the same behavior in our settings [KučeraMayr02]
Most Abstract Model [Grumberg and Bustan 2003]

Simulation equivalence: $\mathcal{M}_1 \simeq \mathcal{M}_2 \equiv \mathcal{M}_1 \preceq \mathcal{M}_2$ and $\mathcal{M}_2 \preceq \mathcal{M}_1$

Purpose:

- build equivalence classes of states of $\mathcal{M}$
- build quotient models
Most Abstract Model [Grumberg and Bustan 2003]

Simulation equivalence: $\mathcal{M}_1 \sim \mathcal{M}_2 \equiv \mathcal{M}_1 \preceq \mathcal{M}_2$ and $\mathcal{M}_2 \preceq \mathcal{M}_1$

Purpose:
- build equivalence classes of states of $\mathcal{M}$
- build quotient models

Theorem

Let $\mathcal{M}/\sim$ be the quotient of $\mathcal{M}$ under $\sim$, then $\mathcal{M}/\sim$ is the most abstract model.
Conclusion and Future work

Conclusion

- State-based object models are more abstract than trace-based models
- Most abstract models can be built using simulation quotient.
Conclusion and Future work

Conclusion

- State-based object models are more abstract than trace-based models
- Most abstract models can be built using simulation quotient.

Future Work

- Generalize settings (trace restriction as precondition, full OO, nondeterminism, concurrency).
- Compare specifications and build common framework.
Generalized State-Based Subject Specification

```plaintext
state spec Subject {
    Subject sbj;
    Observer o₁, o₂;
    Stack<State> st;

    in → sbj.update(s) out → o₁.notify(s)
    ensures st = old(st).push(s);

    in ← o.notify() out → o₂.notify(s)
    requires o = o₁;
    ensures s = old(st).top();

    in ← o.notify() out ← sbj.update()
    requires o = o₂;
    ensures st = old(st).pop();
}
```