Web Service Monitoring - Part II

KPLTL Property Monitor reduction

Fabio Barbon

Astro project
ITC-IRST

05.10.2005

- Web Services are **compositional** entities
  - Interaction via precise syntax
  - Distributed on computer networks
- *Problems* propagate on compositions
- Web Services implementation and executions may be hidden

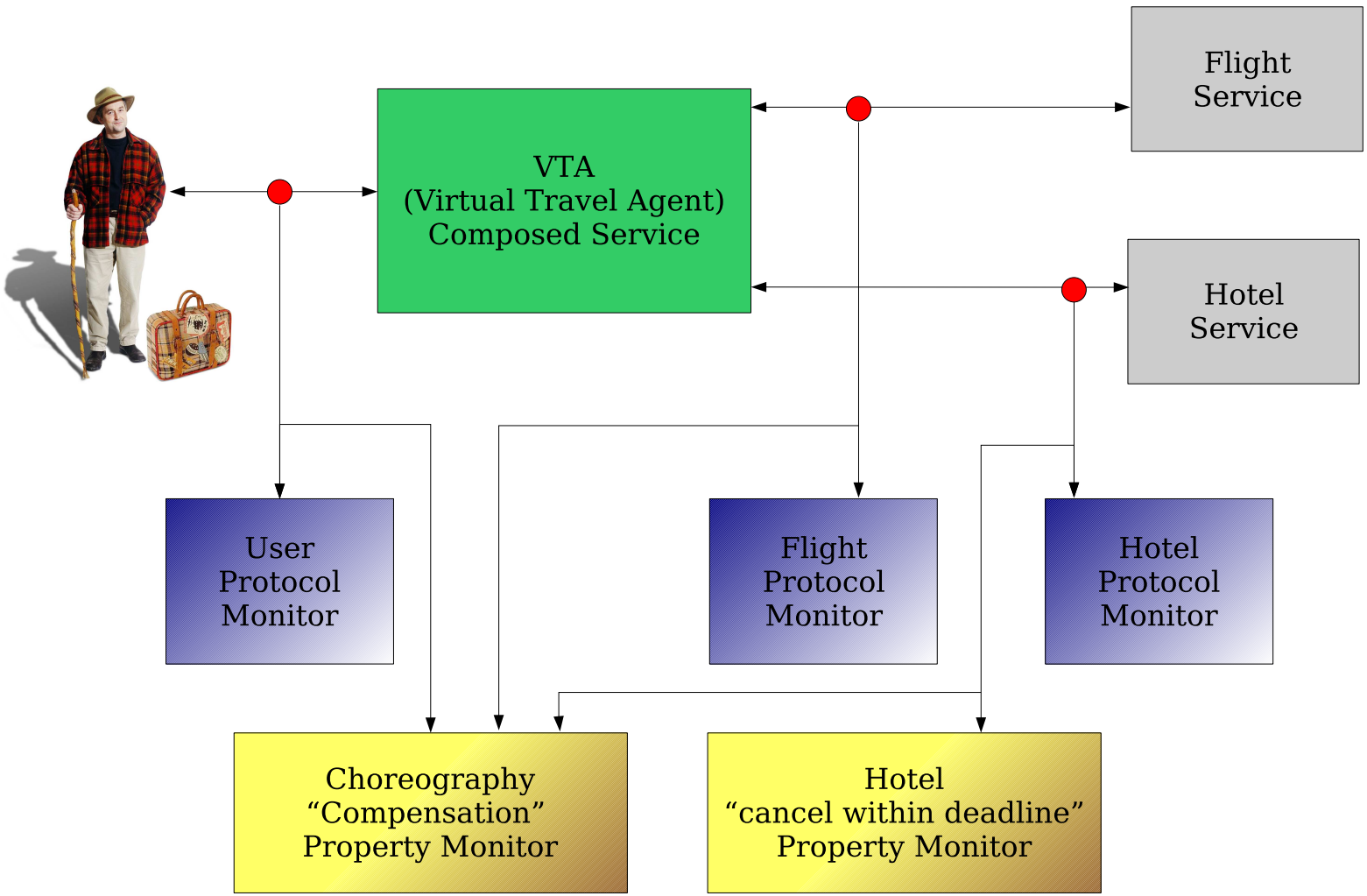
Monitor

A program that runs and evolves in parallel with each web service's instance by grabbing its input/output on client interaction side, and continuously outputs a boolean value.

Boolean value represents *validity* of the message sequence grabbed so far against:

- Web service's public interface
- Additional temporal constraints

Example: *VTA* execution/monitoring scenario



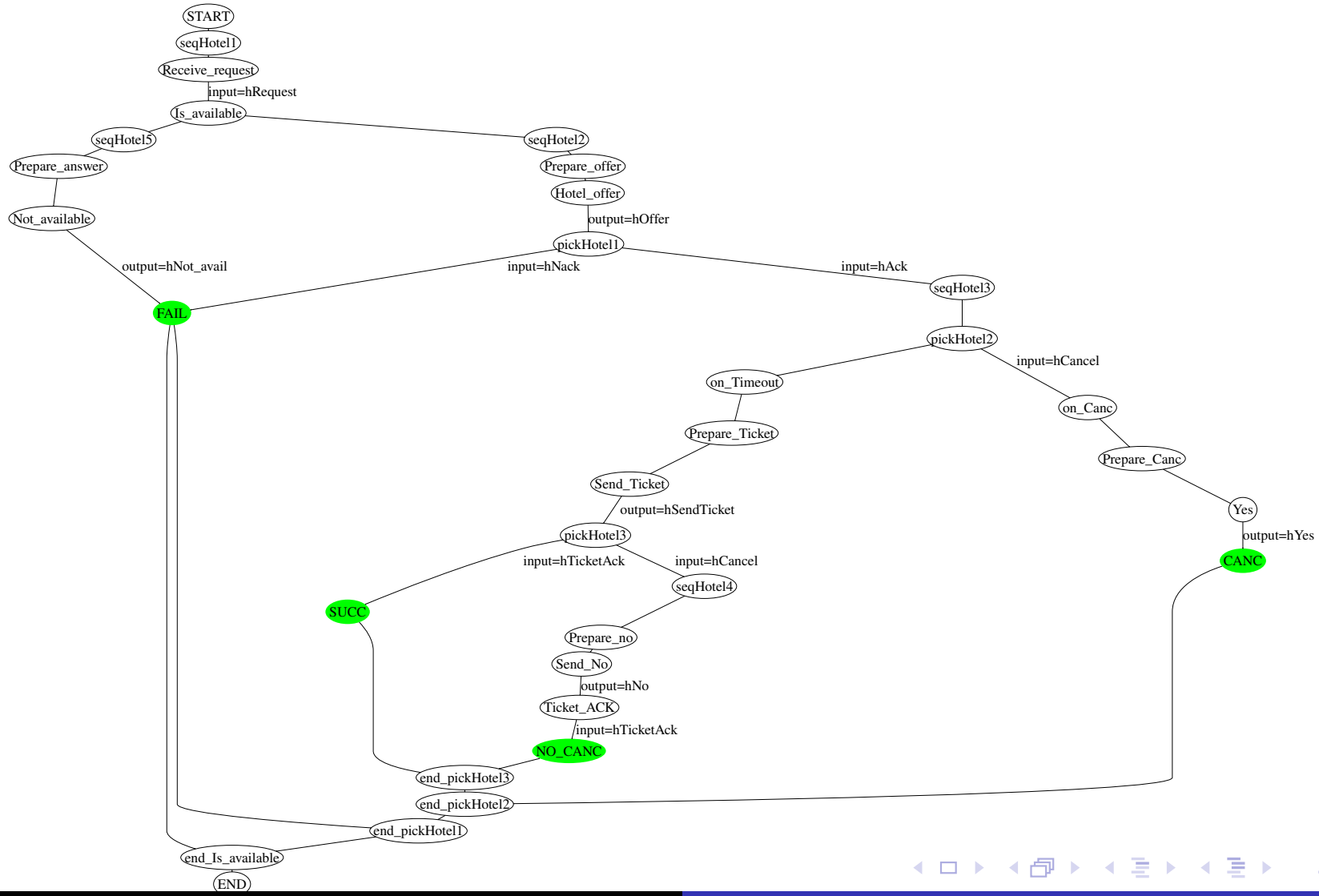
Example: *VTA* Monitors

- "Flight service adheres to its interface"
- "Hotel service won't refuse cancellations requested in time"
- "If a User request for cancellation has been accepted (by the VTA), both Flight and Hotel have canceled their requests"

Silent Actions

If an evolution step of a web service doesn't have any `I/O` associated (for example, `assign` activity or call to another service), the state just before and the state just after the evolution step aren't distinguishable.

Example: Hotel as STS



Example: a SOAP Message

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAP-ENV:Body>
  <requestHotel>
   <time xsi:type="xsd:date">2006-08-14</time>
   <location xsi:type="xsd:string">NewYork</location>
  </requestHotel>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Message space for STSes

Messages

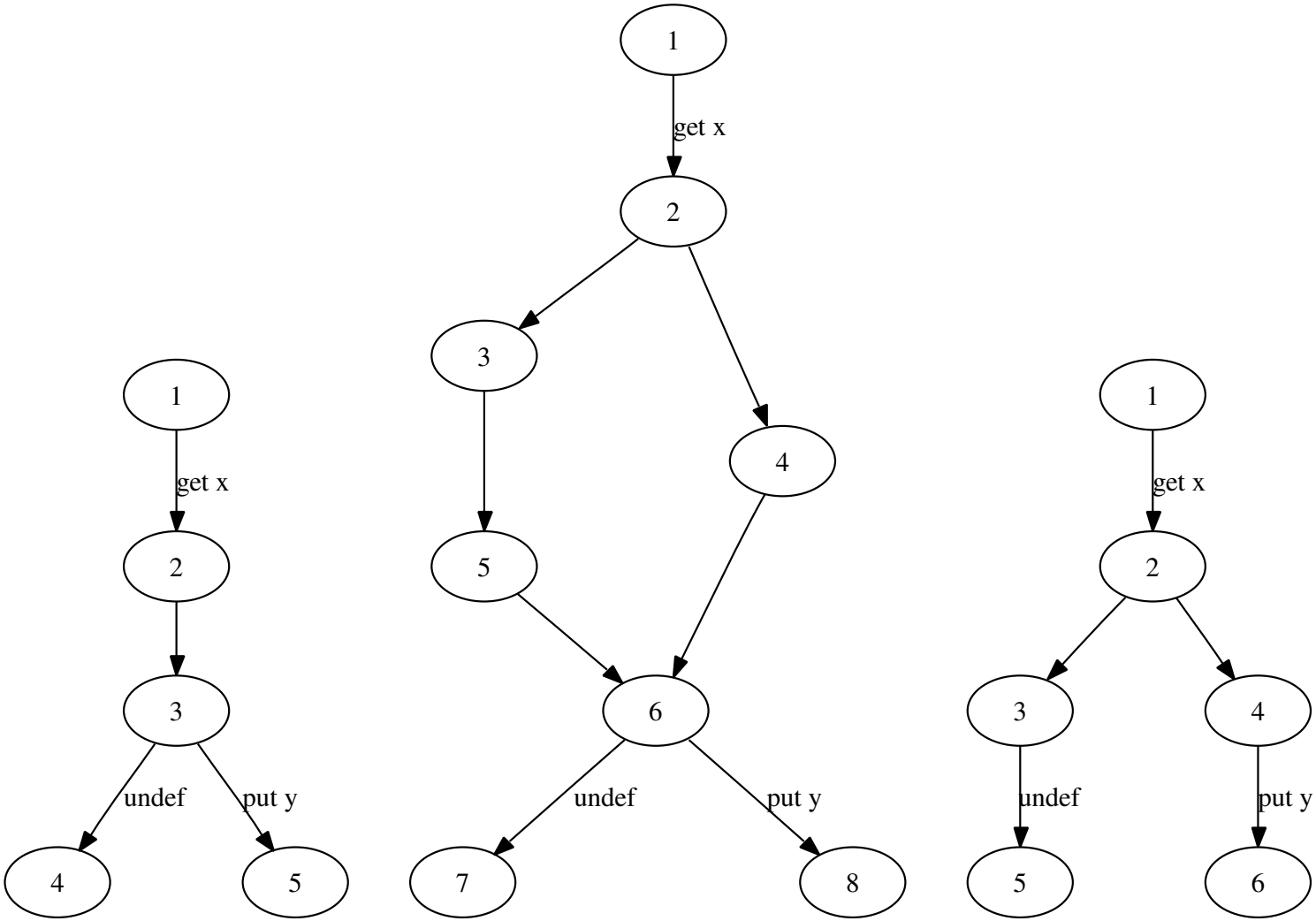
Message space is a set $\mathcal{X}$, τ a special symbol such that $\tau \notin \mathcal{X}$.

- Message space is shared within a STS choreography
- A message is actually a name (`portType`), a (possibly empty) list of parameter names and actual parameter values, and an identification for the two services involved (sender and receiver).
- Example
  - from: `VTA`
  - to: `Hotel`
  - operation: `hRequest(location=NewYork,time=Aug13)`

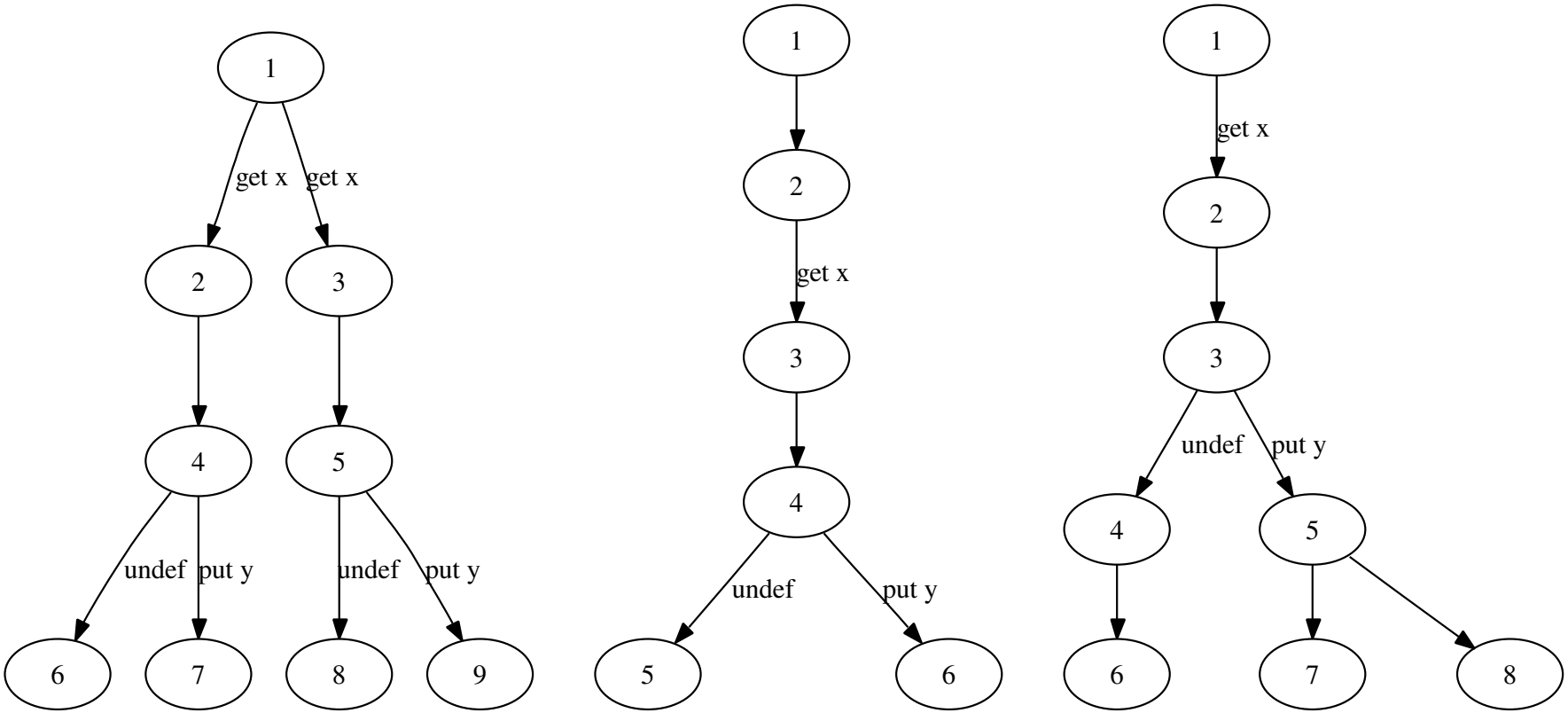
WS as STS

- $\mathcal{S}, \mathcal{X}, \mathcal{P}rop$
- $\Sigma =< \mathcal{S}, \mathcal{I}, \mathcal{X} \cup \{\tau\}, \mathcal{T}, \mathcal{F}, \mathcal{L} >.$
- $\mathcal{I} \subseteq \mathcal{S}$
- $\mathcal{T} \subseteq \mathcal{S} \times (\mathcal{X} \cup \{\tau\}) \times \mathcal{S}$
- $\mathcal{F} \subseteq \mathcal{S}$, the set of final states
- $\mathcal{L} : \mathcal{S} \longrightarrow 2^{\mathcal{P}rop}$
- Silent actions are modeled as τ-transitions $(< s_1, \tau, s_2 > \in \mathcal{T})$

τ-transitions



τ-transitions



Monitor synthesis

build-mon(bs:Belief):Belief
$bs' = \emptyset$
if $bs \neq \emptyset$ **then**
  $bs' = \tau-$closure(bs)
  if $bs' \notin BsSet$ **then**
    $BsSet = BsSet \cup \{bs'\}$
    for all $m \in \mathcal{X}$ **do**
      $bs'' = \mathcal{T}(bs', m)$
      if $bs'' \neq \emptyset$ **then**
        $bs''' =$ **build-mon**(bs'')
        $\mathcal{MS} = \mathcal{MS} \cup \{bs', bs'''\}$
        $\mathcal{MT} = \mathcal{MT} \cup \{< bs', m, bs''' >\}$
      end if
    end for
  end if
end if
return bs'

KPLTL Language

$p \subseteq \mathcal{P}rop$ (p *propositional atoms*)
$s ::= \text{true} \mid \mathbf{K}\, p \mid \neg s \mid s \wedge s \mid \mathrm{Y}\, s \mid \mathrm{O}\, s \mid s \mathrm{S}\, s \mid \mathrm{H}\, s.$

- $\mathrm{Y}\, s$ "previously s";
- $\mathrm{O}\, s$ "at least once in the past s";
- $\mathrm{H}\, s$ "always in the past s";
- $s_1 \mathrm{S}\, s_2$ "s_1 since s_2".

Example: *VTA* Monitors

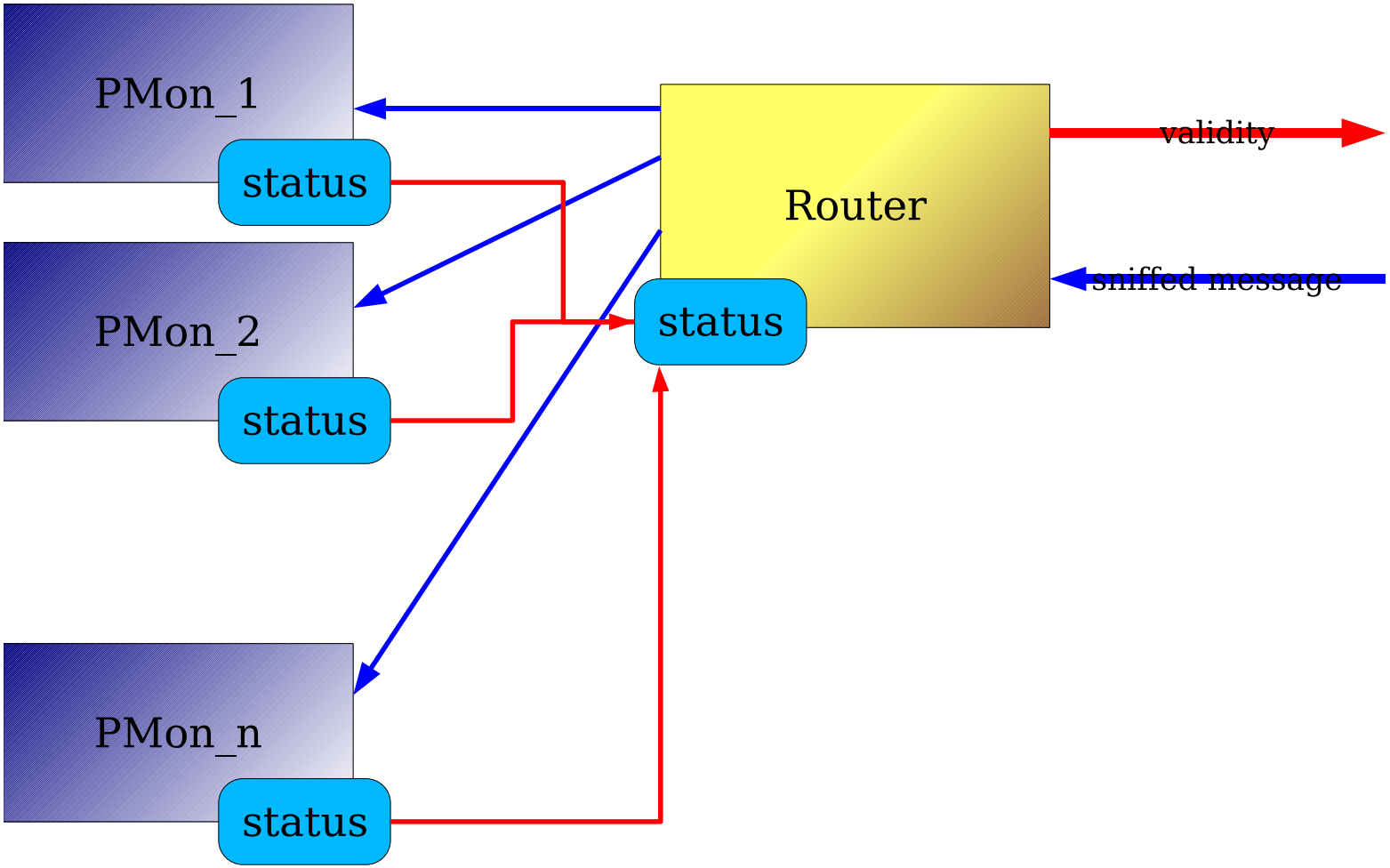
- Hotel service won't refuse cancellations requested in time

$$\mathbf{K}\,(\texttt{state=pickHotel3}) \implies \neg \mathrm{O}\,\mathbf{K}\,(\,\texttt{state=on\_Cancel})$$

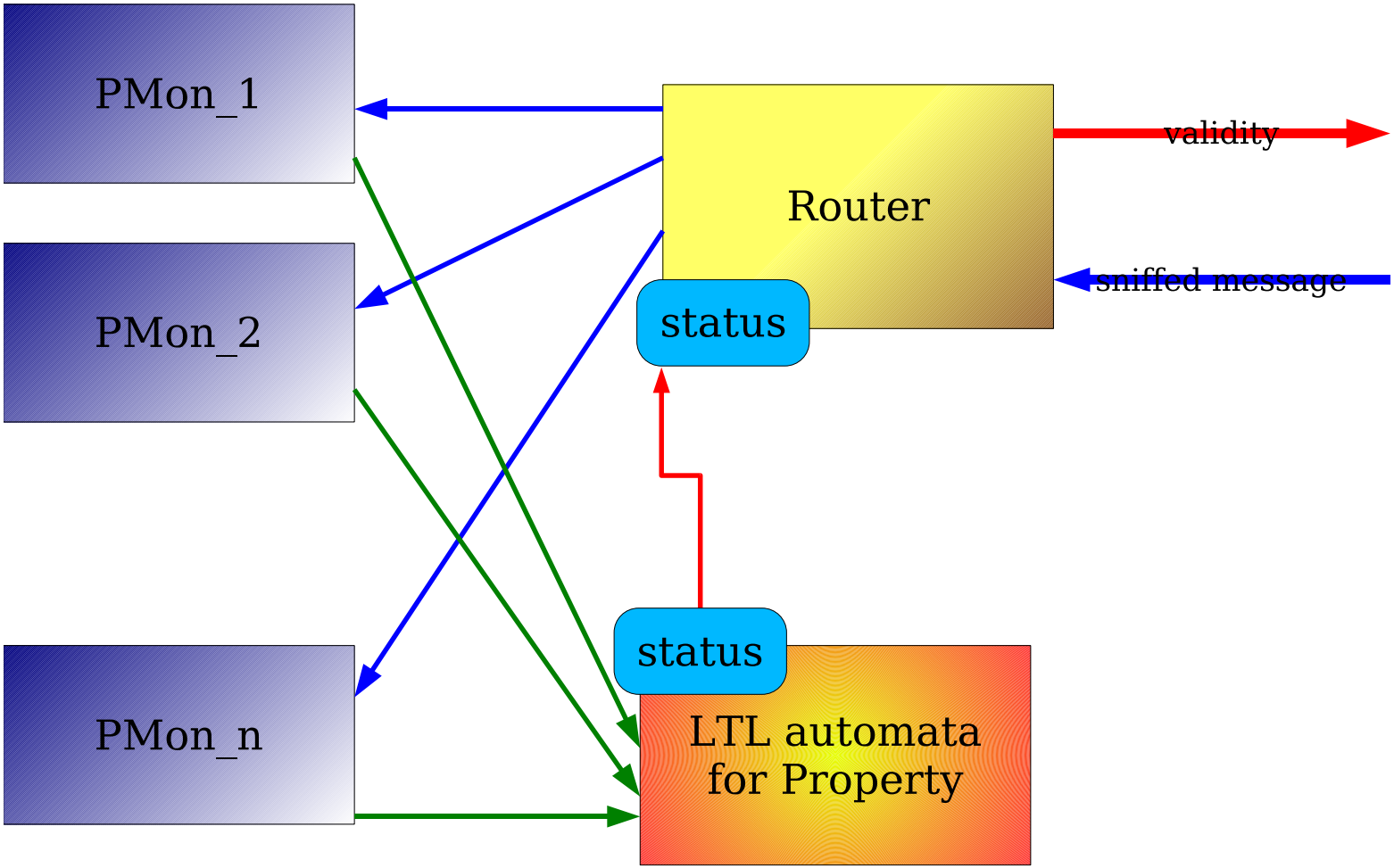
- If a User request for cancellation has been accepted (by the VTA), both Flight and Hotel have canceled their requests

$$\mathbf{K}\,(\texttt{User.state=CANC}) \implies \mathrm{O}\,\mathbf{K}\,(\texttt{Hotel.state=CANC}) \wedge \mathrm{O}\,\mathbf{K}\,(\texttt{Flight.state=CANC})$$

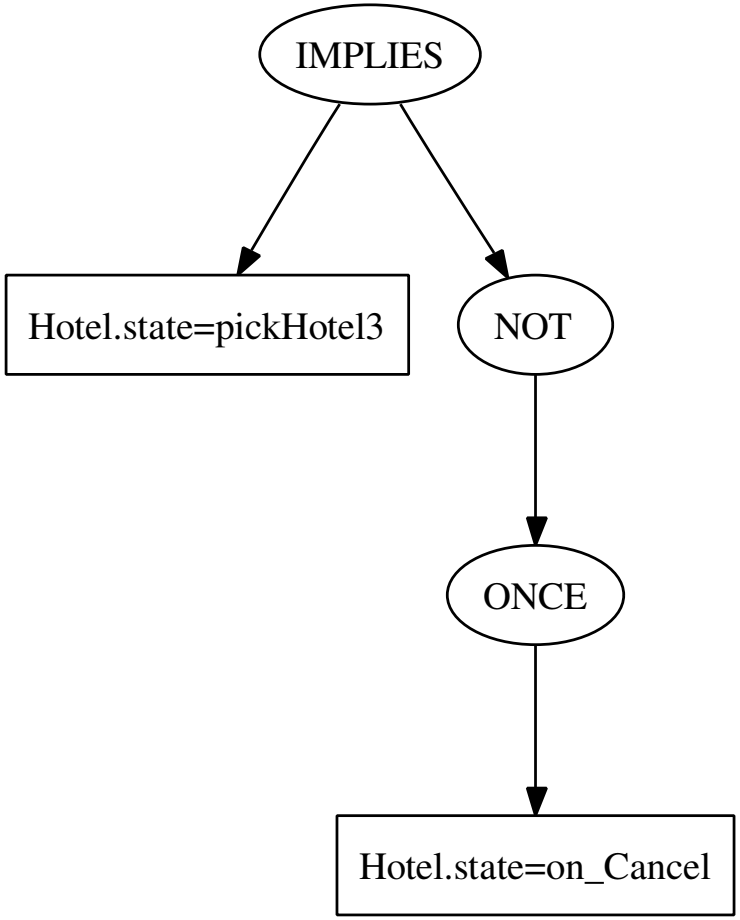
Protocol Monitor Implementation



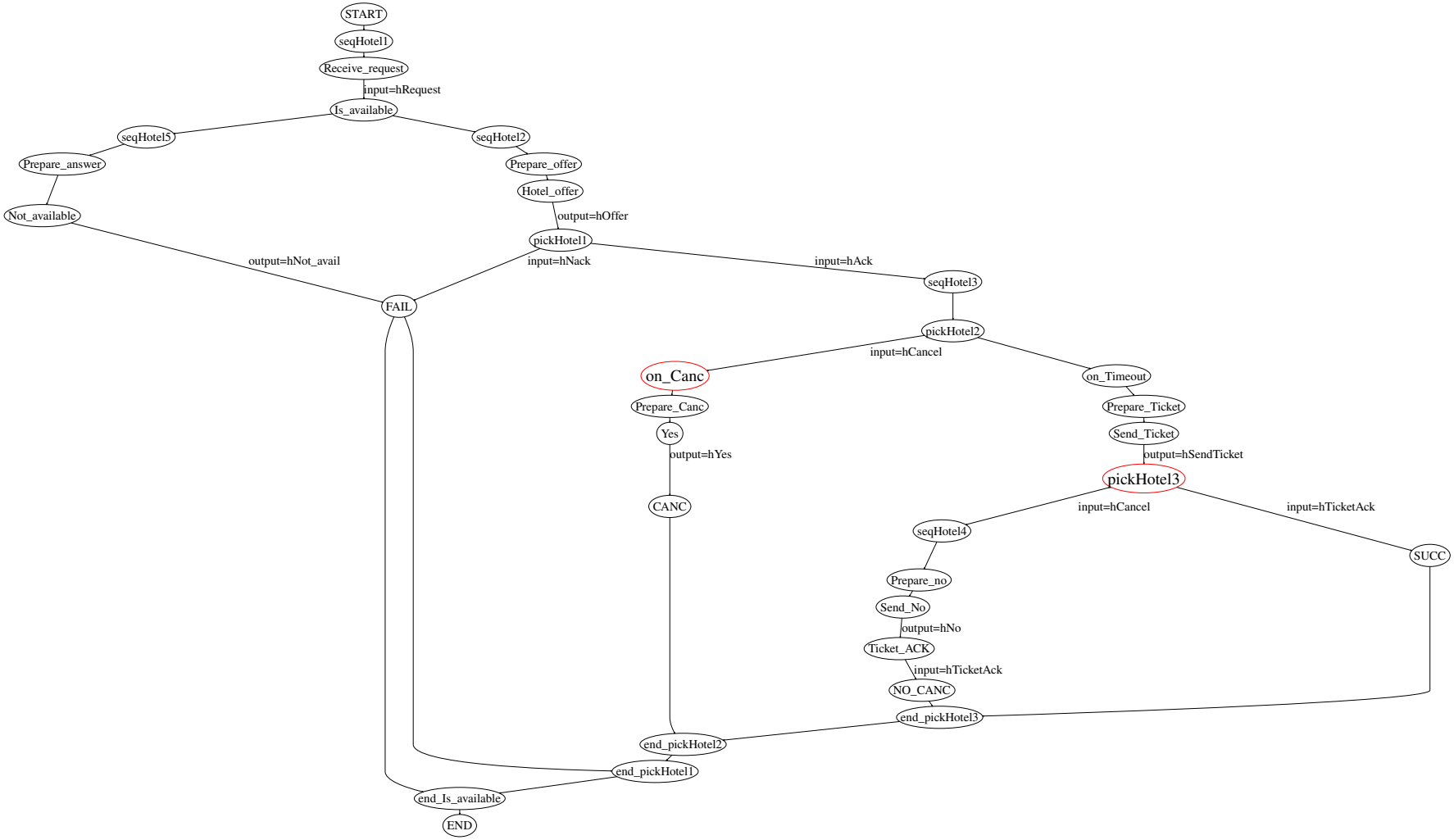
Property Monitor Implementation



K(Hotel.state=pickHotel3) -> ¬O K(Hotel.state=on_Cancel)

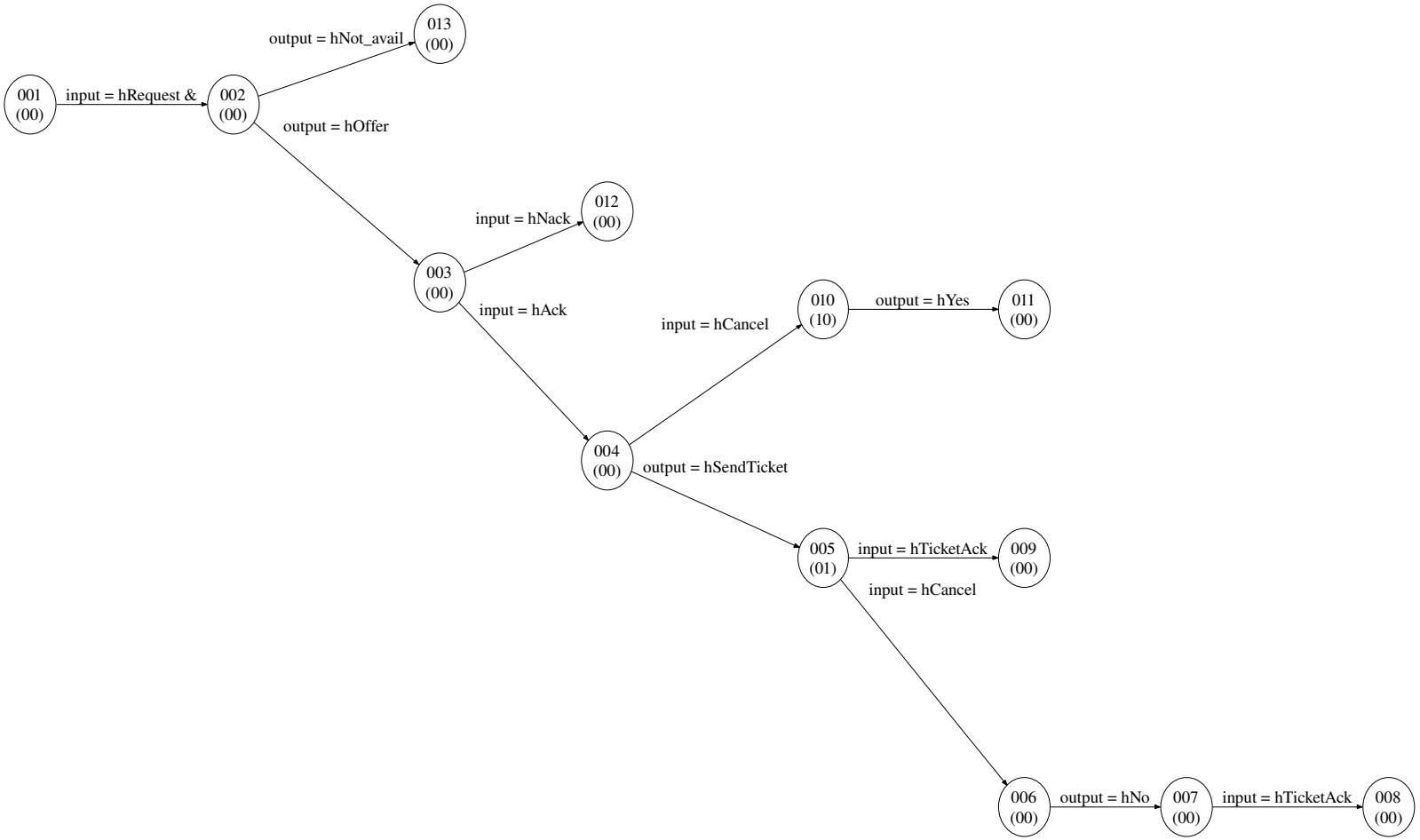


Hotel protocol



(branches on data omitted)

Hotel monitor



(branches on data omitted)

Property Monitor Reduction

- What: verify **P** instead of **P** $\wedge$ **protocols**
- How: simplify protocol monitors used to monitor **P**
- *Local* approach (two beliefs at a time)

Monitor reduction

Collapse beliefs with same truth value on formula's atoms while preserving determinism of automaton.

Algorithm 1

$\mathcal{X}$ message space,
$\mathcal{BS}$ belief space.
$\mathcal{M} =< \mathcal{BS}, \mathcal{I} \subseteq \mathcal{BS}, \mathcal{X} \cup \{\tau\}, \mathcal{T} \subseteq \mathcal{BS} \times (\mathcal{X} \cup \{\tau\}) \times \mathcal{BS} >.$

```
1: void reduce()
2: loop
3:     if ¬ remove-a-belief() then
4:         end
5:     end if
6: end loop
```

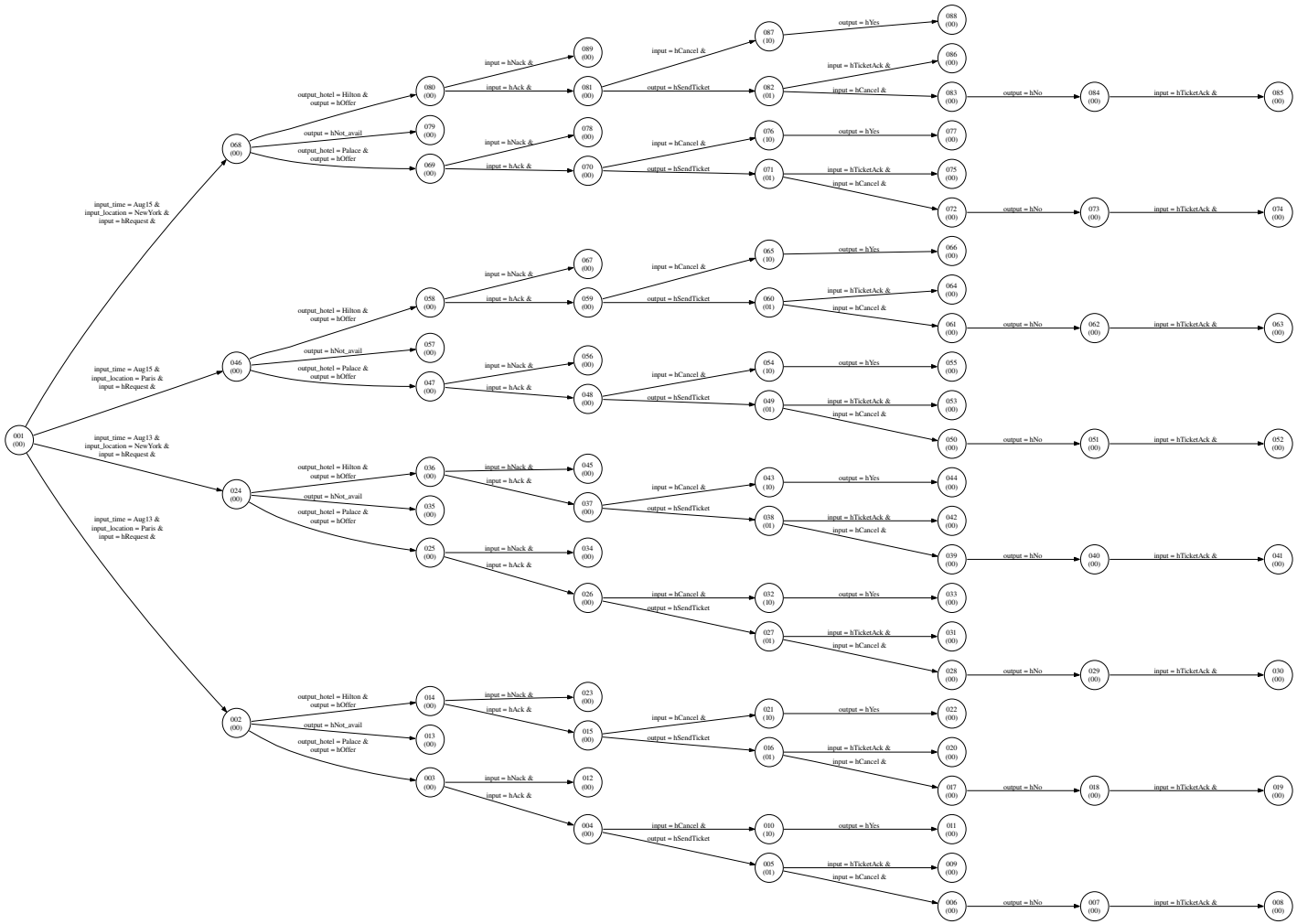
Algorithm 2

```
1: boolean remove-a-belief()
2: for all bs_1, bs_2 ∈ BS do
3:     if bs_1 ≠ bs_2 ∧ same-prop(bs_1, bs_2) then
4:         if fwd-msg-disjoint(bs_1, bs_2) then
5:             collapse-beliefs(bs_1, bs_2)
6:             return true
7:         end if
8:     end if
9: end for
10: return false
```

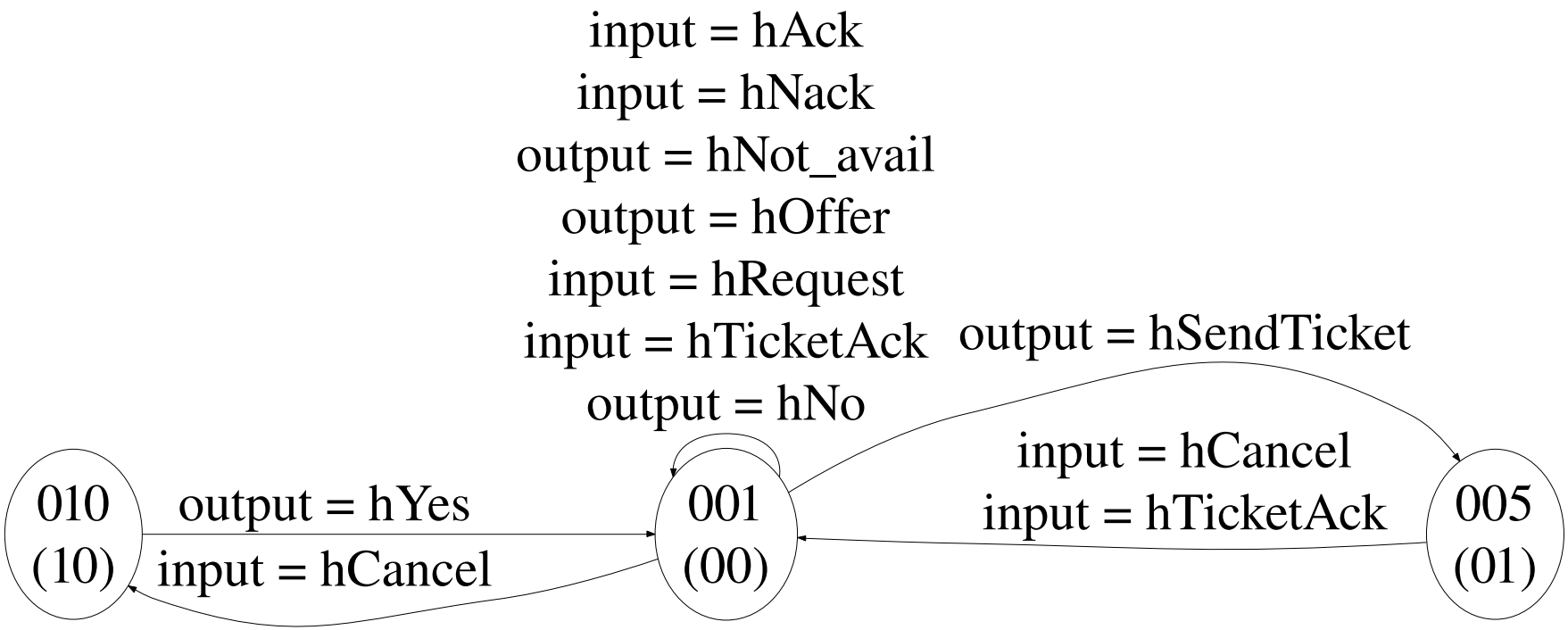
Algorithm 3

1: boolean fwd-msg-disjoint(bs_1, bs_2)
2: **for all** $< bs_1, x, to >, < bs_2, x', to' > \in \mathcal{T}$ **do**
3: **if** $x = x' \wedge to \neq to'$ **then**
4: **return** false
5: **end if**
6: **end for**
7: **return** true

Hotel Monitor (89 beliefs)

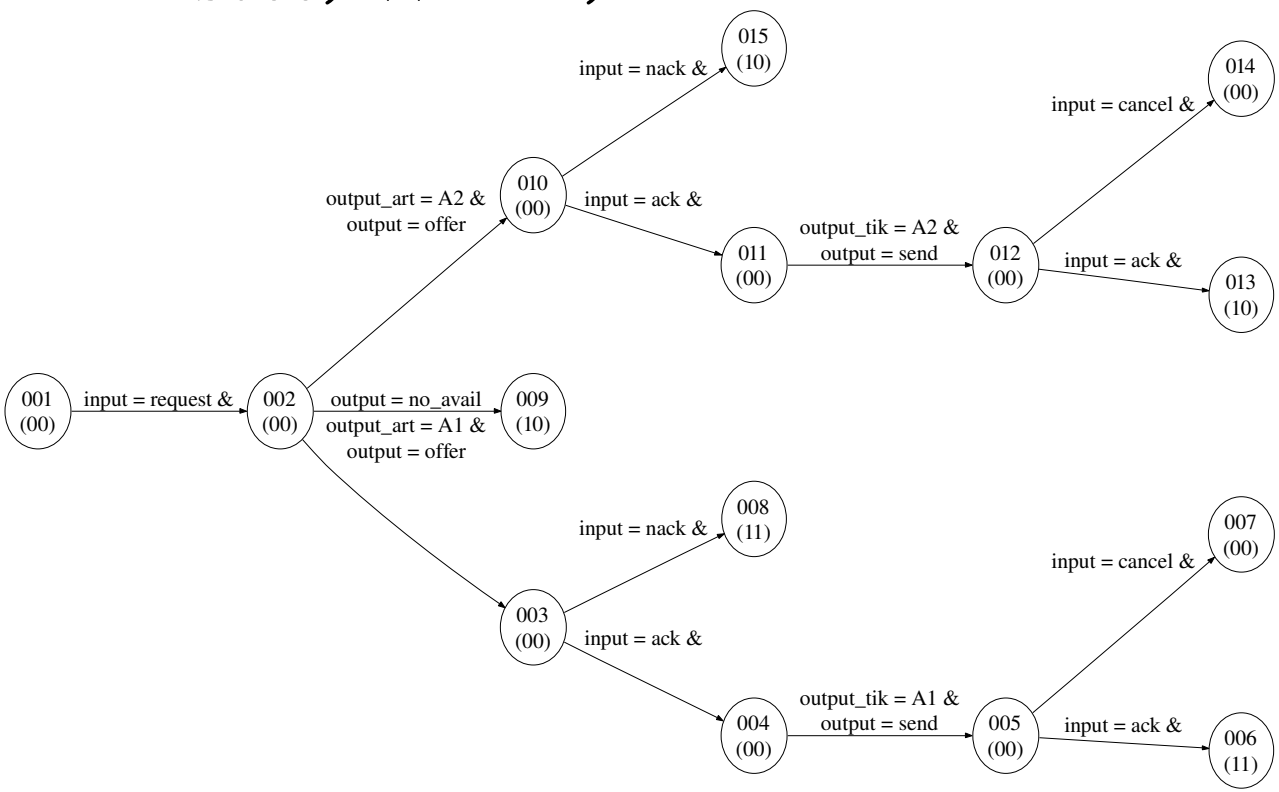


Hotel Monitor reduced

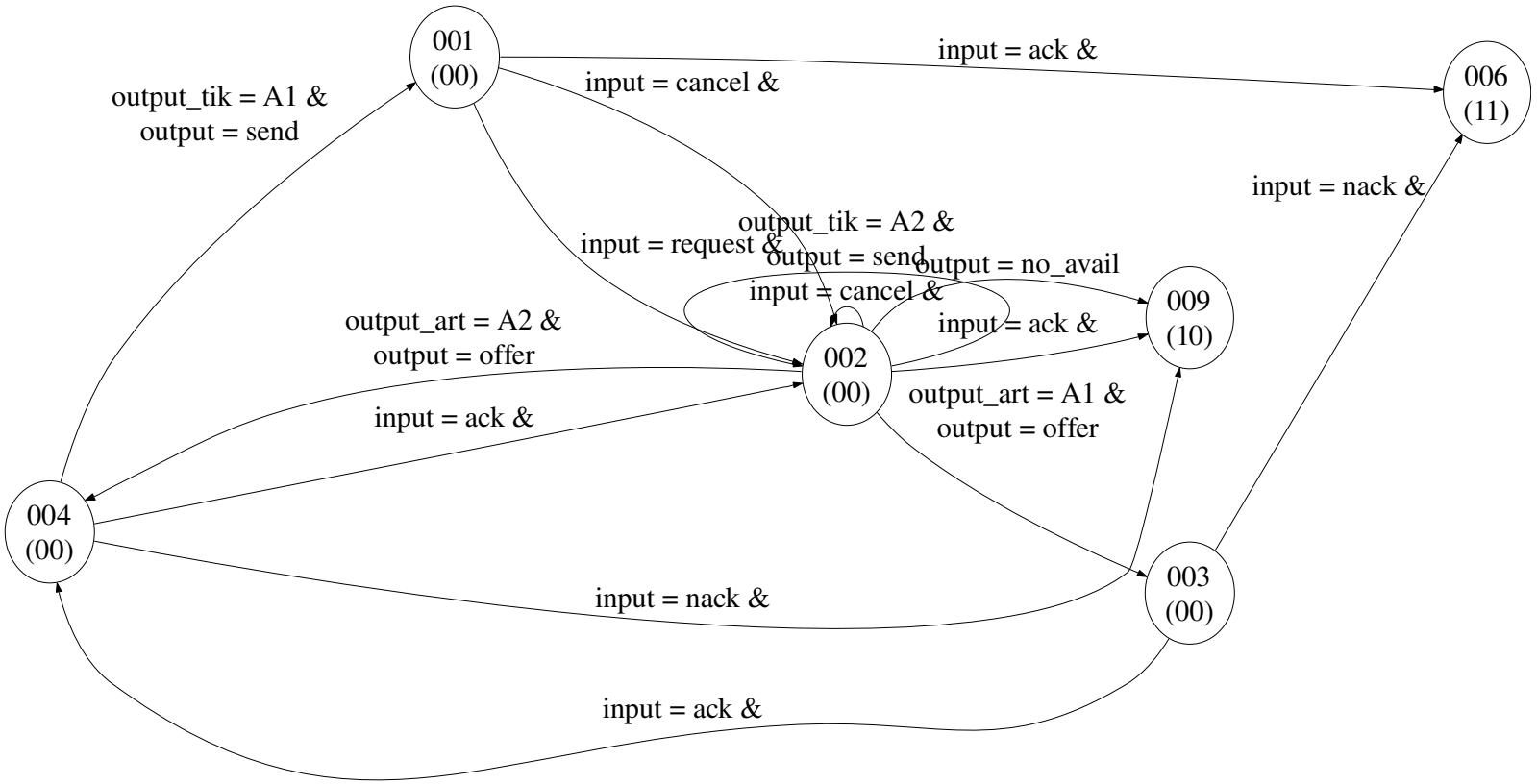


Example 2

$$\texttt{K(state=FAIL} \vee \texttt{state=SUCC)} \wedge \texttt{K((state=FAIL} \vee \texttt{state=SUCC)} \wedge \texttt{b=A1)}$$



Example 2 (reduced)



Issues

- Mutual collapsability
- Fixpoint is **not** guaranteed to be minimum
  - The order in which beliefs are collapsed counts
  - How to choose next couple of beliefs to collapse?
- Current implementation is trivial and inefficient ($O(|\mathcal{BS}|^7)$)
- Reduction doesn't preserve language (existing minimization algorithms, like Hopcroft and Ullman's, don't apply)

Non minimum example (17 beliefs instead of 3)