A simple cloud sync protocol

Norbert Preining

Research Center for Software Verification
Japan Advanced Institute of Science and Technology

Workshop on CafeOBJ and Specification Verification
Kaga, 2013-11-15
## CloudSync in Images

<table>
<thead>
<tr>
<th>Cloud</th>
<th>state</th>
<th>idle</th>
<th>stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC-1</th>
<th>state</th>
<th>idle</th>
<th>stamp</th>
<th>tmp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$k$</td>
<td></td>
<td></td>
<td>o</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC-2</th>
<th>state</th>
<th>idle</th>
<th>stamp</th>
<th>tmp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$l$</td>
<td></td>
<td></td>
<td>o</td>
</tr>
</tbody>
</table>

| ...   | state | idle | stamp | tmp |
|       |       |      |       | o   |

<table>
<thead>
<tr>
<th>PC-n</th>
<th>state</th>
<th>idle</th>
<th>stamp</th>
<th>tmp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m$</td>
<td></td>
<td></td>
<td>o</td>
</tr>
</tbody>
</table>
**CloudSync in images**

<table>
<thead>
<tr>
<th>Cloud</th>
<th>state</th>
<th>busy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$n$</td>
</tr>
</tbody>
</table>

transition: gotvalue

<table>
<thead>
<tr>
<th>PC-1</th>
<th>state</th>
<th>gotvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamp</td>
<td>$k$</td>
<td></td>
</tr>
<tr>
<td>tmp</td>
<td>$n$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC-2</th>
<th>state</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamp</td>
<td>$l$</td>
<td></td>
</tr>
<tr>
<td>tmp</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

...  

<table>
<thead>
<tr>
<th>PC-n</th>
<th>state</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamp</td>
<td>$m$</td>
<td></td>
</tr>
<tr>
<td>tmp</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
**CloudSync in images**

<table>
<thead>
<tr>
<th>Cloud</th>
<th>state</th>
<th>busy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>k</td>
</tr>
<tr>
<td>stamp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

transition: update assuming \( k \geq n \)

<table>
<thead>
<tr>
<th>PC-1</th>
<th>state</th>
<th>update</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>stamp</td>
<td>k</td>
</tr>
<tr>
<td></td>
<td>tmp</td>
<td>k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC-2</th>
<th>state</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>stamp</td>
<td>l</td>
</tr>
<tr>
<td></td>
<td>tmp</td>
<td>o</td>
</tr>
</tbody>
</table>

...|

<table>
<thead>
<tr>
<th>PC-n</th>
<th>state</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>stamp</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>tmp</td>
<td>o</td>
</tr>
</tbody>
</table>
CloudSync in images

Cloud

<table>
<thead>
<tr>
<th>state</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamp</td>
<td>k</td>
</tr>
</tbody>
</table>

transition: gotoidle

<table>
<thead>
<tr>
<th>state</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamp</td>
<td>k</td>
</tr>
<tr>
<td>tmp</td>
<td>o</td>
</tr>
</tbody>
</table>

PC-2

<table>
<thead>
<tr>
<th>state</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamp</td>
<td>l</td>
</tr>
<tr>
<td>tmp</td>
<td>o</td>
</tr>
</tbody>
</table>

PC-n

<table>
<thead>
<tr>
<th>state</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>stamp</td>
<td>m</td>
</tr>
<tr>
<td>tmp</td>
<td>o</td>
</tr>
</tbody>
</table>
SPECIFICATION

ClLabel: {idlecl, busy}

mod! CLLABEL {  
  [ClLabelLt < ClLabel]  
  ops idlecl busy : -> ClLabelLt {constr} .  
  eq (L1:ClLabelLt = L2:ClLabelLt) = (L1 == L2) .  
}
**SPECIFICATION**

ClLabel: \{idlecl, busy\}

PcLabel: \{idlepc, gotvalue, updated\}

\[
\text{mod! PCLABEL} \{ \\
  \text{[PcLabelLt < PcLabel]} \\
  \text{ops idlepc gotvalue updated : } \rightarrow \text{PcLabelLt} \text{ } \{\text{constr}\} \text{.} \\
  \text{eq (L1:PcLabelLt = L2:PcLabelLt) = (L1 == L2).} \\
\}
\]
**SPECIFICATION**

ClLabel: \{idlecl, busy\}
PcLabel: \{idlepc, gotvalue, updated\}
ClState: ClLabel \times \mathbb{N}

\[
\text{mod! CLSTATE} \{
\text{pr(PAIR(NAT, CLLABEL{sort Elt -> ClLabel})\{*{ }
\text{sort Pair -> ClState, op fst -> fst.clstate, }
\text{op snd -> snd.clstate }\})}
\}
\]
**SPECIFICATION**

**ClLabel:** \{idlecl, busy\}

**PcLabel:** \{idlepc, gotvalue, updated\}

**ClState:** ClLabel \times \mathbb{N}

**PcState:** PcLabel \times \mathbb{N} \times \mathbb{N}

\[
\text{mod! PCSTATE } \{
\quad \text{pr}(3\text{TUPLE}(\mathbb{NAT}, \mathbb{NAT},
\quad \text{PCLABEL}\{\text{sort Elt} \rightarrow \text{PcLabel}\}*\n\quad \{\text{sort 3Tuple} \rightarrow \text{PcState}\})
\}
\]
**SPECIFICATION**

ClLabel: \{idlecl, busy\}
PcLabel: \{idlepc, gotvalue, updated\}
ClState: ClLabel \times \mathbb{N}
PcState: PcLabel \times \mathbb{N} \times \mathbb{N}
PcStates: MultiSet(PcState)

```latex
\text{mod! PCSTATES} \{ \\
\text{pr}(\text{MULTISET(\text{PCSTATE}\{\text{sort Elt \to PcState}\}))}^* \\
\{\text{sort MultiSet \to PcStates}\})\\n\}
```
**SPECIFICATION**

ClLabel: \{idlecl, busy\}
PcLabel: \{idlepc, gotvalue, updated\}
ClState: ClLabel × \(\mathbb{N}\)
PcState: PcLabel × \(\mathbb{N} \times \mathbb{N}\)
PcStates: MultiSet(PcState)
State: ClState × PcStates

```plaintext
mod! STATE {
    pr(PAIR(CLSTATE{sort Elt -> ClState}, PCSTATES
    {sort Elt -> PcStates})*{sort Pair -> State})
}
```
**TRANSITIONS**

GetValue: if PC and Cloud is idle, fetch Cloud value
GetValue: if PC and Cloud is idle, fetch Cloud value

\[
\text{mod! GETVALUE \{ pr(STATE) }
\begin{align*}
\text{trans[.getvalue]:} & \\
< & \\
< \ \text{ClVal:Nat, idlecl} > , \\
( \langle <\text{PcVal:Nat}; \text{OldClVal:Nat}; \text{idlepc}> \rangle \ \text{S:PcStates}) & \Rightarrow \\
< & \\
< \ \text{ClVal, busy} > , \\
( \langle <\text{PcVal}; \text{ClVal}; \text{gotvalue}> \rangle \ \text{S}) & .
\end{align*}
\]
**TRANSITIONS**

GetValue: if PC and Cloud is idle, fetch Cloud value

Update: update Cloud/PC according to larger value
**TRANSITIONS**

GetValue: if PC and Cloud is idle, fetch Cloud value

Update: update Cloud/PC according to larger value

```plaintext
mod! UPDATE { pr(STATE)
  trans[update]:
  <
  < ClVal:Nat, busy > ,
  (<<PcVal:Nat;GotClVal:Nat;gotvalue>> S:PcStates)
  > ==
  if PcVal <= GotClVal then
    < <ClVal,busy> , (<<GotClVal;GotClVal;updated>> S)>
  else
    < <PcVal,busy> , (<< PcVal;PcVal;updated >> S) >
  fi .
}
```
TRANSITIONS

GetValue: if PC and Cloud is idle, fetch Cloud value
Update: update Cloud/PC according to larger value
GotoIdle: both PC and Cloud go back to idle
**TRANSITIONS**

GetValue: if PC and Cloud is idle, fetch Cloud value  
Update: update Cloud/PC according to larger value  
GotoIdle: both PC and Cloud go back to idle

```
mod! GOTOIDLE {pr(STATE)
  trans[gotoidle]:
  <
  < ClVal:Nat ,busy > ,
  ( <<PcVal:Nat;OldClVal:Nat;updated >> S:PcStates)
  > ==>
  < <ClVal, idlecl> , ( <<PcVal; OldClVal; idlepc>> S) > .
}
```
Final specification is combination of the three transitions (included modules are shared!)

\[
\text{mod! CLOUD \{ }
\text{pr(GETVALUE + UPDATE + GOTOIDLE)}
\text{\}}
\]
CloudSync

Final specification is combination of the three transitions (included modules are shared!)

```plaintext
mod! CLOUD {
    pr(GETVALUE + UPDATE + GOTOIDLE)
}
```

Goal
CloudSync

Final specification is combination of the three transitions (included modules are shared!)

\[ \texttt{mod! CLOUD \{ pr(GETVALUE + UPDATE + GOTOIDLE) \}} \]

Goal

If PC is in updated state, then the values of the Cloud and the PC agree.
Verification

Hoare style proof
Verification

Hoare style proof
1) show invariant for all initial states
Verification

Hoare style proof
1) show invariant for all initial states
2) show that invariant is preserved over transitions
Verification

Hoare style proof
1) show invariant for all initial states
2) show that invariant is preserved over transitions

In details
- define a set of predicates
  initial : State → Bool
**Verification**

**Hoare style proof**
1) show invariant for all initial states
2) show that invariant is preserved over transitions

**In details**
- define a set of predicates
  - initial : State \(\rightarrow\) Bool
- define a set of predicates
  - invariant : State \(\rightarrow\) Bool
Verification

Hoare style proof
1) show invariant for all initial states
2) show that invariant is preserved over transitions

In details
- define a set of predicates
  initial : State ↦ Bool
- define a set of predicates
  invariant : State ↦ Bool
- show for all states
  ∀ S : initial(S) → invariant(S)
**Verification**

**Hoare style proof**
1) show invariant for all initial states
2) show that invariant is preserved over transitions

**In details**
- define a set of predicates
  
  initial : State $\rightarrow$ Bool

- define a set of predicates
  
  invariant : State $\rightarrow$ Bool

- show for all states
  
  $\forall S : \text{initial}(S) \rightarrow \text{invariant}(S)$

- show for all states
  
  $\forall S : \text{invariant}(S) \rightarrow \text{invariant}(S')$

where $S \rightarrow S'$ is any transition
How to prove $\forall S$?

**Question**
How to prove a statement like

$$\forall S : \text{initial}(S) \rightarrow \text{invariant}(S)$$

?
How to prove $\forall S$?

**Question**

How to prove a statement like

$$\forall S : \text{initial}(S) \rightarrow \text{invariant}(S)$$

?

**Answer**

Show it for any element of a covering set of state expressions.
Covering set

most general: $S$ (state variable) – every state is an instance of $S$
Covering set

most general: $S$ (state variable) – every state is an instance of $S$
more general \{${S_1, \ldots, S_n}$\} such that

$\forall S \exists S_i : S = \sigma(S_i)$

i.e., every state term is an instance of one of the elements of the covering set
Proving with covering sets

Requirements for proving Hoare style
all transitions and predicates have to be *applicable* to terms of the covering set

Covering set

```
ops s1 s2 s3 s4 t1 t2 t3 t4 : -> State .
ops M N K : -> Nat . var PCS : PcStates .
eq s1 = < < N, idlecl > , ( << M; K; idlepc >> PCS ) > .
eq s2 = < < N, idlecl > , ( << M; K; gotvalue >> PCS ) > .
eq s3 = < < N, idlecl > , ( << M; K; updated >> PCS ) > .
eq t1 = < < N, busy > , ( << M; K; idlepc >> PCS ) > .
eq t2 = < < N, busy > , ( << M; K; gotvalue >> PCS ) > .
eq t3 = < < N, busy > , ( << M; K; updated >> PCS ) > .
```
INITIAL PREDICATES

cl-is-idle: Cloud is initially idle

\[
\text{op cl-is-idle-name : } \rightarrow \text{PredName}.
\]

\[
\text{eq[cl-is-idle] : } \text{apply(cl-is-idle-name, S:State)} =
\]
\[
( \text{snd(fst(S))} = \text{idlecl} ).
\]
**Initial Predicates**

- **cl-is-idle**: Cloud is initially idle
- **pcs-are-idle**: All PCs are initially idle

```
op pcs-are-idle-name : -> PredName.
eq[pcs-are-idle] : apply(pcs-are-idle-name,S:State) =
    zero-gotvalue(S) and zero-updated(S).
```
**INITIAL PREDICATES**

cl-is-idle: Cloud is initially idle
pcs-are-idle: all PCs are initially idle
init: cl-is-idle & pcs-are-idle

```plaintext
mod! INITIALSTATE {  
  pr(INITPREDGS)  
  op init-name : -> PredNameSeq .  
  eq init-name = cl-is-idle-name pcs-are-idle-name .  
  pred init : State .  
  eq init(S:State) = apply(init-name, S) .  
}
```
INVARIANT PREDICATES

goal: all PCs in updated state agree with Cloud
INVARIANT PREDICATES

goal: all PCs in updated state agree with Cloud

if Cloud is idle then all PCs, too

only at most one PC is out of the idle state

all PCs in gotvalue state have their tmp value equal to the Cloud value

if Cloud is in busy state, then the value of the Cloud and the gotvalue of the Pcs agree
initial step

\begin{verbatim}
red init(s1) implies invariant(s1) . -- OK
red init(s2) implies invariant(s2) . -- OK
red init(s3) implies invariant(s3) . -- OK
red init(t1) implies invariant(t1) . -- OK
red init(t2) implies invariant(t2) . -- OK
red init(t3) implies invariant(t3) . -- OK
\end{verbatim}
Hoare style in term reduction

induction step search predicate

\textbf{op} inv-condition : State State -> Bool .
\textbf{eq} inv-condition(S, SS) =
\begin{align*}
&\text{not} ( \\
&S =(*,1)\Rightarrow + \ SS \\
&\text{suchThat} \\
&(\text{not} \\
&\quad (\text{(invariant}(S) \implies \text{invariant}(SS)) \\
&\quad == \text{true}) \\
&\quad ) \\
&\quad ) \\
&\quad ) \\
&\) .
\end{align*}
**HOARE STYLE IN TERM REDUCTION**

**induction step**

```
red inv-condition(s1, SS) . -- OK
red inv-condition(s2, SS) . -- OK
red inv-condition(s3, SS) . -- OK
red inv-condition(t1, SS) . -- OK

--> The following condition does not reduce directly
to true, we will deal with it later on
red inv-condition(t2, SS) . -- BAD
red inv-condition(t3, SS) . -- OK
```
Hoare style in term reduction

induction step

red inv-condition(s1, SS) . -- OK
red inv-condition(s2, SS) . -- OK
red inv-condition(s3, SS) . -- OK
red inv-condition(t1, SS) . -- OK

--> The following condition does not reduce directly
--> to true, we will deal with it later on
red inv-condition(t2, SS) . -- BAD
red inv-condition(t3, SS) . -- OK

Rest of the invariant condition with case distinctions
Life run