Introduction to Specification & Verification in CafeOBJ
4. Ticket mutual exclusion protocol

Kazuhiro Ogata
(JAIST)

Gist

• We describe how to conjecture lemmas through a case study where it is proved that a mutual exclusion protocol called Ticket enjoys the mutual exclusion property.
Roadmap

- How to use the built-in equality predicate \_==\_ for proving.
- A case study in which it is proved that Ticket enjoys the mutual exclusion property.

Built-in Equality Predicate (1)

- CafeOBJ provides built-in equality predicate \_==\_.
- Naïve use of \_==\_ may cause some trouble when you conduct a proof.
- Let us try to prove \(\forall n:\text{Nat}. (n == 0)\).

The following is a fictitious proof.

``` OBJ
open NAT
  op n : -> Nat .
  red not (n == 0) .
close
```

CafeOBJ returns true.

- But, we know that the conjecture does not hold because we have the counterexample 0.
- The reason why CafeOBJ returns true is because \(t_1 == t_2\) reduces to true if \(t_1\) and \(t_2\) reduce to a same term and otherwise it reduces to false even if \(t_1\) might equal \(t_2\).
- But, we want to use \_==\_ because it is convenient especially when we use some built-in modules such as \text{NAT}\ (it is impossible to define \_==\_ for each pair of natural numbers).
Built-in Equality Predicate (2)

• An example is shown to describe how to use \( == \) for proving.
  
  mod* NAT+ { pr(NAT) 
  [Nat < Nat+] 
  op \( _==_ \) : Nat+ Nat+ -> Bool {comm} 
  vars X Y : Nat+ vars M N : Nat 
  eq (X = X) = true . 
  eq (M = N) = (M == N) . 
  } 

  In proof passages, you need to use \( \text{Nat}^+ \) instead of \( \text{Nat} \).

• The previous fictitious proof becomes
  
  open NAT+ 
  op n : -> Nat+ . 
  red not(n = 0) . 
  close 
  CafeOBJ returns \((n = 0) \) xor true.

Ticket: A Mutual Exclusion Protocol

• The program executed by each process \( i \):

\[
\text{Loop} \\
\text{Remainder Section} \\
rs: \text{ticket}[i] := \text{atomicInc}(vm); \\
ws: \text{repeat until} \text{ticket}[i] = \text{turn}; \\
\text{Critical Section} \\
cs: \text{turn} := \text{turn} + 1;
\]

Initially,
• each process is at label \( rs \),
• \( vm \) is 0,
• \( turn \) is 0, and
• each \( \text{ticket}[i] \) is an arbitrary number.

• \text{atomicInc}(x)\) atomically increments the value contained in a
  given variable \( x \) and returns the old value of \( x \).
• \( vm \) and \( turn \) are shared by all processes.
• \( \text{ticket}[i] \) is local to process \( i \).
• The type of each variable is natural numbers.
Modeling Ticket as an OTS (1)

- Four observation functions are used:
  - \( \text{bop \, vm} : \text{Sys} \rightarrow \text{Nat}^+ \)
  - \( \text{bop \, turn} : \text{Sys} \rightarrow \text{Nat}^+ \)
  - \( \text{bop \, ticket} : \text{Sys \, Pid} \rightarrow \text{Nat}^+ \)
  - \( \text{bop \, pc} : \text{Sys \, Pid} \rightarrow \text{Label} \)

- For a state \( s \) and a process ID \( i \),
  - \( \text{vm}(s) \) is the value of \( vm \) in state \( s \).
  - \( \text{turn}(s) \) is the value of \( turn \) in state \( s \).
  - \( \text{ticket}(s, \, i) \) is the value of \( ticket[i] \) in state \( s \).
  - \( \text{pc}(s, \, i) \) is the location(rs, ws, or cs) of process \( i \) in state \( s \).

Modeling Ticket as an OTS (2)

- An arbitrary initial state is denoted by constant \text{init} declared as follows:
  - \( \text{op \, init} : \rightarrow \text{Sys} \)

- \text{init} is defined as follows:
  - \( \text{eq} \, \text{vm}(\text{init}) \, = \, 0 \) .
  - \( \text{eq} \, \text{turn}(\text{init}) \, = \, 0 \) .
  - \( \text{eq} \, \text{pc}(\text{init}, \, I) \, = \, \text{rs} \) .
Modeling Ticket as an OTS (3)

- Three transition functions are used:
  - `bop get : Sys Pid -> Sys`
  - `bop try : Sys Pid -> Sys`
  - `bop exit : Sys Pid -> Sys`

  - `get` corresponds to the assignment at label `rs`.
  - `try` corresponds to one iteration of the look at label `ws`.
  - `exit` corresponds to the assignment at label `cs`.

Modeling Ticket as an OTS (4)

- What `get` does can be depicted as follows:
Modeling Ticket as an OTS (5)

• get is defined as follows:
  \[
  \text{op } c\text{-get} : \text{Sys } \text{P} \text{id} \rightarrow \text{Bool}.
  \]
  \[
  \text{eq } c\text{-get}(S,I) = (p\text{c}(S,I) = rs).
  \]
  \[
  --
  \]
  \[
  \text{ceq } \text{vm}(\text{get}(S,I)) = \text{vm}(S) + 1 \text{ if } c\text{-get}(S,I).
  \]
  \[
  \text{eq } \text{turn}(\text{get}(S,I)) = \text{turn}(S).
  \]
  \[
  \text{ceq } \text{ticket}(\text{get}(S,I), J)
  = (\text{if } I = J \text{ then } \text{vm}(S) \text{ else ticket}(S, J) \text{ fi})
  \text{ if } c\text{-get}(S,I).
  \]
  \[
  \text{ceq } \text{pc}(\text{get}(S,I), J)
  = (\text{if } I = J \text{ then } \text{ws} \text{ else pc}(S, J) \text{ fi})
  \text{ if } c\text{-get}(S,I).
  \]
  \[
  \text{bceq } \text{get}(S,I) = S \text{ if not } c\text{-get}(S,I).
  \]

Modeling Ticket as an OTS (6)

• What \textbf{try} does can be depicted as follows:
Modeling Ticket as an OTS (7)

- **try is defined as follows:**
  
  \[
  \text{try} \text{ is defined as follows:} \\
  \text{op \ c-try : Sys Pid} \rightarrow \text{Bool} . \\
  \text{eq \ c-try}(S,I) \\
  \quad = (pc(S,I) = \text{ws and ticket}(S,I) = \text{turn}(S)) . \\
  \quad -- \\
  \text{eq \ vm(try}(S,I)) \\
  \quad = \text{vm}(S) . \\
  \text{eq \ turn(try}(S,I)) \\
  \quad = \text{turn}(S) . \\
  \text{eq \ ticket(try}(S,I),J) = \text{ticket}(S,J) . \\
  \text{ceq \ pc(try}(S,I),J) \\
  \quad = (\text{if} \ I = J \text{ then } \text{cs} \ \text{else } pc(S,J) \ \text{fi}) \\
  \quad \text{if \ c-try}(S,I) . \\
  \text{bceq \ try}(S,I) \\
  \quad = S \text{ if not \ c-try}(S,I) . \\
  \]

Modeling Ticket as an OTS (4)

- **What exit does can be depicted as follows:**

\[
\begin{align*}
\text{vm: } m \\
\text{turn: } n \\
\ldots, \text{ticket[i]: } k, \ldots \\
\ldots, \text{pc[i]: cs, } \\
\end{align*}
\]

\[
\begin{align*}
&\text{exit,} \\
&\text{exit}(s,i)
\end{align*}
\]
Modeling Ticket as an OTS (6)

- **exit is defined as follows:**
  
  \[
  \text{op c-exit : Sys Pid -> Bool}.
  \]
  
  \[
  \text{eq c-exit(S,I) = (pc(S,I) = cs).}
  \]
  
  \[
  \begin{align*}
  \text{--} \\
  \text{eq vm(exit(S,I)) = vm(S).} \\
  \text{eq turn(exit(S,I))} \\
  & \quad = \text{turn(S) + 1 if c-exit(S,I).} \\
  \text{eq ticket(exit(S,I),J) = ticket(S,J).} \\
  \text{ceq pc(exit(S,I),J)} \\
  & \quad = (\text{if I = J then rs else pc(S,J) fi if c-exit(S,I).}} \\
  \text{bceq exit(S,I) = S if not c-exit(S,I).}
  \end{align*}
  \]

Proof of the Mutex Property (1)

- **We prove that Ticket enjoys the mutual exclusion property.**
  
  - To this end, the following are declared in module INV:
    
    \[
    \begin{align*}
    \text{op s : -> Sys} & \quad \text{ops i j : -> Pid} \\
    \text{op invl : Sys Pid Pid -> Bool} & \quad \text{eq invl(S,I,J)} \\
    & \quad = (\text{pc(S,I) = cs and pc(S,J) = cs}) \\
    & \quad \text{implies (I = J).}
    \end{align*}
    \]

  - Then, the following are declared in module ISTEP:
    
    \[
    \begin{align*}
    \text{op s' : -> Sys} & \quad \text{op istepl : -> Bool} \\
    \text{eq istepl} & \quad = \text{invl(s,i,j) implies invl(s',i,j).}
    \end{align*}
    \]
Proof of the Mutex Property (2)

• Let us consider the proof passage:

open ISTEP
  -- arbitrary values
  op k : -> Pid .
  -- assumptions
  -- eq c-try(s,k) = true .
  eq pc(s,k) = ws .
  eq ticket(s,k) = turn(s) .
  --
  eq i = k .
  eq (j = k) = false .
  eq pc(s,j) = cs .
  -- successor state
  eq s' = try(s,k) .
  -- check
  red istep1 .
close

• CafeOBJ returns false.
• Then, a lemma is conjectured from the five equations that characterize the case (or the proof passage).
• A possible way to conjecture a lemma from the five equations is by combining them with conjunctions, negating the result, and replacing the constants denoting arbitrary values with variables.
• Instead of use of $i = k$, $k$ is replaced with $i$ in the rest of the equations.

Proof of the Mutex Property (3)

• The following is the lemma conjectured in the way.

open ISTEP
  -- arbitrary values
  op k : -> Pid .
  -- assumptions
  -- eq c-try(s,k) = true .
  eq pc(s,k) = ws .
  eq ticket(s,k) = turn(s) .
  --
  eq i = k .
  eq (j = k) = false .
  eq pc(s,j) = cs .
  -- successor state
  eq s' = try(s,k) .
  -- check
  red istep1 .
close

• This is declared in INV.
• The following is declared in ISTEP.

eq istep1 = inv1(s,i,j) implies inv1(s',i,j) .
Proof of the Mutex Property (4)

• \textit{inv2} can be used to discharge the case (or the proof passage).

\begin{verbatim}
open ISTEP
  -- arbitrary values
  op k : -> Pid .
  -- assumptions
  -- eq c-try(s,k) = true .
  eq pc(s,k) = ws .
  eq ticket(s,k) = turn(s) .
  --
  eq i = k .
  eq (j = k) = false .
  eq pc(s,j) = cs .
  -- successor state
  eq s' = try(s,k) .
  -- check
  red inv2(s,i,j) implies istep1 .
close
\end{verbatim}

• \textit{CafeOBJ} returns true.

Proof of the Mutex Property (5)

• \textit{inv1} can be proved with
  – case splitting, and
  – one lemma \textit{inv2}.

• To complete the proof, we need to prove \textit{inv2}.
Proof of inv2 (1)

- Let us consider the proof passage:

```plaintext
open ISTEP
-- arbitrary values
op k : -> Pid .
-- assumptions
-- eq c-get(s,k) = true .
-- eq pc(s,k) = rs .
-- eq i = k . eq (j = k) = false .
-- eq (vm(s) = turn(s) and cs = pc(s,j)) = true .
turn(s) = vm(s) . eq pc(s,j) = cs .

-- successor state
eq s' = get(s,k) .
-- check
red istep2 .
close
```

- CafeOBJ returns false.

- Then, a lemma is conjectured from the five equations that characterize the case (or the proof passage) as inv2 was conjectured.

Proof of inv2 (2)

- The following is the lemma conjectured in the way.

```plaintext
eq inv3(S,I,J)
  = not(pc(S,I) = rs and
    not(J = I) and
    turn(S) = vm(S) and
    pc(S,J) = cs) .

-- This is declared in INV.
-- The following is declared in ISTEP.
eq istep3
  = inv3(s,i,j)
  implies inv3(s',i,j) .

-- inv3 can be used to discharge the proof passage:
  inv3(s,i,j) implies istep2
```

```plaintext
open ISTEP
-- arbitrary values
op k : -> Pid .
-- assumptions
-- eq c-get(s,k) = true .
-- eq pc(s,k) = rs .
-- eq i = k . eq (j = k) = false .
-- eq (vm(s) = turn(s) and cs = pc(s,j)) = true .
turn(s) = vm(s) . eq pc(s,j) = cs .

-- successor state
eq s' = get(s,k) .
-- check
red istep2 .
close
```
Proof of inv2 (3)

- Let us consider the proof passage:

```plaintext
open ISTEP
-- arbitrary values
op k : -> Pid .
-- assumptions
-- eq c-try(s,k) = true .
-- eq pc(s,k) = ws .
-- eq ticket(s,k) = turn(s) .
-- eq (i = k) = false .
-- eq (turn(s) = ticket(s,i) and ws = pc(s,i)) = true .
-- successor state
-- check
red istep2 .
```

- CafeOBJ returns false.

- Then, a lemma is conjectured from the six equations that characterize the case (or the proof passage) as inv2 and inv3 were conjectured.

Proof of inv2 (4)

- The following is the lemma conjectured in the way.

```plaintext
eq inv4(S,I,J)
eq not(pc(S,I) = ws and pc(S,J) = ws and
ticket(S,I) = turn(S) and ticket(S,J) = turn(S) and
not(I = J)) .
```

- This is declared as INV.

- The following is declared as ISTEP.

```plaintext
eq istep4 = inv4(s,i,j)
implies inv4(s',i,j) .
```

- inv3 can be used to discharge the proof passage:

```plaintext
inv4(s,i,j) implies istep2
```

```plaintext
open ISTEP
-- arbitrary values
op k : -> Pid .
-- assumptions
-- eq c-try(s,k) = true .
-- eq pc(s,k) = ws .
-- eq ticket(s,k) = turn(s) .
-- eq (i = k) = false .
-- eq (turn(s) = ticket(s,i) and ws = pc(s,i)) = true .
-- successor state
-- check
red istep2 .
```
Proof of \textit{inv2} (5)

- \textit{inv2} can be proved with
  - case splitting, and
  - two lemmas \textit{inv3} & \textit{inv4}.
- To complete the proof, we need to prove \textit{inv3} & \textit{inv4}.

Proof of \textit{inv3} (1)

- Let us consider the proof passage:

  open \textsc{ISTEP}
  \begin{verbatim}
  -- arbitrary values
  op k : -> Pid .
  -- assumptions
  -- eq c-get(s,k) = true .
  eq pc(s,k) = rs .
  -- eq (i = k) = false .  eq (j = k) = false .
  eq (i = j) = false .  eq (turn(s) = vm(s)) = false .
  -- eq (turn(s) = 1 + vm(s)) and
  -- rs = pc(s,i) and cs = pc(s,j) = true .
  eq turn(s) = 1 + vm(s) .  eq pc(s,i) = rs .  eq pc(s,j) = cs .
  --
  -- successor state
  eq s' = get(s,k) .
  -- check
  red istep3 .
  close
  \end{verbatim}
Proof of inv3 (2)

- Suppose that the equations that characterize a proof passage (or a case) for which CafeOBJ returns false are $eq_1, \ldots, eq_n$.
- The lemma conjectured in the way used so far is $\neg(eq_1 \land \ldots \land eq_n)$.
- Although the lemma can be used to discharge the case (or the proof passage), it is often the case that it is not straightforward to prove the lemma.
- Hence, we need to strengthen the lemma.
- Instead of $\neg(eq_1 \land \ldots \land eq_n)$, we need to use a strengthened one $SL$ such that $SL \Rightarrow \neg(eq_1 \land \ldots \land eq_n)$.
- One possible to strengthen $\neg(eq_1 \land \ldots \land eq_n)$ is to drop some equations from $\neg(eq_1 \land \ldots \land eq_n)$.

Proof of inv3 (3)

open ISTEP
-- arbitrary values
op k : -> Pid .
-- assumptions
-- eq c-get(s,k) = true .
  eq pc(s,k) = rs .
-- eq (i = k) = false . eq (j = k) = false .
  eq (i = j) = false . eq (turn(s) = vm(s)) = false .
  eq (turn(s) = 1 + vm(s)) = true .
  eq (turn(s) = 1 + vm(s)) = true .
  eq (pc(s,i) = rs) . eq (pc(s,j) = cs) .
-- successor state
  eq s' = get(s,k) .
-- check
  red istep3 .
close

• A possible lemma can be conjectured from the marked equations.
Proof of \textit{inv3} (4)

- The lemma conjectured from the marked equations is
  \[ \text{not}(\text{pc}(S,I) = rs \text{ and } \text{not}(J = I) \text{ and } \text{turn}(S) = 1 + \text{vm}(S) \text{ and } \text{pc}(S,J) = cs). \]
- But, since \textit{inv3} is
  \[ \text{not}(\text{pc}(S,I) = rs \text{ and } \text{not}(J = I) \text{ and } \text{turn}(S) = \text{vm}(S) \text{ and } \text{pc}(S,J) = cs), \]
  it seems that the proof of the lemma needs another lemma, which is
  \[ \text{not}(\text{pc}(S,i) = rs \text{ and } \text{not}(J = I) \text{ and } \text{turn}(S) = 2 + \text{vm}(S) \text{ and } \text{pc}(S,J) = cs), \]
  which seems to need a similar lemma,...
- So, we need to strengthen \textit{inv3} as follows:
  \[ \text{eq } \text{inv3}(S,J) \]

- Note that the new \textit{inv3} implies the old \textit{inv3} and all those lemmas.

Proof of \textit{inv3} (5)

- Let us consider the proof passage:

  ```
  open ISTEP
  -- arbitrary values
  op k : -> Pid .
  -- assumptions
  -- eq c-try(s,k) = true .
  eq pc(s,k) = ws .
  eq ticket(s,k) = turn(s) .
  --
  eq j = k .
  eq turn(s) < \text{vm}(s) = false .
  -- successor state
  eq s' = try(s,k) .
  -- check
  red istep3 .
  close
  ```

  - CafeOBJ returns false.
  - Then, a lemma is conjectured from the four equations that characterize the case (or the proof passage).
  - The lemmas conjectured is as follows:
    \[ \text{eq } \text{inv5}(S,J) \]
    = \[ \text{not}(\text{pc}(S,J) = ws \text{ and } \text{ticket}(S,J) = \text{turn}(S) \text{ and } \text{not}(\text{turn}(S) < \text{vm}(S))) . \]
  - This is declared in INV.
  - The following is declared in ISTEP.
    \[ \text{eq } \text{istep5} = \text{inv5}(s,j) \]
    implies \[ \text{inv5}(s',j) . \]
  - \textit{inv5}(s,j) can be used to discharged the case (or the proof passage).
Proof of \textit{inv3} (6)

- \textit{inv3} can be proved with
  - case splitting, and
  - two lemmas \textit{inv1} & \textit{inv5}.

- To complete the proof, we need to prove \textit{inv4} & \textit{inv5}.

Proofs of \textit{inv4} & \textit{inv5}

- When we write the proof score of \textit{inv4}, we notice that \textit{inv4} should be strengthened as we did for \textit{inv3}.

- The strengthened \textit{inv4} is as follows:
  \begin{align*}
  \text{eq inv4}(S,I,J) &= \text{not}(pc(S,I) = \text{ws} \quad \text{and} \\
  &\quad \quad pc(S,J) = \text{ws} \quad \text{and} \\
  &\quad \quad \text{ticket}(S,I) = \text{ticket}(S,J) \quad \text{and} \\
  &\quad \quad \text{not}(I = J)) .
  \end{align*}

- When we write the proof score of the new \textit{inv4}, we also notice that \textit{inv5} should be strengthened.
Exercise

• Conjecture a strengthened $\text{inv}_5$ to complete the proof score of $\text{inv}_4$.
• Complete the proof score of $\text{inv}_4$.
• Write the proof score of $\text{inv}_5$. 