Dorel LUCANU

University “Al.I.Cuza” of Iași
Department of Computer Science
Berthelot 16
6600-Iași, Romania

e-mail: dlucanu@infoiasi.ro
Understanding CafeOBJ by examples

(Introduction to concurrent object-oriented specification)

Contents

• Introduction
• Preliminaries
• Small examples
• A first step in concurrent connection
• Hidden rewriting specification
• Refinement
• Simulation under a vertical signature morphism
• Proving refinement using a simulation
• Fairness
• A second step in concurrent connection
• A complex example: watch (?)
• Conclusions
Refinement

A (concrete) specification $C$ refines a (an abstract) specification $A$ via the signature morphism $\phi$ if for every model $M$ of $C$, the reduct $\phi M$ is a model of $A$.

Proving refinement correctness by coinduction

Strategy

1. Define a relation $R$ on $TC$.
2. Prove that $R$ is a $\phi \Sigma$-congruence. This consists in two steps:
   (a) Prove that $t_1 R t_1', \ldots, t_n R t_n'$ implies $\phi(\sigma)(t_1, \ldots, t_n) R \phi(\sigma)(t_1', \ldots, t_n')$ for all $\sigma \in \Sigma$ and $t_i, t_i' \in TC$, $i = 1, \ldots, n$.
   (b) Prove that $t R t'$ implies $t =_{EC} t'$ if $t, t' \in TC_v$ for some $v \in V$.
3. Prove that $\phi(\theta(l)) R \phi(\theta(r))$ for all equations $(\forall X) l = r$ in $E^A$ and assignments $\theta : X \to TC$.

Candidates for $R$:

1. behavioural equivalence $\sim$:
   
   $t \sim t'$ iff $c[t] =_{EC} c[t']$ for all visible contexts $c \in TC[z]$.
2. relation $\sim$ when $\Sigma = \Gamma + \Delta$:
   
   $t \sim t'$ iff $\phi(c)[t] =_{EC} \phi(c)[t']$ for all visible contexts $c \in T_\Delta[z]$.

   We have only to prove that $\sim$ is $\phi(\Gamma)$-congruence.
3. relation $\sim$ when all operators in $\Delta$ are unary:
   
   $t \sim t'$ iff $\phi(\sigma)[t] =_{EC} \phi(\sigma)[t']$ for all $\sigma \in \Delta$. 

Example: counters 1 (definitions)

mod* COUNTER-01 {
  *[Counter1]*
  protecting(NAT)
  op init1 : -> Counter1
  bop inc1 : Counter1 -> Counter1
  bop clear : Counter1 -> Counter1
  bop val : Counter1 -> Nat
  var C : Counter1
  eq val(init1) = 0 .
  eq val(clear(C)) = 0 .
  eq val(inc1(init1)) = 0 .
  eq val(inc1(clear(C))) = 0 .
  eq val(inc1(inc1(C))) = s s val(C) .
}

mod* COUNTER-2 {
  *[Counter2]*
  protecting(NAT)
  op init2 : -> Counter2
  bop inc2 : Counter2 -> Counter2
  bop clear : Counter2 -> Counter2
  bop val : Counter2 -> Nat
  var C : Counter2
  eq val(init2) = 0 .
  eq val(clear(C)) = 0 .
  eq val(inc2(C)) = s s val(C) .
}

mod* XCOUNTER-01 {
  protecting(COUNTER-01)
  bop inc2 : Counter1 -> Counter1
  var C : Counter1
  beq inc2(C) = inc1(inc1(C)) .
}
Example: counters 1 (proof environment)

mod PROOF {
  -- environment for proving that XCOUNTER-01 refines COUNTER-2:
  -- Version 1
  protecting(XCOUNTER-01)
  op _R_ : Counter1 Counter1 -> Bool {comm}
  vars C C1 C2 : Counter1

  eq inc1(init1) R init1 = true .
  eq inc1(inc1(init1)) R inc1(init1) = false .
  eq inc1(inc1(C1)) R inc1(inc1(C2)) = C1 R C2 .
  eq clear(C1) R clear(C2) = true .
  eq inc1(clear(C1)) R clear(C2) = true .
  eq inc1(clear(C1))) R inc1(clear(C2)) = false .

  ops c c1 c2 : -> Counter1

  eq C R C = true .
}
Example: counters 1 (proof score 1)

-- prove that \( R \) is congruent with "val" by structural induction

-- -- induction basis
open PROOF
red val(inc1(init1)) == val(init1) . -- it should be true
close
-- -- inductive step
open PROOF
-- hypothesis
eq val(c1) = val(c2) .
-- conclusion
red val(inc1(inc1(c1))) == val(inc1(inc1(c2))) . -- it should be true
red val(clear(c1)) == val(clear(c2)) . -- it should be true
red val(inc1(clear(c1))) == val(clear(c2)) . -- it should be true
close

-- prove that \( R \) is congruent with "inc2"
open PROOF
-- hypothesis
eq c1 R c2 = true .
-- conclusion
red inc2(c1) R inc2(c2) . -- it should be true
close

-- prove that \( R \) is congruent with "clear" (the proof is trivial)
open PROOF
-- hypothesis
eq c1 R c2 = true .
-- conclusion
red clear(c1) R clear(c2) . -- it should be true
close

-- prove the equations in COUNTER-2
open PROOF
red val(inc2(c)) == s s val(c) . -- it should be true
-- it does not exist behavioural equations
close
Example: counters 1 (proof score 2)

mod PROOF1
{
  -- environment for proving that XCOUNTER-01 refines COUNTER-2:
  -- Version 2
  protecting(XCOUNTER-01)
  op _SMILE_: Counter1 Counter1 -> Bool
  vars C C1 C2 : Counter1

  eq C1 SMILE C2 = (val(C1) == val(C2)) .

  ops c c1 c2 : -> Counter1
}

-- prove that R is \phi(\Gamma)-congruence

open PROOF1
-- hypothesis
eq val(c1) = val(c2) .
-- conclusion
red inc2(c1) SMILE inc2(c2) . -- it should be true
red clear(c1) SMILE clear(c2) . -- it should be true
close

-- prove the equations in COUNTER-2

open PROOF1
red val(inc2(c)) == s s val(c) . -- it should be true
close
Simulation under a vertical signature morphism

Consider \( SP = (H, \Sigma, E) \) and \( SP' = (H', \Sigma', E') \) two hidden specifications over \((V, \Psi, D)\) such that both of them are consistent and lexic, and \( \phi : (H, \Sigma) \rightarrow (H', \Sigma') \) a vertical signature morphism. A relation \( R \subseteq T_\Sigma \times T_{\Sigma'} \) is a \( \phi \)-simulation from \( SP \) to \( SP' \) iff it satisfies:

1. if \( t R_v t', E \equiv t = d, \text{ and } E' \equiv t' = d' \) then \( d = d' \), for all \( t \) in \( T_\Sigma, t' \) in \( T_{\Sigma'} \), \( d \) in \( D \), \( d' \) in \( D' \), and \( v \) in \( V \),

2. if \( t R_h t' \) then \( \sigma(t, d) R \phi(\sigma)(t', d) \), for all \( t \) in \( T_\Sigma, t' \) in \( T_{\Sigma'} \), \( \sigma \) in \( \Sigma \), \( d \) in \( D_w \), and \( h \) in \( H \).

Candidates:

1. behavioural similarity: \( t \sim_{SP,SP'} t' \) iff for all \( c \in T_\Sigma[z]_v, v \in V \)

\[
E \equiv c[t] = d, \quad E' \equiv \phi(c)[t'] = d' \quad \text{implies} \quad d = d'.
\]

2. if \( \Sigma = \Gamma + \Delta \) then we can use the relation \( \sim_{SP,SP'} \) defined by \( t \sim_{SP,SP'} t' \) iff

\[
E \equiv c[t] = d, \quad E' \equiv \phi(c)[t'] = d' \quad \text{implies} \quad d = d'
\]

for all \( c \in T_\Delta[z]_v, v \in V \). The relation \( \sim_{SP,SP'} \) is a simulation (in fact coincides with the behavioural similarity) if it satisfies

\[
t \sim_{SP,SP'} t' \text{ implies } \sigma(t, d) \sim_{SP,SP'} \phi(\sigma)(t', d) \text{ for all } t \in (T_\Sigma)_h, t' \in (T_{\Sigma'})_{\phi(h)},\]

\[
\sigma \in \Gamma, d \in D_w.
\]

3. \( \sim_{SP,SP'} \) when all operations in \( \Delta \) are unary:

\[
t \sim_{SP,SP'} t' \text{ iff } (E \equiv \sigma(t) = d, E \equiv \phi(\sigma)(t') = d' \text{ implies } d = d', \text{ for all } \sigma \in \Delta).
\]
Example: counters 2 (definitions)

mod* COUNTER-1
{
   *[Counter1]*
   protecting(NAT)
   op init1 : -> Counter1
   bop inc1 : Counter1 -> Counter1
   bop clear : Counter1 -> Counter1
   bop val : Counter1 -> Nat
   var C : Counter1
   eq val(init1) = 0 .
   eq val(clear(C)) = 0 .
   ceq val(inc1(C)) = s(s val(C) quo 2) if (val(C) rem 2) == 1 .
   ceq val(inc1(C)) = s(2 * val(C)) if (val(C) rem 2) == 0 .
}

mod* XCOUNTER-1
{
   protecting(COUNTER-1)
   bop inc2 : Counter1 -> Counter1
   var C : Counter1
   eq val(inc2(C)) = val(inc1(inc1(C))) .
}

If we imagine that COUNTER-1 displays its values a screen, then an external observer will see sequences of the form

0, 1, 2, 5, 4, 9, 6, 13, ..., 0, 1, 2, 5, 4, 9, ...
mod PROOF
{
    protecting(XCOUNTER-1)
    protecting(COUNTER-2)

    op _SMILE_ : Counter2 Counter1 -> Bool

    var C1 : Counter1
    var C2 : Counter2

    ops c1 : -> Counter1
    ops c2 : -> Counter2

    eq init2 SMILE init1 = true.
    eq inc2(C2) SMILE inc2(C1) = (C2 SMILE C1).
    eq clear(C2) SMILE clear(C1) = true.

    -- Lemma 1
    ceq val(inc1(inc1(C1))) = s s val(C1) if (val(C1) rem 2) == 0.

    -- Lemma 2
    -- ceq val(C1) rem 2 = 0 if there is C2 such that C1 SMILE C2.
}
Example: counters 2 (proof score)

-- Prove that C2 SMILE C1 implies inc2(C2) SMILE inc2(C1)

open PROOF
-- hypothesis
eq c2 SMILE c1 = true .
-- conclusion
red inc2(c2) SMILE inc2(c1) . -- it should be true
close

-- The proof of "C2 SMILE C1 implies clear(C2) SMILE clear(C1)" is trivial

-- Prove that C2 SMILE C1 implies val(C2) == val(C1) by struct. induction
-- -- induction basis
open PROOF
red val(init2) == val(init1) . -- it should be true
close
-- -- induction step 1
open PROOF
-- hypothesis
eq c2 SMILE c1 = true . -- not used in the proof
eq val(c2) = val(c1) . -- inductive hypothesis
eq val(c1) rem 2 = 0 . -- by Lemma 2
-- conclusion
red val(inc2(c2)) == val(inc2(c1)) . -- it should be true
red val(clear(c2)) == val(clear(c1)) . -- it should be true
close

It is worth to note that XCOUNTER-1 does not refines (via $\phi$) COUNTER-2. For this, it is enough to see that the equation $val(inc2(C)) = s s val(C)$ does not hold in $\phi T_{XCOUNTER-1}$:

%XCOUNTER-1> red val(inc2(t)) == s s val(t) .
-- reduce in % : val(inc2(inc1(init1))) == s (s val(inc1(init1)))
false : Bool
(0.010 sec for parse, 99 rewrites(0.060 sec), 183 match attempts)
Proving refinement using a simulation

1. Define a relation \( R \subseteq T_S \times T_S' \).
2. Prove that \( R \) is a simulation from \( SP \) to \( SP' \).
3. Prove that \( R \) is surjective.

**Example: counters 3 (definitions)**

We consider a modified version of the module \( \text{COUNTER-1} \) where the attribute \( \text{val} \) displays only the even values:

```plaintext
mod* COUNTER-01 {
  *[Counter1]*
  protecting(NAT)
  op init1 : -> Counter1
  bop inc1 : Counter1 -> Counter1
  bop clear : Counter1 -> Counter1
  bop val : Counter1 -> Nat
  var C : Counter1
  eq val(init1) = 0 .
  eq val(clear(C)) = 0 .
  eq val(inc1(init1)) = 0 .
  eq val(inc1(clear(C))) = 0 .
  eq val(inc1(inc1(C))) = s s val(C) .
}

mod* XCOUNTER-01 {
  protecting(COUNTER-01)
  bop inc2 : Counter1 -> Counter1
  var C : Counter1
  beq inc2(C) = inc1(inc1(C)) .
}
```
Example: counters 3 (proof environment)

mod PROOF {
  -- environment for proving that XCOUNTER-01 refines
  -- COUNTER-2 by simulation

  protecting(XCOUNTER-01 + COUNTER-2)
  op _SMILE_ : Counter2 Counter1 -> Bool
  op #inc1 : Counter1 -> Nat

  vars C1 : Counter1
  vars C2 : Counter2

  eq C2 SMILE C1 = val(C2) == val(C1) .

  eq #inc1(init1) = 0 .
  eq #inc1(inc1(C1)) = s #inc1(C1) .
  eq #inc1(clear(C1)) = 0 .

  op c1 : -> Counter1
  op c2 : -> Counter2
}
Example: counters 3 (proof score 1)

-- Prove that SMILE is a simulation

open PROOF2
-- hypothesis
eq val(c2) = val(c1).
-- conclusion
red inc2(c2) SMILE inc2(c1). -- it should be true
red clear(c2) SMILE clear(c1). -- it should be true
close

-- Prove that SMILE is surjective

-- -- Lemma 1: val(C) = val(inc1(C)) if #inc1(C) rem 2 = 0
-- -- The proof is by natural
-- -- induction on n = #inc1(c1)
open PROOF2
-- basis step
red val(init1) == val(inc1(init1)).
red val(clear(C1)) ==
    val(inc1(clear(C1))).
-- inductive step
-- -- hypothesis
eq val(inc1(c1)) = val(c1).
-- -- conclusion
red val(inc1(inc1(c1))) == val(inc1(inc1(inc1(c1)))).
-- it should be true
close
-- -- end of lemma
Example: counters 3 (proof score 2)

-- Prove that for each term t' of sort Counter1 there is a term t of
-- sort Counter2 such that t SMILE t', by structural induction on t'
---- basis step

open PROOF2
red init2 SMILE init1 . ---- it should be true
red init2 SMILE inc1(init1) . ---- it should be true
close

-- -- inductive step
-- -- -- case 1: \#inc1(c1) rem 2 = 0
open PROOF2
-- hypothesis
eq val(c2) = val(c1) .
-- c2 SMILE c1 == true
eq \#inc1(c1) rem 2 = 0 .
eq val(inc1(c1)) = val(c1) . -- by Lemma 1
-- conclusion
red c2 SMILE inc1(c1) . -- it should be true
close

-- -- case 2: \#inc1(c1) rem 2 = 1
open PROOF2
-- hypothesis
op c1' : -> Counter1 .
eq val(c2) = val(c1) . -- c2 SMILE c1 == true
eq c1 = inc1(c1') .
eq \#inc1(c1') rem 2 = 0 .
eq val(inc1(c1')) = val(c1') . -- by Lemma 1
-- conclusion
red inc2(c2) SMILE inc1(c1) . -- it should be true
close
mod! MUTEX-DATA {
    [ A-State B-State CR-State Config ]

    **> operators

    ops B-bcs B-ecs B-bncs B-encs : -> B-State
    ops av non-av : -> CR-State
    op ___ : A-State B-State CR-State -> Config

    **> variables

    var A : A-State
    var B : B-State
    var C : CR-State

    **> transitions


}

Example: mutual exclusion problem (data)
Example: mutual exclusion problem (object)

mod* MUTEX-OBJECT
{
  protecting (MUTEX-DATA)

  *[ State ]*

  bop A-in_ : State -> A-State
  bop B-in_ : State -> B-State
  bop CR-in_ : State -> CR-State

  bops B-ncs B-pre B-cs B-post : State -> State

  var Q : State

  eq A-in A-ncs(Q) = A-encs .
  eq A-in A-pre(Q) = A-bcs .
  eq A-in A-cs(Q) = A-ecs .
  eq A-in A-post(Q) = A-bncs .
  eq A-in B-ncs(Q) = A-in Q .
  eq A-in B-pre(Q) = A-in Q .
  eq A-in B-cs(Q) = A-in Q .
  eq A-in B-post(Q) = A-in Q .

  eq B-in B-ncs(Q) = B-encs .
  eq B-in B-pre(Q) = B-bcs .
  eq B-in B-cs(Q) = B-ecs .
  eq B-in B-post(Q) = B-bncs .
  eq B-in A-ncs(Q) = B-in Q .
  eq B-in A-pre(Q) = B-in Q .
  eq B-in A-cs(Q) = B-in Q .
  eq B-in A-post(Q) = B-in Q .

  eq CR-in B-ncs(Q) = CR-in Q .
  eq CR-in B-pre(Q) = non-av .
  eq CR-in B-cs(Q) = non-av .
  eq CR-in B-post(Q) = av .
  eq CR-in A-ncs(Q) = CR-in Q .
  eq CR-in A-pre(Q) = non-av .
  eq CR-in A-cs(Q) = non-av .
  eq CR-in A-post(Q) = av .
}
Example: mutual exclusion problem (proof score 1)

-- Theorem 1: MUTEX-OBJECT does not deadlock

-- -- Lemma 1: A can enter its critical section
open MUTEX-OBJECT
-- hypothesis
op q : -> State .
eq A-in q = A-encs .  -- A ended its noncritical section
eq B-in q = B-encs .  -- B ended its noncritical section
eq CR-in q = av .  -- CR is available

-- conclusion
red A-in q B-in q CR-in q =>
  A-in A-pre(q) B-in A-pre(q) CR-in A-pre(q) .  -- it should be true
close

-- -- Lemma 2: B can enter its critical section
open MUTEX-OBJECT
-- hypothesis
op q : -> State .
eq A-in q = A-encs .  -- A ended its noncritical section
eq B-in q = B-encs .  -- B ended its noncritical section
eq CR-in q = av .  -- CR is available

-- conclusion
red A-in q B-in q CR-in q =>
  A-in B-pre(q) B-in B-pre(q) CR-in B-pre(q) .  -- it should be true
close
Example: mutual exclusion problem (proof score 2)

-- -- Lemma 3: A and B cannot enter their critical sections
-- -- in the same time
open MUTEX-OBJECT
-- hypothesis
op q : -> State .
eq [hyp1] : A-in q = A-encs . -- A ended its noncritical section
eq [hyp2] : B-in q = B-encs . -- B ended its noncritical section
eq [hyp3] : CR-in q = av . -- CR is available

-- conclusion
start A-in q B-in q CR-in q .
apply hyp1 within term .
apply hyp2 within term .
apply hyp3 within term .
-- rule MUTEX-DATA.11 is [a-pre] : A-encs B av => A-bcs B non-av
apply MUTEX-DATA.11 within term . -- it could be applied
-- rule MUTEX-DATA.15 is [b-pre] : A B-encs av => A B-bcs non-av
apply MUTEX-DATA.15 within term . -- it couldn’t be applied
close

open MUTEX-OBJECT
-- hypothesis
op q : -> State .
eq [hyp1] : A-in q = A-encs . -- A ended its noncritical section
eq [hyp2] : B-in q = B-encs . -- B ended its noncritical section
eq [hyp3] : CR-in q = av . -- CR is available

-- conclusion
start A-in q B-in q CR-in q .
apply hyp1 within term .
apply hyp2 within term .
apply hyp3 within term .
apply MUTEX-DATA.15 within term . -- it could be applied
apply MUTEX-DATA.11 within term . -- it couldn’t be applied
close