theory *EX-SmartCardPurse-fa* = *SALOverflowFWInst*:

1 Smart Card Purse - fully annotated

This program adds a credit C to a balance B if the new balance B + C does not execeed an upper bound MAX (the highest number the safety policy accepts). To check this condition a procedure is called which sets a flag F to NAT 0 if this condition is violated

1.1 Program Variables

constdefs
$B :: nat$ — balance
$B \equiv 0$
$C :: nat$ — credit
$C \equiv 1$
$M :: nat$ — maximum
$M \equiv 2$
$F :: nat$ — flag
$F \equiv 4$
$P :: nat$ — return address storage
$P \equiv 5$
$A :: nat$ — auxiliary variable
$A \equiv 6$

— initial values
constdefs
$B0 :: nat$
$B0 \equiv 4$ — maximal intial balance
$C0 :: nat$
$C0 \equiv 2$ — initial credit

Program Code

constdefs
```
prog :: SALprogram
prog ≡ [
  (0, [(SET B B0, None),
       (SET C C0, None),
       (SET A 0, None),
       (CALL P 1, Some  (λ (pc,m,e).(∃ b. m B = NAT b ∧ b <= B0)
                                  ∧ m B = NAT B0
                                  ∧ m C = NAT C0
                                  ∧ m A = NAT 0)),
       (JMPL F A 2, Some (λ (pc,m,e). (∃ b. m B = NAT b ∧ b <= B0)
                                  ∧ m B = NAT B0
                                  ∧ m C = NAT C0
                                  ∧ m A = NAT 0
```

```
                                   ∧ (∃ f. m F = NAT f
                                           ∧ (f=0 ⟶ (∃ b. m B = NAT b ∧ ((MAX
− C0) < b)))
                        )
          )),
          (ADD B C, None),
          (JMPB 0, Some TrueF)]),
    (1, [(SET M MAX, Some (λ (pc,m,e). m P = RA (incA (pc⃖ e))
                                           ∧ (∃ b. m B = NAT b)
                                           ∧ (∃ c. m C = NAT c)
                                           ∧ (∀ x. x ≠ P ⟶ m x = (m⃖ e) x))),
          (SUB M C, Some (λ (pc,m,e). m P = RA (incA (pc⃖ e))
                                   ∧ (∃ b. m B = NAT b)
                                   ∧ (∃  c. m C = NAT c)
                                   ∧ (m M = NAT MAX)
                                   ∧ (∀ x. x ≠ M ∧ x ≠ P ⟶ m x = (m⃖ e) x)
                      )),
          (SET F 0, Some (λ (pc,m,e). m P = RA (incA (pc⃖ e))
                                   ∧ (∃ b. m B = NAT b)
                                  ∧ (∃ c. m C = NAT c ∧ m M = NAT (MAX − c))
                                   ∧ (∀ x. x ≠ M ∧ x ≠ P ⟶ m x = (m⃖ e) x)
                      )),
          (JMPL M B 2, Some (λ (pc,m,e). m P = RA (incA (pc⃖ e))
                                     ∧ (∃ b. m B = NAT b)
                                  ∧ (∃ c. m C = NAT c ∧ m M = NAT (MAX − c))
                                     ∧ (∀ x. x ≠ F ∧ x ≠ M ∧ x ≠ P ⟶ m x = (m⃖
e) x)
                                     ∧ (m F = NAT 0)
                        )),
          (SET F 1, Some (λ (pc,m,e). m P = RA (incA (pc⃖ e))
                                   ∧ (∃ b. m B = NAT b)
                                   ∧ (∃ c. m C = NAT c)
                                   ∧  (∀ x. x ≠ F ∧ x ≠ M ∧ x ≠ P ⟶ m x = (m⃖
e) x)
                      )),
          (RET P, Some (λ (pc,m,e). (∀ x. x ≠ F ∧ x ≠ M ∧ x ≠ P ⟶ m x =
(m⃖ e) x)
                                  ∧ (m F = NAT 0 ∨ m F = NAT 1)
                                  ∧ (∃ c b. m C = NAT c ∧ m B = NAT b
                                     ∧ (m F = NAT 0 ⟶ (MAX− c) < b))
                    ))])]
```

1.2 The Verification Condition

Generate an ML file vcgSAL.ML that contains an executable VCG and the program source.

generate-code (*vcgSAL.ML*) [*term-of*]
  vcg = vcgSAL

prg = prog

This is the output our ML program for the VCG yield for this example (copy and paste).

constdefs
vc::SALform
vc ≡ (%s. (((%s. (((%(pc, m, e). ((pc = ((0::nat), (0::nat))) & (((cs e) = [((0::nat), m)]) & (((h e) = []) & (ALL X. ((m X) = ILLEGAL)))))) s) --> ((%s. (((%(pc, m, e). ((Suc (Suc (Suc (Suc (0::nat))))) <= MAX)) s) & ((%s. (((%s. (((%(pc, m, e). (pc = ((0::nat), (0::nat)))) s) --> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((Suc (Suc (0::nat))) <= MAX)) s) & ((%s. (((%s. (((%(pc, m, e). (pc = ((0::nat), (Suc (0::nat))))) s) --> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((0::nat) <= MAX)) s) & ((%s. (((%s. (((%(pc, m, e). (pc = ((0::nat), (Suc (Suc (0::nat)))))) s) --> ((%(pc, m, e). ((%s. (((%s. True) s) & ((%s. (((%(pc, m, e). ((EX b. (((m B) = (NAT b)) & (b <= B0))) & (((m B) = (NAT B0)) & (((m C) = (NAT C0)) & ((m A) = (NAT (0::nat))))))) s) & ((%s. True) s))) s))) (((0::nat), (Suc (Suc (Suc (0::nat))))), (update m (Suc (Suc (Suc (Suc (Suc (Suc (0::nat))))))) (NAT (0::nat))), (h-update ((h e) @ [((0::nat), (Suc (Suc (0::nat))))]) e)))) s))) s) & ((%s. True) s))) s))) (((0::nat), (Suc (Suc (0::nat)))), (update m (Suc (0::nat)) (NAT (Suc (Suc (0::nat))))), (h-update ((h e) @ [((0::nat), (Suc (0::nat)))]) e)))) s))) s) & ((%s. True) s))) s))) (((0::nat), (Suc (0::nat))), (update m (0::nat) (NAT (Suc (Suc (Suc (Suc (0::nat))))))), (h-update ((h e) @ [((0::nat), (0::nat))]) e)))) s))) s) & ((%s. True) s))) s))) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%s. (((%s. True) s) & ((%s. (((%(pc, m, e). ((EX b. (((m B) = (NAT b)) & (b <= B0))) & (((m B) = (NAT B0)) & (((m C) = (NAT C0)) & ((m A) = (NAT (0::nat))))))) s) & ((%s. True) s))) s))) s) & ((%s. (((%(pc, m, e). (pc = ((0::nat), (Suc (Suc (Suc (0::nat))))))) s) &
((%s. True) s))) s))) s) --> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))))))))) <= MAX)) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. ((m C) = (NAT c))) & (ALL x. ((x ~= P) --> ((m x) = (callmem e x)))))))) s) & ((%s. True) s))) s))) (((Suc (0::nat)), (0::nat)), (update m (Suc (Suc (Suc (Suc (Suc (0::nat)))))) (RA ((0::nat), (Suc (Suc (Suc (Suc (0::nat))))))), (cs-update (((length (h e)), m) # (cs e)) (h-update ((h e) @ [((0::nat), (Suc (Suc (Suc (0::nat)))))]) e))))) s))) s) & ((%s. True) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). ((EX n. ((m (Suc (Suc (Suc (Suc (0::nat)))))) = (NAT n))) & (EX n. ((m (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))) = (NAT n))))) s) & ((%s. (((%(pc, m, e). ((EX b. (((m B) = (NAT b)) & (b <= B0))) & (((m B) = (NAT B0)) & (((m C) = (NAT C0)) & (((m A) = (NAT (0::nat))) & (EX f. (((m F) = (NAT f)) & ((f = (0::nat)) --> (EX b. (((m B) = (NAT b)) &
((MAX − C0) < b))))))))))) s) & ((%s. True) s))) s))) s) & ((%s. (((%(pc, m, e). ((EX n n'. (((m (Suc (Suc (Suc (Suc (0::nat)))))) = (NAT n)) & (((m (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))) = (NAT n')) & (n < n')))) & (pc = ((0::nat), (Suc (Suc (Suc (Suc (0::nat))))))))) s) & ((%s. True) s))) s))) s) --> ((%(pc, m, e). ((%s. (((%s. True) s) & ((%s. (((%s. True) s) & ((%s. True) s))) s))) (((0::nat), (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))), m, (h-update ((h e) @ [((0::nat), (Suc (Suc (Suc (Suc (0::nat))))))]) e)))) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). ((EX n. ((m (Suc (Suc (Suc (Suc (0::nat)))))) =

(NAT n))) & (EX n. ((m (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))) = (NAT n))))) s) & ((%s. (((%(pc, m, e). ((EX b. (((m B) = (NAT b)) & (b <= B0))) & (((m B) = (NAT B0)) & (((m C) = (NAT C0)) & (((m A) = (NAT (0::nat))) &

(EX f. (((m F) = (NAT f)) & ((f = (0::nat)) --> (EX b. (((m B) = (NAT b)) & ((MAX - C0) < b))))))))))) s) & ((%s. True) s))) s))) s) & ((%s. (((%(pc, m, e). ((EX n n'. (((m (Suc (Suc (Suc (Suc (0::nat)))))) = (NAT n)) & (((m (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))) = (NAT n')) & (~ (n < n'))))) & (pc = ((0::nat), (Suc (Suc (Suc (Suc (0::nat))))))))) s) & ((%s. True) s))) s))) s) --> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((EX n. ((m (0::nat)) = (NAT n))) & ((EX n. ((m (Suc (0::nat))) = (NAT n))) & (EX n. (((lift op + (m (0::nat)) (m (Suc (0::nat)))) = (NAT n)) & (n <= MAX)))))) s) & ((%s. (((%s. (((%(pc, m, e). (pc = ((0::nat), (Suc (Suc (Suc (Suc (Suc (0::nat)))))))))) s) --> ((%(pc, m, e). ((%s. (((%s. True) s) & ((%s. (((%s. True) s) &
((%s. True) s))) s))) (((0::nat), (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))), (update m (0::nat) (lift op + (m (0::nat)) (m (Suc (0::nat))))), (h-update ((h e) @ [((0::nat), (Suc (Suc (Suc (Suc (Suc (0::nat)))))))]) e)))) s))) s) & ((%s. True) s))) s))) (((0::nat), (Suc (Suc (Suc (Suc (Suc (0::nat))))))), m, (h-update ((h e) @ [((0::nat), (Suc (Suc (Suc (Suc (0::nat))))))]) e)))) s))) s) & ((%s. True) s))) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%s. (((%s. True) s) & ((%s. (((%s. True) s) & ((%s. True) s))) s))) s) & ((%s. (((%(pc, m, e). (pc = ((0::nat), (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))))) s) & ((%s. True) s))) s))) s) --> ((%(pc, m, e). ((%s. (((%s. True) s) & ((%s. (((%s. True) s) & ((%s. True) s))) s))) (((0::nat), (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))), m, (h-update ((h e) @ [((0::nat), (Suc (Suc (Suc (Suc (Suc (Suc (0::nat))))))))]) e)))) s))) s) & ((%s. True) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). ((Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (0::nat))))))))))))))) <= MAX)) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) &
((EX b. ((m B) = (NAT b))) & ((EX c. ((m C) = (NAT c))) & (ALL x. ((x ~= P) --> ((m x) = (callmem e x)))))))) s) & ((%s. True) s))) s))) s) & ((%s. (((%(pc, m, e). (pc = ((Suc (0::nat)), (0::nat)))) s) & ((%s. True) s))) s))) s) --> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((EX n. ((m (Suc (Suc (0::nat)))) = (NAT n))) & (EX n. ((m (Suc (0::nat))) = (NAT n))))) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. ((m C) = (NAT c))) & (((m M) = (NAT MAX)) & (ALL x. (((x ~= M) & (x ~= P)) --> ((m x) = (callmem e x))))))))) s) & ((%s. True) s))) s))) (((Suc (0::nat)), (Suc (0::nat))), (update m (Suc (Suc (0::nat))) (NAT (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))))))))))), (h-update ((h e) @ [((Suc (0::nat)), (0::nat))]) e)))) s))) s) & ((%s. True) s))) s) &
((%s. (((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). ((EX n. ((m (Suc (Suc (0::nat)))) = (NAT n))) & (EX n. ((m (Suc (0::nat))) = (NAT n))))) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. ((m C) = (NAT c))) & (((m M) = (NAT MAX)) & (ALL x. (((x ~= M) & (x ~= P)) --> ((m x) = (callmem e x))))))))) s) & ((%s. True) s))) s))) s) & ((%s. (((%(pc, m, e). (pc = ((Suc (0::nat)), (Suc (0::nat))))) s) & ((%s. True) s))) s))) s) --> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((0::nat) <= MAX)) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. (((m C) = (NAT c)) &
((m M) = (NAT (MAX - c))))) & (ALL x. (((x ~= M) & (x ~= P)) --> ((m

x) = (callmem e x)))))))) s) & ((%s. True) s))) s))) (((Suc (0::nat)), (Suc (Suc (0::nat)))), (update m (Suc (Suc (0::nat))) (lift op − (m (Suc (Suc (0::nat)))) (m (Suc (0::nat))))), (h-update ((h e) @ [((Suc (0::nat)), (Suc (0::nat)))]) e)))) s))) s) & ((%s. True) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). ((0::nat) <= MAX)) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. (((m C) = (NAT c)) & ((m M) = (NAT (MAX − c))))) & (ALL x. (((x ~= M) & (x ~= P)) −−> ((m x) = (callmem e x)))))))) s) & ((%s. True) s))) s))) s) &

((%s. (((%(pc, m, e). (pc = ((Suc (0::nat)), (Suc (Suc (0::nat)))))) s) & ((%s. True) s))) s))) s) −−> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((EX n. ((m (Suc (Suc (0::nat)))) = (NAT n))) & (EX n. ((m (0::nat)) = (NAT n))))) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. (((m C) = (NAT c)) & ((m M) = (NAT (MAX − c))))) & ((ALL x. (((x ~= F) & ((x ~= M) & (x ~= P))) −−> ((m x) = (callmem e x)))) & ((m F) = (NAT (0::nat)))))))) s) & ((%s. True) s))) s))) (((Suc (0::nat)), (Suc (Suc (Suc (0::nat))))), (update m (Suc (Suc (Suc (Suc (0::nat))))) (NAT (0::nat))), (h-update ((h e) @ [((Suc (0::nat)), (Suc (Suc (0::nat))))]) e)))) s))) s) & ((%s. True) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). ((EX n. ((m (Suc (Suc (0::nat)))) = (NAT n))) & (EX n. ((m (0::nat)) = (NAT n))))) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. (((m C) = (NAT c)) & ((m M) = (NAT (MAX − c))))) & ((ALL x. (((x ~= F) & ((x ~= M) & (x ~= P))) −−> ((m x) = (callmem e x)))) & ((m F) = (NAT (0::nat)))))))) s) & ((%s. True) s))) s))) s) &

((%s. (((%(pc, m, e). ((EX n n'. (((m (Suc (Suc (0::nat)))) = (NAT n)) & (((m (0::nat)) = (NAT n')) & (n < n')))) & (pc = ((Suc (0::nat)), (Suc (Suc (Suc (0::nat)))))))) s) & ((%s. True) s))) s))) s) −−> ((%(pc, m, e). ((%s. (((%(pc, m, e). (EX pn' i'. (((m (Suc (Suc (Suc (Suc (Suc (0::nat))))))) = (RA (pn', (Suc i')))) & (EX k m' cl css. (((cs e) = ((k, m') # (cl # css))) & ((pn', i') = ((h e) ! k))))))) s) & ((%s. (((%(pc, m, e). ((ALL x. (((x ~= F) & ((x ~= M) & (x ~= P))) −−> ((m x) = (callmem e x)))) & ((((m F) = (NAT (0::nat))) | ((m F) = (NAT (1::nat)))) & (EX c b. (((m C) = (NAT c)) & (((m B) = (NAT b)) & (((m F) = (NAT (0::nat))) −−> ((MAX − c) < b)))))))) s) & ((%s. True) s))) s))) (((Suc (0::nat)), (Suc (Suc (Suc (Suc (Suc (0::nat))))))), m, (h-update ((h e) @ [((Suc (0::nat)), (Suc (Suc (Suc (0::nat)))))]) e)))) s))) s) &

((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). ((EX n. ((m (Suc (Suc (0::nat)))) = (NAT n))) & (EX n. ((m (0::nat)) = (NAT n))))) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. (((m C) = (NAT c)) & ((m M) = (NAT (MAX − c))))) & ((ALL x. (((x ~= F) & ((x ~= M) & (x ~= P))) −−> ((m x) = (callmem e x)))) & ((m F) = (NAT (0::nat)))))))) s) & ((%s. True) s))) s))) s) & ((%s. (((%(pc, m, e). ((EX n n'. (((m (Suc (Suc (0::nat)))) = (NAT n)) & (((m (0::nat)) = (NAT n')) & (~ (n < n'))))) & (pc = ((Suc (0::nat)), (Suc (Suc (Suc (0::nat)))))))) s) & ((%s. True) s))) s))) s) −−> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((Suc (0::nat)) <= MAX)) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. ((m C) = (NAT c))) & (ALL x. (((x ~= F) & ((x ~= M) & (x ~= P))) −−> ((m x) = (callmem e x)))))))) s) & ((%s. True) s))) s))) (((Suc (0::nat)), (Suc (Suc (Suc (Suc (0::nat)))))), m, (h-update ((h e) @ [((Suc (0::nat)), (Suc (Suc (Suc (0::nat)))))]) e)))) s))) s) &

((%s. True) s))) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). ((Suc

(0::nat)) <= MAX)) s) & ((%s. (((%(pc, m, e). (((m P) = (RA (incA (callpc e)))) & ((EX b. ((m B) = (NAT b))) & ((EX c. ((m C) = (NAT c))) & (ALL x. (((x ~= F) & ((x ~= M) & (x ~= P))) --> ((m x) = (callmem e x)))))))) s) & ((%s. True) s))) s))) s) & ((%s. (((%(pc, m, e). (pc = ((Suc (0::nat)), (Suc (Suc (Suc (Suc (0::nat)))))))) s) & ((%s. True) s))) s))) s) --> ((%(pc, m, e). ((%s. (((%(pc, m, e). (EX pn' i'. (((m (Suc (Suc (Suc (Suc (Suc (0::nat))))))) = (RA (pn', (Suc i')))) & (EX k m' cl css. (((cs e) = ((k, m') # (cl # css))) & ((pn', i') = ((h e) ! k))))))) s) & ((%s. (((%(pc, m, e). ((ALL x. (((x ~= F) & ((x ~= M) & (x ~= P))) --> ((m x) = (callmem e x)))) & ((((m F) = (NAT (0::nat))) | ((m F) = (NAT (1::nat)))) & (EX c b. (((m C) = (NAT c)) & (((m B) = (NAT b)) &
(((m F) = (NAT (0::nat))) --> ((MAX - c) < b)))))))) s) & ((%s. True) s))) s))) (((Suc (0::nat)), (Suc (Suc (Suc (Suc (Suc (0::nat)))))), (update m (Suc (Suc (Suc (Suc (0::nat))))) (NAT (Suc (0::nat)))), (h-update ((h e) @ [((Suc (0::nat)), (Suc (Suc (Suc (Suc (0::nat))))))]) e)))) s))) s) & ((%s. True) s))) s) & ((%s. (((%s. (((%s. (((%s. (((%s. (((%(pc, m, e). (EX pn' i'. (((m (Suc (Suc (Suc (Suc (Suc (0::nat))))))) = (RA (pn', (Suc i')))) & (EX k m' cl css. (((cs e) = ((k, m') # (cl # css))) & ((pn', i') = ((h e) ! k))))))) s) & ((%s. (((%(pc, m, e). ((ALL x. (((x ~= F) & ((x ~= M) & (x ~= P))) --> ((m x) = (callmem e x)))) & ((((m F) = (NAT (0::nat))) | ((m F) = (NAT (1::nat)))) & (EX c b. (((m C) = (NAT c)) & (((m B) = (NAT b)) & (((m F) = (NAT (0::nat))) --> ((MAX - c) < b)))))))) s) & ((%s. True) s))) s))) s) & ((%s. (((%s. (((%(pc, m, e). (((m (Suc (Suc (Suc (Suc (Suc (0::nat))))))) = (RA ((0::nat), (Suc (Suc (Suc (Suc (0::nat)))))))) &
(pc = ((Suc (0::nat)), (Suc (Suc (Suc (Suc (Suc (0::nat)))))))))) s) & ((%s. (((%(pc, m, e). ((%(pc, m, e). ((EX b. (((m B) = (NAT b)) & (b <= B0))) & (((m B) = (NAT B0)) & (((m C) = (NAT C0)) & ((m A) = (NAT (0::nat))))))) (let (k, m') = (hd (cs e)); cs' = (tl (cs e)); h' = (take k (h e)) in (((h e) ! k), m', (h-update h' (cs-update cs' e)))))) s) & ((%s. True) s))) s))) s) & ((%s. True) s))) s))) s) --> ((%(pc, m, e). ((%s. (((%(pc, m, e). ((EX n. ((m (Suc (Suc (Suc (Suc (0::nat)))))) = (NAT n))) & (EX n. ((m (Suc (Suc (Suc (Suc (Suc (Suc (0::nat)))))))) = (NAT n))))) s) & ((%s. (((%(pc, m, e). ((EX b. (((m B) = (NAT b)) & (b <= B0))) & (((m B) = (NAT B0)) & (((m C) = (NAT C0)) & (((m A) = (NAT (0::nat))) & (EX f. (((m F) = (NAT f)) & ((f = (0::nat)) --> (EX b. (((m B) = (NAT b)) & ((MAX - C0) < b))))))))))) s) & ((%s. True) s))) s))) (((0::nat), (Suc (Suc (Suc (Suc (0::nat)))))), m, (cs-update (tl (cs e)) (h-update ((h e) @ [((Suc (0::nat)), (Suc (Suc (Suc (Suc (Suc (0::nat)))))))]) e))))) s))) s) & ((%s. True) s))) s) & ((%s. True) s))) s))) s))) s))) s))) s))) s))) s))) s))) s)))

1.3 Program Verification

First we ensure that the program is wellformed.

theorem *wf-prog*:
wf prog
apply (*simp add*: *wf-def checkPos.simps prog-def Let-def split-def fst-conv snd-conv cmd.simps domC.simps ret-succs.simps callpoints-def isCall-def anF.simps*)
done

Then, we prove the verification condition

Manually Guided Proof

lemma *vc-prog-holds*:
provable prog vc
— start up
apply (*simp add*: *provable-def valid-def* | *rule allI* | *rule impI*)+
apply (*rename-tac pn i m e*)
apply (*cut-tac wf-prog*)
apply (*drule vc-proof-startup*)
apply *assumption*
apply (*erule conjE* | *erule exE*)+
apply (*simp only*: *vc-def*)

— main proof
apply (*case-tac prog*,((*pn*,*i*),*m*,*e*) $\models$ (*initF prog*))
apply (*rule context-conjI*)
— (initF prg) implies (isafeF prg (ipc prg))
apply (*simp add*: *initF-def valid-def B-def MAX-def B0-def A-def C-def C0-def update-def fun-upd-apply*)

apply (*simp only*: *initF-def valid-def split-def fst-conv snd-conv*)
apply (*erule conjE*)+
apply *simp*

— case not (initF prg)
apply (*erule isafeP-elims*)
apply *simp*

apply (*rule conjI*)
apply (*rule impI*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def update-def fun-upd-apply*)

apply (*rule conjI*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append*)

apply (*rule conjI*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def*)

apply (*rule conjI*)
apply (*case-tac css*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)

— case "css = a list"
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)

apply (*rule conjI*)
apply (*simp add*: *split-def fst-conv snd-conv*)
apply (*rule impI*)+
apply (*erule conjE* | *erule exE*)+
apply (*drule-tac t=s''* **in** *sym*)
apply (*case-tac css*)

apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def sysinv.simps*)

apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def sysinv.simps sysinv2.simps*)

apply (*rule conjI*)
apply (*simp add*: *split-def fst-conv snd-conv*)
apply (*rule impI*)+
apply (*erule conjE* | *erule exE*)+
apply (*drule-tac t=s''* **in** *sym*)
apply (*case-tac css*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)
apply (*simp add*: *nat-number split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def F-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)

apply (*rule conjI*)
apply (*simp add*: *split-def fst-conv snd-conv*)
apply (*rule impI*)+
apply (*erule conjE* | *erule exE*)+
apply (*drule-tac t=s''* **in** *sym*)
apply (*case-tac css*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)

apply (*simp add*: *nat-number split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def F-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)

apply (*rule conjI*)
apply (*simp add*: *split-def fst-conv snd-conv*)

apply (*drule-tac t=s''* **in** *sym*)
apply (*case-tac css*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)
apply (*simp add*: *nat-number split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def F-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)

apply (*rule conjI*)
apply (*rule impI*)
apply (*erule conjE* |*erule exE*)+
apply *fastsimp*

apply *fastsimp*

apply (*rule conjI*)
apply (*simp add*: *split-def fst-conv snd-conv*)
apply (*drule-tac t=s''* **in** *sym*)
apply (*case-tac css*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)
apply (*simp add*: *nat-number split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def F-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)
apply *fastsimp*

apply *simp*
apply (*rule impI*)
apply (*erule conjE* | *erule exE*)+
apply (*drule-tac t=s''* **in** *sym*)
apply (*case-tac css*)
apply (*simp add*: *split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)
apply (*simp add*: *nat-number split-def fst-conv snd-conv MAX-def C0-def B0-def B-def A-def C-def P-def F-def M-def update-def fun-upd-apply incA-def callpc-def callmem-def callenv-def nth-append lift-def Let-def*)
apply *fastsimp*
done

end