Raport Științific și Tehnic

privind implementarea proiectului
PN-III-P2-2.1-PED-2019-0955
în perioada 01/01/2022–22/10/2022

Implementarea etapei finale a proiectului *Verificare Formală Bazată pe Componente* (*Component-Based Formal Verification*), ce are în vedere dezvoltarea unui limbaj numit COMP și a unor instrumente software asociate corespunzătoare paradigmei de specificare pe bază de obiecte comportamentale, s-a realizat în perioada 1 ianuarie 2022 – 22 octombrie 2022, după cum urmează:

Etapa 3, dedicată, pe de o parte, implementării tehnicilor de verificare formală aferente limbajului COMP și testării și validării lor printr-un studiu de caz și printr-o serie de exemple, și pe de altă parte, diseminării rezultatelor.

Activitățile acestei etape sunt discutate în cele ce urmează în secțiuni separate, conform planului de realizare a proiectului. Prezentări detaliate, care tratează tehnicile de verificare integrate în COMP, instalarea și utilizarea interpretorului, precum și validarea tool-ului prin specificarea și verificarea formală a unui sistem concret de dimensiuni medii, se găsesc în [DŢ22a, DŢ22b] – documente disponibile pe pagina web a proiectului: `http://www.imar.ro/~diacon/COMPproject`.

Etapa 3 – Verificare formală (2022)

Act. 3.1 – Suport pentru metodologii de verificare

Verificarea proprietăților comportamentale ale unui obiect dat este, în forma sa generală, o sarcină imposibil de rezolvat prin mijloace mecanice – deoarece problema satisfiabilității comportamentale nu este recursiv enumerabilă [BR00]. Totuși, soluții viabile există – sub forma unor metode efective de calcul – pentru clase particulare de obiecte, dar suficient de bogate pentru a include o gamă largă de cazuri interesante din punct de vedere practic. Limbajul COMP reifică această abordare prin intermediul unor operatori dedicați de compunere de obiecte și prin metodologii de specificare a

obiectelor care asigură o importantă proprietate de compoziționalitate a relației de echivalență comportamentală [Dia05]. Aceste metodologii, împreună cu proprietatea de compoziționalitate, sunt discutate pe larg în documentul ce conține definiția limbajului COMP [DŢ21]. Metodologiile de verificare pun în valoare proprietatea de compoziționalitate și stabilesc fundamentele unei tehnologii automate de verificare a proprietăților comportamentale – tehnologie implementată în cadrul acestui proiect.

Mai precis, sistemul COMP permite verificarea automată a unor proprietăți de forma $\forall X \cdot t \sim t'$ (echivalențe comportamentale), unde t și t' sunt termeni ce denotă stări ale unui obiect comportamental `B` pentru care dorim să verificăm o astfel de proprietate. Reușita verificării automate depinde de formalizarea obiectului `B` în conformitate cu metodologiile de specificare ale limbajului COMP.

În contextul metodologiilor de specificare descrise în [DŢ21] (și dezvoltate în 2021, în Etapa 2 de implementare a proiectului), metoda generală de verificare constă în descompunerea obiectului `B` și în 'proiectarea' echisatisfiabilă a proprietății comportamentale inițiale într-o serie de alte proprietăți, câte una pentru fiecare componentă a obiectului `B`. Procedeul se repetă atât timp cât obiectul curent este compus. Pentru obiectele de bază (fără componente), verificarea proprietății comportamentale poate fi redusă în continuare la calcule ecuaționale: pentru o echivalență comportamentală $\forall X \cdot t \sim t'$ ca mai sus, este suficient să probăm egalitatea strictă $\forall X \cup W \cdot \zeta(t, W) = \zeta(t', W)$ pentru orice `B`-observație ζ – amintim că, în acest caz, W este un tuplu de variabile noi corespunzătoare parametrilor observației ζ.

Sistemul COMP implementează această metodă de verificare prin comanda **`check`**. Din punct de vedere sintactic, forma generală a comenzii este:

`check` ⟨*property*⟩ [**`forall`** ⟨*pre-constr*⟩] [**`given`** ⟨*post-constr*⟩] **`.`**

unde ⟨*property*⟩ este proprietatea de verificat, iar ⟨*pre-constr*⟩ și ⟨*post-constr*⟩ sunt constrângeri opționale asupra variabilelor folosite în ⟨*property*⟩, precum și asupra variabilelor generate automat în cursul verificării pentru compuneri indexate de obiecte. Constrângerile pot conține egalități, conjuncții, disjuncții, etc.

De exemplu, pentru a verifica dacă obiectul `ACCOUNT-SYS-TRANSFER` prezentat în [DŢ21] garantează că transferul a `N` unități bancare (dintr-un cont `A` într-un cont `B`) imediat după depunerea lor în contul `A` este echivalent cu depunerea a `N` unități bancare direct în contul `B` (evitând astfel contul `A`), putem evalua în COMP:

```
open ACCOUNT-SYS-TRANSFER
  let ax N:Nat <= N:Nat + M:Nat = true [label: Lemma-1] .
  let ax (N:Nat + M:Nat) - N:Nat = M:Nat [label: Lemma-2] .
  check transfer(A/deposit(X:State, N:Nat), N:Nat)
      ~ B/deposit(X:State, N:Nat) .
close
```

Prima linie din fragmentul de cod de mai sus indică interpretorului (prin cuvântul cheie `open`) obiectul pe care dorim să-l analizăm (în acest caz, `ACCOUNT-SYS-TRANSFER`). Următoarele două linii introduc (prin cuvântul cheie `let`) leme de natură ecuațională ce urmează a fi folosite în cursul verificării automate. Acestea sunt triviale și pot fi verificate separat. Ultima linie marchează finalul demonstrației.

Act. 3.2 – Metodologii de debugging

Rezultatul unei verificări formale poate fi fie `Proved! The property holds`, fie `Disproved!`, sau `Open!`. Primul caz corespunde, desigur, unei verificări cu succes: obiectul are proprietatea verificată – mai precis, proprietatea este consecință semantică (potrivit fundamentelor matematice ale limbajului COMP) a specificației obiectului. Celelalte două cazuri corespund unui eșec: ori obiectul nu are proprietatea studiată, ori verificatorul sistemului COMP (implementat prin `check`) nu poate calcula un răspuns definitiv – ceea ce indică absența unor leme sau constrângeri suplimentare.

De exemplu, o verificare (fără ipotezele `Lemma-1` și `Lemma-2` de mai sus) de forma

```
open ACCOUNT-SYS-TRANSFER
  check transfer(A/deposit(X:State, N:Nat), N:Nat)
      ~ B/deposit(X:State, N:Nat) .
close
```

produce rezultatul `Open!`. Metodologiile de debugging dezvoltate pentru limbajul COMP facilitează descoperirea unor ipoteze lipsă precum `Lemma-1` și `Lemma-2`; iar în cazul unui rezultat de tip `Disproved!`, facilitează descoperirea unor eventuale erori de specificare sau de formalizare a proprietății comportamentale studiate.

Sistemul COMP oferă trei metode generale de detectare și raportare a erorilor de specificare sau verificare. Două dintre ele sunt directe: pentru fiecare verificare care se încheie fără un răspuns pozitiv, sistemul prezintă utilizatorului un rezumat al verificării ce include ultima egalitate strictă analizată și un *proof trace* care indică locul (în ierarhia de obiecte) pentru care egalitatea nu a putut fi confirmată. Cea de-a treia metodă este indirectă: sistemul raportează numărul de egalități (comportamentale, respectiv stricte) analizate pentru fiecare nivel al ierarhiei de obiecte.

Pentru a distinge cazurile `Disproved!` și `Open!`, COMP implementează constructori de tipuri de date. De exemplu, în următoarea specificație a numerelor naturale

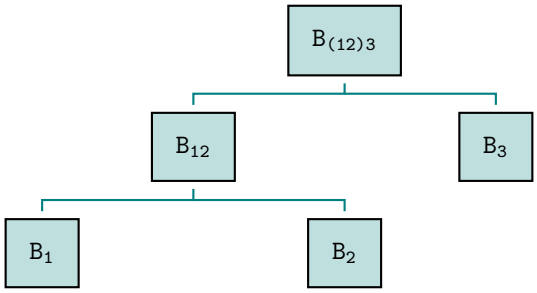
```
data NAT is
  sort Nat .
  op 0 : () -> Nat [ctor] .
  op s_ : Nat -> Nat [ctor] .
enddata
```

operațiile `0` și `s_` sunt declarate cu atributul **`ctor`**, ceea ce denotă că orice număr natural este obținut din zero (`0`) prin aplicări repetate ale operației succesor (`s_`). Toate celelalte operații cu numere naturale sunt declarate fără atributul **`ctor`**. De exemplu, adunarea este definită în modulul `NAT/ADDITION` astfel:

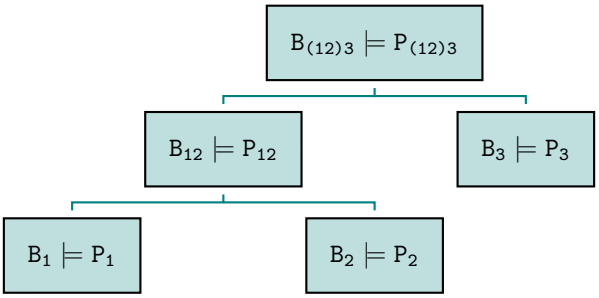
```
data NAT/ADDITION is
  protecting NAT .
  vars M, N : Nat .
  op _+_ : Nat Nat -> Nat [assoc comm] .
  ax 0 + N = N .
  ax (s M) + N = s (M + N) .
enddata
```

În demonstrații, prezența constructorilor ne permite să distingem egalitățile imposibile (`Disproved!`) de cele care nu pot fi determinate (`Open!`). O egalitate de forma $t = t'$ este imposibilă atunci când t și t' sunt termeni diferiți în formă normală obținuți doar din constructori, și nedeterminată atunci când unul dintre cei doi termeni (în formă normală) conține operații derivate (nedeclarate cu atributul **`ctor`**). De exemplu, egalitatea `0 = s 0` este imposibilă ($0 \neq 1$), în timp ce `n = s 0` (unde `n` este o constantă nouă, folosită în cadrul unei demonstrații) este nedeterminată.

Raportarea *proof-trace*-urilor are la bază construcția unui arbore de demonstrații potrivit metodologiei de verificare a limbajului COMP. Concret, pentru un obiect comportamental compus $\mathtt{B}_{(12)3}$ cu următoarea structură ierarhică



și o proprietate $\mathtt{P}_{(12)3}$ de verificat, sistemul COMP generează următorul arbore:



unde $\mathtt{P}_{12}$ este proprietatea corespunzătoare de verificat pentru componenta $\mathtt{B}_{12}$, $\mathtt{P}_3$ este proprietatea corespunzătoare de verificat pentru componenta $\mathtt{B}_3$, și așa mai departe. Rezolvarea problemei inițiale de verificare – $\mathtt{B}_{(12)3} \models \mathtt{P}_{(12)3}$ – se reduce la parcurgerea acestui arbore (de exemplu, în adâncime) și la rezolvarea problemelor (mai simple) corespunzătoare nodurilor terminale. Pentru fiecare proprietate aferentă unui nod terminal care nu poate fi verificată cu succes, un *proof trace* constă dintr-un drum de la rădăcina arborelui de demonstrații la nodul terminal vizat; de exemplu: $\mathtt{B}_{(12)3} \models \mathtt{P}_{(12)3}$, $\mathtt{B}_{12} \models \mathtt{P}_{12}$, $\mathtt{B}_2 \models \mathtt{P}_2$ pentru a indica o eroare în cea de-a doua subcomponentă ($\mathtt{B}_2$) a primei componente ($\mathtt{B}_{12}$) a obiectului analizat.

Act. 3.3 – Suport pentru debugging

Raportarea erorilor de specificare sau de verificare este automată în COMP: în urma fiecărei execuții a comenzii **check**, sistemul prezintă utilizatorului (dacă este cazul) informații de debugging potrivit metodologiei descrise în secțiunea anterioară. De exemplu, în urma rulării scriptului de mai jos (corespunzător verificării proprietății de transfer între cele două tipuri de conturi ale obiectului `ACCOUNT-SYS-TRANSFER`)

```
open ACCOUNT-SYS-TRANSFER
  check transfer(A/deposit(X:State, N:Nat), N:Nat)
      ~ B/deposit(X:State, N:Nat) .
close
```

sistemul generează mesajul:

```
| Open! Could not establish the following properties:
|
| transfer(A/deposit(X:State, N:Nat), N:Nat)
|   ~?~ B/deposit(X:State, N:Nat)
|
| A/Credit(transfer(A/deposit(X:State, N:Nat), N:Nat))
|   ~?~ A/Credit(B/deposit(X:State, N:Nat))
|
| balance(withdraw(deposit(A/Credit(X:State), N:Nat), N:Nat))
|   =?= balance(A/Credit(X:State))
|_________________________________
```

Proof-trace-ul conține în acest caz două egalități comportamentale: prima corespunde rădăcinii arborelui de demonstrații (deci este identică cu proprietatea inițială de verificat), iar cea de-a doua corespunde componentei `A` – fapt semnalat de sistem prin intermediul proiecției `A/Credit` generată automat în urma compunerii

composing (ACCOUNT **as** A) **and** (ACCOUNT **as** B). Ultima egalitate este strictă. Verificatorul nu calculează un răspuns definitiv deoarece nu poate determina:

1. pe de o parte, că N $\leq$ balance(A/Credit(X)) + N – în urma depozitului a N unități bancare în contul de tip A;
2. pe de altă parte, că (balance(A/Credit(X))+N)−N = balance(A/Credit(X)) – în urma transferului celor N unități bancare în contul de tip B.

Ambele sunt proprietăți elementare ale operațiilor cu numere naturale. Pot fi deduse din următoarele axiome declarate în modulele NAT/ORD, NAT/ADDITION, și NAT/SUBTRACTION (incluse în biblioteca de exemple a limbajului COMP):

```
data NAT/ORD is [...]
  op _<=_ : Nat Nat -> Bool .
  ax 0 <= N = true .
  ax (s M) <= 0 = false .
  ax (s M) <= (s N) = M <= N .
enddata

data NAT/ADDITION is [...]
  op _+_ : Nat Nat -> Nat [assoc comm] .
  ax 0 + N = N .
  ax (s M) + N = s (M + N) .
enddata

data NAT/SUBTRACTION is [...]
  op _-_ : Nat Nat -> Nat .
  ax M - 0 = M  .
  ax 0 - N = 0 .
  ax (s M) - (s N) = M - N .
enddata
```

Utilizatorii pot introduce astfel de proprietăți (leme) și în cadrul unei demonstrații, folosind comanda let, potrivit explicațiilor din secțiunea Act. 3.1 de mai sus.

```
open ACCOUNT-SYS-TRANSFER
  let ax N:Nat <= N:Nat + M:Nat = true [label: Lemma-1] .
  let ax (N:Nat + M:Nat) - N:Nat = M:Nat [label: Lemma-2] .
  check transfer(A/deposit(X:State, N:Nat), N:Nat)
      ~ B/deposit(X:State, N:Nat) .
close
```

Rezumatul unei verificări formale poate fi extins astfel încât să includă de asemenea informații referitoare la numărul de egalități comportamentale sau stricte analizate pentru fiecare nivel al ierarhiei de obiecte. Pentru aceasta, COMP furnizează comenzile **`show/hide check stats`**. De exemplu, rulând din nou scriptul demonstrației de mai sus împreună cu **`show check stats`** produce raportul:

```
| Proved! The property holds.
|
| Equations examined for each bobj hierarchy level
| level 0: 1 behavioural and 0 strict
| level 1: 2 behavioural and 1 strict
|____________________________________
```

Astfel de informații contribuie de asemenea la înțelegerea demonstrației: la nivelul 1 al ierarhiei de obiecte, observăm verificarea a două egalități comportamentale (corespunzătoare contului de tip `A`, respectiv de tip `B`) și a unei singure egalități stricte (corespunzătoare contului de tip `A`). Absența unei egalități stricte pentru contul de tip `B` ne arată că egalitatea comportamentală analizată în acest caz este una trivială, de forma $t \sim t$. Într-adevăr, potrivit axiomelor modulului `ACCOUNT-SYS-TRANSFER` pentru proiecții și pentru acțiunile `transfer`, `A/deposit`, și `B/deposit`, avem:

```
B/Credit(transfer(A/deposit(X, N), N))
    = deposit(B/Credit(X), N)
    = B/Credit(B/deposit(X, N))
```

Act. 3.4 – Manualul utilizatorului

A patra activitate a acestei etape de implementare a proiectului a fost dedicată scrierii unui ghid de utilizare a limbajului COMP și a instrumentelor software dezvoltate pentru specificarea și verificarea formală a compunerii ierarhice de obiecte [DŢ22a]. Documentul tehnic este disponibil în format PDF pe pagina web a proiectului, la adresa: `http://www.imar.ro/~diacon/COMPproject/doc/COMP-UserGuide.pdf`.

Manualul utilizatorului cuprinde informații detaliate referitoare la instalarea sistemului COMP și a două tehnologii esențiale pentru rularea interpretorului: Maude (limbajul în care este implementat COMP) și SpeX (o colecție de biblioteci software care facilitează dezvoltarea de limbaje și interpretoare noi precum COMP). În plus, manualul utilizatorului include: (a) o scurtă descriere a limbajului – independentă de documentul tehnic corespunzător fundamentelor matematice ale limbajului [DŢ21]; (b) o prezentare a tehnologiilor de verificare de proprietăți comportamentale integrate în COMP; (c) o prezentare a suportului pentru debugging oferit de COMP; (d) informații despre mesajele de eroare furnizate de interpretor – de exemplu, pentru

specificații sau proprietăți neconforme din punct de vedere sintactic; și (*e*) o listă completă a comenzilor ce pot fi executate de interpretor.

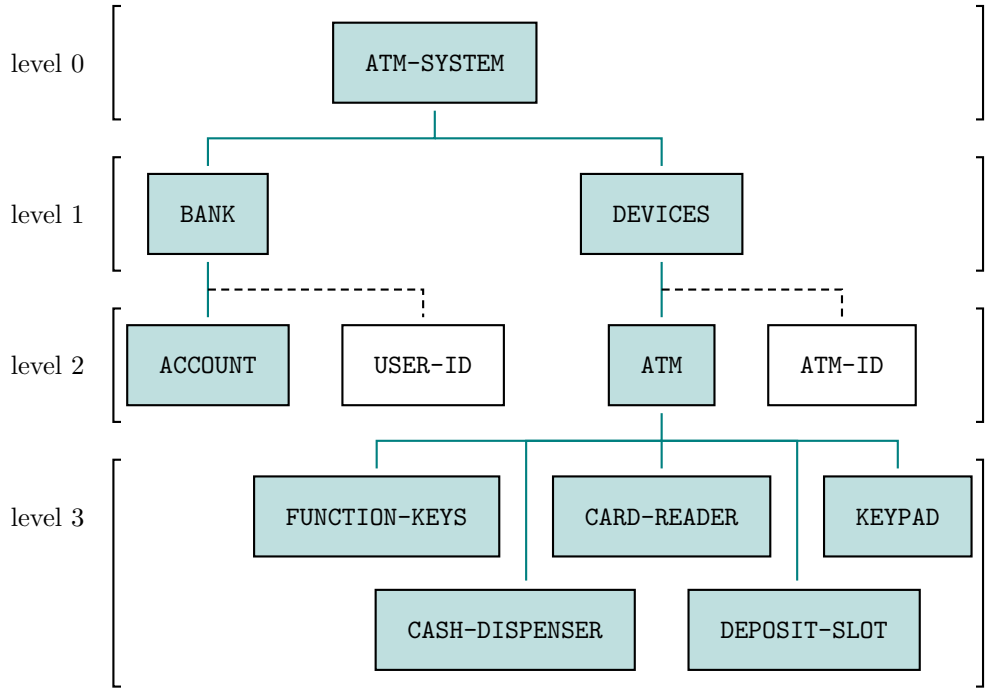
Act. 3.5 – Studiu de caz

Cel mai complex studiu realizat în COMP a avut drept obiectiv specificarea și verificarea formală a unui sistem de ATM-uri (automate bancare) cu un număr arbitrar de mare de utilizatori și de automate, și care permite realizarea unor operații bancare de tip depunere/retragere de fonduri prin intermediul automatelor [DŢ22b]. Raportul tehnic este disponibil în format PDF pe pagina web a proiectului, la adresa: `http://www.imar.ro/~diacon/COMPproject/doc/COMP-CaseStudy.pdf`.

Obiectul comportamental rezultat, numit `ATM-SYSTEM`, are o structură ierarhică distribuită pe patru niveluri, potrivit diagramei de mai jos. Fiecare dreptunghi verde-albăstrui reprezintă un obiect, și fiecare dreptunghi alb reprezintă o specificație de tipuri de date (pentru a simplifica diagrama, considerăm doar acele tipuri de date implicate în compuneri de obiecte). Pe scurt, în post-ordine:

`ACCOUNT` descrie conturi bancare. Specificația este similară celei considerate în [DŢ21], însă acțiunile folosesc o notație specială care permite înlănțuirea lor (*method chaining*) în construcția sistemului de ATM-uri și în demonstrații.

`USER-ID` descrie identificatori de utilizatori (folosind numere naturale).



BANK corespunde unei baze de date de conturi bancare deschise pentru utilizatorii înregistrați în sistem.

ATM corespunde unui automat bancar. Modelarea acestor automate reflectă structura lor fizică și se realizează printr-o compunere cu sincronizare a cinci obiecte/-componente: FUNCTION-KEYS (pentru selectarea operației bancare de executat), CARD-READER (pentru citirea cardului), KEYPAD (pentru introducerea PIN-ului sau a altor valori numerice), CASH-DISPENSER (pentru retragerea de fonduri din cont), și DEPOSIT-SLOT (pentru alimentarea contului).

ATM-ID descrie identificatori de automate (folosind de asemenea numere naturale).

DEVICES corespunde unei baze de date de automate bancare. Obiectul este obținut printr-o compunere indexată de ATM-uri.

ATM-SYSTEM descrie sistemul final de conturi și automate bancare obținut prin sincronizarea obiectelor BANK și DEVICES.

În diagramă, fiecare linie solidă indică o compunere de obiecte: fie sincronizată (de exemplu, pentru ATM și ATM-SYSTEM), fie indexată (pentru BANK și DEVICES). Folosim de asemenea linii întrerupte pentru a indica tipurile de date folosite în compunerile de tip **indexing** (identificatori de utilizatori sau de automate).

Tabelul de mai jos oferă o imagine de ansamblu a complexității celor zece obiecte folosite în specificația sistemului de ATM-uri. Pentru fiecare obiect notăm numărul său total de declarații/sorturi/operații/axiome/etc. – inclusiv cele importate din alte module/obiecte – și indicăm în paranteze câte dintre ele sunt generate automat de COMP. După cum este de așteptat, numărul de declarații generate automat este mic pentru obiectele din partea inferioară a ierarhiei (e.g., ACCOUNT sau KEYPAD), dar crește rapid pentru obiectele din partea superioară a ierarhiei (e.g. DEVICES): mai mult de jumătate din specificația finală este generată automat de COMP.

Obiect	Declarații	Sorturi	Operații	Acțiuni	Observații	Axiome
ACCOUNT	53(1)	4(1)	21(0)	2(0)	1(0)	28(0)
BANK	75(11)	6(2)	32(4)	7(2)	2(1)	41(5)
FUNCTION-KEYS	12(1)	2(1)	7(0)	2(0)	1(0)	3(0)
CARD-READER	12(1)	3(1)	7(0)	1(0)	1(0)	2(0)
KEYPAD	10(1)	2(1)	5(0)	1(0)	1(0)	2(0)
CASH-DISPENSER	11(1)	2(1)	6(0)	2(0)	1(0)	3(0)
DEPOSIT-SLOT	10(1)	2(1)	5(0)	1(0)	1(0)	2(0)
ATM	147(63)	11(6)	55(17)	15(7)	10(5)	81(40)
DEVICES	191(99)	12(7)	72(31)	25(15)	15(10)	107(61)
ATM-SYSTEM	303(171)	15(10)	114(58)	48(32)	23(17)	174(103)

Principala proprietate verificată este absența interferențelor între operațiile bancare executate prin intermediul automatelor. În particular, absența interferențelor între orice două retrageri executate de utilizatori diferiți u1 și u2 a oricăror două automate înregistrate în sistem. Pentru aceasta, considerăm mai întâi o codificare a ipotezelor și a unor construcții auxiliare folosite în demonstrație:

```
open ATM-SYSTEM
  * Theorem: ATM withdrawals are non-interfering.

  let op s : () -> State .
  let ops u, u1, u2 : () -> UId .
  let ops a, a1, a2 : () -> AId .
  let ops m, m1, m1', m2, m2', n1, n2 : () -> Nat .

  let op system-state : Nat Nat -> State .
  let ax system-state(M1:Nat, M2:Nat)
       = s >>= add(a)          :: DEVICES
           >>= add(a1)         :: DEVICES
           >>= add(a2)         :: DEVICES
           >>= add(u, m)       :: BANK
           >>= add(u1, M1:Nat) :: BANK
           >>= add(u2, M2:Nat) :: BANK .

  let ax not u = u1 .
  let ax not u = u2 .
  let ax not u1 = unidentified-user .
  let ax not u2 = unidentified-user .
  let ax not u1 = u2 .
  let ax not a = a1 .
  let ax not a = a2 .

  * (We consider several cases, depending on whether the amount
    withdrawn by ui \s (given by ni), is 0, less than or equal to the
    balance of ui's account \s (given by mi), or greater than the
    balance of ui's account \s (given by mi').)
  let ax not n1 = 0 .
  let ax n1 <= m1 = true .
  let ax n1 <= m1' = false .
  let ax not n2 = 0 .
  let ax n2 <= m2 = true .
  let ax n2 <= m2' = false .
```

Scriptul folosit pentru finalizarea demonstrației este listat mai jos. Execuția sa se încheie cu succes după analizarea a 615 egalități comportamentale (dintre care 135 corespund ultimului nivel al ierarhiei de obiecte) și 21 de egalități stricte.

```
  show check stats
  check system-state(M1:Nat, M2:Nat)
        >>= withdraw(u1, A1:AId, N1:Nat)
        >>= withdraw(u2, A2:AId, N2:Nat)
      ~ system-state(M1:Nat, M2:Nat)
        >>= withdraw(u2, A2:AId, N2:Nat)
        >>= withdraw(u1, A1:AId, N1:Nat)
  forall (A1:AId = a1 and (A2:AId = a1 or A2:AId = a2)
          or A1:AId = a and (A2:AId = a or A2:AId = a2))
     and (N1:Nat = n1 and (M1:Nat = m1 or M1:Nat = m1')
          or N1:Nat = 0 and M1:Nat = m1)
     and (N2:Nat = n2 and (M2:Nat = m2 or M2:Nat = m2')
          or N2:Nat = 0 and M2:Nat = m2)
  given  (<BANK/State.ACCOUNT/State>-UI:UId = u
          or <BANK/State.ACCOUNT/State>-UI:UId = u1
          or <BANK/State.ACCOUNT/State>-UI:UId = u2)
     and <DEVICES/State.ATM/State>-AI:AId = a .
  * QED
close
```

Act. 3.6 – Exemple

Pe lângă `ATM-SYSTEM` (dezvoltat în studiul de caz [DŢ22b] și prezentat pe scurt în Act. 3.5), biblioteca de exemple a sistemului COMP mai conține alte 7 ierarhii de obiecte comportamentale ce ilustrează diferite scenarii de utilizare a tool-ului. Fiecare dintre ele este definită într-un fișier separat cu extensia `.comp` disponibil la adresa: `http://imar.ro/~diacon/COMPproject/COMP.html#examples`.

`ABP.comp` conține o specificație a *alternating bit protocol*, un protocol de comunicare în rețea care asigură retrimiterea mesajelor pierdute sau corupte folosind un canal de comunicare de tip FIFO. Principala sa proprietate constă în garantarea livrării mesajelor (atunci când este permisă de canalul de comunicare) în aceeași ordine în care au fost trimise. Include compuneri cu sincronizare.

`BA.comp` este ierarhia folosită în documentul corespunzător definiției limbajului COMP [DŢ21]. Include compuneri paralele, cu sincronizare, și indexate.

`CwS.comp` conține o specificație a unui contor cu comutator (*counter with switch*). Contorul alternează între două moduri de operare (cu incrementare, respectiv cu decrementare) în funcție de poziția comutatorului. Este un exemplu simplu, menit să ilustreze compunerea cu sincronizare.

`QLock.comp` conține o specificație a protocolului de acces *QLock*, o variantă a semaforului binar introdus de Edsger Dijkstra care rezolvă probleme de concurență în sisteme cu mai multe fire de execuție. Include compuneri cu sincronizare și indexate. Este folosit în tutorialul prezentat pe pagina web a proiectului.

`TS.comp` conține o specificație a unui sistem de telefonie. Este un exemplu simplu, menit să ilustreze compunerile indexate.

`VM.comp` conține o specificație a unui automat de vânzare (*vending machine*). Este un exemplu clasic de interacțiune între două obiecte. Pune accentul pe tipurile de date folosite și pe sincronizarea obiectelor.

`Watch.comp` conține o specificație a unui ceas cu două butoane folosite pentru a controla/ajusta indicatoarele ceasului. Include compuneri cu sincronizare și redenumiri. Este folosit în ghidul de utilizare a sistemului COMP.

Act. 3.7 – Diseminare

Ultima activitate de implementare a proiectului a fost dedicată diseminării rezultatelor, stabilirii unor zone de interes comun cu alte echipe de cercetători, și identificării unor noi direcții de dezvoltare în continuarea efortului depus în cadrul acestui proiect. Activitatea include actualizarea paginii web tehnice a proiectului și realizarea unui tutorial de utilizare a tool-ului și limbajului specificare disponibil online la adresa: `http://imar.ro/~diacon/COMPproject/COMP.html#documentation`.

Câteva dintre principalele rezultate ale proiectului au fost prezentate în perioada 28–30 iunie 2022 la a 26-a ediție a workshopului internațional ADT (*Workshop on Algebraic Development Techniques*, `http://wadt2022.web.ua.pt/`). Acesta face parte dintr-o lungă serie de evenimentele majore din domeniul specificațiilor algebrice și are o tradiție bogată care include, printre altele, rezultate și momente importante ale dezvoltării unora dintre cele mai bine cunoscute limbaje moderne de specificare formală, precum CafeOBJ, Maude, CASL, sau DOL. Workshopul a avut loc în Aveiro, Portugalia, și a fost organizat de membri ai departamentului de matematică din cadrul Universității din Aveiro sub egida IFIP WG 1.3 (*Foundations of System Specification*). Rezumatul și slide-urile prezentate sunt disponibile la adresa: `http://imar.ro/~diacon/COMPproject/COMP.html#publications`. În urma expunerii, am primit o invitație de extindere a lucrării astfel încât să poată fi inclusă în volumul post-proceedings *Recent Trends in Algebraic Development Techniques*, ce urmează a fi publicat de Springer în seria *Lecture Notes in Computer Science*.

În plus, am stabilit legături cu cercetători din Germania, Japonia, Portugalia, România, Spania, și UK – o parte dintre ele, prin participarea la WADT. Rezultate parțiale au fost prezentate de asemenea la *Logic Seminar*, un seminar științific organizat de Centrul de Cercetare pentru Logică, Optimizare și Securitate (LOS) din cadrul Universității din București împreună cu Institutul de Matematică „Simion Stoilow" al Academiei Române. La discuții au participat studenți și cercetători de la Universitatea din București, de la Institutul de Matematică „Simion Stoilow" al Academiei Române, și de la Universitatea „Alexandru Ioan Cuza" din Iași.

Alte întâlniri de lucru s-au realizat online, în cadrul unui seminar intern la care au participat cercetători din Japonia, România, și Spania care dezvoltă de asemenea tehnologii de specificare și verificare formală. Aceste întâlniri au promovat atât COMP cât și SpeX, colecția de biblioteci software folosite în implementarea interpretorului COMP, drept platforme șablon pentru o nouă generație de verificatoare interactive de teoreme. Codul sursă este disponibil online sub licență GPL și poate fi descărcat și instalat/compilat potrivit indicațiilor publicate pe pagina web a proiectului, la adresa: `http://imar.ro/~diacon/COMPproject/COMP.html#download`.

Bibliografie

[BR00] Samuel R. Buss and Grigore Roşu. Incompleteness of behavioral logics. In Horst Reichel, editor, *Coalgebraic Methods in Computer Science, CMCS 2000*, volume 33 of *Electronic Notes in Theoretical Computer Science*, pages 61–79. Elsevier, 2000.

[Dia05] Răzvan Diaconescu. Behavioural specification for hierarchical object composition. *Theoretical Computer Science*, 343(3):305–331, 2005.

[DŢ21] Răzvan Diaconescu and Ionuţ Ţuţu. COMP – Language definition: Syntax, semantics, and methodologies, 2021.

[DŢ22a] Răzvan Diaconescu and Ionuţ Ţuţu. COMP – User guide, 2022.

[DŢ22b] Răzvan Diaconescu and Ionuţ Ţuţu. Revisiting ATMs – A COMP case study, 2022.

Director Proiect,

Răzvan Diaconescu