Raport Științific și Tehnic

privind implementarea proiectului
PN-III-P2-2.1-PED-2019-0955
în perioada 23/10/2020–31/12/2021

Implementarea primelor două etape ale 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 23 octombrie 2020 – 31 decembrie 2021, după cum urmează:

Etapa 1, dedicată modelului matematic și definiției limbajului COMP, în perioada 23 octombrie 2020 – 31 decembrie 2020;

Etapa 2, dedicată, pe de o parte, definirii metodologiilor de specificare și verificare, și pe de altă parte, implementării unui compilator pentru limbajul COMP, în perioada 1 ianuarie 2021 – 31 decembrie 2021.

Principalele rezultatele ale acestor etape sunt discutate în cele ce urmează în secțiuni și subsecțiuni separate, conform planului de realizare a proiectului. O prezentare detaliată, care tratează sintaxa și semantica limbajului COMP, dar și metodologii de specificare și verificare în COMP, se găsește în [DŢ21] – document disponibil pe pagina web a proiectului: `http://www.imar.ro/~diacon/COMPproject`.

Etapa 1 – Definiția COMP (2020)

Prima etapă de implementare a proiectului a avut în vedere studierea unor tehnologii *state of the art* din domeniul specificațiilor formale, stabilirea fundamentelor matematice ale limbajului COMP, și definirea riguroasă a sintaxei și semanticii limbajului. Obiectivele acestei etape au fost îndeplinite în întregime, iar rezultatele sunt disponibile în secțiuni dedicate ale documentului tehnic [DŢ21].

Act. 1.1 – Modelul matematic

COMP reunește două tehnologii majore de specificare: *specificare algebrică*, folosită pentru a descrie tipuri de date; și *specificare comportamentală*, folosită pentru a descrie sisteme sub forma unor ansamble de componente. Fiecare tehnologie de bază este bine fundamentată în literatura de specialitate – a se vedea, de exemplu, monografia [ST12] pentru aspecte particulare specificării algebrice, și [GD92, Dia05], printre multe altele, pentru aspecte care țin de specificare comportamentală.

În această activitate, am identificat și detaliat acele concepte și rezultate specifice celor două tehnologii care constituie fundamentul matematic pentru COMP. Demersul este necesar deoarece în ambele cazuri – algebric, respectiv comportamental – avem în vedere subtehnologii și combinații de subtehnologii care sunt particulare proiectului de față. În cazul algebric, discutăm despre o metodă a semanticii inițiale aplicată pentru algebre multi-sortate; iar în cazul comportamental, despre compuneri ierarhice de obiecte comportamentale, care au la rândul lor la bază algebre cu sorturi ascunse.

În general, în cele ce urmează, prin *specificație* (algebrică sau comportamentală) ne referim din punct de vedere matematic la o pereche (Σ, E), unde:

- Σ este o signatură, adică o colecție structurată de simboluri non-logice ce desemnează tipuri de date (sorturi) sau operații cu date, iar
- E este o mulțime finită de propoziții/enunțuri scrise potrivit unei gramatici bine definite (proprii limbajului) folosind simboluri din Σ.

Definițiile concrete ale noțiunilor de signatură și de propoziție variază în funcție de sistemul logic ales: *many-sorted algebra* (MSA) pentru specificații algebrice, sau *hidden algebra* (HA) pentru specificații comportamentale.

Specificații algebrice

În MSA, prin *signatură* înțelegem o pereche (S, F), unde S este o mulțime de elemente pe care le numim *sorturi* (corespunzătoare tipurilor de date), iar F este o mulțime indexată $\{F_{w\to s} \mid w \in S^*, s \in S\}$ de mulțimi $F_{w\to s}$ de *simboluri de operații* de *aritate* (secvență de sorturi) $w \in S^*$ și *sort* $s \in S$.

De exemplu, o signatură algebrică obișnuită pentru specificarea adunării numerelor naturale este $(S_{\texttt{NAT}}, F_{\texttt{NAT}})$ – specificația `NAT` este predefinită în COMP – unde:

- $S_{\texttt{NAT}} = \{\texttt{Nat}\}$ – i.e., cu un singur sort, `Nat`, corespunzător numerelor naturale;
- $F_{\varepsilon\to\texttt{Nat}} = \{\texttt{0}\}$ – i.e., `0` este o *constantă*, sau operație cu aritate vidă (ε);

  $F_{\texttt{Nat}\to\texttt{Nat}} = \{\texttt{s_}\}$ – simbolul `s_` corespunde operației „succesor”;

  $F_{\texttt{Nat Nat}\to\texttt{Nat}} = \{\texttt{_+_}\}$ – simbolul `_+_` corespunde adunării uzuale;
- $F_{w\to\texttt{Nat}} = \emptyset$ pentru orice altă aritate $w \in \{\texttt{Nat}\}^*$.

Fiecare signatură algebrică (S, F) generează o gramatică elementară pentru construcția de (S, F)-*termeni* de sort $s \in S$, după cum urmează:

- fiecare constantă $c \in F_{\varepsilon \to s}$ determină un termen cu același nume, c;
- pentru fiecare (simbol de) operație $\sigma \in F_{w \to s}$ de aritate $w = s_1 \cdots s_n$, și fiecare termen t_i de sort s_i, cu $1 \leq i \leq n$, $\sigma(t_1, \ldots, t_n)$ este termen de sort s; în acest caz, mai spunem că t_i, pentru fiecare $1 \leq i \leq n$, este *subtermen* al lui t.

Atunci când simbolurile de operații conțin *underscore*-uri, acestea sunt folosite în scrieri *prefix/postfix/mixfix* pentru a indica pozițiile subtermenilor în termeni. De exemplu, $(S_{\texttt{NAT}}, F_{\texttt{NAT}})$-termenul `_+_(s_(0), s_(s_(0)))` corespunzător expresiei $1 + 2$ poate fi scris într-o notație mixfix ca `s 0 + s s 0`.

Propozițiile corespunzătoare unei signaturi (S, F) sunt definite inductiv:

- orice doi termeni de același sort determină o *ecuație* (propoziție atomică) $t = t'$;
- pentru orice două propoziții ρ și ρ', $\neg\rho$ (negația propoziției ρ), $\rho \wedge \rho'$ (conjuncția propozițiilor ρ și ρ'), $\rho \vee \rho'$ (disjuncția propozițiilor), și $\rho \Rightarrow \rho'$ (citită „ρ implică ρ'”) sunt propoziții de asemenea;
- pentru orice mulțime finită de *variabile* (constante noi, distincte de cele prezente în F) și orice propoziție ρ pentru signatura extinsă $(S, F \cup X)$, $\forall X \cdot \rho$ – cuantificarea universală a propoziției ρ – este propoziție de asemenea.

Specificații comportamentale

Hidden algebra este o generalizare a abordării uzuale de specificare algebrică multisortată care pune accentul pe comportamentul obiectelor specificate în locul implementării lor. Limbajul se bucură de o expresivitate sporită deoarece permite formalizarea unor enunțuri ce vizează obiecte care nu sunt neapărat egale, ci doar *echivalente comportamental*. Din punct de vedere tehnic, noțiunea mai generală de echivalență comportamentală este facilitată, pe de o parte, de partajarea sorturilor în *sorturi vizibile*, dedicate tipurilor de date, și *sorturi ascunse*, dedicate obiectelor, și pe de altă parte de introducerea unei noțiuni noi de *operație comportamentală*.

În HA, prin *signatură* înțelegem un tuplu (H, V, F, BF), unde:

- H și V sunt mulțimi disjuncte de sorturi ascunse, respectiv vizibile;
- F este o familie de mulțimi de simboluri de operații astfel încât $(H \cup V, F)$ formează o signatură algebrică;
- BF constă din submulțimi $BF_{w \to s} \subseteq F_{w \to s}$ de simboluri de operații comportamentale, pentru fiecare sort $s \in H \cup V$ și fiecare aritate $w \in (H \cup V)^*$ care conține cel puțin un sort ascuns.

De exemplu, în COMP putem descrie conturi bancare (printr-o specificație `ACCOUNT`) folosind următoarea signatură HA (care extinde signatura MSA a numerelor naturale):

- $H_{\texttt{ACCOUNT}} = \{\texttt{Account}\}$;
- $V_{\texttt{ACCOUNT}} = S_{\texttt{NAT}} = \{\texttt{Nat}\}$;
- $(F_{\texttt{ACCOUNT}})_{w \to \texttt{Nat}} = (F_{\texttt{NAT}})_{w \to \texttt{Nat}}$ pentru orice $w \in \{\texttt{Nat}\}^*$;
- $(BF_{\texttt{ACCOUNT}})_{\texttt{Account Nat} \to \texttt{Account}} = \{\texttt{deposit}, \texttt{withdraw}\}$;

  $(BF_{\texttt{ACCOUNT}})_{\texttt{Account} \to \texttt{Nat}} = \{\texttt{balance}\}$;
- $(BF_{\texttt{ACCOUNT}})_{w \to s} = (F_{\texttt{ACCOUNT}})_{w \to s} = \emptyset$ pentru orice altă aritate w și sort s.

În această signatură, sortul `Account` corespunzător conturilor bancare este ascuns. Mai mult, operațiile `deposit`, `withdraw`, și `balance` sunt comportamentale. Primele două sunt considerate *acțiuni* deoarece generează elemente – obiecte – al căror sort este ascuns (`Account`), în timp ce cea de-a treia este considerată *observație* deoarece generează elemente – date – al căror sort este vizibil (`Nat`). În consecință, potrivit [Dia05], două conturi sunt echivalente comportamental dacă și numai dacă au aceeași balanță (via `balance`) în urma aplicării oricărei secvențe de acțiuni de depunere (`deposit`) sau retragere (`withdraw`) de fonduri.

Propozițiile corespunzătoare unei signaturi HA cu componente (H, V, F, BF) sunt definite asemănător propozițiilor signaturii MSA simplificate $(H \cup V, F)$, cu precizarea că în cazul HA considerăm două tipuri de propoziții atomice: ecuații *stricte* de forma $t = t'$, ca în MSA, și ecuații *comportamentale* de forma $t \sim t'$, menite să indice că interpretările termenilor t și t' sunt echivalente comportamental.

Act. 1.2 – Definiția limbajului de specificare formală COMP

O specificație COMP constă dintr-o secvență de module de date sau comportamentale potrivit modelului matematic descris mai sus. În această secțiune, descriem caracteristicile principale ale celor două tipuri de module, prezentăm pe scurt sintaxa COMP, și introducem câteva exemple. Definiția completă este disponibilă în [DȚ21].

Module corespunzătoare tipurilor de date

Aceste module sunt scrise folosind o sintaxă similară modulelor funcționale din limbajul Maude, înlocuind însă cuvinte cheie precum `fmod … endfm` cu `data … enddata`. Definim sintaxa lor folosind o formă extinsă a notației BNF astfel:

```
⟨data-module⟩      ::= data ⟨module-name⟩ is
                           [ ( ⟨data-declaration⟩ . )* ]
                       enddata
```

unde ⟨*module-name*⟩ este un identificator simplu, adică o secvență de caractere ASCII care nu conține simboluri speciale (e.g., spații, virgule, paranteze).

Declarațiile de date sunt fie *importuri* (ce permit reutilizarea unor declarații din module deja scrise) sau declarații specifice limbajului MSA.

⟨*data-declaration*⟩ ::= ⟨*data-import*⟩ | ⟨*msa-declaration*⟩

COMP admite trei tipuri de importuri, similar limbajului Maude. Ordonate de la cel mai restrictiv la cel mai permisiv tip de import, acestea sunt: `protecting`, `extending`, și `including`, potrivit următoarei producții gramaticale:

⟨*data-import*⟩ ::= `protecting` ⟨*module-name*⟩
| `extending` ⟨*module-name*⟩
| `including` ⟨*module-name*⟩

Declarațiile MSA permit extinderea modulului curent cu noi sorturi, operații, sau propoziții. În cazul operațiilor binare, COMP oferă posibilitatea specificării unor atribute ecuaționale precum `assoc` pentru asociativitate, `comm` pentru comutativitate, și `id:` pentru element neutru. În cazul propozițiilor, putem introduce variabile fie direct în termeni, în forma completă ⟨*symbol*⟩`:`⟨*sort*⟩ (fără spații), fie predeclarat, printr-o construcție separată de forma `var` ⟨*symbol*⟩ `:` ⟨*sort*⟩.

⟨*msa-declaration*⟩ ::= `sort` ⟨*sort*⟩
| `op` ⟨*symbol*⟩ `:` ⟨*arity*⟩ `->` ⟨*sort*⟩ [`[` ⟨*attribute*⟩+ `]`]
| `var` ⟨*symbol*⟩ `:` ⟨*sort*⟩
| `ax` ⟨*sentence*⟩

⟨*arity*⟩ ::= `()` | ⟨*sort*⟩+

⟨*attribute*⟩ ::= `assoc` | `comm` | `id:` `(` ⟨*term*⟩ `)`

Următorul fragment de cod oferă un prim exemplu de specificație în COMP. Constă dintr-un singur modul numit `NAT` ce corespunde signaturii algebrice $(S_{\texttt{NAT}}, F_{\texttt{NAT}})$ din secțiunea Act. 1.1 de mai sus. Pe lângă declarațiile corespunzătoare signaturii, modulul conține și două propoziții MSA care axiomatizează în manieră inductivă adunarea numerelor naturale. În notație matematică, acestea sunt:

- $\forall \{N\} \cdot 0 + N = N$, indicând că 0 este element neutru la stânga pentru adunare;
- $\forall \{M, N\} \cdot \texttt{s}\ M + N = \texttt{s}\ (M + N)$, care definește rezultatul adunării la stânga cu un număr natural nenul.

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

Module corespunzătoare obiectelor comportamentale

Spre deosebire de modulele de date, care sunt esențiale specificării algebrice și disponibile și în alte limbaje, declarațiile de obiecte sunt particulare limbajului COMP. Modulele corespunzătoare obiectelor sunt scrise potrivit următoarei reguli:

⟨*bobj-module*⟩ ::= `bobj` ⟨*bobj-name*⟩ [`with states` ⟨*sort*⟩] `is`
[⟨*composition*⟩ `.`]
[(⟨*bobj-declaration*⟩ `.`)*]
`endbo`

Fiecare obiect comportamental definește un sort ascuns al stărilor sale. Utilizatorii sistemului COMP pot declara explicit sortul stărilor, folosind sintaxa opțională „`with states` ⟨*sort*⟩”; altfel, numele implicit al acestui sort este `State`.

Obiectele *de bază*, sau lipsite de componente, sunt scrise fără compuneri, omițând sintaxa opțională „⟨*composition*⟩ .” Ele constau doar din declarații de bază specifice obiectelor comportamentale, adică importuri de date sau declarații HA:

⟨*bobj-declaration*⟩ ::= ⟨*data-import*⟩ | ⟨*ha-declaration*⟩

Declarațiile HA extind declarațiile MSA cu *acțiuni* și *observații*:

⟨*ha-declaration*⟩ ::= ⟨*msa-declaration*⟩
| `act` ⟨*symbol*⟩ `:` ⟨*arity*⟩ `->` ⟨*sort*⟩
| `obs` ⟨*symbol*⟩ `:` ⟨*arity*⟩ `->` ⟨*sort*⟩

Acestea sunt operații comportamentale monadice: aritățile lor conțin un singur sort ascuns (corespunzător unor stări). Fiecare acțiune, atunci când este aplicată unei stări și (eventual, potrivit arității) unor date, are drept rezultat o stare; iar fiecare observație are drept rezultat un element de sort vizibil / tip de date.

Spre exemplu, următorul fragment de cod cuprinde declarația unui obiect comportamental numit `ACCOUNT`, fără componente, corespunzător unui (tip de) cont bancar – a cărui signatură HA a fost prezentată în secțiunea Act. 1.1 de mai sus.

```
bobj ACCOUNT with states Account is
  protecting NAT/OPS .
  act deposit : Account Nat -> Account .
  act withdraw : Account Nat -> Account .
  obs balance : Account -> Nat .
  var A : Account . var N : Nat .
  ax balance(deposit(A, N)) = balance(A) + N .
  ax balance(withdraw(A, N)) = balance(A) - N if N <= balance(A) .
  ax balance(withdraw(A, N)) = balance(A) if N > balance(A) .
endbo
```

Cele trei propoziții descriu efectul acțiunilor de depunere (`deposit`), respectiv retragere (`withdraw`) de fonduri asupra balanței (`balance`) unui cont.

Obiectele compuse sunt obținute prin unul dintre cele trei tipuri de operații de compunere disponibile în COMP: compunere paralelă, cu sincronizare, indexată.

⟨*composition*⟩ ::= ⟨*parallel-comp*⟩ | ⟨*sync-comp*⟩ | ⟨*indexed-comp*⟩

⟨*parallel-comp*⟩ ::= `composing` ⟨*bobj-ref-list*⟩

⟨*sync-comp*⟩ ::= `syncing` (⟨*bobj-name*⟩ | ⟨*bobj-ref-list*⟩)

⟨*indexed-comp*⟩ ::= `indexing` ⟨*bobj-name*⟩ `on` ⟨*module-name*⟩ `by` ⟨*sort*⟩

⟨*bobj-ref*⟩ ::= ⟨*bobj-name*⟩ | `(` ⟨*bobj-name*⟩ `as` ⟨*bobj-name*⟩ `)`

Acestea au drept efect importul declarațiilor de date / comportamentale din obiectele compuse și, în plus, definirea automată a unor operații comportamentale (proiecții, acțiuni, și observații) axiomatizate corespunzător – potrivit documentului tehnic [DŢ21] – care stabilesc legătura dintre obiectul compus și componentele sale.

Etapa 2 – Metodologii și compilator (2021)

Cea de-a doua etapă de implementare a proiectului a avut în vedere, pe de o parte, elaborarea metodologiilor de specificare și verificare proprii limbajului COMP, și pe de altă parte, dezvoltarea principalelor componente software care permit specificarea de obiecte comportamentale: instrumente dedicate unei parsări preliminare a textelor de specificații, analizei statice a codului sursă, și generării automate de cod în urma compunerii de obiecte. Toate obiectivele au fost îndeplinite în întregime; o parte dintre rezultatele (corespunzătoare metodologiilor de specificare și documentației) sunt disponibile în documentul tehnic [DŢ21], iar altele (corespunzătoare implementării în Maude a sistemului COMP) sunt disponibile în depozite de cod sursă la adresele `https://gitlab.com/ittutu/spex` și `https://gitlab.com/ittutu/comp`.

Act. 2.1 – Metodologii de specificare

Din punct de vedere semantic, conceptul matematic de *obiect comportamental* este fundamental pentru limbajul COMP. Acesta corespunde unei specificații comportamentale SP cu un sort ascuns desemnat, pe care îl notăm în general h_{SP}, care surprinde stările obiectului. De exemplu, pentru modulul `ACCOUNT` listat mai sus:

- În rolul specificației SP, considerăm $SP_{\texttt{ACCOUNT}} = (\Sigma_{\texttt{ACCOUNT}}, E_{\texttt{ACCOUNT}})$, unde $\Sigma_{\texttt{ACCOUNT}}$ este signatura descrisă în secțiunea Act. 1.1, iar $E_{\texttt{ACCOUNT}}$ constă din axiomele declarate în modulul `NAT/OPS`, plus următoarele trei propoziții:

  $$\forall \{A, N\} \cdot \texttt{balance}(\texttt{deposit}(A, N)) = \texttt{balance}(A) + N$$
  $$\forall \{A, N\} \cdot N \leq \texttt{balance}(A) \Rightarrow \texttt{balance}(\texttt{withdraw}(A, N)) = \texttt{balance}(A) - N$$
  $$\forall \{A, N\} \cdot N > \texttt{balance}(A) \Rightarrow \texttt{balance}(\texttt{withdraw}(A, N)) = \texttt{balance}(A).$$

  Fiecare dintre aceste propoziții este obținută printr-o simplă transcriere în notație matematică a unei axiome declarate în modulul `ACCOUNT`.

- În rolul sortului h_{SP} considerăm $h_{\texttt{ACCOUNT}} = \texttt{Account}$, sort ascuns introdus în modulul `ACCOUNT` prin intermediul sintaxei „**`with states`** `Account`”.

Metodologiile de specificare prezentate în această secțiune – descrise cu mai multe amănunte în documentul tehnic [DŢ21] – vizează obiectele comportamentale definite mai sus, sunt menite să garanteze caracterizări practice ale relațiilor de echivalență comportamentală, și facilitează procesul de verificare formală caracteristic limbajului COMP. În ansamblu, urmarea acestor metodologii are drept rezultat:

a) stări de bază (adică ale unor obiecte comportamentale fără componente) care sunt echivalente comportamental dacă și numai dacă generează aceleași valori vizibile / date în urma aplicării observațiilor asociate obiectului;

b) stări compuse (ale unor obiecte cu componente) care sunt echivalente comportamental dacă și numai dacă au componente două-câte-două echivalente comportamental. Cu alte cuvinte, pentru a stabili echivalența comportamentală a unor stări compuse, este suficient să verificăm echivalența componentelor lor.

Metodologii de specificare pentru obiecte de bază

Intuitiv, în cazul obiectelor fără componente, metodologia de specificare pe care o observăm în COMP presupune axiomatizarea efectului fiecărei acțiuni asupra tuturor observațiilor aferente obiectului considerat. Din punct de vedere matematic, obiectele specificate potrivit acestei metodologii au axiome care aparțin următoarelor categorii de ecuații (potențial condiționate):

- Pentru fiecare acțiune σ și fiecare observație ζ, considerăm axiome de forma

$$\{\forall\{X, W, W'\} \cdot C^k_{\sigma,\zeta} \Rightarrow \zeta(\sigma(X, W)) = \tau^k_{\sigma,\zeta} \mid 1 \leq k \leq n_{\sigma,\zeta}\}$$

  unde $\tau^k_{\sigma,\zeta}$ și $C^k_{\sigma,\zeta}$ sunt termeni, respectiv condiții, ce nu conțin acțiuni; mai mult, pentru orice algebră A, avem:

$$A \models \forall\{X, W, W'\} \cdot \bigvee_k C^k_{\sigma,\zeta} \qquad \text{(completitudine)}$$

$$A \models \forall\{X, W, W'\} \cdot \neg(C^k_{\sigma,\zeta} \wedge C^{k'}_{\sigma,\zeta}) \qquad \text{(disjunctivitate)}$$

- Pentru fiecare constantă c corespunzătoare unei stări și fiecare observație ζ, considerăm axiome similare celor de mai sus (pentru acțiuni), cu precizarea că termenii τ sunt toți de sort vizibil (potrivit sortului observației ζ).

Proprietățile de *completitudine* și *disjunctivitate* menționate mai sus au rolul de a garanta definirea totală și corectă a efectului acțiunii σ asupra unei observații ζ. Ele asigură, pentru fiecare tuplu de elemente (x, w, w') – corespunzător variabilelor X, W și W' – prezența unei unice axiome $C^k_{\sigma,\zeta} \Rightarrow \zeta(\sigma(X, W)) = \tau^k_{\sigma,\zeta}$ care caracterizează efectul acțiunii σ asupra observației ζ pentru starea x și parametrii adiționali w și w'.

Se poate observa ușor că obiectul `ACCOUNT` este într-adevăr definit conform metodologiei de specificare pentru obiecte de bază. Pentru acțiunea `deposit` și observația `balance`, specificația conține o singură axiomă:

$$\forall\{A, N\} \cdot \texttt{balance}(\texttt{deposit}(A, N)) = \texttt{balance}(A) + N.$$

În acest caz, $n_{\texttt{deposit,balance}} = 1$ iar $C_{\texttt{deposit,balance}} = \mathit{true}$. Pentru acțiunea `withdraw` și observația `balance`, specificația conține două axiome:

$$\forall\{A, N\} \cdot N \leq \texttt{balance}(A) \Rightarrow \texttt{balance}(\texttt{withdraw}(A, N)) = \texttt{balance}(A) - N$$
$$\forall\{A, N\} \cdot N > \texttt{balance}(A) \Rightarrow \texttt{balance}(\texttt{withdraw}(A, N)) = \texttt{balance}(A).$$

Prin urmare, în acest caz, $n_{\texttt{withdraw,balance}} = 2$ iar cele două condiții de axiome sunt $N \leq \texttt{balance}(A)$ și $N > \texttt{balance}(A)$. Acestea îndeplinesc constrângerile de completitudine și disjunctivitate, deoarece relațiile $x \leq y$ și $x > y$ sunt complementare.

Metodologii de specificare pentru obiecte compuse

Compunerile de obiecte au drept consecință extinderea obiectului specificat cu declarații ale componentelor și cu operații și axiome comportamentale care stabilesc conexiuni între obiectul compus și componentele sale. De exemplu, următoarea compunere paralelă a două obiecte/componente `B1` și `B2`:

```
bobj B is
  composing B1 and B2 .
  ...
endbo
```

are drept consecință declararea automată a următoarelor operații:

- pentru fiecare componentă `Bi`, câte o *proiecție* $\pi_i : h_{\mathtt{B}} \to h_{\mathtt{Bi}}$;
- pentru fiecare acțiune $\sigma \in (BF_{\mathtt{Bi}})_{h_{\mathtt{Bi}} w \to h_{\mathtt{Bi}}}$, câte o `B`-acțiune $\sigma_i : h_{\mathtt{B}} w \to h_{\mathtt{B}}$;
- pentru fiecare observație $\zeta \in (BF_{\mathtt{Bi}})_{h_{\mathtt{Bi}} w \to s}$, câte o `B`-observație $\zeta_i : h_{\mathtt{B}} w \to s$;

și a ecuațiilor:

- $\forall \{X, W\} \cdot \pi_i(\sigma_i(X, W)) = \sigma(\pi_i(X), W)$, pentru orice $i \in \{1, 2\}$;
- $\forall \{X, W\} \cdot \pi_j(\sigma_i(X, W)) = \pi_j(X)$, pentru $\{i, j\} = \{1, 2\}$;
- $\forall \{X, W\} \cdot \zeta_i(X, W) = \zeta(\pi_i(X), W)$, pentru orice $i \in \{1, 2\}$.

Proiecțiile π_1 și π_2 stabilesc legătura dintre stările obiectului compus `B` și stările componentelor sale, `B1` și `B2`. Acțiunile σ_i și observațiile ζ_i axiomatizate ca mai sus permit fiecărei componente să funcționeze independent (fără să fie afectată de starea celeilalte componente) în cadrul obiectului compus.

În COMP, compunerile de obiecte pot fi urmate de declarații ale altor acțiuni, observații, operații cu tipuri de date, sau propoziții. Acestea pot fi utilizate, de exemplu, pentru a inițializa obiecte compuse folosind inițializări separate ale componentelor lor; sau pentru a introduce la nivelul unui obiect compus noi observații ale componentelor sale. Orice astfel de declarație trebuie să conserve proprietatea de compoziționalitate menționată la începutul acestei secțiuni: două stări ale unui obiect compus sunt echivalente comportamental dacă și numai dacă proiecțiile lor au drept rezultat sub-stări (ale componentelor) care sunt echivalente comportamental. Potrivit metodologiei de specificare pentru obiecte compuse, această proprietate poate fi garantată prin respectarea următoarelor reguli:

- Obiectul compus `B` nu poate conține nicio altă acțiune în afara celor predefinite.
- Orice `B`-observație este însoțită de o ecuație necondiționată de forma

  $$\forall \{X, W\} \cdot \zeta(X, W) = c_\zeta[\pi_i(X)]$$

  unde $c_\zeta[z]$ este un `Bi`-context (deci z este o variabilă de sort $h_{\mathtt{Bi}}$).

- Orice constantă c de sort $h_{\mathtt{B}}$ este însoțită de câte o ecuație de forma $\pi_i(c) = c_i$, pentru fiecare componentă `Bi`, unde c_i este o constantă de sort $h_{\mathtt{Bi}}$.
- În afara ecuațiilor de mai sus, `B` nu poate conține nicio altă ecuație.

Pentru forme mai complexe de compunere – e.g., cu sincronizare sau indexate – metodologiile de specificare sunt adaptate în mod corespunzător astfel încât să garanteze de asemenea proprietatea de compoziționalitate a relației de echivalență comportamentală. Detaliile acestor adaptări se găsesc în documentul tehnic [DŢ21].

Act. 2.2 – Metodologii de verificare

Sistemul COMP oferă suport pentru verificarea echivalenței comportamentale a două stări date. Scenariul tipic de verificare este de forma

$$\mathtt{B} \models \forall X \cdot t \sim t'$$

unde `B` este un obiect comportamental cu signatura $(H_{\mathtt{B}}, V_{\mathtt{B}}, F_{\mathtt{B}}, BF_{\mathtt{B}})$, X este o mulțime de variabile, iar t și t' sunt termeni de sort $h_{\mathtt{B}}$ peste signatura extinsă $(H_{\mathtt{B}}, V_{\mathtt{B}}, F_{\mathtt{B}} \cup X, BF_{\mathtt{B}})$. Specificarea obiectului `B` în conformitate cu normele metodologice identificate în Act. 2.1 permite simplificarea efortului de verificare potrivit următoarelor două cazuri:

a) dacă `B` nu are componente, atunci t și t' denotă stări echivalente comportamental dacă și numai dacă $\mathtt{B} \models \forall X \cup W \cdot \zeta(t, W) = \zeta(t', W)$ pentru orice `B`-observație ζ – în acest caz, W este un tuplu de variabile noi corespunzătoare parametrilor observației ζ; aceasta reduce verificarea unei proprietăți comportamentale la verificarea unor proprietăți ecuaționale convenționale;

b) dacă `B` este obiect compus, atunci conform proprietății de compoziționalitate a relației de echivalență comportamentală, t și t' denotă stări echivalente dacă și numai dacă proiecțiile lor (stări corespunzătoare `B`-componentelor) sunt echivalente de asemenea; de exemplu, atunci când `B` constă din compunerea paralelă a două obiecte mai simple, `B1` și `B2`, obținem sarcinile secundare de verificare $\mathtt{B} \models \forall X \cdot \pi_i(t) \sim \pi_i(t')$, pentru fiecare $i \in \{1, 2\}$.

Metoda generală de verificare a unei proprietăți comportamentale în COMP constă în descompunerea repetată a obiectelor, potrivit cazului (*b*) pentru compuneri ierarhice de obiecte, până când obținem obiecte de bază, fără componente, pentru care problema testării echivalenței comportamentale poate fi rezolvată ecuațional conform cazului (*a*). Aceasta oferă posibilitatea verificării automate în COMP a proprietăților comportamentale folosind în întregime argumente ecuaționale.

Pentru a demonstra utilizarea metodei, considerăm obiectul compus cu sincronizare

```
bobj ACCOUNT-SYS with states AccountSys is
  syncing (ACCOUNT as A) and (ACCOUNT as B) .
  var AS : AccountSys . var N : Nat .

  act transfer : AccountSys Nat -> AccountSys .
  ax π_A(transfer(AS, N)) = withdraw(π_A(AS), N) .
  ax π_B(transfer(AS, N)) = deposit(π_B(AS), N) if N <= balance_A(AS) .
  ax π_B(transfer(AS, N)) = π_B(AS) if N > balance_A(AS) .
endbo
```

corespunzător unui sistem cu două tipuri de conturi bancare, `A` și `B`, și cu o operație `transfer` care modelează îndeplinirea unei plăți dintr-un cont de tipul `A` într-un cont de tipul `B`. În acest caz, putem verifica, de exemplu, că realizarea unui transfer a N unități bancare imediat după depunerea lor în contul `A` este echivalentă cu depunerea a N unități bancare direct în contul `B`:

$$\texttt{ACCOUNT-SYS} \models \forall\{AS, N\} \cdot \texttt{transfer}(\texttt{deposit}_{\texttt{A}}(AS, N), N) \sim \texttt{deposit}_{\texttt{B}}(AS, N).$$

Potrivit metodei de verificare descrise mai sus, pentru că `ACCOUNT-SYS` are două componente, `A` și `B`, este suficient să verificăm următoarele proprietăți:

- $\pi_{\texttt{A}}(\texttt{transfer}(\texttt{deposit}_{\texttt{A}}(AS, N), N)) \sim \pi_{\texttt{A}}(\texttt{deposit}_{\texttt{B}}(AS, N))$
- $\pi_{\texttt{B}}(\texttt{transfer}(\texttt{deposit}_{\texttt{A}}(AS, N), N)) \sim \pi_{\texttt{B}}(\texttt{deposit}_{\texttt{B}}(AS, N))$

Acestea sunt reduse, în urma unor simplificări succesive conform axiomatizării operației `transfer` și a acțiunilor $\texttt{deposit}_{\texttt{A}}$ și $\texttt{deposit}_{\texttt{B}}$, la ecuațiile comportamentale:

- $\texttt{withdraw}(\texttt{deposit}(\pi_{\texttt{A}}(AS), N), N) \sim \pi_{\texttt{A}}(AS)$
- $\texttt{deposit}(\pi_{\texttt{B}}(AS), N) \sim \texttt{deposit}(\pi_{\texttt{B}}(AS), N)$

Cea de-a doua este trivială mulțumită reflexivității relației de echivalență comportamentală. Pentru prima ecuație, deoarece termenii $\texttt{withdraw}(\texttt{deposit}(\pi_{\texttt{A}}(AS), N), N)$ și $\pi_{\texttt{A}}(AS)$ sunt de sort `Account` – fără componente – este suficient să verificăm ecuația:

$$\texttt{balance}(\texttt{withdraw}(\texttt{deposit}(\pi_{\texttt{A}}(AS), N), N)) = \texttt{balance}(\pi_{\texttt{A}}(AS))$$

care poate fi probată cu ușurință deoarece, potrivit axiomatizării în modulul `ACCOUNT` a efectului acțiunilor `deposit` și `withdraw` asupra observației `balance`, avem:

$$\texttt{balance}(\texttt{withdraw}(\texttt{deposit}(\pi_{\texttt{A}}(AS), N), N)) = \texttt{balance}(\pi_{\texttt{A}}(AS)) + N - N.$$

Act. 2.3 – Parser

Sistemul COMP este implementat în Maude și distribuit în cadrul platformei SpeX, un framework general de lucru cu specificații formale (implementat de asemenea în Maude) care integrează mai multe limbaje de specificare, asemenea COMP, ce au la bază, de exemplu, logici propoziționale, ecuaționale, sau modale.

Principalul avantaj al platformei SpeX raportat la dezvoltarea de sisteme noi precum COMP este caracterul său modular, care permite o integrare rapidă, fără intervenții majore asupra codului sursă, a parserelor și procesoarelor de limbaje de specificare formală implementate în Maude. Mai mult, SpeX include biblioteci și funcționalități dedicate pentru analiza sintactică și pentru tipărirea specificațiilor formale. Acestea includ, printre altele, tehnici de parsare combinatorială – similare celor disponibile în limbaje funcționale precum Haskell – care facilitează construcția unor instrumente de parsare complexe prin combinarea, conform unor reguli gramaticale date, precum cele din secțiunea Act. 1.2, a altor parsere mai simple.

Pentru a ilustra tehnica de lucru, remarcăm mai întâi că limbajul COMP este în esență eterogen. Acesta cuprinde patru sublimbaje grupate după cum urmează:

- două sublimbaje de specificare de bază, MSA și HA, corespunzătoare declarațiilor interne modulelor de date (MSA) și obiectelor comportamentale (HA);
- două sublimbaje de structurare a specificațiilor; primul, abreviat DL, este bazat pe MSA și permite structurarea modulelor de date prin importuri; iar cel de-al doilea, abreviat OL, este bazat pe HA și permite structurarea obiectelor comportamentale prin compuneri ierarhice.

Apoi, pentru fiecare limbaj și sublimbaj integrat în SpeX, definim – în Maude – un *identificator de limbaj*, adică un sort ce poartă numele limbajului împreună cu o constantă corespunzătoare acelui sort. De exemplu, pentru COMP definim (fragmentele de cod din această secțiune și din secțiunea Act. 2.4 sunt scrise în Maude; sintaxa este similară celei folosite în modulele de date ale limbajului COMP):

```
sort COMP .
op COMP : -> COMP .
```

Parserul limbajului COMP este obținut printr-o combinare (dată de operatorul `or-else`, predefinit în SpeX) a parserelor corespunzătoare sublimbajelor de structurare de specificații pentru date (DL) și pentru obiecte comportamentale (OL):

```
op parse : COMP QidList DB{COMP} -> ParsingOutcome .
eq parse(COMP, QIL, DB)
 = parse(COMP/DL, QIL, DB) or-else parse(COMP/OL, QIL, DB) .
```

În fragmentul de cod de mai sus, `COMP`, `COMP/DL` și `COMP/OL` sunt identificatori ai limbajelor COMP, DL, respectiv OL; variabila `QIL:QidList` corespunde textului de parsat (care, potrivit terminologiei Maude, este o listă de *quoted identifiers*, sau *qids*); iar variabila `DB:DB{COMP}` corespunde unei baze de date a interpretorului COMP, adică unei colecții de module parsate anterior textului curent `QIL`.

Rezultatul parsării (de sort `ParsingOutcome`) are una dintre următoarele forme:

`successful-parsing(T)`, ceea ce indică o parsare cu succes a textului `QIL` în contextul `DB`; în acest caz, `T` este un arbore de parsare corespunzător textului.

`incomplete input`, ceea ce indică necesitatea completării textului `QIL` cu noi intrări; de exemplu, un text precum „`bobj ACCOUNT is`" este incomplet deoarece lipsește cuvântul cheie final `endbo`.

`parsing-error(EM, WA)`, ceea ce indică o eroare de parsare; în acest caz, `EM` corespunde descrierii erorii (ce urmează a fi prezentată utilizatorului), iar `WA` este o listă de argumente folosite, de exemplu, pentru a identifica poziția erorii în `QIL`.

Fiecare limbaj integrat în SpeX necesită de asemenea definirea unei gramatici corespunzătoare. Aceasta nu este transcrisă direct în notație BNF, ci codificată ca modul funcțional al limbajului Maude folosind o corespondență bine cunoscută între gramatici independente de context și signaturi algebrice ordonat-sortate. De exemplu, pentru sublimbajul OL considerăm declarații de operații de forma:

```
op bobj_is`endbo : Header -> BObjModule .
op bobj_with`states_is`endbo : Header Sort -> BObjModule .
op bobj_is_endbo : Header DeclList -> BObjModule .
op bobj_with`states_is_endbo : Header Sort DeclList -> BObjModule .
```

Fiecare dintre ele codifică o regulă gramaticală (cu/fără stări explicite, respectiv cu/fără declarații HA adiționale) folosită pentru a specifica un obiect comportamental fără componente. Pentru obiectele compuse considerăm alte patru declarații similare de operații care conțin sortul `Composition` în aritate. Cea mai complexă dintre acestea, corespunzătoare obiectelor compuse cu stări explicite și declarații HA adiționale, este:

```
op bobj_with`states_is__endbo : Header Sort
                                Composition DeclList -> BObjModule .
```

Analog, regulile gramaticale corespunzătoare compunerilor sunt codificate astfel:

```
op composing_. : BObjRefList -> Composition .
op syncing_. : BObjRefList -> Composition .
op syncing_. : BObjName -> Composition .
op indexing_on_by_. : BObjName ModuleName Sort -> Composition .
```

Distingem prin această codificare cele două forme ale operatorului de sincronizare: prima este folosită pentru referințe de obiecte ce urmează a fi compuse înaintea sincronizării, iar cea de-a doua pentru sincronizarea unui obiect deja compus.

Act. 2.4 – Analiză statică

Parsarea deplină a unui text `QIL` – potrivit notațiilor de mai sus – se realizează în doi pași: mai întâi, o parsare superficială, numită *scanare*, care verifică dacă textul este scris corect conform gramaticii limbajului COMP; apoi, în cazul în care scanarea se încheie cu succes, ceea ce corespunde unui rezultat de forma `successful-parsing(T)`, o *analiză statică* (de obicei, prin inducție structurală) a arborelui de parsare `T`.

Cei doi pași sunt evidențiați în următorul fragment de cod corespunzător definiției funcției de parsare a sublimbajului OL pentru module de forma `bobj … endbo`.

```
op parse : COMP/OL QidList DB{COMP} -> ParsingOutcome .
eq parse(COMP/L, QIL, DB)
 = scan COMP/OL input (QIL : 'BObjModule)
   then analyse COMP/OL module in AState[COMP/OL]{DB} .
```

La linia 3 se încearcă derivarea unui arbore sintactic `T` de sort `BObjModule` pentru textul de intrare `QIL`. Dacă este posibilă și unic definită (i.e., lipsită de ambiguități gramaticale), atunci parsarea continuă cu o analiză statică a arborelui sintactic `T` potrivit unor reguli definite pentru modulele corespunzătoare obiectelor comportamentale. În caz contrar, rezultatul parsării coincide cu cel al scanării inițiale, adică este fie `incomplete input`, fie de forma `parsing-error(EM, WA)`.

Analiza statică are în vedere teste contextuale menite să asigure, de exemplu, buna declarare a referințelor de module de date sau de obiecte comportamentale, a sorturilor și a simbolurilor de operații, etc. Fiecare dintre cele patru sublimbaje integrate în COMP prezintă propriile reguli de analiză statică. De exemplu:

- Pentru DL, fiecare import trebuie să facă referire la un modul de date deja declarat; cu alte cuvinte, declarațiile de module au vizibilitate liniară.
- Pentru MSA, aritatea și sortul fiecărei operații trebuie să fie formate doar din sorturi deja declarate, deci vizibilitatea declarațiilor de sorturi (dar și de operații, variabile, propoziții) este liniară de asemenea. Mai mult, numărul de

caractere *underscore* (atunci când sunt folosite) în numele unei operații trebuie să corespundă lungimii arității operației, iar atributele ecuaționale `assoc`, `comm` și `id:` pot fi specificate doar pentru operații binare.

- Pentru OL, fiecare compunere trebuie să facă referire la obiecte comportamentale deja declarate. În plus, numele componentelor oricărui obiect compus trebuie să fie distincte; iar atunci când sincronizarea este folosită în cea de-a doua formă, analiza statică testează că obiectul sincronizat este într-adevăr compus.

- Pentru HA trebuie verificate în general constrângerile specifice signaturilor cu sorturi ascunse. De exemplu, nu pot exista suprapuneri între declarațiile de sorturi vizibile și declarațiile de sorturi ascunse, iar operațiile comportamentale trebuie să conțină cel puțin un sort ascuns în aritate (respectiv exact un sort ascuns în cazul acțiunilor și observațiilor).

Reguli precum cele de mai sus sunt implementate în Maude prin intermediul unor *scenarii de analiză statică*. La rândul lor, acestea constau din compuneri de *acțiuni* (`AnalysisAction`) prin intermediul unor operatori predefiniți în SpeX (analogi operatorului `or-else` folosit pentru compunerea de parsere). Unul dintre acești operatori este `try`, care apare în următoarea definiție a analizei statice pentru obiecte comportamentale compuse cu stări definite explicit și declarații HA adiționale.

```
op analyse_module : COMP/OL -> AnalysisAction{COMP/OL} .
eq analyse COMP/OL module { 'bobj_with`states_is__endbo [
      H, S, C, DL
    ] : 'BObjModule }
 = try { 'bobj_with`states_is__endbo [
      solve COMP/OL header (H),
      solve COMP/OL sort (S),
      analyse COMP/OL composition (C),
      analyse COMP/OL declarations (DL)
    ] : 'BObjModule } .
```

Rolul operatorului `try` este de a menține structura arborelui sintactic primit drept argument și în același timp de a executa alte acțiuni de analiză statică (e.g., `solve header` sau `analyse composition`) pentru subarborii săi direcți.

Implementarea analizei statice pentru limbajul COMP presupune definirea a peste 120 de scenarii diferite de analiză. Detaliile tuturor acestor scenarii se găsesc în fișierele `DL.maude`, `MSA.maude`, `OL.maude`, și `HA.maude` corespunzătoare codificărilor în Maude a celor patru sublimbaje integrate în COMP.

Act. 2.5 – Generare de cod

Procesarea oricărui modul **bobj** B [**with states** S] **is** ... **endbo** are drept efect imediat compilarea unui obiect comportamental $(SP_{\mathtt{B}}, h_{\mathtt{B}})$, unde – potrivit definiției limbajului COMP – $SP_{\mathtt{B}} = (\Sigma_{\mathtt{B}}, E_{\mathtt{B}})$ este o specificație HA iar $h_{\mathtt{B}}$ este un sort ascuns al signaturii $\Sigma_{\mathtt{B}}$ corespunzător stărilor obiectului B. Atunci când este folosită sintaxa **with states** S, sortul ascuns $h_{\mathtt{B}}$ este S; altfel, numele său implicit este State.

De exemplu, pentru următoarea definiție trunchiată a modului ACCOUNT (scris de această dată fără stări explicite și fără axiome corespunzătoare celor două acțiuni):

```
bobj ACCOUNT is
  protecting NAT .
  act deposit : State Nat -> State .
  act withdraw : State Nat -> State .
  obs balance : State -> Nat .
endbo
```

sistemul COMP generează următoarea listă de declarații HA:

```
states sort State .
data sort Nat .
op 0 : () -> Nat .
op s_ : Nat -> Nat .
act deposit : State Nat -> State .
act withdraw : State Nat -> State .
obs balance : State -> Nat .
```

Compunerile de obiecte extind specificația $SP_{\mathtt{B}}$ cu declarații de sorturi, operații (e.g., acțiuni, observații), și axiome corespunzătoare obiectelor componente. Să considerăm, de exemplu, o simplă compunere paralelă a două obiecte ACCOUNT:

```
bobj ACCOUNT-SYS is
  composing (ACCOUNT as A) and (ACCOUNT as B) .
endbo
```

Componentele obiectului ACCOUNT-SYS sunt redenumite A, respectiv B, deoarece – conform regulilor de analiză statică pentru obiecte compuse – numele date componentelor unui obiect trebuie să fie distincte. Rolul acestei constrângeri este de a facilita accesarea stărilor obiectelor componente și de a evita eventualele ambiguități care ar putea să apară atunci când componente diferite ale unui obiect declară sorturi ale stărilor cu același nume (e.g., atunci când sorturile stărilor sunt implicite).

Interpretorul limbajului COMP distinge stările componentelor prin prefixarea sortului lor cu numele dat componentei căreia îi aparțin. De exemplu, pentru obiectul `ACCOUNT-SYS` listat mai sus, COMP generează următoarele sorturi ascunse:

```
states sort State, A/State, B/State .
```

Primul corespunde stărilor obiectului compus `ACCOUNT-SYS`, cel de-al doilea corespunde stărilor componentei `A`, iar ultimul corespunde stărilor componentei `B`.

Numele date sorturilor componentelor `A` și `B` sunt folosite de asemenea pentru cele două proiecții introduse de compunerea paralelă pentru obiectul compus `ACCOUNT-SYS`. Notația matematică a proiecției $\pi_{\mathtt{A}} \colon h_{\mathtt{ACCOUNT\text{-}SYS}} \to h_{\mathtt{A}}$ este transcrisă în COMP ca `A/State : State -> A/State`. Similar, pentru proiecția $\pi_{\mathtt{B}}$ este folosită notația `B/State`. Obținem astfel următoarele două declarații HA:

```
prj A/State : State -> A/State .
prj B/State : State -> B/State .
```

Convenția prefixării sorturilor stărilor componente cu numele componentelor este folosită și în cazul redeclarării, la nivelul obiectului compus, a acțiunilor și observațiilor corespunzătoare obiectelor componente. Mai precis, pentru fiecare acțiune σ a unei componente `Bi`, `B`-acțiunea σ_i este transcrisă în COMP, folosind caractere ASCII, ca `Bi/`σ. Analog, pentru fiecare `Bi`-observație ζ obținem codificarea `Bi/`ζ.

În concluzie, pentru obiectul compus `ACCOUNT-SYS`, interpretorul COMP generează automat următoarele declarații de acțiuni și observații:

Acțiuni și observații corespunzătoare componentei `A`:

```
act A/deposit : State Nat -> State .
act A/withdraw : State Nat -> State .
obs A/balance : State -> Nat .
```

Acțiuni și observații corespunzătoare componentei `B`:

```
act B/deposit : State Nat -> State .
act B/withdraw : State Nat -> State .
obs B/balance : State -> Nat .
```

Relația dintre aceste operații și acțiunile și observațiile componentelor `A` și `B` este definită axiomatic, conform ecuațiilor descrise în secțiunea Act. 2.1. Astfel, COMP generează de asemenea automat următoarele declarații de propoziții:

Axiome corespunzătoare acțiunii `A/deposit`:

```
ax A/State(A/deposit(AS, N)) = deposit(A/State(AS), N) .
ax B/State(A/deposit(AS, N)) = B/State(AS) .
```

Axiome corespunzătoare acțiunii `A/withdraw`:

```
ax A/State(A/withdraw(AS, N)) = withdraw(A/State(AS), N) .
ax B/State(A/withdraw(AS, N)) = B/State(AS) .
```

Axiome corespunzătoare acțiunii `B/deposit`:

```
ax A/State(B/deposit(AS, N)) = A/State(AS) .
ax B/State(B/deposit(AS, N)) = deposit(B/State(AS), N) .
```

Axiome corespunzătoare acțiunii `B/withdraw`:

```
ax A/State(B/withdraw(AS, N)) = A/State(AS) .
ax B/State(B/withdraw(AS, N)) = withdraw(B/State(AS), N) .
```

Axiome corespunzătoare observației `A/balance`:

```
ax A/balance(AS) = balance(A/State(AS)) .
```

Axiome corespunzătoare observației `B/balance`:

```
ax B/balance(AS) = balance(B/State(AS)) .
```

Act. 2.6 – Definiția limbajului (documentație)

Ultima activitate în cadrul acestei etape de implementare a proiectului a fost dedicată dezvoltării documentației care introduce limbajul de specificare COMP. Aceasta este disponibilă în format PDF pe pagina web a proiectului, la adresa: `http://www.imar.ro/~diacon/COMPproject/doc/COMP-LangDef.pdf`.

Documentul tehnic cuprinde o descriere a fundamentelor matematice ale limbajului COMP, precum și prezentări detaliate ale sintaxei și ale semanticii limbajului, și o expunere amplă a metodologiilor de specificare. O parte dintre acestea au fost incluse, într-o formă mult simplificată, și în prezentul raport, în secțiunile Act. 1.1–2.1.

Bibliografie

[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.

[GD92] Joseph A. Goguen and Razvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig, editor, *Recent Trends in Data Type Specification, 9th Workshop on Specification of Abstract Data Types, Joint with the 4th COMPASS Workshop, Selected Papers*, volume 785 of *Lecture Notes in Computer Science*, pages 1–29. Springer, 1992.

[ST12] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012.

Director Proiect,

Răzvan Diaconescu