luni, 27 martie 2023

Verificare formală utilizând limbajul Event-B și platforma Rodin

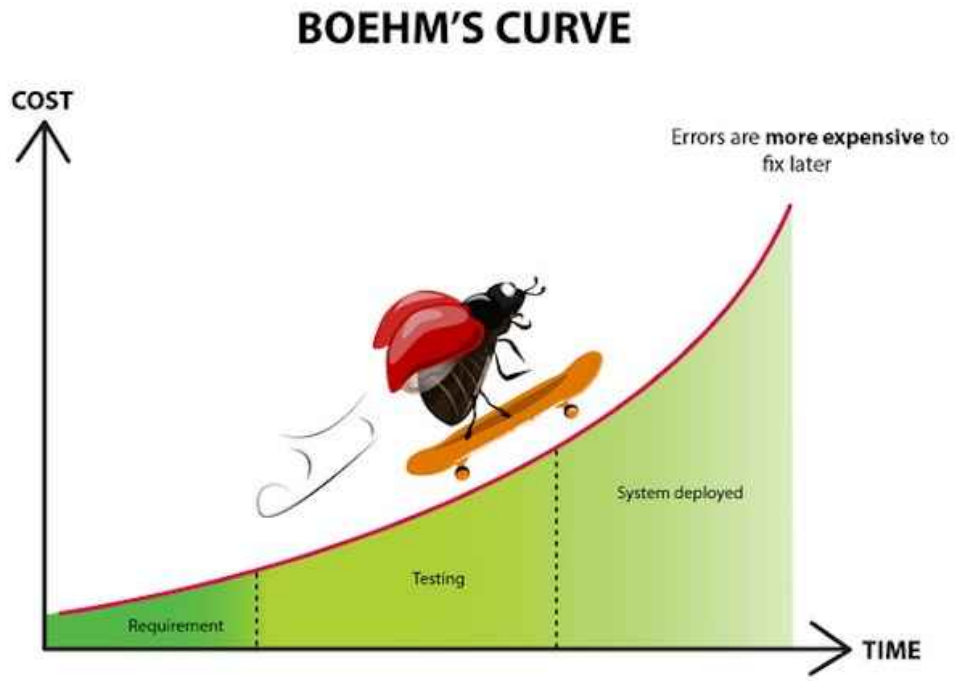
Sorina-Nicoleta Preduț
sorina.predut@unibuc.ro

Cuprins

1. Specificație formală

- O **specificație formală** este o descriere exprimată într-un limbaj formal care surprinde diferitele proprietăți pe care un sistem ar trebui să le satisfacă.
  - Prin crearea unei specificații formale, este posibil să se demonstreze că sistemul se conformează acestor proprietăți și, prin urmare, ar putea fi de ajutor atunci când se stabilește proiectarea unui software de încredere.
- Subiectul **DevOps** (software DEVelopment and IT OPerationS) se învârte în jurul diferitelor practici care ajută la facilitarea proceselor dintre dezvoltarea software și operații.
  - Aceste procese pot permite ca software-ul să fie lansat mai rapid, dar apariția multor erori poate împiedica în mod semnificativ acest proces.
  - În timpul fazei de modelare, aceste erori potențiale pot fi prevenite prin proiectarea sistemului conform unei specificații formale dovedite matematic.
  - Detectarea devreme a acestor erori poate face o diferență uriașă, deoarece **pot fi prevenite** potențialele **costuri viitoare legate de depanare**.

    Acest lucru este subliniat și de **Prima Lege a lui Boehm**:
    - Erorile sunt mai frecvente în timpul cerințelor și activităților de proiectare și sunt mai scumpe cu cât sunt eliminate mai târziu.



2. Event-B

- Limbajul utilizat este formal, deoarece se bazează pe o semantică definită riguros d.p.d.v. matematic în loc de limbajul natural.
- Event-B ne oferă un astfel de limbaj de modelare bazat pe teoria mulțimilor.
- Limbajul a fost introdus în anii 2000 de către Jean-Raymond Abrial și folosit pentru dezvoltarea de modele matematice ale sistemelor complexe care se comportă într-un mod discret. Event-B este o evoluție a limbajului B.
- Limbajul B a fost dezvoltat inițial ca succesor al lui Z de Jean-Raymond Abrial în anii 1990, concentrându-se pe două concepte cheie: utilizarea rafinării pentru a dezvolta treptat modele și instrumente pentru verificare și model checking.
- Există 3 clase de aplicații industriale ale lui B:
  - **B pentru software (B clasic)**: rafinăm specificațiile până când B0, o submulțime low-level a lui B, este atinsă și aplicăm generatoare de cod.
  - **B pentru modelarea sistemului (Event-B)**: verificăm proprietățile critice, înțelegem de ce un sistem este corect.
  - **B pentru validarea datelor**: exprimăm proprietățile în B și verificăm datele (eventual folosind un lanț dublu).
- Event-B este o metodă formală de dezvoltare a sistemelor, utilizată pentru modelarea sistemelor discrete.
- Una dintre caracteristicile principale este **rafinarea,** anume construcția treptată a modelelor cu ajutorul implementării graduale.
- Un model Event-B este compus din 2 componente: **contexte** și **mașini**.
  - Contextele conțin **carrier sets, constante și axiome**.
  - Mașinile conțin **variabile, invarianți și evenimente**.
- Contextul reprezintă partea statica a modelului, iar mașina reprezintă partea dinamică.
- O mașină în Event-B corespunde unui sistem de tranziții, unde variabilele reprezintă stările, iar evenimentele specifică tranzițiile.

3. Rodin

- Platforma Rodin face posibilă modelarea formală folosind Event-B.
- Rodin este un **IDE bazat pe Eclipse** ce poate fi extins folosind diferite plug-in-uri și include funcționalități pentru:

  - modelare,
  - verificarea consistenței modelelor,
  - validarea modelelor.
- Rodin este un acronim pentru **„Rigorous Open Development Environment for Complex Systems**”. Rodin este, de asemenea, **numele unui cunoscut sculptor francez**.
- Platforma este disponibilă în prezent pentru Windows, MacOS și Linux. Pentru a instala Rodin, puteți vizita pagina wiki Event-B (a se vedea referința 2), unde este listată fiecare versiune de Rodin.
- Cea mai recentă versiune disponibilă este 3.7. Wiki trimite către pagina de descărcare SourceForge, indicând cerințele necesare de compatibilitate, cea mai notabilă fiind necesitatea de a avea Java JRE (versiunea 8 sau mai recentă), deoarece aplicația se bazează pe framework-ul Eclipse. Când vizitați pagina de descărcare, asigurați-vă că descărcați cea mai recentă versiune.
- Despachetați arhiva descărcată și rulați executabilul Rodin. Pe MacOS, este necesar să acordați programului permisiunea de a rula, deoarece acesta provine de la un dezvoltator neidentificat (“the Rodin application is not notarized”).
- Înainte de a începe să lucrăm cu Rodin, trebuie să învățăm o teorie care este fundamentală pentru Event-B și care ne va ajuta să creăm și să înțelegem o specificație a modelului Event-B.

4. Logica predicatelor

- Lista următoare cuprinde majoritatea operatorilor necesari sub formă de **{Notație matematică} — {Nume} — {Traducere tastatură Event-B}**:
  - $\neg P$ — Negație — not
  - $P \wedge Q$ — Conjuncție — &
  - $P \vee Q$ — Disjuncție — or
  - $P \Rightarrow Q$ — Implicație — =>
  - $\forall x \mid P$ — Cuantificator universal — \forall
  - $\exists x \mid P$ — Cuantificator existențial — \exists
- Un exemplu de predicat P este $x<10$, $x \in \mathbb{Z}$ sau $x \subseteq S$.

5. Mulțimi

- Mulțimile Event-B predefinite sunt foarte utile, o completare la propriile mulțimi personalizate. Câteva exemple:
  - $\mathbb{N}$ — Mulțimea numerelor naturale — NAT
  - $\mathbb{Z}$ — Mulțimea numerelor întregi — INT
  - BOOL: {TRUE, FALSE} — Tipul de date boolean — BOOL
  - $\varnothing$ — Mulțimea vidă — {}
- Vom enumera câțiva dintre operatorii pe mulțimi:
  - $\in$ — Apartenența — :
  - $\cup$ — Reuniunea — $\vee$
  - $\cap$ — Intersecția — $\wedge$
  - $\setminus$ — Diferența — \
  - $\mathbb{P}$ — Mulțimea putere — POW
  - $\subseteq$ — Incluziunea — <:
- O mulțime este definită prin declararea proprietăților unice ale elementelor sale, apoi este generată folosind criteriile elementelor sale.
  - Să considerăm $\{x|x\in S \wedge P(x)\}$

    Aici x este variabilă și S este mulțimea căreia îi aparține.

    Se pot folosi mulțimile predefinite menționate anterior, cum ar fi $\mathbb{N}$ sau $\mathbb{Z}$.

    P(x) este condiția care depinde de variabila noastră x.

    Un exemplu ar putea fi

    $\{x|x\in\mathbb{Z} \wedge (x > -20 \wedge x < 20)\}$ // Toate numerele întregi între -20 și 20.
- **Relația dintre mulțimi** este o structură matematică utilă în exprimarea specificațiilor formale. O relație este o mulțime de perechi ordonate $s\mapsto t$ unde $s\in S$ și $t\in T$. Notația specială pentru aceasta în Event-B este:
  - $S\longleftrightarrow T=\mathbb{P}(S\times T)$ // × = produs cartezian
  - Un exemplu ar putea fi:

    Persons = {Adam, Bianca, Carl, Dennis, Evelyn}

    Restaurants = {BurgerKing, RedLobster, Max, ChopChop, McDonalds}

    favoriteRestaurants $\in$ Persons$\longleftrightarrow$Restaurants

    favoriteRestaurants = {Adam$\mapsto$BurgerKing, Evelyn$\mapsto$Max, Carl$\mapsto$ChopChop, Carl$\mapsto$McDonalds} // Carl are mai multe restaurante favorite

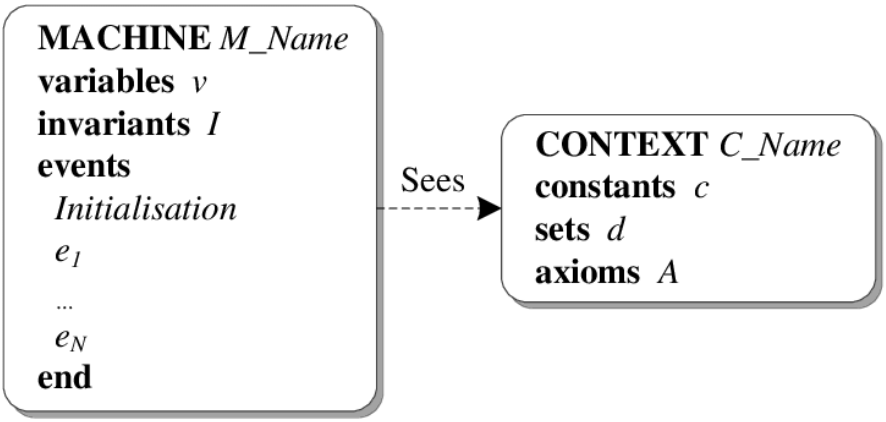
- Operatorii pentru specificarea relațiilor dintre mulțimi sunt:
  - ↦ — Pereche — |->
  - × — produs cartezian — **
  - ⟷ — Relații — <->
  - Relații totale — <<->
  - Relații surjective — <->>
  - Relații surjective totale — <<->>
- Pentru mai mulți operatori matematici și sintaxa acestora în Event-B, consultați Ghidul utilizatorului tastaturii Rodin (a se vedea referința 3).
- **Domeniul unei relații R** este mulțimea formată din primul element al tuturor perechilor din R și se notează **dom(R)** în Event-B. În exemplul nostru anterior:
  - dom(favoriteRestaurants) = {Adam, Evelyn, Carl}
- **Codomeniul** (Range) este similar cu domeniul și returnează mulțimea formată din al doilea element al tuturor perechilor din R și se notează **ran(R)** în Event-B.
  - ran(favoriteRestaurants) = {BurgerKing, Max, ChopChop, McDonalds}
- Restricția și Scăderea Domeniului/Codomeniului sunt 4 operatori foarte utili atunci când lucrăm cu relații. Cu ei putem manipula mulțimea relațiilor și putem elimina părți din domeniu/codomeniu sau îl putem restricționa.
- ◁— Restricția domeniului — <|
  - **Restricția domeniului S ◁ R** este o submulțime care conține toate perechile din R, unde primul element al fiecărei perechi este în S. În exemplul nostru anterior:
  - {Evelyn, Carl} ◁ favoriteRestaurants = {Evelyn↦Max, Carl↦ChopChop, Carl↦McDonalds}
- ⩤— Scăderea domeniului — <<|
  - **Scăderea domeniului S ⩤ R** este o submulțime care conține toate perechile din R, unde primul element al fiecărei perechi nu este în S.
  - {Evelyn, Carl} ⩤ favoriteRestaurants = {Adam↦BurgerKing}
- ▷— Restricția codomeniului— |>
  - **Restricția codomeniului R ▷ S** este o submulțime care conține toate perechile din R, unde al doilea element al fiecărei perechi este în S.

- favoriteRestaurants ▷ {Max, ChopChop} = {Evelyn↦Max ,Carl↦ChopChop}

- ⩥ — Scăderea codomeniului — |>>
  - **Scăderea codomeniului R ⩥ S** este o submulțime care conține toate perechile din R, unde al doilea element al fiecărei perechi nu este în S.
  - favoriteRestaurants ⩥ {Max, ChopChop} = {Adam↦BurgerKing, Carl↦McDonalds}
- **Imaginea relațională R[S]** este mulțimea care conține al doilea element al tuturor perechilor din R pentru care primul element este în S.
  - favoriteRestaurants[{Adam, Carl}] = {BurgerKing, ChopChop, McDonalds}

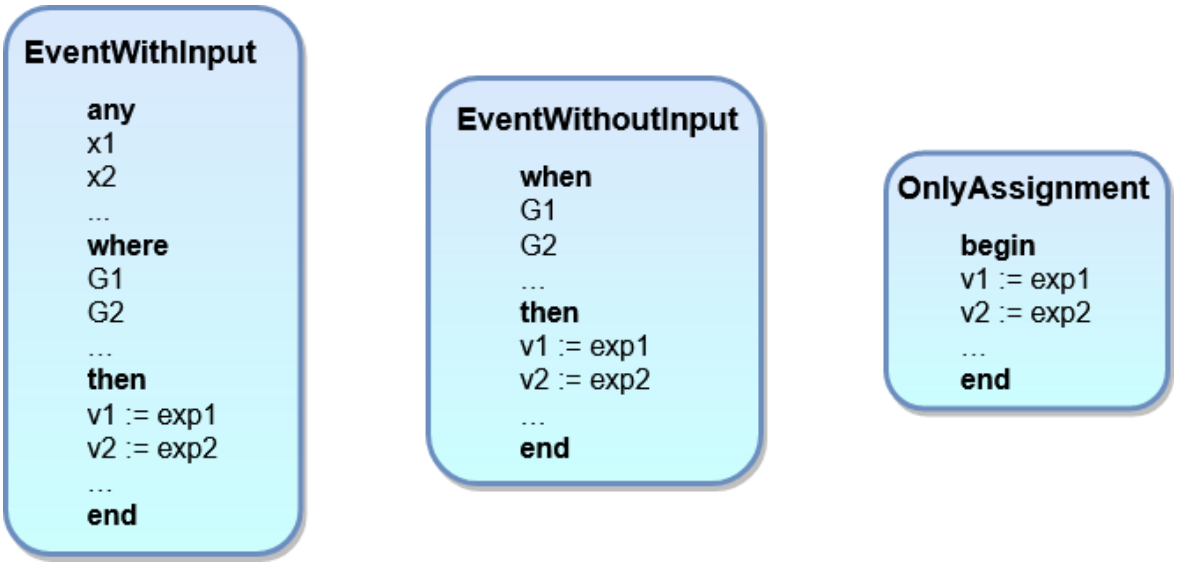
6. Componente

- Event-B constă din două tipuri de componente: Mașini și Contexte.
- Mașinile „văd" contexte în sensul că pot folosi informațiile conținute în acestea.
- Variabilele și evenimentele fac mașinile dinamice, în timp ce informațiile din contexte sunt statice.
- Reprezentarea generală a componentelor Event-B:



- Un **context** este format din:
  - **sets** (sau carrier sets): mulțimi definite de utilizator (personalizate) pot fi declarate în această secțiune.
  - **constants**: Aici sunt declarate constante, fiecare tip trebuie declarat în axioms.
  - **axioms**: Listă de predicate care definesc reguli pentru elementele contextului și nu ar trebui să fie încălcate niciodată. Acestea pot fi considerate ca fiind evidente și nu mai trebuie demonstrate.
  - **theorems**: Axiomele marcate ca teoreme pot fi demonstrate folosind axioma scrisă chiar înainte/deasupra lor.

- O **mașină** constă din:
  - **sees**: informațiile contextului care vor fi utilizate de mașină.
  - **variables**: Fiecare variabilă necesită valoare de inițializare, poate fi nedeterministă, iar tipul fiecărei variabile trebuie declarat ca invariant.
  - **invariants**: predicate ce ar trebui să fie întotdeauna adevărate pentru toate stările accesibile.
  - **events**: Evenimentele pot primi intrări, pot atribui valori variabilelor și sunt executate numai atunci când sunt îndeplinite condițiile (gărzile). Inițializarea este un tip special de eveniment, este un cuvânt cheie rezervat. Un eveniment poate fi reprezentat sub oricare dintre următoarele trei forme:



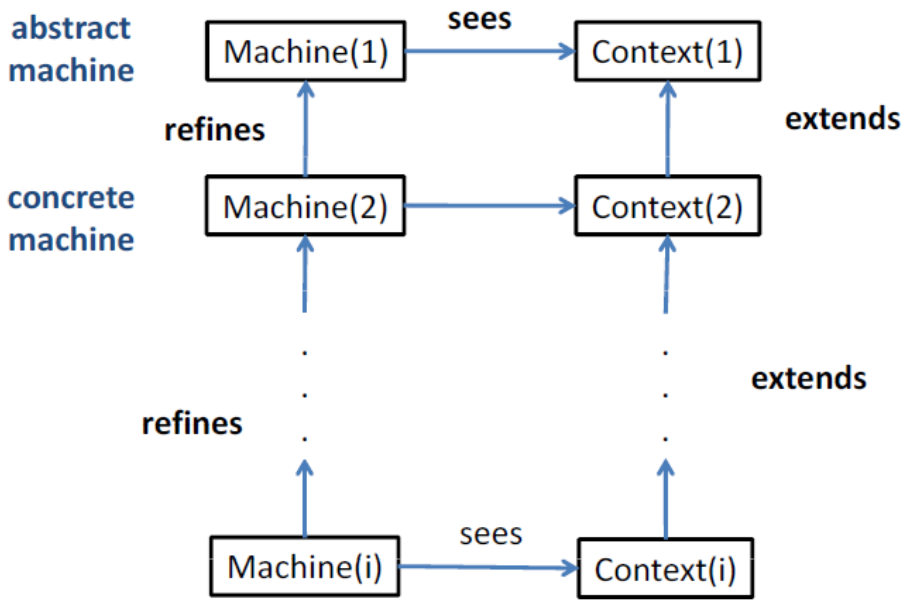
7. Obligații de demonstrat (Proof obligations)

- **Secvențele** sunt descrierea formală a ceva ce vrem să demonstrăm.
  - Sunt formate din ipotezele H și un scop G. Sunt de următoarea formă: **H** $\vdash$ **G**. Aceasta poate fi citită ca „Sub ipotezele H, scopul G este demonstrabil”.
  - Ipotezele și scopurile sunt date atât de utilizator, cât și de sistem. Există, de asemenea, modalități de a ajuta la demonstrarea unui scop, dar acestea sunt puțin mai avansate.
- **Contextele** trebuie să fie consistente și, prin urmare, trebuie să satisfacă următoarele proprietăți:
  - Axiomele trebuie să fie bine definite (axm/WD)
  - Teoremele trebuie să fie bine definite (thm/WD)
  - Teoremele trebuie demonstrate (thm/THM)

- **Mașinile** trebuie să fie, de asemenea, consistente și să aibă proprietăți satisfăcute.
  - Menționate anterior: thm/WD și thm/THM
  - Invarianții trebuie să fie bine definiti (inv/WD)
  - Gărzile și evenimentele trebuie să fie bine definite (grd/WD & act/WD)
  - Evenimentele nedeterministe trebuie să fie fezabile (evt/act/FIS)
  - Invarianți stabiliți prin inițializare (INIT/inv/INV)
  - Invarianți adevărați pentru toate evenimentele (evt/inv/INV)
- Toate numele scrise cu minuscule sunt substituenți pentru numele real al unui eveniment (evt), acțiune (act), axiomă (axm), etc.
- Proof obligations în Rodin:

8. Dezvoltarea modelului

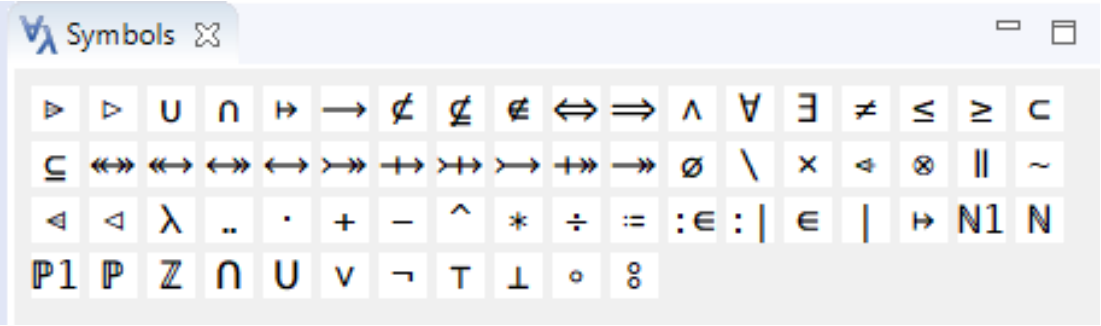
- Pe lângă informațiile deja menționate conținute în mașini și contexte, există Rafinări și Extinderi, care sunt modalități de a construi peste mașini, respectiv, contexte definite anterior. Putem rafina o mașină sau extinde un context utilizând cuvintele cheie menționate, ceea ce ne permite să utilizăm acea componentă ca bază pentru a construi ceva nou și probabil mai bun. Dezvoltarea modelului este o caracteristică mai avansată a Event-B. Vom aprofunda rafinarea și extinderea în exemplul nostru de proiect.
- Relația dintre mașini și contexte:



9. Exemplu: Zoom

- Vom prezenta un exemplu simplu despre cum să începem modelarea unei specificații formale folosind platforma Rodin. Acesta va fi centrat în jurul unei **aplicații imaginare de videoconferință** numită Zoom.
- Zoom este utilizat în principal pentru prelegeri online, ceea ce înseamnă că studenții și lectorii sunt principalii participanți care iau parte la întâlniri. Sistemul nostru se va concentra, în primul rând, pe modul în care este gestionată o întâlnire.
- Atât studenții, cât și lectorii au propriul lor id unic, care este folosit pentru a-i identifica. **Fiecare întâlnire este limitată la cel mult 5 prelegeri și 45 de studenți**. Vom începe cu o versiune mai simplă și apoi vom crea o versiune mai complicată prin rafinarea celei anterioare.
- Începem prin a defini cerințele de bază pentru sistemul nostru:
  - **REQ1**: O întâlnire poate avea cel mult 45 de studenți participanți.
  - **REQ2**: O întâlnire poate avea cel mult 5 lectori participanți.
  - **REQ3**: Ar trebui să fie posibilă adăugarea de studenți la o întâlnire.
  - **REQ4**: Ar trebui să fie posibilă adăugarea de lectori la o întâlnire.
  - **REQ5**: Ar trebui să fie posibilă eliminarea participanților dintr-o întâlnire.
- În timpul procesului de modelare în Rodin, va trebui să folosim diferite notații matematice. Pentru a le scrie în Rodin, consultați secțiunea de teorie unde sunt

specificate traducerile Event-B. Cele mai multe dintre acestea sunt date în fila „Simboluri” din Rodin.



- Așa cum am menționat anterior, proiectele Rodin sunt formate din contexte și mașini. Aceste componente vor fi incluse într-un proiect Event-B. Pentru a crea un proiect, mergem la File / New / Event-B Project. Va apărea un mic asistent unde putem specifica numele proiectului.
- Vom continua prin crearea mai întâi a unui context Event-B. Acest lucru poate fi realizat accesând File / New / Event-B Component. Va apărea un alt asistent unde vom denumi contextul **„Meeting_c0”** și alegem tipul context pentru proiectul Zoom nou creat.
- În centrul spațiului de lucru Rodin, vom vedea o versiune completă a contextului. Anterior, am menționat că un context este alcătuit din mulțimi, constante, axiome și teoreme. Le putem adăuga utilizând asistenții din bara de instrumente. Enumerated Set | Axiom | Carrier Set | Constant:

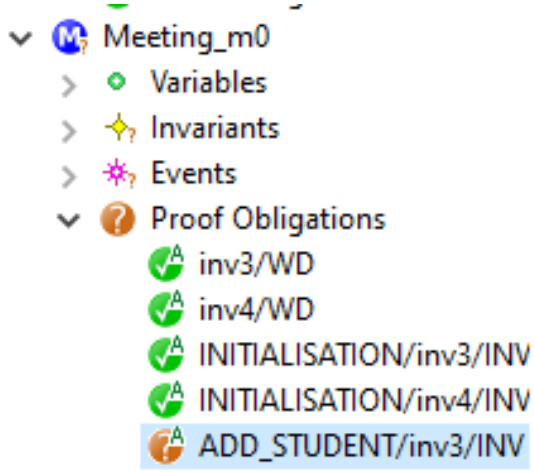
- Vom modifica contextul prin adăugarea unei mulțimi (carrier set) denumită prin identificatorul **„ALLOWED_PARTICIPANTS**”. Convenția de denumire Event-B este de a scrie numele mulțimilor de constante cu majuscule.
- ALLOWED_PARTICIPANTS va conține id-urile tuturor membrilor cărora li s-a permis să se alăture întâlnirii. Ulterior, vom adăuga constante și axiomele lor. Denumim prima constantă **„student_limit”** și setăm axioma să afirme că valoarea este egală cu 45. A doua constantă **„lecturer_limit”** și setăm axioma să afirme că valoarea este egală cu 5.
  Va trebui să adăugăm o axiomă suplimentară care afirmă că mulțimea ALLOWED_PARTICIPANTS este finită, altfel nu putem folosi card() pe nicio

mulțime bazată pe aceasta. Folosind asistentul pentru axiome putem afirma că mulțimea este finită prin următorul predicat:
„finite(ALLOWED_PARTICIPANTS)”.

- Cu ajutorul contextului creat, vom defini limitele numărului de studenți și lectori, precum și o mulțime formată din toți participanții permisi.
- Vom continua să extindem specificația prin crearea unei mașini care va „vedea” contextul. Putem crea o mașină mergând, din nou, la File / New / Event-B Component. Vom denumi mașina **„Meeting_m0”** și alegem tipul machine pentru proiectul Zoom creat. Putem adăuga informații unei mașini cu ajutorul unor asistenți din bara de instrumente.
  Variable | Variant | Invariant | Event:

- **Trebuie să ne asigurăm că mașina „vede” contextul Meeting_m0**. Nu există un asistent pentru această acțiune. În schimb, vom face clic dreapta pe „MACHINE”, alegând opțiunea „[Child]->Sees context relationship”. Pentru a alege ce context să „vedă”, facem clic stânga pe „— undefined —” și alegem Meeting_c0 din listă.
- Vom adăuga în continuare variabile care vor reprezenta proprietățile dinamice ale sistemului nostru: studenți și lectori, care vor fi mulțimi formate din id-uri unice pentru studenții, respectiv, lectori participanți.
- Utilizând asistentul pentru variabile, în cazul studenților, vom seta **identificatorul „students”, îl inițializam cu mulțimea vidă și specificăm tipul variabilei adăugând un invariant care afirmă că studenții reprezintă o submulțime a mulțimii ALLOWED_PARTICIPANTS**.
- Repetăm acest proces pentru a adăuga variabila **lecturers**.
- Înainte de a crea evenimente, trebuie să adăugăm **invarianți suplimentari** care să impună cerințele oferite de descrierea sistemului:
  card(students) ≤ student_limit
  card(lecturers) ≤ lecturer_limit
- Student_limit și lecturer_limit sunt informații pe care le primim din contextul Meeting_c0 pe care mașina Meeting_m0 „le vede”.

- Acum putem începe să adăugăm funcționalități prin crearea de evenimente folosind asistentul de evenimente. Pentru a distinge între diferite evenimente, vom schimba numele etichetelor pentru a fi mai descriptive despre ceea ce face evenimentul.
- Primul **eveniment** se va numi **„ADD_STUDENT”** care va adăuga studenți la întâlnire. Va avea un parametru student_id ce îndeplinește condiția (guard): **student_id** ∈ **ALLOWED_PARTICIPANTS** care specifică tipul parametrului.
  Evenimentul va avea o acțiune care adaugă valoarea parametrului, student_id, la mulțimea studenților folosind următoarea expresie logică:
  students ≔ **students** ∪ **{student_id}**
  După salvare, observăm o eroare în obligațiile de demonstrat (=> proof obligation undischarged):



  care spune că evenimentul ADD_STUDENT nu îndeplinește invariantul inv3 care afirmă că numărul de elemente din studenți trebuie să fie întotdeauna mai mic sau egal cu 45. Acesta este rezultatul faptului că evenimentul ADD_STUDENT poate adăuga mai mulți studenți participanți, chiar dacă suma actuală este mai mare sau egal cu 45. **Pentru a demonstra această obligație, trebuie să adăugăm următoarea condiție în eveniment:**
  card(students) < student_limit
  Condiția împiedică executarea evenimentului dacă adăugarea unui alt student va duce la depășirea valorii student_limit.
  Pentru a adăuga fără asistent, facem clic dreapta pe punctul alb din stânga numelui evenimentului, ceea ce ne permite să alegem „[Child] -> Add Guard”.

O nouă condiție va apărea în acest caz, înlocuim „T” cu noua specificație. După salvare, vom vedea că obligația de demonstrat devine verde. Dacă nu, facem clic dreapta pe „Proof Obligations” și alegem „Retry Auto Prover”.

- Repetăm procesul pentru a crea un nou **eveniment** numit **ADD_LECTURER**.
- Ultimul eveniment pe care trebuie să-l adăugăm ar trebui să elimine participanții permiși. Vom numi acest **eveniment REMOVE_PARTICIPANT**. Va avea un parametru, participant_id ce îndeplinește condiția:

  participant_id ∈ **ALLOWED_PARTICIPANTS**

  Putem apoi elimina participantul din ambele mulțimi, indiferent dacă au aparținut anterior sau nu mulțimii:

  students ≔ **students** ∖ **{participant_id}**

  lecturers ≔ **lecturers** ∖ **{participant_id}**
- După adăugarea celor 3 evenimente, vom vedea că toate obligațiile de demonstrat trec și prima versiune a specificației este terminată. A se vedea arhiva pentru versiunea completă în Rodin.

Rafinare mașină abstractă - nivel intermediar

- Acum vom rafina versiunea anterioară adăugând mai multe funcționalități. Această funcționalitate se referă la **adăugarea de administratori care pot interzice anumitor persoane să participe la întâlnire**.

  Există, de asemenea, cererea pentru o nouă mulțime care să conțină informații despre toți participanții la întâlnire pentru o căutare mai ușoară.
- Stabilim noile cerințe pe care dorim să le îndeplinească sistemul.
- **REQ6**: Ar trebui să existe o mulțime care să conțină informații despre toți participanții și rolurile acestora.
- **REQ7**: Ar trebui să fie posibil să adăugăm administratori.
- **REQ8**: Nu ar trebui să fie mai mult de 5 administratori într-o întâlnire.
- **REQ9**: Ar trebui să fie posibilă interzicerea participării studenților.
- Vom crea un nou context care extinde contextul anterior Meeting_c0 cu clic dreapta pe Meeting_c0 în Event-B Explorer și alegând opțiunea **Extend**. Denumim noul context **„Meeting_c1”.**
- Cerințele noastre stabilesc că ar trebui să existe o mulțime de roluri posibile diferite. Acest lucru poate fi realizat în Event-B folosind o mulțime de tip

enumerated set, cu elemente predefinite. Vom denumi mulțimea **„ROLES”** și vom adăuga următoarele elemente: **Student, Lector, Administrator și Banned**.

- De asemenea, trebuie să existe o constantă împreună cu o axiomă care specifică numărul maxim de administratori într-o întâlnire, numită „**admin_limit**”.
- După crearea noului context, vom crea o nouă mașină care o rafinează pe cea anterioară, Meeting_m0. Acest lucru se face într-un mod similar cu extinderea unui context prin clic dreapta și alegerea opțiunii **Refine**. Mașina rafinată va fi numită **„Meeting_m1**”. Vom schimba contextul pe care îl vede mașina în cel nou creat numit Meeting_c1. Vom adăuga 3 variabile noi: administrators, banned și participants. Pentru fiecare variabilă vom adauga un invariant:

  administrators ⊆ **ALLOWED_PARTICIPANTS**

  banned ⊆ **ALLOWED_PARTICIPANTS**

  participants ∈ **ALLOWED_PARTICIPANTS↔ROLES**

  card(administrators) ≤ admin_limit

  Sintaxa pentru participanți specifică o relație între ALLOWED_PARTICIPANTS și ROLES.
- Vom adăuga evenimente noi necesare.
- Mai întâi trebuie să putem interzice participarea studenților, așadar vom crea un eveniment numit **„BAN_STUDENT”** care are 2 parametri, **student_id** și **admin_id** ce specifică id-ul studentului și id-ul administratorului care dorește să interzică participarea studentului. Vom folosi condițiile pentru a ne asigura că studentul este un participant la întâlnire și că admin_id-ul dat este un administrator:

  student_id ∈ **students**

  admin_id ∈ **administrators**

  După aceea, trebuie să eliminăm studentul din mulțimea studenților și să-l adăugăm la mulțimea studenților interzișii (banned).

  Apoi, vom adăuga studentul și noul său rol Banned la mulțimea participanților. Putem face acest lucru folosind următoarele acțiuni:

  students ≔ **students** ∖ **{student_id}**

  banned ≔ **banned** ∪ **{participant_id}**

  participants ≔ **participants** ∪ **{participant_id ↦ Banned}**

  Vom vedea că o obligație nu poate fi demonstrată: BAN_STUDENT/students/

EQL. Motivul este o cerință de coerență nemenționată anterior pentru rafinare. **Orice eveniment nou al unei mașini concrete nu poate modifica o variabilă a mașinii abstracte**. Deoarece students este o variabilă a lui Machine_m0, nu avem voie să o modificăm în evenimentul BAN_STUDENT. Acest lucru poate fi rezolvat prin adăugarea unui eveniment numit BAN_STUDENT la Machine_m0, care elimină doar un student din mulțimea studenților prin acțiunea:

students ≔ **students** ∖ **{student_id}**

După aceea, putem rafina evenimentul BAN_STUDENT în Machine_m1. Făcând clic dreapta pe punctul alb din stânga BAN_STUDENT vom putea alege **„[Child]-> Add Refine Event Relationship”**.

În partea dreaptă a numelui evenimentului este textul „nu este extins obișnuit” (not extended ordinary), ceea ce înseamnă că dorim să redefinim totul din eveniment. Dacă dorim să menținem caracteristica de bază a evenimentului sau să construim numai pe aceasta vom modifica textul în „extended ordinary”. Pentru mai multe cerințe de consistență, citiți Manualul utilizatorului Rodin (a se vedea referința 1).

- Vom crea un eveniment care ne permite să adăugăm noi administratori la întâlnire numit **„ADD_ADMIN”,** specificat similar cu ADD_STUDENT și ADD_LECTURER create anterior. Vom adăuga administratorul și rolul său de Administrator la mulțimea participanților.
- De fapt, trebuie să adăugăm roluri pentru toți participanții noi. Vom adăuga o altă acțiune la ADD_STUDENT și ADD_LECTURER care adaugă perechea **{placeholder_id ↦placeholder_role}** la mulțimea participanților.
  Pentru a adăuga o acțiune nouă la un eveniment existent, putem face clic dreapta pe punctul alb din stânga numelui evenimentului și alegem **„[Sibling]-> Add action”**.
- În plus, trebuie să putem elimina un administrator din întâlnire. Acest lucru se face prin adăugarea unei alte acțiuni în evenimentul **REMOVE_PARTICIPANT**, care este foarte asemănătoare cu cele deja existente.
- De asemenea, trebuie să putem elimina un participant din mulțimea participanților. Acest lucru se poate face folosind scăderea domeniului. Vom adăuga încă o acțiune în REMOVE_PARTICIPANT, care atribuie participanților valoarea curentă a domeniului participanților din care eliminăm participant_id.

Aceasta înseamnă că eliminăm toate perechile din participanți unde participant_id este primul element:

participants ≔ **{participant_id}** ⩤ **participants**

- În plus, trebuie să ne asigurăm că nu putem adăuga un student interzis la întâlnire. Acest lucru înseamnă că trebuie să adăugăm o nouă condiție în ADD_STUDENT. Garda poate fi specificată într-unul din 2 moduri, primul fiind foarte asemănător cu ceea ce am făcut anterior:

  student_id ∉ **banned**

  sau putem folosi mulțimea participanților adăugând o nouă condiție în ADD_STUDENT care verifică dacă Banned nu este unul dintre rolurile student_id. Acest lucru se poate face folosind imagine relațională:

  {Banned} ⊈ **participants[{student_id}]**
- A se vedea arhiva pentru versiunea completă în Rodin.

Rafinare mașină concretă - nivel avansat - temă

- După cum probabil se observă, stabilirea noilor participanți face ca toate celelalte mulțimi să fie redundante. Astfel, prima sugestie este să creați o mașină nouă care să aibă o singură mulțime pentru a le guverna pe toate. Aceasta înseamnă că vor fi mai puține evenimente și variabile, ceea ce va face modelul mai ușor de înțeles și mai ușor de întreținut.
- Vă sugerăm să implementați și următoarele cerințe:

  REQ10: Faceți posibilă anularea interdicției unei anumite persoane.

  REQ11: Permiteți numai studenților de la un anumit curs să se alăture întâlnirii.

  REQ12: Faceți posibilă adăugarea de noi cursuri.

  În acest caz este nevoie de mai mult de o mulțime.
- Pentru mai multe exemple, consultați Event-B Wiki, Exemple (a se vedea referința 4).
- Demo.

10. Exemplul 2: Sistem de control al vitezei de croazieră al unei biciclete electrice (e-Bike)

- A se vedea prezentarea EDMA18 și referințele 6, 7, 8.
- Demo.

Bibliografie

1. Rodin User's Handbook v.2.8, https://www3.hhu.de/stups/handbook/rodin/current/pdf/rodin-doc.pdf
   https://www3.hhu.de/stups/handbook/rodin/current/html/
2. Event-B and Rodin Documentation Wiki, https://wiki.event-b.org/index.php/Main_Page
3. Rodin Keyboard User Guide, https://wiki.event-b.org/index.php/Rodin_Keyboard_User_Guide
4. Event-B Examples, https://wiki.event-b.org/index.php/Event-B_Examples
5. Rodin: Modelling with Event-B, https://medium.com/@ruwaid4/rodin-modelling-with-event-b-8fdab6c65003#5581
6. UML-B tutorial, https://www.uml-b.org/gettingStarted.html
7. ProB tutorial, https://prob.hhu.de/w/index.php?title=Tutorial_Rodin_First_Step
8. S. Preduț, F. Ipate, M. Gheorghe, F. Câmpean, Formal Modelling of Cruise Control System Using Event-B and Rodin Platform, Proc. of the IEEE HPCC/SmartCity/DSS 2018, 1541-1546, 2018, http://www.ifsoft.ro/~florentin.ipate/publications/EDMA18.pdf