MODEL CHECKING OF A ROBOTIC MECHANISM

- LINTE ROBERT-OVIDIU 331
- POPESCU PAULLO-ROBERTTO-KARLOSS 331
- VOICULESCU ALINA-VIRGINIA 331

UTILITATEA MODEL CHECKING-ULUI

- Majoritatea device-urilor din jurul nostru vin împreună cu o bucată de cod ce asigură controlul lor.
- Acest cod este controller-ul device-ului. De exemplu, avioanele calculează traiectoria prin valorile curente ale mai multor parametri.
- De asemenea, ajută la rezolvarea unei multitudini de erori de programare.

THERAC-25 RADIATION OVERDOSING (1985-1987)

- Aparat de iradiere pentru tratarea pacienților cu cancer
- Cel puțin 6 cazuri de supradoză între 1985 si 1987 (s-a depășit doza admisă de 100 ori)
- Trei pacienți au murit
- Sursă: eroare de proiectare în software-ul de control (race condition)
- Software-ul era scris în limbaj de asamblare

AT&T TELEPHONE NETWORK OUTAGE (1990)

- Ianuarie 1990: o problemă la rețeaua telefonică din New York a dus la 9 ore fără semnal în mare parte din US
- Costurile s-au ridicat la aproape 100 milioane $
- Sursă: defect software (intercepție greșită a unui break în C)

ARIANE 5 CRASH

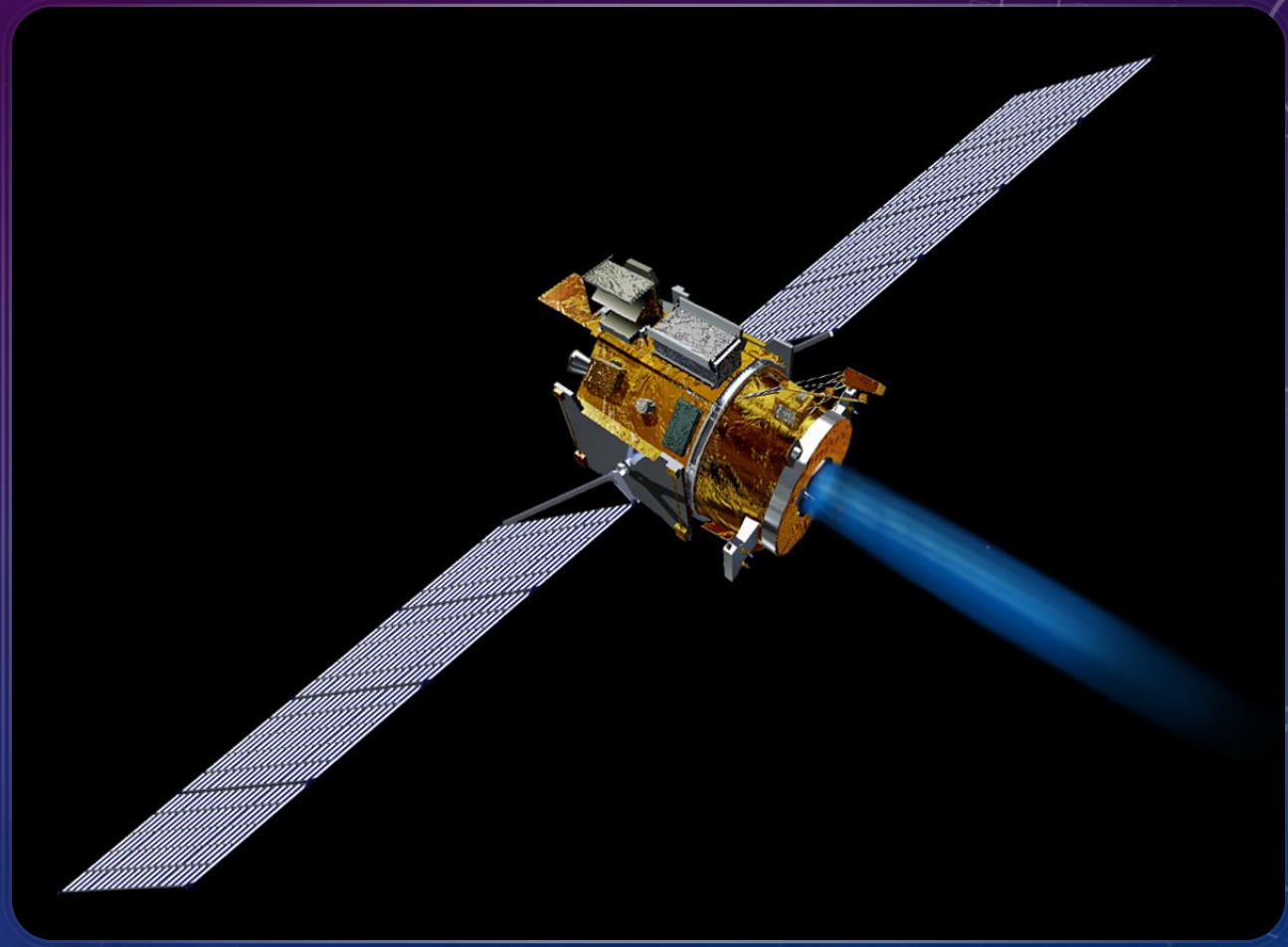
- Prăbușirea rachetei Ariane 5 a avut loc în iunie 1995
- Cost: peste 500 milioane $
- Sursă: defecțiune software (conversie greșită de la un număr pe 64 biți cu virgulă mobilă la un număr întreg pe 16 biți)

PENTIUM FDIV BUG (1994)

- FDIV: Floating Point Fivision Unit
- Anumite operații de divizare în virgule mobile au produs rezultate incorecte
- 1 din 9 miliarde de împărțiri cu virgulă mobilă cu parametri aleatori ar produce rezultate inexacte
- Cost: aproximativ 500 milioane $ și pierderi enorme din punct de vedere al imaginii Intel

CE ESTE MODEL CHECKING-UL

- Descriere informală: Model Checking-ul este o tehnică automată care, având în vedere un model cu stări finite al unui sistem și o proprietate formală, verifică sistematic dacă această proprietate este valabilă pentru acel model
- Model Checking-ul a fost aplicat mai multor module ale acestei nave spațiale (NASA's Deep Space-1)



AVANTAJELE MODEL CHECKING-ULUI

- Aplicabil la scară largă (hardware, software, etc.)
- Permite verificarea parțială (doar a celor importante proprietăți)
- Sprijină interesul industrial de dezvoltare rapidă
- În cazul în care proprietatea este încălcată, se analizează contraexemplele generate oferite de simulare, apoi se îmbunătățesc modelul, design-ul și/sau proprietățile și se repetă întregul proces
- Are la bază proprietăți matematice solide

DEZAVANTAJELE MODEL CHECKING-ULUI

- Accentul se pune mai mult pe aplicațiile intensive în control
- Verificarea unui model este la fel de “bună” precum modelul sistemului
- Nu există nicio garanție cu privire la completitudinea rezultatelor
- Nu se pot verifica generalități

Cu toate acestea, Model Checking-ul este o tehnică foarte eficientă pentru a expune potențialele erori de proiectare.

EXEMPLE DE UTILIZARE ALE MODEL CHECKING-ULUI

- Securitate: protocolul de criptare Needham-Schroeder, eroare ce nu a fost descoperită timp de 17 ani
- Sisteme de transport: modelul unui sistem de trenuri continue 10^{476} stări
- Utilizate in limbaje comune precum C, Java, C++
- Utilizat (și dezvoltat) de Microsoft, Digital si NASA
- Utilizat cu succes la bariera olandeza împotriva furtunii în Nieuwe Waterweg
- În sistemele software ale actualelor generații de rachete spațiale
  - Mars Pathfinder al NASA, Deep Space-1, JPL LARS Group

Bariera olandeză

MODEL CHECKING

- Majoritatea device-urilor din jurul nostru vin împreună cu o bucată de cod care asigură controlul lor.
- Acest cod este controller-ul device-ului.
- De exemplu, avioanele calculează traiectoria prin valorile curente ale mai multor parametri.

CONTROLLER

- Controller-ul ar trebui să fie reliable. El vine cu un set de cerințe care trebuie să fie satisfăcute.
- Pentru a verifica faptul că un controller satisface cerințele, putem crea teste manual și să verificăm output-ul acestora (manual verification).
- În anumite cazuri, testarea manuală nu este cea mai bună abordare deoarece ar putea fi omise anumite erori fără a fi observate.

CONTROLLER

- O a doua modalitate, care teoretic este mai adecvată, este să transformăm controller-ul nostru într-un model matematic.
- Cerințele vor fi transformate folosind notații formale.
- Dacă modelul nostru matematic satisface notațiile formale, atunci modelul este corect.
- Avantajul acestei modalități este că deja există tool-uri online ce pot verifica pentru un model matematic dacă satisface anumite cerințe în notație formală.

EXEMPLE

Bounded Model Checking

- Avem o mulțime cu stări valide care este inclusă într-o mulțime cu stări invalide

`mulțime validă inclusă U mulțime stări invalide = mulțimea stărilor valide + mulțimea stărilor invalide`

Combinatorial Module

- Calcularea unui operații printr-o metodă, verificarea rezultatelor prin altă metodă, iar dacă ambele returnează același rezultat, atunci modelul este corect; în caz contrar, nu

Cover

- Verifică dacă poate ajunge dintr-o stare inițială într-o anumită stare, având ca input starea inițială și starea în care vrem să ajungem și un număr maxim de tranziții prin care putem trece. Pentru a realiza acest lucru se vor genera toate drumurile posibile.

Prove (metoda inducției)

- Presupune demonstrarea valabilității unei afirmații de bază (Base Case), urmată de demonstrarea că afirmația este validă și pentru următoarea stare, pornind de la starea precedentă (Următorul Case). Dacă starea precedentă este validă, atunci starea curentă va fi, de asemenea, validă, ceea ce implică faptul că, în final, toate stările vor fi valide. Astfel, dacă Base Case este valid și starea inițială este validă, atunci următoarea stare va fi, de asemenea, validă.

DEMONSTRAȚIE CALCUL MAXIM FOLOSIND NOTAȚII FORMALE

1) Cazul în care tabloul unidimensional are un singur element, nu intră în instrucțiunea repetitivă *for* și returnează direct primul element ca fiind maxim.

```
int Maximum(int[] values)
{
  int max = values[0];

  for (int i = 1;
       i < values.Length;
       i++)
  {
    if (values[i] > max)
    {
      max = values[i];
    }
  }

  return max;
}
```

$max \leftarrow Max\{k = 0 | values[k]\}$

Entire loop skipped if values.Length = 1

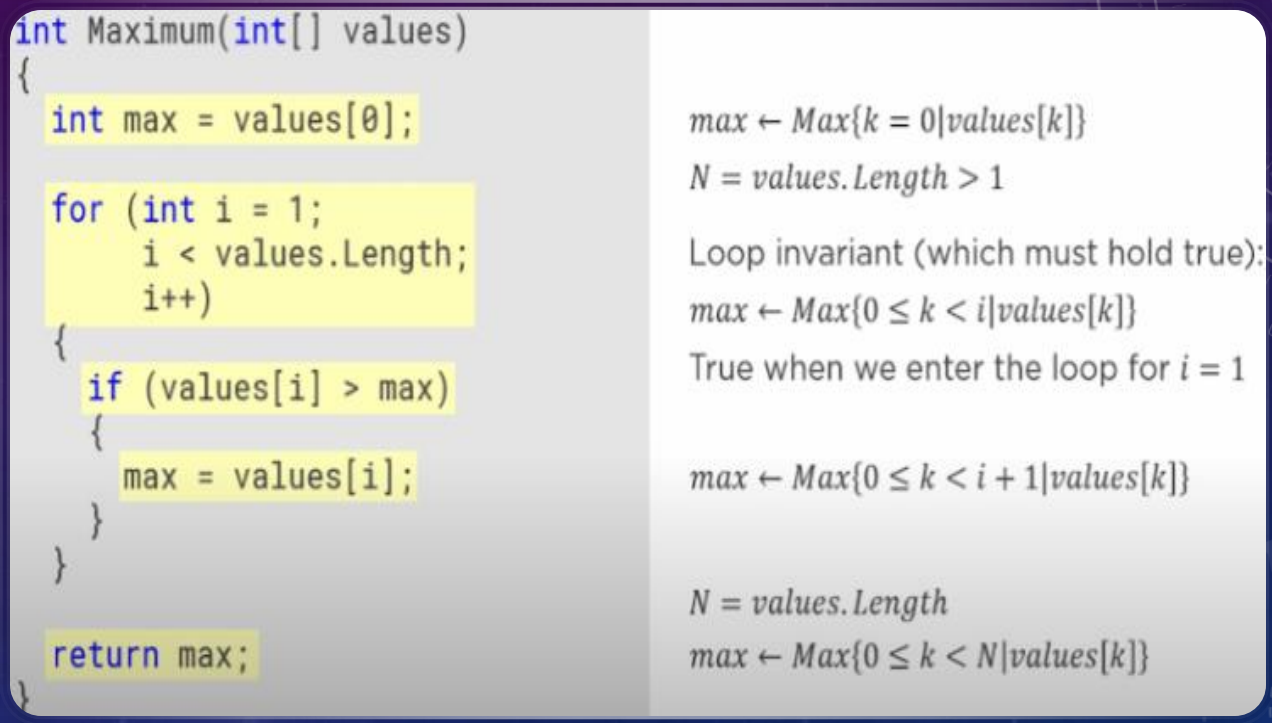
$N = values.Length$

$max \leftarrow Max\{0 \leq k < N | values[k]\}$

DEMONSTRAȚIE CALCUL MAXIM FOLOSIND NOTAȚII FORMALE

2) Cazul în care intra în instrucțiunea repetitivă *for*, parcurge toate elementele din tabloul unidimensional și verifică la fiecare pas dacă elementul curent este mai mare decât maximul actual. Dacă este mai mare, atunci actualizăm maximul.

După ce am terminat de parcurs tabloul unidimensional, returnăm maximul.



```
int Maximum(int[] values)
{
  int max = values[0];

  for (int i = 1;
       i < values.Length;
       i++)
  {
    if (values[i] > max)
    {
      max = values[i];
    }
  }

  return max;
}
```

$max \leftarrow Max\{k = 0 | values[k]\}$

$N = values.Length > 1$

Loop invariant (which must hold true):

$max \leftarrow Max\{0 \leq k < i | values[k]\}$

True when we enter the loop for $i = 1$

$max \leftarrow Max\{0 \leq k < i + 1 | values[k]\}$

$N = values.Length$

$max \leftarrow Max\{0 \leq k < N | values[k]\}$

MODEL CHECKING OF A ROBOTIC SYSTEM

- Articolul “On Model Checking of a Robotic System”, realizat de Adrian Turcanu, Talal Shaikh și Cristina Nicoleta Mazilu, discută despre verificarea de modele, platforma Rodin, mașinile cu stări finite și cinematica mecanismelor robotice.
- Rodin este o platformă utilizată pentru modelele Event-B, fiind un limbaj de modelare matematic riguros. Aceasta suportă rafinarea și demonstrația automată și are plugin-uri precum ProB. Verificarea de modele este o tehnică automată de verificare a sistemelor cu stări finite care explorează toate stările posibile ale sistemului. ProB suportă modelele B, Event-B, CSP-M, TLA+ și Z.
- Modelele Event-B constau în mașini și contexte. Mașinile conțin structura dinamică a sistemului, în timp ce contextele descriu structura statică. Event-B a fost aplicat cu succes în diferite domenii de cercetare.
- O mașină cu stări finite este un model de calcul care constă dintr-un set de stări, un alfabet de intrare și o funcție de tranziție. Există mai multe tipuri de mașini cu stări finite, inclusiv mașinile Moore și Mealy, mașinile de stări UML și mașinile de stări ierarhice. O mașină cu stări finite determinată este reprezentată printr-un tuplu de cinci, care include alfabetul de intrare, setul de stări, starea inițială, funcția de tranziție a stării și stările finale.

MODEL CHECKING OF A ROBOTIC SYSTEM

- În articolul “On Model Checking of a Robotic System”, se discută un studiu de caz intitulat „Un manipulator r-θ cu două articulații”, care implică un robot cu brațul controlat de la distanță. Mișcarea robotului este controlată de două motoare distincte capabile să-l manevreze în plan vertical și în plan orizontal.
- Pentru a asigura bună funcționare a acestuia, brațul robotului este supus unei simulări folosind modelul FSM, implementat utilizând limbajul de modelare formală Rodin și sistemul de dezvoltare bazat pe evenimente Event-B. Prin acest proces se verifică automat modelul FSM și este garantată siguranța brațului robotului în scenarii din viața reală, fără a pune în pericol utilizatorii sau echipamentele din jurul său.

MODEL CHECKING OF A ROBOTIC SYSTEM

- Modelarea cu FSM implică evenimente, tranziții și stări. Starea actuală a robotului este reprezentată de stări, în timp ce tranzițiile reflectă mișcarea acestuia de la o stare la alta. Evenimentele, pe de altă parte, descriu semnale de intrare care declanșează tranziții.
- Pentru modelarea brațului robotului folosind FSM, s-a construit un model Event-B care conține 2 contexte și 2 mașini, care corespund părții statice și a celei dinamice.

MODEL CHECKING OF A ROBOTIC SYSTEM

Contextul 1 (C1)

- Primul context (C1) conține setul de stări ale FSM-ului de rotație: `{INIT, s1, s2, STOP, CRASH}` și `{interm}`, cea din urmă fiind considerată pentru simplitatea modelării.

Mașina 1 (M1)

- Mașina corespunzătoare (M1) descrie tranziția între aceste stări conform setului dat de acțiuni.
- Evenimentul de inițializare pentru M1 este utilizat pentru a atribui valorile inițiale variabilelor folosite în model: starea inițială este `INIT`, valoarea lui `theta` fiind `0`, iar brațul poate fi rotit în orice poziție (`auxTheta`) din intervalul `[1, 359]`, direcția rotației fiind dată de `sign` (`-1` sau `1`).
- Evenimentul `move` este activat doar dacă starea FSM-ului este `INIT`, `s1`, sau `s2`. În funcție de rezultatul acțiunilor evenimentului, starea FSM-ului se schimbă în `s1`, `s2`, `STOP` sau `CRASH`. De asemenea, atunci când există o schimbare de stări, evenimentul `move` este activat (doar în cazul în care niciuna dintre stările `STOP` sau `CRASH` nu este atinsă).
- Un alt eveniment al mașinii M1 este `reachStop`, care se activează în starea intermediară atunci când valoarea lui `newTheta` (variabilă care reprezintă valoarea actualizată a lui `theta`) este reprezentată de una dintre valorile extreme `-180` sau `180`.

MODEL CHECKING OF A ROBOTIC SYSTEM

Contextul 2 (C2)

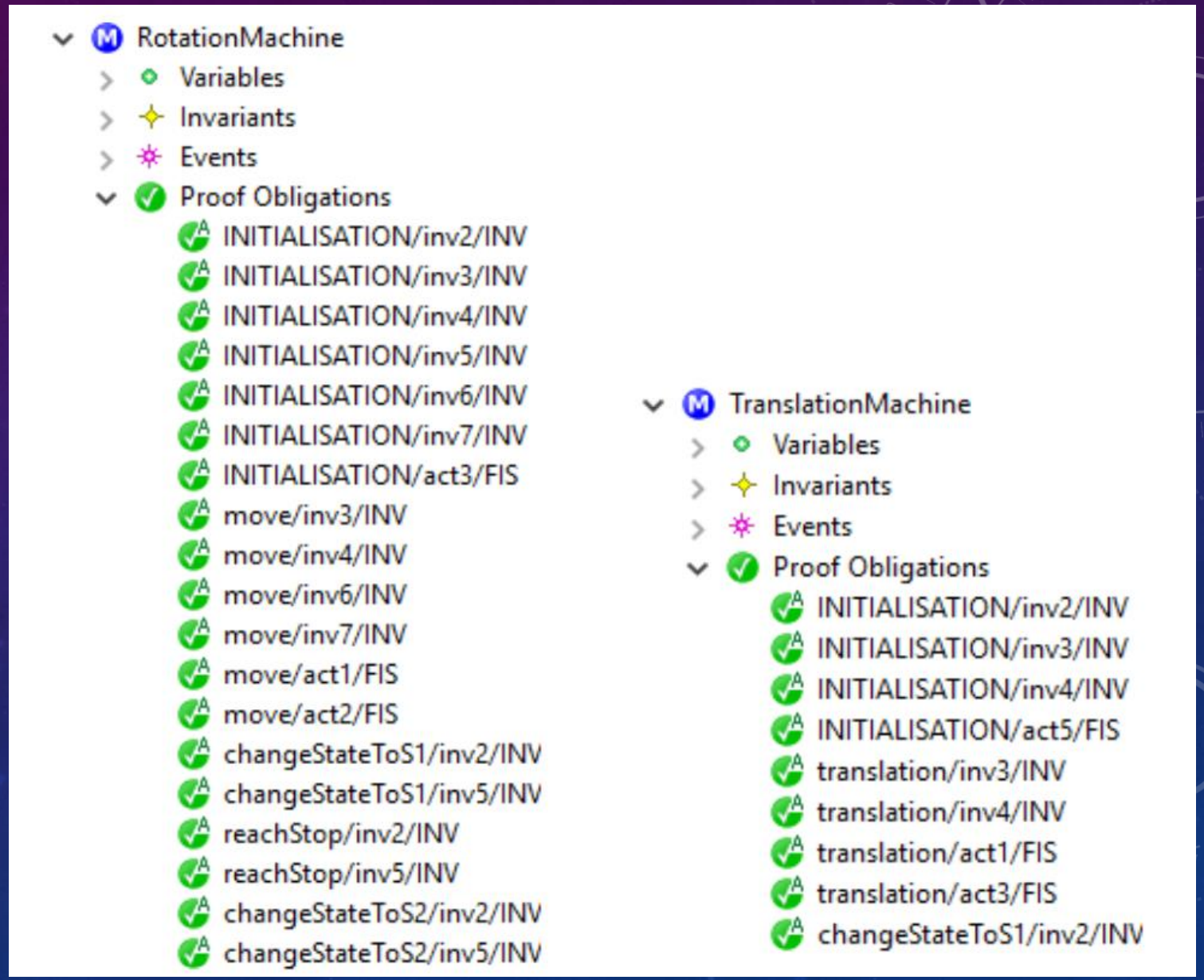
- Cel de-al doilea context (C2) conține setul de stări al FSM-ului de translație: `{INIT, s1, final, CRASH}` și `interm`, o stare intermediară. În acest context se regăsesc și axiomele pentru a seta lungimea inițială a brațului `r` și valoarea maximă a extensiei `MAX`.

Mașina 2 (M2)

- În mașină corespunzătoare (M2), evenimentul cheie este `translation`, care este activat în orice stare `INIT` și `s1`. Dacă lungimea maximă a brațului este atinsă, starea este actualizată la `FINAL`. În caz contrar, este activată starea `s1`.

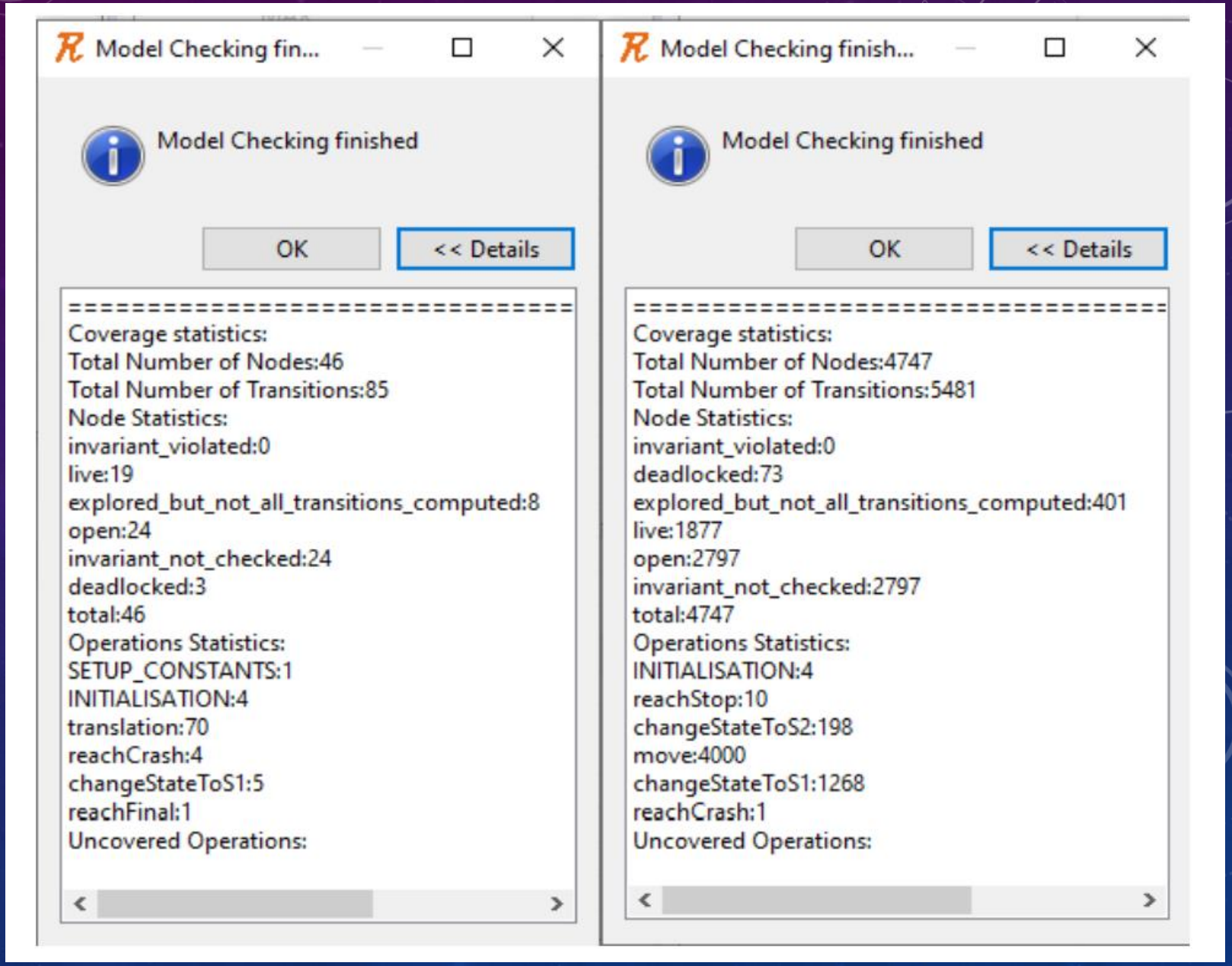
VALIDAREA MODELULUI

- Ulterior, modelul s-a validat folosind diferite tool-uri specifice Rodin. Mai întâi, toate teoremele au fost demonstrate automat de către sistem (proof of obligations).



VALIDAREA MODELULUI

- Apoi, modelul s-a rulat pe ambele mașini, constatându-se faptul că toate stările au fost vizitate și că nu există nicio încălcare de invariants.



VERIFICAREA MODELULUI

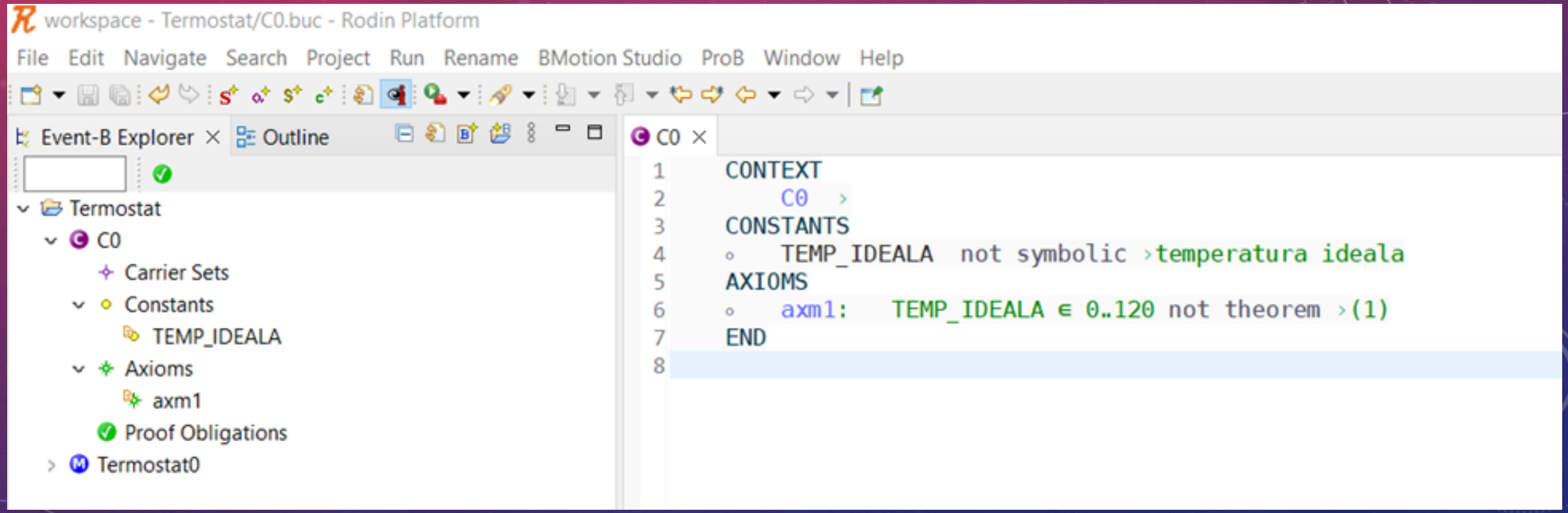
- Proprietățile LTL care descriu componentele manipulatorului au fost verificate și ele, pentru a îmbunătăți validarea modelului.

LTL property	**Meaning**	**Result**
G{*state*/ = *STOP*}	State stop is not reached for M1.	A counterexample has been found: (Move → changeStatetoS2)^{179} → move → reachStop
G{(*theta* > -50)&(*theta* < 50)}	Theta is in (-50, 50)	A counterexample has been found: (Move → changeStatetoS2)^{50}
G {*x* < 12}	The length of arm is less than 12	A counterexample has been found: (Translation → changeStatetoS1)^{2}
G{*state*/ = *final*}	State final is not reached for M2.	A counterexample has been found: (Translation → changeStatetoS1)^{35} → translation → reachFinal

DEMO

- Se dorește modelarea sistemului unui termostat și al unui aparat de aer condiționat.
- Cerințele sistemului:
  1. Sistemul are o temperatura ideală (care mereu este aceeași), nu este niciodată negativă și nu poate depăși 120 de grade.
  2. Temperatura termostatului poate fi controlată într-un mod controlat (maxim 3 grade) fie pentru a o crește, fie pentru a o scădea. Termostatul nu citește niciodată o temperatura negativă și niciodată nu trece peste 120 de grade.
  3. Aerul condiționat poate fi pornit sau oprit. Acesta poate fi pornit doar atunci când temperatura pe care o citește termostatul este mai mare decât temperatura ideală.
  4. Dacă aerul condiționat este pornit, temperatura nu poate crește (poate doar să scadă).
  5. O dată ce aerul condiționat este pornit, se poate închide doar dacă termostatul arată o temperatura mai mică sau egală cu temperatura dorită (cea citită de termostat).

CONTEXT

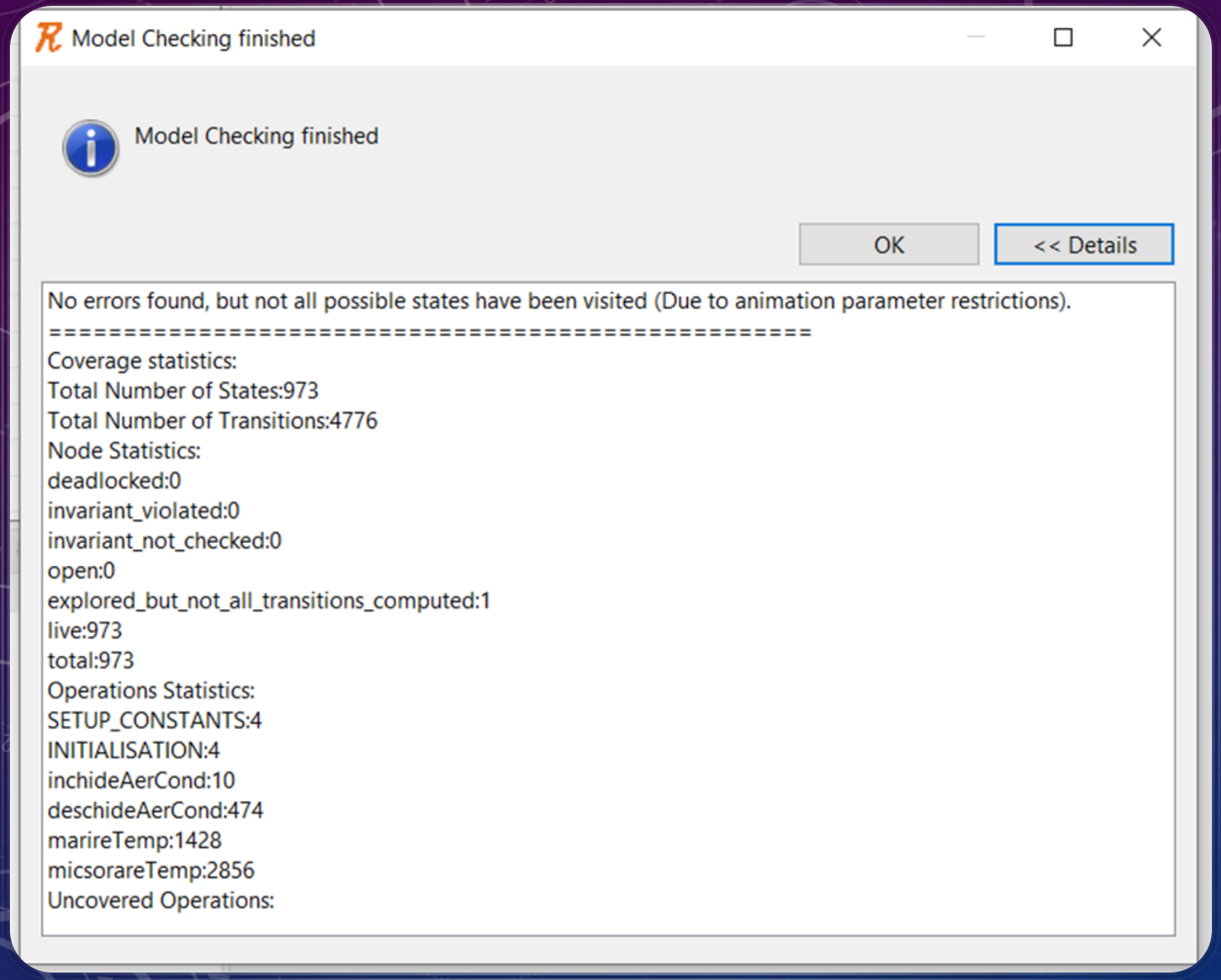

workspace - Termostat/C0.buc - Rodin Platform
File Edit Navigate Search Project Run Rename BMotion Studio ProB Window Help
Event-B Explorer
Outline
Termostat
C0
Carrier Sets
Constants
TEMP_IDEALA
Axioms
axm1
Proof Obligations
Termostat0
C0
1 CONTEXT
2 C0 ›
3 CONSTANTS
4 TEMP_IDEALA not symbolic ›temperatura ideala
5 AXIOMS
6 axm1: TEMP_IDEALA ∈ 0‥120 not theorem ›(1)
7 END
8

MAȘINĂ

Termostat0

```
MACHINE
    Termostat0  ›
SEES
◦    C0
VARIABLES
◦   tempTermostat    ›temperatura termostatului
◦   statusAerCond    ›TRUE = deschis, FALSE = inchis
INVARIANTS
◦   inv1:   tempTermostat ∈ 0‥120 not theorem ›(2)
◦   inv2:   statusAerCond ∈ BOOL not theorem ›(3)
EVENTS
◦   INITIALISATION:     not extended ordinary ›
    THEN
    ◦   act1:   tempTermostat ≔ 0 ›
    ◦   act2:   statusAerCond ≔ FALSE ›initial, AC-ul este inchis
    END

◦   inchideAerCond:     not extended ordinary ›
    WHERE
    ◦   grd1:   tempTermostat ≤ TEMP_IDEALA not theorem ›(5)
    ◦   grd2:   statusAerCond = TRUE not theorem ›(5)
    THEN
    ◦   act1:   statusAerCond ≔ FALSE ›
    END

◦   deschideAerCond:    not extended ordinary ›
    WHERE
    ◦   grd1:   tempTermostat > TEMP_IDEALA not theorem ›(3)
    ◦   grd2:   statusAerCond = FALSE not theorem ›(3)
    THEN
    ◦   act1:   statusAerCond ≔ TRUE ›
    END

◦   marireTemp:     not extended ordinary ›
    ANY
    ◦   t    ›
    WHERE
    ◦   grd1:   t ∈ 1‥3 not theorem ›(2)
    ◦   grd2:   tempTermostat + t < 120 not theorem ›(2)
```

```
◦   marireTemp:     not extended ordinary ›
    ANY
    ◦   t    ›
    WHERE
    ◦   grd1:   t ∈ 1‥3 not theorem ›(2)
    ◦   grd2:   tempTermostat + t ≤ 120 not theorem ›(2)
    ◦   grd3:   statusAerCond = FALSE not theorem ›(4)
    THEN
    ◦   act1:   tempTermostat ≔ tempTermostat + t ›
    END

◦   micsorareTemp:  not extended ordinary ›
    ANY
    ◦   t    ›
    WHERE
    ◦   grd1:   t ∈ 1‥3 not theorem ›(2)
    ◦   grd2:   tempTermostat − t ≥ 0 not theorem ›(2)
    THEN
    ◦   act1:   tempTermostat ≔ tempTermostat − t ›
    END

END
```



MODEL CHECKING

THANK YOU!