Model Checking cu NuSMV

...

prezentat de Simona Alexandru, Adriana Vladone & Alina Georgescu
SAL - grupa 510

Totul incepe cu TL (Temporal Logic)

- Logica clasică este specializata in descrierea structurilor statice.
- În contextul reprezentării sistemelor computationale, majoritatea structurilor sunt dinamice și în continuă schimbare.
- Logica temporală (TL) introduce operatori temporali care pot procesa schimbările, fără a se referi direct la unitatea tradiționale de timp.
- Tipurile de TL pot varia în felul în care se vizualizeaza succesiunea schimbărilor; doua tipuri relevante sunt LTL si CTL.

Logica LTL (Linear Temporal Logic)

- Este o extensie a logicii propoziționale (operează cu formule / variabile propoziționale)

- Reprezintă un formalism pentru descrierea evoluției unei structuri dinamice printr-o singură succesiune liniară de stări.

Logica LTL (Linear Temporal Logic)

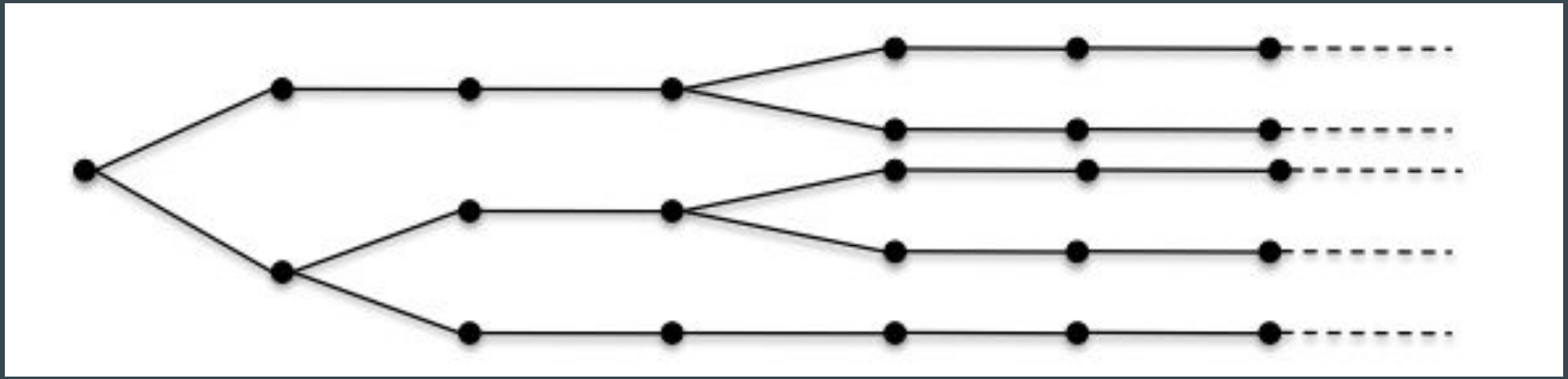
- Operatori temporali:
  - Fie f o formula propozițională:
    - F (Future) – f va fi adevărată cândva în viitor;
    - G (Globally) – f va fi adevărată în toate stările viitoare;
    - X (neXt) – f va fi adevărată în următoarea stare;
    - U (Until) – f va fi adevărată pana cand o alta proprietate devine adevărată.

- Exemplu: G (request –> F response): dacă o cerere este realizată, atunci va exista un răspuns în viitor.

Logica CTL (Computational Tree Logic)

- Ca și LTL, CTL este o extensie a logicii propoziționale.

- CTL este folosită în descrierea schimbărilor unei structuri dinamice unde pot exista mai multe stări posibile; acest aspect creează o vizualizare temporal arborescentă.



Logica CTL (Computational Tree Logic)

- Operatori temporali:
  - Fie f o formulă propoziţională:
    - A (All paths) – f este adevărată pe toate căile posibile;
    - E (Exists path) – f este adevărată pe cel puţin o cale;
  - F, G, X, U se regăsesc şi în CTL, dar fiecare dintre ele este în combinaţie cu A sau E.

- Exemplu: AG (request –> F response): Pe toate “drumurile” dacă o cerere este făcută, atunci va exista un răspuns în viitor.

LTL vs CTL

- LTL - model checking-ul implică exploatarea tuturor execuțiilor posibile în vederea verificării proprietăților specificate.

- CTL - model checking-ul implică crearea unui arbore de execuție, verificându-se, dacă proprietățile sunt adevărate, bazându-se pe tipul de verificare pe stări și al algoritmilor de traversare a arborilor.

Apoi, Model Checking...

- Model checking-ul este o tehnică de verificare formală a unui model ce satisface diferite proprietăți specificate.

- În funcție de tehnicile utilizate și tipurile de sisteme analizate, exista mai multe forme de model checking: symbolic, explicit-state, bounded etc.

Symbolic Model Checking

- Symbolic model checking este o tehnică puternică de verificare formală care asigura corectitudinea și siguranța sistemelor.

- În loc să enumere stările unei structuri în mod explicit, folosește structuri simbolice, cum ar fi Binary Decision Diagrams (BDDs), Satisfiability Modulo Theories (SMT) etc.

- Poate fi automatizată, scalată ușor pentru sisteme cu volum mare de date și operează mai eficient decat alte tipuri de model checking.

NuSMV

- NuSMV este un tool de verificare formală pentru un număr de stări finite.
- Combină symbolic model checking și utilizează diagrame de BDD pentru a verifica proprietăți ale logicii temporale, LTL și CTL.
- Scopul principal este de a descrie, folosind expresii din logica propozițională, tranziția între stări într-un cadru Kripke finit.

NuSMV

- Caracteristici:
    - Verificare completă: Permite verificarea completă a sistemului, explorând toate stările posibile și identificând erori precum blocaje sau comportamente incorecte.
    - Generare de contraexemple: pentru proprietățile care nu sunt îndeplinite, NuSMV poate genera un contraexemplu care arată secvența de stări ce duc la încălcarea acesteia.

NuSMV

- Caracteristici:
    - Analiza invariantelor: rutine specializate permit verificarea formulelor care trebuie să fie valabile în mod uniform pe întregul model, în timpul analizei de accesibilitate.
    - Metode de partiționare: modelul poate fi partitionat conjunctiv și disjunctiv, iar partițiile pot fi inspectate și, în cazul celei conjunctive, ordonate conform heuristicii definite.
-

Proprietăți LTL & CTL

Se verifică proprietățile LTL & CTL într-un sistem de tip cerere-răspuns.

- LTLSPEC G (request -> F response); Response always follows a Request

  De fiecare data cand o cerere este făcută va exista un răspuns în viitor.

  - Stări:
    - Starea 1.1
      - cerere: False
      - răspuns: False
    - Starea 1.2
      - cerere: True
      - răspuns: False
    - Starea 1.3
      - cerere: True
      - răspuns: True

- LTLSPEC G (response -> X ! response); Responses are not given repeatedly

  Dacă un răspuns este dat, atunci în următoarea stare, nu poate răspuns = True.

- CTLSPEC AG (request -> AF response); If a Request occurs, a Response will eventually occur

  Pe toate căile dacă o cerere este făcută, atunci va exista, eventual un răspuns în viitor.

- CTLSPEC AG (request -> AF (response | request)); If a Request is made, eventually either a Response is Given or a New Request is made

  Pe toate căile dacă o cere este făcută, atunci în cele din urma fie va exista un răspuns fie o alta solicitare.

- LTLSPEC G (response -> F request); No Response always without Request

  De fiecare data cand un răspuns este dat, a existat o cerere. În starea 1.3 proprietatea F(request) - eventual request va fi adevărat, face ca response -> F (request) să devină falsă.

- CTLSPEC AG (request -> EX (response | request)); If a Request is made, in the next state Response is Given or a New Request is made

  Pe toate căile dacă o cere este făcută, în următoarea stare va exista un răspuns fie o alta solicitare

```
-- specification AG (request -> AF response)  is true
-- specification AG (request -> AF (response | request))  is true
-- specification AG (request -> EX (response | request))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    request = FALSE
    response = FALSE
  -> State: 1.2 <-
    request = TRUE
  -> State: 1.3 <-
    response = TRUE
-- specification  G (request ->  F response)  is true
-- specification  G (response ->  F request)  is true
-- specification  G (response ->  X !response)  is true
```