# Verificarea și Validarea Sistemelor Soft

## Curs 09. Verificarea modelelor

Lector dr. Camelia Chisăliță-Crețu

Universitatea Babeș-Bolyai
Cluj-Napoca

28 Aprilie 2020

# Verificarea sistemelor. Motivație

- necesitatea utilizării unui software de calitate:
  - **motivația financiară** – defectele soft/hard au consecințe negative asupra producătorului:
    - bug-ul procesorului Intel Pentium II la împărțirea cu virgulă; **consecințe:** pierderi de 475 milioane $, înlocuirea procesoarelor defecte, reputația de producător de încredere (*engl.* **reliable**);
    - bug-ul unui sistemului de gestionare a bagajelor a dus la amânarea deschiderii aeroportului din Denver cu 9 luni; **consecințe:** pierderi de peste 1 milion $ pe zi;
    - imposibilitatea efectării rezervărilor on-line a biletelor pentru o companie aeriană timp de o zi; **consecințe:** pierderi majore și falimentul companiei.

# Verificarea sistemelor. Motivație (cont.)

- necesitatea utilizării unui software de calitate:
  - **motivația siguranței** – defectele soft/hard pot avea consecințe negative asupra vieții:
    - defectele dezastruoase în controlul softului unor dispozitive: racheta Ariane-5, naveta spațială Mars Pathfinder, avioane Airbus;
    - bug-ul dezastruos în controlul dispozitivului Therac-25 utilizat în terapia cu radiații; **consecințe:** 6 pacienți bolnavi expuși la o supradoză de radiații au decedat.

# Tehnologia informației și a comunicațiilor ([KB08], Cap. 1)

- Tehnologia Informației și a Comunicațiilor (TIC) – (*engl.* Information and Communications Technology, ICT);
  - peste 25 de instrumente TIC sunt utilizate zilnic (telefon, e-mail, internet banking, ATM, aparatură medicală, etc.);
- software reliability – aspect esențial în procesul de dezvoltare al unui produs soft [KB08].
  - complexitatea și interacțiunea cu alte componente și sisteme $\Rightarrow$ vulnerabilitate la defecte (numărul de defecțiuni crește exponențial cu numărul de componente care interacționează);
  - număr redus de defecte, timp redus de dezvoltare (*engl.* time-to-market) $\Rightarrow$ activitate de dezvoltare complexă.

# Clasificarea metodelor de verificare a sistemelor

- după nivel:
  - software
    - non-formale (practice):
      - statice: peer reviewing (**Curs 01. Inspectare**);
      - dinamice: testare (**Curs 02-06. Testare**);
    - formale (teoretice);
      - verificarea modelelor (*engl.* **model checking**) – tehnică de verificare care folosește specificația formală a sistemului;
      - demonstrarea corectitudinii programelor (**Curs 07. Corectitudine (Floyd, Hoare, Dijkstra)**);
  - hardware
    - emularea (testare), analiza structurală (analiza timpului de execuție, verificarea echivalențelor).

# Metode formale ([KB08], Cap. 1)

- **aplicabilitate:** pentru produsele soft complexe costul (timp și efort) realizării verificării este mai mare decât cel necesar construirii propriu-zise;
- **avantaje:**
  - stabilirea corectitudinii programelor aplicând rigoarea matematică;
  - introducerea timpurie a verificării în procesul de dezvoltare $\Rightarrow$ identificarea timpurie a defectelor.

# Tehnici de verificare bazate pe modele ([KB08], Cap. 1)

- o tehnică de verificare bazată pe modele
  - permite descrierea comportamentului sistemului folosind notații matematice precise;
- tipuri verificare
  - **testare** – observarea comportamentului sistemului în timpul execuției;
  - **verificarea modelelor** – explorarea exhaustivă a stărilor sistemului;
  - **simularea** – derularea unor experimente cu scenarii restrictive asupra modelului;
- verificarea modelelor poate descoperi erori subtile care nu au fost identificate în timpul testării sau simulării;
- *orice tehnică de verificare bazată pe modele este bună în măsura în care modelul asociat sistemului este bun.*

# Verificarea sistemelor bazată pe modele ([KB08], Cap. 1)

- **verificarea sistemelor bazată pe modele**
  - **obiectiv:** stabilește dacă procesul sau produsul studiat are anumite proprietăți, acestea fiind verificate la nivelul unui model asociat;
  - **proprietățile** se descriu pe baza *specificațiilor programului*, e.g., sistemul nu trebuie să ajungă într-o stare de blocaj – din care să nu se poată merge mai departe (*engl.* **deadlock scenario**);
- sistem dezvoltat corect – toate proprietățile obținute din specificații sunt satisfăcute;
  - **corectitudinea se exprimă relativ la specificație**, nu este o caracteristică absolută a produsului soft.

# Verificarea sistemelor bazată modele ([KB08], Cap. 1)

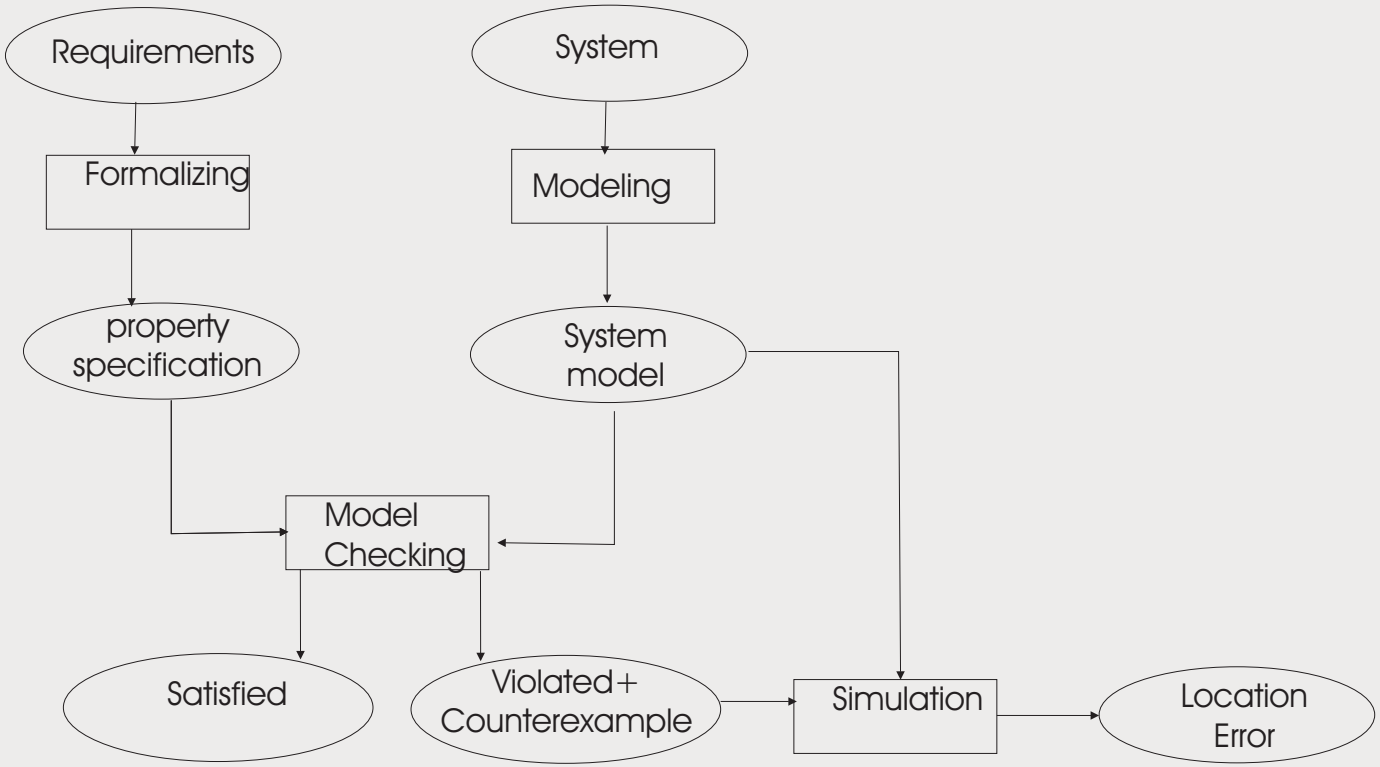



Figure: Verificarea sistemelor în abordarea bazată pe modele (model checking)

# Verificarea modelelor. Caracteristici. ([KB08], Sect. 1.1, 1.2)

- **verificarea modelelor** (*engl.* **model checking**)
  - tehnică automată prin care, pentru un model cu număr finit de stări asociat unui sistem și o proprietare formală, se verifică sistematic dacă proprietatea este satisfăcută de model;
- **model checker**
  - instrument soft care verifică pentru un model, dacă o anumită proprietate este satisfăcută, folosind sistematic toate scenariile posibile.

# Verificarea modelelor. Etape de realizare ([KB08], Sect. 1.2.1)

- desfășurarea procesului de verificare a modelului
  1. **modelarea sistemului** (*engl.* modeling)
     - modelarea sistemului pe baza limbajului de descriere a modelului;
     - formalizarea proprietăților care trebuie verificate, pe baza unui limbaj de specificare a proprietăților;
  2. **execuția** (*engl.* running)
     - verificarea unor proprietăți în modelul dezvoltat;
  3. **analiza rezultatelor** (*engl.* analysis).

# Verificarea modelelor. Etape de realizare (cont.)

- **analiza rezultatelor** (*engl.* analysis)
  - **proprietatea este satisfăcută** $\Rightarrow$
    (a) următoarea proprietate de verificat;
    (b) terminarea verificării modelului;
  - **proprietatea nu este satisfăcută** $\Rightarrow$ prin simulare se identifică drumul stărilor (counterexample) care a condus la eroare:
    • *eroare de modelare* (*engl.* **modeling error**) – modelul nu reflectă cerințele sistemului $\Rightarrow$ corectarea modelului și reluarea verificării modelului pentru **toate** proprietățile;
    • *eroare de proiectare* (*engl.* **design error**) – verificarea se încheie cu rezultat negativ $\Rightarrow$ proiectarea sistemului și modelul trebuie îmbunătățite și reluarea verificării modelului pentru **toate** proprietățile;
    • *eroare de proprietate* (*engl.* **property error**) – verificarea arată că proprietatea nu reflectă descrierea informală a cerințelor $\Rightarrow$ modificarea proprietății și reluarea verificării modelului pentru proprietatea curentă;
  - **gestionare dificilă a modelului** – resursele nu permit verificarea modelului (e.g., memorie insuficientă) $\Rightarrow$ simplificarea modelului și reluarea verificării modelului.

# Avantaje ale verificării modelelor ([KB08], Sect. 1.2.2)

- avantaje:
  - metodă generală de verificare
    $\Rightarrow$ se poate aplica pe tipuri diferite de aplicații;
  - **permite verificarea parțială**
    $\Rightarrow$ proprietățile pot fi verificate individual, sistemul nu necesită specificare completă;
  - furnizează informații justificative
    $\Rightarrow$ la invalidarea unei proprietăți $\Rightarrow$ utilitate în depanare;
  - ușurință în utilizare
    $\Rightarrow$ nu necesită cunoștințe de specialitate sau experiență;
  - interes din partea firmelor
    $\Rightarrow$ marile companii au propriile laboratoare de verificare;
  - ușor de integrat în ciclurile de dezvoltare
    $\Rightarrow$ scade timpul de dezvoltare.

# Dezavantaje ale verificării modelelor ([KB08], Sect. 1.2.2)

- dezavantaje
  - potrivită pentru aplicații de control, dar nu și pentru prelucrarea datelor
    $\Rightarrow$ domeniile de valori asociate stărilor pot fi infinite;
  - nu este eficientă pe structuri de date abstracte
    $\Rightarrow$ se pune problema decidabilității;
  - **verifică un model atașat sistemului, nu sistemul propriu-zis**
    $\Rightarrow$ sunt necesare metode complementare pentru a identifica anumite tipuri de erori, e.g., testare;
  - **verifică doar cerințele descrise (precizate), nu garantează completitudinea**
    $\Rightarrow$ validitatea proprietăților care nu sunt verificate nu poate fi discutată.

# Dezavantaje ale verificării modelelor (cont.) ([KB08], Sect. 1.2.2)

- dezavantaje (continuare):
  - **existența problemei exploziei spațiului de stări**
    $\Rightarrow$ numărul de stări crește exponențial cu numărul de variabile și numărul de componente necesare modelării sistemului $\Rightarrow$ depășește capacitatea de memorare a sistemului;
  - **reducerea dimensiunii modelului verificat nu este trivială**
    $\Rightarrow$ sunt necesare cunoștințe aprofundate pentru identificarea abstractizărilor adecvate care reduc dimensiunea modelului și permit specificarea formală a proprietăților de verificat;
  - **nu garantează obținerea unor rezultate corecte**
    $\Rightarrow$ **model checker-ul este un produs soft, el însuși poate avea defecte.**

# Limbaje de modelare

- limbaje formale pentru:
  - **descrierea modelului verificat**
    - sisteme de tranziție;
  - **specificarea proprietăților verificate**
    - logici temporale ([Fre10, KB08]):
      - LTL (Linear Temporal Logic);
      - CTL (Computation Tree Logic).

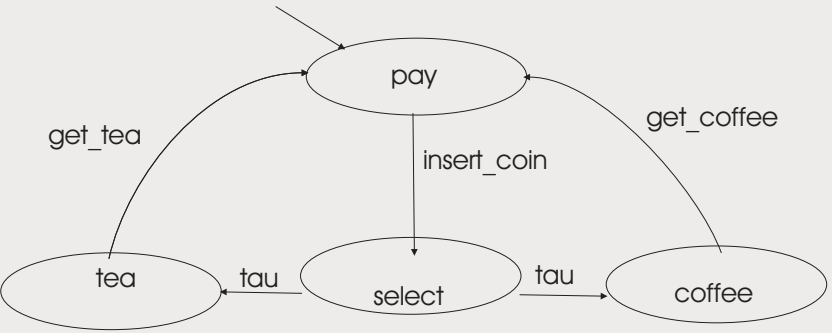
# Sistem de tranziții ([KB08], Sect. 2.1; [Fre10], Cap. 5)

- sistem de tranziții
  - model pentru descrierea comportamentului unui sistem soft;
  - graf orientat:
    - vârfuri (*engl.* nodes) - stările sistemului;
    - arce (*engl.* edges) - tranzițiile modelului, sugerează modificarea stărilor sistemului;
- un **sistem de tranziții** (*engl.* transition system, **TS**) este un 6-tuplu TS=(S, Act, $\rightarrow$, I, AP, L), unde:
  - **S** – mulțime de stări;
  - **Act** – mulțime de acțiuni (transformări);
  - $\rightarrow \subseteq S \times Act \times S$ – relație de tranziție între stări;
  - $I \subseteq S$ – mulțime de stări inițiale;
  - **AP** – mulțime de propoziții atomice (*engl.* atomic proposition, **ap**);
  - $L : S \rightarrow 2^{AP}$ – funcție de etichetare a stărilor.
- TS este finit dacă S, Act și AP sunt finite.

# Sistem de tranziții. Observații. ([KB08], Sect. 2.1; [Fre10], Cap. 5)

- comportamentul intuitiv al sistemelor de tranziții
  - stare inițială $s_0 \in I$
  - evoluția (transformarea) sistemului se realizează pe baza relației de tranziție între stări ($\rightarrow$);
  - dacă $s$ este o stare curentă, $s \xrightarrow{\alpha} s'$ este o tranziție;
  - procedura de selectare se repetă și se încheie atunci când nu există tranziții din starea în care s-a ajuns;
- funcția de etichetare L asociază o mulțime $L(s) \in 2^{AP}$ ale propozițiilor atomice cu orice stare $s$;
  $L(s)$ reprezintă acele propoziții atomice $a \in AP$ care sunt satisfăcute de starea $s$;
- fiind dată $\phi$ – o propoziție logică, atunci $s$ satisfce propoziția $\phi$ dacă evaluarea lui $L(s)$ determină propoziția $\phi$ să fie `true`,
  $s \models \phi$ iff $L(s) \models \phi$.

# Automatul de Cafea – CoffeeATM




- $S = \{pay, select, tea, coffee\}$;
- $I = \{pay\}$;
- $Act = \{insert\_coin, get\_tea, get\_coffee, \tau\}$;

- relația de tranziție, e.g., $pay \overset{insert\_coin}{\rightarrow} select$, $coffee \overset{get\_coffee}{\rightarrow} pay$;
- propoziții atomice ($AP$) și funcția $L$:
  - propozițiile atomice depind de proprietățile verificate
    • soluția imediată – numele stărilor să fie propozițiile atomice, i.e., $L(s) = \{s\}$;
  - propozițiile atomice nu fac referire la proprietățile verificate
    • e.g., "CoffeeATM livrează băuturile după ce s-a introdus o monedă;"
    • $AP = \{paid, drink\}$, $L(pay) = \emptyset$, $L(tea) = L(coffee) = \{paid, drink\}$, $L(select) = \{paid\}$.

# Proprietăți ale sistemelor ([KB08], Sect. 1.2.1, Cap. 2)

- corectitudine (engl. **correctness**);
  - "sistemul face ceea ar trebui să facă? "
- identificarea blocajelor (*engl.* **deadlock**, **reachability**);
  - "este posibil să se ajungă într-o stare de blocare? "
- proprietate de siguranță (*engl.* **safety**);
  - "nothing bad should happen! ";
    - e.g., Numărul de monede introduse este întodeauna cel puțin numărul de băuturi livrate;
- proprietate de certitudine (*engl.* **liveness**);
  - "something good will happen in the future! ";
- proprietate de execuție optimă (*engl.* **real-time**);
  - "sistemul reacționează în timpul precizat/stabilit? "

# Logica temporală. Definiție. ([KB08], Sect. 5.1; [Fre10], Cap. 5)

- **logica (propozițională) temporală**
  - extensie a logicii propozițiilor cu operatori care sugerează comportamentul sistemului în timp;
  - furnizează o notație matematică pentru descrierea unor proprietăți ale relațiilor existente între etichetelor stărilor, i.e., propozițiile atomice, în timpul execuției;
- adjectivul "temporal"
  - sugerează ordinea relativă a evenimentelor:
    - mașina s-a oprit după ce șoferul a frânat;
    - mesajul a fost primit după ce a fost trimis;
  - **nu poate referi durata de realizare a evenimentelor sau distanța în timp dintre evenimente**;
- domeniul de valori al timpului este discret $\Rightarrow$ starea sistemului este observabilă la momente de timp ordonate: 0, 1, 2, ...

# Logica temporală. Caracteristici. ([KB08], Sect. 5.1; [Fre10], Cap. 5)

- componenta timpului în logica temporală:
  - **liniară**;
  - **arborescentă (cu ramificații)**;
- operatori
  - moșteniți (logica propozițiilor): $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$;
  - "**cândva**" (cândva în viitor, *engl.* **eventually**) – $\Diamond$;
  - "**întotdeauna**" (acum și în viitor, *engl.* **always**) – $\Box$.

# Logica temporală liniară ([KB08], Sect. 5.1; [Fre10], Cap. 5)

- **logica temporală liniară** (*engl.* linear temporal logic, **LTL**)
  - logică temporală în care, pentru orice moment de timp există un singur moment următor (succesiv);
  - caracteristici:
    - stare – moment de timp;
    - tranziție – avansarea cu o unitate de timp;
    - moment următor – următoarea stare;
    - sintaxa LTL - reguli de construire a formulelor LTL;
    - semantica LTL – **secvență cu număr infinit de stări** ([Fre10]).

# Sintaxa LTL ([KB08], Sect. 5.1.1; [Fre10], Cap. 5)

- sintaxa LTL
  - reguli de construire a specificațiilor proprietăților (formule);
- operanzi: $a \in AP$, $a$ etichete ale stărilor $s$;
- operatori:
  - moșteniți (logica propozițiilor): $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$;
  - temporali, specifici (logica temporală):
    - "cândva" (*engl.* **eventually**) – $\Diamond$;
    - "întotdeauna" (*engl.* **always**) – $\Box$;
  - temporali, proprii (LTL) – liniaritate:
    - "următorul" (*engl.* **next**) – $\bigcirc$;
      - $\bigcirc\ \varphi$ – satisfăcută într-o stare $s_i$ dacă $\varphi$ are loc în starea următoare ($s_{i+1}$);
    - "până când" (*engl.* **until**) – $\bigcup$;
      - $\varphi\ \bigcup\ \psi$ – satisfăcută dacă $\psi$ are loc cândva (în viitor) într-o stare $s$, iar $\varphi$ are loc până se ajunge în starea $s$;
- aritatea operatorilor: • unari: $\neg, \Diamond, \Box, \bigcirc$; • binari: $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$, $\bigcup$.

# Formule LTL ([KB08], Sect. 5.1.1; [Fre10], Cap. 5)

- formulele LTL se definesc pe mulțimea $AP$, corespunzător gramaticii:
  - $\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \bigcup \varphi_2$, unde $a \in AP$.
- AP – mulțimea propozițiilor atomice atașate stărilor sistemului de tranziții;
- operatori logici pentru conjuncție ($\wedge$) și negație ($\neg$);
- operatori temporali liniari "next" ($\bigcirc$) și "until" ($\bigcup$).

# Semantica operatorilor temporali. Exemple.

- operatorul "until" permite obținerea construcției
  $\Diamond$ ("eventually", cândva în viitor) și
  $\Box$ ("always", începând cu starea curentă):
  - $\Diamond\varphi = \text{true} \bigcup \varphi$;
    - $\Diamond\varphi$ asigură că $\varphi$ va avea loc cândva în viitor;
  - $\Box\varphi = \neg\Diamond\neg\varphi$;
    - $\Box\varphi$ are loc dacă și numai dacă nu există posibilitatea ca $\neg\varphi$ să aibă loc, i.e., $\varphi$ are loc.

# Semantica operatorilor temporali. Exemple. (cont.)

- combinarea operatorilor temporali $\Diamond$ și $\Box$, permite obținerea unor construcții temporale noi:
  - $\Box\Diamond\varphi$ – "infinitely often $\varphi$";
    • la orice moment j există un moment $i$, $i \geq j$ la care o stare etichetată cu $\varphi$ este vizitată, i.e., o stare etichetată cu $\varphi$ este vizitată de o infinitate de ori;
  - $\Diamond\Box\varphi$ – "eventually forever $\varphi$";
    • de la un moment j mai departe doar stările etichetate cu $\varphi$ sunt vizitate.

# Semnificația intuitivă a operatorilor temporali

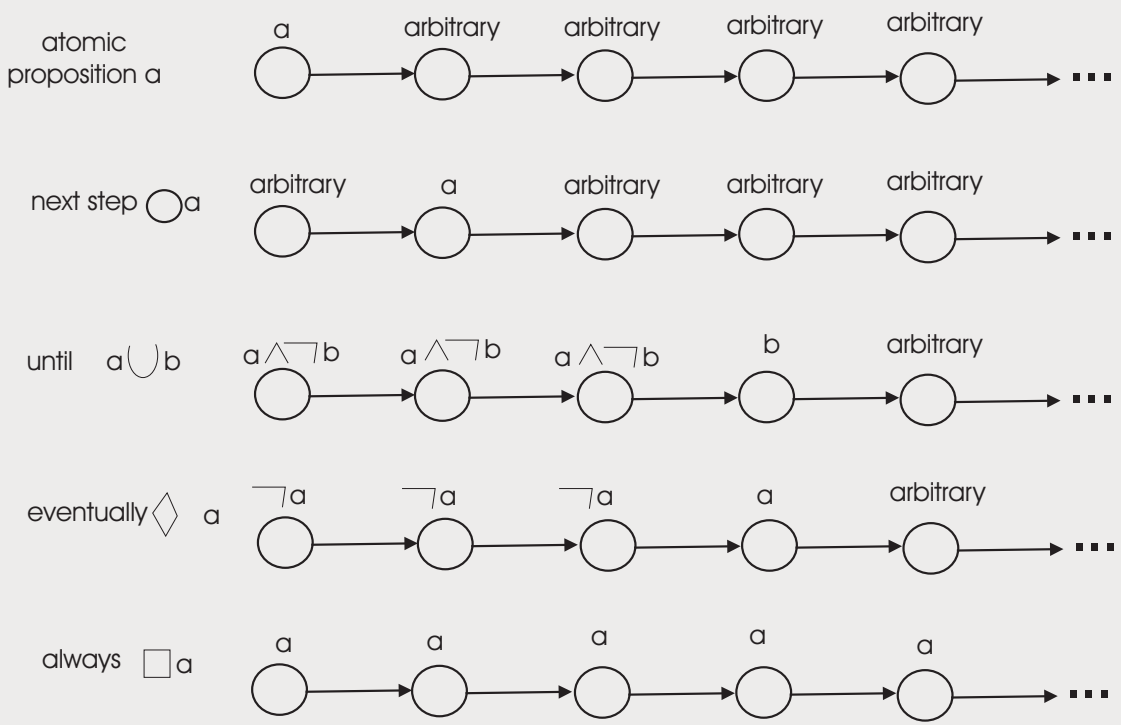


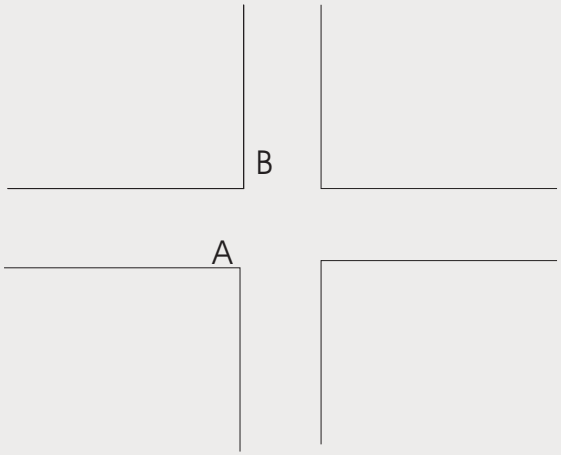

Figure: Semnificația intuitivă a operatorilor temporali

# Specificarea proprietăților. Exemple (1)

- semafor cu 3 stări: **roșu**, **galben** și **verde**;
- specificarea proprietăților/cerințelor:
  - **certitudine** (*engl.* **liveness**, ceva bun se va întâmpla):
    - **(întotdeauna) semaforul va fi (cândva) verde;**
    - $\Box \Diamond \mathit{verde}$;
  - cerințe:
    - **(întotdeauna) culoarea roșie a semaforului nu se schimbă imediat în verde;**
      $\Box (\mathit{rosu} \rightarrow \neg \bigcirc \mathit{verde})$;
    - **culoarea roșie a semaforului se poate schimba în verde;**
      $\Box (\mathit{rosu} \rightarrow \bigcirc (\mathit{rosu} \bigcup (\mathit{galben} \wedge \bigcirc (\mathit{galben} \bigcup \mathit{verde}))))$.

# Specificarea proprietăților. Exemple (2)

- intersecție cu două semafoare: $A$ și $B$;
- fiecare poate avea una din culorile:
  roșu, galben sau verde, i.e., $A,\ B \in \{rosu,\ galben,\ verde\}$;
- proprietăți:
  - $\square(\neg(A = verde\ \wedge\ B = verde))$;
    - (întotdeauna) A și B nu pot fi simultan verde;
  - $\square(A = galben \rightarrow \Diamond A = rosu)$;
    - (întotdeauna), dacă A este galben va deveni cândva roșu;
  - $\square(A = galben \rightarrow \bigcirc(A = rosu))$;
    - (întotdeauna), dacă A este galben, el va deveni rosu în starea următoare;
  - $\square(\neg(B = verde) \bigcup (A = rosu))$;
    - (întotdeauna) B nu va fi verde până când A nu va fi roșu.



# Logica computațională arborescentă ([KB08], Sect. 6.1; [Fre10], Cap. 5)

- **logica computațională arborescentă** (*engl.* computation tree logic, **CTL**)
  - logică temporală în care, pentru orice moment de timp pot există mai multe momente următoare (succesive);
  - caracteristici:
    - stare – moment de timp;
    - tranziție – avansarea cu o unitate de timp;
    - moment următor – una dintre stările ulterioare posibile;
    - sintaxa CTL - reguli de construire a formulelor CTL;
    - semantica CTL – **arbore cu număr infinit de stări** ([Fre10]).

# Sintaxa CTL ([KB08], Sect. 6.2.1; [Fre10], Cap. 5)

- sintaxa CTL
  - reguli de construire a specificațiilor proprietăților (formule);
- operanzi: $a \in AP$, $a$ etichete ale stărilor $s$;
- operatori:
  - moșteniți (logica propozițiilor): $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$;
  - temporali, specifici (logica temporală):
    - "eventual" (*engl.* **eventually**) – $\Diamond$;
    - "întotdeauna" (*engl.* **always**) – $\Box$;
  - temporali, specifici (LTL):
    - "următorul" (*engl.* **next**) – $\bigcirc$;
    - "până când" (*engl.* **until**) – $\bigcup$;
  - temporali, specifici (CTL) – permit ramificarea:
    - "toate/orice" (*engl.* **all**; cuantificator universal) – $\forall$;
      - $\forall \Diamond \Phi$ – toate stările satisfac proprietatea $\Diamond \Phi$;
    - "există/câteva" (*engl.* **some**; cuantificator existențial) – $\exists$;
      - $\exists \Diamond \Phi$ – există cel puțin o stare care satisface $\Phi$ și va fi atinsă în viitor.

# Formule CTL ([KB08], Sect. 6.2.1; [Fre10], Cap. 5)

- tipuri de formule CTL:
  - **formule CTL pentru stări** – proprietăți ale stărilor;
  - **formule CTL pentru drumuri** – proprietăți temporale ale drumurilor, i.e., o secvență infinită de stări;
    - formulele CTL pentru drumuri vs. formule LTL:
      • **asemănări:** folosesc operatorii temporali **next** ($\bigcirc$) și **until** ($\bigcup$);
      • **diferențe:**
      - operatorii temporali nu se combină cu operatorii de tip Boolean;
      - nu este permisă imbricarea operatorilor temporali.

# Formule CTL (cont.) ([KB08], Sect. 6.2.1; [Fre10], Cap. 5)

- **formulele stărilor CTL** se definesc pe mulțimea $AP$, corespunzător gramaticii:

$$\phi ::= true \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists\varphi \mid \forall\varphi,$$

unde $a \in AP$ și $\varphi$ este o formulă pentru drumuri;

- **formulele drumurilor CTL** se definesc corespunzător gramaticii:

$$\varphi ::= \bigcirc \phi \mid \phi_1 \bigcup \phi_2,$$

unde $\phi$, $\phi_1$ și $\phi_2$ sunt formule ale stărilor CTL.

# Semnificația intuitivă a operatorilor temporali

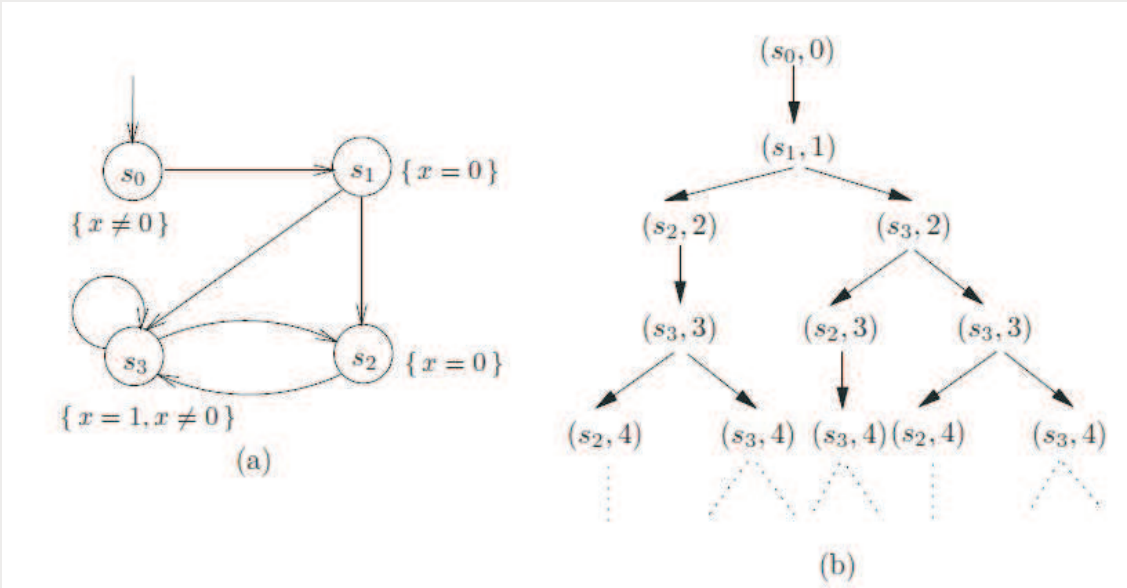



Figure: (a) un sistem de tranzitii; (b) prefixul unui arbore computational logic infinit

# Semantica operatorilor CTL

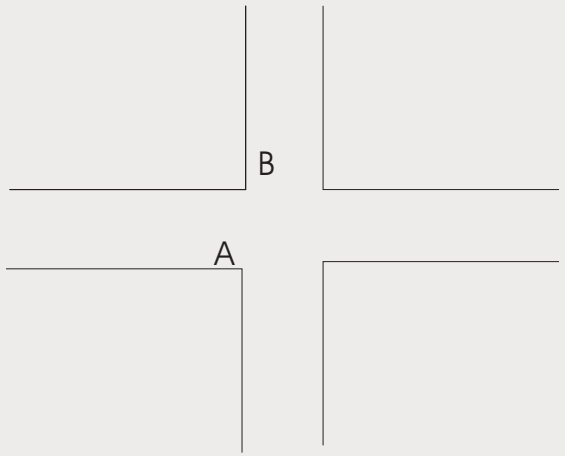
- operatori temporali pentru drumuri $\bigcirc$ și $\bigcup$
  - $\bigcirc\phi$ este satisfăcută dacă $\phi$ are loc în următoarea stare a drumului;
  - $\phi \bigcup \psi$ este satisfăcută pe un drum dacă există o stare $s$ de-a lungul drumului pentru care $\psi$ are loc, iar $\phi$ are loc pentru toate stările predecesoare ale lui $s$.
- formulele pentru drumuri se pot converti în formule pentru stări prin prefixare:
  - **cuantificator de drum existențial** ($\exists$),
    cu semnificația *există un drum*, (*engl.* "for some path");
    • $\exists\phi$ – are loc într-o stare $s$ dacă există un drum care satisface $\phi$ și care pornește din starea $s$;
  - **cuantificator de drum universal** ($\forall$),
    cu semnificația *pentru toate drumurile* (*engl.* "for all paths");
    • $\forall\phi$ – are loc într-o stare $s$ dacă toate drumurile care pornesc din această stare satisfac $\phi$.

# Specificarea proprietăților în CTL. Exemple (1)

- semafor cu 3 stări: **roșu**, **galben** și **verde**;
- specificarea proprietăților:
  - siguranță (*engl.* **safety**, *nu se va întâmpla ceva rău*):
    - **fiecare culoare roșie este precedată de galben;**
    - $\forall \Box (galben \vee \forall \bigcirc \neg rosu)$;
  - certitudine (*engl.* **liveness**, *ceva bun se va întâmpla*):
    - **semaforul este (cândva) verde de o infinitate de ori;**
    - $\forall \Box \forall \Diamond verde$.

# Intersecție cu 2 semafoare

- intersecție cu două semafoare: $A$ și $B$;
- fiecare poate avea una din culorile:
  roșu, galben sau verde $A,\ B \in \{rosu,\ galben,\ verde\}$;
- proprietate:
  - $\forall\Box(B = galben \rightarrow \forall \bigcirc (B = rosu))$.
    - (întotdeauna), dacă semaforul B este galben el va deveni imediat roșu.



# Întrebări pentru examen

- verificare formală: definiție, etape de realizare, avantaje, dezvantaje;
- sistem de tranziții - definiție;
- definirea unor proprietăți ale sistemului: correcteness, liveness, safety, real-time, deadlock;
- logica temporală (TL): definiție, operatori;
- logica temporală liniară (LTL) + logica temporală computațională (CTL): definiție, operatori, specificarea proprietăților (exemplele de la curs).

# Urmează...

- săpt.10, Vineri, 08 Mai, orele 08:00-10:00
  Curs 10. QA. QC - Prezentare Endava (online);
- săpt.11 - Curs 11. Recapitulare pentru examen.
  Model de subiect pentru examen;
  ?Simulare examen scris?
- săpt.12 - Curs 12. Prezentarea referatelor.

# Bibliografie I

[Fre10] M. Frentiu.
*Verificarea și validarea sistemelor soft*.
Presa Universitară Clujeană, 2010.

[KB08] J.P. Katoen and C. Baier.
*Principle of Model Checking*.
MIT Press Cambridge, London, UK, 2008.