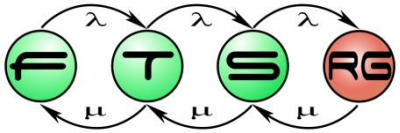
Rendszermodellezés

Modellellenőrzés, programverifikáció



Ismétlés: Mire használunk modelleket?

- Kommunikáció, dokumentáció
- Gondolkodás, tervezés támogatása
- Analízis
- Származtatás
- Szimuláció
- …belefér egy számítógépbe / emberi agyba

Ismétlés: Felhasználás - Analízis

- Emberi erővel vagy (részben) automatizáltan
- Cél
  - Ellenőrzés, hibák keresése (best effort)
  - Szolgáltatásbiztonsági kritériumok igazolása (erősebb!)
  - Jellemzők számítása, tervezése (pl. ütemezés)
- Módszer
  - Felületes, statikus elemzés – végrehajtás nélkül
    - Formális állítások bizonyításával
  - Dinamikus (végrehajtva)
    - Szimuláció / teszt (szúrópróba jelleg)
    - Kimerítő állapottér-bejárás, **modellellenőrzés**

Félreértések eloszlatása

„Modellellenőrzés” ≠ Modellek ellenőrzése

MODELLELLENŐRZÉS

Modellek kimerítő dinamikus analízise

Modellellenőrzés

- Adott egy (nemdeterminisztikus) viselkedésmodell
  - Diszkrét állapotátmenetek („tranzíciós rendszer”)
- Adottak formális követelmények
  - Deadlockmentesség
  - Biztonsági kritériumok (invariánsok)
  - Vezérelhetőség
  - …
- A modell kielégíti-e a követelményeket?
  - A teljes állapottér felderítésével eldönthető

Motiváció

- Biztonságkritikus beágyazott rendszerek
  - Pl. repülőgép/autó fedélzeti elektronika
  - Hibák előzetes felderítése létfontosságú
- Modell alapú fejlesztés és kódgenerálás
  - pl. SCADE – adatfolyam alapú fejlesztőeszköz
- Rockwell Collins esettanulmány
  - ADGS-2100 Adaptive Display & Guidance System WM
  - Lockheed Martin Aero robotrepülő vezérlő
  - *komponensenként* akár 10^{13} méretű állapottér
  - 0 hibát találtak teszteléssel, 12-t modellellenőrzéssel

Motiváció

- Miért nem használ mindenki modellellenőrzést?
  - Nehéz, lassú folyamat
  - Magas képzettséget igényel
- Mikor használják mégis?
  - Biztonságkritikus rendszerek tervezése
  - Protokollhibák keresése
  - Programverifikáció
- Trend
  - Egyre jobb eszközök
  - Egyre több célra éri meg

Mit ellenőrzünk?

- TFH már létező rendszert modelleztünk
- Modellezés validálása
  - Valóban jól modelleztük a rendszert?
  - Azt modelleztük, amit akartunk?
  - Szimuláció
  - Ellenőrző kritériumok igazolása
- Modell verifikációja
  - Az eredeti rendszer követelményei alapján
  - Ez a cél, a validálás csak feltétel

Hogyan ellenőrizzünk?

- Naiv módszer: **állapottér-bejárás**
- Minden elérhető állapotra igazak a kritériumok?
- Ha igen: **siker**
- Ha nem: **ellenpélda**
  - Rossz állapot az odavezető úttal együtt
  - Segít megtalálni a hibát
  - Elhárítása után új ellenőrzés

csak biztonsági kritériumok (invariánsok)

Problémák

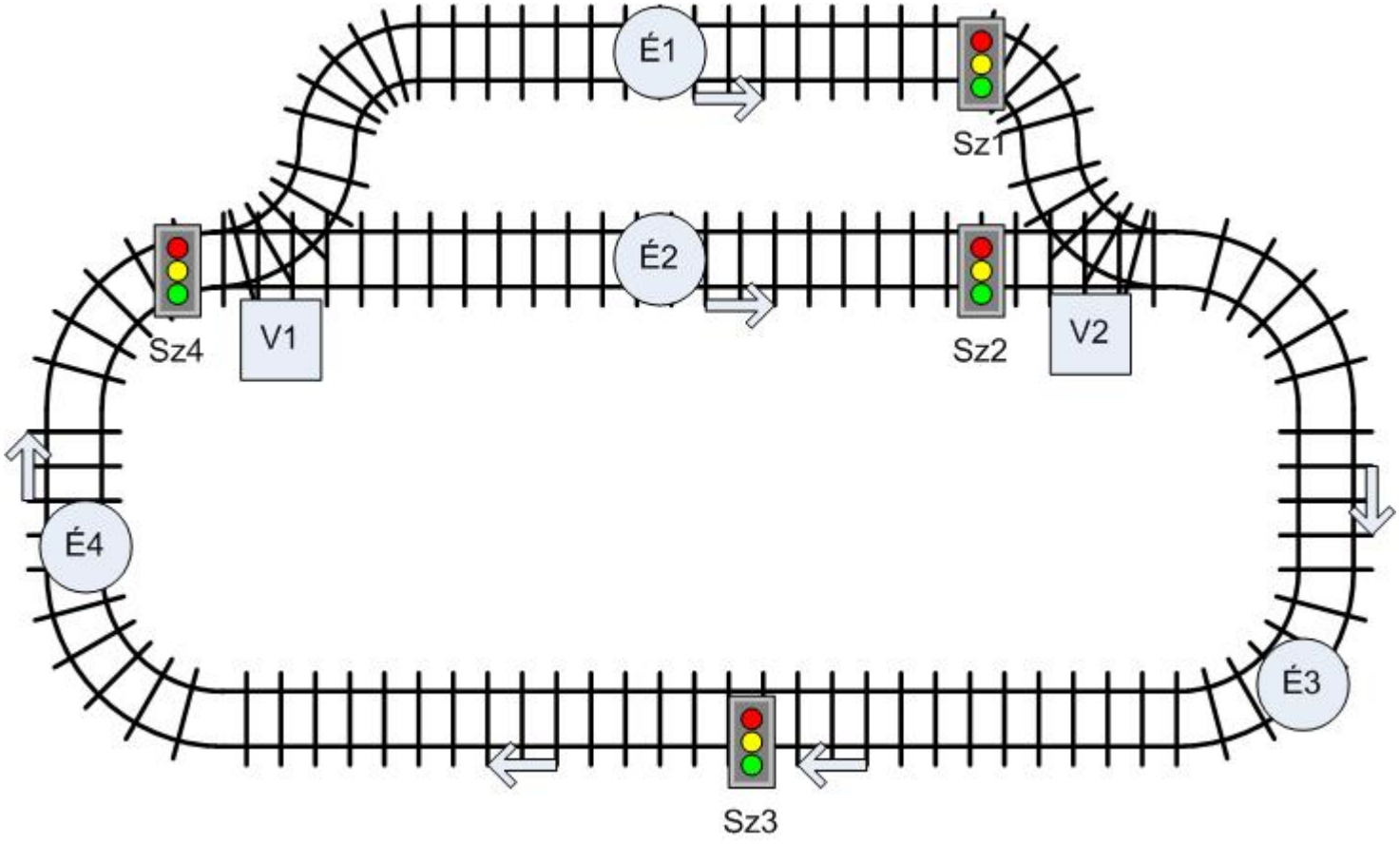
- Exponenciálisan nagy lehet az állapottér!
  - Állapotváltozók értékkészletének Descartes-szorzata…
  - Ne kézzel, hanem szoftverrel…
  - …és ügyes algoritmusokkal, heurisztikákkal
- Mit kezdünk a nem állapotra felírt kritériumokkal?
  - „Ha A állapotba lép, később elkerülhetetlenül B lesz”
  - Ügyes algoritmus, vagy megkötött nyelv kell
- Ezért kellenek: MC (Model Checker) eszközök

Kritériumok leírása

- Egy gyakori formalizmus: *temporális logikák*
- Állapotokra vonatkozó állítások
- Logikai operátorok ($\neg \wedge \vee$)
- Útvonalkvantorok
  - $\forall$ (néha A) - minden lehetséges esetben
  - $\exists$ (néha E) - legalább egyféle lefutásban
- Temporális operátorok
  - $\Box$ (néha G) – végig, állandóan
  - $\Diamond$ (néha F) - legalább egyszer

Példa

- Vezérlőrendszert fejlesztünk egy modellvasúthoz



DEMO UPPAAL 4.0.7

- Modell építése
  - Automaták aszinkron kompozíciója
  - (csatornákkal lehetne szinkronizálni)
- Kritériumok megfogalmazása
  - Leegyszerűsített temporális logika
- Kritériumok ellenőrzése
  - Kritérium be nem tartása esetén:
    - Ellenpélda vizsgálata
    - Modell kijavítása

DEMO Kritérium példák

- Validáció: vonatok száma megmarad
  - **A[]** vonatok_szama == foglaltsag[0] + foglaltsag[1] + …
- Vonat nem siklik ki
  - **A[]** not Thomas.kisiklott
- Nincs karambol
  - **A[]** forall(sz: szakasz) foglaltsag[sz] < 2
- Mindenhova elérhet a vonat
  - **E<>** Thomas.s1, stb.

UPPAAL specialitás

- *Időzített* automata (timed automaton)
  - Stopperszerű órák definiálhatóak
  - Állapotátmenetek indíthatják
  - Óraintervallum megszabható őrfeltételként
  - Óralimit megszabható állapot-invariánsként
  - Vannak azonnali állapotátmenetek is
- Vizsgálható: időzítések, real-time kényszerek
  - Elég gyorsan vált-e a szemafor pirosra, hogy a vonat mindig meg tudjon állni a következő sínszakasz előtt?
- Szorgalmi feladat

Formális módszerek

- Formális módszerek
  - Matematikailag szabatos eszközök
  - Mérnöki rendszerek, hardverek és szoftverek
    - Specifikációjára
    - Tervezésére, fejlesztésére
    - Verifikációjára, analízisére
- BMEVIMIM100 Formális módszerek
  - Elsőéves MSc tárgy
  - 3/0/0/f
  - 4 kredit

PROGRAMVERIFIKÁCIÓ

Dinamikus programverifikáció

- Mi a legfontosabb viselkedésmodell?
- Informatikusként: a **program**!
  - Szúrópróbaszerű analízis: szoftvertesztelés
  - Kimerítő dinamikus analízis
    - Hagyományos ellenőrzés: modell absztrahálása
    - Ellenőrizhető-e közvetlenül?
- NASA Java Pathfinder (JPF)
  - Különleges Java VM, bájtkódot futtat
  - Teljes állapottér-bejárásra képes
    - Szálak ütemezése
    - Random
    - Input, GUI események

Statikus programverifikáció

- „Futtatás”, állapotbejárás nélküli ellenőrzés
  - Típusellenőrzés, null, stb: legtöbb modern nyelvben
- „Design by contract”: programkód annotálása állításokkal (*Hoare logika*)
  - Prekondíció: aminek a művelet előtt teljesülnie kell
    - pl. lep(): Csak arról a szakaszról lépünk el, ahol voltunk
  - Posztkondíció: ami a művelet után garantált
    - pl. lep(): Be lesz jegyezve, hogy az új szakaszra ért a vonat
  - Invariáns: állandóan teljesülnie kell
    - pl. foglaltságok összege a vonatok száma
  - Ciklusinvariáns: ≅ indukciós feltevés
    - pl. minimumkiválasztásos rendezés: index előtti rész rendezett

Statikus programverifikáció

- „Design by contract” → **statikus** verifikálás
- Nyelvek
  - Eiffel
  - ADA → SPARK
  - Java + JML (Java Modeling Language)
  - .NET Code Contracts, C# → Spec#
- Verifikáció a megfelelő eszközökkel
  - pl. ESC/Java2 a JML-hez

Statikus programverifikáció

- Előny, ha utólag is bármikor ellenőrizhető
  - pl. Java bájtkód: type safety betöltéskor verifikálva
- Proof-carrying code: viszi magával a bizonyítást
- Microsoft Research: **Singularity** OS koncepció
  - Spec# → Sing# kommunikációs csatorna protokollokkal
  - Processzek csak deklarált csatornákon beszélhetnek
  - Telepítéskor statikusan verifikált programok
  - Minden program mehet egyetlen memóriatérbe
  - Context switch olcsóbb lesz, teljesítménynövekedés