Részletes tervek ellenőrzése

Majzik István

http://www.inf.mit.bme.hu/

Tartalomjegyzék

Részletes tervezés

- A részletes tervezés szerepe
  - Rendszerszintű részletes tervezés
    - Rendszerszintű algoritmusok: Komponensek együttműködése
    - Globális adatstruktúrák: Több modul által használtak
  - Komponens (modul) szintű részletes tervezés
    - Belső algoritmusok
    - Belső adatstruktúrák
- Módszerek jellegzetességei
  - Viselkedés leírása hangsúlyos: Állapotok, akciók, időbeliség, ...
  - Adatstruktúrák leírása: Absztrakt adatstruktúrák $\rightarrow$ implementáció
- Tervezési, modellezési nyelvek
  - Formális, félformális, strukturált módszerek
  - Implementáció-közeli „pszeudo-kód” is lehet
- Verifikációs szempontok
  - Megfelelőség a szoftverkövetelmény-specifikációnak
  - Megfelelőség az előző fejlesztési lépések döntéseinek

Ellenőrzési módszerek

- Követhetőség (követelmények fedettsége)
- Statikus ellenőrzés felülvizsgálattal
  - Ellenőrzőlisták, hibabecslés
    - Teljesség, ellentmondás-mentesség, megvalósíthatóság, tesztelhetőség
  - Vezérlési folyam elemzése
    - Strukturáltság
    - Határérték elemzés
  - Adatáramlás elemzése
    - Értékadás és felhasználás kapcsolata
  - Szimbolikus végrehajtás
- Dinamikus ellenőrzés
  - Szimuláció (modellek alapján)
  - Prototípus készítés és animáció
- Formális verifikáció

Eszköz példa: IAR visualSTATE Verificator

Verificator™

How do you test that your state machine design model and embedded application does not contain any of the following problematic properties?

- State, Local and System wide dead lock conditions Conflicting transitions between states
- Unreachable states, i.e. states that cannot be entered by any sequence of events from the environment
- Unused events or signals, i.e. stimuli to the system that is not acted upon
- Unused transitions, i.e. transitions that will never fire, regardless of the event sequence fed into the system
- Unused actions or assignments
- Unused variables, parameters and constants.

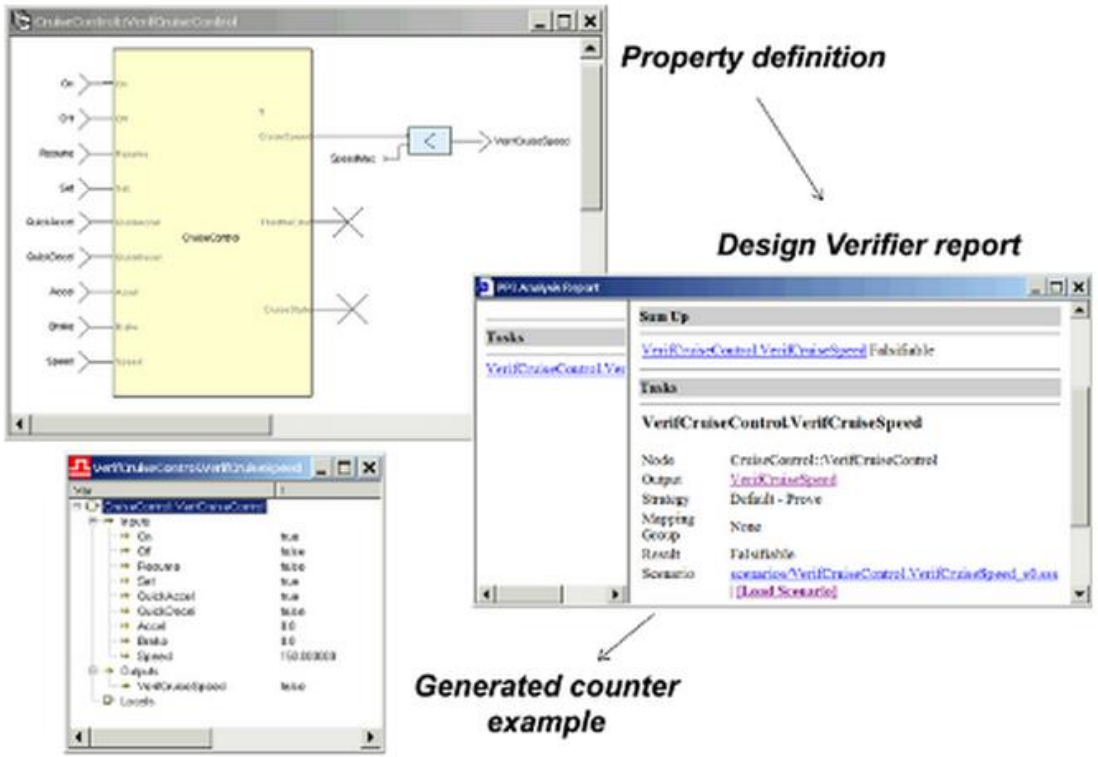
Each of these properties, if present in the application, is a potential sign of an inconsistent design or - even worse - a design with fatal logical gaps.

http://www.iar.com/en/Products/IAR-visualSTATE/Components/Verificator/

Eszköz példa: Esterel SCADE Suite

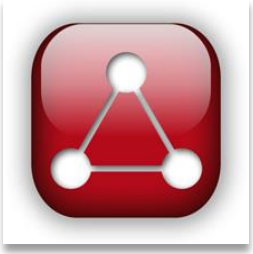
SCADE Suite Design Verifier (DV)

SCADE Suite Design Verifier is a formal verification assistant used to express and assess safety requirements. It provides a productive way to find defects early in the development process. Properties to be verified are defined with Scade itself; if a safety property is not obeyed, DV automatically produces a counter-example. DV can, for example, be used to verify that two different designs coming from the same requirements produce the same results.



http://www.esterel-technologies.com/products/scade-suite/verify/design-verifier/

Eszköz példa: Verum ASD:Suite



ASD Verification Technology

Advantage: Precision

Verum's patented ASD technology is at the core of the model-driven design approach. It offers a new way to formally verify the completeness and correctness of software designs using our own mathematical methods.

Until now, most software architects have been unable to employ Formal Methods techniques to verify product design and function. Either they were considered too difficult to learn or too expensive to use and maintain for common code bases.

With the introduction of ASD:Suite software design (CAD) platform, the focus moves from defect removal to defect prevention. Its formal verification engine gives software architects the power to:

- Verify that specifications are complete.
- Model the behaviours of all system components and interfaces.
- Produce defect-free code after verifying designs are complete and correct.
- Guarantee equivalence between specifications, designs, formal models and behaviour of generated code.
- Ensure mathematical integrity throughout the design process.

http://www.verum.com/Technology/analytical-verification.aspx

Tartalomjegyzék

A felülvizsgálat (review) módszere

- Fázisok (meghatározott folyamat)
  1. Tervezés: Dokumentumok, résztvevők, kritériumok
  2. Kick-off: Folyamat ismertetése, tervek szétosztása
  3. Előkészítés: Tervek átvizsgálása, problémák feltárása
  4. Megbeszélés: Problémák egyeztetése, rögzítése, javaslatok
  5. Átdolgozás: Javítások, módosítások
  6. Követés: Javítások ellenőrzése, metrikák képzése, kilépés
- Szerepek
  - Moderátor, szerző (tervező), jegyzőkönyvvezető, átvizsgálók
- Típusok (a felülvizsgálat célja szerint)
  - Átvizsgálás (walkthrough): A szerző által végzett bemutató, célja információgyűjtés és konszenzus az átvizsgálókkal
  - Technikai felülvizsgálat (technical review): A technikai megközelítés tekintetében törekszik közös álláspontra jutni
  - Vizsgálat (inspection): Dokumentált eljárás a hibák és kiindulási dokumentumokhoz képest történő eltérések felderítése érdekében

Tartalomjegyzék

Formális verifikáció

- Matematikai eszközök használata
  - Formális nyelv: Formális szintaxis és szemantika
    - Viselkedés leírás (terv, implementáció)
    - Tulajdonság leírás (követelmények, specifikáció)
  - Algoritmus (itt: verifikáció) a formális modellen
- A verifikáció módja
  - „Önmagában való” vizsgálat
    - Pl. tervek, modellek ellentmondás-mentessége
  - „Elvárt tulajdonságok” ellenőrzése
    - Pl. szoftverkövetelmény-specifikáció betartása
  - „Változások” vizsgálata
    - Pl. tervezői döntések hatása: funkció, tulajdonság megtartása
- Ideális formális verifikáció
  - Formális modell validálása (feltételezések ellenőrzése)
  - Verifikációs eszközök helyességének belátása

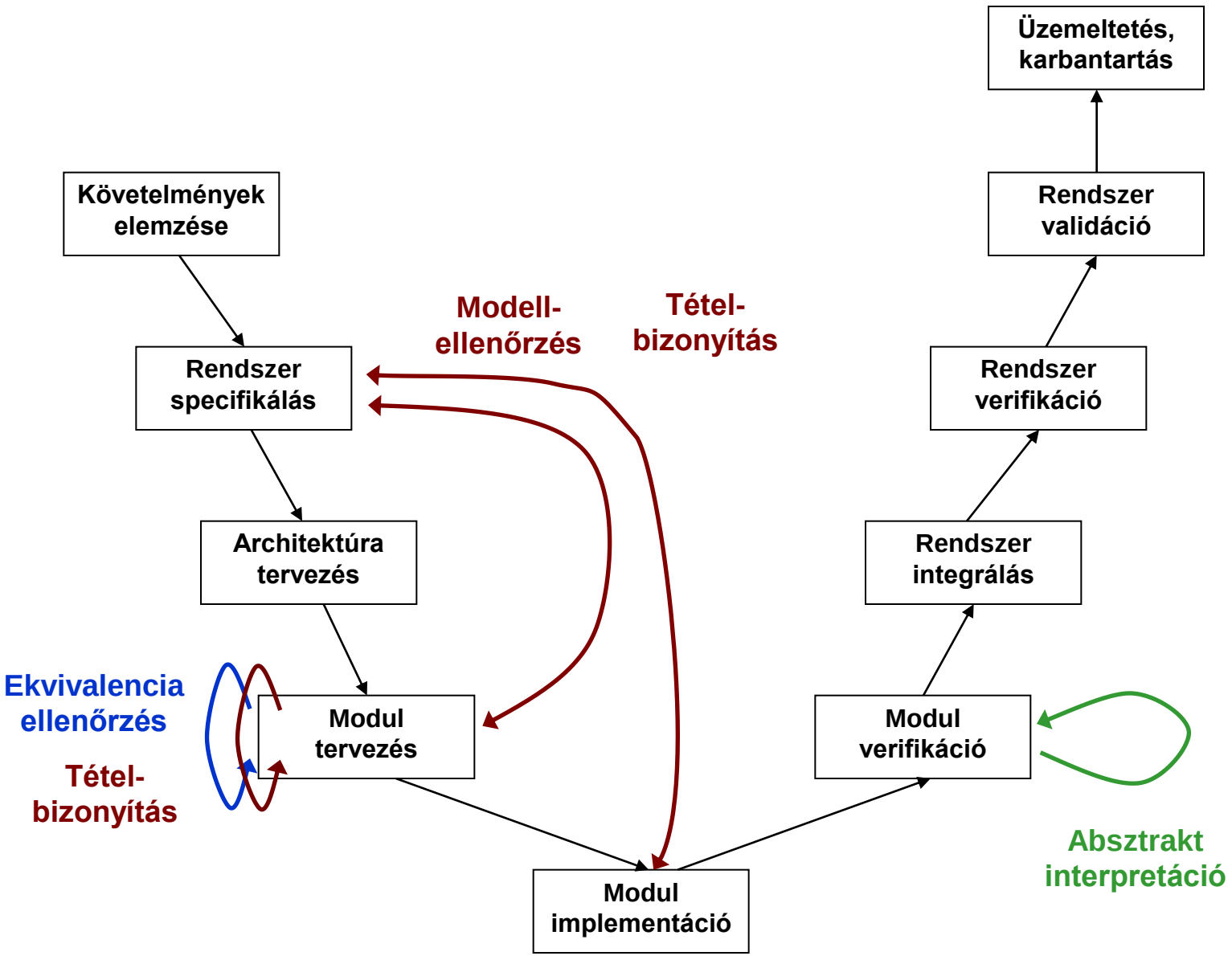
Tervek és modellek szerepe a formális verifikációban

- Viselkedés leírás lehetőségei
  - Alapszintű:
    - KS, LTS, KTS, automaták, Büchi automaták
  - Magasabb szintű:
    - Vezérlés orientált: Hierarchikus automata, Petri-háló
    - Adatfeldolgozás orientált: Adatfolyam háló
    - Kommunikáció orientált: Processz algebrák
  - Mérnöki modellek:
    - UML diagramok formális szemantikával
- Tulajdonság leírás lehetőségei
  - Alapszintű:
    - Elsőrendű logika, temporális logika, referencia automata
  - Magasabb szintű:
    - Message Sequence Charts, Live Sequence Charts, …

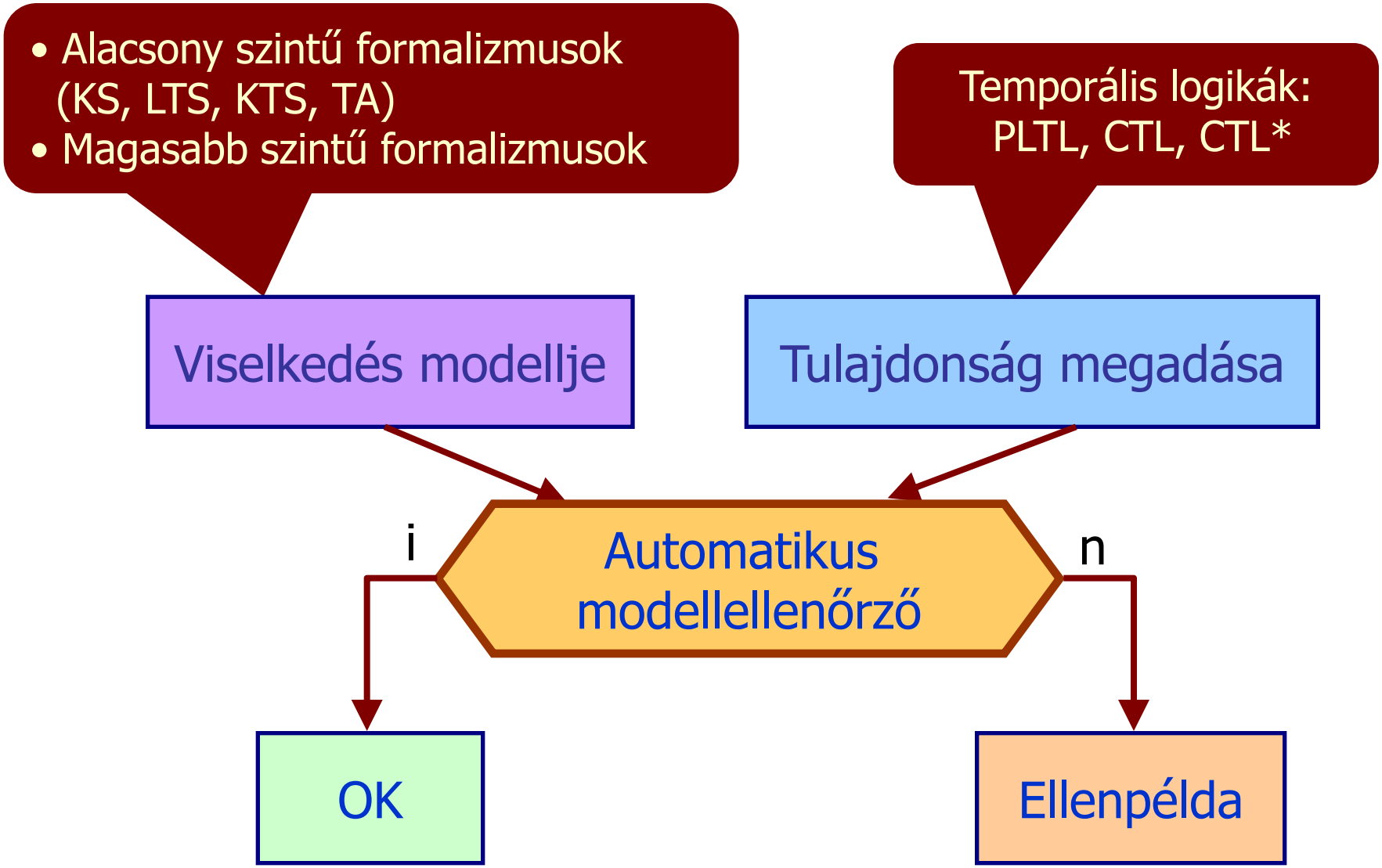
A legelterjedtebb formális verifikációs technikák

Modell / technika	**Viselkedés modell (alapszintű)**	**Tulajdonság leírás (alapszintű)**
Modellellenőrzés	Kripke struktúra	Temporális logika
Ekvivalencia ellenőrzés	Címkézett tranzíciós rendszer (LTS), automata	Referencia viselkedés: LTS, automata
Tételbizonyítás	Dedukciós rendszer (axiómák, levezetési szabályok)	Bizonyítandó tétel
Statikus analízis: absztrakt interpretáció	Kripke tranzíciós rendszer (programból absztrakcióval)	Assertion (elsőrendű logikai)

A formális verifikációs technikák szerepe



Modellellenőrzés



Modellellenőrzési technikák

- Háttér algoritmusok (ld. Formális módszerek)
  - Teljes (kimerítő) modellellenőrzés
    - Tabló alapú (kibontás a modell alapján)
    - Szemantikán alapuló „fixpont” algoritmus (állapottér bejárás)
    - Hatékony állapottér reprezentáció (ROBDD)
  - Korlátos modellellenőrzés
    - Az állapottérben adott mélységig ellenőriz
      (kezdőállapottól indulva adott végrehajtási útvonal hosszig)
    - Hatékony SAT technikákra visszavezethető
- Előnyök és hátrányok
  - ☺ Teljesen automatikus ellenőrzés
  - ☺ Ellenpélda generálás (hibakereséshez)
  - 😐 Vezérlés-orientált ellenőrzés
  - ☹ Állapottér robbanás (részben kezelhető)

Példa: UML állapottérképek modellellenőrzése

- Viselkedés-megtartó modelltranszformáció:
  Kiterjesztett UML → Időzített automata (UPPAAL)

  - Aktív objektumok (osztályok) leképezése automatákká
  - Eseménykezelés leképezése
    - Külső környezet modellezése: Eseményeket „generál”
    - Eseménysor, esemény ütemező, RTC lépések
  - Adatkezelés leképezése
    - Egyszerű adattípusok, korlátozott adattartományok
  - Konkurens és hierarchikus állapotmodell leképezése
    - Állapotkonfigurációk
  - Időkezelés (kiterjesztés)
    - Megfigyelő (clock observer) idő eseményeket generál

Tartalomjegyzék

Használat: Tervezői döntések ellenőrzése

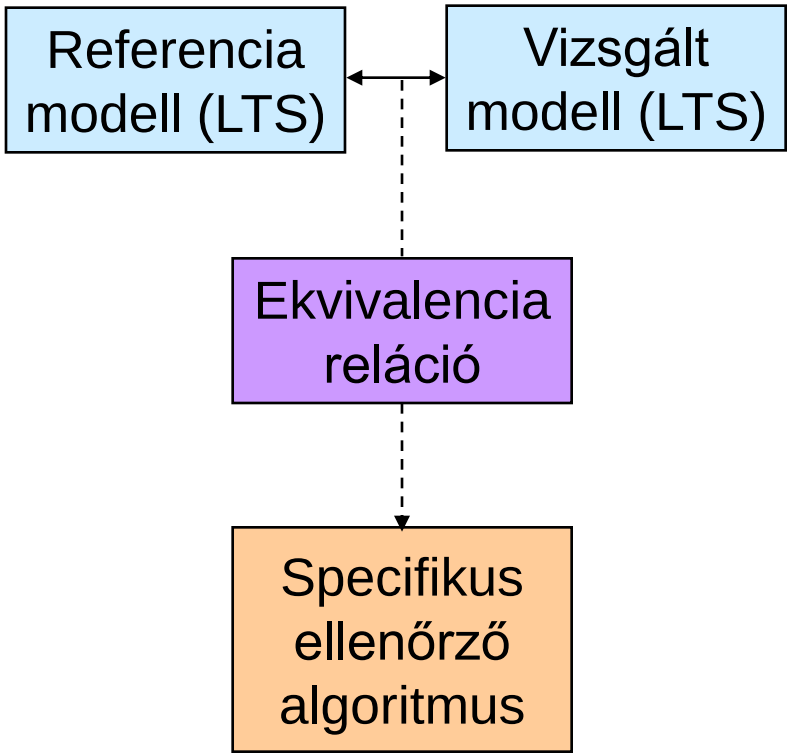
- Megfelelőség (ekvivalencia) modellek között
  - Referencia modell ↔ Vizsgált (módosított) modell
  - Specifikáció (absztrakt) ↔ Megvalósítás (konkrét)
  - Elvárt protokoll ↔ Nyújtott protokoll
  - Ideális rendszer ↔ Hibatűrő rendszer hibák mellett
- Finomítás (rendezés) modellek között:
  - Referencia viselkedés megtartása, meghatározott bővítésekkel
  - Nemdeterminizmus csökkentése

Használt matematikai relációk modellek között:

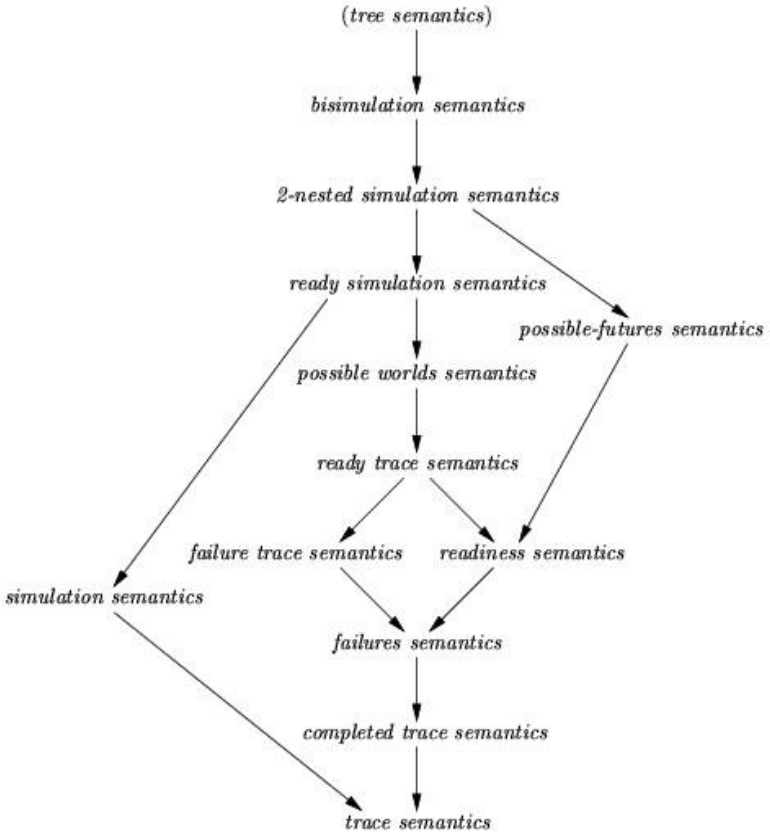
- Ekvivalencia: Reflexív, tranzitív, szimmetrikus
- Rendezés: Reflexív, tranzitív, antiszimmetrikus

A probléma formalizálása

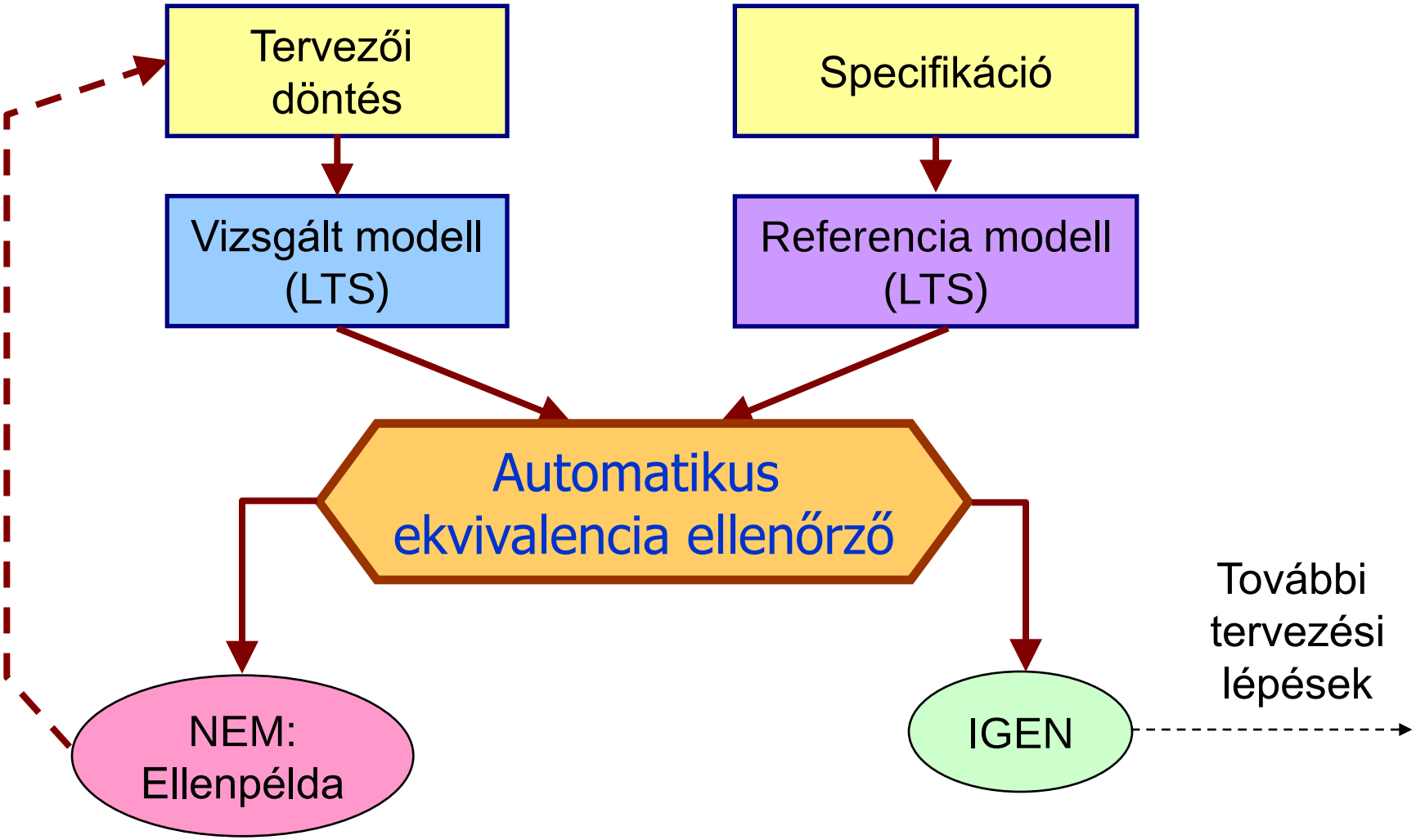
Ekvivalencia ellenőrzés:



Ekvivalencia relációk:



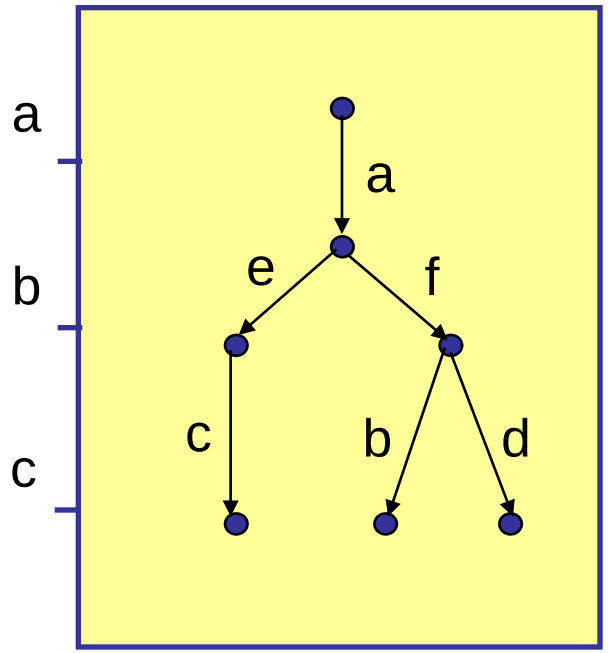
Ekvivalencia ellenőrzés



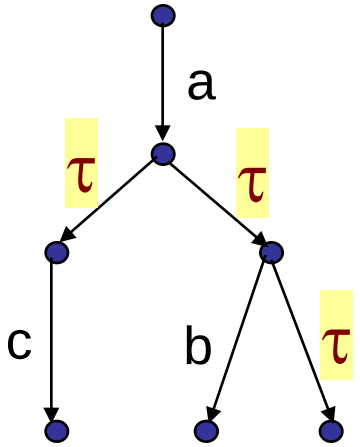
A komponensek „fekete doboz” modellje

- Megfigyelhető akciók
  - A vizsgált komponens (modul) interfészén megjelenő, a környezet számára érdekes („releváns”) viselkedés
    - Metódus hívása, metódushívás fogadása
    - Üzenet küldése, üzenet fogadása
- Nem megfigyelhető akciók (i, τ)
  - Az interfészen nem megjelenő, vagy a környezet számára nem érdekes („don’t care”) viselkedés
    - Belső működés (metódusok)
    - Figyelmen kívül hagyható hívások, üzenetek
- Nemdeterminizmus
  - A környezet szempontjából nem megfigyelhető belső működés is okozhatja (ezt eltakarja a fekete doboz modell)

Megfigyelhető viselkedés



Komponens
belső viselkedés



Megfigyelhető
viselkedés

Trace ekvivalencia: Jelölések

- Minta: Automaták ekvivalenciája

$$A_1 = A_2 \text{ ha } L(A_1) = L(A_2)$$

- LTS-ek esetén analógia:
  - Minden állapot „elfogadó állapot”
  - Nyelv: Minden akciószekvencia (trace)
- Jelölések:

$$\alpha = a_1 a_2 a_3 a_4 \ldots a_n \in Act^* \text{ véges akciószekvencia } (\varepsilon \text{ az üres})$$

$$s \xrightarrow{\alpha} s' \text{ ha } \exists s_0 s_1 \ldots s_n \text{ állapotsorozat ahol } s_0 = s,\ s_n = s',\ s_i \xrightarrow{a_{i+1}} s_{i+1}$$

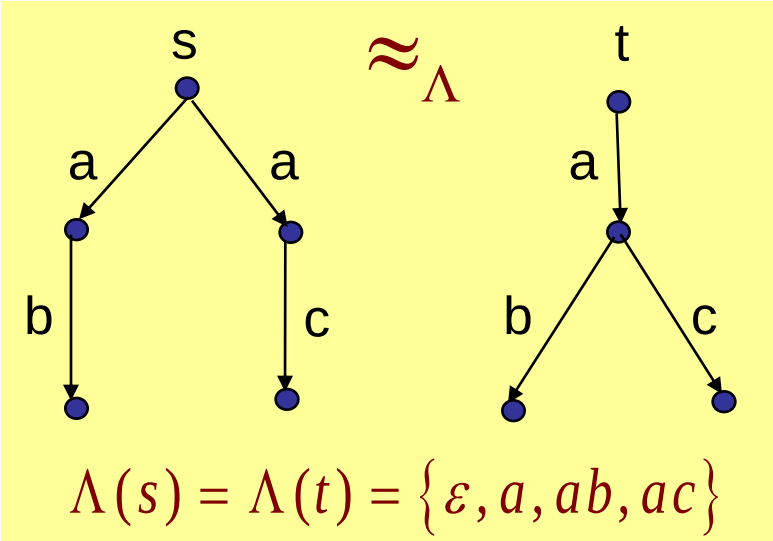
$$\Lambda(s) \text{ legyen } s \text{ trace-einek halmaza: } \Lambda(s) = \left\{ \alpha \mid \exists s' : s \xrightarrow{\alpha} s' \right\}$$

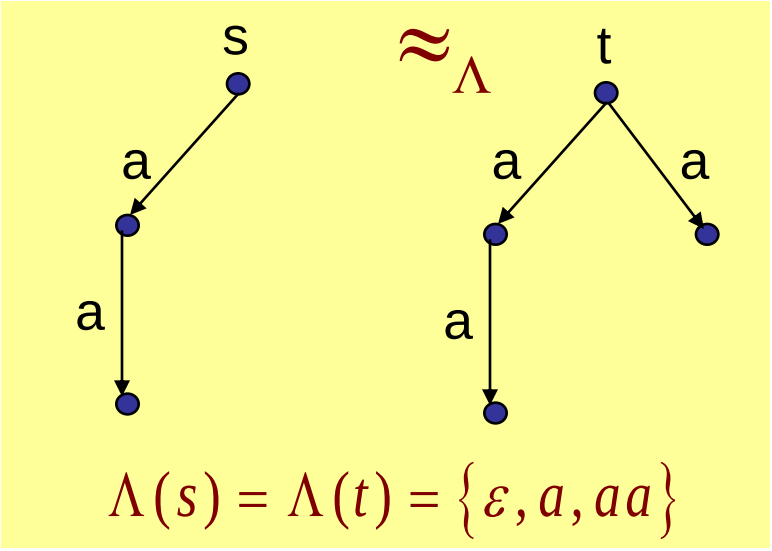
Trace ekvivalencia: Definíció és példák

- Legyen T_1 és T_2 két LTS, s_1 és s_2 kezdőállapottal
- Definíció:

$$T_1 \approx_\Lambda T_2 \quad \text{a.cs.a.} \quad \Lambda(s_1) = \Lambda(s_2)$$

- Példák:





- Problémák:
  - Ekvivalens LTS-ek eltérő viselkedése, pl. megállás szempontjából
  - Azonos trace-ek különböző állapotokon keresztül

Megfigyelési ekvivalencia (gyenge biszimuláció)

- Jellegzetességek
  - Lényege: Két megfigyelhető viselkedés ekvivalens, ha egymást szimulálni tudják:
    - Azonosak a megfigyelhető akciószekvenciák
    - Ekvivalens állapotokon keresztül figyelhetők meg
  - Nem érzékeny a hatás nélküli belső átmenetekre (azok megjelenésére, számára)
- Jelölések:

$\alpha \in Act^*$ véges akciószekvencia (ε az üres)

$\hat{\alpha} \in (Act - \tau)^*$ megfigyelhető akciószekvencia τ törlésével

ha $\alpha = \tau$ akkor $\hat{\alpha} = \varepsilon$

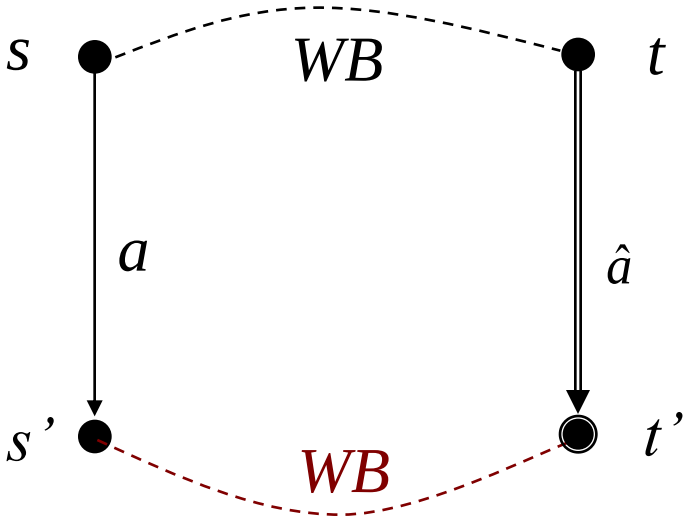
$s \overset{\beta}{\Rightarrow} s'$ ha $\exists \alpha : s \overset{\alpha}{\rightarrow} s'$ és $\beta = \hat{\alpha}$

Gyenge biszimuláció: Definíció

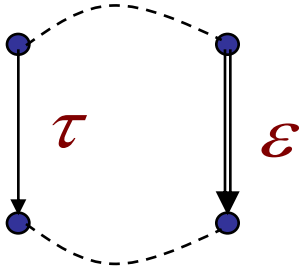
- **Definíció:**

  $WB \subseteq S \times S$ gyenge biszimuláció, ha minden $(s,t) \in WB$ és bármely $a \in Act,\ s',t' \in S$ esetén fennáll:

  - ha $s \xrightarrow{a} s'$ akkor $\exists t' : t \overset{\hat{a}}{\Rightarrow} t'$ és $(s',t') \in WB$
  - ha $t \xrightarrow{a} t'$ akkor $\exists s' : s \overset{\hat{a}}{\Rightarrow} s'$ és $(s',t') \in WB$



Extrém eset:

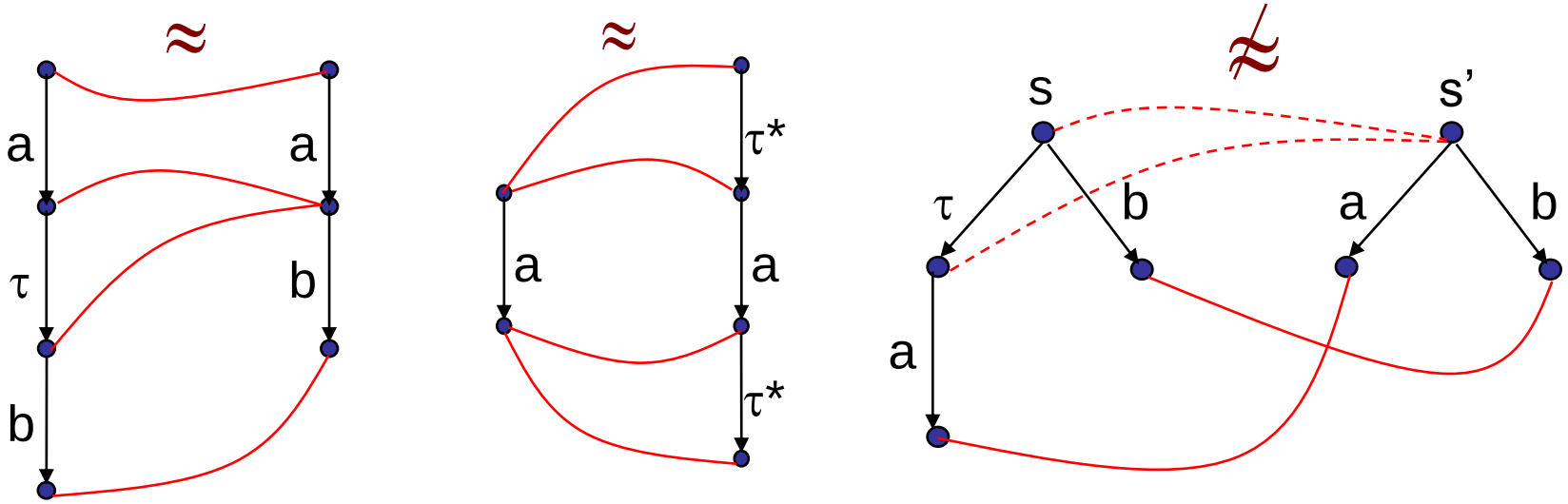


Megfigyelési ekvivalencia: Definíció és példák

- Megfigyelési ekvivalencia
  = gyenge biszimuláció ekvivalencia (observation equivalence)

$$T_1 \approx T_2 \quad \text{a.cs.a.} \quad s_1 \approx s_2 \quad \text{azaz} \quad \exists WB : (s_1, s_2) \in WB$$

- Példák



Ekvivalencia relációk számítási módszere

Partíció finomítás:

1. Kezdetben minden állapotpár eleme a relációnak

   Egy partíciót (ekvivalencia osztályt) képeznek

2. Minden állapotpárra:

   Ha az egyikből indulva van olyan átmenet, amit a másik nem tud a definíció szerint szimulálni, akkor

   – Az adott állapotpár kizárása a relációból (nem ekvivalensek)

   – Következmények végigvezetése a bejövő átmenetek végén lévő állapotokra

     • Nem ekvivalensek, ha nem ekvivalens állapotokba kerülnek

3. Ha már nincs változás (fixpontig jutunk): Végleges ekvivalencia osztályok adódtak

   Ha a kezdőállapotok azonos ekvivalencia osztályban vannak, akkor az LTS-ek ekvivalensek

Egy rendezési reláció: Lehetséges viselkedés szerint

- Jelölések:

$\beta \in (Act - \tau)^*$ megfigyelhető akciószekvencia τ törlésével

$s \overset{\beta}{\Rightarrow} s'$ ha $\exists \alpha \in Act^*: s \overset{\alpha}{\rightarrow} s'$ és $\beta = \hat{\alpha}$

$\Delta(s)$ az s-ből induló megfigyelhető akciószekvenciák halmaza:

$$\Delta(s) = \left\{ \beta \mid \exists s': s \overset{\beta}{\Rightarrow} s' \right\}$$

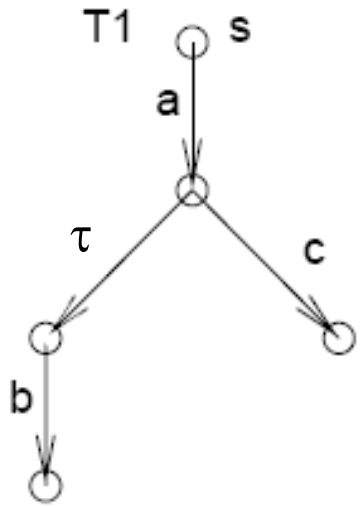
- Lehetséges viselkedés szerinti rendezés T_1 és T_2 LTS-ek esetén:

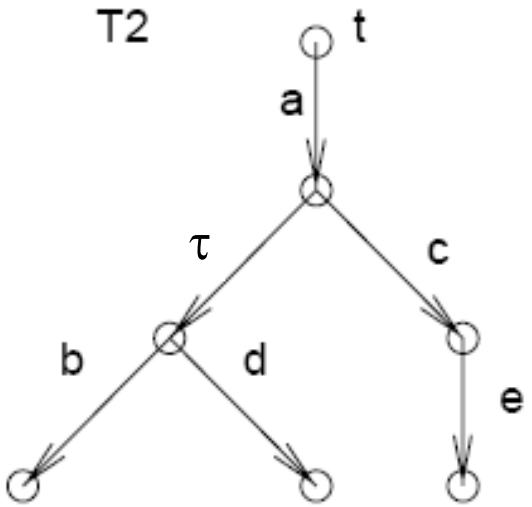
$T_1 \leq_\Delta T_2$ a.cs.a. $\Delta(s_1) \subseteq \Delta(s_2)$

itt T_2-ben több a megfigyelhető akciószekvencia

Lehetséges viselkedés szerinti rendezés: Példa

- Eredeti viselkedés bővítése:





$$\Delta(s) = \{a, ab, ac\}$$

$$\Delta(t) = \{a, ab, ac, ad, ace\}$$

Lehetséges viselkedés szerinti rendezés és a tesztelés

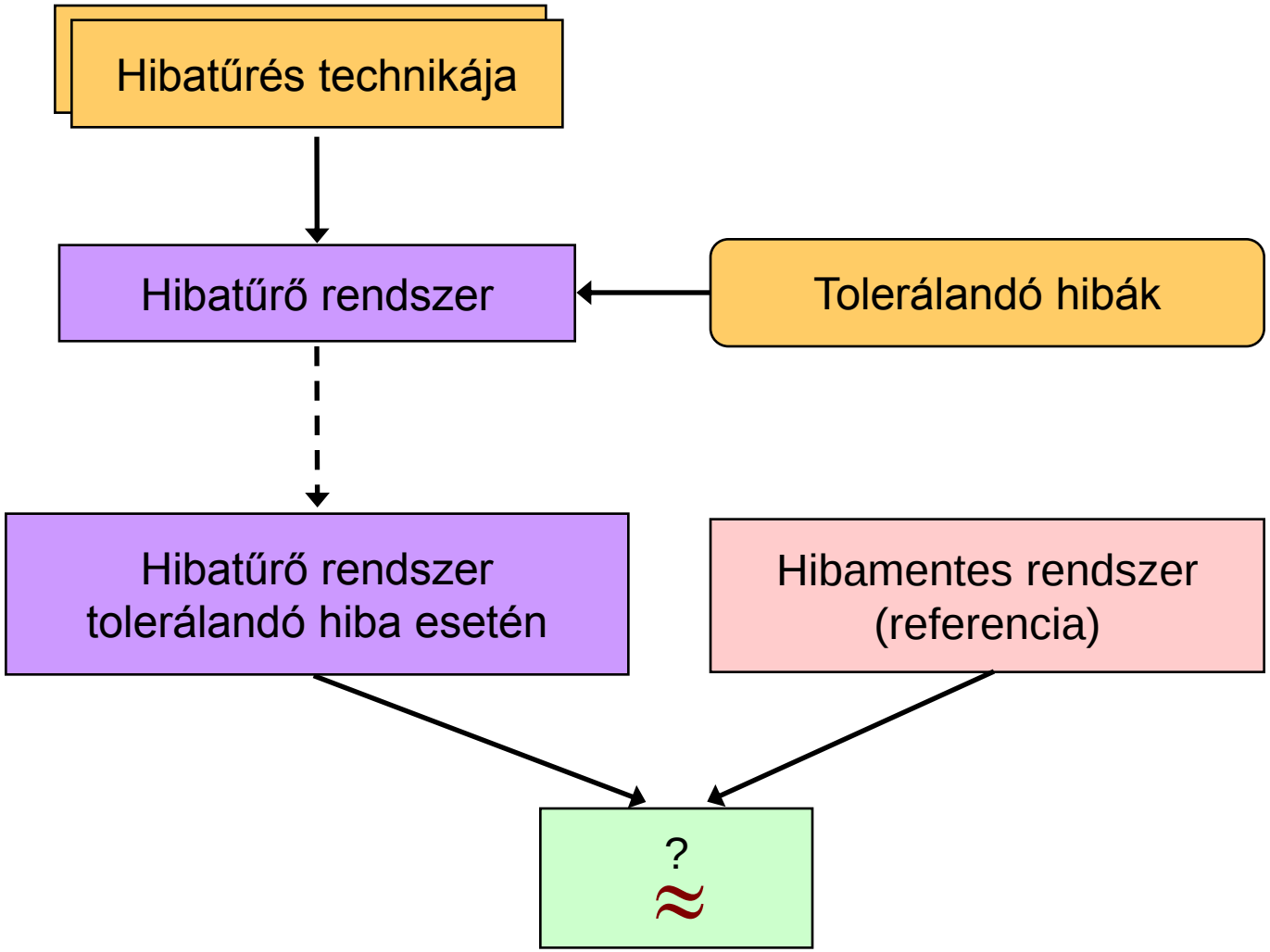
- Kapcsolat a teszteléssel: $T_1 \leq_\Delta T_2$ esetén:
  - Minden teszt, ami T_1 esetén sikeres lehet, T_2 esetén is sikeres lehet,
    de belső akciók miatt sikertelen is lehet (nemdeterminizmus)
  - Másképp: T_1 lehetséges sikeres tesztjei T_2 lehetséges sikeres tesztjei között vannak
- Jellegzetességek:
  - Olyan finomítást definiál, amelynek során nem „veszik el” lehetséges megfigyelhető viselkedés

Egy másik rendezési reláció: Szükséges viselkedés

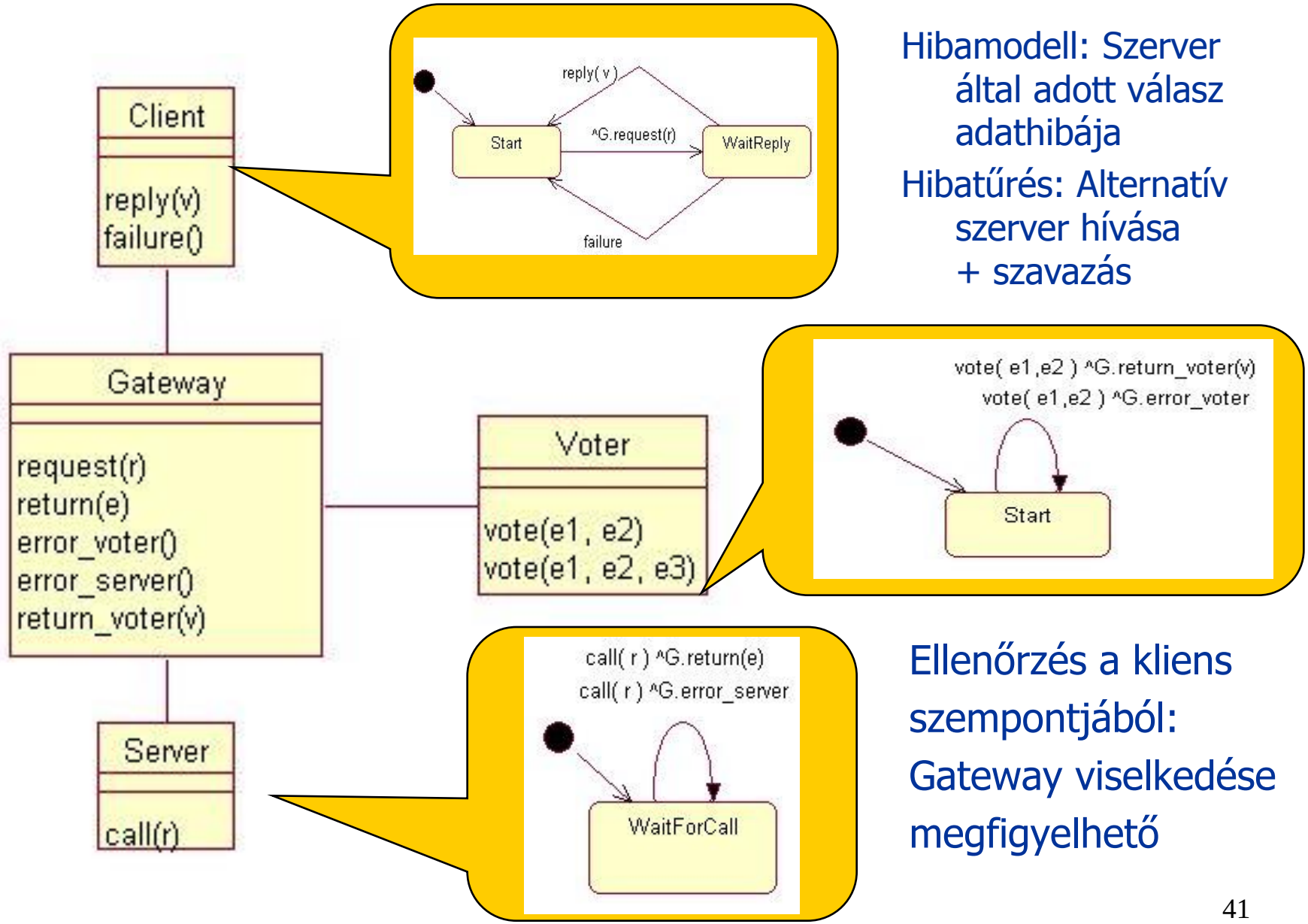
  - Olyan finomítás, ami a mindig sikeres tesztekre („kötelező” viselkedésre) ad megkötést

Tartalomjegyzék

Mintapélda: Hibatűrés verifikációja



Rendszerarchitektúra



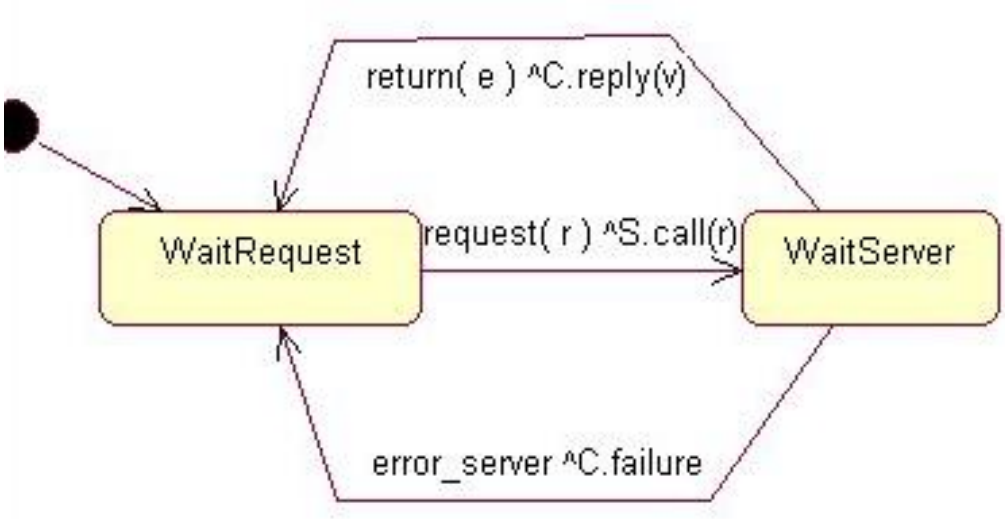
Hibamodell: Szerver által adott válasz adathibája

Hibatűrés: Alternatív szerver hívása + szavazás

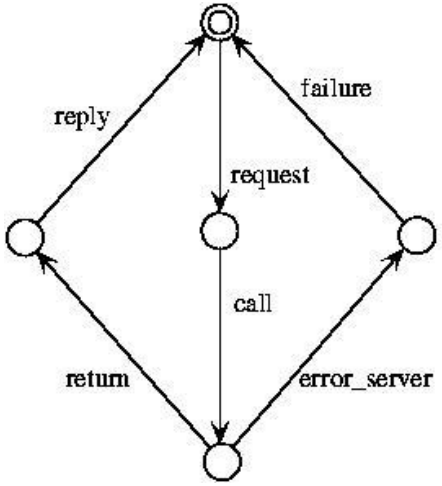
Ellenőrzés a kliens szempontjából: Gateway viselkedése megfigyelhető

A Gateway komponens hibamentes esetben

- Állapotdiagram:

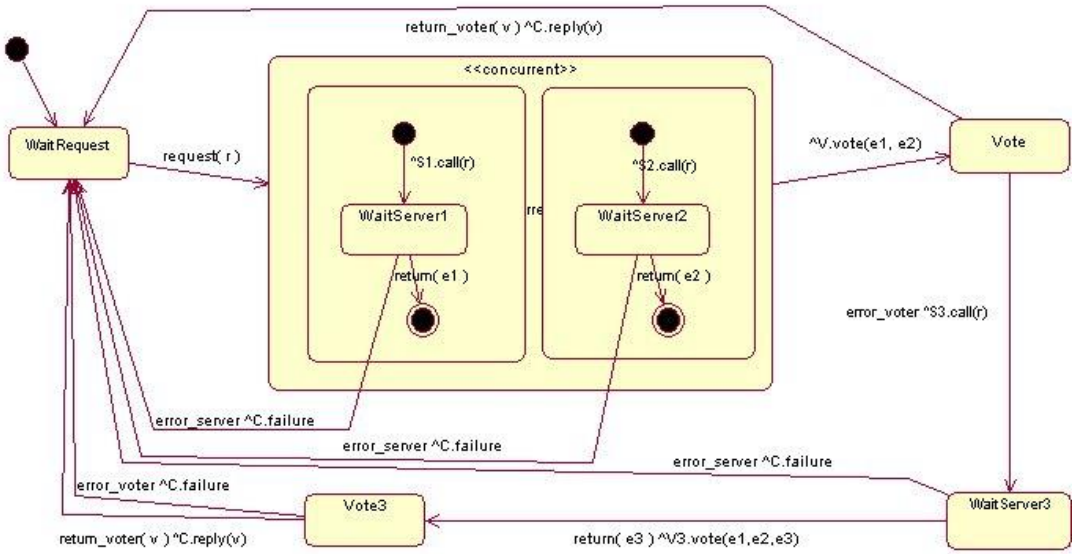


- LTS:

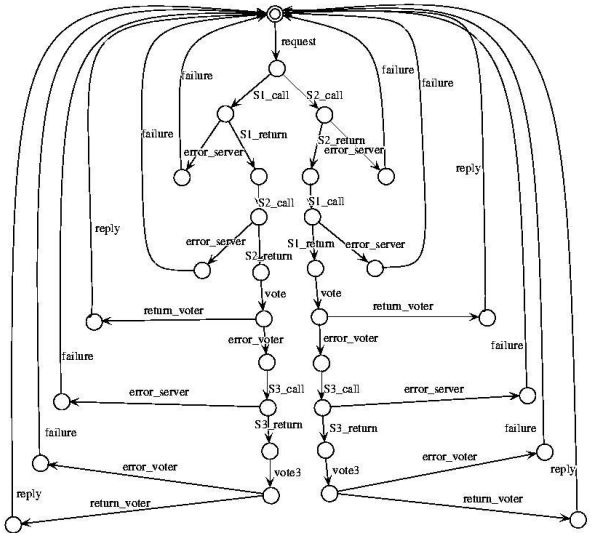


A Gateway komponens hibatűrő esetben

- Állapotdiagram:

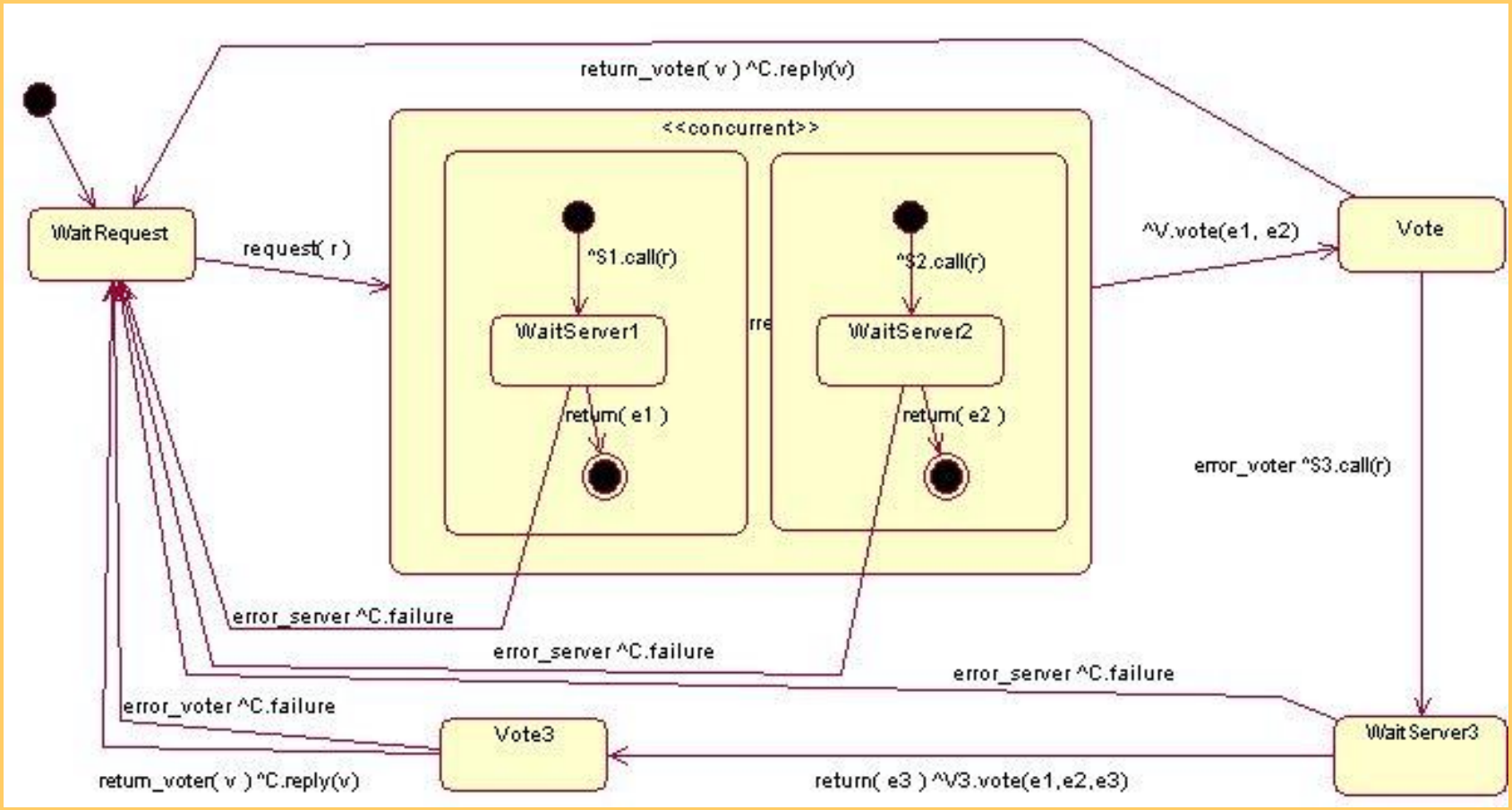


- LTS:



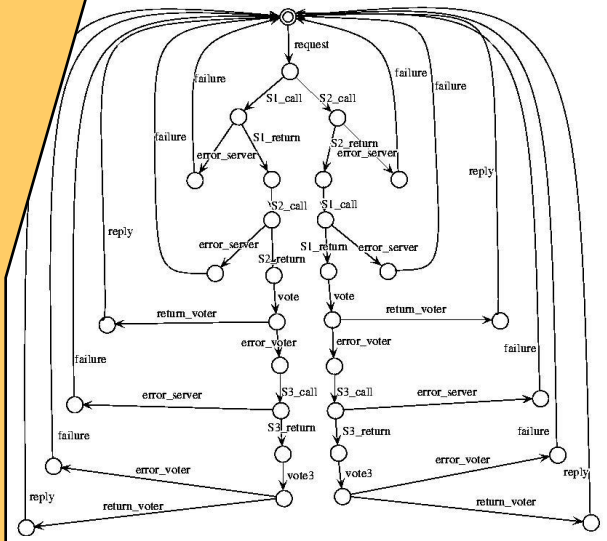
A Gateway komponens hibatűrő esetben

- Állapotdiagram:
- LTS:

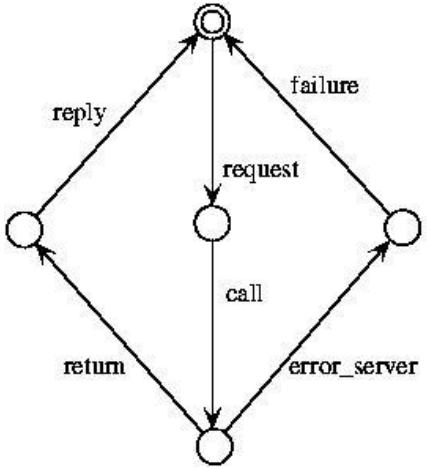


A Gateway komponens hibatűrő esetben

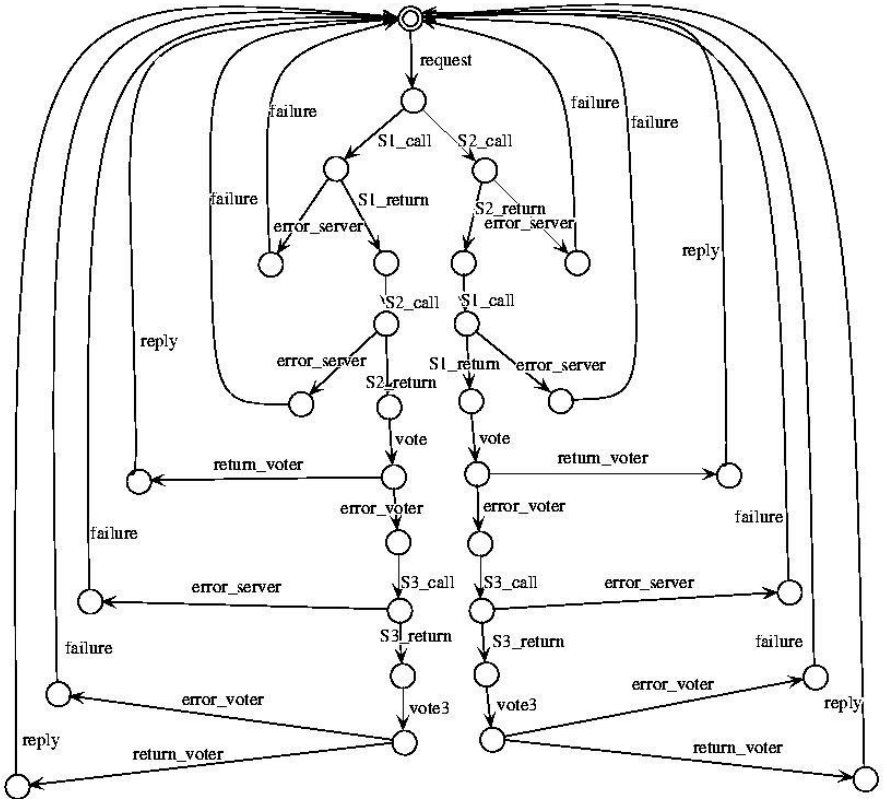
LTS:



Viselkedési ekvivalencia igazolása



≈



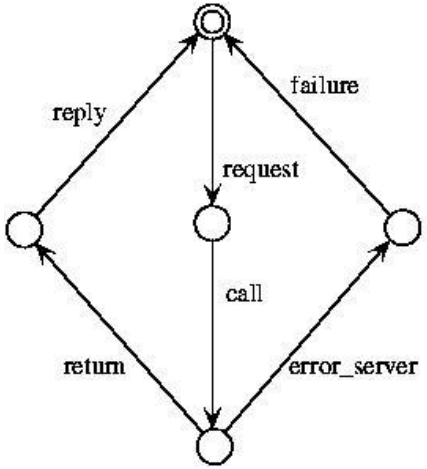
Gateway referencia viselkedés

Mit tudunk az ekvivalencia alapján kijelenteni?

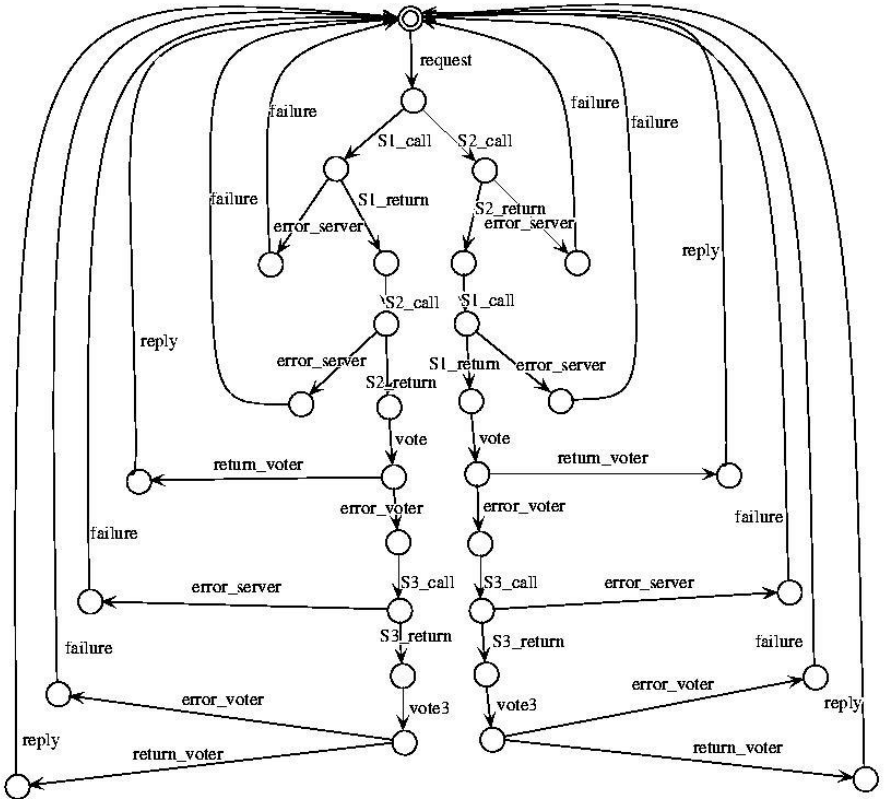
Hibatűrő Gateway teljes viselkedése;

Minden olyan akció τ lesz,
ami nincs a referencia viselkedésben!

Viselkedési ekvivalencia igazolása



≈



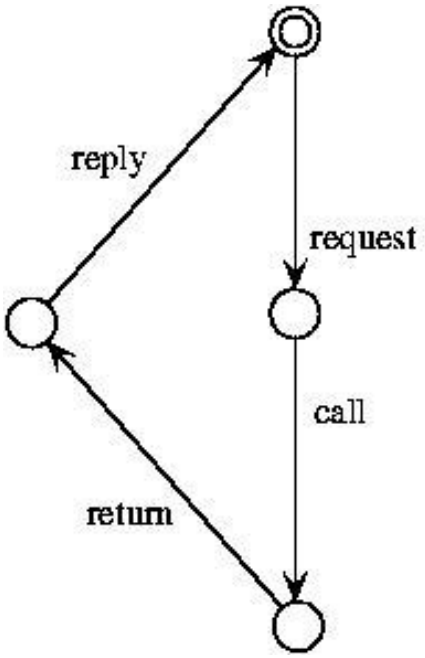
Gateway referencia viselkedés

Igazolható, hogy a kliens számára hibamentes esetben transzparens a hibatűrő mechanizmus működése.

Hibatűrő Gateway teljes viselkedése;

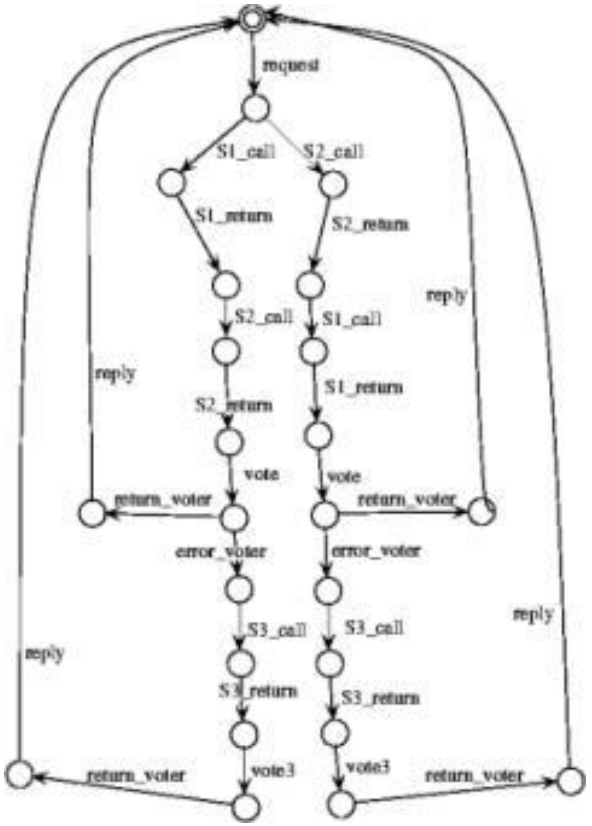
Minden olyan akció τ lesz, ami nincs a referencia viselkedésben!

Hibatűrés igazolása az első szerver adathibája esetén



Hibamentes
Gateway

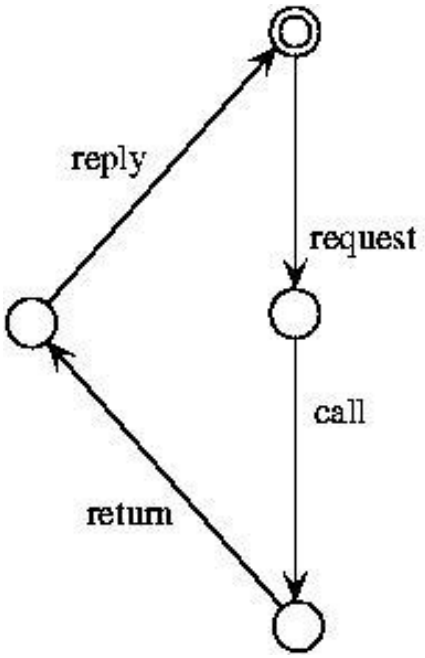
≈



Mit tudunk az ekvivalencia alapján kijelenteni?

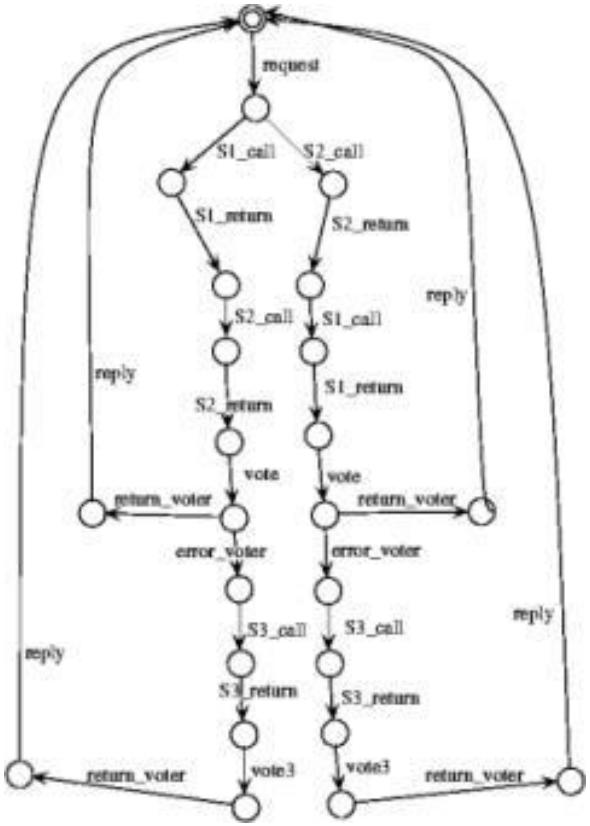
Hibatűrő Gateway a hiba esetén;
Minden olyan akció τ,
ami nincs a referencia viselkedésben

Hibatűrés igazolása az első szerver adathibája esetén



Hibamentes Gateway

≈



Az adott hiba esetén a kliens szempontjából megvalósul a hibatűrés.

Hibatűrő Gateway a hiba esetén;
Minden olyan akció τ,
ami nincs a referencia viselkedésben

Miről volt szó?

- Áttekintés
  - Milyen szerepe van a részletes terveknek?
  - Milyen ellenőrzési módszerek vannak?
- Statikus ellenőrzés felülvizsgálattal
- Formális verifikáció modellellenőrzéssel
  - Módszerek áttekintése
  - Modellellenőrzés mérnöki modellek alapján
- Formális verifikáció ekvivalencia ellenőrzéssel
  - Trace ekvivalencia
  - Megfigyelési ekvivalencia (gyenge biszimuláció)
  - Lehetséges viselkedés szerinti rendezés
  - Mintapélda: Hibatűrés verifikációja