1 TEMPORAALILOGIIKKA

- Pelkästään saavutettavuusgraafissa navigoimalla on hankalaa vakuuttua jonkin monimutkaisen ominaisuuden paikkansapitävyydestä.
- Tarvitaan kieli jolla ominaisuuksia ilmaistaan, ja jolle on määritelty algoritmit, jotka tarkastavat ko. ominaisuudet.

Johdanto

- Turvallisuusominaisuuksia:
  "Mitään pahaa ei tapahdu"
  - lukkiutumattomuus (deadlock-freedom)
  - toisensa poissulkevuus (mutual exclusion)
  - tietoturvallisuus,
  - yl. järjestelmä ei koskaan mene tietyyn tilaan
- Elävyysominaisuuksia:
  "Jotain hyvää tapahtuu joskus"
  - reiluus
  - lähetetty viesti menee perille
  - jokainen palvelupyyntö suoritetaan

Johdanto

- Muita:
  - P tapahtuu äärettömän usein
  - P:n jälkeen Q tapahtuu n:n askeleen/sekunnin jälkeen
  - joka tilassa pätee C
- Miten kuvata ominaisuuksia yhteisellä kielellä?
- Yhteistä:
  - *joskus* tulevaisuudessa
  - *aina* tulevaisuudessa
- Kvantifioidaan siis *ajan* yli.

Johdanto

- Ajan rakenne:
  1. lineaarinen?
  2. haarautuva?
  3. diskreetti?
  4. jatkuva?
  5. kuuluko nykyhetki myös tulevaisuuteen?
- Puhutaanko tiloista vai tapahtumista?

Johdanto

- Valinnat:
  - Ajan rakenne diskreetti, koska tilasiirtymät ovat diskreettejä, ja järjestelmässä on korkeintaan numeroituva määrä tiloja.
  - Nykyhetki kuuluu tulevaisuuteen.
  - Lineaarisen ja haarautuvan ajan välillä on vaikeampi valita.
  - Osoittautuu, että on luontevaa puhua tiloista tapahtumien sijaan.
  - Saadaan kaksi logiikkaa:
    - LTL (linear temporal logic): tarkastellaan saavutettavuusgraafia joukkona *äärettömiä* transitiojonoja.
    - CTL (Computational tree logic): tarkastellaan saavutettavuusgraafia *äärettömänä* puuna.

LTL

- Atomiset kaavat: $p \in \Phi$ jos p on paikan nimi
- Kaavat $Fma(\Phi)$:
  - Atomiset kaavat: $p \in \Phi \Rightarrow p \in Fma(\Phi)$
  - Kaavat: $A \in Fma(\Phi)$ missä $A ::= p \mid \bot \mid A_1 \Rightarrow A_2 \mid \Box A$
- $\Box A$ tarkoittaa *aina tulevaisuudessa* A.
- Muut konnektorit voidaan määritellä:

$$\begin{aligned} \neg A &\quad \text{on} \quad A \Rightarrow \bot \\ A_1 \vee A_2 &\quad \text{on} \quad (\neg A_1) \Rightarrow A_2 \\ A_1 \wedge A_2 &\quad \text{on} \quad \neg(A_1 \Rightarrow \neg A_2) \\ \Diamond A &\quad \text{on} \quad \neg \Box \neg A \\ \top &\quad \text{on} \quad \neg \bot \end{aligned}$$

LTL

- Esimerkkejä:
  - Viesti menee aina perille:

$$\Box(Sender(\langle 0, rts\rangle) \Rightarrow \Diamond Sender(\langle 1, rts\rangle)) \quad (1)$$

  - Filosofit 1 ja 2 eivät voi syödä samanaikaisesti:

$$\Box\neg(E(\langle ph_1\rangle) \wedge E(\langle ph_2\rangle))$$

  - q pätee vain äärellisen monessa tilassa:

$$\Diamond\,\Box\,\neg q$$

  - q pätee äärettömän monessa tilassa:

$$\Box\Diamond q$$

  - Turvallisuusominaisuus:

$$\Box A$$

  - Elävyysominaisuus:

$$\Diamond A$$

Semantiikka

- Semantiikka:
  - LTL-kaavojen semantiikka on määritelty (äärettömien) sekvenssien suhteen.
  - Pari $F = (S, R)$ muodostaa *kehyksen*, missä S on ei-tyhjä (numeroituva) joukko (tilojen joukko), ja $R \subseteq S \times S$ (saavutettavuusrelaatio).
  - saavutettavuusrelaatiolle pätee: $\forall s \exists! t : (sRt)$, ts. jokaisella tilalla on vain yksi seuraaja.
  - Relaation R refleksiivistä ja transitivista sulkeumaa merkitään R^*.
  - Φ-*malli* on kolmikko $M = (S, R, V)$, missä $V : \Phi \to 2^S$.
  - $V(p)$ on niiden tilojen joukko, joissa atominen kaava p pätee.

SEMANTIIKKA

- Kaava A pätee mallissa M pisteessä s, jota merkitään

$$M \models_s A,$$

määritellään induktiivisesti:

$$\begin{aligned}
M \models_s p \quad &\text{joss} \quad s \in V(p) \\
M \not\models_s \bot \quad & \\
M \models_s A_1 \Rightarrow A_2 \quad &\text{joss} \quad M \models_s A_1 \text{ implikoi M} \models_\text{s} \text{A}_2 \\
M \models_s \Box A \quad &\text{joss} \quad \forall t \in S, sR^*t \text{ implikoi M} \models_\text{t} \text{A}
\end{aligned}$$

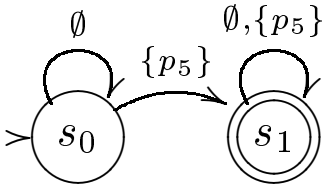
- Kaava A pätee mallissa M: $M \models A$ joss $M \models_{s_0} A$.

Semantiikka

- Saavutettavuusgraafin alkumerkinnästä alkavat äärettömät transitiosekvenssit generoivat LTL-malleja.
- Mallin tilat ovat merkintöjä, ja $m_i R m_k$ joss $m_i[t > m_k$.
- Merkitään verkon saavutettavuusgraafin SG_N generoimien mallien joukkoa $\mathcal{M}(SG_N)$.
- Kaava A pätee verkossa N joss $\forall M \in \mathcal{M}(SG_N) M \models A$
- Koska saavutettavuusgraafi jossa on silmukoita generoi ylinumeroituvan määrän LTL-malleja ei tämä ole mielekäs tapa tarkastaa päteekö jokin ominaisuus verkossa.
- Seuraavassa kuvataan pääpiirteissä menetelmä jolla kaavan pätevyys voidaan tarkastaa (kts. viite (2)).

Mallintarkastus

- Koska tilojen joukko voidaan numeroida, voidaan malli myös tulkita kuvauksena $\mathbf{N} \times \mathbf{\Phi} \to \{\mathbf{T}, \mathbf{F}\}$, eli kuvauksena $\mathbf{N} \to (\mathbf{\Phi} \to \{\mathbf{T}, \mathbf{F}\})$.
- Kuvaus $\mathbf{N} \to (\mathbf{\Phi} \to \{\mathbf{T}, \mathbf{F}\})$ voidaan nähdä äärettömänä merkkijonona jonka aakkosto on $2^{\Phi} \cong (\Phi \to \{\mathbf{T}, \mathbf{F}\})$.
- LTL-kaava A siis määrittelee äärettömien sekvenssien kielen: niiden mallien joukko joissa A pätee.
- Esim. $\Diamond p_5$ pätee kaikille sekvensseille joissa p_5 pätee joskus, eli kaava määrittelee kielen jonka aakkosto $\Sigma = \{\emptyset, \{p_5\}\}$ ja jonka sanoissa p_5 esiintyy ainakin kerran.
- Kielen sanoja voidaan tunnistaa automaatin avulla:



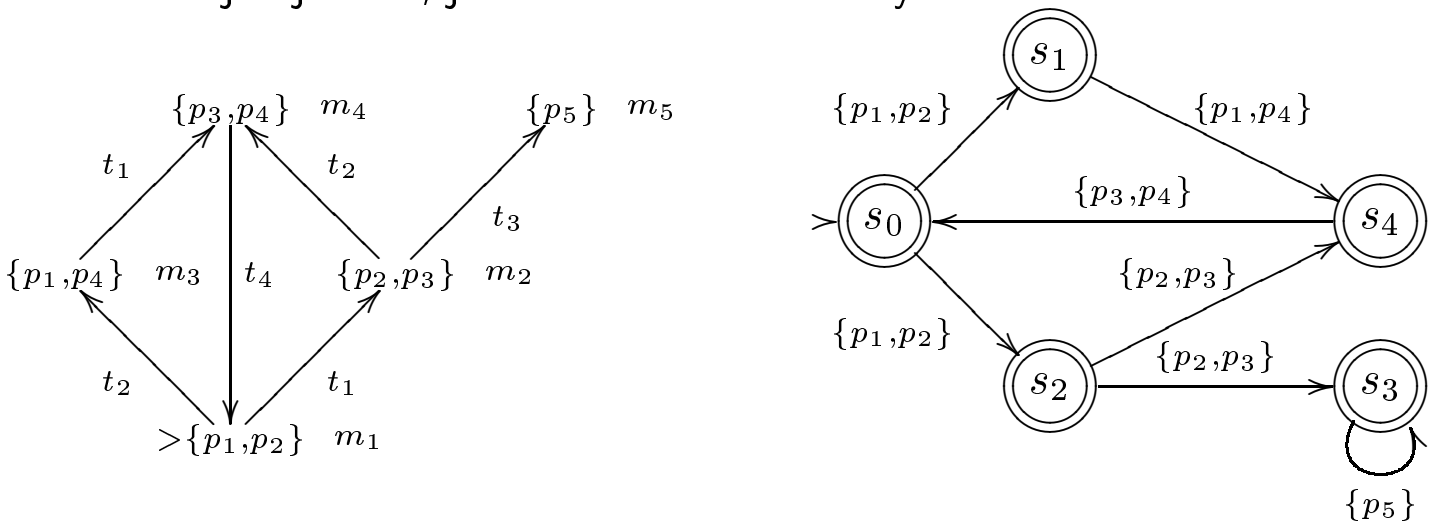
missä s_0 on *alkutila*, ja s_1 on *hyväksyvä tila*

Mallintarkastus

- Koska sanat ovat äärettömiä sekvenssejä on automatin hyväksymisehto seuraava: Automaatti Q hyväksyy äärettömän sanan $w = a_0, a_1, \ldots$, joss on olemassa ääretön sekvenssi $\sigma = s_0, s_1, \ldots$ s.e. s_0 on automaatin alkutila, jokainen s_i saadaan tilasta s_{i-1} ja symbolista a_{i-1} tilasiirtymäfunktion avulla, ja ääretön määrä kokonaislukuja $i \in \mathbf{N}$ se. $s_i = s$, missä s on hyväksyvä tila.
- Tällaista automaattia kutsutaan *Büchi-automatiksi*.
- Jokaiselle LTL-kaavalle A voidaan rakentaa Büchi-automatti $B(A)$ joka hyväksyy vain ne äärettömän sanat jotka totetuttavat kaavan.

Mallintarkastus

- Vastaavasti saavutettavuusgraafista SG voidaan ajatella Büchi-automaatiksi $B(SG)$, jossa jokainen tila on hyväksymistila ja jossa tilasiirtymän nimi on niiden atomikaavojen joukko, jotka ovat tosia tilasiirtymän lähtötilassa.



- Vastaus kysymykseen päteekö kaava A verkossa N saadaan laskemalla automaattien $B(\neg A)$ ja $B(SG_N)$ leikkaus- (tulo-) automaatti. Jos leikkausautomaatti hyväksyy vain tyhjän kielen kaava A pätee.

PROD

- PROD:ssa LTL-kaavojen verifiointi tapahtuu suorittamalla mallintarkastusta lennosta testerin avulla.
- PROD tarkistaa nyt saavutettavuusgraafia generoidessa kaavan paikkaansapitävyyden.
- Jos kaava ei päde, saadaan vastauksena vastaesimerkki. Jos kaava pätee, ei PROD vastaa mitään.

Laajennuksia

- Usein on kätevää voida ilmaista "ennenkuin" tai "kunnes" aikamääreitä.
- Logiikan kieli voidaan laajentaa *until*-konnektiivilla:
- $M \models_s A\mathcal{U}B$ joss on olemassa t, s.e. sR^*t ja $M \models_t B$ ja $M \models_u A$ kaikille u s.e. sR^*u ja uR^*t.
- Vaatimus, että vastaus v ei saavu ennen kysymyksen k lähettämistä voidaan nyt ilmaista:

$$\Diamond v \Rightarrow (\neg v)\mathcal{U}k$$

- Tilaa jossa q pätee pitää edeltää tila jossa p pätee:

$$\neg q\mathcal{U}p \vee \Box p$$

- Until-konnektiivi laajentaa logiikan ilmaisuvoimaa sillä:

$$\begin{aligned} \Diamond A &\equiv \top\mathcal{U}A \\ \Box A &\equiv \neg(\top\mathcal{U}\neg A) \end{aligned}$$

Laajennuksia

- Usein määritellään myös "seuraavaa tilaa" kuvaava operaattori *next*:

$$M \models_i \bigcirc A \text{ joss } \mathrm{M} \models_{\mathrm{j}} \mathrm{A} \text{ missiRj.}$$

- $\bigcirc$ operaattorille pätee:

$$\begin{aligned} \Box A &\equiv (A \wedge \bigcirc \Box A) \\ \Diamond A &\equiv (A \vee \bigcirc \Diamond A) \\ A \mathcal{U} B &\equiv (B \vee (A \wedge \bigcirc (A \mathcal{U} B)) \end{aligned}$$

- Näin ollen esim:

$$\begin{aligned} \Diamond \Box \neg A &\equiv \Box \neg A \vee \bigcirc \Diamond \Box \neg A \\ &\equiv (\neg A \wedge \bigcirc \Box \neg A) \vee \bigcirc \Diamond \Box \neg A, \end{aligned}$$

tai

$$\begin{aligned} \Box \Diamond A &\equiv \Diamond A \wedge \bigcirc \Box \Diamond A \\ &\equiv ((A \vee \bigcirc \Diamond A) \wedge \bigcirc \Box \Diamond A. \end{aligned}$$

Ilmaisuvoimasta

- Tarkastellaan ominaisuutta p, joka pätee jokaisessa parillisessa tilassa (tilat numeroitu kokonaisluvuin).
- Eli esim:

		0	1	2	3	4	5	6	7	8	9	10 ...
s_1	=	p	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	$p \ldots$
s_2	=	p	p	p	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	$p \ldots$

mutta ei

		0	1	2	3	4	5	6	7	8	9	10 ...
s_3	=	$\neg p$	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	p	$\neg p$	$p \ldots$
s_4	=	p	p	p	$\neg p$	$\neg p$	$\neg p$	p	$\neg p$	p	$\neg p$	$p \ldots$

- Tällaista ominaisuutta ei voida kuvata annetuilla konnektiiveilla.

Ilmaisuvoimasta

- Esim. kaavat

$$p \wedge \Box(p \Rightarrow \bigcirc \neg p) \wedge \Box(\neg p \Rightarrow \bigcirc p)$$

tai

$$p \wedge \Box(p \Rightarrow \bigcirc \bigcirc p)$$

eivät päde edellä esitetyissä malleissa.

- Yleisesti pätee seuraava lause:

Millä tahansa kaavalla, joka koostuu atomisista kaavoista ja korkeintaa n:stä $\bigcirc$ operaattorista, on sama totuusarvo kaikille sekvensseille

$$\sigma_k = p^k(\neg p)p^\omega,$$

missä $k > n$.

Viitteitä

(1) Z. Manna & A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems: Specification*

Temporaalilogiikan käyttö järjestelmien spesifiointiin, verifointi todistamalla.

(2) P. Wolper: *Temporal Logic*. Chapter 4 in A. Thayse, *From Modal Logic to Deductive Databases*.

Mallintarkastusalgoritmit.

Reiluus

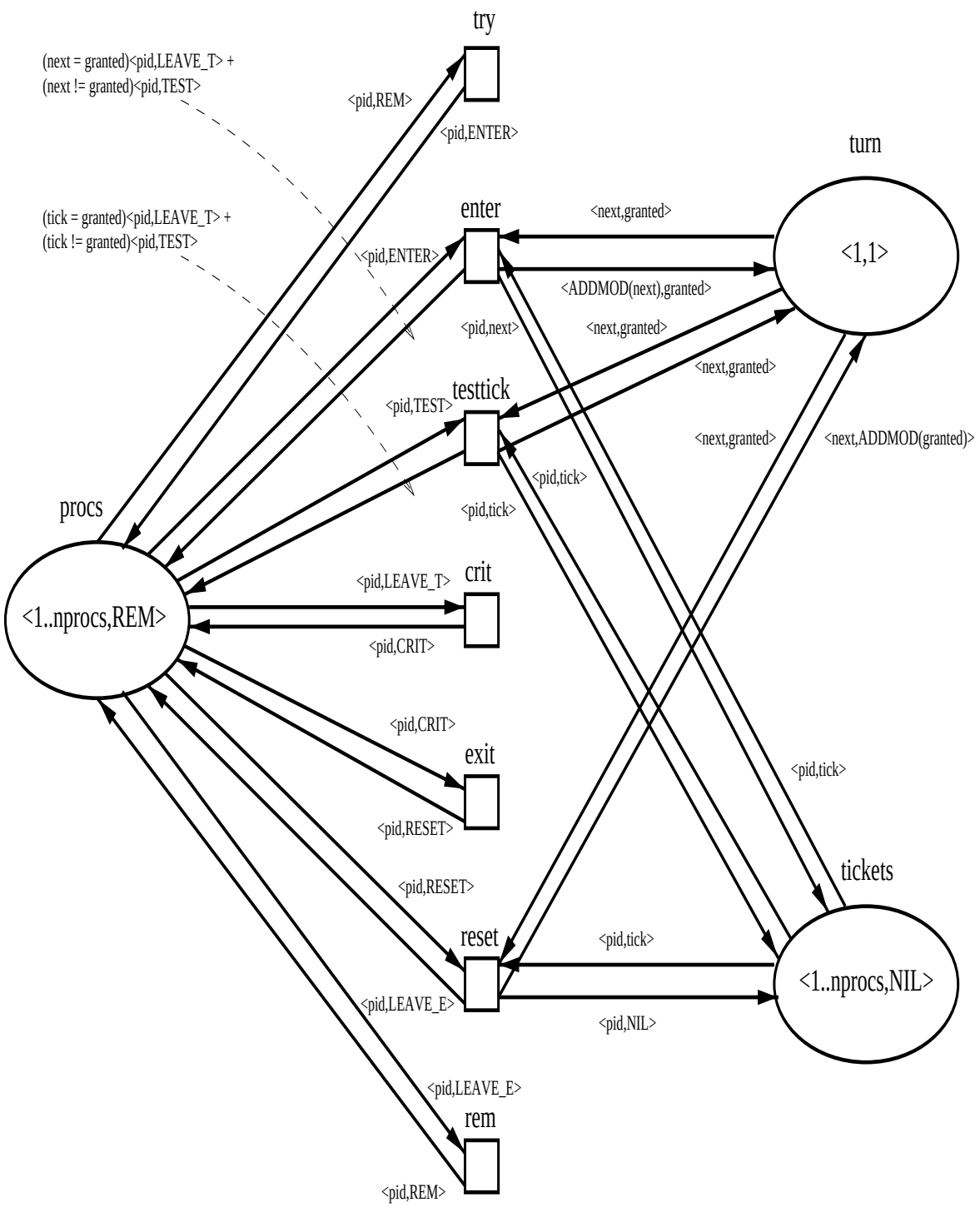
Tutkitaan temporaalilogiikan avulla seuraavan MUTEX-algorithmin toimintaa:

TicketME algorithm (informal):

A shared variable holds a pair $\langle next, granted \rangle$ of values in $\{1, \ldots, n\}$, initially $\langle 1, 1 \rangle$. The *next* component represents the next "ticket" into the critical section, while the *granted* component represents the last "ticket" that has been granted permission to enter the critical section. When a process enters the trying section, it "takes a ticket", that is, it copies and increments the *next* component modulo n. When a process's ticket is equal to the *granted* component, it goes to the critical region. When process exits the critical section, it increments the *granted* component modulo n.

Paraphrased from: *Nancy A. Lynch: Distributed Algorithms, 1996, (ISBN: 1-55860-348-4).*

Ticket Mutex Pr/T-verkko



Ominaisuuksien verifionti

Olkoon Ticket-ME algoritmin malli kahdelle prosessille N_{T_2}. Siitä voidaan voidaan verifioida verifioida seuraavia ominaisuuksia:

- Prosessit ovat toisensa poissulkevia:

$$N_{T_2} \models \Box(\neg(procs(\langle 1, CRIT\rangle) \wedge procs(\langle 2, CRIT\rangle))) \quad (2)$$

- On olemassa suoritus, jossa prosessi pääsee kriittiseen sektioon (Verifiointi vastaväitteellä):

$$N_{T_2} \not\models \Box(\neg(procs(\langle 1, CRIT\rangle))) \quad (3)$$

$$N_{T_2} \not\models \Box(\neg(procs(\langle 2, CRIT\rangle))) \quad (4)$$

- Jos prosessi haluaa kriittiseen sektioon, se lopulta myös sinne pääsee (Tätä ei mallista saada verifioitua!):

$$N_{T_2} \models \Box(procs(\langle 1, ENTER\rangle) \Rightarrow \Diamond procs(\langle 1, CRIT\rangle)) \quad (5)$$

$$N_{T_2} \models \Box(procs(\langle 2, ENTER\rangle) \Rightarrow \Diamond procs(\langle 2, CRIT\rangle)) \quad (6)$$

Reiluus

Miksei mallissa prosessi aina halutessaan lopulta pääse kriitiseen sektioon?

Syy tähän on se, ettei malli ota huomioon *reiluutta* prosessien välillä. Mallissa ovat mahdollisia myös sellaiset suoritukset, joissa toinen prosessi ei jonkun määrän askelia jälkeen enää suorita yhtään transitiota.

Esimerkki:

Tarkastellaan suoritusta, jossa suoritetaan transitiot (suorittavan prosessin numerolla indeksoituna):

$$try_1, try_2, enter_1, enter_2, crit_1, testtick_2, testtick_2, \ldots$$

Tässä suorituksessa prosessi 1 pääsee kriittiseen sektioon, muttei koskaan poistu sieltä. Prosessi 2 on tilassa, jossa se testaa jatkuvasti pääsyä kriittiseen sektioon transitiolla $testtick_2$, eikä prosessi 1 koskaan pääse poistumaan kriittisestä sektiosta transitiolla $exit_1$.

Monesti halutaan mallista poistaa tällaiset tilanteet, joita ei todellisissa järjestelmissä välttämättä esiinny, koska esim. prosesseja skeduloiva järjestelmä takaa kaikille prosesseille tasapuoliset mahdollisuudet edetä.

REILUUS

Joudumme siis määrittelemään, mitkä suoritukset ovat reiluja.

Reiluus on mallikohtainen ominaisuus, emmekä mene tässä kurssissa syvemmälle erilaisiin reiluuden määritelmiin.

Koska reiluus on suoritusten ominaisuus, voimme käyttää LTL-temporaalilogiikkaa myös reilujen suoritusten määrittelyyn.

Tällöin sen sijasta että verifiomme kaavan:

$$N \models property$$

verifioimmekin kaavan:

$$N \models fairness \Rightarrow property,$$

jossa $property$ on verifioitava LTL-kaava, ja $fairness$ on *reiluus-oletus*.

Reiluus-oletus on LTL-kaava joka on tosi kaikille reiluille poluille, ja vastaavasti epätosi kaikille ei reiluille poluille. Se poistaa ominaisuuden pätemiseen tutkittavista suorituksista kaikki suoritukset jotka eivät ole reiluja.

REILUUS

Käytämme esimerkissämme seuraavaa reiluuden määritelmää:

Reiluja ajoja ovat kaikki ne äärettömät ajot, joissa kumpikin prosessi suorittaa äärettömän monta transitiota.

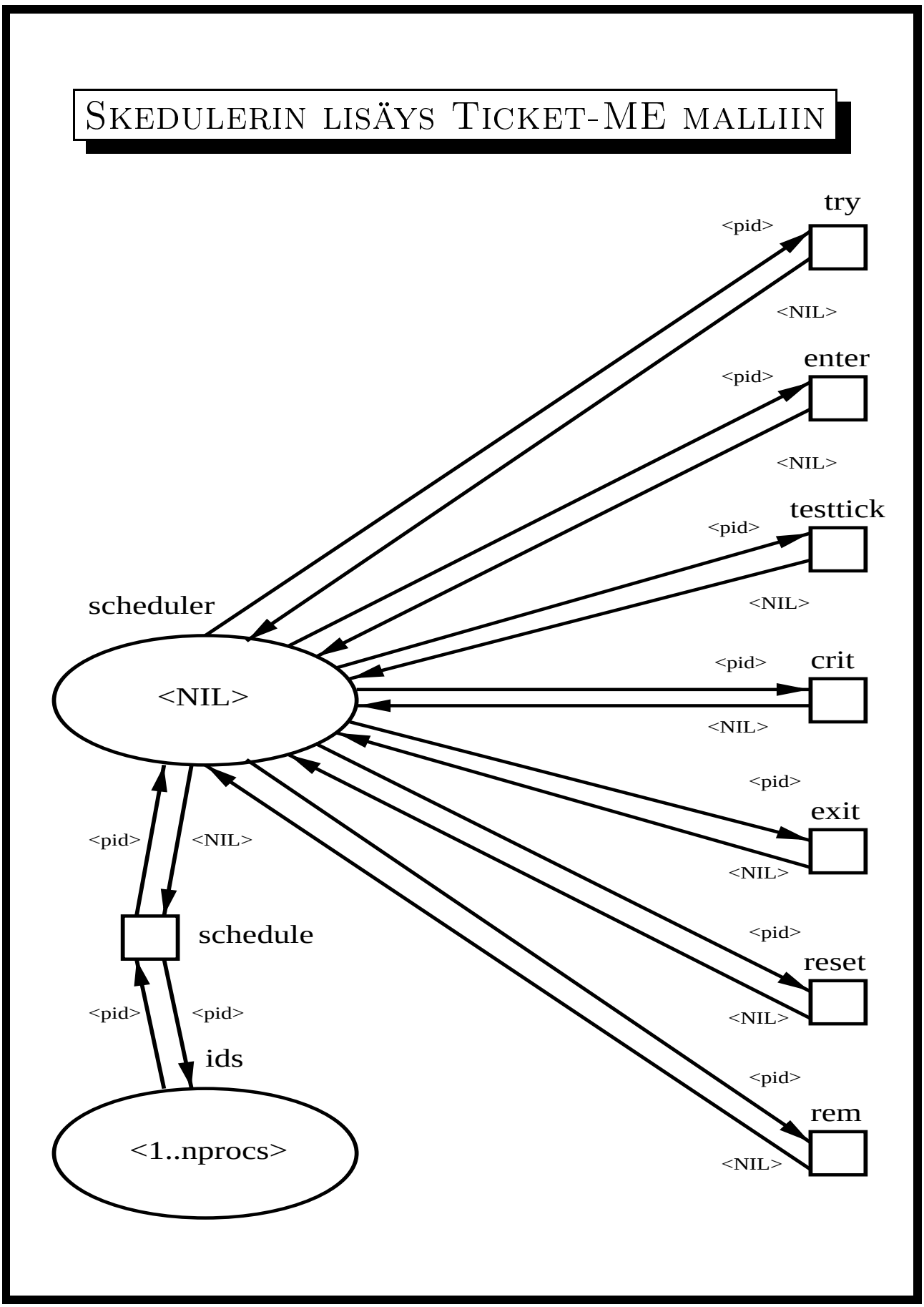
Toisin sanottuna kiellämme suoritukset, joissa jompi kumpi prosesseista jätetään jonkin ajanhetken jälkeen kokonaan suorittamatta.

Käyttämämme logiikka ei puhu mitään transitioista, joten joudumme tekemään malliin muutoksen saadaksemme saman asian kuvattua pelkkien mallin tilojen avulla.

Lisätään malliin prosessien skedulointi lisäämällä kaksi uutta paikkaa, yksi transition, ja kaaret molempiin suuntiin kaikista jo olemassa olevista transitioista. Olkoon saatu uusi malli N'_{T_2}. (Katso seuraava sivu.)

Järjestelmän suorituksissa nyt joka toinen transitio on $schedule$ -transitio, ja joka toinen joku järjestelmän muista transitioista.

Skedulerin lisäys Ticket-ME malliin



REILUUS

Nyt haluamamme reiluus-oletus on helppo kuvata seuraavalla LTL-kaavalla:

$$fairness \equiv \Box\Diamond scheduler(\langle 1 \rangle) \wedge \Box\Diamond scheduler(\langle 2 \rangle)$$

Kaava sallii vain sellaiset ajot, joissa molempia prosesseja skeduloidaan loputtomasti.

Nyt haluttujen elävyys-ominaisuuden verifiointi onnistuu:

$$\begin{aligned} N'_{T_2} \quad \models \quad & (\Box\Diamond scheduler(\langle 1 \rangle) \wedge \Box\Diamond scheduler(\langle 2 \rangle)) \Rightarrow \\ & (\Box(procs(\langle 1, ENTER \rangle) \Rightarrow \Diamond procs(\langle 1, CRIT \rangle))) \end{aligned}$$

$$\begin{aligned} N'_{T_2} \quad \models \quad & (\Box\Diamond scheduler(\langle 1 \rangle) \wedge \Box\Diamond scheduler(\langle 2 \rangle)) \Rightarrow \\ & (\Box(procs(\langle 2, ENTER \rangle) \Rightarrow \Diamond procs(\langle 2, CRIT \rangle))) \end{aligned}$$

Reiluutta ei tarvita turvallisuus-ominaisuuksia tarkistettaessa. (Kaikkien suoritusten, jotka rikkovat turvallisuus-ominaisuuden voidaan katsoa päättyvän ominaisuuden rikkovaan tilaan, ja suoritus voi olla olematta reilu vain olessaan päättymätön.)