

hyväksymispäivä arvosana

arvostelija

Yleinen vakautuva paikallinen synkronointialgoritmi

Panu Luosto

Helsinki 29.10.2007

Seminaarikirjoitelma

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Sisältö

1	Johdanto	1
2	Määritelmät	1
2.1	Hajautettu järjestelmä ja vakautuvuus	1
2.2	Äärellinen askeltava järjestelmä ja palautus	2
2.3	Verkkoteoreettisia käsitteitä	3
3	Yleinen vakautuva synkronointialgoritmi	3
3.1	Algoritmin kuvaus	4
3.2	Esimerkkejä algoritmin soveltamisesta	5
3.3	Parametrit K ja α	7
4	Viive ja lukkiutumattomuus	8
4.1	Viiveen määritelmä ja syklit	8
4.2	Viiveen avulla määritelty järjestys	9
4.3	Alaraja parametrin K arvolle	9
5	Palautusvaihe ja vakautuvuus	10
5.1	Lukkiutunut järjestelmä	10
5.2	Palautusverkko	11
5.3	Vakautuvuus ja yläraja parametrin α arvolle	12
	Lähteet	14

1 Johdanto

Prosessien välinen synkronointi ilman globaalia tahdistusta kuuluu perustavanlaatuisimpiin ongelmiin hajautettujen järjestelmien teoriassa. Usein kyse on paikallisesta synkronoinnista, jossa yksittäisen prosessin tahdistusvaatimus esitetään vain sen itsensä ja prosessin naapureiden avulla. Klassisista ongelmista kuten keskinäinen poissulkeminen, lukijat-kirjoittajat ja ryhmien keskinen poissulkeminen on ole-massa paikalliset muunnelmansa. Hyvä synkronointialgoritmi sallii mahdollisimman suuren yhtäaikaisuuden prosessien välillä ja on lisäksi vikasietoinen. Kaikkein yleisin malli vikasietoisuudesta on vakautuva järjestelmä, joka pystyy äärellisessä ajassa siirtymään lailliseen tilaan mielivaltaisesta alkutilanteesta.

Tässä tutkielmassa esiteltävä algoritmi on vakautuva ja ratkaisee minkä tahansa synkronointiongelman, jonka oikeellisuus on määriteltävissä paikallisesti. Ratkaisu on tilankäytön suhteen varsin säästeliäs: pahimmassakin tapauksessa prosessikohtaisen kellomuuttujan on pystyttävä ilmaisemaan vain $2n - 1$ erilaista tilaa, kun n on prosessien lukumäärä. Toisaalta esimerkiksi puumaisessa verkossa tarvittavien tilojen lukumäärä on aina 3 riippumatta verkon solmujen (prosessien) lukumäärästä. Algoritmi ovat esittäneet Boulinier, Petit ja Villain [BPV04].

2 Määritelmät

Tässä luvussa määritellään täsmällisesti tutkielmassa käytettävät käsitteet. Ensimmäinen aliluku sisältää yleisiä hajautettuihin järjestelmiin liittyviä asioita. Sitä seuraavat kaksi alilukua esittelevät äärellisen askeltavan järjestelmän sekä muutamia tarvittavia verkkoteoreettisia käsitteitä.

2.1 Hajautettu järjestelmä ja vakautuvuus

Hajautettu järjestelmä on suuntaamaton äärellinen yhtenäinen verkko $G = (V, E)$, missä V on solmujen (prosessien) joukko ja E on niitä yhdistävien särmien joukko. Joukon V alkioden määrä on $|V| = n$, $n \geq 2$. Kaksi solmua p ja q ovat *naapureita*, jos ja vain jos särmä (p, q) kuuluu joukkoon E . Solmun n naapureiden joukkoa merkitään \mathcal{N}_p .

Jokaisella prosessilla on rekisterinsä (muuttujansa), joihin se voi kirjoittaa. Lisäksi prosessi voi lukea oman ja naapuriprosessien rekistereiden sisällön. Prosessi suorittaa ohjelmaa, joka koostuu seuraavanlaisista ilmaisuista:

$$\textit{tunnus} : \textit{ehto} \longrightarrow \textit{toimenpide}.$$

Prosessin p ohjelman *ehto* on totuusarvoinen väittämä, jossa voi esiintyä viittauksia p :n ja sen naapureiden rekisterien arvoihin. *Toimenpide* suoritetaan vain silloin, kun ehto on tosi, ja se muuttaa yhtä tai useampaa prosessin p rekisterin arvoa. Mikäli eh-

to on tosi, ehdon arvon määrittäminen ja toimenpiteen suorittaminen muodostavat yhden jakamattoman kokonaisuuden.

Prosessin rekisterien arvot määrittelevät prosessin *tilan*, ja kaikkien prosessien tilat määrittelevät järjestelmän *konfiguraation*. Kaikkien mahdollisten konfiguraatioiden joukko olkoon \mathcal{C} , jolloin hajautettu protokolla voidaan määritellä relaationa R joukossa \mathcal{C} . Tätä relaatiota vastaa *siirtymäverkko* $S = (\mathcal{C}, R)$. Jono $e = \gamma_0\gamma_1\dots$ on protokollan *suoritusketju* täsmälleen silloin, kun $(\gamma_i, \gamma_{i+1}) \in R$ kaikilla $i \geq 0$.

Prosessin p sanotaan olevan *aktiivinen* konfiguraatiossa $\gamma \in \mathcal{C}$, jos prosessin ohjelmassa on toimenpide, jonka ehto on tosi. Tässä kirjoituksessa oletetaan, että siirtymät järjestelmän konfiguraatiosta toiseen määrää *epäreilu demoni*. Demoni valitsee yhden laskenta-askelen aikana aina yhden tai useamman aktiivisen prosessin suoritukseen. Epäreilisuus tarkoittaa, että vaikka prosessi p olisi jatkuvasti aktiivinen, sitä ei välttämättä koskaan valitaan suoritukseen, mikäli on olemassa jokin toinen aktiivinen prosessi kuin p .

Järjestelmän konfiguraatioon liittyvä ehto P on *suljettu* siirtymäverkossa S , jos ja vain jos jokaisessa suoritusketjussa $e = \gamma_0\gamma_1\dots$, jossa konfiguraatio γ_0 täyttää ehdon P , ehto täyttyy kaikilla γ_i , $i \geq 0$. Siirtymäverkko S on *vakautuva* ehdon P suhteen täsmälleen silloin, kun ehto P on suljettu verkossa S ja jokaisessa suoritusketjussa e on konfiguraatio, joka täyttää ehdon P .

2.2 Äärellinen askeltava järjestelmä ja palautus

Tarkoittakoon merkintä \bar{a} joukon $\{0, 1, \dots, K-1\}$ ($K > 2$) kokonaislukua, jolla $a \equiv \bar{a} \pmod{K}$. Kahden kokonaisluvun a ja b välinen *K -etäisyys* (tai lyhyesti *etäisyys*) olkoon $d_K = \min\{a-b, b-a\}$. Kahden kokonaisluvun sanotaan olevan *paikallisesti vertailtavia*, jos ja vain jos $d_K(a, b) \leq 1$. Paikallinen järjestysrelaatio \leq_l määritellään seuraavasti: $a \leq_l b \iff 0 \leq b-a \leq 1$. Olkoot a ja b paikallisesti vertailtavia kokonaislukuja. Tällöin määritellään

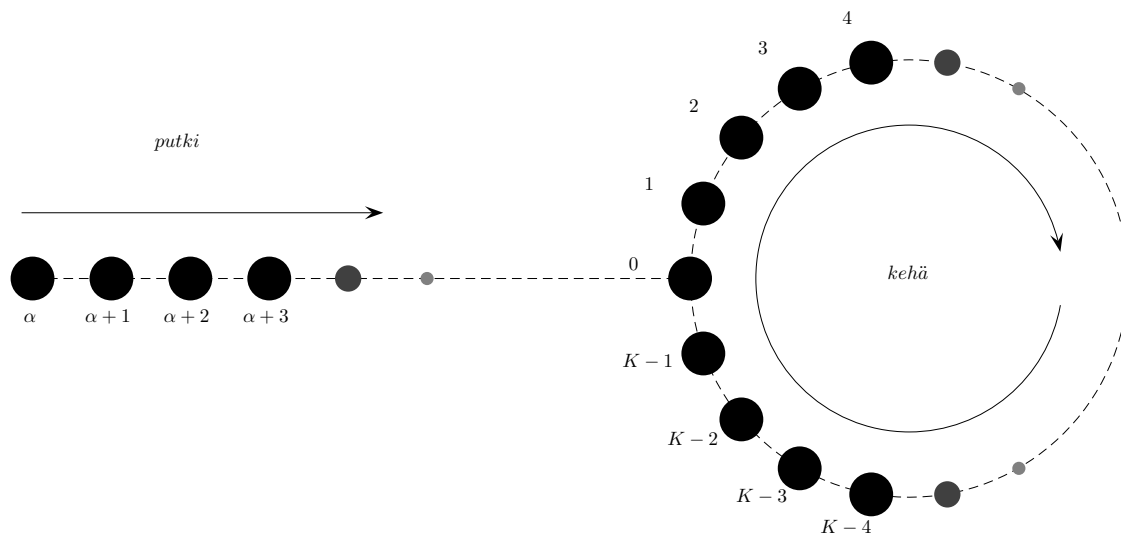
$$b \ominus_l a = \begin{cases} -1, & \text{jos } a \equiv b+1 \pmod{K} \\ 0, & \text{jos } a = b \\ 1, & \text{jos } b \equiv a+1 \pmod{K} \end{cases}$$

Olkoon $\mathcal{X} = \{\alpha, \alpha+1, \dots, 0, \dots, K-2, K-1\}$, missä α on negatiivinen kokonaisluku. Olkoon lisäksi $\varphi : \mathcal{X} \rightarrow \mathcal{X}$,

$$\varphi(x) = \begin{cases} x+1, & \text{jos } x < K-1 \\ 0, & \text{jos } x = K-1. \end{cases}$$

Pari (\mathcal{X}, φ) on *äärellinen askeltava järjestelmä*, jonka sykli on K . *Palautus* tarkoittaa sijoitusta, jossa muuttujan joukkoon $\mathcal{X} \setminus \{\alpha\}$ kuuluva arvo korvataan arvolla α .

Olkoon *putki* $_{\varphi} = \{\alpha, \alpha+1, \dots, -1, 0\}$ ja *kehä* $_{\varphi} = \{0, 1, \dots, K-2, K-1\}$. Merkitään lisäksi *putki* $_{\varphi}^* = \text{putki}_{\varphi} \setminus \{0\}$.



Kuva 1: Äärellinen askeltava järjestelmä

2.3 Verkkooteoreettisia käsitteitä

Suuntaamattoman verkon *syklikanta* (engl. cycle basis) voidaan etsiä seuraavasti. Kiinnitetään jokin verkon virittävä puu. Nyt jokainen puuhun kuulumaton kaari voidaan täydentää sykliksi yhdistämällä kaaren solmut sillä yksikäsitteisellä polulla, joka on solmujen välillä käytetyssä virittävässä puussa. Näin muodostettujen syklien joukko on eräs verkon syklikanta. Syklikanta ei ole yleensä yksikäsitteinen. Kannan muodostamisesitelmä on esitetty Kirchhoffin vuonna 1847 ilmestyneessä artikkelissa [Kir47].

Verkon G *lyhin syklikanta* on se syklikanta B , jonka syklien pituuksien summa on pienin kaikkien G :n syklikantojen joukossa. Joukko B on samalla ratkaisu ongelmaan löytää se syklikanta, jonka pisin sykli on lyhin kaikkien syklikantojen pisimpien syklien joukossa.

Verkon G *jänteetön sykli* eli *reikä* C on sykli, jonka minkään kahden solmun välillä verkossa G ei ole sykliin C kuulumatonta kaarta.

3 Yleinen vakautuva synkronointialgoritmi

Tässä luvussa esitetään itse synkronointialgoritmi ja muutama sen käytännön sovellus. Algoritmin toiminnan kannalta keskeisiä askeltavan järjestelmän parametreja K ja α sivutaan viimeisessä aliluvussa.

Vakiot	
\mathcal{N}_p	prosessin p naapureiden joukko
Muuttujat	
$r_s \in \mathcal{X} \ (s \in \mathcal{N}_p \cup \{p\})$	prosessin s kellorekisteri $ehto_p(q)$ voi käyttää myös muita p :n ja q :n muuttujia
Totuusarvoiset makrot	
$ehto(p, q)$	prosessin p ja sen naapurin q välinen sykronointiehto
$kehällä(p, q)$	$\equiv r_p \in kehä_\varphi \wedge r_q \in kehä_\varphi$
$askelehto(p, q)$	$\equiv kehällä(p, q) \wedge ((\varphi(r_p) = r_q) \vee ((r_p = r_q) \wedge ehto(p, q)))$
$pariehto(p, q)$	$\equiv askelehto(p, q) \vee askelehto(q, p)$
$naapurustoehto(p)$	$\equiv \forall q \in \mathcal{N}_p : pariehto(p, q)$
$kehäaskel(p)$	$\equiv \forall q \in \mathcal{N}_p : askelehto(p, q)$
$putkiaskel(p)$	$\equiv r_p \in putki_\varphi^* \wedge (\forall q \in \mathcal{N}_p : (r_q \in putki_\varphi) \wedge (r_p \leq r_q))$
$palautusaskel(p)$	$\equiv \neg naapurustoehto(p) \wedge (r_p \notin putki_\varphi)$
Ohjelma	
$KEH : kehäaskel(p)$	\longrightarrow varsinainen tehtävä ; $r_p := \varphi(r_p)$
$PUT : putkiaskel(p)$	\longrightarrow $r_p := \varphi(r_p)$
$PAL : palautusaskel(p)$	\longrightarrow $r_p := \alpha$

Taulu 1: Yleinen paikallinen sykronointialgoritmi (prosessin p ohjelma)

3.1 Algoritmin kuvaus

Algoritmi on kuvattu taulussa 1. Tarkasteltavana on äärellinen askeltava järjestelmä (\mathcal{X}, φ) , jossa $K > 2$ ja $\alpha \leq 0$. Kullakin prosessilla p on kellorekisteri $r_p \in \mathcal{X}$. Ongelmalle ominainen laillisuusehto kahden naapurisolmun välillä on sisällytetty totuusarvoiseen makroon $ehto(p, q)$, joka voi olla riippuvainen myös muista muuttujista v_p ja v_q . Ohjelmaosa sisältää kolme erilaista tapausta, joilla on prosessin kellorekisterin arvon suhteen yksinkertaiset tulkinnat. Yksi liittyy askellukseen joukossa $kehä_\varphi$, toinen askellukseen joukossa $putki_\varphi$, ja kolmas tapaus on palautusaskel, jossa rekisterin arvoksi asetetaan α .

Funktio $askelehto(p, q)$ arvo on tosi, jos solmu q sallii solmun p suorittaa varsinaisen synkronoidun tehtävänsä. Edellytyksenä tälle on ensinnäkin, että molempien solmujen kellorekisterien arvot ovat joukossa $kehä$. Lisäksi joko rekisterin r_p arvon täytyy olla yhden askelluksen verran jäljessä rekisterin r_q arvosta – tai arvot ovat yhtäsuuret, ja ongelmalle ominaisen makron $ehto(p, q)$ arvo on tosi. Kun $ehto$ määritellään sopivasti, voidaan ratkaista erityyppisiä sykronointia vaativia ongelmia. Tässä mahdollisesti tarvitaan muitakin muuttujia kuin prosessien kellorekisterejä. Merkitään prosessin p kaikkia lisämuuttujia $v_p \in \Sigma$, missä joukon Σ voidaan katsoa edustavan mitä tahansa tietotyyppiä.

Jos $askelehto(p, q)$ on tosi jokaisen p :n naapurin q kanssa, prosessi p on oikeutettu suorittamaan varsinaisen synkronoitavan tehtävänsä. Tämän jälkeen kellorekisterin

r_p arvoa kasvatetaan.

Prosessi p ei saa suorittaa synkronoitua tehtäväänsä, jos *naapurustoehto*(p) on epätosi. Jos tällöin pätee lisäksi $r_p \notin \text{putki}_\varphi$, suoritetaan askel *PAL*, eli asetetaan rekisterin r_p arvoksi α . Mikäli joko askelen *PAL* ehto on voimassa prosessin p kohdalla, tai $r_p \in \text{putki}_\varphi^*$, solmun p sanotaan olevan *palautusvaiheessa*. Ohjelma-askel *PUT* liittyy palautuksen jälkeiseen vaiheeseen, jossa prosessin kellolaskurin arvoa kasvatetaan synkronoidusti kaikki naapurien kanssa.

Algoritmin tarkastelussa tarvitaan jatkossa koko järjestelmään liittyvää käsitettä *kehälle vakiintunut*. Se määritellään seuraavasti:

$$\forall p \in V : \text{naapurustoehto}(p) ,$$

eli yhtäpitävästi

$$\forall p \in V, \forall q \in \mathcal{N}_p : \text{askelehto}(p, q) \vee \text{askelehto}(q, p) .$$

Kun järjestelmä on kehälle vakiintunut, mikään sen prosessi ei ole palautusvaiheessa.

Olkoon $\Psi(p)$ prosessin p kuva sen jälkeen, kun p on suorittanut askelen *KEH*. Nyt voimme lausua viimeiset tarvittavat oletukset:

$$\begin{aligned} \mathcal{O}_1 & : (\text{askelehto}(p, q) \Rightarrow \text{askelehto}(\Psi(p), \Psi(q))) \\ & \quad \wedge (\text{askelehto}(p, q) \Rightarrow \text{pariehto}(\Psi(p), q)) \\ \mathcal{O}_2 & : (r_p = r_q = 0) \Rightarrow \text{pariehto}(p, q) . \end{aligned}$$

Nämä ehdot vaikuttavat siihen, miten varsinainen synkronoitava tehtävä ja makro *ehto* voidaan määritellä. Oletuksesta \mathcal{O}_1 seuraa, että jos järjestelmä on jollakin hetkellä kehälle vakiintunut, se on kehälle vakiintunut myös sen jälkeen, kun prosessi p on suorittanut normaalin ohjelma-askelensa *KEH*. Erityisesti ehto on totta siitä riippumatta, suorittavatko solmun p naapurit oman ohjelma-askelensa *KEH* samanaikaisesti.

3.2 Esimerkkejä algoritmin soveltamisesta

Tässä aliluvussa esitetään lyhyesti, miten eräät klassiset synkronointiongelmat voidaan ratkaista, kun makro *ehto* on sopivasti määritelty. Kaikissa tapauksissa jokaisen prosessin tulee suorittaa varsinainen tehtävänsä eli ohjelma-askel *KEH* äärettömän monta kertaa. Esimerkeissä näytetään, että ehdot \mathcal{O}_1 ja \mathcal{O}_2 toteutuvat kussakin tapauksessa.

Oletuksena on, että järjestelmä on kehälle vakiintunut. Seuraavassa luvussa osoitetaan, että kehälle vakiintunut järjestelmä ei voi lukkiutua, jos parametrin K arvo on riittävän suuri. Myöhemmin tullaan lisäksi näyttämään, että algoritmit ovat vakautuvia kehälle vakiintuneisuuden suhteen, kunhan vakio α on valittu sopivasti.

Asynkroninen unisono on algoritmi, jossa naapurisolmujen kellorekisterien arvojen K-etäisyys toisistaan on korkeintaan 1. Algoritmi on toteutettavissa triviaalisti asettamalla makro $ehto(p, q)$ identtisesti todeksi. Tällöin voidaan sieventämällä todeta, että

$$pariehto(p, q) = kehällä(p, q) \wedge (r_p = \varphi(r_q) \vee r_p = r_q \vee r_q = \varphi(r_p)).$$

Koska järjestelmä on kehälle vakiintunut, kaikki kellorekisterien arvot kuuluvat joukkoon $kehä_\varphi$, ja $kehällä(p, q)$ on tosi. Voidaan helposti todeta, että ehdot \mathcal{O}_1 ja \mathcal{O}_2 toteutuvat. Samalla algoritmin laillisuusehto selvästi täyttyy.

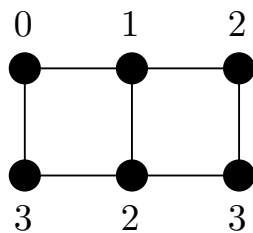
Paikallinen poissulkeminen, paikallinen ryhmien keskinen poissulkeminen ja paikallinen lukijoiden-kirjoittajien ongelma ovat ratkaistavissa *geneerisen algoritmin* avulla. Algoritmi määrittellään seuraavasti. Olkoot $v_p, v_q \in \Sigma$ prosessien p ja q ylimääräiset muuttujat, ja olkoon \triangleleft joukon Σ täydellinen järjestys. Geneerisen sykronointiongelman laillisuusehto on silloin: $ehto(p, q) \equiv v_p \triangleleft v_q$. Nyt $pariehto(p, q)$ on sama kuin asynkronisen unisonon tapauksessa ja siten riippumaton järjestysrelaation \triangleleft määrittelystä. Ehdot \mathcal{O}_1 ja \mathcal{O}_2 toteutuvat nytkin selvästi.

Paikallisen poissulkemisen ongelman ratkaiseminen ei ole mahdollista mielivaltaisessa verkossa ilman solmuihin liitettyjä yksilöllisiä tunnisteita [KY02]. Jos prosessilla p on tunniste v_p , ja \triangleleft on tunnisteiden täydellinen järjestys, ongelma ratkeaa geneerisen algoritmin avulla. Ongelman laillisuusehto on: *kaksi naapuriprosessia ei koskaan pidä yhtäaikaisesti hallussaan suojattua resurssia*. Näytetään, että tämä vaatimus täyttyy. Oletetaan väitteen olevan väärä ja että suojatun resurssin sai ennen laillisuusehdon rikkoutumista ensimmäisenä käyttöönsä prosessi p . Olkoon prosessi q on ensimmäinen p :n naapuri, joka jakaa suojatun resurssin. Ennen laittoman ohjelmaaskelen KEH ottamista joko $\varphi(r_q) = r_p$ tai $r_q = r_p$. Tarkastellaan ensimmäistä tapausta.

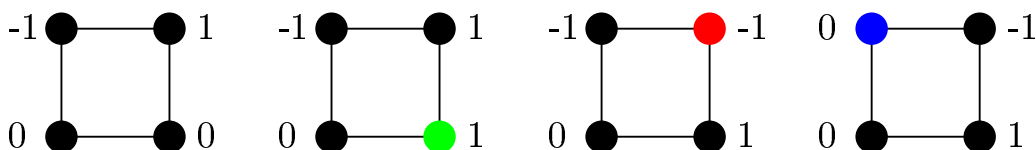
Prosessin p kellorekisterin arvo ei ole muuttunut sen jälkeen, kun p sai suojatun resurssin haltuunsa hetkellä t . Koska prosessi q ei ole aikaisemmin pitänyt yhtäaikaisesti p :n kanssa hallussaan resurssia ja koska tämän aliluvun yleisoletuksen mukaisesti järjestelmä on kehälle vakiintunut, myöskään prosessin q kellorekisterin arvo ei ole muuttunut hetken t jälkeen. Päädyttiin ristiriitaan, koska p ei olisi voinut suorittaa ohjelma-askelta KEH tilanteessa, jossa $\varphi(r_q) = r_p$ (tällöin $pariehto(p, q)$ olisi ollut epätosi).

Toisessa tapauksessa $r_q = r_p$. Jotta q saa suorittaa ohjelma-askelen KEH , joka tuo suojatun resurssin sen käyttöön, pitää väitteen $v_q \triangleleft v_p$ olla tosi. Ristiriitaan päädytään samantapaisella järjelyllä kuin edellisessä tapauksessa, koska prosessin p ottaessa resurssin haltuunsa on ehtona ollut $v_p \triangleleft v_q$.

Käsitellään lopuksi ryhmien keskinen poissulkemisen ongelmaa, johon myös lukijoiden ja kirjoittajien ongelma on palautettavissa. Sen laillisuusehto voidaan lausua seuraavasti: *kaksi naapuriprosessia eivät saa käyttää yhtäaikaisesti eri resursseja*. Olkoon $v_s.id$ prosessin s yksilöllinen tunniste ja $v_s.re$ resurssi, jonka prosessi s haluaa käyttöönsä. Voidaan sopia, että prosessi haluaa käyttöön erityisen tyhjän resurssin \perp , jos se ei tavoittele mitään todellista resurssia. Nyt ongelman laillisuusehdon



Kuva 2: Lukkiutunut tilanne kehälle vakiintuneessa järjestelmässä, kun $K = 4$.



Kuva 3: Suoritusketjun osa, kun $\alpha = -1$. Laittomasta tilanteesta toipuminen voi viedä äärettömän kauan.

toteuttava järjestysrelaatio voidaan määritellä muunnelmalla edellisen algoritmin tapauksesta: $v_p \triangleleft v_q \equiv (v_p.re = v_q.re) \vee (v_p.id \leq v_q.id)$, missä \leq on täydellinen järjestys prosessitunnisteiden joukossa.

3.3 Parametrit K ja α

Jotta edellä esitetty algoritmi toimisi halutulla tavalla, täytyy parametrien K ja α arvot valita sopivasti. Kehälle vakiintuneiden konfiguraatioiden joukko on suljettu siirtymäverkossa. Tämä ei kuitenkaan sulje pois sitä mahdollisuutta, että järjestelmä voi joutua lukkiutuneeseen tilaan. Esimerkkinä olkoon kuva 2. Siinä käytetään parametrin arvoa $K = 4$, ja järjestelmän tila on kehälle vakiintunut mutta samalla lukkiutunut. Jokaisella solmulla p on nimittäin naapuri q siten, että $\varphi(r_q) = r_p$. Siten *askelehto*(p) on epätosi.

Parametri α vaikuttaa järjestelmän toipumiseen häiriöstä. Kuvassa 3 vakion α arvo on liian pieni ($\alpha = -1$). Vihreällä korostetut muutokset ovat kehäaskelen aiheuttamia, punainen väri liittyy palautukseen ja sininen putkiaskeleeseen. Suoritusketjun ansiosta kuvio pyörähti 90° myötäpäivään. Symmetrian perusteella nähdään, että laittomasta tilanteesta toipuminen voi tässä tapauksessa viedä äärettömän kauan.

4 Viive ja lukkiutumattomuus

Tässä luvussa oletetaan järjestelmän olevan kehälle vakiintunut. Uuden käsitteen *viive* avulla määritellään, mikä on pienin mahdollinen parametrin K arvo, jolla järjestelmän lukkiutumattomuus voidaan taata.

4.1 Viiveen määritelmä ja syklit

Kehälle vakiintuneessa järjestelmässä kahden naapurisolmun kellorekisterin arvojen K -etäisyys toisistaan on nolla tai yksi, eli rekisterien arvot ovat paikallisesti vertailtavia. Tällöin määritellään polun $\mu = p_0p_1 \dots p_k$ viive Δ_μ seuraavasti:

$$\Delta_\mu = \sum_{i=0}^{k-1} (r_{p_{i+1}} \ominus_l r_{p_i}).$$

Selvästi polun μ ja sen käänteispolun $\tilde{\mu}$ viiveet ovat toistensa vastalukuja: $\Delta_\mu = -\Delta_{\tilde{\mu}}$. Lisäksi jos $\mu_1 = p_0p_1 \dots p_j$ ja $\mu_2 = p_jp_{j+1} \dots p_k$, niin $\Delta_{\mu_1\mu_2} = \Delta_{\mu_1} + \Delta_{\mu_2}$.

Induktiolla polun pituuden suhteen voidaan näyttää, että polulla $p_0p_1 \dots p_k$ on $r_{p_0} + \Delta_{p_0p_k} \equiv r_{p_k} \pmod{K}$. Perustapaukset $k = 0$ ja $k = 1$ seuraavat suoraan määritelmästä. Jos oletetaan väitteen pätevän tapauksessa $k \geq 1$, niin polulla $p_0p_1 \dots p_kp_{k+1}$ pätee $r_{p_0} + \Delta_{p_0p_k} \equiv r_{p_k}$ ja $r_{p_k} + \Delta_{p_kp_{k+1}} \equiv r_{p_{k+1}} \pmod{K}$, joista saadaan laskemalla puolittain yhteen $r_{p_0} + \Delta_{p_0p_k} + \Delta_{p_kp_{k+1}} \equiv r_{p_{k+1}} \pmod{K}$. Koska $\Delta_{p_0p_k} + \Delta_{p_kp_{k+1}} = \Delta_{p_0p_{k+1}}$, väite tuli todistetuksi.

Kahden prosessin välinen viive voidaan määrittellä luontevasti, jos viive ei ole riippuvainen niiden välisen polun valinnasta. Sanomme viiveen olevan *rajoitettu* tietystä konfiguraatiossa, jos ja vain jos viive on 0 jokaisessa syklissä. Esimerkiksi koska puut ovat syklittömiä, kaikki puumaisen järjestelmän konfiguraatioiden viiveet ovat rajoitettuja.

Syklissä $\mu = p_0p_1 \dots p_kp_0$ pätee $r_{p_0} + \Delta_\mu \equiv r_{p_0} \pmod{K}$, joten $\Delta_\mu \equiv 0 \pmod{K}$. Esimerkiksi kuvassa 2 viive on kuuden mittaisessa syklissä positiiviseen kiertosuuntaan kuljettaessa -4 , ja $K = 4$.

Tarkastellaan prosesseja p ja q , jotka ovat naapureita kehälle vakiintuneessa järjestelmässä. Jos vähintään toinen prosesseista on aktiivinen, kyse on yhdestä seuraavista tapauksista:

1. p on aktiivinen. Silloin $\varphi(r_p) \ominus r_q = r_q \ominus r_p + 1$.
2. Sekä p että q ovat aktiivisia. Silloin $\varphi(r_p) \ominus \varphi(r_q) = r_p \ominus r_q$.
3. q on aktiivinen. Tässä tapauksessa $r_p \ominus \varphi(r_q) = r_p \ominus r_q - 1$.

Nähdään helposti, että syklin $\mu = p_0p_1 \dots p_0$ viive on muuttumaton kehälle vakiintuneessa järjestelmässä. Näin ollen jos viive on rajoitettu jossakin konfiguraatiossa, viive myös säilyy rajoitettuna koko suorituksen ajan.

4.2 Viiveen avulla määritelty järjestys

Jos viive on rajoitettu, voidaan määritellä Δ_{pq} tarkoittamaan viiveen arvoa kaikilla solmujen p ja q välisillä poluilla. Sanomme, että solmu p edeltää solmua q , jos ja vain jos $\Delta_{pq} \leq 0$. Koska järjestelmää kuvaava verkko G on yhtenäinen, näin saadaan solmujen täydellinen järjestys. Koska lisäksi verkon oletetaan olevan äärellinen, minimaalisten alkioiden joukko on aina epätyhjä.

Näytetään, että järjestelmässä on aina minimaalinen prosessi, joka on aktiivinen. Koska tässä luvussa oletetaan järjestelmän olevan kehälle vakiintunut, riittää tarkastella, onko *kehäehto*(p) tosi jonkin minimaalisen prosessin p kohdalla. Asynkronisen unisonon tapauksessa algoritmikohtainen makro *ehto* oli identtisesti tosi, joten minimaalinen prosessi on selvästi aktiivinen. Geneerisessä algoritmista vaadittiin, että \triangleleft on täydellinen järjestys. Niinpä viiveen avulla määriteltyyn järjestyksen mukaan minimaalisten prosessien joukossa on aina prosessi, joka on relaation \triangleleft suhteen minimaalinen. Tämä prosessi on aktiivinen.

Edellisen päättelyn tulos voidaan lausua seuraavassa muodossa.

Lause 1 *Jos viive on rajoitettu, kehälle vakiintunut järjestelmä ei voi lukkiutua geneerisen algoritmin ja asynkronisen unisonon tapauksissa.*

4.3 Alaraja parametrin K arvolle

Seuraavaksi määritellään sellainen alaraja parametrin K arvolle, joka takaa viiveen olevan rajoitettu.

Tarkastellaan järjestelmäverkon G särmäjoukkoa E jollakin hetkellä, kun järjestelmä on kehälle vakiintunut. Kiinnitetään särmille jokin mielivaltainen suunnistus, ja olkoon \vec{E} tämä suunnattujen särmien joukko. Samastetaan suunnattu sykli C sen suunnattujen särmien joukon kanssa: $C = \{\vec{e}_1, \vec{e}_2, \dots, \vec{e}_k\}$, missä $\vec{e}_i \in \vec{E}$ tai $\overleftarrow{e}_i \in \vec{E}$ kaikilla $i \in \{1, 2, \dots, k\}$. Olkoon seuraavassa $-1 \cdot C = -C$ syklin C käänteissykli, joka saadaan kääntämällä C :n kaikki särmät. Merkitään vastaavasti $1 \cdot C = C$. Olkoon kahden syklin summa $C_1 + C_2 = C_1 \cup C_2 \setminus \{\vec{e} \mid \vec{e} \in C_1 \cup C_2 \text{ ja } \overleftarrow{e} \in C_1 \cup C_2\}$.

Olkoon B jokin verkon G syklikanta, jolloin jokainen G :n sykli voidaan lausua summana $C_k = \sum_{i=1}^{k_m} a_{k_i} C_{k_i}$, missä $C_{k_i} \in B$ ja $a_{k_i} \in \{-1, 1\}$ kaikilla $i \in \{1, 2, \dots, k_m\}$.

Ajatellaan, että jokaiseen särmään $\vec{e}_k = p_i p_j \in \vec{E}$ on liitetty viiveen määritelmässä esiintyvä kahden naapurisolmujen kellorekisterien arvojen erotus $\delta_k = r_{p_j} \ominus r_{p_i}$. Silloin polun $\mu = \vec{e}_0 \vec{e}_1 \dots \vec{e}_k$ viive voidaan määritellä summana

$$\Delta(\mu) = \sum_{i=0}^k a_i \delta_i,$$

missä $a_i = 1$, jos $\vec{e}_i \in \vec{E}$ ja $a_i = -1$ muuten. Viiveen lineaarisuuden perusteella

jokaisen syklin viive voidaan ilmaista syklikannan avulla muodossa

$$\Delta(C_k) = \Delta\left(\sum_{i=1}^{k_m} a_{k_i} C_{k_i}\right) = \sum_{i=1}^{k_m} a_{k_i} \Delta(C_{k_i}).$$

Jos viive on kaikissa syklikannan sykleissä 0, viive on kaikissa verkon sykleissä 0. Toisaalta jos viive on 0 kaikissa sykleissä, se on 0 myös 0 kaikissa syklikannan sykleissä, koska kaikki syklikannan syklit ovat verkon syklejä. Näin on saatu tulos, jonka mukaan viive on rajoitettu järjestelmässä täsmälleen silloin, kun viive on 0 kaikissa verkon mielivaltaisen syklikannan sykleissä.

Määritelmä 2 Verkon syklikarakteristiikka C_G . Jos G on sykkitön verkko, sen syklikarakteristiikka C_G on 2. Olkoon muuten Λ eräs verkon syklikanta ja $\lambda(\Lambda)$ syklikannan Λ pisin sykli. Silloin C_G on pienin $\lambda(\Lambda)$ kaikkien G :n syklikantojen joukossa.

Lause 3 Jos $K > C_G$ ja järjestelmän käynnistyskonfiguraatio on kehälle vakiintunut, geneerinen algoritmi ja asynkroninen unisono eivät voi lukkiutua.

Jos G on puu, K on vähintään 3, mikä vaaditaan askeltavan järjestelmän määrittämiseksi. Syklittömässä verkossa viive on aina rajoitettu, ja lauseen 1 mukaan väite on tosi.

Oletetaan nyt, että verkossa G on ainakin yksi sykli. Olkoon Λ_G verkon minimaalinen syklikanta, eli $\lambda(\Lambda_G) = C_G > 2$. Kuulukoon sykli $\mu = p_0 p_1 \dots p_k$, missä $p_k = p_0$, joukkoon Λ_G . Koska järjestelmä on kehälle vakiintunut, kahden naapurisolmun välisen K -etäisyyden itseisarvo on korkeintaan 1, ja siten

$$|\Delta_\mu| = \left| \sum_{i=0}^{k-1} (r_{p_{i+1}} \ominus_l r_{p_i}) \right| \leq C_G.$$

Koska $\Delta_\mu \equiv 0 \pmod{K}$ ja $K > C_G$, niin $\Delta_\mu = 0$. Siis viive on rajoitettu, eikä järjestelmä voi lukkiutua. \square

5 Palautusvaihe ja vakautuvuus

Tässä luvussa näytetään ensin, että kaikki lukkiutuneet konfiguraatiot ovat kehälle vakiintuneita. Uuden käsitteen *palautusverkko* avulla todistetaan, että järjestelmän suoritusketjussa voi olla vain äärellinen määrä palautuksia. Tämän jälkeen voidaan osoittaa, että algoritmi on vakautuva kehälle vakiintuneisuuden suhteen.

5.1 Lukkiutunut järjestelmä

Lause 4 Jos järjestelmä on lukkiutunut, se on kehälle vakiintunut.

Jos verkossa on vain yksi solmu, väite on tosi. Oletetaan sitten, että solmuja on useampi kuin yksi ja että järjestelmä ei ole kehälle vakiintunut. Jaetaan mahdolliset tapaukset kahteen luokkaan.

1. Oletetaan, että on olemassa vähintään yksi prosessi, jonka kellorekisterin arvo kuuluu joukkoon $putki_\varphi^*$. Olkoon p näistä prosesseista se, jonka kellorekisterin arvo on pienin. Olkoon $q \in \mathcal{N}_p$ p :n naapuri. Silloin täytyy olla $r_q \in putki_\varphi$, muutenhan ehto $palautusaskel(q)$ olisi tosi. Mutta nyt $putkiaskel(p)$ on tosi, eli tilanne ei ole lukkiutunut. Päädyttiin ristiriitaan.
2. Oletetaan kaikkien prosessien kellorekisterien arvojen kuuluvan joukkoon $kehä_\varphi$. Koska järjestelmä ei ole kehälle vakiintunut, on olemassa naapuriprosessit p ja q siten, että $askelehto(p, q)$ on epätosi ja $askelehto(q, p)$ on epätosi. Tästä seuraa, että $naapurustoehto(p)$ on epätosi. Oletuksen mukaisesti $r_p \in kehä_\varphi$. Ehto $palautusaskel(p)$ ei ole voimassa lukkiutuneisuusoletuksen perusteella, joten täytyy olla $r_p = 0$. Symmetrian perusteella voidaan päätellä, että myös $r_q = 0$. Algoritmiin 1 liittyvän oletuksen \mathcal{O}_2 perusteella $pariehto(p, q)$ on tosi. Ristiriita. \square

5.2 Palautusverkko

Määritelmä 5 Palautus. *Olkoon $e = \gamma_0\gamma_1\dots$ algoritmin 1 suoritusketju. Palautus on pari (p, t) , jossa p on prosessi ja $t > 0$ sellainen indeksi, että p suorittaa ohjelmaskelen PAL siirtymässä $\gamma_{t-1} \mapsto \gamma_t$. Tällöin $r_p \notin putki_\varphi$ konfiguraatiossa γ_{t-1} ja $r_p = \alpha$ konfiguraatiossa γ_t . Sanotaan, että p palautetaan arvoon α konfiguraatiossa γ_t (tai hetkellä t).*

Olkoot (p_1, t_1) ja (p_2, t_2) kaksi palautusta. Sanotaan, että (p_1, t_1) aiheuttaa palautuksen (p_2, t_2) , jos ja vain jos seuraavat kolme ehtoa ovat tosia:

1. $t_1 < t_2$ ja $p_2 \in \mathcal{N}_{p_1}$,
2. $\forall t \in [t_1, t_2 - 1] : r_{p_2} \notin putki_\varphi$,
3. $r_{p_2} = \alpha$ konfiguraatiossa γ_{t_2} .

Tätä merkitään $(p_1, t_1) \overset{r}{\rightsquigarrow} (p_2, t_2)$.

Koska $(p_1, t_1) \overset{r}{\rightsquigarrow} (p_2, t_2)$ implikoi $t_1 < t_2$, relaatio $\overset{r}{\rightsquigarrow}$ määrittelee suunnatun syklittömän verkon, jota kutsutaan *palautusverkoksi*. Jos mikään palautus ei ole aiheuttanut palautusta (p_1, t_1) , sanotaan palautuksen (p_1, t_1) olevan *alkupalautus*. *Palautussilmukka* on palautusverkon polku $(p_0, t_0)(p_1, t_1) \dots (p_k, t_k)$, jonka projektio $p_0p_1 \dots p_k$ on sykli.

Lemma 6 *Suoritusketjussa on olemassa prosessia p kohden korkeintaan yksi t niin, että (p, t) on alkupalautus.*

Oletetaan väitteen vastaisesti, että on olemassa p, t_1, t_2 siten, että (p, t_1) ja (p, t_2) ovat kumpikin alkupalautuksia. Tarkastellaan suoritusketjua $\gamma_{t_1}\gamma_{t_1+1}\dots\gamma_{t_2}$. Koska (p, t_1) ja (p, t_2) ovat alkupalautuksia, palautuksen määritelmän mukaisesti $r_p = \alpha$ konfiguraatiossa γ_{t_1} , ja $r_p \notin \text{putki}_\varphi$ (eli $r_p > 0$) konfiguraatiossa γ_{t_2-1} . Siis on olemassa suurin $t_3 < t_2 - 1$ siten, että hetkellä t_3 *askelehto*(p) ja siten myös *naapurustoehdo*(p) on tosi ja että hetkellä t_3+1 *naapurustoehdo*(p) on epätosi. Niinpä ainakin yksi p :n naapuri q suoritti ohjelma-askelen *PAL* siirtymässä $\gamma_{t_3} \mapsto \gamma_{t_3+1}$. Toisin sanoen $(q, t_3 + 1) \overset{r}{\rightsquigarrow} (p, t_2)$. Ristiriita. \square

Polku, joka on muotoa $(p, t_1)(q, t_2)(p, t_3)$, on *palautuskieppi*.

Lemma 7 *Palautusverkossa ei ole yhtään palautuskieppiä.*

Oletetaan väitteen vastaisesti, että verkossa on palautuskieppi $(p, t_1)(q, t_2)(p, t_3)$. $(p, t_1) \overset{r}{\rightsquigarrow} (q, t_2)$ niinpä $\forall t \in [t_1, t_2 - 1]$ pätee $r_q \notin \text{putki}_\varphi$. Solmut p ja q ovat naapureita, joten koska hetkellä t_1 $r_p = \alpha$, on myös $r_p = \alpha$ kaikilla $t \in [t_1, t_2]$. Koska $(q, t_2) \overset{r}{\rightsquigarrow} (p, t_3)$ merkitsee, että $r_p \notin \text{putki}_\varphi$ kaikilla $t \in [t_2, t_3 - 1]$. Päädyttiin ristiriitaan. \square

Määritelmä 8 *Mikäli jokaisessa palautusverkon rei'ässä $p_1p_2\dots p_ip_1$ pätee*

$$(p_1, t_1) \overset{r}{\rightsquigarrow} (p_2, t_2) \overset{r}{\rightsquigarrow} \dots \overset{r}{\rightsquigarrow} (p_i, t_i) \Rightarrow (p_1, t_1) \overset{r}{\rightsquigarrow} (p_i, t_i),$$

sanotaan palautusverkon olevan transitiivinen rei'issä.

Lause 9 *Jokainen rei'issä transitiivinen palautusverkko on äärellinen. Jos järjestelmäverkko G on sykliiton, sen jokainen palautusverkko on äärellinen.*

Jos palautusverkossa on palautussilmukka, nähdään helposti, että verkossa on myös palautussilmukka, jonka projektio järjestelmäverkossa on reikä. Niinpä jos rei'issä transitiivisessa palautusverkossa on palautussilmukka, siinä on polku

$$(p_1, t_1)(p_2, t_2)\dots(p_i, t_i)(p_1, t_{i+1}),$$

missä $p_1p_2\dots p_ip_1$ on verkon G reikä. Transitiivisuusominaisuuden tähden $(p_1, t_1) \overset{r}{\rightsquigarrow} (p_i, t_i)$, ja koska $(p_i, t_i) \overset{r}{\rightsquigarrow} (p_1, t_{i+1})$, verkossa on palautuskieppi. Tämä on kuitenkin mahdotonta lemmän 7 mukaan, joten rei'issä transitiivinen palautusverkko ei voi sisältää palautussilmukkaa. Luonnollisesti myöskään sykliittömän järjestelmäverkon tapauksessa palautussilmukkaa ei ole. Koska järjestelmäverkko on äärellinen, niin lemmasta 6 seuraa, että palautusverkossa on äärellinen määrä alkupalautuksia. Siten palautusverkko on äärellinen. \square

5.3 Vakautuvuus ja yläraja parametrin α arvolle

Lause 10 *Jos palautusverkko on äärellinen, niin algoritmi 1 on vakautuva kehälle vakiintuneisuuden suhteen.*

Olkoon e suoritusketju. Jos e on äärellinen, viimeinen konfiguraatio on kehälle vakiintunut lauseen 4 mukaan. Oletetaan sitten, että e on ääretön. Koska palautusverkko on äärellinen, suoritusketjussa on loppuosa e' , jossa ei tapahdu palautuksia.

Olkoon R äärettömän kauan palautusvaiheessa pysyvien prosessien joukko. Oletetaan, että R on epätyhjä. Suoritusketjulla e' on loppuosa e'' , jossa joukon R prosessien kellorekisterien arvot eivät muutu. Joukosta R yhden kaaren päässä olevat prosessit ovat silloin lukkiutuneita. Kahden kaaren päässä olevat prosessit voivat muuttaa kellorekisterinsä arvoa korkeintaan kaksi kertaa. Induktiolla voidaan näyttää, että etäisyydellä k olevat prosessit voivat muuttaa rekisterinsä arvoa korkeintaan $2(k-1)$ kertaa. Mutta koska järjestelmäverkko G on äärellinen, e'' ei voi olla ääretön. Päädyttiin ristiriitaan, joten R on tyhjä.

Olkoon nyt e_0 e' :n loppuosa, jossa mikään prosessi ei ole palautusvaiheessa. Olkoon suoritusketjussa e_0 jollakin hetkellä joukko $R' = \{p \in V \mid \neg \text{naapurustoehto}(p)\}$ epätyhjä. Tällöin R' on äärettömän kauan epätyhjä, sillä jos $p \in R'$, niin jollakin $q \in \mathcal{N}_p$ pätee $\neg \text{pariehto}(p, q)$ eli yhtäpitävästi $\neg \text{pariehto}(q, p)$. Tilanne ei voi muuttua, koska suoritusketjussa e_0 ei tapahdu palautuksia. Joukosta R' etäisyydellä 1 olevat prosessit voivat muuttaa kellolaskurinsa arvot korkeintaan kahdesti. Induktiivisesti voidaan päätellä, että etäisyydellä k olevat prosessit voivat muuttaa kellorekisterinsä arvoa korkeintaan $2k$ kertaa. Koska järjestelmäverkko G on äärellinen, e_0 ei voi olla ääretön. Päädyttiin ristiriitaan, joten R' on tyhjä.

Jokaisella suoritusketjulla e on siis nollaa pidempi loppuosa, jonka jokainen konfiguraatio on kehälle vakiintunut. \square

Merkitään seuraavassa $\varphi^0(x) = x$ ja $\varphi^j(x) = \overbrace{(\varphi \circ \varphi \circ \dots \circ \varphi)}^{j \text{ kpl}}(x)$.

Lemma 11 *Olkoon $(p_0, t_0)(p_1, t_1) \dots (p_i, t_i)$, $i \in \mathbb{N}$ polku palautusverkossa. Silloin kaikilla $t \in [t_{i-1}, t_i[$ pätee $r_{p_0} \in \{\varphi^j(\alpha) \mid j \in \{0, 1, \dots, i-1\}\}$.*

Todistetaan väite induktiolla polun pituuden suhteen. Jos $i = 1$ väite on tosi palautuksen määritelmän perusteella. Oletetaan väite todeksi, kun polun pituus on i . Tarkastellaan polkua $(p_0, t_0)(p_1, t_1) \dots (p_{i+1}, t_{i+1})$, jonka pituus on $i + 1$. Induktiioletuksen perusteella kaikilla $t \in [t_{i-1}, t_i[$ pätee

$$\begin{aligned} r_{p_0} &\in \{\varphi^j(\alpha) \mid j \in \{0, 1, \dots, i-1\}\} \\ r_{p_1} &\in \{\varphi^j(\alpha) \mid j \in \{0, 1, \dots, i-2\}\} \\ &\dots \\ r_{p_{i-1}} &\in \{\varphi^j(\alpha) \mid j \in \{0\}\} . \end{aligned}$$

Hetkellä t_i prosessi p_i palautetaan, ja $r_{p_i} = \alpha$ ainakin hetken t_{i+1} asti. Siten p_{i-1} voi muuttaa kellorekisterinsä arvoa hetken t_i jälkeen korkeintaan kerran, ja kaikilla $t \in [t_i, t_{i+1}[$ pätee $r_{p_{i-1}} \in \{\varphi^j(\alpha) \mid j \in \{0, 1\}\}$. Niinpä samalla aikavälillä naapuriprosessin p_{i-2} kellorekisterin arvo $r_{p_{i-2}}$ kuuluu joukkoon $\{\varphi^j(\alpha) \mid j \in \{0, 1, 2\}\}$. Päättyä jatkamalla nähdään, että $\forall t \in [t_i, t_{i+1}[$ on $r_{p_0} \in \{\varphi^j(\alpha) \mid j \in \{0, 1, \dots, i\}\}$. Induktioperiaatteen mukaisesti väite pätee siis kaikilla $i \in \mathbb{N}$. \square

Määritellään verkkoon G liittyvä vakio T_G . Jos G on puu, $T_G = 2$, muuten olkoon T_G yhtä suuri kuin verkon G pisimmän reiän ympäräysmitta.

Lause 12 Jos $|\alpha| \geq T_G - 2$, algoritmi 1 on vakautuva kehälle vakiintuneisuuden suhteen.

Jos järjestelmäverkko on puu, lauseen 9 mukaan palautusverkko on äärellinen. Silloin lauseen 10 perusteella väite on tosi.

Oletetaan sitten, että verkossa G on syklejä. Tarkastellaan palautusverkon polkua $(p_0, t_0)(p_1, t_1) \dots (p_i, t_i)$, missä $p_0 p_1 \dots p_i p_0$ on reikä verkossa G . Lemman 11 mukaan

$$\begin{aligned} \forall t \in [t_0, t_1[: r_{p_0} &\in \{\varphi^j(\alpha) \mid j \in \{0\}\} \\ \forall t \in [t_1, t_2[: r_{p_0} &\in \{\varphi^j(\alpha) \mid j \in \{0, 1\}\} \\ &\dots \\ \forall t \in [t_{i-1}, t_i[: r_{p_0} &\in \{\varphi^j(\alpha) \mid j \in \{0, 1, \dots, i-2\}\} \end{aligned}$$

Koska $i < T_G$ ja $|\alpha| \geq T_G - 2$, niin $i - 2 < |\alpha|$, ja $\forall t \in [t_0, t_i[: r_{p_0} \in \{\alpha, \alpha + 1, \dots, k\}$, missä $k < 0$. Koska $(p_{i-1}, t_{i-1}) \xrightarrow{r} (p_i, t_i)$, niin kaikilla $t \in [t_{i-1}, t_i[$ pätee $r_{p_i} \notin \text{putki}_\varphi$. Lisäksi koska solmu p_0 on solmun p_i naapuri, täytyy olla $\forall t \in [t_0, t_i[: r_{p_i} \notin \text{putki}_\varphi$. Jos nimittäin jollakin $t \in [t_0, t_{i-1}[$ olisi $r_{p_i} \in \text{putki}_\varphi$, prosessi p_i ei olisi naapurinsa p_0 takia voinut kasvattaa kellorekisterinsä arvoa nolaa suuremmaksi hetkeen $t - 1$ mennessä. Koska siis $\forall t \in [t_0, t_i[: r_{p_i} \notin \text{putki}$ ja hetkellä t_i prosessi p_i palautetaan, niin $(p_0, t_0) \xrightarrow{r} (p_i, t_i)$. Palautusverkko on siis transitiivinen rei'issä, palautusverkko siten äärellinen ja lauseen 10 perusteella väite tuli todistetuksi. \square

Lähteet

- BPV04 Boulinier, C., Petit, F. ja Villain, V., When graph theory helps self-stabilization. *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, New York, NY, USA, 2004, ACM Press, sivut 150–159.
- Kir47 Kirchhoff, G., Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. *Annalen der Physik und Chemie*, 148, sivut 497–508.
- KY02 Kakugawa, H. ja Yamashita, M., Self-stabilizing local mutual exclusion on networks in which process identifiers are not distinct. 2002, sivut 202–211.