Detekcia chýb

Cieľ: identifikácia procesov, ktoré prestali pracovať

T je čas. $F(\tau)$ označuje množinu procesov, ktoré v čase τ nepracujú (majú fatálnu chybu).

$\tau_1 \leq \tau_2 \Rightarrow F(\tau_1) \subseteq F(\tau_2)$ **žiaden reštart**

$Crash(F) = \bigcup_{\tau \in T} F(\tau)$ $Corr(F) = P - Crash(F)$

$\mathbf{H(p,\tau)}$ je množina procesov, o ktorých procesor p **predpokladá**, že v čase τ majú fatálnu chybu.

Každému výpočtu priradíme vzor chýb F a históriu detektora chýb H.

Vyžadujeme, aby **detektor chýb** bol **úplný**: Ak niektorý procesor prestane pracovať, v konečnom čase ho každý korektný procesor bude považovať za chybný.

$$\exists \tau \forall p \in Crash(F)\ \forall q \in Corr(F)\ \forall \tau' \geq \tau\ \ p \in H(q, \tau')$$

Implementácia detektorov chýb

detektor chýb je **(úplne) presný**, ak podozrieva len naozaj chybné procesy:

$$\forall \tau \; \forall p, q \notin F(\tau) \;\; p \notin H(q, \tau)$$

detektor chýb je **oneskorene (úplne) presný** ak od nejakého času podozrieva len naozaj chybné procesy:

$$\exists \tau \; \forall \tau' \geq \tau \; \forall p, q \notin Crash(F) \;\; p \notin H(q, \tau')$$

(Úplne) presná detekcia

Predpoklady:

- každý korektný proces rozpošle každých σ časových jednotiek správu `alive`
- existuje horný odhad μ na komunikačné zdržanie

Každý proces, od ktorého neprišla správa počas $\sigma+\mu$ časových jednotiek, prestal pracovať.

Tento detektor chýb je *úplný* a *úplne presný*.

Oneskorene (úplne) presná detekcia

Predpoklady:

- každý korektný proces rozpošle každých σ časových jednotiek správu `alive`
- existuje (*neznámy*) horný odhad na komunikačné zdržanie

Úplný a **oneskorene úplne presný** detektor
– každý proces p začína s $\mu_p = 1$

– ak p nedostane žiadnu správu od q počas $\sigma + \mu_p$ časových jednotiek, p podozrieva q

– ak p prijalo správu od podozrivého procesu q, q prestane byť podozrivým a p si prestaví $\mu_p \leftarrow \mu_p + 1$

Detektor chýb je **slabo presný**, ak *existuje korektný* proces, ktorý sa nikdy nestane podozrivým:

$$\exists p \notin Crash(F)\ \forall \tau \forall q \notin F(\tau)\ \ p \notin H(q, \tau)$$

Detektor chýb je **oneskorene slabo presný**, ak *existuje korektný* proces, ktorý sa od nejakého časového okamihu nikdy nestane podozrivým:

$$\exists \tau \exists p \notin Crash(F)\ \forall \tau' \geq \tau\ forall q \notin Crash(F)\ \ p \notin H(q, \tau')$$

Dohoda s detektorom chýb

(N-1)-f-robustný algoritmus dohody s rotáciou koordinátora

– **slabo presný** detektor chýb

– procesy sú očíslované $p_1, \ldots, p_N$, iniciované na hodnotu 0 alebo 1

k-te kolo — koordinátor p_k

- Korektný proces p_k spustí broadcast svojej hodnoty
- Každý proces čaká:
  - príchod správy od p_k; ak ju dostane, prevezme prijatú hodnotu
  - kým nezačne podozrievať p_k, že prestal pracovať

Po N kolách každý korektný proces rozhodne svoju hodnotu.

Veta 1 *Nech $t \geq N/2$. Potom* ***neexistuje*** *t-f-robustný algoritmus dohody založený na* ***oneskorene úplne presnom*** *detektore chýb.*

– **disjunktné** množiny S, T: $|S| = |T| = N - t$;

– $S(T)$ dostatočne dlho podozrieva procesy z $T(S)$, že prestali pracovať

– v S, T môže nezávisle dôjsť k rozhodnutiu $0, 1$

Chandra-Toueg algoritmus dohody– t-f-robustný algoritmus dohody s rotujúcim koordinátorom $t < N/3$, **oneskorene slabo presný** detektor chýb

Kolo k koordinuje proces p_c, pričom $c = (k \ mod \ N) + 1$

- q pošle $\langle \mathbf{val_q}, \mathbf{k} \rangle$ procesu p_c
- p_c počká prijatie $N - t$ správ tohto kola; prestaví val_{p_c} podľa väčšiny (pri rovnosti 1)
  Ak sú všetky prijaté správy rovnaké, nastaví agr_{p_c} na T, inak na F
- p_c spustí broadcast $\langle \mathbf{agr_{p_c}}, \mathbf{val_{p_c}}, \mathbf{k} \rangle$
- Každý proces q čaká:
  - alebo na správu od p_c - vtedy preberie prijatú hodnotu a ak $agr_{p_c} = T$, q rozhodne
  - alebo dovtedy, kým začne podozrievať p_c

Veta 2 *Algoritmus CH-T terminuje s dohodou.*

1. Ak koordinátor kola k nie je podozrivý, $\forall$ korektné procesy sa zhodnú na b
2. Potom má $\geq N - t$ (korektných) procesov rovnakú hodnotu b, takže v kolách $l, l > k$ má koordinátor aspoň $N - 2t > \frac{N-t}{2}$ hlasov b a spustí broadcast s hodnotou b
3. Ak koordinátor niektorého kola l, $l > k$, nie je podozrivý, $\forall$ korektné procesy rovnako rozhodnú b

Toto je zjednodušená verzia, pôvodná verzia je funkčná pre $t < N/2$.

Stabilizácia

nevydeľujeme počiatočné konfigurácie, všetky konfigurácie sú počiatočné

Systém S **stabilizuje k špecifikácii** P, ak existuje podmnožina legitímnych konfigurácií $\mathcal{L}$ taká, že:

korektnosť – každá realizácia, ktorá začína v konfigurácii z $\mathcal{L}$, spĺňa P

konvergencia – každá realizácia obsahuje konfiguráciu z $\mathcal{L}$

Výhody

Odolnosť voči chybám – spamätajú sa z chyby

Inicializácia – netreba sa venovať precíznej inicializácii

Robustnosť pre dynamické topológie – systémy, ktoré počítajú funkcie závislé od topológie, sa vedia so zmenou topológie vysporiadať

Nevýhody

počiatočná nekonzistencia- kým nezačne pracovať korektne, môže dávať nesprávne výstupy

veľká zložitosť

chýba detekcia stabilizácie- procesy teda nevedia, kedy pracujú korektne...

Komunikácia

- predpokladáme **zdieľané premenné** (jeden píše, druhý číta)
- modely so stavom - proces číta stav svojich susedov
- spojené registre - medzi procesmi dva registre (do jedného píšem, z druhého čítam)

scheduler/démon určuje, kto realizuje krok – strong, rsp. weak

central – len jeden proces

k-fair – jeden proces nanajvýš $k-$krát po sebe

weakly-fair – ak je neustále pripravený, v konečnom čase dostane slovo

distributed unfair – ľubovoľná podmnožina (pripravených) procesov

Problém vzájomného vylúčenia - každý proces musí niekedy vykonať kritickú oblasť, ale vždy je v kritickej oblasti nanajvýš jeden; chránené predikátom:

1. v každej konfigurácii má nanajvýš jeden proces privilégium
2. každý proces je privilegovaný nekonečne veľa razy

Dijkstra - token v kruhu

riešenie pre stratu, resp. násobný výskyt tokena v kruhu

Procesy $p_0, \ldots, p_{N-1}$ tvoria orientovaný kruh; p_i má hodnotu $\sigma_i \in \{0, \ldots, K\}$, kde $K > N$

- p_i, pre ktoré $0 < i < N$, je privilegovaný ak $\sigma_{i-1} \neq \sigma_i$
- p_0 je privilegovaný ak $\sigma_0 = \sigma_{N-1}$

Každý privilegovaný proces môže zmeniť svoju hodnotu, čo spôsobí stratu privilégia

- $\sigma_i \leftarrow \sigma_{i-1}$, ak $\sigma_i \neq \sigma_{i-1}$, $0 < i < N$
- $\sigma_0 \leftarrow (\sigma_{N-1} + 1) \mod K$ keď $\sigma_0 = \sigma_{N-1}$

Veta 3 *Dijkstrov algoritmus stabilizuje k vzájomnému vylúčeniu*

D: V $\forall$ konfig. má privilégium aspoň jeden proces. Krok $\sharp$ privilegovaných nezvyšuje.
$\mathcal{L}$ — množina konfigurácií, v ktorých má privilégium práve jeden proces.

Výpočet, kt. začína v legitímnej konfigurácii, spĺňa vzájomné vylúčenie - token cirkuluje po kruhu

- $\sigma_i \leftarrow \sigma_{i-1}$, ak $\sigma_i \neq \sigma_{i-1}$, $0 < i < N$
- $\sigma_0 \leftarrow (\sigma_{N-1} + 1) \mod K$ keď $\sigma_0 = \sigma_{N-1}$

Dijkstrov kruh s tokenom konverguje k $\mathcal{L}$

— Po max $\frac{1}{2}(N-1)N$ udalostiach v $p_1, \ldots, p_{N-1}$ musí nastať udalosť aj v p_0.

— **Norma** $F = \sum_{i \in S}(N - i)$ $\quad S = \{i, i > 1$ a proces i má privilégium $\}$

— Hodnota F klesá s každým krokom v procese inom ako p_0

— $\gamma_0 \in I$ obsahuje max N rôznych stavov, $K > N \Rightarrow K - N$ stavov v nej nie je.

— p_0 prechádza cez stavy z K, $p_1, \ldots, p_{N-1}$ kopírujú stavy $\Rightarrow \exists$ konfigurácia γ, $\sigma_0 \neq \sigma_i$, $0 < i < N$

— $\gamma \rightarrow \ldots \rightarrow \delta$, pričom v δ platí: $\sigma_0 = \sigma_1 = \cdots = \sigma_{N-1}$

Počet správ – Vzájomné vylúčenie sa dosahuje po $O(N^2)$ udalostiach

proces $p_i, 0 < i < N$ kopíruje počiatočné hodnoty $p_0, \ldots, p_{i-1}$

(spolu: $\leq \frac{1}{2}(N-1)N$ udalostí)

proces p_0 nadobudne nanajvýš N hodnôt, kým dosiahne *novú* hodnotu. Tie hodnoty sa môžu kopírovať do $p_1, \ldots, p_{N-1}$

(spolu: $\leq N^2$ udalostí)

scheduler/démon určuje, kto realizuje krok – strong, rsp. weak

central – len jeden proces
k-fair – jeden proces nanajvýš $k-$krát po sebe
weakly-fair – ak je neustále pripravený, v konečnom čase dostane slovo
distributed unfair – ľubovoľná podmnožina pripravenýchprocesov

Local Mutual Exclusion

Strong safety – v každej konfigurácii je aspoň jeden privilegovaný proces

Fairness index k – $\forall$ typ démona platí, že medzi dvomi po sebe idúcimi realizáciami kritickej sekcie procesora p žiaden proces nevstúpi do kritickej sekcie viac ako $k-$krát

Service time – celkový počet realizácií kritických sekcií ostatných procesov medzi dvomi nasledujúcimi realizáciami kritickej sekcie jedného procesu

$\triangleright$ – relácia "susednosti" nad zdieľanými premennými x

virtuálna orientácia – hrana medzi p a q je virtuálne orientovaná $\overrightarrow{\textbf{z } p \textbf{ do } q}$ **iff** $x.p \triangleright x.q$

privilegovaný proces p – v komunikačnom grafe sú $\forall$ hrany virtuálne orientované k p

Algoritumus ULME - unbounded local mutual exclusion

konštanty $id.p$(identifikátor), $N.p$(susedia)
zdieľané premenné $L.p$ - celé číslo **(!)**
lokálne premenné CS - bool. flag nad kritickou sekciou
funkcia $(\forall q \in N.p)\ p \prec q \equiv L.q \rhd L.p$
akcie
$A: \quad \forall q \in N.p,\ p \prec q$
$CS \leftarrow 1$; realizuj ktirickú sekciu; $L.p \leftarrow \max\{L.q \mid q \in N.p\} + 1$

def $\mathbf{L.p \rhd L.q}$ iff $(\mathbf{L.p < L.q}) \vee ((\mathbf{L.p = L.q}) \wedge (\mathbf{id.p < id.q}))$

$\Rightarrow$ $\rhd$ je úplné usporiadanie

legitímna konfigurácia
(i) aspoň jeden privilegovaný
(ii) neexistujú privilegovaný susedia

Fakt Komunikačný graf pre systém s ULME je acyklický.

vzdialenosť medzi p, q - # procesorov na najkratšej (p-q)-ceste +1

$SPdist_i.p$, $|SPdist_i.p|$ - množina procesov vo vzd. i a jej mohutnosť

- medzi dvomi po sebe idúcimi realizáciami KS v p realizujú všetci jeho susedia KS práve raz
- medzi dvomi po sebe idúcimi realizáciami KS v p môže $q \in SPdist_i.p$ realizovať KS nanajvýš $i-$krát
- každý výpočet je nekonečný
- každý proces je privilegovaný nekonečne veľa razy
- každý výpočet v konečnom čase dosiahne legitímnu konfiguráciu

Veta 4 *Algoritmus ULME je self-stabilizing, pričom*
fairness index je $(n-1)$
service time je $\leq \frac{n(n-1)}{2}$

cyklické porovnanie – Nech $x, y \leq B,\ B \geq 2$

Definujme $\rhd$:
$\forall x \in [0, \frac{B}{2}]$
(1.) $y \rhd x$ iff $y \in [x+1, x+\frac{B}{2}]$
(2.) $x \rhd y$ iff $y \in [\frac{B}{2}+x+1, B-1] \cup [0, x-1]$

$\forall x \in [\frac{B}{2}+1, B-1]$
(1.) $y \rhd x$ iff $y \in [x+1, B-1] \cup [0, x+\frac{B}{2}]$
(2.) $x \rhd y$ iff $y \in [x+\frac{B}{2}+1, x-1] \cup [0, x-1]$

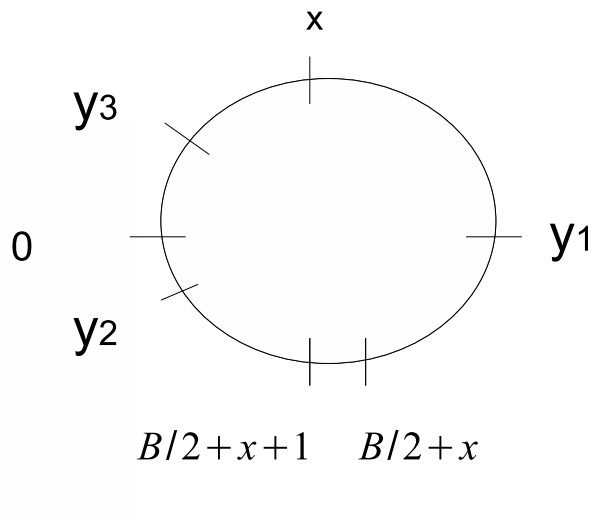
ohraničená virtuálna orientácia – virtuálna orientácia s (vyššie) def. $\rhd$ a L ohraničeným n^2(n párne), resp. n^2+1 (n nepárne)

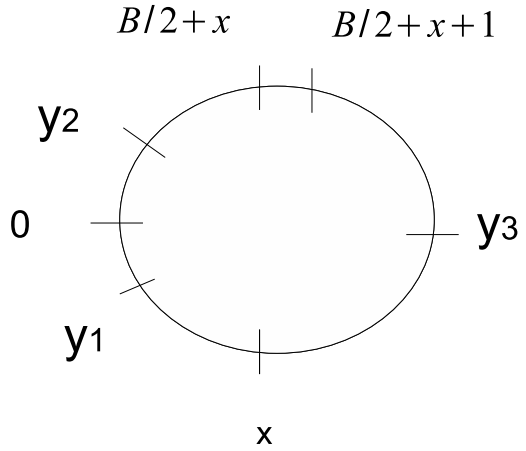
balansované procesy p, q – $p \sim q$ iff $|L.p - L.q| < n \wedge ((L.p \neq L.q) \vee (L.p = L.q = 0))$

balansovaná konfigurácia – všetky procesy balansované vzhľadom k susedom

nebalansované procesy – existuje dvojica, ktoré nie sú baansované $p \nsim q$

nebalansovaná konfigurácia – existujú nebalansované procesy





Algoritmus BLME

funkcie

$MaxL(p) \equiv L.i;\ |L.i - L.p| \mod B = \max(|L.j - L.p| \mod B, \forall j \in N.p)$

$\forall i \in N.p:\ p \prec i \equiv (L.i \rhd L.p) \vee ((L.p = L.i = 0) \wedge (id.p < id.i))$

$(\forall i \in N.p):\ p \sim i \equiv ((|L.p - L.i| \mod B) < n) \wedge ((L.p \neq L.i) \vee (L.p = L.i = 0))$

akcie

$A_1: (R.p = 0) \wedge (\forall i \in N.p, p \prec i \wedge p \sim i \wedge R.i = 0) \longrightarrow$
$CS \leftarrow 1$; realizuj KS; $L.p = (MaxL(p) + 1) \mod B$

$R_1: (R.p = 0) \wedge (\exists i \in N.p, p \nsim i \vee R.i = 1) \wedge (L.i \neq 0 \vee L.p \neq 0)) \longrightarrow$
$CS \leftarrow 0; R.p = 1$

$R_2: (R.p = 1) \wedge (L.p \neq 0) \wedge (\forall i \in N.p,\ R.i = 1) \longrightarrow$
$CS \leftarrow 0; L.p = 0$

$R_3: (R.p = 1) \wedge (L.p = 0) \wedge (\forall i \in N.p,\ L.i = 0) \longrightarrow$
$CS \leftarrow 0; R.p = 0$

- komunikačný graf systému s BLME je acyklický
- ak je systém balansovaný, existuje privilegovaný proces a žiadne dva susedné procesy nie sú privilegované

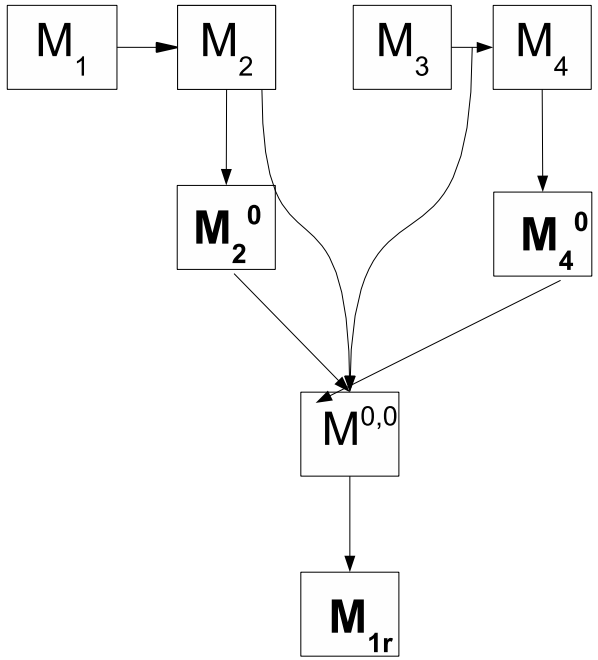
legitímna konfigurácia

- systém je v balansovanej konfigurácii
- reset marker R je v každom procese nastavený na 0

- v legitímnej konfigurácii procesy realizujú len akciu A
- každá konfigurácia dosiahnuteľná z legitímnej konfigurácie je legitímna
- ak výpočet začína v nelegitímnej konfigurácii s $MT \neq M_1$, tak niekedy nastane jedna z akcií $\{R_1, R_2, R_3\}$
- ak výpočet začína v nelegitímnej konfigurácii s $MT \neq M_1$, tak niekedy budú všetky hrany G patriť do M_{1r}

$\Rightarrow$ Algoritmus BLME je stabilizujúci

- $M_1 = \{(p,q) \mid p \sim q, (R.p = 0 \wedge R.q = 0)\}$
- $M_2 = \{(p,q) \mid p \sim q, (R.p = 1 \vee R.q = 1)\}$
- $M_3 = \{(p,q) \mid p \nsim q, (R.p = 0 \vee R.q = 0)\}$
- $M_4 = \{(p,q) \mid p \nsim q, (R.p = 1 \wedge R.q = 1)\}$
- $MT = \bigcup_{i=1}^{4} M_i$
- $\forall i \in \{2,3,4\}$:
  $M_i^0 = \{(p,q) \mid (p,q) \in M_i \wedge (L.p = 0 \wedge L.q = 0)\}$
- $M^{(00)} = \{(p.q) \mid L.p = 0 \wedge L.q = 0\}$



Maximálne párovanie

$\mathbf{pref_p}$ — informáciu o susedovi, s ktorým chce tvoriť hranu z maximálneho párovania

$\mathbf{wait(p)} \equiv pref_p = q \in Neigh_p \wedge pref_q = nil$ waiting

$\mathbf{match(p)} \equiv pref_p = q \wedge pref_q = p$ matching

$\mathbf{chain(p)} \equiv pref_p = q \in Neigh_p \wedge pref_q = r \in Neigh_q \wedge p \neq r$ chaining

$\mathbf{dead(p)} \equiv pref_p = nil \wedge \forall q \in Neigh_p : match(q)$ dead

$\mathbf{free(p)} \equiv pref_p = nil \wedge \exists q \in Neigh_p : \neg match(q)$ free

$\mathbf{M_p}$: $\{pref_p = nil \wedge pref_q = p\}$
$pref_p := q$

$\mathbf{S_p}$: $\{pref_p = nil \wedge \forall r \in Neigh_p - \{q\} : \ pref_r \neq p \wedge pref_q = nil\}$
$pref_p := q$

$\mathbf{U_p}$: $\{pref_p = q \wedge pref_q \neq p \wedge pref_q \neq nil\}$
$pref_p := nil$

výstupná podmienka — $\psi_{\mathbf{p}} : (\mathbf{match(p)} \vee \mathbf{dead(p)})$
označme $\mathbf{MM} = \{(\mathbf{p}, \mathbf{pref_p}) : \mathbf{pref_p} \neq \mathbf{nil}\}$

Fakt 1 *Ak ψ platí, MM je maximálne párovanie.*

Fakt 2 *γ je terminálna práve vtedy, keď ψ platí.*

Fakt 3 *Algoritmus dosiahne terminálnu konfiguráciu v $O(N^2)$ krokoch.*

Uvažujme normu F - $\mathbf{F}(\gamma) = (\mathbf{c} + \mathbf{f} + \mathbf{w}, \mathbf{2c} + \mathbf{f})$, kde
c je počet chaining procesov v konfigurácii γ (chain(p)=T)
f je počet free procesov v konfigurácii γ (free(p)=T)
w je počet waiting procesov v konfigurácii γ (wait(p)=T)

$$c + f + w \leq N$$
$$2c + f \leq 2N$$

F môže nadobúdať nanajvýš $(N+1)(2N+1)$ hodnôt $\Longrightarrow \mathbf{O(N^2)}$

Voľba a konštrukcia kostry

Predpoklady:

- koreňom bude proces s maximálnym číslom

- **read-all state model** (v jednom kroku vie proces čítať obsah stavu svojich susedov)

- súvislý graf

Premenné v procese p — $Root_p, Par_p, Dis_p$

Predikáty

$$
\begin{aligned}
root(p) &\equiv Root_p = p \wedge Dis_p = 0 \\
child(p,q) &\equiv Root_p = Root_q > p \wedge Par_p = q \in Neigh_p \wedge Dis_p = Dis_q + 1 \\
tree(p) &\equiv root(p) \vee \exists q : child(p,q) \\
lmax(p) &\equiv \forall q \in Neigh_p : Root_p \geq Root_q \\
sat(p) &\equiv tree(p) \wedge lmax(p)
\end{aligned}
$$

výstupná podmienka $\psi : \forall \mathbf{p}\ \mathbf{sat(p)}$

Fakt 4 *Ak platí ψ, tak hrany $\{(p,q) : child(p,q)\}$ tvoria kostru, ktorej koreňom je proces s maximálnym číslom.*

var $Root_p$, Par_p , Dis_p ; (* Describe tree structure *)
Req_p, $From_p$, To_p, Dir_p ; (* Request forwarding *)

$\mathbf{B}_p$: (* Become root *)
{ $\neg tree(p)$ }
$Root_p := p$; $Dis_p := 0$;
$Req_p := p$; $To_p := q$; $Dir_p := Ask$

$\mathbf{A}_p$: (* Ask permission to join *)
{ $tree(p) \wedge \neg lmax(p)$ }
Select $q \in Neigh_p$ with maximal value of $Root_q$;
$Req_p := p$; $From_p := p$; $To_p := q$; $Dir_p := Ask$

$\mathbf{J}_p$: (* Join tree *)
{ $tree(p) \wedge \neg lmax(p) \wedge grant(To_p, p)$ }
$Par_p := q$; $Root_p := Root_q$; $Dis_p := Dis_q + 1$;
$Req_p := From_p := To_p := Dir_p := udef$

$\mathbf{C}_p$: (* Clear request variables *)
{ $sat(p) \wedge \neg\exists q : forw(p, q) \wedge \neg idle(p)$ }
$Req_p := From_p := To_p := Dir_p := udef$

$\mathbf{F}_p$: (* Forward request *)
{ $sat(p) \wedge idle(p) \wedge asks(q, p)$ }
$Req_p := Req_q$; $From_p := q$; $To_p := Par_p$

$\mathbf{G}_p$: (* Grant join request *)
{ $sat(p) \wedge root(p) \wedge forw(p, q) \wedge Dir_p = Ask$ }
$Dir_p := Grant$

$\mathbf{R}_p$: (* Relay grant *)
{ $sat(p) \wedge grant(Par_p, p) \wedge Dir_p = Ask$ }
$Dir_p := Grant$

Ku korektnosti algoritmu

premenné v procese p

$Root_p, Par_p, Dis_p$	popisujú stromovú štruktúru
Req_p	procesy, ktorých požiadavka na pripojenie sa spracováva
$From_p$	sused, ktorého požiadavka sa číta
To_p	sused, ktorému sa požiadavka forwarduje
Dir_p	indikuje, či sa požiadavka spracovala

predikáty

$$
\begin{aligned}
idle(p) &\equiv Req_p = From_p = To_p = Dir_p = undef \\
asks(p,q) &\equiv ((root(p) \wedge Req_p = p) \vee child(p,q)) \\
&\quad \wedge To_p = q \wedge Dir_p = Ask \\
forw(p,q) &\equiv Req_p = Req_q \wedge From_p = q \wedge To_q = p \wedge To_p = Par_p \\
grant(p,q) &\equiv forw(p,q) \wedge Dir_p = Grant
\end{aligned}
$$

Metódy stabilizácie - skladanie protokolov

prvý výstupná podmienka θ

druhý pri vstupnej podmienke θ dosahuje výstupnú podmienku ψ

Definícia 1 *Nech S_1, S_2 sú dva programy, pričom žiadna premenná, do ktorej píše S_2, sa nevyskytuje v S_1.* ***Paralelným zložením*** *S_1, S_2; $\mathbf{S_1 \diamondsuit S_2}$, je taký program, ktorý obsahuje všetky premenné a akcie oboch programov.*

Realizácia $S_1 \diamondsuit S_2$ je korektná vzhľadom k S_i ak obsahuje nekonečne veľa krokov S_i, alebo obsahuje nekonečne dlhý sufix taký, že sa v ňom nedá aplikovať krok S_i.

Veta 5 *Ak platia podmienky*

- *S_1 stabilizuje k θ*
- *ak platí θ, stabilizuje S_2 k ψ*
- *odkedy platí θ, nemení S_1 premenné, ktoré číta S_2*
- *všetky realizácie sú korektné vzhľadom k S_1, S_2*

tak $S_1 \diamondsuit S_2$ stabilizuje k ψ

Metódy stabilizácie - výpočet minimálnych ciest

- D funkcia z množiny ciest do úplne usporiadanej množiny U
- $D(\pi)$ je cena cesty π
- $D(p_0, \ldots, p_k, p_{k+1}) = f_{p_{k+1}p_k}(D(p_0, \ldots, p_k))$
- cena prázdnej cesty (p) nulovej dĺžky z p do p je c_p, a proces p ju vie
- nech $\pi = p_0, \ldots, p_k$, potom $D(\pi) = f_\pi(c_{p_0})$

Vyžadujeme, aby funkcie spĺňali

monotónnosť– nech pq je hrana, $x \in U$ a π jednoduchá cesta do q taká, že neobsahuje p. Potom $D(\pi) < x \Rightarrow f_{pq}(D(\pi)) \leq f_{pq}(x)$

nárast v cykle– ak π je cyklus, $f_\pi(x) > x \; \forall x \in U$

predlžovanie– $\exists B$ také, že pre každú cestu ρ dĺžky aspoň B, $\forall x \in U$ a $\forall$ jednoduchú cestu π dĺžky nanajvýš $N - 1$, $f_\rho(x) > D(\pi)$

Problém minimálnych ciest - $\forall p$ nájsť cestu minimálnej ceny $k(p)$, ktorá končí v p a predchodcu $\phi(p) = q$ na nej

1. $\forall p$ existuje jednoduchá cesta $\pi(p)$, ktorá končí v p taká, že cena každej cesty, ktorá končí v p, je aspoň $D(\pi(p))$
2. existuje les kostier (spanning forest) taký, že cena (jedinej)cesty z koreňa do p má cenu $D(\pi(p))$

Výstupná podmienka - $\psi \equiv (\forall \mathbf{p} : \mathbf{K_p} = \mathbf{k(p)} \wedge \mathbf{L_p} = \phi(\mathbf{p}))$

Algoritmus Update minimálnych ciest

var K_p : D ;
L_p : $Neigh_p \cup \{nil\}$;

$\mathbf{C}_p$: $K_p := \min(c_p, \min\{f_{pq}(K_q) : q \in Neigh_p\})$;
if $K_p = c_p$ **then** $L_p := nil$
else $L_p := q$ s.t. $K_p = f_{pq}(K_q)$

Veta 6 *Algoritmus Update minimálnej cesty stabilizuje k minimálnym cestám*

K_p je dolný odhad na $k(p)$ - označme $k_i(p)$ minimálnu cenu cesty nanajvýš dĺžky i. Od skončenia kola i platí, že $K_p \leq k_{i-1}(p)$

niekedy sa K_p stane horným odhadom $k(p)$ - ukáže sa, že po i kolách odpovedá K_p alebo existujúcej ceste alebo pôvodnej informácii, ktorá pri putovaní spravila aspoň i krokov/hopov

argumentácia axiomy predlžovania - po kole B je hodnota $f_\rho(K_r^*)$ väčšia ako ľubovoľná existujúca jednoduchá cesta

Aplikácie

smerovanie výpočet tabuliek - pre fixované v je: počítame paralelne pre všetky v

- $D\pi$ váha cesty π, ak začína vo v, inak je to ∞;
- $c_p = 0$ pre $p = v$, inak ∞;
- $f_{pq}(C) = C + \omega_{pq}$;

lexikograficky najkratšie cesty definujú **DFS kostru** s koreňom v minimálnom vrchole;

- cena cesty bude zoznam vrcholov, ktoré obsahuje; pritom jednoduchá cesta má vždy menšiu cenu ako cesta, ktorá obsahuje cyklus