Üzenetküldés szemantikája

Csatornaváltozók

Párhuzamos és elosztott rendszereket gyakran írunk le folyamathálózatok formájában. A folyamatokat dobozokkal jelöljük, közöttük üzenetküldéssel teremtünk kapcsolatot. A csatornaváltozók használatával egyirányú aszinkron kommunikációt valósíthatunk meg. Van egy feladó folyamat és egy vagy több fogadó. Feltételezzük, hogy a csatorna biztonságos, üzenetek nem vésznek el út közben és nem is "keletkeznek", valamint a küldött üzeneteket a küldési sorrendben lehet a másik oldalon leolvasni. Épp ezért a csatornára úgy tekintünk, mint a sor adatszerkezetre.

A csatornán a még fel nem dolgozott üzeneteket tárolni tudjuk (mivel aszinkron). Ez itt egy végtelen kapacitású sor, a valóságban persze mindig van valami korlát (a valóságban ha ezt a korlátot elérjük, a folyamat blokkolódik). Ezt a sort hívjuk mondjuk -nek. Tartozik hozzá még egy sor, a történetváltozó, ennek jelölése: $\overline{x}$. A történetváltozó tartalmaz minden valaha x-re küldött üzenetet (a küldésnek megfelelő sorrendben), innen nem lehet eltávolítani se az elemeket. Ez az implementációban sehol se szerepel(het), de a csatornát használó programok specifikációjában jól jön egy ilyen absztrakt segédváltozó.

Jelölés: az $(x, \overline{x}) \in Ch(Int)$ pár egy egész típusú adatokat szállító csatorna.

Műveletek:

- inicializálás: $x :=<>$
- üzenet küldése: $x := hiext(x, e)$ vagy $x := x; e$
- üzenet olvasása: $x.lov$ feltéve $x \neq <>$
- üzenet eltávolítása: $x := lorem(x)$ feltéve $x \neq <>$
- üres-e: $x =<>$
- sor mérete: $length(x)$ vagy $|x|$

Itt csupán x-ként hivatkoztunk magára a csatornára, a háttérben persze $\overline{x}$ is szerepet kap. A fenti műveletek jelentését megadhatjuk az lf-jükkel, és akkor látjuk azt is, milyen hatással vannak a műveletek $\overline{x}$-re.

- $lf(x :=<>, R) = R^{x \leftarrow <>, \overline{x} \leftarrow <>}$, tehát inicializáljuk az x-hez tartozó történetváltozót is.
- $lf(x := x; e, R) = R^{x \leftarrow x;e, \overline{x} \leftarrow \overline{x};e}$, tehát a történetváltozóhoz is hozzáírjuk e-t
- $lf(x := lorem(x), ha\ x \neq <> , R) = x \neq <> \rightarrow R^{x \leftarrow lorem(x)} \wedge x =<> \rightarrow R$, a történetváltozó változatlan.

Szinkron kommunikáció

Egy se küldő folyamat több re_i fogadónak is küld. $i \in [1..n] : x \to y(i)$

Modellje ($se \to re_i$) eset, sok fogadó:

$$\| \; y(i) := x, \; ha \underset{i \in [1..n]}{\wedge} (y(i) = 0 \wedge x \neq 0) \; \| \; x := 0$$

Tehát x küld minden y-nak, és ezzel egyidejűleg nullázza magát. De feltétel, hogy minden y üres kell legyen, ezért sokáig fog blokkolni, gyakorlatilag szekvencializálódik a párhuzamos program.

($se \to re$ eset, egy fogadó):

$$x, y := 0, x \; ha \; x = 0 \wedge x \neq 0$$

Elvárásaink: küldőnél $x \neq 0$ stabil, fogadóknál $y(i) = 0$ stabil.
Kérdés: csatornaváltozók vagy közös memória? Ez nem kérdés, mert a gyakorlatban ez a kettő ugyanarra vezet.
Pufferváltozó (u) használatával aszinkronitás keletkezik, a küldő továbbhaladhat, azaz x-re tölthet anélkül, hogy olvasva lenne, addig u-ban pihen az adat. Annyi szinkronitás van (ugye ebből minél kevesebb kell), hogy amíg az egyik írja az u-t, addig a másik nem tudja.

$$se : x, u := 0, x \; ha \; u = 0 \wedge x \neq 0 \; \square \; re : u, y := 0, u \; ha \; y = 0 \wedge u \neq 0$$

Ez a transzformáció megtartja a program lényegi tulajdonságait (pl. minek felel meg), de mégis megszüntette a szinkronitást.

Amúgy, ha nincsenek osztott változóink, csak csatik, akkor az egyes folyamatok közötti kapcsolat jól ellenőrizhetővé válik. Ezt fogalmazza meg a lokalitás tétel:

- Ha egy P állítás változói között csak P_1 folyamat lokális változói, illetve P_1 bemenő csatornaváltozói vannak, akkor a P állítás stabil a többi folyamatban.
- Ha $P \Rightarrow P^{\overline{x} \leftarrow \overline{x}; e}$ és $V(P) = \{\overline{x}\}$, akkor P stabil a teljes folyamathálózatban.