1. fejezet

Programozási módszertan elmélete

1.1. Az absztrakt adattípus egy matematikai modellje. (Univerzális algebra és absztrakt adattípus). Az adattípus egy komplex leírása, típusosztály morfizmusdiagramja. Interfész leképezések, típusöröklődések.

1.1.1. Az absztrakt adattípus definíciója

1. Definíció. $\Sigma = (S, OP)$ *szignatúra, ha:*

- S*: szortok (halmaznevek) véges halmaza*
- OP*: függvényszimbólumok (műveleti jelek) véges halmaza*
- $\forall w \in OP : w : s_1 \dots s_k \to s \quad (s_1 \dots s_k, s \in S, k \geq 0)$

Ha $|S| = 1$, akkor *homogén szignatúra*, ha $|S| > 1$, *akkor heterogén szignatúra.*

2. Definíció. $Alg = (S_A, OP_A)$ *a* $\Sigma = (S, OP)$ *szignatúra feletti* Σ-algebra*, ha*

- $\forall s \in S \exists A_S \in S_A$ *halmaz*
- $S_A = \{A_s | s \in S\}$ *véges*
- $\forall w : s_i s_j \dots s_k \to s_l \in OP : \exists w_A : A_{s_i} A_{s_j} \dots A_{s_k} \to A_{s_l} \in OP_A$
- $OP_A = \{w_A | w \in OP\}$ *véges*

Ha $|S_A| = 1$, akkor *homogén univerzális algebra*, ha $|S_A| > 1$, akkor *heterogén univerzális algebra*. Ha a leképezés totális függvény, akkor *totális algebra*, egyébként *parciális algebra* ???. Adott Σ szinatúrához tartozó összes Σ-algebrák osztálya: $Alg(\Sigma)$.

3. Definíció. *Specifikáció: ha $\Sigma = (S, OP)$ egy szignatúra, E a szemantikát meghatározó specifikáció (axiómák, műveletek kiszámítási szabályainak vagy elő- és utófeltételeinek halmaza), akkor $SPEC = (\Sigma, E)$ egy specifikáció.*

A specifikáció szokásos jelölése:

```
specnév =
  sorts: S
  oprs: OP
  eqns: E
```

4. Definíció. *Az (S_A, OP_A) Σ-algebra egy* absztrakt adattípus (ADT), *ha:*

- $S_A = \{A_0, \ldots, A_n\}$, *ahol* A_0 kitüntetett alaphalmaz *(TOI), a típusobjektumok halmaza, $A_1, \ldots A_n$ pedig a kisegítő alaphalmazok, a típusobjektumok feléptítésében, jellemzésében játszanak szerepet.*
- $OP_A = \{f_0, \ldots, f_m\}$ *az $A_0, \ldots A_n$-en értelmezett parciális függvények halmaza, $f_0 :\to A_0$ konstans operáció*
- A_0 *megkonstruálható, azaz minden eleme előáll $f_0, \ldots, f_m$ véges kompozíciójaként*

Ha $|S_A| = 1$, akkor egyszerű ADT (homogén univerzális algebra, csak TOI van), egyébként összetett ADT (heterogén univerzális algebra).

Operációk típusai:

Konstruktor: OP_A-ból az a minimális függvényhalmaz, amik véges kompozíciójával az egész A_0 előáll. Nem biztos, hogy egyértelmű. Jele: F_C.

Szelektor: $F_S = OP_A \backslash F_C$

1.1.2. A típusosztály

- Az adattípus leírása lehet absztrakt vagy konkrét.
- Absztrakt szinten csak a műveletek jelentését adjuk meg, a megvalósítással nem törődünk.
- Konkrét szinten megadjuk az implementációt is.
- A *típusosztály* ennek a két megjelenési formának az egyesítése.

- A specifikációmorfizmus segítségével a típusosztály leírása rövidebb lehet.

5. Definíció. $\Sigma = (S, OP)$ *és* $\Sigma' = (S', OP')$ *két szignatúra, ekkor* $\mu : \Sigma \to \Sigma'$ szignatúramorfizmus Σ *és* Σ' *között, ha:*

- $\mu = (\mu_S, \mu_{OP})$
- $\mu_S : S \to S'$
- $\mu_{OP} : OP \to OP'$
- $\forall n : s_1 \dots s_k \to s \in OP : \mu_{OP}(n) = m : \mu_S(s_1) \dots \mu_S(s_k) \to \mu_S(s) \in OP'$

6. Definíció. $SPEC = (\Sigma, E)$ *és* $SPEC' = (\Sigma', E')$ *két specifikáció. Ekkor* $\mu : SPEC \to SPEC'$ *egy* specifikációmorfizmus*, ha:*

- $\mu = (\mu_S, \mu_{OP}, \mu_E)$
- μ *egy* $(S, OP) \to (S', OP')$ *szignatúramorfizmus kiterjesztése termekre*
- $\mu_E : E \to E'$
- $\mu(E) \subseteq E'$

A specifikációmorizmus segítségével a típusosztály leírása tömörebb, illet az öröklődés kifejezésére is használhatjuk.

7. Definíció. *A* Típusosztály *egy* $(PAR, EXP, IMP, BOD, e, i, eb, ib)$ *nyolcas, ahol*

- *PAR: specifikáció. A típusosztály paraméterei, és azokkal szemben támasztott elvárások. Pl elsőbbségi sorban az egyes elemek között legyen rendezés.* `C++`*-ban template-paraméterek.*
- *EXP: export felület, kitüntetett szortú specifikáció. A típusobjektumok halmazának (TOI) és a rajtuk értelmezett műveletek definíciója, pl verem esetén a veremműveletek szintaxisa, szemantikája.* `C++`*-ban* `public` *interfész.*
- *IMP: import felület, specifikáció. Azon adattípusok specifikációja, amikkel ábrázolni szeretnénk a típusosztályt.* `C++`*-ban* `#include`
- *BOD: body, kitüntetett szortú specifikáció. Az EXP-ben secifikált műveletek megvalósítása az IMP-beli típusokkal. Pl push műveletre új elem felvétele a vektorba, és a veremmutató mozgatása.* `C++`*-ban memberfüggvények megvalósítása.*

- $e : PAR \to EXP$
- $i : PAR \to IMP$
- $eb : EXP \to BOD$ *kintüntetett szortú specifikációmorfizmus*
- $ib : IMP \to BOD$
- *utóbbi négy függvény specifikációmorfizmusok (interfészleképezések), amelyekre igaz, hogy* $eb \circ e = ib \circ i$
- *mind a négy tartalmazás, de eb ábrázolás is*

???kitüntetett szortú specifikáció mit jelent?

1.1.3. Interfészleképezések fajtái

Az interfészleképezések specifikációmorfizmusok, amelyek két specifikáció között hatnak. Fajtái:

- Üres
- Identitás
- Újraelnevezés: szort vagy operáció új nevet kap. Jelölése: `sorts [oprs]: újnév = réginév`
- Tartalmazás: bővül a szortok, operációk vagy egyenletek halmaza
- Ábrázolás: a class szort helyére más szortok Descartes-szorzata kerül, pl $eb($`verem`$) =$ `vector nat`

A tartalmazás segít a tömörebb leírásban, amit az egyik részben specifikáltunk, azt nem ismételjük meg a másikban, mert a tartalmazásmorfizmus áthozza.

1.1.4. Típusöröklődés

- Típusosztályok definíciói gyakran nagyon hasonlítanak egymásra. Egyes esetekben arra is van lehetőség, hogy már meglévő, hibátlanul működő szolgáltatásokat egy osztálytól örökléssel átvegyünk.
- Az öröklődést specifikációmorfizmusokkal definiáljuk.
- Két fajtája van: specializációval és újrafelhasználással történő öröklés

8. Definíció.
Típusöröklődés*: ha* $C_1 = (PAR_1, EXP_1, IMP_1, BOD_1, e_1, i_1, eb_1, ib_1)$ *és* $C_2 = (PAR_2, EXP_2, IMP_2, BOD_2, e_2, i_2, eb_2, ib_2)$ *két, azonos szignatúrájú típusosztály, ekkor:*

- C_2 *a* C_1*-ből* specializációval *származik, ha* $\exists f : EXP_1 \rightarrow EXP_2$ *kitüntetett szortú specifikációmorfizmus*
- C_2 *a* C_1*-ből* újrafelhasználással *származik, ha* $\exists f : EXP_1 \rightarrow BOD_2$ *kitüntetett szortú specifikációmorfizmus*
- *f mindkét esetben lehet identitás, újraelnevezés és tartalmazás*

1.2. A típusosztály morfizmusdiagramjának elemzése. Az absztrakt típusspecifikáció elemzése. A korrekt realizáció. A kettős specifikációval kapcsolatos alapfogalmak. Reprezentáció, a reprezentáció elemzése. Implementáció, az implementáció elemzésére vonatkozó tételek

1.2.1. A morfizmusdiagram elemzése

1. *eb*: identitás
2. *eb*: ábrázolás és tartalmazás
3. *e*: tartalmazás, *eb*: ábrázolás és tartalmazás

A többi specifikáció és specifikációmorfizmus üres. A megvalósítás kívülről látható, ha *eb* identitás, és kívülről nem látható (beburkolás), ha *eb* ábrázolás és tartalmazás.

1.2.2. A specifikáció elemzése

Egy adott típusosztály elemzése három lépésből áll:

1. absztrakt specifikáció elemzése (koncepciónak megfelel-e a specifikáció)
2. reprezentáció elemzése (absztrakt típusobjektumok jól vannak-e reprezentálva a konkrét típusobjektumokkal)
3. implementáció (az EXP műveleteit jól implementálja-e BOD konkrét szinten)

1.2.3. Az absztrakt specifikáció elemzése

- Az informális elvárásoknak megfelel a formális leírás?
- Adott: $SPEC = (\Sigma, E)$, ahol $\Sigma = (S, OP)$, és S, OP és E a szokásosak.
- Az osztály EXP része algebrai axiómákkal adott. (Algebrai, előfeltételek-utófeltételes és procedurális specifikáció egyre konkrétabbak, EXP-et szokás absztrakt, BOD-ot pedig konkrét specifikációnak nevezni.)
- Adott az exportfelület absztrakt specifikációja: $\Sigma = (S, OP)$, $Alg = (S_A, OP_A)$ és $SPEC_A = (S_A, OP_A, E_A)$, ahol E_A axiómákkal van megadva.
- Másik jelölés: $SPEC_a = d_a = (A, F, E_a) = (\{A_0, \ldots, A_n\}, \{f_0 :\rightarrow A_0, \ldots\}, \{\ldots \alpha(a) \Rightarrow f_s(f_c(s)) = h(a) \ldots\})$, ahol $a \in A_i \times \ldots \times A_k$

Kérdések az absztrakt specifikációval kapcsolatban.

- A specifikáció konzisztens? (axiómák között nincs-e ellentmondás, és nem kell-e újakat hozzávenni)
- Errormentes? A műveletek parciálisak, néha „`error`"-t adnak. Ha a művelet argumentuma `error`, akkor az eredménye is `error`. Hogyan terjednek ezek a hibák?
- Minden típusobjektumot előállít F_C, tehát a megválasztása helyes?
- Az absztrakt specifikáció megfelel a koncepciónak?
  - Bizonyítani nem lehet a koncepció informális volta miatt.
  - Az axiómák nem rögzítenek minden elvárt tulajdonságot.
  - Tételeket kell bizonyítani.
  - Ezeknek két fajtája: típusobjektumok egyenlősége és paraméterek egyenlősége

9. Definíció. Egyenlőségaxióma*: legyen $d_a = (A, F, E_a)$ egy specifikáció és $a_1, a_2 \in A_0$. Ekkor*

$$a_1 = a_2 \Leftrightarrow (a_1 = f_0 \wedge a_2 = f_0) \vee \forall f_S \in F_S : f_S(a_1) = f_S(a_2).$$

10. Definíció. Kibővített egyenlőségaxióma*: nem követeli meg az összes szelektor egyenlőségét, például vektornál csak a* `read`*-et kell. Ez tételként bizonyítható.*

11. Definíció. Strukturális indukció*: legyen* $d_A = (A, F, E_A)$ *egy specifikáció és* P *egy* A_0*-n értelmezett predikátum. Ha* $P(f_0) = igaz$ *és* $\forall a \in A_0 :$

$$P(a) = igaz \Rightarrow \forall f_C \in F_C : P(f_C(a)) = igaz,$$

akkor $\forall a \in A_0 : P(a) = igaz.$

1.2.4. Kettős specifikáció

- Adott a koncepció és absztrakt specifikáció, és feltételezzük, hogy utóbbi helyes a koncepció szerint.
- Megírjuk a konkrét specifikációt így, kérdés, hogy megfelel-e az absztraktnak.
- A típusosztály specifikációja az absztraktot (EXP) és a konkrétat (BOD) is tartalmazza, ezért *kettős specifikáció* a neve. Itt dől el, hogy helyes-e a konkrét.
- Ehhez kell a reprezentáció és a helyesség fogalma.

1.2.5. Reprezentáció

12. Definíció. Reprezentáció*: legyen* $d_a = (A, F, E_a)$ *és* $d_c = (C, G, E_c)$ *két azonos szignatúrájú specifikáció, és* $\varphi : C \to A$ *morfizmus. Ekkor a* C *objektumhalmaz az* A *egy* φ *melletti reprezentációja, ha* φ *szürjektív.*

???a specifikáció nem szignatúra+műveletek?

Megjegyzés: $A = \{A_0, \ldots, A_n\}$, $C = \{C_0, \ldots C_n\}$ és $\varphi = (\varphi_0, \ldots, \varphi_n)$ egy függvénycsalád, $\varphi_i : C_i \to A_i$, de ezek közül általában csak φ_0 nem identitás, ezért φ_0 helyett φ-t írunk.

1. Tétel. Elégséges feltétel a reprezentációra*: legyen* $d_a = (A, F, E_a)$ *és* $d_c = (C, G, E_c)$ *két azonos szignatúrájú specifikáció, és* $\varphi : C \to A$ *morfizmus. Amennyiben* $\forall f_C \in F_C : \forall c \in C$ *esetén*

$$(g_C(c) \in C \wedge f_C(\varphi(c)) \in A) \Rightarrow f_C(\varphi(c)) = \varphi(g_C(c))$$

teljesül, úgy a C *objektumhalmaz az* A *egy reprezentációja.*

13. Definíció. *A reprezentációs függvény implicit definíciója:*

$$f_0 = \varphi(g_0)$$

$$\forall f_C \in F_C : f_C(\varphi(c)) = \varphi(g_C(c))$$

14. Definíció. *A reprezentációs függvény explicit (rekurzív) alakja: tegyük fel, hogy* $\exists g_c, g_s : c = g_c(g_s(c))$*, ekkor:*

$$\varphi(c) = \mathbf{if} c = g_0 \mathbf{then} f_0 \mathbf{else} f_C(\varphi(g_s(c))) \mathbf{fi}$$

Explicit módon több alakban is meg lehet adni a reprezentáló függvényt, de azok ekvivalensek lesznek.

1.2.6. A reprezentációelemzés feladata

- φ szürjektív-e
- bizonyítani hogy a φ-k különböző definíciói ekvivalensek
- ha $c_1 = c_2$, akkor igaz-e, hogy $\varphi(c_1) = \varphi(c_2)$ (egyenlőségaxióma)
- a műveletek konkrét és absztrakt változata ugyanazt adja-e

1.2.7. Helyesség

15. Definíció. Helyesség*: legyen $d_a = (A, F, E_a)$ és $d_c = (C, G, E_c)$ két azonos szignatúrájú specifikáció, és $\varphi : C \to A$ morfizmus. Ekkor ha*

1. *C az A reprezentációja φ mellett,*
2. *$\forall f_i \in F, c \in C \wedge \varphi(c) \in A \wedge f_i(\varphi(c))$ értelmezve van, akkor $g_i(c)$ is,*
3. *$\forall f_i \in F, c \in C \wedge c' = g_i(c) \wedge c' \in C \wedge \varphi(c) \in A \wedge \varphi(c') \in A$, akkor $f_i(\varphi(c)) = \varphi(g_i(c))$*

akkor d_c d_a szerint helyes.

Ez a definíció azt mondja, hogy g az f hatását szimulálja a konkrét térben, de definiálható azonos szemantikával is a helyesség.

A specifikációk egyre közelebb kerülnek a megvalósításhoz, de minden lépésben helyességet kell bizonyítani.

2. Tétel. *Az algebrai axiómák átírhatók elő-utófeltételes formára. Módszer:*

1. *legyen $I(a)$ invariáns*
2. *$E_a = \{\ldots, \alpha(a) \Rightarrow f_S(f_C(a)) = h(a), \ldots, \sim I_a(f_C(a)) \Rightarrow f_C(a) =$ "$undefined$"$\}$*
3. *$\alpha(a) \Rightarrow f_S(f_C(a)) = h(a)$ átírása: $\{\alpha(a) \wedge b = f_C(a)\}b' = f_S(b)\{b' = h(a)\}$*
4. *f_0 kezdeti objektum szemantikája: $\{true\}a = f_0\{I(a)\}$*
5. *$f_C \in F_C$ konstruktor szemantikája: $\{I(f_C(a))\}b = f_C(a)\{I(b) \wedge b = f_C(a)\}$*

3. Tétel. *Elő-utófeltételes konkrét specifikáció (d_c) elő-utófeltételes absztrakt specifikáció (d_a) szerinti helyessége (elégséges feltétel):*

- $I_c(c) \Rightarrow I_a(\varphi(c))$
- $post_{g_0}(c) \Rightarrow I_c(c)$

- $post_{g_0}(c) \Rightarrow post_{f_0}(\varphi(c))$
- $I_c(c) \wedge pre_{f_i}(\varphi(c)) \Rightarrow pre_{g_i}(c)$
- $I_c(c) \wedge pre_{f_i}(\varphi(c)) \wedge post_{g_i}(c, c') \wedge I_c(c') \Rightarrow post_{f_i}(\varphi(c), \varphi(c'))$
- $I_c(c) \wedge pre_{g_i}(c) \wedge post_{g_i}(c, c') \Rightarrow I_c(c')$

fennállása esetén d_c helyes d_a szerint.

4. Tétel. *Elségséges feltétel elő-utófeltételes absztrakt és procedurális konkrét specifikáció helyességére: adott $d_a = (A, F, E_a)$ és $d_c = (C, G, E_c)$ azonos szignatúrával. $E_a = \{\{true\}a = f_0\{post_{f_0}(a)\}, \ldots, \{pre_{f_i}(a)\}a' = f_i(a)\{post_{f_i}(a, a')\}, \ldots\}$, $E_c = \{\ldots, Q_{g_i}, \ldots\}$. Az invariánsok I_a és, I_c, $\varphi : C \to A$ morfizmus. Ekkor ha*

- $\forall c \in C : I_c(c) \Rightarrow I_a(\varphi(c))$
- $\{"true"\}Q_0\{post_{f_0}(\varphi(c)) \wedge I_c(c)\}$
- $\forall f \in F : \{pre_{f_i}(\varphi(c)) \wedge I_c(c)\}Q_i\{post_{f_i}(\varphi(c), \varphi(c')) \wedge I_c(c')\}$

akkor d_c helyes d_a szerint.

1.2.8. Az implementációelemzés feladata

A kettős specifikáció helyességének bizonyítása. ???

1.3. Determinisztikus és nemdeterminisztikus programok. Szintaxis, szemantika, helyességbizonyítás, Hoare-módszer.

1.3.1. Determinisztikus és nem determinisztikus programok

Szekvenciális program:

- determinisztikus program → valódi program → strukturált program
- nemdeterminisztikus program

16. Definíció. *A* valódi programot *a program gráfjával definiáljuk, amely:*

- *összefüggő*
- *van egy* `start` *és egy* `end` *csúcsa*
- *minden ponton keresztül vezet út a* `start`*-ból az* `end`*-be*

Az előbbi két csúcson kívül a következők szerepelhetnek a gráfban:

- *függvény csomópont*
- *elágazás*
- *gyűjtő csomópont*

???rajzolni

1.3.2. Szintaxis

17. Definíció. *Strukturált program szintaxisa rekurzív BNF-formában:*

$S ::= skip | y \leftarrow f(x,y) | S_1; S_2 |$ `if` $\alpha(x,y)$ `then` S_1 `else` S_2 `fi` | `while` $\alpha(x,y)$ `do` S_1 `od`

Ahol

- S, S_1, S_2 *determinisztikus programok,*
- X *az* S *bemenő adatainak halmaza és* $x \in X$*,*
- Y *az* S *változóinak halmaza és* $y \in Y$*,*
- $f(x,y)$*: kifejezés,*
- $\alpha(x,y)$*: x-től és y-tól függő, kvantorfüggetlen logikai kifejezés.*

Itt x bemenő, y változó.

5. Tétel. *Strukturált programozás alaptétele (Böhm, Jacopini): Minden valódi programhoz vele ekvivalens strukturált program készíthető.*

18. Definíció. *Nemdeterminisztikus program szintaxisa:*

$$S ::= skip | b \rightarrow y \leftarrow f(x,y) | S_1; S_2 |$$ `if` $\alpha_1 \rightarrow S_1 \ldots \alpha_1 \rightarrow S_n$ `fi` $|$

`do` $\alpha_1 \rightarrow S_1 \ldots \alpha_1 \rightarrow S_n$ `od` | `do` $S_B \square S_E;$ `exit od`

Ahol

- $\alpha_i(x,y)$*: filter vagy őrfeltétel, x-től és y-tól függő, kvantorfüggetlen logikai kifejezés (a fenti BNF leírásban csak* α_i*-nek rövidítve),*
- `if` $\alpha_1 \rightarrow S_1 \ldots \alpha_1 \rightarrow S_n$ `fi` *nemdeterminisztikus kiválasztás,*
- `do` $\alpha_1 \rightarrow S_1 \ldots \alpha_1 \rightarrow S_n$ `od` *nemdeterminisztikus iteráció,*
- `do` $S_B \square S_E;$ `exit od` *kilépéses nemdeterminisztikus iteráció az* S_B *az* $\alpha_1 \rightarrow S_1 \ldots \alpha_1 \rightarrow S_n$ *magjának a rövidítése.*

???kilépéses iteráció szemantikája?

1.3.3. Szemantika

Jelölések:

- S program állapotainak halmaza: Σ,
- $\sigma \in \Sigma$ állapot,
- átmenet: $\langle S, \sigma \rangle \to \langle S', \sigma' \rangle$ (S' maradékprogram),
- ha σ' végállapot, akkor a maradékprogramot T jelöli
- üres program: E,
- hibás állapot: $\sigma \equiv \texttt{fail}$,
- divergencia: $\sigma \equiv \perp$,
- $\sigma(\alpha)$: α predikátum értéke σ állapotban
- $\sigma^{y \leftarrow f(x,y)}$: σ állapotban y helyére $f(x,y)$ helyettesítése.

A programok szemantikáját ezekkel a jelölésekkel kifejezett axiómákkal adjuk meg.

19. Definíció. *A determinisztikus, strukturált programok szemantikai axiómái:*

$$\langle skip, \sigma \rangle \to \langle E, \sigma \rangle$$

$$\langle y \leftarrow f(x,y), \sigma \rangle \to \left\langle E, \sigma^{y \leftarrow f(x,y)} \right\rangle$$

$$\frac{\langle S_1, \sigma_1 \rangle \to \langle S_2, \sigma_2 \rangle}{\langle S_1; S, \sigma_1 \rangle \to \langle S_2; S, \sigma_2 \rangle}$$

$$\sigma(\alpha) = \texttt{"true"} \Rightarrow \langle \texttt{if } \alpha \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}, \sigma \rangle \to \langle S_1, \sigma \rangle$$

$$\sim \sigma(\alpha) = \texttt{"true"} \Rightarrow \langle \texttt{if } \alpha \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}, \sigma \rangle \to \langle S_2, \sigma \rangle$$

$$\sigma(\alpha) = \texttt{"true"} \Rightarrow \langle \texttt{while } \alpha \texttt{ do } S \texttt{ od}, \sigma \rangle \to \langle S; \texttt{while } \alpha \texttt{ do } S \texttt{ od}, \sigma \rangle$$

$$\sim \sigma(\alpha) = \texttt{"true"} \Rightarrow \langle \texttt{while } \alpha \texttt{ do } S \texttt{ od}, \sigma \rangle \to \langle E, \sigma \rangle$$

20. Definíció. *A nemdeterminisztikus programok szemantikai axiómái:*

$$\sigma(b) = \texttt{"true"} \Rightarrow \langle b \to y \leftarrow f(x,y), \sigma \rangle \to \left\langle E, \sigma^{y \leftarrow f(x,y)} \right\rangle$$

$$\sim \sigma(b) = \texttt{"true"} \Rightarrow \langle b \to y \leftarrow f(x,y), \sigma \rangle \to \langle E, \texttt{fail} \rangle$$

$$\exists i \in [1 \dots n] : \sigma(\alpha_i) = \texttt{"true"} \Rightarrow$$
$$\langle \texttt{if}\ \ \alpha_1 \to S_1 \dots \alpha_n \to S_n\ \ \texttt{fi}, \sigma \rangle \to \langle S_i, \sigma \rangle$$

$$\sim \exists i \in [1 \dots n] : \sigma(\alpha_i) = \texttt{"true"} \Rightarrow$$
$$\langle \texttt{if}\ \ \alpha_1 \to S_1 \dots \alpha_n \to S_n\ \ \texttt{fi}, \sigma \rangle \to \langle E, \texttt{fail} \rangle$$

$$\exists i \in [1 \dots n] : \sigma(\alpha_i) = \texttt{"true"} \Rightarrow$$
$$\langle \texttt{do}\ \ \alpha_1 \to S_1 \dots \alpha_n \to S_n\ \ \texttt{od}, \sigma \rangle \to \langle S_i; \texttt{do}\ \ \alpha_1 \to S_1 \dots \alpha_n \to S_n\ \ \texttt{od}, \sigma \rangle$$

$$\sim \exists i \in [1 \dots n] : \sigma(\alpha_i) = \texttt{"true"} \Rightarrow$$
$$\langle \texttt{do}\ \ \alpha_1 \to S_1 \dots \alpha_n \to S_n\ \ \texttt{od}, \sigma \rangle \to \langle E, \sigma \rangle$$

1.3.4. Helyességbizonyítás

21. Definíció. S_0 *program* végrehajtási sorozata σ_0 *kiindulási állapottal:*

$$\tau : \langle S_0, \sigma_0 \rangle \to \langle S_1, \sigma_1 \rangle \dots,$$

ahol minden átmenethez létezik egy $(S_i, \alpha_i \to r_i, S_{i+1})$ tranzakció *úgy, hogy* $\sigma(\alpha_i) = \texttt{"true"}$ *és* $\sigma_{i+1} = r_i(\sigma_i)$

22. Definíció.

$$Val(\tau) := \begin{cases} \sigma_n & \text{ha } \langle S_n, \sigma_n \rangle \text{ az utolsó elem}, S_n = T, \sigma_n \in \Sigma \\ \texttt{fail} & \text{ha } \langle S_n, \sigma_n \rangle \text{ az utolsó elem}, S_n \neq T, (\texttt{fail} \notin \Sigma) \\ \bot & \text{ha a végrehajtási sorozat végtelen} \end{cases}$$

23. Definíció. *comp*$(S)(\sigma)$*: az* S *összes,* σ *kezdőállapothoz tartozó végrehajtási sorozatának halmaza.*

24. Definíció. Szemantikai függvény *(program jelentése):*

$$M[S](\sigma) = \{Val(\tau) | \tau \in comp(S)(\sigma)\}$$

Ha S *végrehajtása* sikertelen*:* $\texttt{fail} \in M[S](\sigma)$*,*
ha S *végrehajtása* divergens*:* $\bot \in M[S](\sigma)$*.*
$P(\Sigma)$*: parciálisan helyes eredményállapotok halmaza, ekkor*

- parciális helyességi szemantika*:* $M[S] : \Sigma \to P(\Sigma)$,
- teljes helyességi szemantika*:* $M[S] : \Sigma \to P(\Sigma) \cup \{\bot\}$

25. Definíció. S program specifikációja*:* (φ, ψ) *elő- és utófeltételek, azaz*

$$\varphi(\sigma_0) = \texttt{"true"} \wedge \forall \sigma \in M[S](\sigma_0) : \psi(\sigma) = \texttt{"true"}$$

26. Definíció. S *program* eredményes, *ha* $\forall \sigma \in \Sigma : \varphi(\sigma) \Rightarrow \texttt{fail} \notin M[S](\sigma)$

27. Definíció. S *program* befejeződik, *ha* $\forall \sigma \in \Sigma : \varphi(\sigma) \Rightarrow \perp \notin M[S](\sigma)$

28. Definíció. S *program* (φ, ψ) *specifikáció szerint parciálisan helyes* $(\{\varphi\}S\{\psi\})$*, ha minden, az előfeltételnek megfelelő helyről indítva helyes eredményt ad, ha befejeződik.*
Formálisan:

$$\forall \sigma \in \Sigma : (\varphi(\sigma) = \texttt{"true"} \wedge \sigma' \in M[S](\sigma)) \Rightarrow \psi(\sigma') = \texttt{"true"}$$

29. Definíció. S *program* (φ, ψ) *specifikáció szerint teljesen helyes* $(\{\{\varphi\}\}S\{\{\psi\}\})$*, ha*

- *minden megfelelő helyről indítva befejeződik,*
- *minden befejeződés megfelel az utófeltételnek.*

Formálisan:

$$\forall \sigma \in \Sigma : \varphi(\sigma) \Rightarrow \{\texttt{fail}, \perp\} \cap M[S](\sigma) = \emptyset$$

$$\forall \sigma \in \Sigma : (\varphi(\sigma) = \texttt{"true"} \wedge \sigma' \in M[S](\sigma)) \Rightarrow \psi(\sigma') = \texttt{"true"}$$

1.3.5. A Hoare-módszer

Dedukciós módszer („axiómák+következtetési szabályok ⇒ tétel”) a strukturált programok helyességének bizonyítására.

30. Definíció. *A Hoare-módszer axiómái:*

$$\{P(x,y)\}\texttt{skip}\{P(x,y)\}$$

$$\{P(x, g(x,y))\}y \leftarrow g(x,y)\{P(x,y)\}$$

31. Definíció. *A Hoare-módszer következtetési szabályai:*

$$\frac{\{P\}S_1\{Q_1\} \wedge \{Q_1\}S_2\{Q_2\}}{\{P\}S_1; S_2\{Q_2\}}$$

$$\frac{\{P \wedge \alpha\}S_1\{Q\} \wedge \{P \wedge \sim \alpha\}S_2\{Q\}}{\{P\}\texttt{if } \alpha \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}\{Q\}}$$

$$\frac{\{P \wedge \alpha\}S\{P\} \wedge (P \wedge \sim \alpha) \Rightarrow Q}{\{P\}\texttt{while}\ \alpha\ \texttt{do}\ S\ \texttt{od}\ \{Q\}}$$

$$\frac{P \Rightarrow P_1 \wedge \{P_1\}S\{Q_1\} \wedge Q_1 \Rightarrow Q}{\{P\}S\{Q\}}$$

Ezek alapján csak parciális helyességet lehet bizonyítani, a teljes helyességhez be kell vezetni minden ciklushoz invariánst ($I(x,y)$), terminálófüggvényt ($E(x,y)$), és a következő szabályt:

$$(P(x,y) \Rightarrow I(x,y)) \wedge (I(x,y) \Rightarrow E(x,y) \in W) \wedge$$

$$\{\{I(x,y) \wedge \alpha(x,y) \wedge E = E(x,y)\}\}S\{\{I(x,y) \wedge E(x,y) < E\}\} \wedge$$

$$\frac{I(x,y) \wedge \sim \alpha(x,y) \Rightarrow Q(x,y)}{\{\{P(x,y)\}\}\texttt{while}\ \alpha(x,y)\ \texttt{do}\ S\ \texttt{od}\ \{\{Q(x,y)\}\}}$$

Ahol W egy $<: W \times W$ rendezési relációval (tranzitív, irreflexív, antiszimmetrikus) jól rendezett halmaz, azaz nincs benne $\ldots < w_1 < w_0$ sorozat.

6. Tétel. *Strukturált programok esetén a Floyd-módszer ekvivalens a Hoare-módszerrel.*

1.3.6. A Floyd-módszer

???kell?

1.4. Párhuzamos és osztott programok. Szintaxis, szemantika, helyesség bizonyítására szolgáló módszerek. Párhuzamos rendszerek alapmodelljei: kölcsönös kizárás, termelő-fogyasztó rendszer, író-olvasó modell. Kiéheztetésmentesség.

1.4.1. Párhuzamos programok szintaxisa, szemantikája

32. Definíció. Párhuzamos végrehajtás: *a* ***parbegin*** *...* ***parend*** *– blokk.*

- Szintaxisa: ***parbegin*** $S_1 || \ldots || S_n$ ***parend***, *ahol* $S_1, \ldots, S_n$ *párhuzamos programok.*
- Szemantikája:
  - $S_1, \ldots, S_n$ *párhuzamosan hajtódnak végre.*

– Fair ütemezés: *Ha egy művelet előfordulása végtelen sokszor megtörténhet egy végrehajtási sorozatban, akkor az végre is fog hajtódni.*

– *A blokk akkor ér véget, ha mindegyik S_i véget ért.*

33. Definíció. Kommunikációs utasítás: *a párhuzamosan futó folyamatok kommunikációja osztott változókon keresztül történhet. Ezek változók, melyeket több folyamat is használ.*

34. Definíció. Várakoztató utasítás: *az* `await`.

- Szintaxisa:
  - `await` α `then` S `ta` $\Leftrightarrow$ $\langle \alpha \to S \rangle$ *: együttműködés*
  - `await` *true* `then` S `ta` $\Leftrightarrow$ $\langle S \rangle$ *: kölcsönös kizárás*
  - `await` α `then` $SKIP$ `ta` $\Leftrightarrow$ `wait` α *: szinkronizáció*
- Szemantikája: *Ha az őrfeltétel nem teljesül, a végrehajtás az* `await`*-nél vár. Ha a várakozó folyamatok közül valamelyikeknek az őrfeltétele igazzá válik, véletlenszerűen kiválasztásra kerül az egyik, és S atomi módon végrehajtódik.*

Az atomi módon végrehajtandó utasítások a következők:

- logikai kifejezés kiértékelése,
- változó olvasása vagy írása, valamint
- az `await` utasítás

A programok úgy viselkednek, mintha az atomi utasítások valamilyen sorrendben szekvenciálisan futnának le, ez az *összefésüléses szemantika.*

1.4.2. Helyességbizonyítás

Az *Owicki–Gries* módszer a Hoare-módszer kiterjesztése.

35. Definíció. *Az* `await` *szabály (parciális és teljes helyességhez is).*

$$\frac{\{Q \wedge \alpha\} S \{R\}}{\{Q\} \ \texttt{await} \ \alpha \ \texttt{then} \ S \ \texttt{ta} \ \{R\}}$$

36. Definíció. *A* `parbegin` $\ldots$ `parend` *szabály (parciális helyességhez).*

$$\frac{\{Q_1\}S_1\{R_1\}, \ldots, \{Q_n\}S_n\{R_n\}, NI, *}{\{\wedge_{i=1}^{n} Q_i\} \ \texttt{parbegin} \ S_1 || \ldots || S_n \ \texttt{parend} \ \{\wedge_{i=1}^{n} R_i\}},$$

ahol

- NI = *Az előző bizonyítások nem interferálnak.*
- $*$ = **parbegin** $S_1||\ldots||S_n$ **parend**-*ben minden* **await**-*en kívüli értékadás betartja az* egy kritikus hivatkozás *szabályát (egy értékadáson belül nem szabad ugyanazt a közös változót olvasni és írni is, pl.* $x := x + 1$*)*

A következő két definícióban *utasítás* alatt vagy `await` utasítást, vagy `await` utasításon kívüli értékadást értünk.

37. Definíció. Utasítás NI bizonyítással: *Az* u *utasítás nem interferál a* $\{Q\}S\{R\}$ *bizonyítással, ha annak egyetlen köztes feltételét sem rontja el, azaz:*

$$\forall\{q\}s\{r\} \text{ tételre a } \{Q\}S\{R\} \text{ bizonyításból:}$$
$$\{q \wedge pre(u)\}u\{q\} \ \wedge \ \{r \wedge pre(u)\}u\{r\}$$

38. Definíció. Utasítások NI: *A* $\{Q_1\}S_1\{R_1\}, \ldots, \{Q_n\}S_n\{R_n\}$ *bizonyítások nem interferálnak, ha*

$$\forall i \in [1..n], u \in S_i : u \ NI \ \{Q_j\}S_j\{R_j\} \qquad (j \neq i \in [1..n])$$

39. Definíció. *A* **parbegin** *...* **parend** *szabály (teljes helyességhez).*

$$\frac{\{\{Q_1\}\}S_1\{\{R_1\}\}, \ldots, \{\{Q_n\}\}S_n\{\{R_n\}\}, NI, *, H}{\{\{\wedge_{i=1}^n Q_i\}\} \ \textbf{parbegin} \ S_1||\ldots||S_n \ \textbf{parend} \ \{\{\wedge_{i=1}^n R_i\}\}},$$

ahol

- NI = *Az előző bizonyítások nem interferálnak teljes helyességi értelemben.*
- $*$ = *mint fent*
- H = **parbegin** $S_1||\ldots||S_n$ **parend** *holtpontmentes*

A következő két definícióban *utasítás* alatt vagy `await` utasítást, vagy `await` utasításon kívüli értékadást értünk.

40. Definíció. Utasítás NI bizonyítással: *Az* u *utasítás nem interferál a* $\{\{Q\}\}S\{\{R\}\}$ *bizonyítással, ha annak egyetlen köztes feltételét sem rontja el, és egyik ciklus terminálófüggvényét sem növeli, azaz:*

$$\forall\{\{q\}\}s\{\{r\}\} \text{ tételre a } \{\{Q\}\}S\{\{R\}\} \text{ bizonyításból:}$$
$$\{\{q \wedge pre(u)\}\}u\{\{q\}\} \ \wedge \ \{\{r \wedge pre(u)\}\}u\{\{r\}\}$$

és

$$\forall \text{ ciklusra az } S\text{-ből: } \{f = c \wedge pre(u)\}u\{f \leq c\},$$

ahol f *a ciklus terminálófüggvénye a* $\{\{Q\}\}S\{\{R\}\}$ *bizonyításban.*

41. Definíció. Utasítások NI: *ugyanaz, mint parciális helyességnél.*

42. Definíció. Holtpontmentes program: *Az S párhuzamos program a Q előfeltétellel holtpontmentes, ha nincs olyan végrehajtása, mely Q-ból indulva holtpontba jutna.*

7. Tétel. Elégséges feltétel párhuzamos program holtpontmentességéhez: *Legyen S egy párhuzamos program. Címkézzük meg az utasításait:*

- *A legkülső szinten levő* `await`*-eket, (tehát amik nincsenek* `parbegin` ... `parend` *blokkban):*

$$A_j : \texttt{await}\ \alpha_j\ \texttt{then}\ B_j\ \texttt{ta} \qquad (j \in [1..n])$$

- *A legkülső szinten levő* `parbegin` ... `parend` *blokkokat:*

$$c_k : \texttt{parbegin}\ S_{k1} || \ldots || S_{km_k}\ \texttt{parend} \qquad (k \in [1..m])$$

Tekintsük az S parciális helyességének egy bizonyítását (elő- és utófeltételek előállításához)!

$$D(S) := \bigvee_{j=1}^{n} (pre(A_j) \wedge \neg\alpha_j) \vee \bigvee_{k=1}^{m} D_1(c_k)$$

(Szemléletesen: valamelyik `await` *vagy* `parbegin-parend` *várakozik.)*

$$D_1(c_k) := \bigwedge_{i=1}^{m_k} (post(S_{ki} \vee D(S_{ki})) \wedge \bigvee_{i=1}^{m_k} D(S_{ki})$$

(Szemléletesen: mindegyik ág véget ért, vagy vár, és valamelyik vár.)
Ekkor $D(S) = false \Rightarrow S$ holtpontmentes.

1.4.3. Osztott programok szintaxisa, szemantikája

Osztott és párhuzamos program közötti különbség.

- párhuzamos programnál a kommunikáció gyakori és gyors (itt: kommunikáció közös változókkal),
- osztott programoknál a kommunikáció ritka és költséges (itt: kommunikáció üzenetküldéssel).

A kommunikációs csatorna utasításai.

- Őrfeltételes küldő utasítás
  - Szintaxisa: $b\texttt{;}c\texttt{!}e$
  - Szemantikája: A program az utasításhoz érve megvárja a b őrfeltétel teljesülését, majd a c csatornára küldi az e értéket. Az utasítás blokkolódik, ha a túloldalon nincs várakozó fogadó utasítás.
  - Jelölés: $c\texttt{!}e \ := \ true\texttt{;}c\texttt{!}e$
- Őrfeltételes fogadó utasítás
  - Szintaxisa: $b\texttt{;}c\texttt{?}x$
  - Szemantikája: A program az utasításhoz érve megvárja a b őrfeltétel teljesülését, majd beolvas egy értéket a c csatornáról az x változóba. Az utasítás blokolódik, ha a túloldalon nincs várakozó küldő utasítás.
  - Jelölés: $c\texttt{?}x \ := \ true\texttt{;}c\texttt{?}x$

43. Definíció. Osztott program: *a* ***parbegin*** $S_1||\ldots||S_n$ ***parend*** *program osztott, ha:*

- *Az* $S_1, \ldots, S_n$ *programok nem tartalmaznak közös változókat:*

$$var(S_i) \cap var(S_j) = \emptyset \quad (i \neq j \in [1..n])$$

- *A folyamatok pontról-pontra összekötöttek:*

$$channel(S_i) \cap channel(S_j) \cap channel(S_k) \neq \emptyset \quad (i, j, k \in [1..n])$$

1.4.4. Párhuzamos rendszerek alapmodelljei

Kölcsönös kizárás

Feladat: n folyamat használ 1 erőforrást, de egyszerre csak egyikük használhatja. Az erőforrás használata a kritikus szakasz. A folyamatok ciklikusan vannak kritikus és nem kritikus szakaszban.

Peterson megoldása $n = 2$-re.

s:=1; y_1:=0; y_2:=0;
`parbegin` $S_1||S_2$ `parend`;

S_1 :

while true **do**

{nem kritikus szakasz}
y_1:=1; s:=1;
`wait` $y_2 = 0 \vee s \neq 1$
{kritikus szakasz}
$y_1 := 0$;
end while

$$S_2 :$$

while true **do**
{nem kritikus szakasz}
y_2:=1; s:=2;
`wait` $y_1 = 0 \vee s \neq 2$
{kritikus szakasz}
$y_2 := 0$;
end while

Dijkstra megoldása.
s:=1; {az erőforrást egyszerre 1 folyamat használhatja}
`parbegin` $S_1 || \ldots || S_n$ `parend`;

$$S_i :$$

while true **do**
{nem kritikus szakasz}
P(s);
{kritikus szakasz}
V(s);
end while

$$\mathrm{P}(s) :$$
$$\langle s > 0 \to s := s - 1 \rangle$$
$$\mathrm{V}(s) :$$
$$\langle s := s + 1 \rangle$$

Termelő–fogyasztó rendszer

Feladat: n termelő és m fogyasztó folyamat használ egy k kapacitású tárolót.

Megoldás.
s:=0; {ennyi áru van éppen raktáron}
`parbegin` $T_1 || \ldots || T_n || F_1 || \ldots || F_m$ `parend`;

T_i :

while true **do**
  {termelés}
  $\langle s < k \to berak; s := s + 1 \rangle$;
end while

F_i :

while true **do**
  $\langle s > 0 \to kivesz; s := s - 1 \rangle$;
  {feldolgozás}
end while

Író–olvasó modell

Feladat: n író és m olvasó folyamat használ egy erőforrást. Egyszerre legfeljebb egy folyamat írhat, ami közben tilos olvasni. Egyszerre több folyamat is olvashat, de közben tilos írni.

Megoldás. A megoldásban kiéheztetés előfordulhat: az olvasók monopolizálhatják a rendszert.

w:=1; {író szemafor}
sr:=1; {olvasó szemafor}
$rcnt$:=0; {éppen olvasók száma}
`parbegin` $W_1||\ldots||W_n||R_1||\ldots||R_m$ `parend`;

W_i :

while true **do**
  {dolgozik}
  P(w);
  {ír}
  V(w);
end while

R_j :

while true **do**
  P(sr);

$rcnt := rcnt + 1$;
if $rcnt = 1$ **then**
P(w)
end if
V(sr);
{olvas}
P(sr);
$rcnt := rcnt - 1$;
if $rcnt = 0$ **then**
V(w)
end if
V(sr);
{dolgozik}
end while

1.4.5. Kiéheztetés-mentesség

44. Definíció. Kiéheztetés-mentesség: *S párhuzamos program, melynek folyamatai ciklikusak. A rendszer egy közös erőforrással rendelkezik, melyet a folyamatok időnként használnak. Ekkor S kiéheztetés-mentes, ha nincs a folyamatoknak egy olyan csoportja, melyek monopolizálhatják az erőforrást.*

45. Definíció. Monopolizálás: *A folyamatok egy csoportja monopolizálta az erőforrást, ha mindig van köztük olyan, amelyik lekötve tartja az erőforrást, miközben a csoporton kívüli folyamatok egy része folyamatosan várakozik arra.*

46. Definíció. Kiéheztetés szempontjából veszélyes utasítások: *Utasításoknak egy $\{A_1, \ldots, A_k, B_1, \ldots, B_l, C_1, \ldots, C_m\}$ $(k, l \geq 1; m \geq 0)$ halmaza kiéheztetés szempontjából veszélyes, ha mind különböző folyamatól való, és*

- *$A_1, \ldots, A_k$ (az éhes utasítások)* `await` *α_i* `then`*... utasítások, melyek folyamatosan várakozhatnak az erőforrás használatára, mivel folyamatok egy másik csoportja kizárja, hogy α_i teljesüljön;*
- *$B_1, \ldots, B_l$ (az éheztető utasítások)*
  - *vagy* `await` *β_i* `then`*... utasítások, melyek az erőforrás használatát őrzik, de feltételük teljesül, még ha a csoport más tagja éppen használják is az erőforrást;*
  - *vagy az erőforrás használatát jelentő utasítások;*
- *$C_1, \ldots, C_m$ (a kész utasítások) üres utasítások, melyeknél valamelyik S_i folyamat befejeződött.*

8. Tétel. Elégséges feltétel a kiéheztetés-mentességhez: *S párhuzamos program ciklikus folyamatokkal. S teljes helyességének bizonyítása adott. Ekkor ha $\forall\{A_1, \ldots, A_k, B_1, \ldots, B_l, C_1, \ldots, C_m\}$ kiéheztetés szempontjából veszélyes utasításhalmazra*

$$\begin{aligned} pre(A_1) \wedge \neg\alpha_1 \wedge \ldots pre(A_k) \wedge \neg\alpha_k \wedge \quad & \textit{(azaz } A_i\textit{-k blokkoltak)} \\ pre(B_1) \wedge \beta_1 \wedge \ldots pre(B_l) \wedge \beta_l \wedge \quad & \textit{(azaz } B_i\textit{-k szabadok)} \\ post(C_1) \wedge \ldots \wedge post(C_m) \quad & \textit{(azaz } C_i\textit{-k befejeződtek)} \\ = false & \end{aligned}$$

teljesül, akkor S kiéheztetés-mentes.

1.5. Nemdeterminisztikus programok: szintaxis, szemantika. Osztott rendszerek alapfogalmai, nyelvi eszközei. Kommunikációs csatorna. Komponens alapú programozás. Ágensrendszerek, kontraktusvezérelt programfejlesztés.

1.5.1. Nemdeterminisztikus programok: szintaxis, szemantika

Lásd 3. tétel.

1.5.2. Osztott rendszerek

A párhuzamos programhoz képest (ami közös változókat használ), az osztott rendszerek kommunikációja költségesebb (csatornákkal történik).

Az osztott rendszer külön dolgozó, önálló erőforrásokkal rendelkező komponensek halmaza, melyek időről időre üzeneteket küldenek egymásnak. Fajtái: szinkron, aszinkron és vegyes.

Nyelvi eszközei: új folyamatok indítása, csatornautasítások (küldés, fogadás).

Kommunikációs csatornák

Az üzenetküldés és -váltás fajtái:

- szinkron üzenetküldés: a küldő blokkolódik
- aszinkron üzenetküldés: a küldő nem blokkolódik
- késleltetett szinkron üzenetváltás: szinkron üzenetküldés és aszinkron válasz

- visszahívásos üzenetváltás: aszinkron üzenetküldés és aszinkron válasz

A csatornák általában típustalanok, kétirányúak és full duplexek.

47. Definíció. *Egy kétirányú csatorna* full duplex*, ha egyidejűleg mindkét irányban használható. Ellenkező esetben* half duplex.

1.5.3. Komponens alapú programozás

A komponens alapú programozás egy szoftvertervezési módszertan. Lényege, hogy a rendszert funkcionális vagy logikai komponensekre bontjuk, és jól definiált interfésszel látjuk el őket a kommunikáció számára.

A szoftverkomponens egy szoftver olyan része, amely előre specifikált szolgáltatásokat nyújt az interfészen keresztül. Általában egy osztály vagy osztálykönytár.

1.5.4. Ágensrendszerek

A multiágens-rendszer több, egymással kölcsönhatásban álló ágensből álló rendszer.

Egy szoftverágens tulajdonságai:

- Nem kívülről hívják a metódusait, hanem mindig fut (perzisztens). Ciklikusan megnézi a környezetét, és maga dönti el, hogy mit reagáljon rá (autonóm).
- Az ágensek kommunikálhatnak egymással: együttműködhetnek, versenghetnek.
- Viselkedésük célorientált.

A multiágens-rendszerek fontos tulajdonsága az *emergencia*: az ágensek egyszerűek, a kölcsönhatásuk is egyszerű, rendszerszinten mégis komplex jelenségek figyelhetők meg, melyek nehezen jósolhatók meg az ágensek egyszerű felépítéséből.

1.5.5. Kontraktusvezérelt programfejlesztés

A kontraktusvezérelt programfejlesztés egy szoftvertervezési módszer, mely megköveteli, hogy a programot precíz, verifikálható specifikációval lássuk el, azaz már a tervezés során minden eljáráshoz formális elő- és utófeltételeket, minden osztályhoz invariánst írunk.

Ekkor az osztályok metódusaival a következő szerződésben állunk: ha teljesítjük az előfeltételt és az invariánst, a metódus visszatérésekor teljesül az utófeltétel és az invariáns.

A *design by contract* kifejezést Bertrand Meyer (az Eiffel tervezője) találta ki. DBC-re tanít a bevprog, a pp és a progmód.

Ha egy programozási nyelvben lehetőség van az előfeltétel, utófeltétel és invariáns megadására (pl. Eiffel), ekkor ez lehetővé teszi a feltételek ellenőrzését fordítási időben.

1.6. Típusosztály specifikációja konkurens környezetben. Absztrakt specifikációs módszer esemény elvű megközelítés alapján. Alapvető szinkronizációs feladatok megoldásai. Az absztrakt specifikáció elemzése.

1.6.1. Típusosztály specifikációja konkurens környezetben

Konkurens környezetben az objektumok metódusai párhuzamosan is hívhatóak. Ez elronthatja az osztály invariánsát és a program helyességét, ezért az objektumot további védelemmel látjuk el a helytelen használat ellen: meghatározzuk azt is, hogy a típusosztály metódusai milyen sorrendben hívhatók, ez a szinkronizáció.

A típusosztály specifikációja konkurens környezetben kiegészül a szinkronizáció specifikációjával, melyet a `syn` kulcsszó után helyezünk el absztrakt és konkét szinten egyaránt (`EXP` és `BOD`).

A szinkronizáció többféleképpen specifikálható:

- Útkifejezésekkel
- Kontrollmodulokkal
- Ütemezési predikátumokkal
- Eseményelvű specifikációval
- Állapottér módszerével

1.6.2. Eseményelvű specifikáció

Esemény

Egy objektum minden műveletének végrehajtásában 3 fázist, vagyis eseményt különítünk el:

- *request*: igénybejelentés a művelet végrehajtására;
- *enter*: a művelet végrehajtásának megkezdése;
- *exit*: a művelet végrehajtásának befejezése.

A műveleti azonosító $f[i](x)$ alakú, ahol

- f a művelet neve;
- i a művelet végrehajtásának sorszáma;
- x a művelet szinkronizáció szempontjából fontos paramétere vagy paraméterei (opcionális).

Például:

- $f[i]$ jelentése: az f i-edik aktiválása.
- $f[i](x)$ jelentése: az f i-edik aktiválása x argumentummal.
- $f[i](x).ex$ jelentése: $f[i](x)$ exit fázisa.

Rendezési klóz

Minden művelet minden végrehajtásához 3 eseményt definiáltunk. Feltesszük, hogy egyszerre csak egy ilyen esemény léphet fel, és így az eseményeken az időbeliség alapján teljes rendezés érvényes.

A *rendezési klóz* alakja: $A \to B$, ahol A és B események. Jelentése: A megelőzi B-t.

A rendezési klóz alakja *argumentumkorlátozással*:

$$R(x, y) \wedge f[i](x).\chi \to g[i](y).\varrho$$

Jelentése: $f[i](x).\chi$ megelőzi $g[i](y).\varrho$-t, és az argumentumokra fennáll az $R(x, y)$ reláció.

Specifikáció

Az eseményelvű specifikáció egy rendezési klózokból előálló elsőrendű logikai formula.

Jelentése: a futtatókörnyezet feladata biztosítani, hogy a logikai formula teljesüljön a program futása során. Ha egy *enter* esemény hatására a szinkronizációs specifikáció elromlana, a művelet megkezdése előtt a hívó folyamat blokkolódik.

Természetes axiómák, melyeket a specifikáció megadásánál nem szokás kiírni, de igazak:

$$\forall i : f[i].req \to f[i].ent \to f[i].ex$$

$$\forall i < j : f[i].req \to f[j].req$$

1.6.3. Alapvető szinkronizációs feladatok megoldásai

A következő formulák $\forall i, j$-re értendők.

Kölcsönös kizárás

p és q egyszerre nem hajtódhat végre (de több p vagy több q igen). Megoldása:

$$p[i].ex \to q[j].ent \quad \vee$$
$$p[j].ex \to q[i].ent$$

Író–olvasó probléma

w-vel párhuzamosan semmi sem futhat. Több r futhat párhuzamosan. Megoldás:

$$w[i].ex \to w[i+1].ent \quad \wedge$$
$$(w[i].ex \to r[j].ent \vee r[j].ex \to w[i].ent)$$

First come first served

q és p közül a korábban meghívott hajtódik először végre. Megoldás:

$$p[i].req \to q[j].req \quad \Leftrightarrow \quad p[i].ent \to q[j].ent$$

Prioritás

p-nek prioritása van q-val szemben. Megoldás:

$$p[i].req \to q[j].ent \quad \Rightarrow \quad p[i].ent \to q[j].ent$$

1.6.4. Az absztrakt specifikáció elemzése

Konzisztencia

Az objektumok két védőgyűrű veszi körül.

- A belső védőgyűrű statikus. Akkor engedi végrehajtani a műveletet, ha az az argumentumaival értelmezve van az adott objektumon. A művelet előfeltételének felel meg, és az `eqns`-ben specifikáljuk.
- A külső védőgyűrű dinamikus. Akkor engedi végrehajtani a műveletet, amikor a szinkronizációs specifikáció szerint meg lehet kezdeni a művelet végrehajtását. A `syn`-ben definiáljuk.

48. Definíció. *Egy típusosztály absztrakt specifikációja* konzisztens, *ha a külső védőgyűrű feltételéből következik a belső védőgyűrű feltétele. Eseményelvű specifikáció esetén ez azt jelenti, hogy minden lehetséges lefutási sorrendre, az osztály minden f műveletére, annak minden i-edik aktiválására $S_{f[i].ent} \Rightarrow pre(f)$ teljesül, ahol $S_{f[i].ent}$ az eseményelvű szinkronizációs specifikáció igazságértéke, ha az $f[i].ent$ esemény bekövetkezne.*

Holtpontmentesség

Tekintsünk egy párhuzamos rendszert, melynek folyamatai egy típusosztály egy objektumát használják, ettől eltekintve azonban nem várhatnak egymásra!

49. Definíció. *Egy típusosztály szinkronizációs specifikációja* holtpontmentes*, ha a fent leírt párhuzamos rendszer mindig holtpontmentes.*

9. Tétel. Szükséges és elégséges feltétel az absztrakt szinkronizációs specifikácó holtpontmentességéhez. *Adott egy típusosztály absztrakt specifikációja. A szinkronizáció eseményelvű specifikációval adott, és nem tartalmaz argumentumkorlátozást. Hozzuk az S szinkronizációs specifikációt diszjunktív normálfomára: $S = S_1 \vee \ldots \vee S_n$! Ekkor minden S_k rendezési klózok, vagy azok negáltjainak konjunkciója. A negált rendezési klózokat alakítsuk át, felhasználva a teljes rendezést: $\neg A \to B = B \to A$! Rendeljünk minden S_k-hoz egy irányított gráfot, melynek*

- csúcsai: $\forall f[i]$*-hez három különböző csúcs:* $f[i].req$, $f[i].ent$, $f[i].ex$*;*
- élei: $\forall f[i].\chi \to g[j].\varrho \in S_k$*-ra behúzzuk a gráfban a megfelelő irányított élt, a természetes axiómákból származókat is beleértve!*

Ekkor az absztrakt szinkronizációs specifikáció holtpontmentes $\Leftrightarrow$ $\forall S_k$*-hoz rendelt irányított gráf körmentes.*

1.7. Az absztrakt szinkronizációs specifikáció implementációja, transzformációs szabályok. Állapottér bevezetése, kapufeltételek, szinkronizációs specifikációk konkrét megvalósítása. Holtpontmentesség, konzisztencia.

1.7.1. Konkrét szinkronizációs specifikáció

Tegyük fel, hogy megírtuk és elemeztük a típusosztály absztrakt specifikációját (`EXP`). Tegyük fel, hogy a szinkronizációtól eltekintve megírtuk és elemeztük a konkrét specifikációt (`BOD`).

1.7.2. Az állapottér módszere

Az állapottér módszere a szinkronizáció konkrét megadására szolgál. Egy objektum minden g műveletének végrehajtásában három fázist különböztetünk meg, és mindegyikhez egy számlálót rendelünk:

- $count(g.req)$: a g műveletre eddig érkezett végrehajtási kérelmek száma;
- $count(g.ent)$: eddig hányszor engedélyeztük a g művelet végrehajtásának megkezdését;
- $count(g.ex)$: eddig hányszor fejeződött be a g művelet.

További jelölések:

- $wait(g) := count(g.req) - count(g.ent)$
- $exec(g) := count(g.ent) - count(g.ex)$

50. Definíció. Szinkronizációs állapottér: *a fenti számlálók lehetséges értékeinek Descartes-szorzata. Ha az objektumnak n művelete van, akkor a szinkronizációs állapottér* $3n$*-dimenziós.*

A szinkronizációs állapotot minden műveleti fázis végrehajtásakor módosítani kell, vagyis léptetni a megfelelő számlálót. Az állapot módosítása kritikus szakasz, így a kölcsönös kizárást biztosítani kell.

51. Definíció. Kapufeltétel: *Egy művelet request és exit eseménye azonnal végrehajtható, az enter eseménynek azonban feltétele van: csak akkor hajtható végre, ha belépésük összeegyeztethető a szinkronizáció követelményeivel. Ez a művelet kapufeltétele. Jele:* $k(g.ent)$ *= a* g *művelet kapufeltétele.*

A típusosztály konkrét eljárásait kiterjesztjük a szinkronizációs állapot módosításával, ez kerül a BOD/`syn` specifikációjába:

procedure $g(x)$ **is**
begin
  $\langle count(g.req) := count(g.req) + 1 \rangle$
  $\langle k(g.ent) \rightarrow count(g.ent) := count(g.ent) + 1 \rangle$
  $\{g(x)$ eredeti kódja$\}$
  $\langle count(g.ex) := count(g.ex) + 1 \rangle$
end

1.7.3. Kontrollmodulok

A *kontrollmodul* a kapufeltételek rendszerezett leírására szolgál. Az azonos kapufeltételű műveletek közös *partícióba* sorolhatók. Szintaxisa:

control module *modul neve*
begin
  $g_1, \ldots, g_n$: **procedure**; {A típusosztály műveleteinek felsorolása}
  queue $part_1/\ldots/part_m$; {Partíciók felsorolása.}

condition($part_1$): $kapufelt_1$
⋮
condition($part_m$): $kapufelt_m$
end

A kontrollmodulokkal a szinkronizációs specifikáció öröklődésnek is alávethető. Az öröklődés szintaxisa:

inherit *modulnév*;
redefined
condition($part_i$): $kapufelt_i$
⋮
rd;

1.7.4. A konkrét specifikáció elemzése

Konzisztencia

Ugyanaz, mint az absztrakt specifikáció elemzésénél (1.6.4. fejezet).

A konkrét szinkronizációs specifikáció *konzisztens*, ha az osztály minden g műveletére $k(g.ent) \Rightarrow pre(g)$ teljesül.

Holtpontmentesség

A definíció ugyanaz, mint az absztrakt specifikációnál.

10. Tétel. Elégséges feltétel a konkrét szinkronizációs specifikáció holtpontmentességére. *Adott egy típusosztály konkrét specifikációja, ebben a szinkronizáció az állapottér módszerével.*

- *A típusosztály műveletei:* $G = \{g_0, \ldots, g_n\}$
- *Egy* g_i *művelet blokkoltsági feltétele:*

$$B(g_i) := \neg k(g_i.ent) \wedge wait(g_i) > 0 \wedge exec(g_i) = 0.$$

- *Egy művelethalmaz blokkoltsági feltétele:*

$$\forall\, \emptyset \neq H \subseteq G: \quad B(H) := \bigwedge_{g \in H} B(g).$$

- *Ha* $B(H)$ = *hamis, vagy* $B(H)$*-ban van hivatkozás* H*-n kívüli műveletre is, akkor azt mondjuk, hogy* H *nem jelöl holtponthelyzetet.*

Ekkor $\forall\, \emptyset \neq H \subseteq G$: H *nem jelöl holtponthelyzetet* $\Rightarrow$ *a konkrét szinkronizációs specifikáció holtpontmentes.*