Az objektumorientált öröklodés és polimorfizmus funkcionális megközelítése

Kovács Lehel István
Babes-Bolyai Tudományegyetem
Számítógépes Rendszerek Tanszék, tanársegéd

According to the purely functional paradigm, the value of an expression depends only on the values of its constant subexpressions, if any. In this paper we introduce this principle in the object-oriented paradigm. The simplicity and power of functional languages is due to properties like pure values, first-class functions, and implicit storage management. We must extend these properties with a strong type-system. The values must be typed, the type system used for this purpose is the higher-order, explicitly-typed, polymorph lambda-calculus with subtyping, called $F_?^?$ *.This type-system must be prepared for basic mechanisms of object-oriented programming: encapsulation, message passing, subtyping and inheritance. Polymorph functions arise naturally when lists are manipulated and lists with elements of any types can be accomplished by a straightforward generalization of inheritance. Interesting questions are also, how to introduce the object-oriented inheritance, the subtyping mechanism, and polymorphism.*

1. Bevezetés

Jelen dolgozatban bevezetjük a tisztán funkcionális paradigmát az objektumorientált programozásba és az öröklodés egyes válfajainak megoldásait vizsgáljuk. A funkcionális nyelvek egyszerusége és ereje a tiszta értéku jellemzokben, elso-osztályú függvényekben, és az implicit tárhely-gazdálkodásban rejlik. Mi kibovítjük ezeket a jellemzoket egy eros típusrendszerrel, amelyet $F_?^?$ -nak nevezünk.

A szakirodalom három objektummodellt ismer: a *rekord-modellt*, az *exisztenciális-típusmodellt*, és az *axiomatikus modellt* [2]. Az objektumok reprezentációjára a rekurzívan definiálható rekord-modellt használjuk, melyben az osztályok adatokból és metódusokból álló rekordként vannak ábrázolva. A típusos lambda-kalkulust és az $F_?^?$ típusrendszert használva eljutunk a tisztán funkcionális objektumorientált paradigmához. Az $F_?^?$ típusrendszer azonban rekordokat használ. A cikk célja az, hogy a típusegyezési relációkat halmazokkal írjuk le, így a fordítóprogram egy egyszeru IN muvelet segítségével eldöntheti, hogy milyen esettel áll szemben.

2. Az $F_?^?$ típus-rendszer

Kezdetben volt a *lambda-kalkulus*. Később megszületett a *típusos lambda-kalkulus*. Ezt Church vezette be és a szakirodalomban F_1, F_2 nevet viseli Girard és Reynolds másodrendu típusos lambda-kalkulusának megfeleloen. A rendszert tovább lehet bovíteni, így jön létre az F_3, amely F_2-bol származtatható, olyan típuskonstrukciók segítségével, amelyek *kind* := * | * ? *kind* alakban új típusokat hoznak be. Iteratívan folytatva a sort, rendre magasabb fokú *kind* szerkezetet használva, képezhetjük az F_4, F_5, ... rendszereket. Mindezek egyesítése képezi az $F^?$ rendszert, amelyben *kind* := * |*kind* ? *kind* a típuskonstrukció szintaxisa.

A [3]-ban bemutatott általánosítás kibovíti az $F^?$ rendszert az altípusképzés fogalmával, így születik meg az $F_?^?$ eros típusrendszer. Ennek a rendszernek egy további általánosítása [5] a *rekord* fogalmának a bevezetése, amely segítségével már objektumorientált rendszereket is tudunk modellezni [1, 4].

A *rekord típus*, valamint a *rekord term* képzésének szintaktikus szabályai a következok:

<típus> := {|<név$_1$>:<típus$_1$>, …, <név$_n$>:<típus$_n$>|}
<term> := {<név$_1$>=<term$_1$>, …, <név$_n$>=<term$_n$>}
| <term>.<név>

Felkészítjük az $F_?^?$ típusrendszert *rekord* típusok ábrázolására.

2.1. Szabály: *kind* szabály *rekord* típusokra:

? + <típus$_i$> ? * ? *i*
? ? + {|<név$_1$>:<típus$_1$>, …, <név$_n$>:<típus$_n$>|} ? *

2.2. Szabály: *bevezetési* szabály *rekord* típusokra:

? + <$term_i$>:<$típus_i$> ? * ? *i*

? ? + {<$név_1$>=<$term_1$>, …, <$név_n$>=<$term_n$>}:{|<$név_1$>:<$típus_1$>, …, <$név_n$>:<$típus_n$>|}

2.3. Szabály: *eliminációs* szabály *rekord* típusokra:

<$term$> ? {<$név_1$>=<$term_1$>, …, <$név_n$>=<$term_n$>}:

? + <$term$>:{|<$név_1$>:<$típus_1$>, …, <$név_n$>:<$típus_n$>|}

? ? + <$term$>.<$név_1$>:<$típus_i$>

2.4. Szabály: *altípusképzo* szabály *rekord* típusokra:

? Az altípus *rekord* legalább ugyanazokkal a mezokkel rendelkezik, mint az a *rekord*, amelybol képeztük,

? Az altípus *rekord* minden mezejének típusa altípusa kell, hogy legyen annak a *rekord-nak* megfelelo mezo típusának, amelybol képeztük, ha létezik ez a mezo.

{$név_{1,1}$, …,$név_{1,n}$} ? {$név_{2,1}$, …,$név_{2,n}$}

? + <$típus_{1,i}$> ? <$típus_{2,i}$> ? <$név_{1,i}$> = <$név_{2,i}$>

? ? + {|<$név_{1,1}$>:<$típus_{1,1}$>, …, <$név_{1,n}$>:<$típus_{1,n}$>|} ?

{|<$név_{2,1}$>:<$típus_{2,1}$>, …, <$név_{2,n}$>:<$típus_{2,n}$>|}

A fenti általános $F_?^?$ típusrendszert *rekordok* segítségével definiáltuk. Jelen dolgozat célja az, hogy az $F_?^?$ típusrendszert *halmazok* segítségével definiálja.

3. Általánosított muveletek típushalmazokra

Legyenek $v_1, v_2, ..., v_i$ változók, $v = \{v_1, v_2, ..., v_i\}$ változók halmaza, valamint $V = \{V_1, V_2, ..., V_i\}$ a változók típusosztálya, ? $k = 1, i$-re a v_k változó V_k típusú.

Hasonlóan legyenek $v'_1, v'_2, ..., v'_j$ változók, $v' = \{v'_1, v'_2, ..., v'_j\}$ változók halmaza, valamint $V' = \{V'_1, V'_2, ..., V'_j\}$ a változók típusosztálya, ? $k = 1, j$-re a v'_k változó V'_k típusú.

Általánosítjuk a halmazelmélet „részhalmaz”, ? és ? muveleteit a következoképpen:

3.1. Definíció: (Típusosztályokra a ? muvelet) *Legyen V* ? *V' akkor és csakis akkor, ha:*

(1.) $j > i$ *és* $V'_1 = V_1, ..., V'_i = V_i$ *(aritmetikai részhalmaz),*
(2.) $j = i$ *és* $V'_1 ? V_1, ..., V'_i ? V_i$
(3.) (1.)-nek és (2.)-nek a kombinációja.

3.2. Definíció: (Típusosztályokra a ? muvelet) *Legyen V* ? *V' akkor és csakis akkor, ha:*

(4.) $j ? i$ *és* $V'_1 = V_1, ..., V'_i = V_i$ *(aritmetikai részhalmaz),*
(5.) $j = i$ *és* $V'_1 ? V_1, ..., V'_i ? V_i$
(6.) (1.)-nek és (2.)-nek a kombinációja.

Hasonlóan általánosítjuk a halmazelméleti „eleme”, ? muveletet a következoképpen:

3.3. Definíció: (Az IN muvelet) *Legyen IN : v ? V* ? *V' ? {false, true}. Az IN muvelet true-t ad vissza akkor és csakis akkor, ha a v változó V típusú. Különben false-ot ad vissza.*

4. Az objektumorientált öröklodés

Ha már definiáltunk egy osztályt, bármikor lehetoségünk van arra, hogy az adott osztályt más osztályok definiálására felhasználjuk, azzal a céllal, hogy a már meglévo kódot újra fel tudjuk használni, illetve azzal a céllal, hogy muködésében kibovítsük, testre szabjuk a már meglévo osztályt. Ez a mechanizmus úgy valósul meg, hogy a második osztályt *leszármaztatjuk* az elso osztályból. Ezt *öröklodésnek* nevezzük, és az osztályok ilyenképpen *öröklodési hierarchiába* szervezhetok. Ilyen értelemben beszélhetünk *ososztályokról* és *leszármazottakról, gyerek osztályokról.* Természetesen egy leszármazott a maga során lehet ososztálya egy másik osztálynak vagy más osztályoknak.

Az öröklodés tulajdonképpen két síkon nyilvánul meg: a leszármazott kiterjeszti az os interfészét a behozott új attribútumokkal, metódusokkal (az osztály, a típus szintjén), de ugyanakkor leszukíti az objektumok fogalmi szintjét (példányosítás).

Ha öröklodésrol beszélünk, definiálnunk kell a *helyettesíthetoség fogalmát is.* A helyettesíthetoség azt jelenti, hogy a származtatott osztály objektumai bármilyen körülmények között helyettesíteni tudják az ososztály objektumait, vagyis a származott osztály felveheti az ososztály szerepét, viselkedését, és nem lehet meg-

különböztetni az ososztály valamelyik példányától, ha hasonló környezetben használjuk. Ez a folyamat természetes, mivel a leszármaztatott osztályban szerepel az ososztály minden adata és metódusa, így bármikor úgy viselkedhet, mint maga az ososztály. Vagy azt is mondhatjuk, hogy az ososztály szerepelhet formális paraméterként bárhol, ahol a leszármazott aktuális paraméterként elofordulhat.

4.1. Definíció: (A helyettesíthetoség fogalma) *A származtatott osztály objektumai bármilyen körülmények között helyettesíteni tudják az ososztály objektumait. Ha C a származtatott osztály, P az ososztály, C = subst(P) azt jelenti, hogy a C bármely példánya használható ott, ahol a P bármely példánya elofordul.*

A helyettesíthetoség fogalmát még *is_a relációnak* is szoktuk nevezni. Ez kifejezi azt, hogy az ostol a leszármazott irányába *specifikálás*, a leszármazottól az os felé pedig *általánosítás* történik. A gyakorlatban, azonban gyakran azért is használjuk az öröklodést, hogy leszukítsük, testreszabjuk az os muködését. Vagy azért is, mert a már meglévo osztályon a konstrukció szempontjából csak keveset kell módosítanunk, és máris egy új leszármazottat nyertünk. Ilyen esetekben nem áll fenn az *is_a reláció*, nem áll fenn a helyettesíthetoség. Fogalmi szinten is elkülönítjük ezeket az öröklodési típusokat. Ha fennáll az *is_a reláció*, akkor a leszármazott *altípusa* (*sub-type*) az osnek, ha nem áll fenn, akkor *alosztálya* (*sub-class*) az osnek.

4.2. Definíció: (Altípus, sub-type) *Az altípus egy olyan osztály, amely kielégíti a helyettesíthetoség fogalmát (C = subst(P)).*

Informálisan, egy *?* típus altípusa *?*-nak (jelölés: *? ? ?*), ha bármely *?* típusú kifejezés használható olyan kontextusban, amely *?* típust igényel. Az altípus képezés szabálya pedig: *? ? ? ? ? ' ? ?'*, akkor és csakis akkor, ha *? ' ? ?*, *és ? ? ?'*. Ezt formalizáljuk úgy, hogy kiterjesztjük a lambda-kalkulus rendszerünket egy *altípus* relációval, a következoképpen: $? + C ? P$, vagyis C a P altípusa a $?$ követelményrendszer fölött.

4.3. Definíció: (Alosztály, sub-class) *Az alosztály egy olyan öröklodéssel létrehozott tetszoleges osztály, amely nem elégíti ki a helyettesíthetoség fogalmát (C ? subst(P)).*

Ezt formalizáljuk úgy, hogy kiterjesztjük a lambda-kalkulus rendszerünket egy *alosztály* relációval, a következoképpen: $? + C \square P$, vagyis C a P alosztálya a $?$ követelményrendszer fölött.

A gyakorlatban mégis mindketto használható, attól függoen, hogy melyik elonyösebb, melyik biztosít gyorsabb kódmódosítást és újrahasználást. *De vigyázzunk, mert ha nem áll fenn az is_a reláció, akkor problémák léphetnek fel (akár fogalmi, akár fizikai szinten – mint a példából is láthatjuk) a helyettesítésekkor.*

Amint az objektumorientált paradigma az osztály fogalmát két részre bontja, *adattagok* és *metódusok*, mi is bevezetjük a következo jelöléseket: legyenek $v_1, v_2,\ldots, v_i$ egy osztály változói (adattagjai), $m_1,m_2,\ldots,m_j$ pedig a metódusai. Legyen $V ? \{V_1, V_2 ..., V_i\}$, $M ? \{M_1, M_2, ..., M_j\}$ a P osztály változóihoz és metódusaikhoz (a metódusok szignatúrája) hozzárendelt típusosztályok, i a változók száma, j a metódusok száma. $P = V ? M$.

Ekkor az altípusképzés szabálya C és P között a következo:

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$
$$V' ? \{V_1', V_2' ..., V_{i'}'\}, \quad M' ? \{M_1', M_2', ..., M_{j'}'\},$$
$$V ? V', \quad M ? M', \; P ? V \square M, \quad C ? V \square M', \; i' ? i, j' ? j,$$
$$C = subst(P),$$
$$? + \{|v_1 : V_1, ..., v_i : V_i, m_1 : M_1, ..., m_j : M_j|\} ? *,$$
$$? + \{|v_1' : V'_1, ..., v'_{i'} : V'_{i'}, m'_1 : M'_1, ..., m'_{j'} : M'_{j'}|\} ? *$$
$$? \quad ? + C ? P$$

Hogy jobban tudjuk formalizálni a C = subst(P) vagy C ? subst(P) eseteket, az öröklodés következo válfajait különböztetjük meg:

4.1. Specializálás

Specializáljuk az ososztályt. Nem változtatjuk meg a meglévo metódusokat, adatokat, de behozhatunk újakat. Ebben az esetben fennáll az *is_a reláció*. Az öröklodés leggyakrabban használt, ideális esete, amely jó

programstruktúrát eredményez. Például a **Halászhajó** a **Hajó**nak egy speciális altípusa, egy olyan hajó, amely rendelkezik a **Hajó**k összes tulajdonságával, de pluszban még **halászni** is tud. Vagy pl. a **TextEditWindow** (olyan ablak, amelyben szöveget tudunk szerkeszteni) a **Window** (általános ablak) speciális esete.

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$
$$V' ? \{V_1', V_2' ..., V'_{i'}, ..., V_{i'}'\}, \quad M' ? \{M_1', M_2', ..., M'_{j'}, ..., M_{j'}'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M', i' ? i, j' ? j,$$

-- új változók és új metódusok kerültek be --

$$? + \{|v_1 : V_1, ..., v_i : V_i, m_1 : M_1, ..., m_j : M_j|\} ? *,$$
$$? + \{|v_1' : V'_1, ..., v'_{i'} : V'_{i'}, m'_1 : M'_1, ..., m'_{j'} : M'_{j'}|\} ? *$$
$$? \quad ? + C ? P$$

4.2. Specifikálás

Ez abban az esetben áll fenn, amikor az os egy általános osztály, a leszármazottak pedig konkrét implementációk. Ezt az esetet használjuk fel a *homogén interfészek* létrehozására is. Minden leszármazott ugyanúgy viselkedik, ugyanolyan nevu metódusokat tartalmaz. Nem hoz be újabb metódusokat. Ebben az esetben is fennáll az *is_a reláció*. Például a **Vonat**nak, mit általános ososztálynak specifikált leszármazottjai a **Személyvonatok**, **Gyorsvonatok**, **InterCity**-k. Semmilyen új metódust nem hoznak be, csak a menetido változik, és persze a jegy ára.

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M'_1, M'_2, ..., M'_j\},$$
$$V' ? \{V_1', V_2' ..., V_i'\}, \quad M' ? \{M_1', M_2', ..., M_j'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M',$$

-- a változók típusa módosul --

$$? + \{|v_1 : V_1, ..., v_i : V_i, m_1 : M'_1, ..., m_j : M'_j|\} ? *,$$
$$? + \{|v_1' : V'_1, ..., v'_{i'} : V'_{i'}, m'_1 : M'_1, ..., m'_{j'} : M'_{j'}|\} ? *$$
$$? \quad ? + C ? P$$

4.3. Konstrukció vagy Reprezentáció

Az os biztosítja a gyerek felépítését, de logikailag más kontextust nem biztosít. Ez a módszer logikailag nem a leghelyesebb, és az *is_a reláció* sem áll fenn. Például **Hidroplán** és **Vízi jármuvek**, vagy ha a **Halmaz** osztályt a **Lista** osztályból származtatjuk (a halmaz egy olyan lista, amiben minden elem csak egyszer fordul elo – konstrukció szempontjából jó, logikailag helytelen). Hasonlóan gyakran elofordul például, hogy a grafikus objektumokat a **Pont** osztályból származtatjuk: a **Kör** az **x**, **y** középpontot tartalmazó **Pont**ot kiterjeszti úgy, hogy behoz egy **r** sugarat (konstrukció szempontjából kényelmes megoldás, de matematikailag helytelen, mert a **Kör** nem **Pont**!).

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$
$$V' ? \{V_1', V_2' ..., V_{i'}'\}, \quad M' ? \{M_1', M_2', ..., M_{j'}'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M', i' ? i, j' ? j,$$

-- új változók, új metódusok, a változók típusa módosul, a metódusok funkcionalitása módosul --

$$? + \{|v_1 : V_1, ..., v_i : V_i, m_1 : M_1, ..., m_j : M_j|\} ? *,$$
$$? + \{|v_1' : V'_1, ..., v'_{i'} : V'_{i'}, m'_1 : M'_1, ..., m'_{j'} : M'_{j'}|\} ? *$$
$$? \quad ? + C \square P$$

4.4. Általánosítás

Általánosítjuk az ost. Újrafelhasználjuk a kódot, újabb metódusokat, adatokat hozhatunk be. Bizonyos esetekben nem lesz helyettesítheto az os, bizonyos esetekben igen. Például az **Vitorlás motorcsónak** általánosítása a **Vitorlás**nak, hisz szükség esetén, ha szélcsend van, motorral is mehet.

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$
$$V' ? \{V_1', V_2' ..., V_{i'}'\}, \quad M' ? \{M_1', M_2', ..., M_{j'}'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M', i' ? i, j' ? j,$$

-- új változók, új metódusok, a változók típusa módosul, a metódusok funkcionalitása általánosabbá válik --

$$? + \{|v_1 : V_1, ..., v_i : V_i, m_1 : M_1, ..., m_j : M_j|\} ? *,$$
$$? + \{|v_1' : V'_1, ..., v'_{i'} : V'_{i'}, m'_1 : M'_1, ..., m'_{j'} : M'_{j'}|\} ? *$$
$$? \quad ? + C ? ? \square P$$

4.5. Kibovítés

Kibovítjük az ososztályt, de megtartjuk az összes jellegzetességét. Nem hozunk be új metódusokat, hanem a meglévo metódusok funkcionalitásait kibovítjük. Helyettesítheto lesz. Például **Vonat** és **Tehervonat**, olyan vonat, amely árut szállít, vagy a **StringLista** olyan **Lista**, amely stringeket, karakterláncokat tartalmaz.

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$
$$V' ? \{V_1', V_2' ..., V_i'\}, \quad M' ? \{M_1', M_2', ..., M_j'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M',$$

-- a metódusok funkcionalitása kibovül --

$$? + \{|v_1 : V_1, ..., v_i : V_i, m_1 : M_1, ..., m_j : M_j|\} ? *,$$
$$? + \{|v_1' : V'_1, ..., v'_{i'} : V'_{i'}, m'_1 : M'_1, ..., m'_{j'} : M'_{j'}|\} ? *$$
$$? \quad ? + C ? P$$

4.6. Leszukítés

Konstrukció szempontjából egy már meglévo osztály bizonyos funkcióitól eltekintünk, és így új osztály jön létre, nem lesz helyettesítheto. Például, ha a **Repülogép**et úgy definiáljuk, mit egy olyan **Hidroplán**, amely nem tud a vízre szállni. Vagy a **Pingvin** egy olyan **Madár**, amely nem tud repülni.

$$V ? \{V'_1, V'_2 ..., V'_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$
$$V' ? \{V_1', V_2' ..., V_i'\}, \quad M' ? \{M_1', M_2', ..., M_j'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M',$$

-- a metódusok funkcionalitása leszukül --

$$? + \{|v_1 : V_1, ..., v_i : V_i, m_1 : M_1, ..., m_j : M_j|\} ? *,$$
$$? + \{|v_1' : V'_1, ..., v'_{i'} : V'_{i'}, m'_1 : M'_1, ..., m'_{j'} : M'_{j'}|\} ? *$$
$$? \quad ? + C \square P$$

4.7. Egyezés

A hasonló jellegu, hasonló feladatokat megoldó osztályokat egymás alá helyezzük (megfeleloen örököltetünk). Logikailag nem teljesen helyes és nem helyettesítheto. Helyette az a megoldás használható, hogy egy közös, általános osbol származtatjuk le oket. Például, ha a **Teherautót** a **Személygépkocsi**ból származtatjuk, abból a meggondolásból, hogy hasonló jelleguek. Helyette az a megoldás javasolható, hogy hozzunk létre egy közös ost, például **Szárazföldi jármuvek** és mindkettot ebbol származtassuk.

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$
$$V' ? \{V_1', V_2' ..., V_i'\}, \quad M' ? \{M_1', M_2', ..., M_j'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M',$$

-- a változók típusai és a metódusok funkcionalitásai módosulnak --

$$? + \{|v_1 : V_1, ..., v_i : V_i, m_1 : M_1, ..., m_j : M_j|\} ? *,$$

$$? + \{|v_1':V'_1,...,v'_{i'}:V'_{i'},m'_1:M'_1,...,m'_{j'}:M'_{j'}|\}? *$$
$$? \quad ? + C \square P$$

4.8. Kombinálás

Tipikus példája a többszörös öröklodés. Összekombinál két vagy több meglévo osztályt.

$$V ? \{V_1,V_2...,V_i\}, \quad M ? \{M_1,M_2,...,M_j\},$$
$$V'? \{V_1',V_2'...,V_{i'}'\}, \quad M'? \{M_1',M_2',...,M_{j'}'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M', i'? i, j'? j,$$

-- új változók, új metódusok, a változók típusa és a metódusok funkcionalitása módosul, többszörös öröklodés --

$$? + \{|v_1:V_1,...,v_i:V_i,m_1:M_1,...,m_j:M_j|\}? *,$$
$$? + \{|v_1':V'_1,...,v'_{i'}:V'_{i'},m'_1:M'_1,...,m'_{j'}:M'_{j'}|\}? *$$
$$? \quad ? + C ? ? \square P$$

5. Összefoglalás – Öröklodés

A következo táblázat összefoglalja a különbözo öröklodési módokat:

Öröklodési mód	Új adattagok	Új metódusok	Helyettesít-hetoség	Régi adattagok megváltoztatása	Régi metódusok megváltoztatása
Specializálás	*Igen*	*Igen*	*Igen*	*Nem*	*Nem*
Specifikálás	*Nem*	*Nem*	*Igen*	*Igen*	*Nem*
Reprezentáció	*Igen*	*Igen*	*Nem*	*Igen*	*Igen*
Általánosítás	*Igen*	*Igen*	*? (Igen, Nem)*	*Igen*	*Igen*
Kibovítés	*Nem*	*Nem*	*Igen*	*Nem*	*Igen*
Leszukítés	*Nem*	*Nem*	*Nem*	*Nem*	*Igen*
Egyezés	*Nem*	*Nem*	*Nem*	*Igen*	*Igen*
Kombinálás	*Igen*	*Igen*	*? (Igen, Nem)*	*Igen*	*Igen*

6. Az objektumorientált polimorfizmus

Az egybezártság és az öröklodés mellett a polimorfizmus az objektumorientáltság harmadik, és talán legszebb, legtermészetesebb tulajdonsága. A polimorfizmus (többalakúság, alakváltás) azt jelenti, hogy ugyanarra az üzenetre különbözo objektumok különbözoképpen reagálhatnak, minden objektum a saját (az üzenetnek megfelelo) metódusával [6]. A polimorfizmus négyféleképpen nyilvánulhat meg, és minden esetben meg kell adnunk az altípusképzo szabályokat.

6.1. Operátorok felüldefiniálása (overloading)

Ez a típusú polimorfizmus az operátorokra vonatkozik. Hasznos és egyértelmu, hogy különbözo adattípusokra ugyanazt vagy hasonló jellegu muveletet ugyanazzal az operátorral jelöljük. Például a + operátor összeadást jelent egész számok, valós számok esetén is. De ezek alaptípusok. Felvetodhet az a kérdés, hogy ha definiálni akarunk egy **Complex** osztályt, amely a komplex számokat és az ezekkel végezheto muveleteket ábrázolja, tartalmazza, az összeadást végzo metódust miért ne nevezhetnénk át operátorrá, és legyen ennek is a jele a +. Másképp fogalmazva, miért ne bovítenénk ki a + operátor szerepkörét úgy, hogy metódus legyen és a komplex számokkal végzett összeadást is el tudja végezni (vagyis ha van három **a, b, c: Complex** objektumom, akkor a **c = a.add(b);** metódushívást egyszeruen **c = a + b;**-nek tudjuk írni).

Természetesen az operátorok felüldefiniálása nem változtathatja meg a muvelet jellegét: az operandusainak számát, a prioritását, vagy az asszociativitását.

$$V ? \{V_1,V_2...,V_i\}, \quad M ? \{M_1,M_2,...,M_j\},$$
$$V'? \{V_1',V_2'...,V_{i'}'\}, \quad M'? \{M_1',M_2',...,M_{j'}'\},$$
$$V ? V', \quad M ? M', P ? V \square M, \quad C ? V \square M', i'? i, j'? j,$$

-- új változók, új metódusok, a változók típusa és a metódusok funkcionalitása módosul --

$$? + \{|v_1 : V_1,...,v_i : V_i, m_1 : M_1,...,m_j : M_j|\} ? *,$$

$$? + \{|v_1' : V'_1,...,v'_{i'} : V'_{i'}, m'_1 : M'_1,...,m'_{j'} : M'_{j'}|\} ? *$$

$? k ? \{1, ..., j\}: signature(m_k) = signature(m'_k)$

$? operator: operator \sim name(m_k)$

$$? \quad ? + C ? P$$

Ahol *name(m)* az *m* metódus neve, *signature(m)* az *m* metódus aláírása (név és paraméterlista), az *operator* pedig a nyelv egy operátora.

6.2. Polimorfizmus a paraméterátadásban: metódusnevek túlterhelése

Egy osztály több metódusát is nevezhetjük ugyanúgy, ha a paraméterlistája különbözo, vagyis a formális paraméterek száma és/vagy típusa nem egyezik meg. A metódus neve és a metódus paraméterlistája képezi a metódus *aláírását* (signature), és ez az aláírás azonosítja egyértelmuen az illeto metódust. A metódusnevek túlterhelése és az öröklodés számos kérdést vet fel. Az egyszerubb kérdés a túlterhelt metódusok felüldefiniálásának a kérdése. Természetesen, egy metódus csak a vele pontosan megegyezo aláírású metódust definiálhat felül. A bonyolultabb kérdés az azonos nevu metódusok közüli választás pontos algoritmusa, különös tekintettel arra az esetre, mikor a túlterhelt metódusok egymásnak megfelelo paraméterei os-leszármazott viszonyban vannak, így a helyettesíthetoség szabálya életben van, vagy tekintettel azokra az esetekre, mikor alapértelmezett (*default*) paramétereket használunk, és a metódus hívásakor, a ki nem írt paraméter vagy paraméterek miatt a metódus aláírása megegyezik egy másik metódus aláírásával. A szabály ilyenkor az, hogy ha a kód nem egyértelmu, a fordítóprogram hibát jelez, más esetekben elfogadja a kódot.

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$

$$V' ? \{V'_1, V'_2 ..., V'_{i'}\}, \quad M' ? \{M'_1, M'_2, ..., M'_{j'}\},$$

$$V ? V', \quad M ? M', \; P ? V \square M, \quad C ? V \square M', \; i' ? i, j' ? j,$$

-- új változók, új metódusok, a változók típusa és a metódusok funkcionalitása módosul --

$$? + \{|v_1 : V_1,...,v_i : V_i, m_1 : M_1,...,m_j : M_j|\} ? *,$$

$$? + \{|v_1' : V'_1,...,v'_{i'} : V'_{i'}, m'_1 : M'_1,...,m'_{j'} : M'_{j'}|\} ? *$$

$? k ? \{1, ..., j\}: name(m_k) = name(m'_k) ? signature(m_k) ? signature(m'_k)$

$$? \quad ? + C ? P$$

6.3. Absztrakt polimorfizmus (deferring)

A polimorfizmus legalább olyan fontos a program tervezése, mint a kódmegosztás szempontjából. Az os-osztályok definiálnak egy közös interfészt, egy közös metóduskészletet, amelyen keresztül a leszármazottak egységesen kezelhetok. A hierarchia tetején álló ososztályok szerepe inkább az, hogy a leszármazottak interfészének egységességét biztosítsa, nem pedig az, hogy konkrét megoldást adjon valamire. Így ezek az osztályok törzsnélküli, absztrakt metódusokat deklarálnak. Az absztrakt polimorfizmus az olyan metódusokkal foglalkozik, amelyek az osökben csak deklarálva voltak és a konkrét implementációjuk a leszármazottakban történik meg. A másik feladatköre az úgynevezett *sablon* (*template*) *osztályok* vagy *generikus* (*generic*) *osztályok*. Ezek olyan osztályok, amelyek a kódírás pillanatában még ismeretlen típusú adatokkal operálnak, vagy olyan általános osztályok, amelyek különbözo típusú, de hasonló jellegu adatokra tudnak muködési keretet biztosítani. Ennek feltétele, hogy a hívás pillanatában az adat típusát is, mintegy plusz paraméterként megkapja az osztály.

$$V ? \{V_1, V_2 ..., V_i\}, \quad M ? \{M_1, M_2, ..., M_j\},$$

-- csak aláírások, metódus testek nincsenek --

$$V' ? \{V'_1, V'_2 ..., V'_{i'}\}, \quad M' ? \{M'_1, M'_2, ..., M'_{j'}\},$$

$$V ? V', \quad M ? M', \; P ? V \square M, \quad C ? V \square M', \; i' ? i, j' ? j,$$

$$? + \{|v_1 : V_1,...,v_i : V_i, m_1 : M_1,...,m_j : M_j|\} ? *,$$

$$? + \{|v_1' : V'_1,...,v'_{i'} : V'_{i'}, m'_1 : M'_1,...,m'_{j'} : M'_{j'}|\} ? *$$

$? k ? \{1, ..., j\}: signature(m_k) = signature(m'_k)$

$$? \quad ? + C ? P$$

6.4. Metódusok felüldefiniálása (overriding)

A polimorfizmus talán legtöbbször használt formája a metódusok felülírása. Az öröklodés biztosítja azt, hogy a leszármazott osztályok öröklik az ososztály összes metódusát, így használni is tudják azokat. Mi történik azonban akkor, ha a leszármazott osztályban a metódus másképp kell, hogy viselkedjen, vagyis más kóddal kell, hogy rendelkezzen? Ezt a lehetoséget biztosítja a metódusok *felüldefiniálása*. Vagyis az öröklodési hierarchiában különbözo osztályokhoz ugyanolyan névvel definiálhatunk különbözo kódú metódusokat. Ezáltal egy metódusnévhez több kód is tartozhat, attól függoen, hogy hol helyezkedik el a hierarchiában. Természetesen lehetoséget kell biztosítani arra is, hogy a leszármazott osztály metódusából meg tudjuk hívni az ososztály ugyanolyan nevu metódusát, vagyis azt a metódust, amelyet épp most definiálunk felül. Ezt, mint már láttuk, megtehetjük az *Ososztály.Metódus(paraméterlista);* vagy az *inherited* (vagy *super*) *Metódus(paraméterlista);* konstrukciókkal.

Felvetodhet az a kérdés is, hogy osztálymetódusokat felül lehet-e definiálni. A válasz erre a kérdésre: *nem.* Az osztálymetódusok nem példányokon, hanem magán az osztályon fejtik ki hatásukat, így nem *dinamikus* kötést, hanem *statikus* kötést biztosítanak, ami nem biztosít lehetoséget a felüldefiniálásra. Osztálymetódusokat viszont el lehet fedni. Egy osztálymetódus *elfedi* az osökben definiált, vele megegyezo aláírású metódusokat. A felüldefiniálás dinamikus vagy virtuális (futás alatti) kötést vonz, az elfedés pedig statikus kötést. Megkötés, hogy példánymetódusokat osztálymetódusokkal nem lehet elfedni.

$$V\ ?\ \{V_1, V_2 ..., V_i\},\quad M\ ?\ \{M_1, M_2, ..., M_j\},$$

$$V'?\ \{V_1', V_2' ..., V_{i'}'\},\quad M'?\ \{M_1', M_2', ..., M_{j'}'\},$$

$$V\ ?\ V',\quad M\ ?\ M',\ P\ ?\ V \square M,\quad C\ ?\ V \square M',\ i'?\ i, j'?\ j,$$

-- új változók, új metódusok, a változók típusa és a metódusok funkcionalitása módosul --

$$?\ +\ \{|v_1 : V_1, ..., v_i : V_i, m_1 : M_1, ..., m_j : M_j|\}\ ?\ *,$$

$$?\ +\ \{|v_1' : V'_1, ..., v'_{i'} : V'_{i'}, m'_1 : M'_1, ..., m'_{j'} : M'_{j'}|\}\ ?\ *$$

$$?\ k\ ?\ \{1, ..., j\}: signature(m_k) = signature(m'_k)$$

$$?\quad ?\ +\ C\ ?\ P$$

7. Összefoglalás – Polimorfizmus

A következo táblázat összefoglalja a különbözo polimorfizmus módokat:

Polimorfizmus	Azonos név	Azonos test	Azonos aláírás	Absztrakt metódusok	Szintaxis bovítés
overloading	*Igen*	*Nem*	*Igen*	*Nem*	*Igen*
paraméterek	*Igen*	*Nem*	*Nem*	*Nem*	*Nem*
deferring	*Igen*	*Nem*	*Igen*	*Igen*	*Nem*
overriding	*Igen*	*Nem*	*Igen*	*Nem*	*Nem*

Könyvészet

[1] Atsuchi Igarashi, Benjamin C. Pierce, *Foundations for Virtual Types*, ECOOP'99, LNCS 1628, 1999, 161-185.

[2] Martin Abadi, Luca Cardelli, *A Theory of Objects*, Springer-Verlag, 1996.

[3] Luca Cardelli, *Notes about* $F_?^?$, Unpublished manuscript. http://citeseer.nj.nec.com/cs, October 1993.

[4] Kathleen Fisher, John C. Mitchell, *The Development of Type Systems for Object-Oriented Languages*, Stanford University, STAN-CS-TN-96-30.

[5] Benjamin C. Pierce, *Type Systems*, Draft, 2000.

[6] Luca Cardelli, Peter Wagner, *On Understanding Types, Data Abstraction and Polymorphism*, ACM Computing Surveys 17 (4), 1995, 471-522. o.