Typová kontrola a objektově orientovaný jazyk

František Huňka,

Ostravská univerzita, Přírodovědecká fakulta, KIP

Abstrakt

Problematika typové kontroly v programovacích jazycích které, mají třídy s typovými parametry je stále aktuální. Důležitým rysem těchto jazyků je optimální rovnováha mezi typovou kontrolou prováděnou při překladu programu a typovou kontrolou prováděnou za běhu programu. Příspěvek se zabývá rozborem možností jednotlivých druhů kontrol pro jazyk s typovým systémem a kvalifikací odkazů.

Úvod

Typová kontrola se používá ze dvou důvodů:

- jako prostředek pro reprezentování konceptů v aplikačních doménách,
- dále jako prostředek pro detekování určitého druhu chyb během překladu programu.

Síla typového systému je relativní záležitost a můžeme ji chápat jako množství informací doprovázející typ výrazu. Ve slabém typovém jazyku nese typ velmi málo informací o tom, zda-li je zaslaná zpráva legální pro daný objekt. Naopak velmi silný typový jazyk bude mít výrazy, kde typ nese všechny informace o označeném objektu. Jazyky s hierarchickým typovým systémem a kvalifikací odkazů slouží jako kompromis, protože některé, ale ne nutně všechny operace na objektech, se mohou odvozovat z kvalifikace odkazu.

Kontrola během překladu programu je důležitá ze třech důvodů:

- zlepšuje čitelnost programu,
- umožňuje detekovat určitou skupinu chyb během překladu programu,
- umožňuje generovat daleko efektivnější kód.

Programovací jazyk musí poskytovat odpovídající vyváženost mezi flexibilitou a kontrolou během kompilace. Mnoho z takzvaných silně typových jazyků spoléhá na kombinaci typové kontroly během překladu a typové kontroly za běhu programu.

Základní rysy programovacího jazyka BETA

BETA je moderní, objektově orientovaný jazyk patřící do Skandinávské školy OOP. Spustitelný program v BETĚ se skládá z kolekce objektů, které jsou instancemi od vzorů. Pojem vzoru sjednocuje pojmy programovacího jazyka jako: třída, metoda, funkce, korutina, proces a výjimka. To má za následek syntakticky malý jazyk, ale je také zdrojem matení pojmů pro programátory v tradičních jazycích, protože vlastně třídy a metody mají v BETĚ stejnou syntaxi. Obecná deklarace vzoru *V*, kde *V* je jméno má následující tvar:

```
V: Super
 (# Dekl1; Dekl2; .. Dekln  (* část atributů *)
  enter In                  (* vstupní část *)
      do Imp1; Imp2; .. Impm (* výkonná část *)
  exit Out                  (* výstupní část *)
  #)
```

Super je volitelný nadvzor vzoru *V*. V části atributů je seznam deklarací referenčních atributů, částečných objektů a vnořených vzorů. Nejdůležitější tvary deklarací jsou:

- *R1:^Q* kde *Q* je vzor, který deklaruje *R1* jako dynamický odkaz na instanci od vzoru *Q*. *R1* je podobné jako pointer v C++ a může odkazovat na různé objekty v různých časových okamžicích.
- *R2:@Q* deklaruje *R2* jako statický odkaz na instanci od vzoru *Q*. Statický odkaz odkazuje na stále stejný objekt, je tedy jakýmsi konstantním ukazatelem. Takto vytvořený objekt se nazývá statický objekt nebo částečný objekt (part-object), což vyjadřuje, že je součástí objektu, který obsahuje jeho deklaraci.
- *R3:Q(# .. #). R3* deklaruje a pojmenovává vložený vzor s nadvzorem od *Q*. *R3* může být použit jako podtřída od *Q*, nebo vzor metody.
- *R4:< Q*. Deklaruje *R4* jako nový virtuální vzor, který může být specializovaný v dalších podvzorech *V*.
- *R5:##Q*. Deklaruje *R5* jako dynamický odkaz na vzor, kterému je umožněno odkazovat na vzor *Q*, nebo libovolný podvzor od *Q*.

BETA je příkladem silně typového objektově orientovaného jazyka s optimální rovnováhou mezi kontrolou prováděnou při překladu a za běhu programu. Umožňuje také používat hierarchii tříd pro typovou kontrolu. V BETĚ je odkaz na objekt kvalifikován pomocí jména vzoru. Kvalifikace určuje, že odkaz může odkazovat pouze na objekty, které jsou vytvořeny podle daného vzoru, nebo jeho podvzoru.

Kvalifikované reference

Dynamické odkazy v BETĚ jsou kvalifikovány (typovány) na určitý vzor, který určuje množinu objektů, na kterou se lze odkazovat. Z tohoto hlediska mají klasifikační hierarchie značný význam, neboť odkaz kvalifikovaný na nějaký nadvzor může odkazovat na libovolný objekt z množiny instancí všech jeho podvzorů.

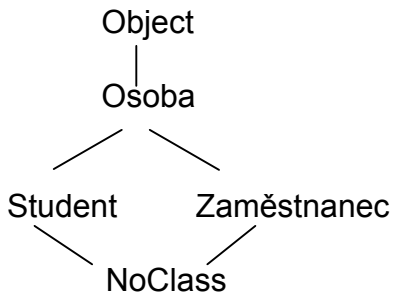
Mějme následující hierarchii tříd (vzorů):

Osoba: (# jméno: @text; věk: @integer #);
Zaměstnanec: Osoba(# plat: @integer #);
Student: Osoba(# ročník: @integer #);

anOsoba: ^Osoba; aZaměstnanec: ^Zaměstnanec; aStudent: ^Student;

anOsoba, aZaměstnanec, aStudent jsou odkazy (reference) na uvedené objekty.
Atributy *jméno*, *věk*, *ročník* a *plat* jsou lokální proměnné.

Reference je kvalifikovaná třídou (vzorem). Kvalifikace omezuje množinu objektů, na které reference smí odkazovat. Reference *anOsoba* je kvalifikována pomocí vzoru *Osoba*. To znamená, že *anOsoba* může odkazovat na instanci třídy *Osoba* a na instance podtříd *Osoba*. Podobně to platí i pro ostatní reference.
Grafické znázornění popsané hiererchie je následující:



Speciální třída *NoClass* je podtřídou všech tříd. Odkaz na neexistující objekt má hodnotu *NONE*.

Abychom mohli popsát typový systém, zavedeme formální, jednoduché označení:

Psub <= Psuper kde *sub* značí podtřídu a *super* nadtřídu.

Další dvě formální funkce jsou:
object() vrací objekt, na který reference aktuálně ukazuje
qual() vrací formální třídu reference, nebo aktuální třídu objektu.

Deklarace *R:^T* zahrnuje, že *qual(R)=T*. To znamená, že *qual(R*), kde *R* je reference, může být zjištěna (vypočtena) v době překladu. Význam třídy *NoClass* je k zajištění, aby *qual(object(R))* bylo definováno pro libovolnou referenci.
Funkce *object(R)* se mění během běhu programu, protože *R* může odkazovat na různé objekty. Význam role této funkce je vyjádření závislosti na chování za běhu programu.

Idea kvalifikované reference začíná s následující relací, která musí být vždy pravdivá pro všechny reference programu.

qual(object(ref)) <= qual(ref)

jako příklad uvažujme

qual(object(anOsoba))<= qual(anOsoba)

Přiřazení reference

Kvalifikované reference poskytují méně flexibility než nekvalifikované. Cenou za flexibilitu je kontrola za běhu programu pro každou poslanou zprávu.
Porovnatelná cena za kvalifikovanou referenci je významně menší. Kontrola za běhu programu se uskuteční pouze v několika případech přiřazení referencí viz následující případy:

{1} new Zaměstnanec[]->aZaměstnanec[] {2} aZaměstnanec[]->anOsoba[]
{3} anOsoba[]->aZaměstnanec[] {4} aStudent[]->aZaměstnanec[]

V {1} je vytvořena nová instance, která je triviálně legální a může být otestována během kompilace.

V {2} je přiřazení legální, protože *anOsoba* může odkazovat na objekty vzoru *Zaměstnanec*. Typová kontrola se provede během překladu. Přípustnost může být staticky kontrolována:

qual(object(aZaměstnanec))<=Zaměstnanec **a současně**

Zaměstnanec<=qual(anOsoba)=Osoba

V {3} je přiřazení legální pouze tehdy, odkazuje-li *anOsoba* na instanci od třídy *Zaměstnanec*. Ve všeobecnosti tato kontrola nemůže být prováděna v době kompilace. To znamená, že v tomto případě bude provedena kontrola za běhu programu. Je vidět, že statické informace

qual(object(anOsoba))<=Osoba
nemůže garantovat, že
qual(object(aZaměstnanec))<=Zaměstnanec
Kompilátor musí provést kontrolu za běhu programu, aby zajistil, že:
qual(object(anOsoba))<=Zaměstnanec

Ve {4} přiřazení je nelegální, protože není možné, aby *aStudent* odkazoval na objekt *aZaměstnanec*.

Většina případů přiřazení je stejná jako ve {2} případu, nebo je kvalifikace rovna. Schopnost zeslabit typovou informaci na objekt jako ve {3} je velmi užitečná, protože umožňuje psát všeobecný kód pro fronty a manipulace se seznamy. Možnost explicitně zesílit typovou informaci je nezbytně nutná. To dává programátorovi možnost, nahlížet na objekty z různých úrovní abstrakce. V uvedeném příkladě by byly možné následující pohledy: prvek ve frontě jako *anOsoba*, *aZaměstnanec* nebo *aStudent*.

Uvažujme příklad se třemi registry: *Osoba-Registr*, *Zaměstnanec-Registr* a *Student-Registr*. Registr *Osoba* může obsahovat odkazy jak na objekty *Zaměstnanec*, tak i na objekty *Student*. Kvalifikace referencí použita na vytvoření instance registr *Osoba* bude kvalifikována vzorem *Osoba* a operace získání referencí na objekty v registru dodá referenci kvalifikovanou vzorem *Osoba*. Podobně registr *Zaměstnanec* bude používat reference kvalifikované vzorem *Zaměstnanec*. Úloha výběru objektů *Zaměstnanec* ze všeobecného registru *Osoba* a jejich vložení do registru *Zaměstnanec* bude zahrnovat nezbytně přiřazení jako v případě {3}.

Přiřazení hodnoty

Jedná se o kopírování příslušných datových atributů. Uvažujme následující přiřazení:

{1} aZaměstnanec->anOsoba
Reference *aZaměstnanec* odkazuje na objekt, který je alespoň *Zaměstnanec* a *anOsoba* odkazuje na objekt, který je alespoň *Osoba*. Pouze atributy *Osoba* jsou zkopírovány z objektu *Zaměstnanec* do objektu Osoba.

{2} anOsoba->aZaměstnanec

Zde je situace opačná. Obsah objektu je kopírován do potenciálně širšího objektu. V době kompilace se ví, že oba objekty mají alespoň shodné atributy *Osoba*. Druhá možnost je vyžadovat, že *anOsoba* označuje objekt kvalifikovaný vzorem *Zaměstnanec*. (Je analogické situaci přiřazení referencí *anOsoba[]->aZaměstnanec[]*). V tomto případě budou kopírovány atributy *Zaměstnanec*.

Třídy s typovými parametry

Virtuální třídy v BETĚ umožňují definovat třídy parametrizované jinými třídami nebo typy. Je to velmi silný jazykový nástroj, ale zároveň také komplikovaný na pravidla kontroly a legality přiřazení. Uvažujme proceduru pro vstup libovolné *anOsoba* do nějakého typu registru. Jako část vstupu *Osoba* se přiřadí *věk* osoby.

```
Insert: proc
        (# par: ^Osoba
        enter par[]
        do novýVěk->par.věk
        #)
```

Pokud je známo, že parametr *par* odkazuje na instanci alespoň třídy *Osoba*, je bezpečné zpřístupnit atribut *věk*. Proto následující vyvolání *Insert* by mělo být legální:

aZaměstnanec[]->Insert

Legalita tohoto výrazu je daná pravidlem pro referenční přiřazení. Uvažujme ale, že *Insert* bude částí všeobecné třídy *Registr*, parametrizované typem *Osoba*, který bude v registru. Záměrem je definovat specializovaný registr, omezený na ukládání objektů třídy *Zaměstnanec* (a jejich podtříd) a podobně specializovat registry pro objekt *Student*. Kvalifikace *Type* parametru *par* procedury *Insert* je deklarován virtuálně. Je

to uděláno proto, aby byla posílena kvalifikace ve specializaci *Registr*. Všeobecný registr je schopen ukládat jakékoli *Osoba* tj. reference na objekty tříd *Zaměstnanec*, *Student* a samozřejmě objekty *Osoba*.

```
Registr: class
        (# Type: virtual class Osoba;
                Insert: virtual proc
                  (# par: ^Type
                    enter par[]
                        do …;
                                novýVěk->par.věk;
                                …; INNER; ..
                        #);
        #);
```

Z této všeobecné třídy *Registr* můžeme deklarovat podtřídu, která bude ukládat pouze *Zaměstnanec*.

```
ZaměstnanecRegistr: class Registr
        (# Type: extended class Zaměstnanec;
          Insert: extended proc
                (#
                  do (par.plat ,par.jméno[])->kontrolaPlatnosti; INNER
                #)
        #);
```

Omezení typu *Type* na *Zaměstnanec* slouží dvěma účelům. Umožňuje nám říct, že parametr k proceduře *Insert* musí být alespoň *Zaměstnanec,* jiné instance nebudou ukládány.
Druhým důvodem je, že uvnitř *ZaměstnanecRegistr* bude možné zpřístupnit atributy třídy *Zaměstnanec*. *Par* kvalifikován jako *Osoba* v *Registru* a jako *Zaměstnanec* v *ZaměstnanecRegistru*. Pro uvedené reference kvalifikované virtuální třídou máme následující relace:

qual(par)=Type<=Osoba

protože *Type* má různé rozšíření v různých podtřídách *Registr*, nemůžeme určit *qual(par)* v době překladu. Kvalifikace *aRegistr.Type* závisí na kvalifikaci *aRegister*. Existují následující tvrzení:

aRegistr.Type=Osoba – if qual(object(aRegistr))=Registr
a
aRegistr.Type=Zaměstnanec –
if qual(object(aRegistr))=ZaměstnanecRegister

Ve všeobecnosti každá podtřída v Registru způsobuje tvrzení tohoto druhu. Použijme označení:

object(aRegistr).Type

k označení virtuální kvalifikace. Kvalifikace parametru *par* také závisí na kvalifikaci *aRegister*. K označení kvalifikace specifického *par* použijeme označení:

qual(object(aRegistr).Insert.par)
Můžeme dedukovat, že
object(aRegistr).Type<=Osoba
a
qual(object(aRegistr).Insert.par)<=Osoba

ale tyto vztahy nemají velké praktické použití, když stanovujeme legalitu přiřazení k *par,* jak uvidíme později.

Pro přiřazení
anOsoba[]->aRegistr.Insert[]
musíme dokázat otestovat, že platí

qual(object(aRegistr.Insert.par))<=qual(object(aRegistr).Insert.par)

Tento vztah určuje, že kvalifikace objektu, na který se odkazuje *par* musí být podtřídou kvalifikace *par* asociovanou s objektem, na který se aktuálně odkazujeme pomocí *aRegistr*. Protože levá strana vztahu je <= *Osoba*, můžeme uzavřít, že požadavek na objekt odkazovaný pomocí *anOsoba* je alespoň *Osoba*.

Dále analyzujeme tři situace přiřazení k virtuálně kvalifikovaným referencím. V prvém případě je přiřazení nezávislé na kontextu. Použijeme následující označení:

anOsoba: ^Osoba; aRegistr: ^Registr;
V {1} uvažujeme přiřazení
anOsoba[]->aRegistr.Insert;

Problém je v tom, že kvalifikace *object(aRegistr)* nemůže být stanovena během kompilace. Jestliže

qual(object(aRegistr))=Registr
potom
aRegistr.Type=Osoba

a přiřazení {1} je legální. Pokud ale na druhé straně

qual(object(aRegistr))=ZaměstnanecRegistr
potom
aRegistr.Type=Zaměstnanec

a přiřazení {1} je legální pouze je-li

qual(object(anOsoba))<=Zaměstnanec
To znamená, že ve všeobecném případě přiřazení k virtuálně kvalifikovaným referencím nemůže být staticky kontrolováno. Pokud je více informací, je v zásadě

možné vypočítat horní hranici potřebné kvalifikace. To je provedeno v následujících dvou případech:

Ve {2} uvažujme příkazy:

new Registr[]->aRegistr[];
new Zaměstnanec[]->anOsoba[];
anOsoba[]->aRegistr.Insert;

zde máme:

qual(object(aRegistr)) = Registr
qual(aRegistr.Insert.Par) = aRegistr.Type = Osoba
qual(anOsoba) = Zaměstnanec

a všechny podmínky budou splněny pokud *Zaměstnanec* < *Osoba.*
Ve {3} uvažujme příkazy:

new ZaměstnanecRegistr[]->aRegistr[];
new Osoba[]->anOsoba[];
anOsoba[]->aRegistr.Insert;

Zde máme:

qual(object(aRegistr))=ZaměstnanecRegistr
qual(aZaměstnanecRegistr.Insert.Par)=ZaměstnanecRegistr.Type=
Zaměstnanec
qual(anOsoba) = Zaměstnanec
Případ {3} je nelegální, protože vede k překročení omezení.

Závěr

Typová kontrola, kterou jsme popsali je všeobecná a vyskytuje se v řadě různých jazykových konstrukcí. Třídy s typovými parametry ukazují na stejný problém a to když je podtřídám dovoleno posílit kvalifikaci typového parametru. Stejného efektu lze dosáhnout s typy odkazujícími se sami na sebe, jako např. *Current* nebo *thisClass*. Avšak jiným příkladem je třída s virtuální procedurou.

Podstata kvalifikovaných odkazů je v tom, že garantují, že reference bude označovat alespoň objekt jisté třídy. To je užitečné, protože je pak s určitostí možné předpokládat, že objekt má atributy jisté třídy. Označování kvalifikace referencí nám může pomoct vypočítat horní hranici těchto požadavků.

Literatura

[1] Madsen, O.L.: Object-Oriented Programming in the BETA programming Language. Addison Wesley 1993
[2] Meyer, B.: Object-oriented Software Construction. Prentice Hall 1988
[3] Abadi, M.: A Theory of Objects. Springer 1996