

Grundlagen des Übersetzerbaus: Verfahren

Marco Ammon (my04mivo)

25. Juni 2020

Inhaltsverzeichnis

1	AST-Transformationen	2
1.1	Innere Klassen	2
1.2	Generics	2
2	Geschachtelte Funktionen	3
2.1	ohne Display	3
2.2	mit Display	3
3	Objekt-orientierte Sprachen	3
3.1	Methodenauswahl	3
3.2	Einfachvererbung	3
3.2.1	Dynamischer Methodenaufruf	3
3.2.2	Casts/Typprüfung zur Laufzeit	3
3.3	Interfaces	3
3.3.1	Dynamischer Methodenaufruf	3
3.3.2	Casts/Typprüfung zur Laufzeit	3
3.4	Mehrfachvererbung	3
3.4.1	Dynamischer Methodenaufruf	3
3.4.2	Casts/Typprüfung zur Laufzeit	3
4	Code-Selektion	3
4.1	Mit Registerzuteilung	3
4.1.1	Naiver Code-Generator	3
4.1.2	getreg	3
4.1.3	Sethi-Ullman-Algorithmus	3
4.2	Ohne Registerzuteilung	3
4.2.1	Baumtransformationen	3
4.2.2	Verfahren von Graham/Glanville	3
4.2.3	Dynamische Programmierung	3
5	Registerzuteilung	3

1 AST-Transformationen

1.1 Innere Klassen

innere Klasse: in `Outer` enthaltene, nicht statische Klasse `Inner`

1. flache Hierarchie durch Verschieben der inneren Klasse außerhalb der umgebenden Klasse(n): `Outer.Inner` \rightarrow `Outer$Inner`
2. Konstruktor der inneren Klasse um Parameter ggf. erzeugen und um Parameter `Outer this$i` ergänzen (mit i als Schachtelungstiefe von `Outer`), zusätzlich gleichnamige Instanzvariable einfügen
3. Zugriffen auf Instanzvariablen von `Outer` ein `this$i.` voranstellen
4. Hilfsmethoden für Zugriff auf private Instanzvariablen von `Outer` in `Outer` einfügen (mit aktueller Java-Version durch spezielles Attribut in Klassendatei nicht mehr notwendig)
5. Alle Auftreten von `Inner` durch `Outer$Inner` ersetzen
6. Bei von `Inner` erbenden Klassen `(new Outer()).super();` im Konstruktor ergänzen, damit `Outer`-Instanz erzeugt wird
7. Bei in Blöcken deklarierten inneren Klassen wird der Zugriff auf finale (oder „effectively-final“) Variablen durch Ergänzen des Konstruktors um diese Variablen ermöglicht

1.2 Generics

1. „Ausradieren“ der Typen („type erasure“):
 - `GenericClass<TypeParameter>` \rightarrow `GenericClass`
 - Typ `A` bleibt gleich
 - Typparameter `A` \rightarrow `Object`
2. Brückenmethoden einfügen, die `Object` zu `A` casten und dann eigentliche Implementierung aufrufen
3. Wenn Typparameter `A` einer Methode nicht aus den Argumenten ableitbar ist, Verwendung des abgeleiteten Typs `*`, der Untertyp aller Typen ist

2 Geschachtelte Funktionen

2.1 ohne Display

2.2 mit Display

3 Objekt-orientierte Sprachen

3.1 Methodenauswahl

3.2 Einfachvererbung

3.2.1 Dynamischer Methodenaufruf

3.2.2 Casts/Typprüfung zur Laufzeit

3.3 Interfaces

3.3.1 Dynamischer Methodenaufruf

3.3.2 Casts/Typprüfung zur Laufzeit

3.4 Mehrfachvererbung

3.4.1 Dynamischer Methodenaufruf

3.4.2 Casts/Typprüfung zur Laufzeit

4 Code-Selektion

4.1 Mit Registerzuteilung

4.1.1 Naiver Code-Generator

4.1.2 getreg

4.1.3 Sethi-Ullman-Algorithmus

4.2 Ohne Registerzuteilung

4.2.1 Baumtransformationen

4.2.2 Verfahren von Graham/Glanville

4.2.3 Dynamische Programmierung

5 Registerzuteilung