

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	Transformation zu Zwischensprache	2
3	Funktionsaufrufe	3
4	Geschachtelte Funktionen	3
5	Objekt-orientierte Sprachen	4
5.1	Methodenauswahl	4
5.2	Einfachvererbung	4
5.2.1	Dynamischer Methodenaufruf	4
5.2.2	Casts/Typprüfung zur Laufzeit	4
5.3	Interfaces	4
5.3.1	Dynamischer Methodenaufruf	4
5.3.2	Casts/Typprüfung zur Laufzeit	4
5.4	Mehrfachvererbung	4
5.4.1	Dynamischer Methodenaufruf	4
5.4.2	Casts/Typprüfung zur Laufzeit	4
6	Code-Selektion	4
6.1	Mit Registerzuteilung	4
6.1.1	Naiver Code-Generator	4
6.1.2	getreg	4
6.1.3	Sethi-Ullman-Algorithmus	4
6.2	Ohne Registerzuteilung	4
6.2.1	Baumtransformationen	4
6.2.2	Verfahren von Graham/Glanville	4
6.2.3	Dynamische Programmierung	4
7	Registerzuteilung	4

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 Transformation zu Zwischensprache

- mehrdimensionale Arrays meistens zu eindimensionalen Array linearisiert
- Operatorenabbildung in „Post-Order“-Reihenfolge
- Kurzschlusssemantik:
 - `code(a && b, Ltrue, Lfalse)` \rightarrow `code(a, L1, Lfalse); L1: code(b, Ltrue, Lfalse)`
 - `code(a || b, Ltrue, Lfalse)` \rightarrow `code(a, Ltrue, L1); L1: code(b, Ltrue, Lfalse)`
 - `code(!a, Ltrue, Lfalse)` \rightarrow `code(a, Lfalse, Ltrue)`
- `code(while e do st od)` \rightarrow `jmp Lcond; Ltrue: code(st); Lcond: code(e, Ltrue, Lfalse); Lfalse:`
- **switch-case:**
 - if-Kaskade

- **lookupswitch**: Tabelle aus (c_i, L_i) -Tupel von Konstante c_i und Sprungziel L_i wird durchsucht
- **tableswitch**: Konstante wird als Index in Tabelle mit Sprungzielen („jump table“) gewählt

3 Funktionsaufrufe

1. Vorbereitung:
 - a) Argumentauswertung gemäß Übergabemechanismus
 - b) Sichern von Caller-Save-Registern auf dem Stack
 - c) Argumente in Registern/auf dem Stack ablegen
 - d) Funktionsaufruf
2. Prolog:
 - a) Sichern des alten FP und Allokation des Stackframes
 - b) Sichern von Callee-Save-Registern im Stackframe
3. Funktionsrumpf
4. Epilog:
 - a) Ablage des Rückgabewerts in Register/auf dem Stack
 - b) Restauration von Callee-Save-Registern
 - c) Freigabe des Stackframes und Restauration des FP
 - d) Rückkehr
5. Nachbereitung:
 - a) Abspeichern des Ergebnis an vorgesehener Stelle
 - b) Entfernen der Argumente vom Stack
 - c) Restauration der Caller-Save-Register

4 Geschachtelte Funktionen

- ohne Display:
 - Aufruf der geschachtelte Funktion mit Zeiger auf Aktivierungsrahmen der umschließenden Funktion (sog. statischer Vorgängerverweis SV)
 - bei Aufruf aus tieferer Schachtelungstiefe SV des Aufrufers ggf. bis zum relevanten Aktivierungsrahmen verfolgen
- mit Display (gesondertes, globales Array) zur Speicherung der SV:
 - Bei Betreten von Funktion der Schachtelungstiefe t , ihren FP an Index t im Display speichern und ggf. bereits bestehenden Wert einer Schwesterfunktion im eigenen Aktivierungsrahmen sichern
 - Durch statisch bekannte Schachtelungstiefe Größe des Displays zur Übersetzungszeit bekannt und Zugriff auf lokale Variablen aus umschließenden Kontext durch Dereferenzieren des SV aus statisch bekannter Position im Display
- Funktionszeiger: auch Argumentwerte müssen mit Zeiger gespeichert werden

5 Objekt-orientierte Sprachen

5.1 Methodenauswahl

5.2 Einfachvererbung

5.2.1 Dynamischer Methodenaufruf

5.2.2 Casts/Typprüfung zur Laufzeit

5.3 Interfaces

5.3.1 Dynamischer Methodenaufruf

5.3.2 Casts/Typprüfung zur Laufzeit

5.4 Mehrfachvererbung

5.4.1 Dynamischer Methodenaufruf

5.4.2 Casts/Typprüfung zur Laufzeit

6 Code-Selektion

6.1 Mit Registerzuteilung

6.1.1 Naiver Code-Generator

6.1.2 getreg

6.1.3 Sethi-Ullman-Algorithmus

6.2 Ohne Registerzuteilung

6.2.1 Baumtransformationen

6.2.2 Verfahren von Graham/Glanville

6.2.3 Dynamische Programmierung

7 Registerzuteilung