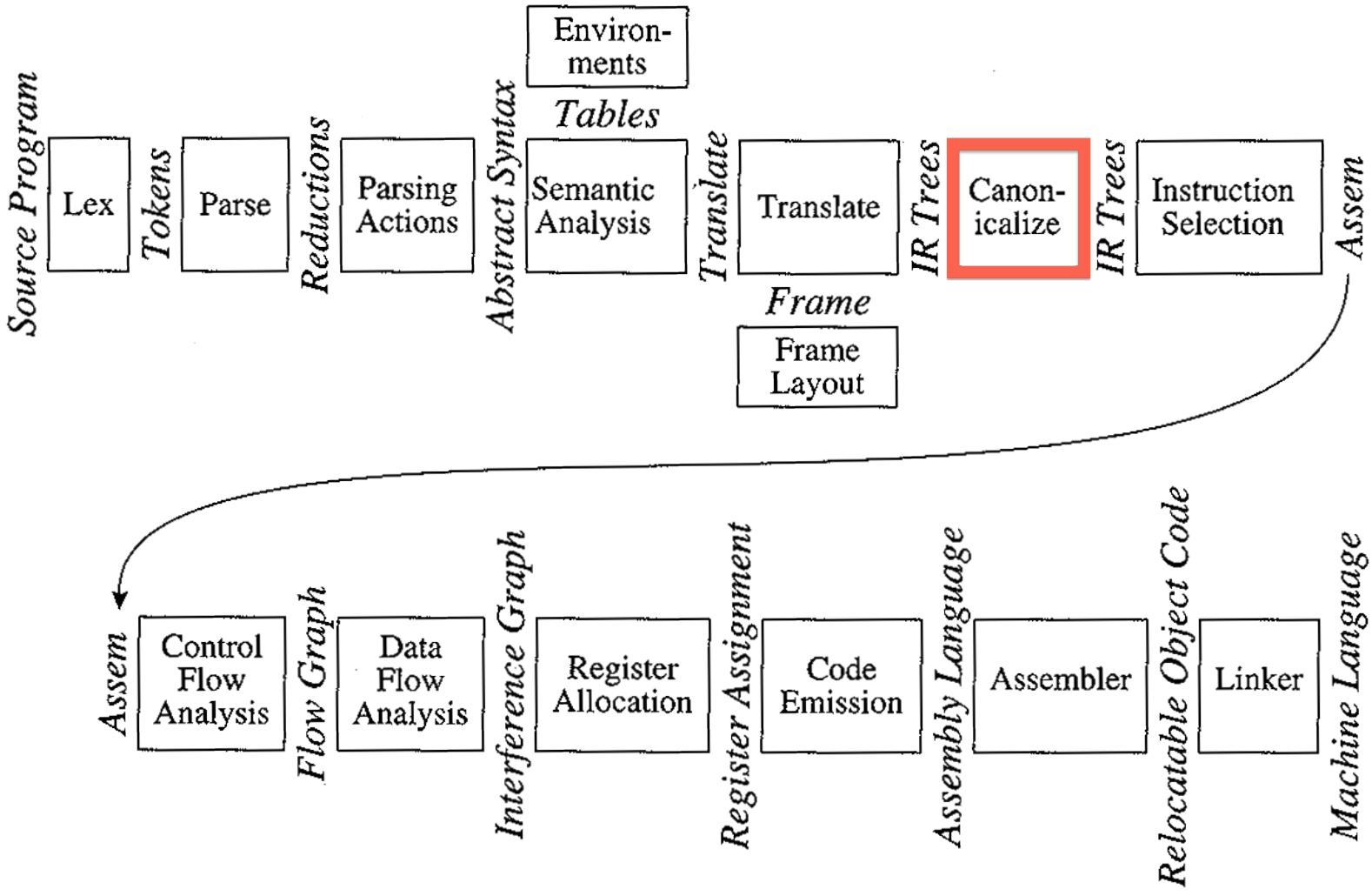
Basisblöcke

Basisblöcke



Basisblöcke

Der erzeugte Zwischencode enthält **verschachtelte Ausdrücke**.
Zur Erzeugung von Maschinencode müssen diese zunächst „linearisiert“ werden.
Linearisierung erfolgt in 3 Schritten:

1. Umwandlung in „kanonischen“ Code, i.e. ohne SEQ oder ESEQ
2. Gruppierung der Instruktionen in **Basisblöcke**
3. Gruppierung der Basisblöcke in „**Traces**“

1. Kanonisierung

Kanonisierung des Zwischencodes:

- Umwandlung von geschachtelten SEQ-Anweisungen in eine Liste von Anweisungen (`TreeStm`-Objekten)
- so dass in allen Anweisungen *s* gilt:
  - jeder Ausdruck *e* in *s* ist ESEQ-frei
  - CALL in *e* nicht überall erlaubt
  - alle Seiteneffekte von *e* sind vor die eigentliche Anweisung *s* geschoben

Ziel:

- Annäherung an „flachen“, unstrukturierten Maschinencode
- alle Ausdrücke *e* in *s* können in beliebiger Reihenfolge ausgewertet werden (Optimierungen)

Kanonische Ausdrücke

Kanonisierung eines Ausdrucks: $e \longrightarrow \mathrm{ESEQ}(s, e')$

- so dass s eine kanonische Anweisung ist und e' seiteneffektfrei
- e' enthält kein ESEQ oder CALL
- jedes CALL in e wird durch $\mathrm{TEMP}(t)$ ersetzt;
  s wird ergänzt um $\mathrm{MOVE}(\mathrm{TEMP}(t), \mathrm{CALL}(\ldots))$
- jedes ESEQ in e wird nach oben geschoben: siehe Buch

Kanonische Anweisungen

Kanonisierung einer Anweisung: $s \longrightarrow s'$

- jeder Ausdruck e in s wird kanonisiert: $\mathrm{ESEQ}(s_c, e_c)$
- e wird durch e_c ersetzt; e_c enthält kein ESEQ oder CALL
- Ausnahme: keine Kanonisierung von CALL in $\mathrm{MOVE}(\mathrm{TEMP}(t), \mathrm{CALL}(\ldots))$, $\mathrm{EXP}(\mathrm{CALL}(\ldots))$
- Seiteneffekte s_c werden vor s geschoben

Verschieben von Seiteneffekten

Beim Kanonisieren werden Seiteneffekte von Ausdrücken „herausgezogen“.

Vorsicht: Diese Seiteneffekte können die Auswertung anderer Ausdrücke beeinflussen. Beispiel:

$$\mathrm{BINOP}(op, e_1, \mathrm{ESEQ}(s, e_2)) \longrightarrow \mathrm{ESEQ}(s, \mathrm{BINOP}(op, e_1, e_2))$$

Diese Umformung ist nur möglich, wenn s und e_1 kommutieren, d.h. s nicht die Auswertung von e_1 beeinflusst. Andernfalls muss die Auswertung von e_1 zwischengespeichert werden:

$$\mathrm{ESEQ}(\mathrm{SEQ}(\mathrm{MOVE}(\mathrm{TEMP}(t), e_1), s),$$
$$\mathrm{BINOP}(op, \mathrm{TEMP}(t), e_2)$$

Auflösen von SEQ und ESEQ

- Eine kanonische Anweisung enthält SEQs wenn überhaupt nur ganz oben im Baum. Sie kann also auch als Liste von `TreeStm`-Objekten dargestellt werden.
- Ein kanonischer Ausdruck enthält maximal ein ESEQ ganz oben. Er kann also als Paar aus einer `TreeStm`-Liste und eine `TreeExp`-Objekt dargestellt werden.
- Dies sind auch die Rückgabetypen der Visitors `CanonStm` und `CanonExp`.

2. Basisblöcke

Zur Gruppierung des Codes muss der Kontrollfluss eines Programms berechnet werden, d.h. zu jeder Instruktion muss die Menge der möglichen Nachfolgerinstruktionen berechnet werden.

Nur JUMP und CJUMP sind dafür interessant.

Instruktionen mit linearem Kontrollfluss werden in einen **Basisblock** zusammengefasst, d.h. jeder Basisblock hat folgende Struktur

- Er beginnt mit einem LABEL;
- die letzte Instruktion ist ein JUMP oder CJUMP;
- ansonsten enthält er keine JUMP oder CJUMP Instruktionen.

2. Basisblöcke

Taktik: Linearer Durchlauf über die Liste der Instruktionen, wobei ein LABEL einen neuen Basisblock beginnt und ein JUMP oder CJUMP einen Basisblock beendet.

Falls...

- einem Label kein Sprung vorhergeht, wird ein Sprung auf dieses Label eingefügt
- einem Sprung kein Label folgt, wurde „dead code“ gefunden
- der erste Block nicht mit Label anfängt, wird einer erfunden
- der letzte Block nicht mit einem Sprung endet, wird ein Sprung auf ein End-Label eingefügt, welches später am Ende angehängt wird

3. Tracing

Da jeder Basisblock mit einem Sprung endet, können die Basisblöcke in beliebiger Reihenfolge erzeugt werden. Insbesondere wählen wir eine Reihenfolge, sodass

- der else-Zweig immer unmittelbar der entsprechenden CJUMP Instruktion folgt („fall-through“);
  - wenn nur noch der then-Zweig verfügbar ist, kann die CJUMP-Bedingung negiert und die Labels vertauscht werden
  - wenn beide Zweige nicht mehr verfügbar sind, muss ein Dummy-else-Zweig mit einem Sprung auf den echten else-Zweig eingefügt werden
- wenn möglich einem JUMP der entsprechende Basisblock folgt;

Taktik: Linearer Durchlauf über die Basisblöcke.
Abschließend können bei JUMP *l*; LABEL *l* Sequenzen die Sprünge eliminiert werden, und nicht angesprungene Labels entfernt werden.

Ihre heutige Aufgabe

Schreiben Sie einen Tracer für den MiniJava-Zwischencode.

- Auf der Vorlesungshomepage finden Sie die Klassen für den Kanonisierer.
- Schreiben Sie eine Methode, die eine Liste von `TreeStm`-Statements in Basisblöcke aufteilt.
- Schreiben Sie eine Methode, die eine Liste von Basisblöcken optimiert anordnet.
- Sie können die Ausgabe wieder mit dem C-Code-Übersetzer von letzter Woche testen.