# Kapitel 18

# Palsberg-Schwartzbach Typinferenz

- Ziel: Optimierung des Methodenaufrufs (insbesondere dynamische Bindung)
- Technik: Berechnung der möglichen Typen (Klassen) jedes Ausdrucks
- Idee: berechne *Mengen* möglicher Typen
- falls Menge leer: Typfehler
- falls Menge einelementig: keine dynamische Bindung
- *Typinferenz*: Herleitung der (kleinsten !) Typmengen aus den Variablen*verwendungen*
- Typinferenz kann für typfreie Sprachen (Smalltalk) auch die notwendigen Laufzeittests bestimmen

Schreibweise:

$$[\![expr]\!] = \{\tau_1, \ldots, \tau_n\}$$

Typconstraints:

$$\begin{aligned} \{\tau_1, \ldots, \tau_n\} &\subseteq X && \text{(explizites Constraint)} \\ X &\subseteq Y && \text{(Propagationsconstraint)} \\ \tau \in X &\Rightarrow Y \subseteq Z && \text{(Bedingtes Constraint)} \\ X &\subseteq \{\tau_1, \ldots, \tau_n\} && \text{(Sicherheitsconstraint)} \\ X &= Y && \text{(Mengengleichheit)} \end{aligned}$$

weitere Schreibweise:

$$\downarrow C = \{\tau \mid \tau \leq C\}$$

Dies nennt man ein *Ideal* in der Klassenhalbordnung

## 18.1 Elementare Regeln

Wir betrachten zunächst nur Kernsprache ohne Überladungen, Konversionen

- $k \in Const$:

$$[\![k]\!] = \{\text{int}\} =\downarrow \text{int}$$

$$[\![\text{true}]\!] = \{\text{bool}\}$$

  ähnlich für andere Konstanten

- `null, this, new`:

$$[\![\text{null}]\!] = \emptyset =\downarrow \text{void}$$

$$[\![\text{this}_C]\!] = [\![\text{new } C]\!] = \{C\}$$

- $expr_1 + expr_2$:

$$[\![expr_1]\!] = [\![expr_2]\!] = [\![expr_1 + expr_2]\!] = \{\text{int}\}$$

  ähnlich für andere arithmetische/logische Operatoren

- Zuweisung $Id = expr$: Typ-Konformanz!

$$[\![expr]\!] \subseteq [\![Id]\!]$$

- if $(expr_1)$ $expr_2$ else $expr_3$:

$$[\![expr_1]\!] = \{\text{bool}\}, \quad [\![\text{if...}]\!] = [\![expr_2]\!] \cup [\![expr_3]\!]$$

  ähnlich für andere Anweisungen

- Deklaration $C\ Id$:

$$[\![Id]\!] =\downarrow C$$

- Methodenaufruf $expr.Id(expr_1, \ldots, expr_n)$:

$$[\![expr]\!] \subseteq \{C \mid C \text{ implementiert } Id\}$$

Sei ferner die Implementierung
$\text{type } Id \ (\text{type}_1 \ Id_1, \ldots, \text{type}_n \ Id_n) \ \{...\text{return } expr_0...\}$ in Klasse $C$ vorhanden. Falls $C \in [\![expr]\!]$ muß gelten:

$$[\![expr_i]\!] \subseteq [\![Id_i]\!], \quad i = 1..n$$

$$[\![expr.Id(expr_1, \ldots, expr_n)]\!] \supseteq\downarrow type \supseteq [\![expr_0]\!]$$

da es mehrere Implementierungen der Methode geben kann, handelt es sich tatsächlich um bedingte Constraints. Vollständige Form:

$$C \in [\![expr]\!] \Rightarrow$$
$$\Big([\![expr_i]\!] \subseteq [\![Id_i]\!] \ \wedge \ [\![expr.Id(...)]\!] \supseteq [\![expr_0]\!]\Big)$$

Beispiel:

```
class C {
  int n(int i) {
    return i+1
  }
}
C x = new C();
int k = x.n(true);
```

$$[\![i]\!] = [\![1]\!] = [\![i+1]\!] = \{\text{int}\}, \quad [\![\text{true}]\!] = \{\text{bool}\}$$
$$[\![x]\!] = [\![\text{new } C]\!] = \{C\}, \quad [\![k]\!] = \{\text{int}\} \supseteq [\![x.n(\text{true})]\!]$$

Für den Methodenaufruf muß gelten:

$$[\![x]\!] \subseteq \{C\}, \quad [\![\text{true}]\!] \subseteq [\![i]\!], \quad [\![x.n(\text{true})]\!] \supseteq \{\text{int}\} \supseteq [\![i+1]\!]$$

Daraus folgt

$$\{\text{bool}\} \subseteq \{\text{int}\}$$

was nicht gilt: TYPFEHLER

## 18.2 Typinferenz

- beliebige Typmengen statt Ideale
  Typen in Deklarationen werden ignoriert
- $[\![e]\!]$ beschreibt *minimale Anforderungen* an die Typen von $e$, berechnet aus den *Verwendungen* von Variablen

  $\emptyset$ = „keine Anforderungen“
- Korrektheitskriterium: falls $e$ zur Laufzeit Typ $\tau$ haben kann, so $\tau \in [\![e]\!]$
- $\Rightarrow$ möglichst kleine Typmengen
  $\Rightarrow \supseteq$ statt $=$ in den Constraints!
- Aus den Constraints wird durch *Fixpunktiteration* der minimale Fixpunkt berechnet
- es gibt immer trivialen Fixpunkt $[\![x]\!] =\downarrow Object$ für alle Variablen/Ausdrücke $x$
- Fixpunktiteration startet für alle $x$ mit der leeren Menge und vergrößert diese dann durch Anwenden von Constraints
- effiziente Implementierung für volle Sprache
  nichttrivial!

Beispiel 1:

```
class C {
   ? n(? i) {
    return i+1
  }
}
? x = new C();
? k = x.n(true);
```

$$[\![i]\!] \supseteq \{\text{int}\},\ [\![1]\!] \supseteq \{\text{int}\},\ [\![i+1]\!] \supseteq \{\text{int}\},$$
$$[\![\text{true}]\!] \supseteq \{\text{bool}\},\ [\![x]\!] \supseteq \{C\},\ [\![\text{new } C]\!] \supseteq \{C\},$$
$$k \supseteq [\![x.n(\text{true})]\!] \supseteq [\![\text{return } i+1]\!] \supseteq \{\text{int}\}$$

Für den Methodenaufruf muß wie gehabt gelten:

$$[\![x]\!] \subseteq \{C\},\ \ [\![\text{true}]\!] \subseteq [\![i]\!],\ \ [\![x.n(\text{true})]\!] \supseteq [\![i+1]\!]$$

demnach $[\![i]\!] \supseteq \{\text{int}\}$ und $[\![i]\!] \supseteq \{\text{bool}\}$

Die minimale Lösung ist:

$$[\![i]\!] = \{\text{int}, \text{bool}\}\ \ (!),\ [\![1]\!] = \{\text{int}\},$$
$$[\![i+1]\!] = \{\text{int}\},\ [\![\text{true}]\!] = \{\text{bool}\}$$
$$[\![x]\!] = \{C\},\ [\![\text{new } C]\!] = \{C\},$$
$$[\![k]\!] = \{\text{int}\},\ [\![x.n(\text{true})]\!] = \{\text{int}\}$$

Mithin hat $i$ keinen eindeutigen Typ; eine Deklaration für $i$ läst sich nicht angeben.

Beispiel 2:

```
class A {
  ? x; ? y;
  ? m(? f) {
    return f.m(x):
  }
}
class B {
  ? m(? g) {
    return this;
  }
}
? u = (new A).m(new B).m(null);
```

**1.** Constraints für $f.m(x)$:
Wg. Bedingung $[\![f]\!] \subseteq \{A, B\} = \{c \mid c \text{ hat } m\}$ je ein bedingtes Constraint für jede mögliche Bindung von $m$, nämlich $A :: m$ und $B :: m$

a) eventuell ist $m = A :: m$:

$$A \in [\![f]\!] \Rightarrow [\![x]\!] \subseteq [\![f]\!] \text{ und } [\![f.m(x)]\!] \supseteq [\![f.m(x)]\!]$$

b) oder es ist $m = B :: m$:

$$B \in [\![f]\!] \Rightarrow [\![x]\!] \subseteq [\![g]\!] \text{ und } [\![f.m(x)]\!] \supseteq [\![\text{this}_B]\!]$$

**2.** Constraints für this, new $A$, new $B$:

$$[\![\text{this}_B]\!] \supseteq \{B\},\ [\![\text{new } A]\!] \supseteq \{A\},\ [\![\text{new } B]\!] \supseteq \{B\}$$

**3.** Constraints für $(\text{new } A).m(\text{new } B)$: wiederum 2 Fälle

**a)** $A \in [\![\text{new } A]\!] \Rightarrow [\![\text{new } B]\!] \subseteq [\![f]\!]$ und
$[\![(\text{new } A).m(\text{new } B)]\!] \supseteq [\![f.m(x)]\!]$

**b)** $B \in [\![\text{new } A]\!] \Rightarrow [\![\text{new } B]\!] \subseteq [\![g]\!]$ und
$[\![(\text{new } A).m(\text{new } B)]\!] \supseteq [\![\text{this}_B]\!]$

**4.** Constraints für $(\text{new } A).m(\text{new } B).m(\text{null})$:

**a)** $A \in [\![(\text{new } A).m(\text{new } B)]\!] \Rightarrow [\![\text{null}]\!] \subseteq [\![f]\!]$ und
$[\![(\text{new } A).m(\text{new } B).m(\text{null})]\!] \supseteq [\![f.m(x)]\!]$

**b)** $B \in [\![(\text{new } A).m(\text{new } B)]\!] \Rightarrow [\![\text{null}]\!] \subseteq [\![g]\!]$ und
$[\![(\text{new } A).m(\text{new } B).m(\text{null})]\!] \supseteq [\![\text{this}_B]\!]$

**5.** Constraint für die Zuweisung:

$$[\![u]\!] \supseteq [\![(\text{new } A).m(\text{new } B).m(\text{null})]\!]$$

Wir haben also

$$\{B\} \subseteq [\![\text{new } B]\!] \subseteq [\![f]\!]$$

$$\{B\} \subseteq [\![\text{this}_B]\!] \subseteq [\![f.m(x)]\!] \subseteq [\![(\text{new } A).m(\text{new } B)]\!]$$

$$\{B\} \subseteq [\![(\text{new } A).m(\text{new } B).m(\text{null})]\!]$$

Andere Anforderungen gibt es nicht; insbesondere wird
$A \in [\![f]\!], [\![f.m(x)]\!], [\![(\text{new } A).m(\text{new } B)]\!],$
$[\![(\text{new } A).m(\text{new } B).m(\text{null})]\!]$ nicht erzwungen!

Die minimale Lösung ist deshalb:

$$[\![x]\!] = [\![y]\!] = [\![g]\!] = [\![\text{null}]\!] = \emptyset; \quad [\![\text{new } A]\!] = \{A\}$$

$$[\![f]\!] = [\![f.m(x)]\!] = [\![\text{this}_B]\!] = [\![\text{new } B]\!] = \{B\}$$

$$[\![(\text{new } A).m(\text{new } B)]\!] = \{B\}$$

$$= [\![(\text{new } A).m(\text{new } B).m(\text{null})]\!]$$

Damit ist die Mehrdeutigkeit der $m$-Aufrufe aufgelöst:
$(\text{new } A).m(\text{new } B)$ ruft mit Sicherheit $A :: m$;
$(\text{new } A).m(\text{new } B).m(\text{null})$ ruft mit Sicherheit $B :: m$

Ferner ist offenbar geworden, daß $x$, $y$ beliebigen Typ haben können (minimale Anforderungen = leer)

## 18.3 Lösen von Mengenungleichungssystemen

P/S-Typinferenz führt (wie etliche andere Analysen) letztendlich auf Mengenungleichungssystem:

$$\begin{aligned} M_1 &\subseteq N_1 \\ M_2 &\subseteq N_2 \\ &\vdots \\ M_k &\subseteq N_k \end{aligned}$$

Dabei kann durchaus $M_i = N_j$ sein (zB $M_i = N_j = [\![\text{this}_B]\!]$)

Wie löst man das System effektiv?

1. Wähle effiziente Mengenimplementierung. In P/S: numeriere (endliche Zahl von) Klassen durch ⇒ Bitvektoren!
2. Initialisiere alle unbekannten $M_i, N_i = \emptyset$ (falls zB $M_i = \{x_1, \ldots, x_k\}$ natürlich nicht!)
3. Stelle *Abhängigkeitsgraph* auf:
   Knoten = $M_i, N_i$, Kante $M_i \rightarrow N_i$ für jede Ungleichung $M_i \subseteq N_i$
4. Berechne die $M_i, N_i$ in *topologischer Reihenfolge*: Berechen $N_i$ erst, wenn *alle* Vorgänger im Graphen berechnet wurden (→ topologische Sortierung)
5. Für Knoten $X$ mit Vorgängern $Y^1, \ldots, Y^j$ berechne $X := X \cup Y^1 \cup Y^2 \cup \ldots \cup Y^j$. Spezialfall: Knoten ohne Vorgänger sind unveränderlich.

6. falls Graph zyklenfrei: Berechnung fertig nach $2k$ Schritten
7. falls bedingte Ungleichungen $\tau_i \in U_i \Rightarrow M_i \subseteq N_i$: sobald $\tau_i \in U_i$, „feuert“ $M_i \subseteq N_i$
8. falls Zyklen: propagiere nur aus Zyklus heraus, wenn Zyklus stabil geworden ist!
9. Zyklen erfordern globale Iteration, bis die $M_i, N_i$ stabil sind $\Rightarrow$ typische Gesamtkomplexität $O(k^3)$
10. Verbesserung: ersetze Zyklus $X^1 \subseteq X^2 \subseteq \ldots \subseteq X^j \subseteq X_1$ durch einen Knoten $X = X^1 \cup X^2 \cup \ldots \cup X^j$, sobald Zyklus entdeckt wird. vermerke $X^i = X$ für alle $i$
11. weitere Optimierungen sind Gegenstand neuerer Forschung