1 Kontextanalyse

Ziel der Kontextanalyse

Überprüfung der **kontextuellen Einschränkungen**:
- Geltungsbereiche von Bezeichnern
- Typregeln (Typüberprüfung)
- Konsistenzregeln (z. B. Funktionsaufrufe, Parameteranzahl, etc.)

1.1 Geltungsbereiche und Symboltabellen

1.1.1 Grundlagen

- Jede Deklaration **bindet** einen Namen.
- Jede Verwendung eines Bezeichners muss einer gültigen **Bindung** zugeordnet sein.
- Fehlende Bindung ⇒ **Fehler**.

Symboltabelle

Datenstruktur zur **Zuordnung von Namen zu Attributen**.
- Schnelles Nachschlagen (z. B. Hash-Tabelle)
- Enthält Art, Typ, Sichtbarkeit, evtl. Adresse
- Hierarchische Struktur für verschachtelte Blöcke

1.1.2 Blockstrukturen

- **Monolithisch:** Alle Deklarationen global (BASIC, COBOL)
- **Flach:** Global + lokal (FORTRAN)
- **Verschachtelt:** Beliebige Tiefe (Pascal, Ada, Java)

Geltungsbereichsregeln

- Kein Bezeichner mehrfach innerhalb eines Blocks.
- Verwendung nur mit Deklaration im lokalen oder umschließenden Block.
- Lokale Deklarationen überdecken äußere.

1.2 Attribute

Attribute

Informationen, die zu einem Bezeichner gespeichert werden:
- Art (Konstante, Variable, Funktion)
- Typ (int, char, boolean, array, record, . . .)
- Sichtbarkeit, ggf. Speicheradresse

Verwendung:

- Überprüfung von Geltungsbereichs- und Typregeln
- Vorbereitung auf Code-Generierung

Speicherung:

- Einfach: explizite Speicherung in Klassen
- Besser: Referenz auf **AST-Knoten** der Deklaration

1.3 Identifikation

Identifikation

Erste Phase der Kontextanalyse. Ordnet jede **Verwendung** einer **Definition** zu.

Ziel: Aufbau einer Symboltabelle durch AST-Durchlauf.

Kombination mit: Typprüfung (zweite Phase).

1.4 Typprüfung

Typ

Einschränkung der möglichen Interpretationen eines Speicherbereiches oder Ausdrucks.

- **Statische Typisierung:** zur Compile-Zeit (z. B. Pascal, C)
- **Dynamische Typisierung:** zur Laufzeit (z. B. Python)

1.4.1 Typüberprüfung – Prinzip

1. Bestimme Typen von Teilausdrücken (*Bottom-Up* im AST)
2. Prüfe, ob Typanforderungen des Kontextes erfüllt sind

Typregeln:

- `if E then ...` $\rightarrow E$ muss vom Typ `Boolean` sein.
- Zuweisung $x := E \rightarrow$ Typ von x = Typ von E.
- Binäre Operatoren: E_1 `op` E_2 typkorrekt, wenn $E_1 : T_1$, $E_2 : T_2$ und $op : T_1 \times T_2 \rightarrow R$.

1.5 Implementierung der Kontextanalyse

AST-Dekoration

Dekorierter AST speichert zusätzliche Informationen:
- Typ jedes Ausdrucks
- Verweis auf bindende Deklaration

Vorgehensweisen:

- OO-Ansatz: Jede AST-Klasse implementiert `check()`.
- Funktionaler Ansatz: Separate Typprüfungsfunktion.
- **Visitor-Pattern:** Modularer Ansatz zur Trennung von Struktur und Verhalten.

Visitor-Pattern

Ermöglicht neue Operationen auf AST-Knoten, ohne deren Klassen zu verändern.

Beispiele:

- `visitAssignCommand`: prüft Typkompatibilität von LHS und RHS.
- `visitLetCommand`: öffnet/schließt Scope in Symboltabelle.
- `visitIfCommand`: prüft Boolean-Bedingung.

1.6 Standardumgebung

Standardumgebung

Vordefinierte Typen, Konstanten und Operationen, die zur Analyse vorhanden sein müssen.

Beispiele:

- Typen: `Integer`, `Boolean`, `Char`
- Konstanten: `true`, `false`
- Operatoren: `+`, `-`, `*`, `<`, `=`

Implementierung:

- Als AST-Einträge vor der Analyse eingefügt.
- Liegen auf äußerster Scope-Ebene (Ebene 0/1).

1.7 Typäquivalenz

Typäquivalenz

Regel, wann zwei Typen als "gleich" gelten.

1.7.1 Ansätze

- **Strukturelle Äquivalenz:** gleiche Struktur ⇒ äquivalent.
- **Namenäquivalenz:** nur identische Typdefinitionen sind äquivalent.

In Triangle: strukturelle Typäquivalenz.

Beispiele:

- `record n: Integer; c: Char end` ≠ `record c: Char; n: Integer end`
- `array 8 of Char` = `array 8 of Char`

1.7.2 Komplexe Typen

- Verweis auf Typbeschreibung im AST (`TypeDenoter`)
- Struktureller Vergleich von Sub-ASTs zur Typprüfung

1.8 Algorithmus der Kontextanalyse

Algorithmus

Ein kombinierter AST-Durchlauf:
1. Tiefensuche (DFS) über AST
2. Identifikation (Symboltabelle aufbauen)
3. Typprüfung (Dekoration des ASTs)

Voraussetzung: Bindungen erscheinen im Code **vor** Verwendungen.

Ergebnis: Dekorierter AST mit allen Typ- und Scopeinformationen.

1.9 Zusammenfassung

- Kontextanalyse = Identifikation + Typprüfung
- Symboltabelle verwaltet Geltungsbereiche
- Typprüfung sichert korrekte Operationen
- Visitor-Pattern modularisiert Implementierung
- Standardumgebung liefert primitive Typen
- Triangle nutzt strukturelle Typäquivalenz

Prüfungsrelevante Kernbegriffe

Symboltabelle, Geltungsbereich, Attribut, Typüberprüfung, AST-Dekoration, Visitor, Standardumgebung, Typäquivalenz