5. Semantische Analyse und Transformation

Eingabe: Strukturbaum

Aufgaben:	**Übersetzermodul:**
Namensanalyse	Umgebungsmodul
Eigenschaften von Programmobjekten	Definitionsmodul
Typanalyse, Operatoridentifikation	Signaturmodul
Transformation	Baumgenerator

Ausgabe: **Zielbaum, Zwischen-Code**
Zielprogramm bei Source-to-Source

Standardimplementierungen und Generatoren für Übersetzermodule

Operationen der Übersetzermodule werden an Knoten des Strukturbaums aufgerufen

Modell: abhängige Berechnungen im Baum

Spezifikation: **attributierte Grammatiken**

generiert: Baumdurchlauf mit Aufrufen der Operationen

Definitionsmodul

zentrale Datenstruktur

ordnet Programmobjekten z*. B. Typen, Variablen, ...*
Eigenschaften zu
z. B. T*yp einer Variablen, Elementtyp eines Array-Typs.*

Objekte werden durch *Schlüssel (key)* identifiziert.

Operationen:

NewKey ( )	liefert neuen Schlüssel
ResetP (k, v)	setzt zum Schlüssel k die Eigenschaft P mit Wert v
SetP (k, v, d)	setzt (ersetzt) zum Schlüssel k die Eigenschaft P mit Wert v (bzw. d)
GetP (k, d)	liefert Wert der Eigenschaft P des Schlüssels k; liefert d, falls P zu k nicht gesetzt ist

Aufrufe: abhängige Operationen im Baum

Implementierung: z. B. Liste von Eigenschaften zu jedem Schlüssel

Generierung des Definitionsmoduls:
Aus Spezifikationen

Eigenschaftsname: Eigenschaftstyp;

werden Funktionen Reset, Set, Get generiert.

Namensanalyse (Bezeichneridentifikation)

Bezeichner benennen (statische) Programmobjekte.

Definition für Bezeichner b führt neues Programmobjekt ein und bindet es an den Bezeichner.

Die Bindung gilt in einem bestimmten Bereich des Programms: **Gültigkeitsbereich** der Definition.

Aufgabe: jedem Auftreten eines Bezeichners den Schlüssel des Programmobjektes zuordnen. *(konsistente Umbenennung)*

Grundlage: Gültigkeitsregeln der Sprache *(scope rules)*

Verdeckungsregeln für Sprachen mit geschachtelten Strukturen:

- **Algol-Regel:** Die Definition eines Bezeichners b gilt im **ganzen** kleinsten sie umfassenden Abschnitt, aber nicht in darin enthaltenen Abschnitten mit einer Definition von b.

- **C-Regel:** Die Definition eines Bezeichners b gilt im kleinsten sie umfassenden Abschnitt **von der Definitionsstelle an**, aber nicht in darin enthaltenen Abschnitten mit einer Definition von b von der Definitionsstelle an**.**

Abschnitt je nach Sprache: Block. Prozedur, Modul, ...

Implementierung: Operationen des **Umgebungsmoduls** in Baumkontexten aufrufen.

Umgebungsmodul

implementiert den abstrakten Datentyp **Environment**:

hierarchisch geschachtelte Mengen von Bindungen
(Bezeichner, Schlüssel)

Funktionen:

NewEnv () erzeugt ein neues Environment e als Wurzel.

NewScope (e1) erzeugt neues Environment e2.
Alle Bindungen aus e1 sind auch in e2, falls sie nicht durch Bindungen in e2 verdeckt werden.

DefineIdn (e, b) fügt Bindung (b, k) in e ein,
falls sie dort noch nicht exisitiert;
k ist dann ein neuer Objektschlüssel.
Liefert in jedem Fall den an b gebundenen Schlüssel k.

KeyInEnv (e, b) liefert den Schlüssel k eines Paares (b, k) aus e (einschließlich aller e_i, aus denen e erzeugt wurde).
Liefert NoKey, falls kein solches Paar existiert.

KeyInScope (e, b) liefert den Schlüssel eines Paares (b, k), falls unmittelbar in e enthalten, NoKey sonst.

Umgebungsoperationen im Baum

Operationen in Baumkontexten und Reihenfolge ihrer Ausführung modelliert Gültigkeitsregeln.

Wurzelkontext `Root`:

```
Root.Env = NewEnv ();
```

Abschnitte `Range`, die Definitionen enthalten können:

```
Range.Env =
  NewScope(INCLUDING (Range.Env, Root.Env));
           nächst umgebende Range oder Root
```

definierendes Auftreten eines Bezeichners `IdDefScope`:

```
IdDefScope.Key =
  DefineIdn(INCLUDING Range.Env,
            IdDefScope.Symb);
```

angewandtes Auftreten eines Bezeichners `IdUseEnv`:

```
IdUseEnv.Key =
  KeyInEnv (INCLUDING Range.Env,
            IdUseEnv.Symb);
```

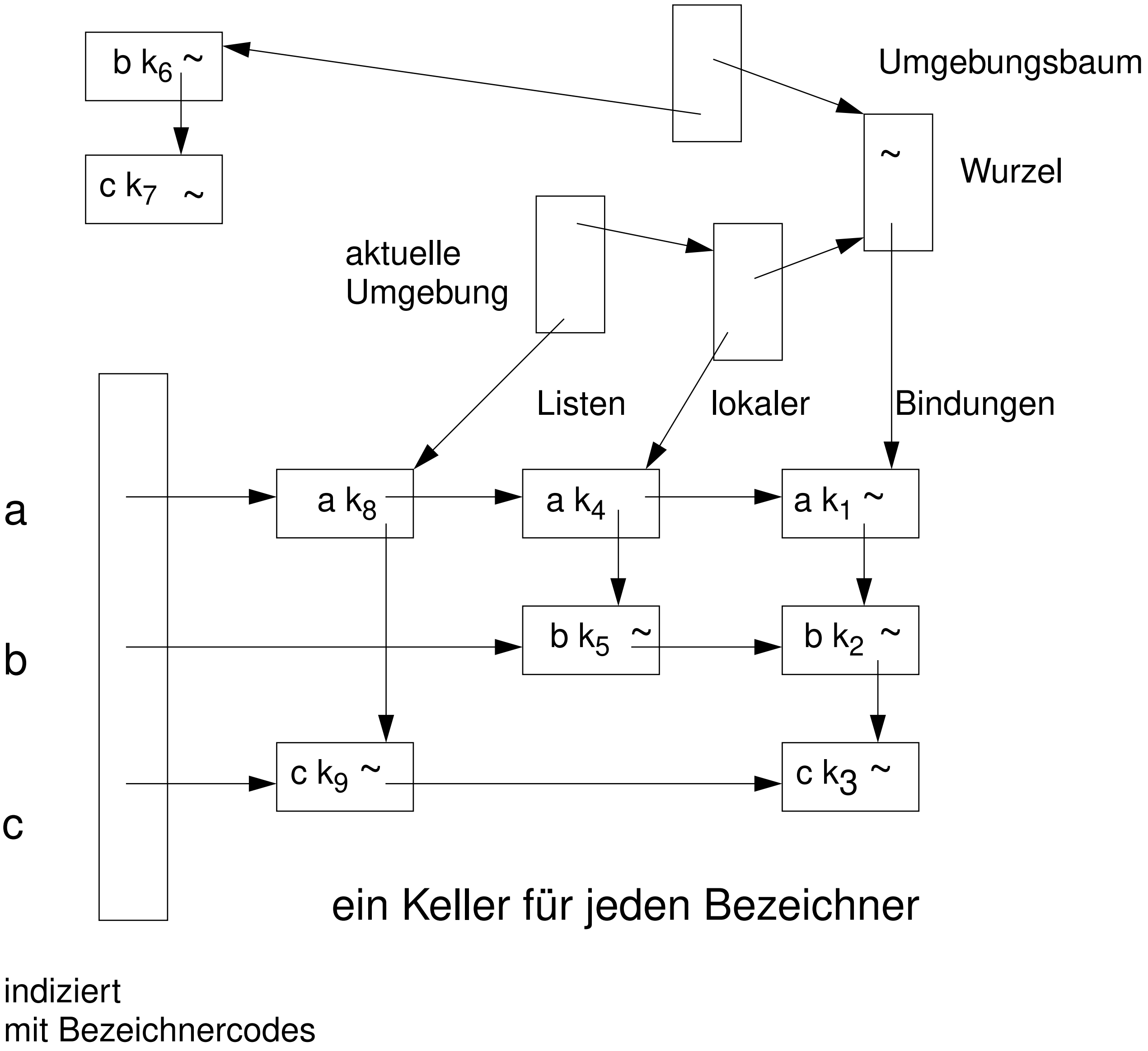
Vorbedingungen je nach Gültigkeitsregeln:
Algol-Regel: Vorbedingung für `KeyInEnv`:
alle `DefineIdn` aller umgebenden Ranges sind ausgeführt.

C-Regel: `KeyInEnv` und `DefineIdn` werden in der Reihenfolge des Programmtextes ausgeführt (links-abwärts).

weitere Prüfungen an dem Ergebnis der Bez.identifikation:

- keine Anwendung ohne Definition
- keine 2 Definitionen für einen Bezeichner in einem Range
- keine Anwendung vor der Definition (Pascal)
- Vorwärtsdefinitionen
- ...

Umgebungsmodul: Datenstruktur



k_i: Schlüssel des definierten Objektes

Operationen:

enter (e) lokale Definitionen von e kellern

leave (e) lokale Definitionen von e entkellern

KeyInEnv hat <u>kostanten</u> Aufwand!

Typanalyse: Aufgaben

Bestimmung und Prüfung von Typen zur Übersetzungszeit

- **Definierte Objekte** (z. B. Variable)
  haben Typ als Eigenschaft (im Definitionsmodul speichern)

- **Programmkonstrukte** (z. B. Ausdruck, Indizierung)
  haben Typ als Attribut des Baumknotens
  Spezielle Aufgabe: überladene Operatoren identifizieren

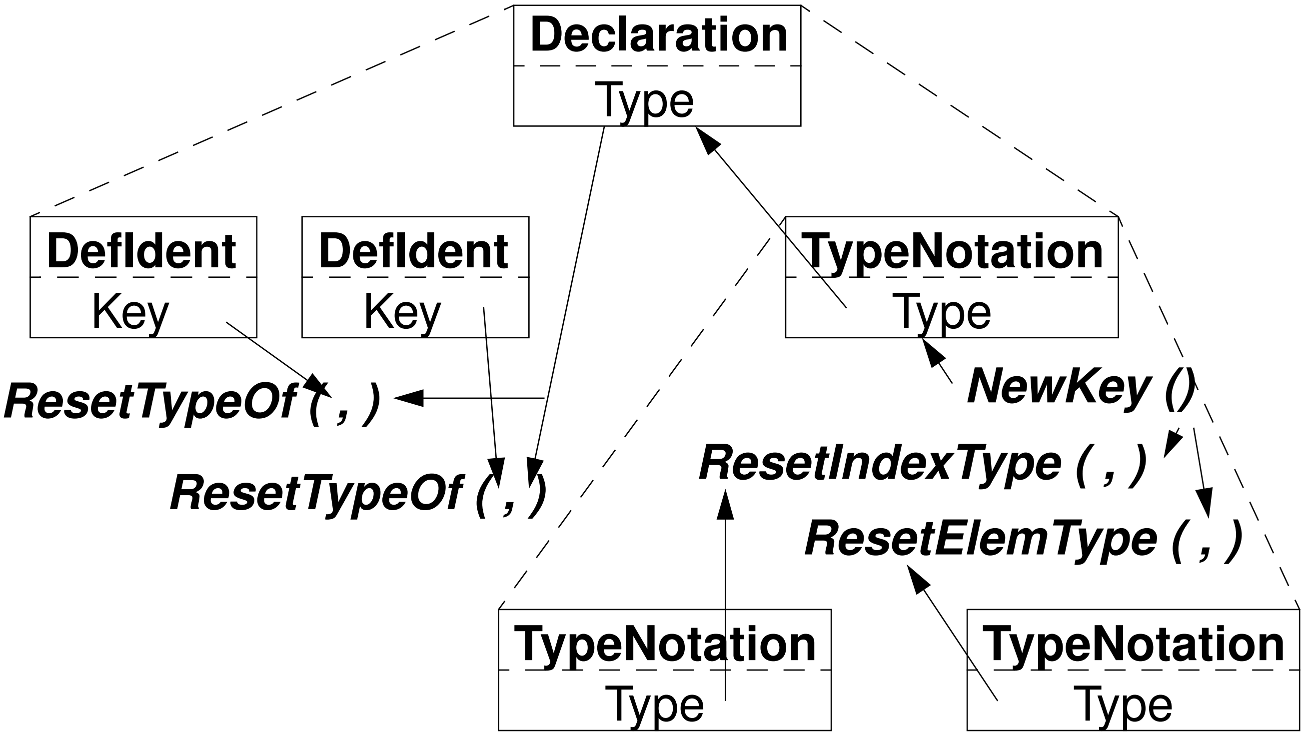
- **Typen selbst sind Programmobjekte**
  repräsentiert durch key
  benannt: in Typdefinitionen
  unbenannt: in komplexen Typangaben

- **Typen haben Eigenschaften**
  z. B. Elementtyp eines Array-Typs
  bestimmt in Typangaben

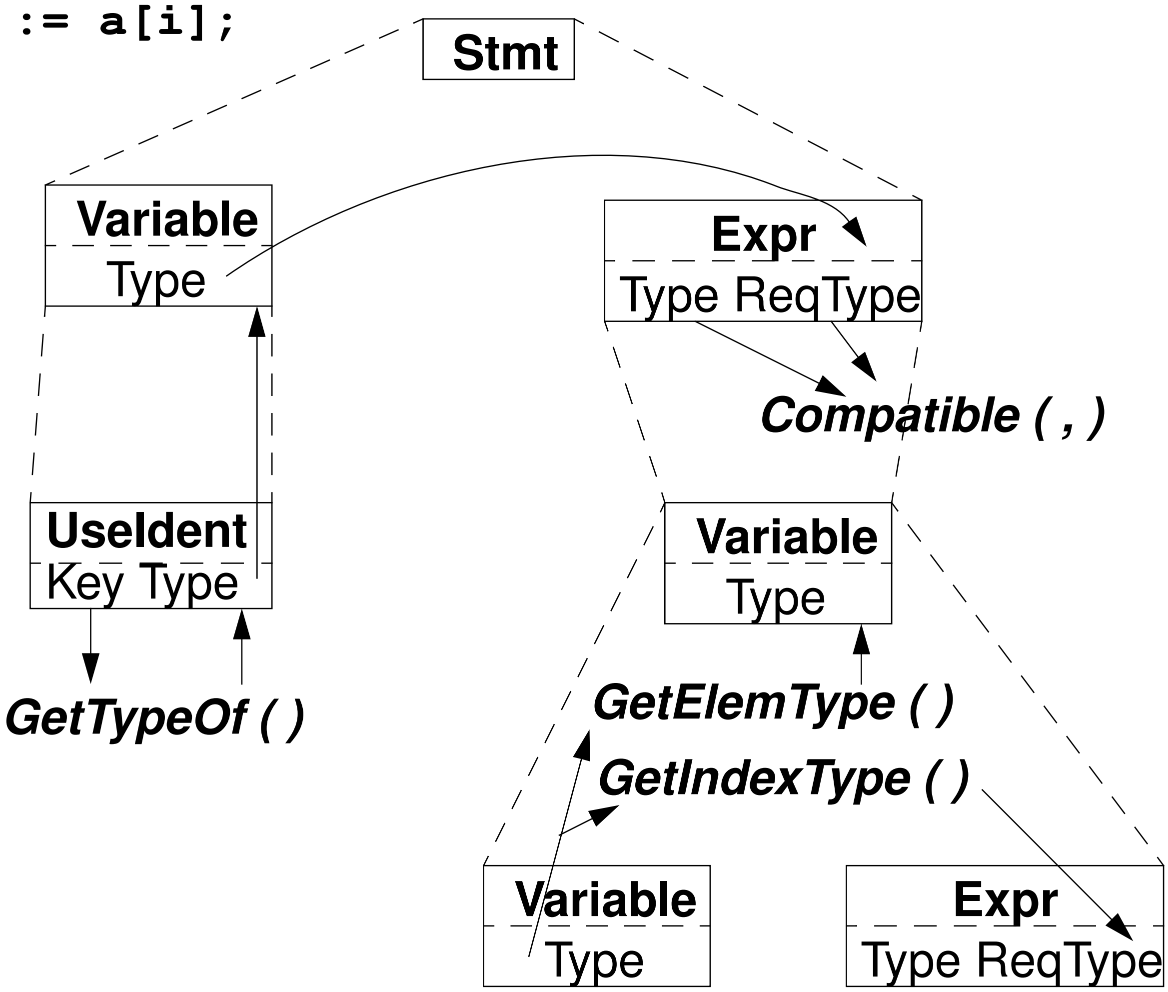
- **Typprüfung für Programmobjekte und Konstrukte:**
  Typ muß/darf nicht bestimmte Eigenschaften haben
  Vergleich erwarteter - tatsächlicher Typ
  (gleich, verträglich, ...)

Typanalyse: Operationen im Baum

```
a, b: array [1..10] of real;
```




```
x := a[i];
```



Identifikation überladener Operatoren

Gleiches Operatorsymbol (Quelloperator) für Zieloperatoren mit verschiedenen Signaturen und Bedeutungen

Tabelle aus Sprachdefinition:

Zeichen	Signatur	Bedeutung
+	int ✕ int -> int	Ganzzahladdition
+	real ✕ real -> real	Gleitpunktaddition
+	set ✕ set -> set	Mengenvereinigung
/	real ✕ real -> real	Gleitpunktdivision
=	t ✕ t -> boolean	Vergleich

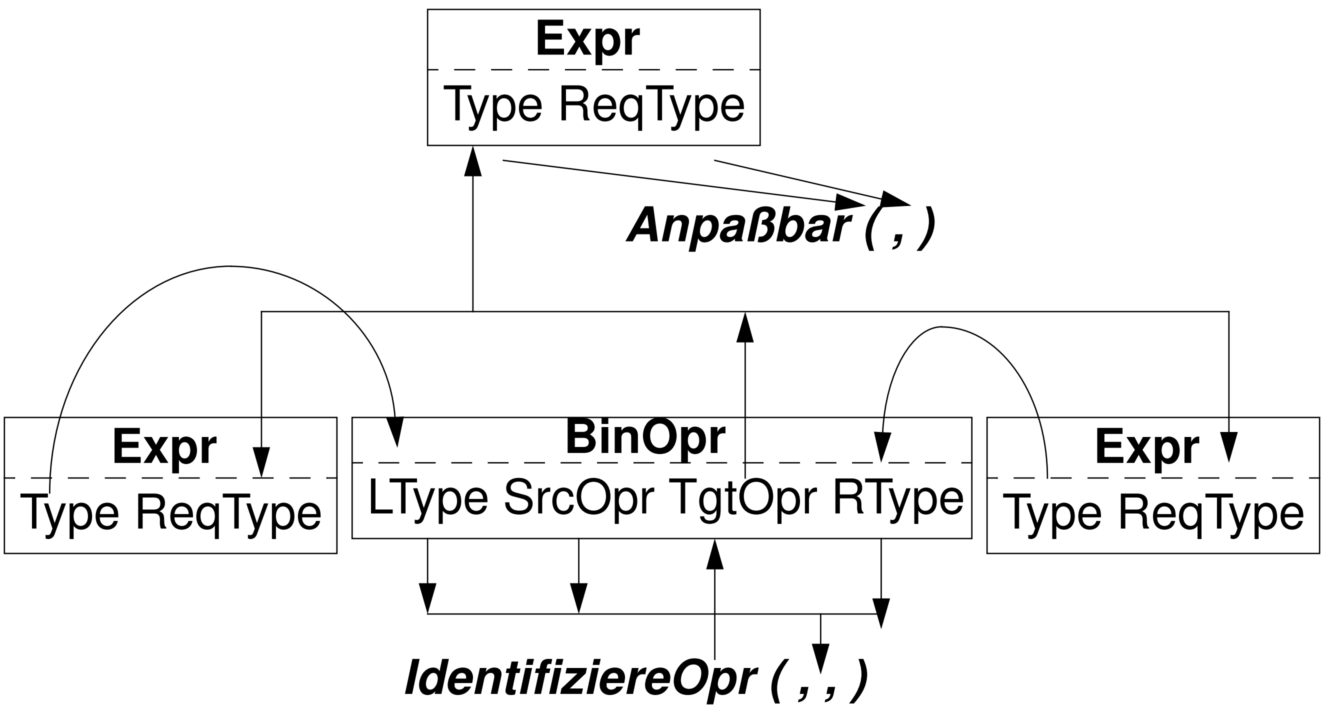
Coercion: implizit anwendbare Typanpassung

z. B. int -> real, char -> string, ...

Operatoridentifikation:

geg.: Quelloperator und Operandentypen
ges.: Zieloperator mit passender Signatur
bei minimalen Anpassungen

Operationen im Strukturbaum:



Behandlung semantischer Fehler

Grundsätze für den Entwurf:

Quellbezogene Meldungen:

- **explizite oder implizite Forderung der Sprachdefinition**
  ein oder mehrere Prüfungen als Operationen im Baum
- **Prüfungen im kleinsten betroffenen Kontext**
  liefert präzise Quellposition für Fehlermeldung
  ggf. nötige Information dorthin transportieren
- **aussagekräftiger Meldungstext**
- **unterschiedliche Meldungen zu verschiedenen Symptomen**

Trotz Fehler werden alle Operationen im Baum ausgeführt:

- **Fehlerwerte einführen,** z. B. `NoKey, NoType, NoOpr`
- **Operationen, die Werte liefern, tun das auch im Fehlerfall**
- **Jede Operation akzeptiert auch Fehlerwerte als Parameter**
- **Meldungen zu Folgefehlern vermeiden**
  z. B. jeder Typ ist mit `NoType` verträglich

Transformation

Programm der Zwischensprache als **Zielbaum** (Wald) erzeugen.

Zwischensprache vorgegeben oder gemäß Operationen der abstrakten Quellmaschine **entwerfen**.

Entwurfsgrundsätze:

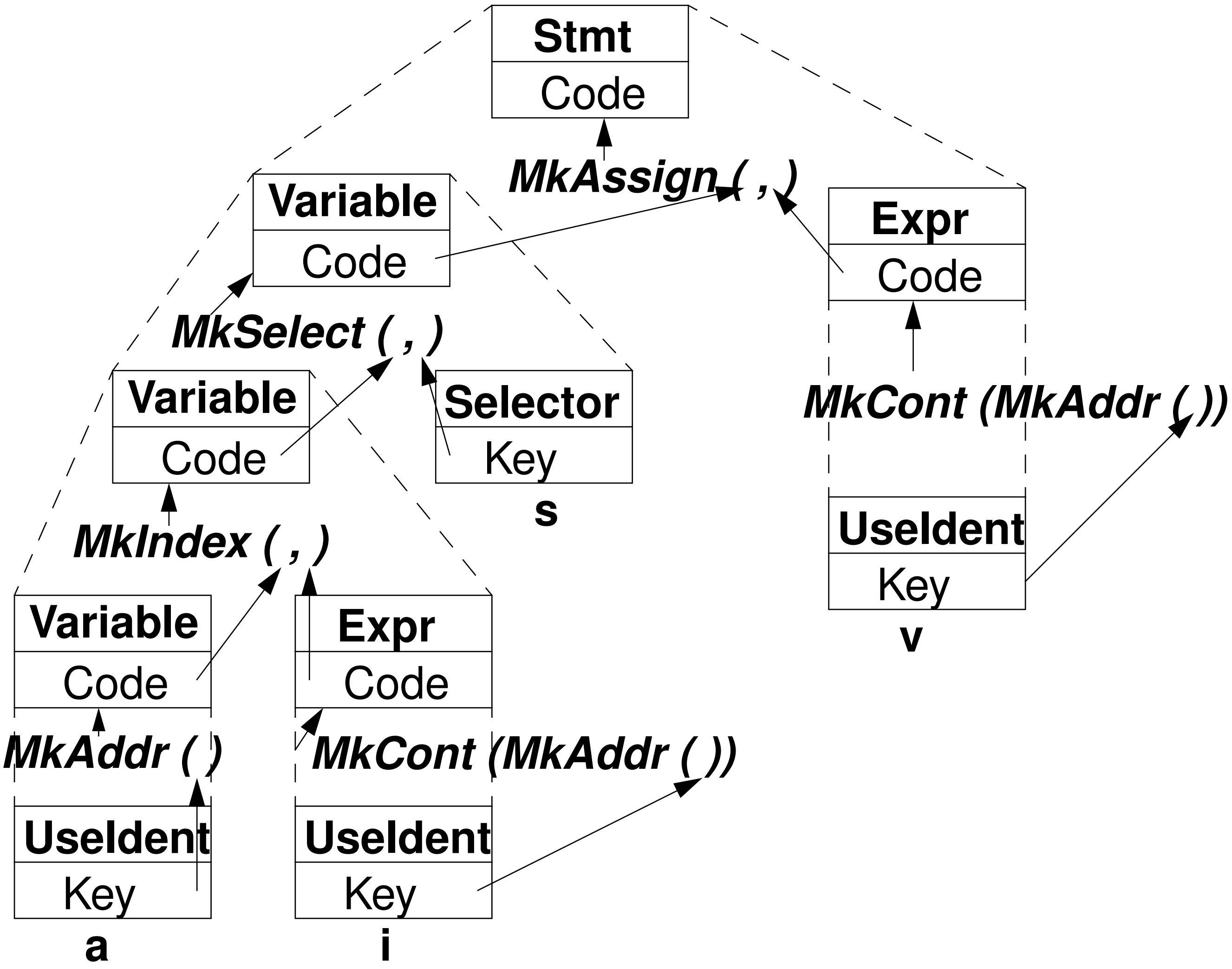
- **Struktur vereinfachen:**
  nur Strukturen und Eigenschaften, die für Synthese nötig,
  Deklarationen, Typangaben weglassen
  (sind im Definitionsmodul)
- **Strukturen vereinheitlichen**
  z. B. Schleifen
- **Zieloperatoren für überladene Operatoren**
- **implizite Operatoren einsetzen**
  Typanpassung, Laufzeitprüfungen

Zwischensprachbaum und Definitionsmodul an Synthese übergeben (Datenstruktur, Aufruffolge oder Datei)

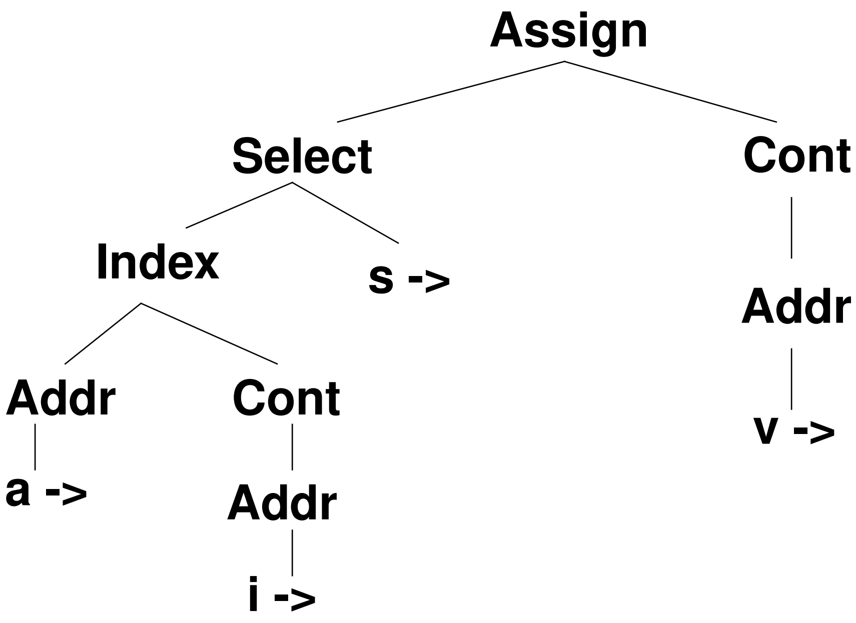
Bei **Source-to-Source-Übersetzung** repräsentiert der Zielbaum das **Zielprogramm**.
Text aus Baum erzeugen durch **rekursive Anwendung von Textmustern**.

Beispiel: Zielbaum erzeugen

Strukturbaum für `a[i].s := v;`



Zielbaum:



Definitionsmodul:

a -> ...
i -> ...
s -> ...
v -> ...