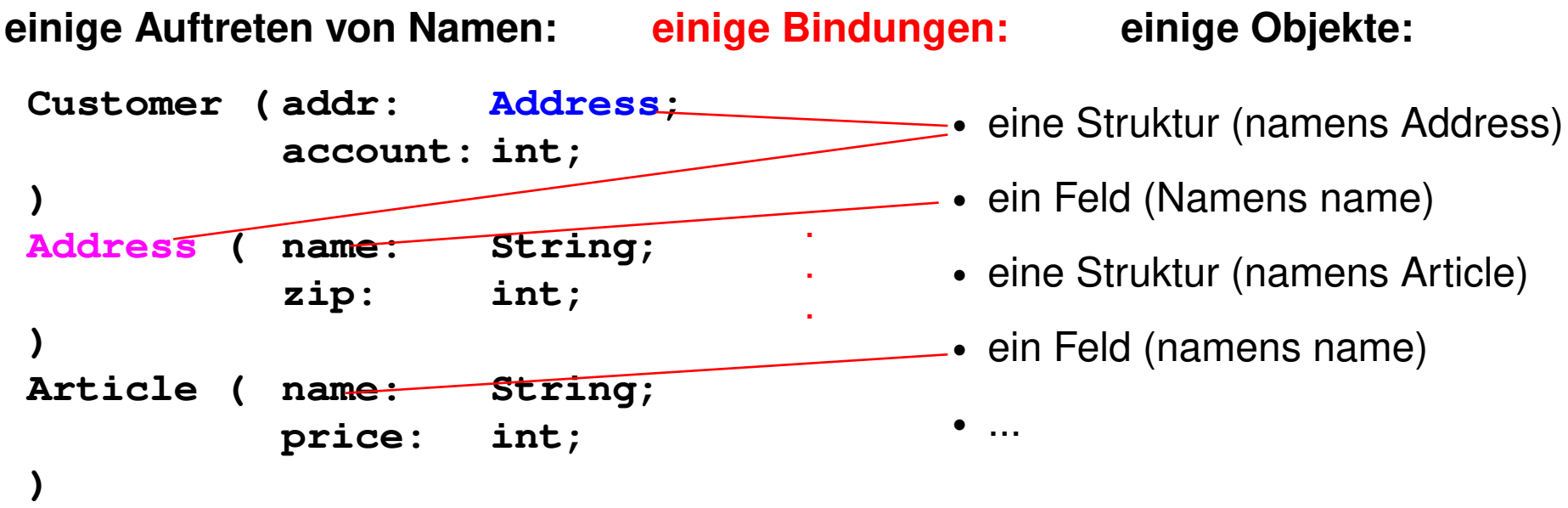
5. Namen und Eigenschaften

Namen in der Beschreibung repräsentieren **Objekte** als Teil der Bedeutung des Textes

Auftreten von Namen sind an **Objekte** gebunden.

Die Bindung wird durch **Gültigkeitsregeln** der Beschreibungssprache geregelt. Z. B:

- Struktur-Namen werden in der ganzen Beschreibung identisch gebunden.
- Feld-Namen werden für jede Struktur separat gebunden.
- Typ-Namen sind angewandte Auftreten von Namen.
  Sie erfordern, dass es irgendwo ein definierendes Auftreten desselben Names gibt.



© 2002 bei Prof. Dr. Uwe Kastens

Schlüssel und Eigenschaften

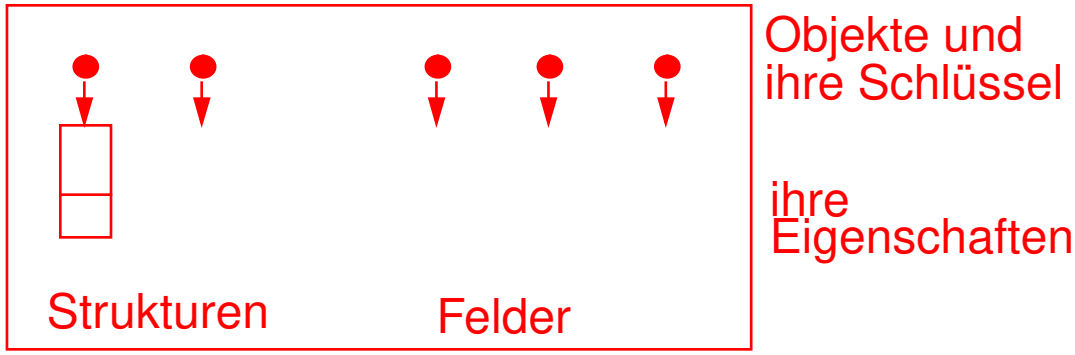
attributierter Strukturbaum

Umgebungen

Eli Werkzeuge implementieren Eigenschaften von Objekten und Umgebungen

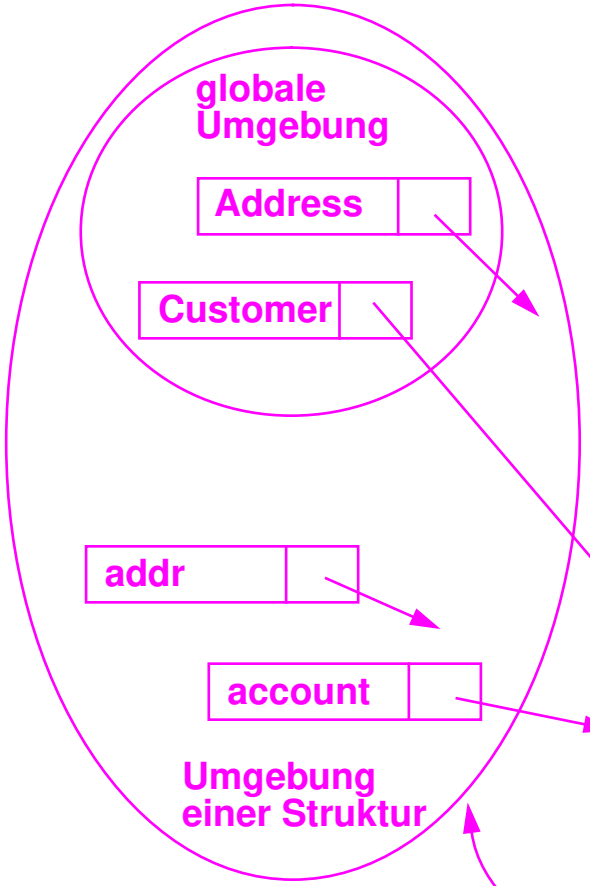
Olbjekte werden durch **Schlüssel** repräsentiert.
Eigenschaften werden ihnen zugeordnet.

Strukturen haben eine Eigenschaft „Umgebung“



© 2002 bei Prof. Dr. Uwe Kastens

Bindungen und Umgebungen



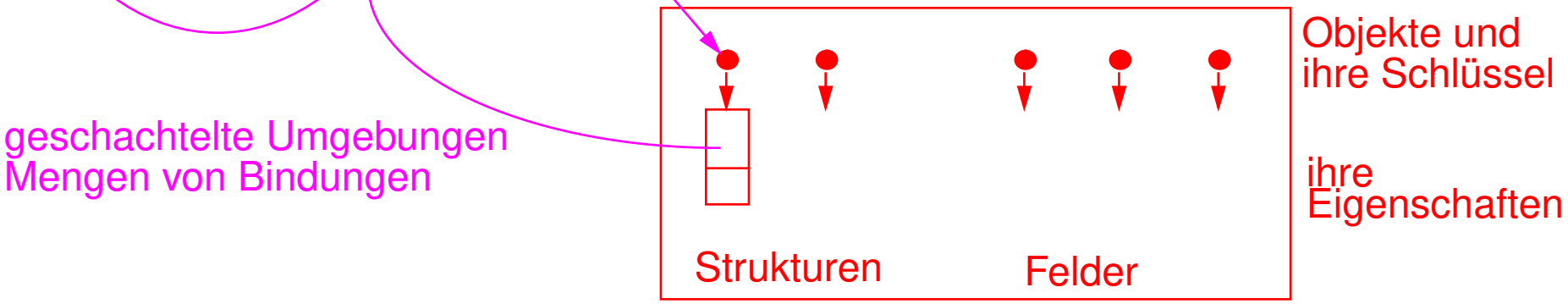
Umgebung: geschachtelte Mengen von Bindungen

Bindung: verknüpft Name mit Schlüssel

Die globale Umgebung bindet alle Struktur-Namen.

Umgebung einer Struktur bindet ihre Feld-Namen.

Eli Werkzeuge implementieren Eigenschaften von Objekten und Umgebungen

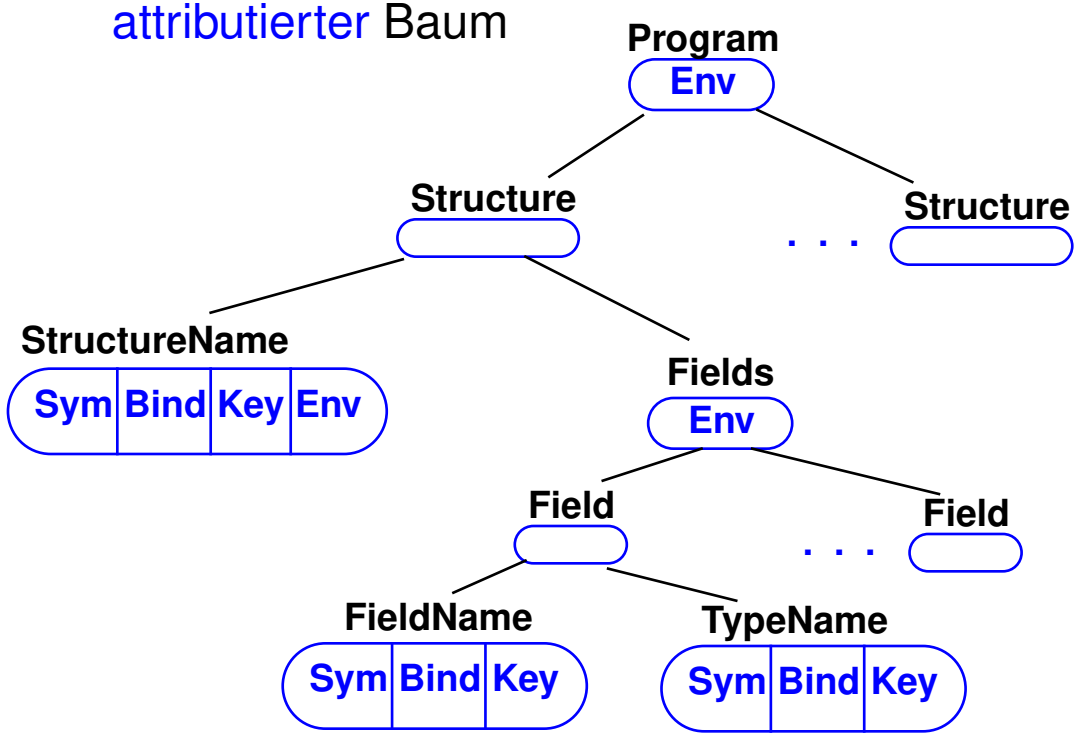


© 2002 bei Prof. Dr. Uwe Kastens

Attributierter Baum für Namensanalyse

Attribute der Baumknoten
beschreiben Eigenschaften des repräsentierten Konstruktes

attributierter Baum



Program hat die globale Umgebung

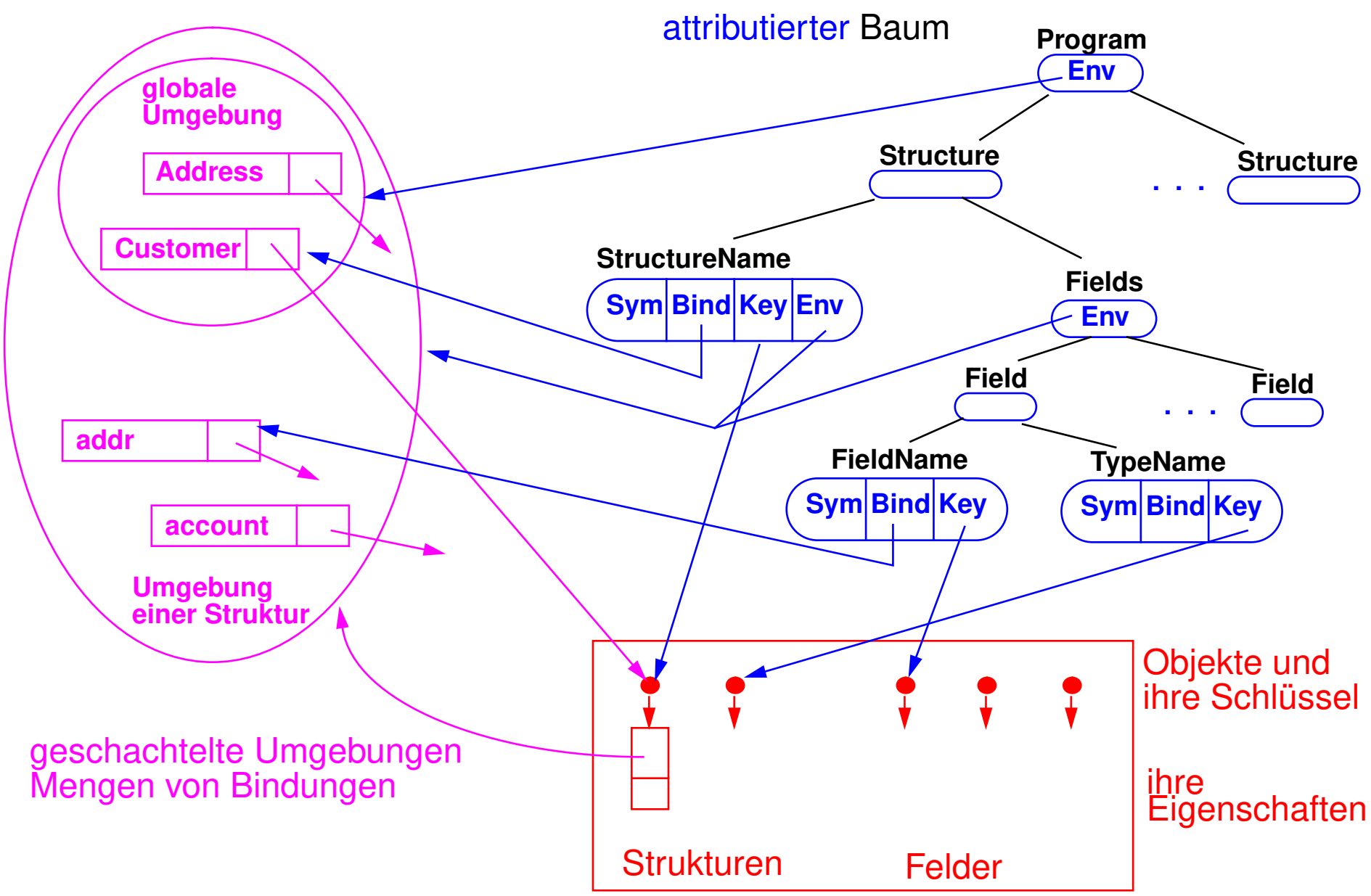
StructureName und Fields haben die Umgebung der Struktur

Jeder Namensknoten hat Attribute für

- die Codierung des Symbols,
- die Bindung seines Namens, und
- sein Schlüssel

© 2002 bei Prof. Dr. Uwe Kastens

Attribute, Umgebungen und Schlüssel



© 2002 bei Prof. Dr. Uwe Kastens

Umgebungsmodul

Implementiert den abstrakten Datentyp **Environment** (Umgebung):
hierarchisch geschachtelte Mengen von **Bindungen (Name, Umgebung, Schlüssel)**

Funktionen:

NewEnv ()	bildet eine neue Umgebung e, als Wurzel einer Hierarchie; eingesetzt im **Wurzelkontext**
NewScope (e_1)	bildet eine neue Umgebung e_2 das in e_1 geschachtelt ist. Jede Bindung von e_1 ist auch eine Bindung von e_2, falls sie dort nicht verdeckt ist; eingesetzt im **Range-Kontext**
BindIdn (e, id)	bildet eine neue Bindung (id, e, k) falls e noch keine Bindung für id hat; dann ist k ein neuer Schlüssel für ein neues Objekt; das Ergebnis ist immer das Bindungstripel (id, e, k); eingesetzt für **definierende Auftreten**.
BindingInEnv (e, id)	liefert ein Bindungstripel (id, e_1, k) aus e oder einer umfassenden Umgebung von e; liefert NoBinding, falls es keine solche Bindung gibt; eingesetzt für **angewandte Auftreten**
BindingInScope (e, id)	liefert ein Bindungstripel (id, e, k) aus e, falls e es direkt enthält, sonst NoBinding; eingesetzt z. B. für **qualifizierte Namen**

© 2002 bei Prof. Dr. Uwe Kastens

PDL: Generator für Definitionsmodule

zentrale Datenstruktur ordnet **Objekten 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: in abhängigen Operationen im Baum

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

aus **Spezifikationen**

Eigenschaftsname: Eigenschaftstyp;

werden Funktionen **`ResetP, SetP, GetP`** für die Eigenschaft **`P`** generiert.

© 2002 bei Prof. Dr. Uwe Kastens

Beispiel: Namen und Eigenschaften für Struktur-Generator

Abstrakte Syntax

```
RULE: Descriptions  LISTOF Import | Structure                END;
RULE: Import ::= 'import' ImportNames 'from' FileName        END;
RULE: ImportNames   LISTOF ImportName                        END;
RULE: Structure ::= StructureName '(' Fields ')'             END;
RULE: Fields        LISTOF Field                             END;
RULE: Field ::=     FieldName ':' TypeName ';'               END;
RULE: StructureName ::= Ident                                END;
RULE: ImportName ::=    Ident                                END;
RULE: FieldName ::=     Ident                                END;
RULE: TypeName ::=      Ident                                END;
```

Verschiedene Nichtterminale für Bezeichner in unterschiedlichen Rollen, wenn unterschiedliche Berechnungen erwartet werden, z. B. für definierende und angewandte Auftreten.

© 2002 bei Prof. Dr. Uwe Kastens

Umgebungsattribute berechnen

2 Symbole spielen die **Rolle eine Range**: Unterbaum begrenzt Gültigkeit darin enthaltener Definitionen

```
CLASS SYMBOL Range: Env: Environment;
SYMBOL Descriptions INHERITS Range END;
SYMBOL Fields  INHERITS Range END;
```

Wurzel der Umgebungshierarchie

```
SYMBOL Descriptions COMPUTE
   SYNT.Env = NewEnv ();
END;
```

Jedes Structure-Objekt hat eine **Umgebung als Eigenschaft**.

Darin werden die Feldnamen gebunden.

Sie wird in die globale Umgebung eingebettet

```
RULE: Structure ::= StructureName '(' Fields ')'
COMPUTE
   Fields.Env = StructureName.Env;
END;

SYMBOL StructureName COMPUTE
   SYNT.GotEnvir =
      IF (EQ (GetEnvir (THIS.Key, NoEnv), NoEnv),
         ResetEnvir
            (THIS.Key,
             NewScope (INCLUDING Range.Env)));

   SYNT.Env =
      GetEnvir (THIS.Key, NoEnv) <- SYNT.GotEnvir;
END;
```

© 2002 bei Prof. Dr. Uwe Kastens

Definierende Auftreten von Bezeichnern

Alle **IdentOcc**-Knoten haben diese 3 Attribute

Diese Brechnungen sind gleich für alle **IdentOcc**-Knoten

```
CLASS SYMBOL IdentOcc:
   Sym: int,
   Bind: Binding,
   Key: DefTableKey;

CLASS SYMBOL IdentOcc COMPUTE
   SYNT.Sym = TERM;
   SYNT.Key =  KeyOf (SYNT.Bind);
END;
```

Alle definierenden Auftreten **binden** ihren Namen im Env-Attribut des **nächst-umfassenden Range**

```
CLASS SYMBOL DefineAnIdent INHERITS IdentOcc
COMPUTE
   SYNT.Bind =
      BindIdn (INCLUDING Range.Env, THIS.Sym);
END;

SYMBOL StructureName INHERITS DefineAnIdent END;
SYMBOL ImportName    INHERITS DefineAnIdent END;
SYMBOL FieldName     INHERITS DefineAnIdent END;
```

© 2002 bei Prof. Dr. Uwe Kastens

Angewandte Auftreten von Bezeichnern

Vorbedingung für das Suchen einer Bindung:

Alle Bindungen in den umfassenden Ranges sind hergestellt.

```
SYMBOL Descriptions COMPUTE
   SYNT.GotBindings =
      CONSTITUENTS DefineAnIdent.Bind SHIELD Range;
END;

SYMBOL Fields COMPUTE
   SYNT.GotBindings =
      CONSTITUENTS DefineAnIdent.Bind
      <- INCLUDING Range.GotBindings;
END;
```

Für den Bezeichner eine Bindung in den umfassenden Umgebungen suchen

Misserfolg melden

Nur ein angewandtes Auftreten

```
CLASS SYMBOL UseAnIdent INHERITS IdentOcc COMPUTE
   SYNT.Bind =
      BindingInEnv (INCLUDING Range.Env, THIS.Sym)
         <- INCLUDING Range.GotBindings;

   IF (EQ (SYNT.Bind, NoBinding),
      message (ERROR,
         CatStrInd ("Name is not defined: ",
                    THIS.Sym),
          0, COORDREF));
END;

SYMBOL TypeName   INHERITS UseAnIdent END;
```

© 2002 bei Prof. Dr. Uwe Kastens

Eigenschaften definieren, setzen und prüfen

Beispiel: Ein Feldname darf nicht an der Stelle eines Typnamens auftreten:

```
Customer (addr: Address; account: addr;)
```

Boolsche Eigenschaft isField definiert für das Werkzeug PDL:

```
isField: int;
```

Eigenschaft auf wahr setzen für FieldName:

```
SYMBOL FieldName COMPUTE
   SYNT.GotisField =
      ResetisField (THIS.Key, 1);
END;
```

Eigenschaft prüfen im TypeName Kontext:

```
SYMBOL TypeName COMPUTE
  IF (GetisField (THIS.Key, 0),
    message (ERROR,
      CatStrInd
        ("Field name not allowed",
         THIS.Sym),0, COORDREF))
   <- INCLUDING Program.GotisField;
END;
```

Vorbedingung garantiert, dass alle ResetisField aufgerufen sind, bevor ein GetisField aufgerufen wird:

```
SYMBOL Program COMPUTE
  SYNT.GotisField =
    CONSTITUENTS FieldName.GotisField;
END;
```

Eigenschaften von Objekten
- beschreiben die **Bedeutung**
- gegen **Einschränkungen** prüfen
- zur **Transformation** verwenden

© 2002 bei Prof. Dr. Uwe Kastens

Richtlinien zu: Eigenschaften von Objekten (B)

Vorbereitung:

- Objekte werden meist durch Bezeichner in der Beschreibung benannt; durch **`DefTableKeys`** repräsentiert
- Modul der **Namensanalyse** bindet Auftreten von Bezeichnern.
- Symbolknoten der Bezeichner haben ein **`Key`**-Attribut; es identifiziert das Objekt

Entwurfsschritte für Berechnungen von Eigenschaften:

1. Bestimme **Namen und Typ der Eigenschaft** und spezifiziere sie für PDL.
2. Bestimme die **Kontexte und die Operation für das Setzen** der Eigenschaft.
3. Bestimme die **Kontexte für das Lesen** der Eigenschaft.
4. Bestimme die **Abhängigkeiten zwischen (2) und (3)**.
   In einfachen Fällen ist dies: "alle Setzen vor jedem Lesen".
5. Spezifiziere (2), (3) und das Abhängigkeitsmuster (4).

Eigenschaften des Objektes **setzen oder lesen**, möglichst im **Kontext des Symbols für den Bezeichner** platzieren
nicht die die **`Key`**-Werte im Baum weiterreichen.

Wenn möglich, **SYMBOL-Berechnungen** verwenden (siehe A).

© 2002 bei Prof. Dr. Uwe Kastens

Themen für Projekte entwickeln

- spezielles, enges Anwendungsgebiet
  (aus Programmentwicklung, Web-Entwurf, Informatik-fremdes Gebiet, Hobby)
- gleichartige, strukturierte Texte in verschiedenen Varianten benötigt
- systematisch erzeugbar aus einfachen Beschreibungen

Beispiele:

- Datenstruktur-Generator
- Roboter-Steuerung
- Molekül-Darstellung
- Spielplan-Gestaltung

Vorgehen:

1. Thema formulieren
2. Beispiele für Ausgabe entwickeln
3. Beispiele für Beschreibungen (Eingabe) entwickeln
4. Regeln für Erzeugung von Ausgabe aus Beschreibungen angeben

Prüfen: **Thema geeignet?**

© 2002 bei Prof. Dr. Uwe Kastens

Technik: Do it once

Aufgabe:

- Viele Auftreten eines Namens an dasselbe Objekt gebunden
- An genau einem Auftreten eine Berechnung ausgeführen (z. B. Zieltext für das Objekt)

Lösung:

Boolsches Attribut berechnen:
An genau einem Auftreten wahr
sonst falsch.
Eigenschaft benutzen; Entwurfsschritte:

1. Eigenschaft definieren: `Done: int;`
2. Setzen im Namenskontext, wenn noch nicht gesetzt
3. Lesen im Namenskontext
4. keine Abhängigkeiten
5. siehe rechts

```
CLASS SYMBOL DoItOnce:
                  DoIt: int;

CLASS SYMBOL DoItOnce
      INHERITS IdentOcc COMPUTE
   SYNT.DoIt =
      IF (GetDone (THIS.Key, 0),
         0,
         ORDER
          (ResetDone (THIS.Key, 1),
          1));
END;
```

Anwendung:

```
SYMBOL StructName INHERITS DoITOnce
COMPUTE
   SYNT.Text =
      IF (THIS.DoIt,
         PTGTransform (...),
         PTGNULL);
END;
```

© 2003 bei Prof. Dr. Uwe Kastens