4. Names, Entities, and Properties

Program constructs in the tree
(e.g. definitions) may

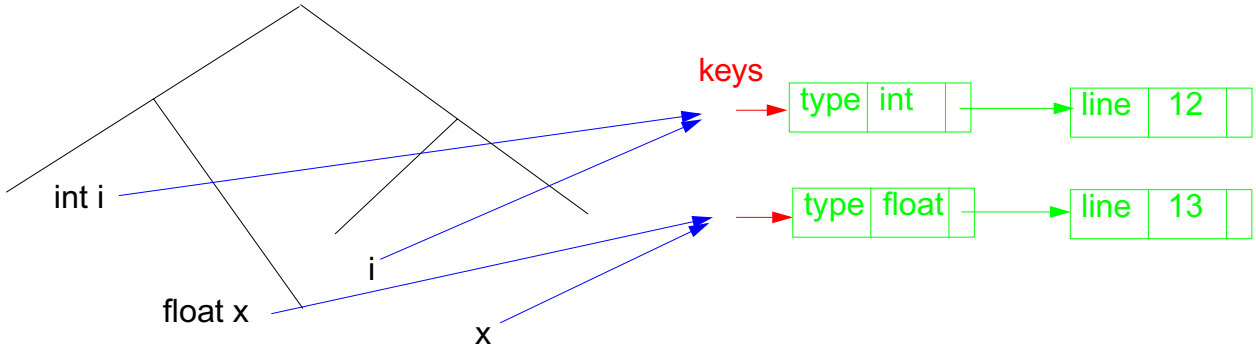
- introduce an **entity**
  (e.g. a variable, a class, or a function)
- **bind the entity to a name**
- associate **properties to the entity**
  (e.g. type, kind, address, line)

The **definition module** stores **program entities with their properties**, e.g. a variable with its type and the line number where it is defined.

Entities are identified by keys of the definition module.

Name analysis binds names to entities.

The **properties** of an entity are represented by a list of **(kind, value)-pairs**



Lecture Generating Software from Specifications SS 2012 / Slide 401

Objectives:

Understand the use of a definition module

In the lecture:

The concepts will be explained.

Basic name analysis provided by symbol roles

Symbol roles:

Grammar root:

```
SYMBOL Program INHERITS RootScope END;
```

Ranges containing definitions:

```
SYMBOL Block INHERITS RangeScope END;
```

Defining identifier occurrence:

```
SYMBOL DefIdent INHERITS IdDefScope END;
```

Instantiation in a `.specs` file for Algol-like scope rules:

```
$/Name/AlgScope.gnrc:inst
```

for C-like scope rules:

```
$/Name/CScope.gnrc: inst
```

Applied identifier occurrence:

```
SYMBOL UseIdent INHERITS IdUseEnv, ChkIdUse END;
```

Required attributes:

```
CLASS SYMBOL IdentOcc: Sym: int;
CLASS SYMBOL IdentOcc COMPUTE SYNT.Sym = TERM; END;

SYMBOL DefIdent INHERITS IdentOcc END;
SYMBOL UseIdent INHERITS IdentOcc END;
```

Provided attributes:

```
SYMBOL DefIdent, UseIdent: Key: DefTableKey, Bind: Binding;
SYMBOL Program, Block: Env: Environment;
```


Lecture Generating Software from Specifications SS 2012 / Slide 401a

Objectives:

Basic name analysis is provided by a library module

In the lecture:

- The roles of the module are explained.
- Their use is explained.

PDL: A Generator for Definition Modules

central data structure associates **properties to entities**,
e.g. *type of a variable, element type of an array type.*

Entities are identified by a **key** (type **`DefTableKey`**).

Operations:

`NewKey ( )`	yields a new key
`ResetP (k, v)`	for key **`k`** the property **`P`** is set to the value **`v`**
`SetP (k, v, d)`	for key **`k`** the property **`P`** is set to the value **`v`**, if it was not set, otherwise to the value **`d`**
`GetP (k, d)`	for key **`k`** it yields the value of the property **`P`** if it is set, otherwise it yields **`d`**

Functions are called in **computations in tree contexts**.

PDL generates functions **`ResetP, SetP, GetP`** from specifications of the form

e.g. **PropertyName: ValueType;**

```
Line: int;
Type: DefTableKey;
```


Lecture Generating Software from Specifications SS 2012 / Slide 402

Objectives:

Introduction of the property genrator PDL

In the lecture:

The functions are explained.

Example: Set and Get a Property

The line number is associated as a property in a **`.pdl`** file:

```
Line: int;
```

It is set in definition contexts and got in use contexts.

All set computations in **definition** contexts have to precede any get in **use** contexts.

```
SYMBOL Program INHERITS RootScope END;
RULE: Program LISTOF Definition | Use COMPUTE
   Program.GotLine = CONSTITUENTS Definition.GotLine;
END;

RULE: Definition ::= 'def' NameDef END;
RULE: Use ::= 'use' NameUse END;

SYMBOL NameDef INHERITS IdentOcc, IdDefScope COMPUTE
   SYNT.GotLine = ResetLine (THIS.Key, LINE);
   printf ("%s defined in line %d\n", StringTable(THIS.Sym), LINE);
END;

SYMBOL NameUse INHERITS IdentOcc, IdUseEnv, ChkIdUse COMPUTE
   printf ("%s defined in line %d used in line %d\n",
            StringTable(THIS.Sym), GetLine (THIS.Key, 0), LINE)
   <- INCLUDING Program.GotLine;
END;
```


Lecture Generating Software from Specifications SS 2012 / Slide 403

Objectives:

Learn to use the PDL functions in tree contexts

In the lecture:

The following aspects are explained

- The tree contexts,
- the attributes Sym and Key,
- the property definition,
- the PDL function calls,
- the dependences based on pre- and post conditions (see GSS-3.7).

The functions are explained.

Design Rules for Property Access (B)

Preparation:

- Usually identifiers in the tree refer to entities represented by **`DefTableKeys`**; an identifier is bound to a key using the **name analysis module** (see Ch.5).
- Symbol nodes for identifiers have a **`Key`** attribute; it identifies the entity

Design steps for the computation of properties:

1. Specify **name and type of the property** in the notation of PDL.
2. Identify the **contexts where the property is set**.
3. Identify the **contexts where the property is used**.
4. Determine the **dependences between (2) and (3)**.
   In simple cases it is: "all set operations before any get operation".
5. Specify (2), (3), and the pattern of (4).

Try to locate the computations that **set or get properties** of an entity **in the context of the identifier**, if possible; avoid to propagate the **`Key`** values through the tree.

Use **SYMBOL computations** as far as possible (see design rules A).

Lecture Generating Software from Specifications SS 2012 / Slide 404

Objectives:

Apply PDL operations systematically

In the lecture:

The design steps are applied to the following examples:

- Report a message for more than one occurrence of an entity.
- Output a line number at every defining occurrence.
- At a using occurrence output the line number of the defining occurrence.
- At an occurrence output the line number of the previous occurrence.
- Report a message if a use occurs before its definition.

The functions are explained.

Technique: Do it once

Task:

- Many occurrences of an identifier are bound to the same entity (key)
- For each entity a computation is executed at exactly one (arbitrary) occurrence of its identifier (e.g. output some target code)

Solution:

Compute an **attribute of type bool**:
True at exactly one occurrence of the key, false elsewhere.

Design steps:

1. Property specification: `Done: int;`
2. Set in name context, if not yet set.
3. Get in name context.
4. No dependences!
5. see on the right:

```
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;
```


Lecture Generating Software from Specifications SS 2012 / Slide 405

Objectives:

Learn to use the technique

In the lecture:

The technique is explained