4. Names, Entities, and Properties

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

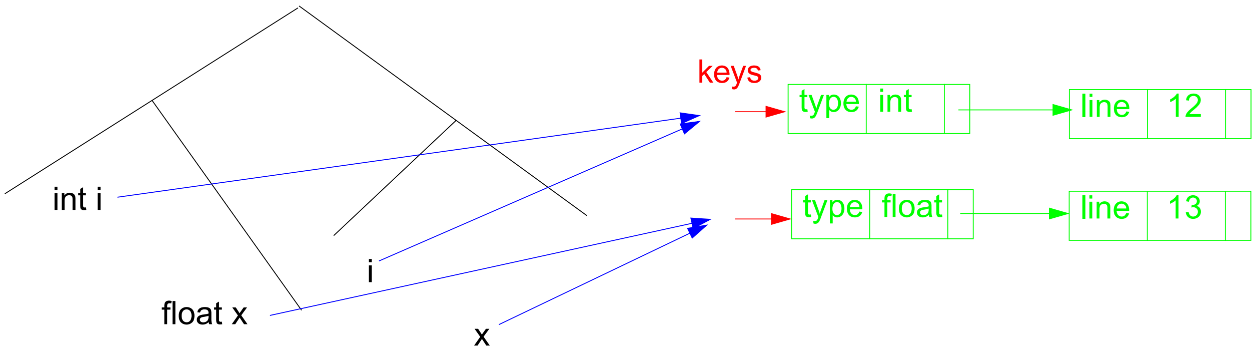
- introduce an **entity** (e.g. a variable, a class, or a function)
- **bind the entity to a name**
- associate **properties to the entity** (e.g. type, kind, address, line)

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**



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;
```

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;
```

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

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

for C-like scope rules:

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

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

PropertyName: ValueType;

e.g.

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

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;
```

GSS-4.4

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).

GSS-4.5

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;
```