4.2 Definition module

Central data structure, stores properties of program entities e. g. *type of a variable, element type of an array type*

A program entity is identified by the key of its entry in the data structure.

Operations:

NewKey()	yields a new key
ResetP (k, v)	sets the property P to have the value v for key k
SetP (k, v, d)	as ResetP; but the property is set to d if it has been set
GetP (k, d)	yields the value of the Property P for the key k; yields the default-Wert d, if P has not been set

Operations are called as dependent computations in the tree

Implementation: a property list for every key, for example

Generation of the definition module: From specifications of the form

Property name : property type; ElementNumber: int;

functions ResetElementNumber, SetElementNumber, GetElementNumber are generated.

4.3 Type analysis

Task: Compute and check types of program entities and constructs at compile time

- defined entities (e. g. variables) have a type property, stored in the definition module
- program constructs (e. g. expressions) have a type attribute, associated to their symbol resp. tree node special task: resolution of overloaded operators (functions, methods)
- types themselves are program entities represented by keys;
 named using type definitions; unnamed in complex type notations
- types have properties e. g. the element type of an array type
- type checking for program entities and for program constructs a type must / may not have certain properties in certain contexts compare expected and given type; type relations: equal, compatible; compute type coercion

Lecture Compiler I WS 2001/2002 / Slide 79

Objectives:

CI-79

CI-80

before

Properties of program entities

In the lecture:

- Explain the operations,
- explain the generator,
- give examples.

Suggested reading:

Kastens / Übersetzerbau, Section S. 130 unten

Assignments:

• Use the PDL tool of Eli

Questions:

• Give examples where calls of the operations are specified as computations in tree contexts. Describe how they depend on each other.

Lecture Compiler I WS 2001/2002 / Slide 80

Objectives:

Learn to categorize the tasks

In the lecture:

- · Motivate type analysis tasks with typical properties of strongly typed languages;
- give examples

Suggested reading:

Kastens / Übersetzerbau, Section 6.1

Questions:

- Give examples for program entities that have a type property and for others which don't.
- Enumerate at least 5 properties of types in Java, C or Pascal.
- Give an example for a recursively defined type, and show its representation using keys.



Overloading resolution for operators

Overloading: same operator symbol (source operator) is used for several target operators having different signatures and different meanings, e. g. specified by a table like:

symbol signature

=

meaning addition of integral numbers

- int X int -> int + +
 - floating point addition real X real -> real union of sets
- set X set -> set +
 - t X t -> boolean comparison for values of type t

Coercion: implicitly applicable type conversion: e. g. int -> real, char -> string, ...



Type analysis for object-oriented languages

Class hierarchy is a type hierarchy:

Circle k = new Circle (...); GeometricShape f = k;

implicit type coercion: class -> super class explicit type cast: class -> subclass

k = (Circle) f;

Variable of class type may contain an object (reference) of its subclass

Check signature of overriding methods:

calls must be type safe; Java requires the same signature; following weaker requirements are sufficient (contra variant parameters, language Sather):

call of dynamically Variable: X x; A a; P p; a = x.m(p);bound method: C c; B b; class X { C m (Q q) { Use of q;... return c; } } super class subclass class Y { B m (R r) { USe of r;... return b; } }

Analyse dynamic methode binding; try to decide it statically:

static analysis tries to further restrict the run-time type:

GeometricShape f;...; f = new Circle(...);...; a = f.area();

Lecture Compiler I WS 2001/2002 / Slide 83

Objectives:

CI-83

CI-84

Understand the task of overloading resolution

In the lecture:

Explain

- · overloaded operators, functions, and methods,
- attribute computations,
- Eli tool OIL

Suggested reading:

Kastens / Übersetzerbau, Section 6.1

Assignments:

• overloading resolution as in C (Exercise 23)

Lecture Compiler I WS 2001/2002 / Slide 84

Objectives:

Understand classes as types

In the lecture:

Explain

- · class hierarchy type coercion
- · type checking for dynamically bound methods calls
- predict the runtime classs of objects

Questions:

• Why would overridden methods not be type safe if they had "covariant" parameters (all 3 arrows between the classes X and Y would point up)? That is the situation in Eiffel.

