EDAN65: Compilers

Reference Attribute Grammars

Parameterized Attributes and Collection Attributes

Görel Hedin

Revised: 2023-09-18

Adapted for EDAP15: Program Analysis
Christoph Reichenbach
Revised: 2024-01-21

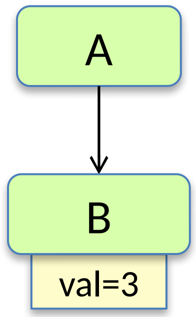
Parameterized attributes

Parameterized attributes

an attribute can have one or more parameters

Example: Find out if B's val is over some given limit

```
A ::= B;
B ::=
<val:int>;
```



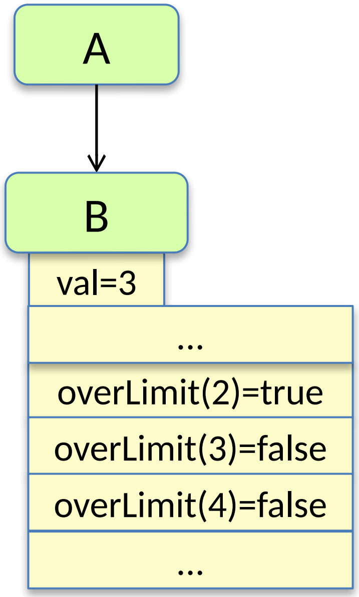
Parameterized attributes

an attribute can have one or more parameters

Example: Find out if B's val is over some given limit

```
A ::= B;
B ::=
<val:int>;
```

```
syn boolean B.overLimit(int limit) =
      getval() > limit;
```



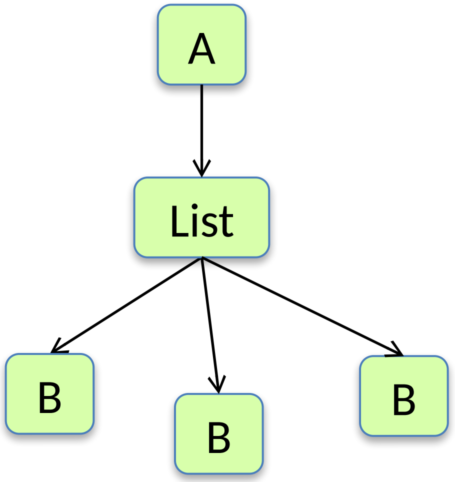
Unbounded number of attribute instances – one for each argument.
Similar to functions. But accessed values are cached.
Only accessed attribute instances will be evaluated.

Parameterized attributes

list equations can use both index and parameters

Draw some isBefore attributes and their values!

```
A ::= B*;
B;
```



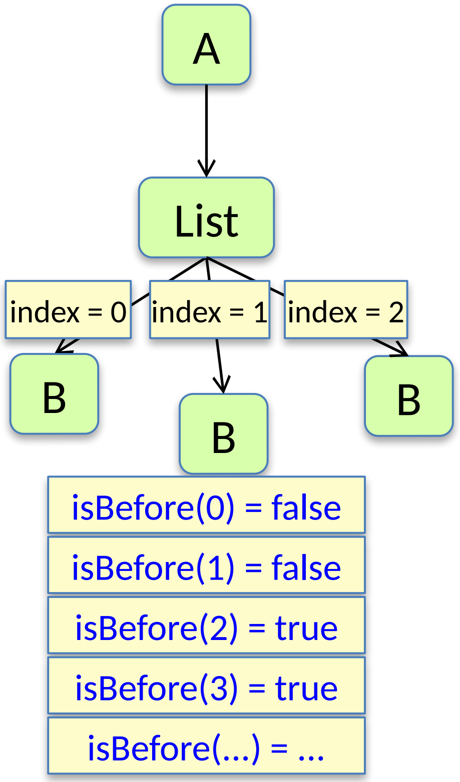
```
inh boolean B.isBefore(int i);
eq A.getB(int index).isBefore(int i) = index
< i;
```

Parameterized attributes

list equations can use both index and parameters

```
A ::= B*;
B;
```

```
inh boolean B.isBefore(int i);
eq A.getB(int index).isBefore(int i) = index
< i;
```



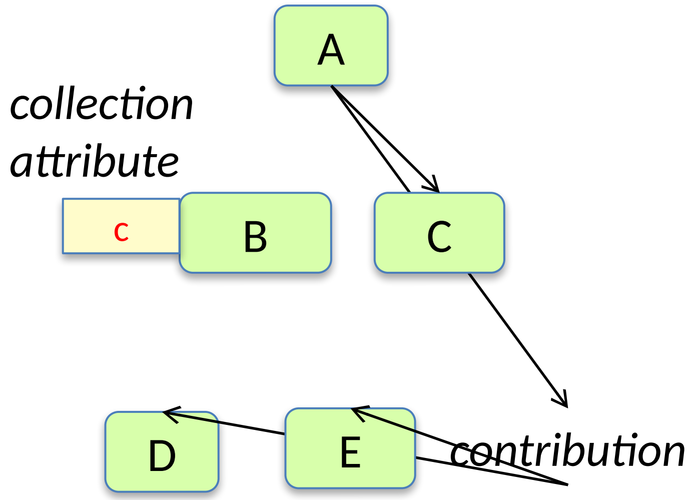
Collection attributes

Collection attributes

motivation

A collection attribute is defined by *contributions*, instead of by a single equation.

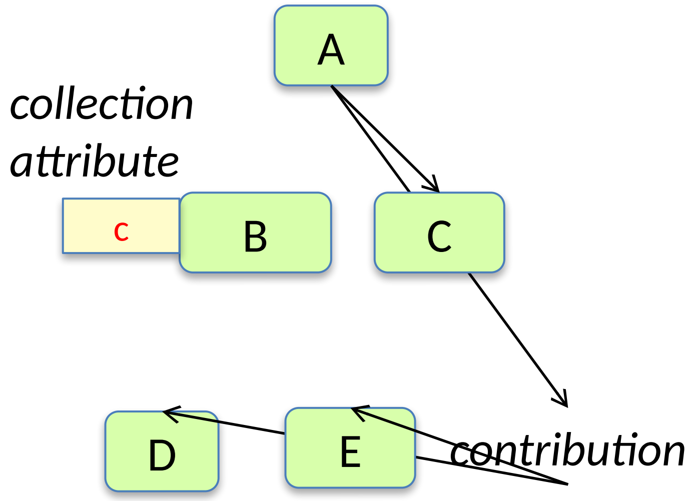
Use for values combined from many small parts spread out over the tree.



Collection attributes

motivation

A collection attribute is defined by *contributions*, instead of by a single equation.



Use for values combined from many small parts spread out over the tree.

Example uses:

- collect compile-time errors in a program
- collect what uses are bound to a specific declaration
- count the number of if-statements in a method

When a collection attribute is accessed, the attribute evaluator will automatically traverse the AST and find the contributions.

Collection attribute

structure

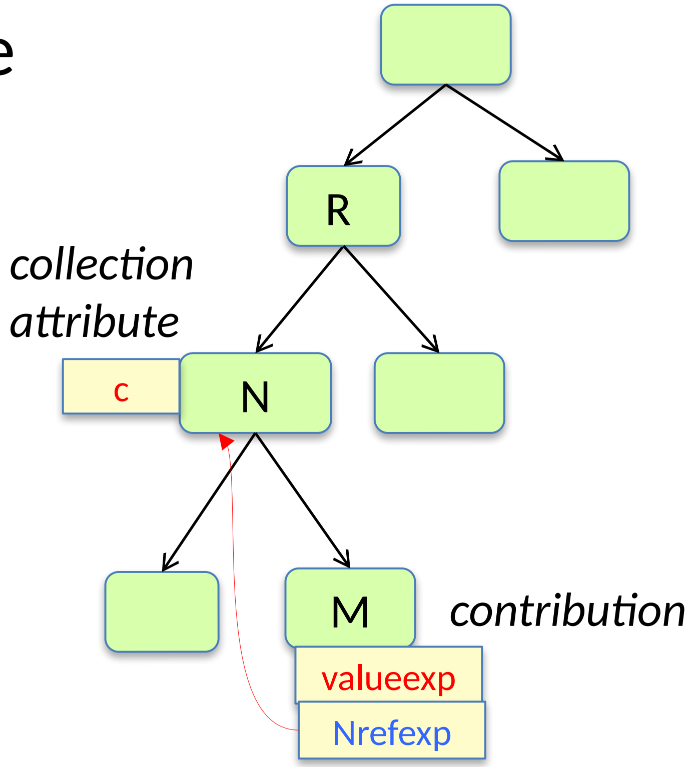
Declaration of **collection** attribute c in node type N:

```
coll T N.c() [freshexp] with m root
R;
```

The method m must be **commutative**

A **contribution** from a node type M:

```
M contributes
valueexp
  when condition
  to N.c()
  for Nrefexp
```



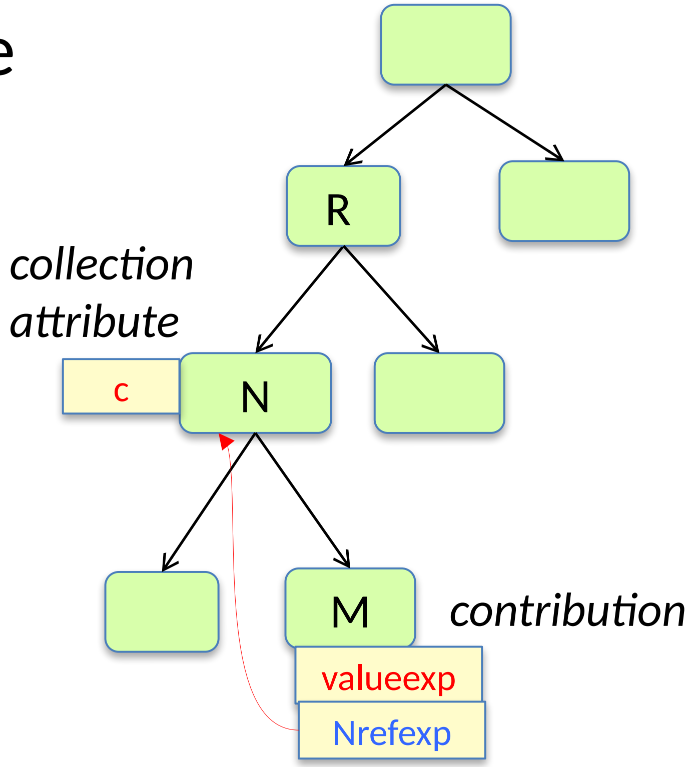
Collection attribute

structure

Declaration of **collection** attribute c in node type N:

```
coll T N.c() [freshexp] with m root
R;
```

- T is the type of c
- *freshexp* is a fresh T object (empty collection)
- m is a **commutative** mutating method used for adding contributions to c
- R is an AST node type, identifying the subtree where contributions can be



A **contribution** from a node type M:

```
M contributes
valueexp
  when condition
  to N.c()
  for Nrefexp
```

- *valueexp* is the value to be contributed
- *condition* is a condition indicating if *valueexp* should be added or not
- *Nrefexp* is a reference to an N node

Evaluation algorithm

When c is accessed for the first time:

- the empty collection is created using *freshexp*
- the subtree at the upward nearest R is traversed, and all contributions are added to c
- c is cached

Collection attribute

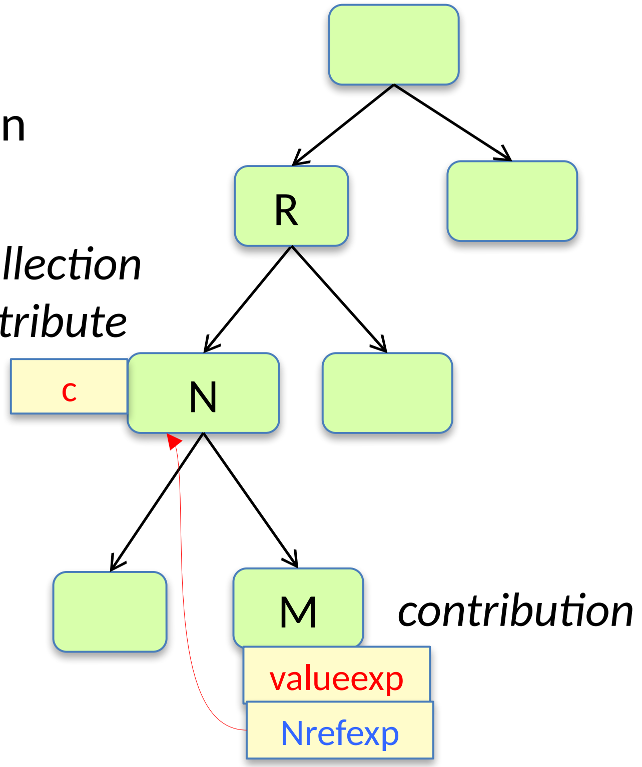
optional elements in the definition

Declaration of **collection** attribute c in node type N:

```
coll T N.c() [freshexp] with m root
R;
```

The method m must be **commutative**

- if "[freshexp]" is left out, the default constructor for T will be used.
- if "with m" is left out, the method name "add" is used
- if "root R" is left out, R is set to the type of the root node



A **contribution** from a node type M:

```
M contributes
valueexp
  when condition
  to N.c()
  for Nrefexp
```

- if "when condition" is left out, the value will always be added
- "for Nrefexp" can be left out if N=R

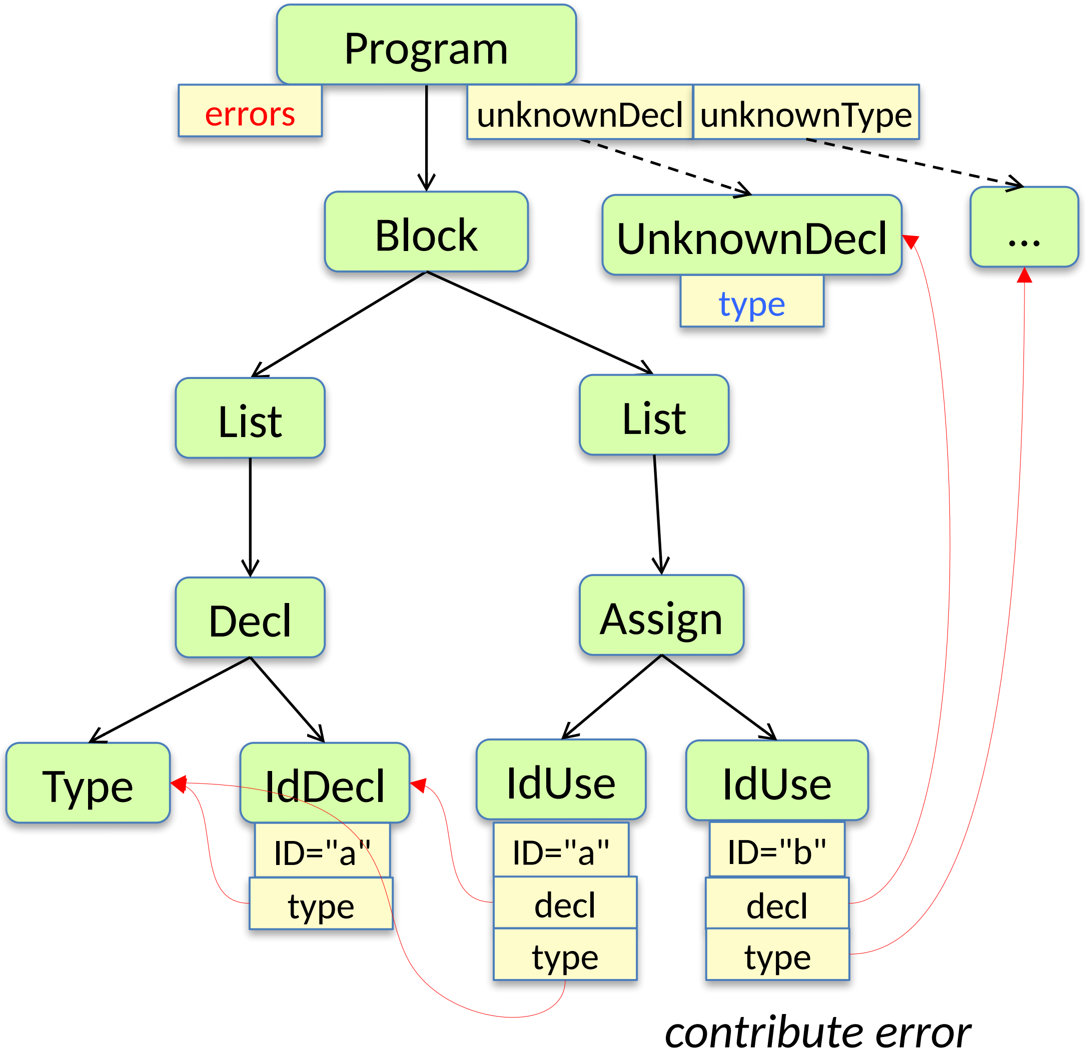
Collect errors

Example: Collect errors

Error checking: collect all errors

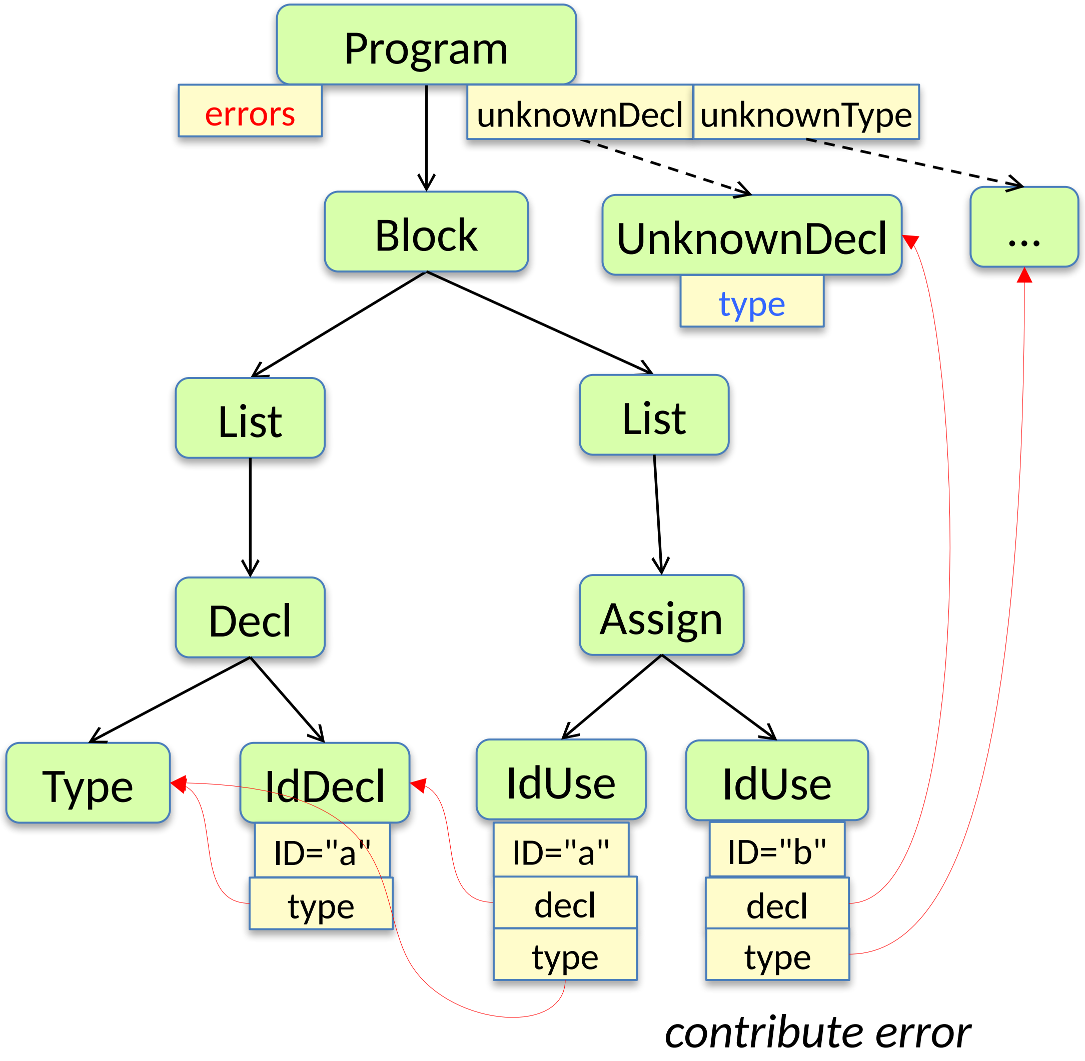
We would like an attribute errors in the root, containing all error messages.

We would like an easy way to "contribute" different kinds of errors from different nodes in the AST.



Example: Collect errors

Error checking: collect all errors

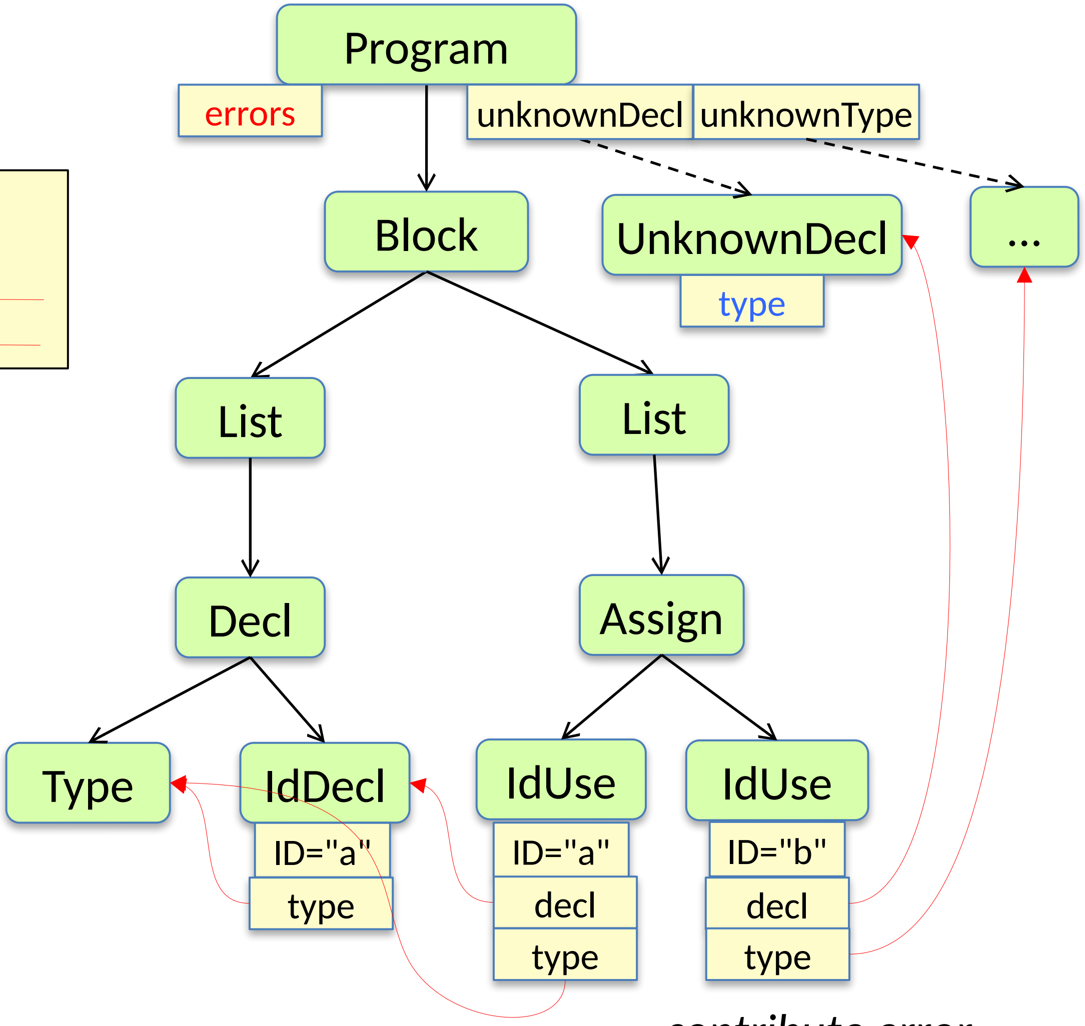


Example: Collect errors

Error checking: collect all errors

Declare the errors collection:

```
coll Set<String> Program.errors()
  [new HashSet<String>()]
 ~~with add~~
 ~~root Program;~~
```



contribute error

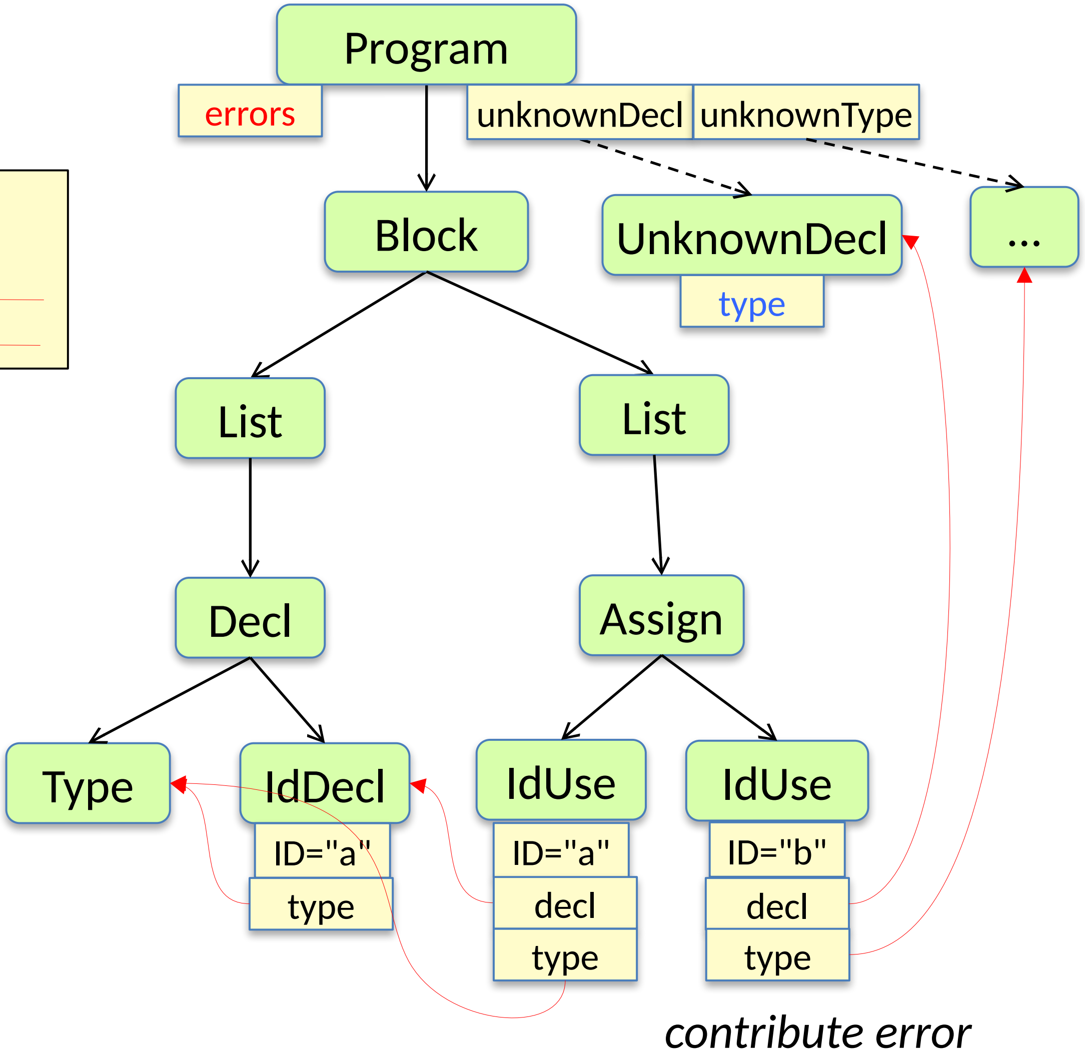
because of defaults, these optional parts can be skipped in this case

Example: Collect errors

Error checking: collect all errors

Declare the errors collection:

```
coll Set<String> Program.errors()
  [new HashSet<String>()]
 ~~with add~~
 ~~root Program;~~
```



because of defaults, these optional parts can be skipped in this case

Example: Collect errors

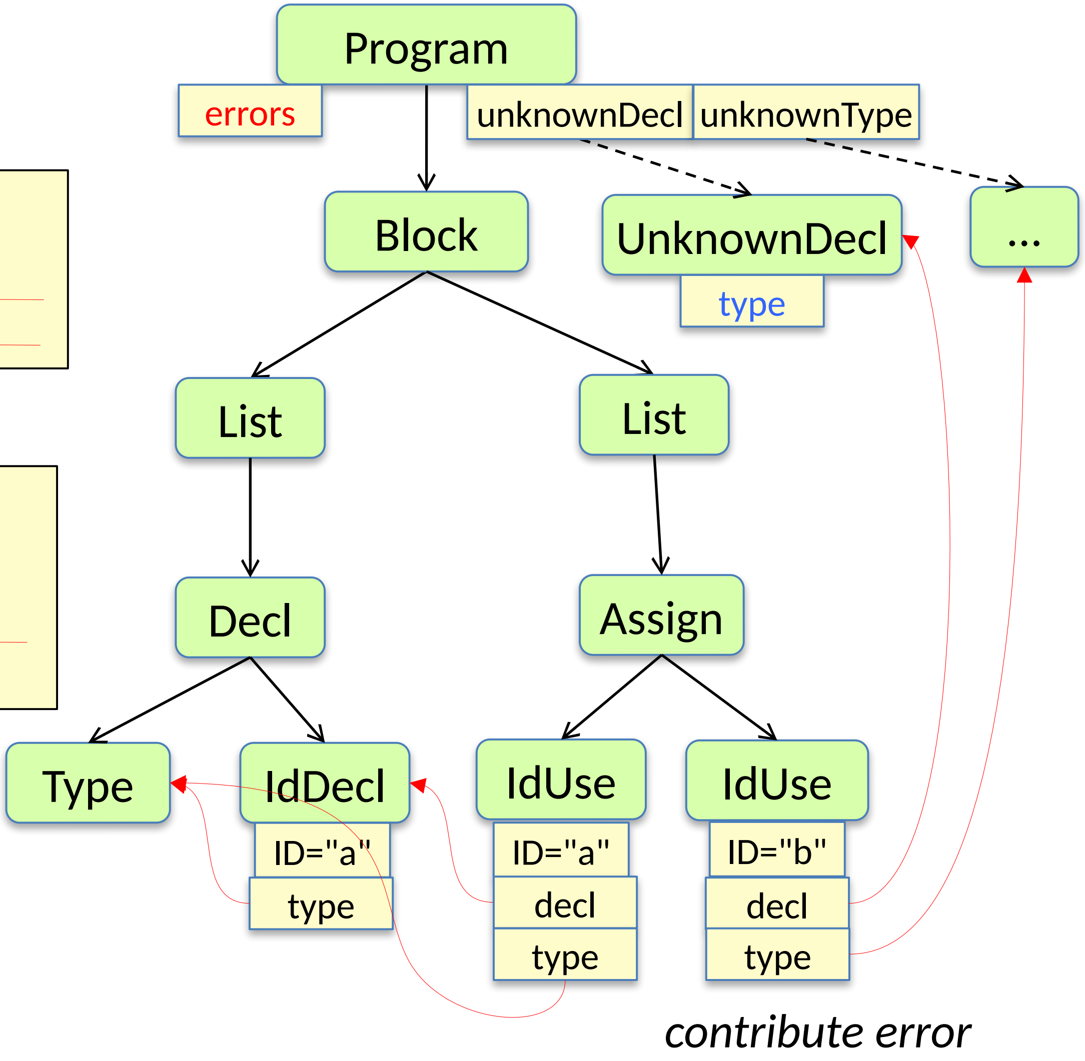
Error checking: collect all errors

Declare the errors collection:

```
coll Set<String> Program.errors()
  [new HashSet<String>()]
  ~~with add~~
  ~~root Program;~~
```

Contribute an error

```
IdUse contributes "Undeclared
variable"
  when decl().isUnknown()
  ~~to Program.errors()~~
  for theProgram();
```



because of defaults, these optional parts can be skipped in this case

Summary questions:

collection attributes, error checking

- What is a collection attribute?
- How can a collection of error message be implemented?