

Compiler Construction

~ Error Recovery ~

Goal

Observation

Reporting only one error at a time

- is costly for the user
 - is frustrating
 - is CPU consuming
- ⇒ Think to our planet!

Problem

How to report all (syntactic) errors
simultaneously?

Error recovery: different strategies

Error Recovery

Error Recovery is the process of adjusting input stream to handle errors:

- Deletion of token types from input stream
- Insertion of token types
- Substitution of token types

There are two classes of recovery:

- **Local Recovery:** adjust input at point where error was detected.
- **Global Recovery:** adjust input before point where error was detected

Local Recovery (1/2)

Use an error symbol ! \Rightarrow Try to find a **synchronizing token** and resume parsing!

Let's consider the following grammar:

```
exps -> exp
exps -> exps ; exp

exp -> NUMBER
exp -> exp + exp
exp -> (exps)

exp -> (error)
exps -> error ; exp
```

Local Recovery (2/2)

error is considered as a terminal symbol

When the (LR) parser reaches an error state, it proceeds as follows:

- 1 Dig in the stack to find a nice place (where **error** is shifted)
- 2 Throw away all unpleasant look-ahead
- 3 Resume normal parsing

Global Recovery (1/2)

Try to insert/delete token from the input stream at a point before the error was detected.

Let's consider the following example:

```
let type a := intArray [10]
      of 0
in /* ... */
end
```

The parser will try to replace
type with **var**.

Global Recovery (2/2)

In practice

The parser finds the smallest set of insertions and deletions that would turn the input stream into a correct stream.

Burke-Fisher error repair

Try every single token insertion, deletion or replacement at every point that occurs no earlier than K tokens in the past.
The grammar stays unmodified, only the parsing engine is modified.

Semantic values for errors

An AST node must be produced for errors!

Summary

local error
repair

global
error
repair

Burke-
Fisher

Semantic
values