Error Recovery

- Michael Wollowski
- Rose-Hulman Institute of Technology

Error recovery goals

- What should happen when your parser finds an error in the user's input?
  - stop immediately and signal an error
  - record the error but try to continue
- In the first case, the user must recompile from scratch after possibly a trivial fix
- In the second case, the user might be overwhelmed by a whole series of error messages, all caused by essentially the same problem

Strategies

- There are many different general strategies that a parser can employ to recover from a syntactic error.
- Although no one strategy has proven itself to be universally acceptable, a few methods have broad applicability.
- Here we introduce the following strategies:
  - panic mode
  - phrase level
  - error productions
  - global correction

Panic-mode recovery

- In *panic-mode recovery*, on discovering an error, the parser discards input symbols one at a time until one of a designated set of synchronizing tokens is found.
- The synchronizing tokens are usually delimiters, such as a semicolon, whose role in the source program is clear.
- The compiler designer must select the synchronizing tokens appropriate for the source language.

Panic-mode recovery - Evaluation

- This is the simplest method to implement and can be used by most parsing methods.
- While panic-mode correction often skips a considerable amount of input without checking it for additional errors, it has the advantage of simplicity.
- In situations where multiple errors in the same statement are rare, this method may be quite adequate.

Phrase-level recovery

- In *phrase-level recovery*, on discovering an error, a parser may perform local correction on the remaining input; that is, it may replace a prefix of the remaining input by some string that allows the parser to continue.
- A typical local correction would be to replace a comma by a semicolon, delete an extraneous semicolon, or insert a missing semicolon.
- The choice of the local correction is left to the compiler designer.
- We must be careful to choose replacements that do not lead to infinite loops, as would be the case, for example, if we always inserted something on the input ahead of the current input symbol.

Phrase-level recovery - Evaluation

- This type of replacement can correct any input string and has been used in several error-repairing compilers.
- The method was first used with top-down parsing.
- Its major drawback is the difficulty it has in coping with situations in which the actual error has occurred before the point of detection.

Error productions

- If we have a good idea of the common errors that might be encountered, we can augment the grammar for the language at hand with *error productions* that generate the erroneous constructs.
- We then use the grammar augmented by these error productions to construct a parser.
- If an error production is used by the parser, we can generate appropriate error diagnostics to indicate the erroneous construct that has been recognized in the input.

Global correction

- Ideally, we would like a compiler to make as few changes as possible in processing an incorrect input string.
- There are algorithms for choosing a minimal sequence of changes to obtain a globally least-cost correction.
- Given an incorrect input string *x* and grammar *G,* these algorithms will find a parse tree for a related string *y,* such that the number of insertions, deletions, and changes of tokens required to transform *x* into *y* is as small as possible.

Global correction - Evaluation

- Unfortunately, these methods are in general too costly to implement in terms of time and space, so these techniques are currently only of theoretical interest.
- We should point out that a closest correct program may not be what the programmer had in mind.
- Nevertheless, the notion of least-cost correction does provide a yardstick for evaluating error-recovery techniques, and it has been used for finding optimal replacement strings for phrase-level recovery.