LR Error Recovery

An LR parser will detect an error when it consults the parsing action table and find a blank or error entry. Errors are never detected by consulting the goto table. An LR parser will detect an error as soon as there is no valid continuation for the portion of the input thus far scanned. A canonical LR parser will not make even a single reduction before announcing the error. SLR and LALR parsers may make several reductions before detecting an error, but they will never shift an erroneous input symbol onto the stack.

Panic-mode Error Recovery

We can implement panic-mode error recovery by scanning down the stack until a state $s$ with a goto on a particular nonterminal $A$ is found. Zero or more input symbols are then discarded until a symbol $a$ is found that can legitimately follow $A$. The parser then stacks the state GOTO($s$, $A$) and resumes normal parsing. The situation might exist where there is more than one choice for the nonterminal $A$. Normally these would be nonterminals representing major program pieces, e.g. an expression, a statement, or a block. For example, if $A$ is the nonterminal $stmt$, a might be semicolon or $\}$, which marks the end of a statement sequence.

This method of error recovery attempts to eliminate the phrase containing the syntactic error. The parser determines that a string derivable from $A$ contains an error. Part of that string has already been processed, and the result of this processing is a sequence of states on top of the stack. The remainder of the string is still in the input, and the parser attempts to skip over the remainder of this string by looking for a symbol on the input that can legitimately follow $A$. By removing states from the stack, skipping over the input, and pushing GOTO($s$, $A$) on the stack, the parser pretends that if has found an instance of $A$ and resumes normal parsing.

Phrase-level Recovery

Phrase-level recovery is implemented by examining each error entry in the LR action table and deciding on the basis of language usage the most likely programmer error that would give rise to that error. An appropriate recovery procedure can then be constructed; presumably the top of the stack and/or first input symbol would be modified in a way deemed appropriate for each error entry.

In designing specific error-handling routines for an LR parser, we can fill in each blank entry in the action field with a pointer to an error routine that will take the appropriate action selected by the compiler designer. The actions may include insertion or deletion of symbols from the stack or the input or both, or alteration and transposition of input symbols. We must make our choices so that the LR parser will not get into an infinite loop. A safe strategy will assure that at least one input symbol will be removed or shifted eventually, or that the stack will eventually shrink if the end of the input has been reached. Popping a stack state that covers a nonterminal should be avoided, because this modification eliminates from the stack a construct that has already been successfully parsed.