Subject: MCR10153, Enhancements for the reductions Command (version 3.0)

Author: Gary Dixon

Date: 2025-06-08

Multics Ticket: http://multics-trac.swenson.org/ticket/370

Input source to the reductions (rdc) command must be parsed into tokens that can be analyzed by the reduction_compiler_.rd subroutine. At the time this MCR is published, only the lex_string_.pl1 subroutine has been available for this parsing. It parses an input string into character-string tokens separated by whitespace or break characters. Each token is described by a token structure declared in lex_descriptors_.incl.pl1.

A new parser in being proposed as part of enhancements to the ssu_ subsystem. ssu_$get_tokens will parse an input string into tokens that describe several higher-level constructs:

- character tokens (similar to those of lex_string_).
- pl1 quoted character-string tokens surrounded by double-quote (") characters.
- pl1 quoted bit-string tokens surrounded by double-quote (") characters and ending with a radix suffix (b or b1, b2, b3, b4) specifying digits within quotes to be binary, quaternary, octal or hexadecimal.
- lists of tokens coming in four flavors.
    - parenthesized list surrounded by left/right parenthesis characters: (...)
    - square-bracket list surrounded by left/right bracket characters: [...]
    - braced list surrounded by left/right brace characters: { ... }
    - angle-bracket list surrounded by less-than (<) and greater-than (>) characters: < ... >
- parent/child pointers forming a 2-dimensional threading for tokens found within a list.

ssu_$get_tokens will generate a threaded list of token structures as declared in ssu_token.incl.pl1. These token structures have almost the same elements as those produced by lex_string_.

The two varieties of token structure will have the same storage footprint (same length in words); the same layout for shared elements describing start and length of the described character string and its line number in the source string, etc. Structures from ssu_$get_tokens will add elements beyond those of lex_string_ by using formerly reserved spaces within the token structure to hold these new attributes. They could add storage at the end of the structure to hold new attributes. The only requirements are that elements shared by lex_string_ have the attributes, length, and location within the token structure specified by the token declaration in lex_descriptors_.incl.pl1. This includes using the same mechanisms for threads between adjacent token structures.

While the reductions command and its reduction_compiler_.rd subroutine will treat all tokens as if they were generated by lex_string_, an input source (XXX.rd) using ssu_$get_tokens needs to reference the token structure described by ssu_token.incl.pl1. To support this dual overlay for the token data within a single XXX.rd source, the XXX.rd program needs to be divided into two sections.

- The top part defines the reduction statements and provides syntax and semantic subroutines named in those reductions. All code in the top part will reference the token structure as declared in ssu_token.incl.pl1.

- The SEMANTIC_ANALYSIS subroutine generated by the reductions compiler will reference those same tokens using the token structure as declared in lex_descriptors_.incl.pl1. All rdc_XXX_.incl.pl1 segments that depend upon this token structure will be visible only within the scope of the SEMANTIC_ANALYSIS procedure.

The reductions command needs to be told whether to use the original structure for an analyzer (referencing only lex_descriptors_.incl.pl1) or this new, two-part structure. A new Reduction Attribute Statement will provide this information.

```
VERSION n \
```

where n is 2 if the analyzer uses tokens from lex_string_; or 3 if the analyzer uses tokens from ssu_$get_tokens or some other parser. 2 is the default version assumed if no VERSION statement is found.

By knowing the version, the reductions command can produce a slightly different source when it generates the SEMANTIC_ANALYSIS subroutine to analyze incoming tokens. For VERSION 2, it will produce the same code generated now for the analyzer.

For VERSION 3, it will produce code in which the lex_descriptor_.incl.pl1 segment is visible only within the scope of the SEMANTIC_ANALYSIS subroutine. The top part of the XXX.rd source program can have its own declaration of the token structure (via use of an %include ssu_token; macro or some other declaration) to reference the tokens provided by its chosen token parser. Its syntax and semantic routines can then access the extra elements described in the new style of token.

Additional Fixes

While enhancing the reductions command, several minor problems should be repaired.

FIX: The rdc_XXX_.incl.pl1 segments have header comments that use underlined words for emphasis. 1970s computer terminals and printers properly handled such underlined text. Today's video terminals only display them as a sequence of characters separated by backspace (BS) characters.

A. The use of underlined words should be removed from comments in the rdc_XXX_.incl.pl1 segments to make those comments more legible on video terminals.

FIX: The version 1 pl1 compiler did not support use of a %page; macro to separate sections of the .pl1 source program from one another. Today's version 2 pl1 does support this macro.

B. Change both reduction_compiler_.rd and the various rdc_XXX_.incl.pl1 segments it includes to use a %page; macro to insert page breaks. These would replace use of a new-page (\014) character in the .pl1 source lines within the generated analyzer.

FIX: The reductions command translates an XXX.rd (describing a new language) into an XXX.pl1 source program that can analyze statements in the new language. It outputs a header in front of the XXX.rd code. This header summarizes the version of the reductions command that generated the analyzer, date and time of generation, etc. Including such header followed conventions of existing translators of the time by documenting these translation details at the top of the .list file.

However, the reductions command does not generate its own .list segments. It just inserts a PL/I comment string atop the user-supplied XXX.rd text and lets the pl1 command include the comment when it generates a .list if one is requested.

Unfortunately, this extra comment at the top of the XXX.pl1 source means that any problems found by the pl1 compiler in the XXX.pl1 lines are reported with a line number that does not match position of the offending source line in the XXX.rd segment.

The shift in line numbers within pl1 error statements can be avoided by adding a feature to the reductions command.

C. If the XXX.rd segment begins with 10 blank lines (10 consecutive newline (NL) characters), then the reductions command will replace those 10 blank lines with its PL/I comment documenting reductions version, date/time of translation, etc. As a result of this new feature, all line numbers will be the same in both XXX.rd (original source) and the XXX.pl1 (derived source) segments.

FIX: An ambiguity exists in control arguments accepted by the reductions command. Its existing -trace {on|off} control argument controls addition of a trace facility to the generated language analyzer. Tracing is either turned on by default or turned off by default.

Because the reductions command uses its own reduction statements to analyze each input reduction statement in the XXX.rd source program, it is unclear whether -trace refers to tracing reductions used by the rdc command; or to adding a trace facility to the new analyzer being generated. Some way is needed to turn on/off tracing of the rdc reductions while executing the reductions command.

D. Add new -tracing and -no_tracing control arguments to the reductions command to control tracing of its reductions while it is translating an XXX.rd source. The existing -trace {on|off} will continue specify whether to add a tracing facility to the generated analyzer program.

Proposed Changes

The enhancements described above affect every component of the reductions command and its subsystem of include segments, and info segment. The exact changes are summarized below.

reduction_compiler_.rd

RC1) Add a VERSION reduction attribute to distinguish between all earlier translators produced by reductions (VERSION 2); and a new generation of translators using ssu_$get_tokens (or other parsers) to tokenize inputs (VERSION 3). VERSION 2 is the default if no VERSION attribute is given.

RC2) Change generated code to provide new calling sequence for the SEMANTIC_ANALYSIS subroutine in VERSION 3 analyzers.

```
call SEMANTIC_ANALYSIS();                          /* for VERSION 2 */
```

becomes:

```
call SEMANTIC_ANALYSIS( tokenP, stmtP, comment );  /* for VERSION 3 */
```

where tokenP is set by the caller to point to the first token structure to be analyzed. While the SEMANTIC_ANALYSIS subroutine is running, tokenP is updated to point to the current token being examined by the analyzer. That permits the syntax and semantic routines provided by the top part of XXX.rd to have access to the current token structure so any extended attributes are available to those analyzer helpers.

RC3) Add a new control bit (supplied by reductions command) to set the TRACING control variables used by the reductions command itself.

-trace {on|off}
controls whether a trace facility is added to the new translator generated by reductions command.

-tracing, -no_tracing
enables setting of TRACING shared variable to control display of the reductions command's matched reductions as they generate the new analyzer.

RC4) Replace use of new-page (\014) chars in generated translator.pl1 by a %page; macro (now supported by v2pl1 compiler).

reduction_compiler.pl1

R1) Add -tracing and -no_tracing control arguments to the reductions command.

rdc_start_3_.incl.pl1

RI1) Add new include file derived from rdc_start_.incl.pl1 to support VERSION 3 calling sequence for SEMANTIC_ANALYSIS subroutine. rdc_start_.incl.pl1 will continue to be used for VERSION 2 translators.

RI2) Correct header comments to remove use of underlines for emphasis. Underlined text does not display properly on today's video terminals.

RI3) Replace use of new-page (\014) char with %page; macro.

All other rdc_xxx_.incl.pl1

Rx1) Correct header comment to remove use of underlined text for emphasis. Underlines do not display properly on today's video terminals.

Rx2) Replace use of new-page (\014) char with %page; macro.

reductions.info

R1) Change reductions.info to describe -tracing and -no_tracing.

R2) Correct wording and diagrams in several sections of this info seg. They have gotten mis-formatted over time.

R3) Add new sections:

Notes on the translation process

Notes on parsing a source string into tokens

Notes on analyzing with the lex_string_ parser

Notes on analyzing with the ssu_$get_tokens parser

- Document new SEMANTIC_ANALYSIS calling sequence in VERSION 3 translators.

Notes on compiling a reduction analyzer

Notes on the reduction language

The full list of segments being changed is shown below.

Changes --

```
 Bound_obj:    bound_misc_translatrs_          IN: tools  UPDATE;
   source:     reduction_compiler.pl1           REPLACE  compiler: pl1 -ot;
   source:     reduction_compiler_.rd           REPLACE  compiler: rdc
                                                          -ot -trace off;

 Include:      rdc_delete_.incl.pl1            IN: sss.include  REPLACE;
 Include:      rdc_delete_stmt_.incl.pl1       IN: sss.include  REPLACE;
 Include:      rdc_end_.incl.pl1               IN: sss.include  REPLACE;
 Include:      rdc_error_.incl.pl1             IN: sss.include  REPLACE;
 Include:      rdc_lex_.incl.pl1               IN: sss.include  REPLACE;
 Include:      rdc_next_stmt_.incl.pl1         IN: sss.include  REPLACE;
 Include:      rdc_stack_fcns_.incl.pl1        IN: sss.include  REPLACE;
 Include:      rdc_start_.incl.pl1             IN: sss.include  REPLACE;
 Include:      rdc_start_3_.incl.pl1           IN: sss.include  ADD;
 Include:      rdc_tracing_fcns_.incl.pl1      IN: sss.include  REPLACE;

 Info:         reductions.info                 IN: sss.info  REPLACE;
                add_name:  rdc.info;
```

Testing

With introduction of the reduction VERSION attribute, testing of the changes in this proposal was divided into two parts:

1. Ensure that the reductions command operating on existing translator source programs (all of which are VERSION 2) continues to generate analyzer code with the SEMANTIC_ANALYSIS() interface and the %include lex_descriptors_; macro being visible throughout the analyzer. The generated analyzer should be comparable to the analyzer produced by the currently-installed reductions command (reductions version 2.5).

2. Ensure that the reductions command operating on one of the new VERSION 3 source programs generates a two part analyzer: top part includes its own declaration for the token structure, and calls the version 3 SEMANTIC_ANALYSIS(tokenP, stmtP, commentP) interface; bottom part limits all visibility to the lex_descriptors_.incl.pl1 references of token to the scope of that new SEMANTIC_ANALYSIS subroutine so there are no conflicting references to elements of the token structure in either part of the analyzer.

Part 1 testing was performed by re-translating code for reduction_compiler_.rd source using first the installed reductions 2.5 translator; then an uninstalled copy of the reductions 3.0 command. The reduction_compiler_.pl1 generated by each version were then compared (using compare_ascii) to ensure that nothing had been changed (other than applying the “additional fixes” mentioned in the change proposal).

Tests were performed using the 2.5 version of reduction_compiler_.rd (the installed source) being translated by both reductions 2.5 and reductions 3.0. The two outputs compared as expected (with only the “additional fixes” being made).

The installed reduction_compiler_.rd code does not have any VERSION statement and so depends upon VERSION 2 being assumed as the default. The 3.0 version of reduction_compiler_.rd includes a VERSION 2 statement. The reductions 3.0 command should generate the same out from both the 2.5 version of reduction_compiler_.rd (without a VERSION statement) and the 3.0 version (with a VERSION 2 statement).

Part 2 testing was performed using a new implementation of the cv_ptr_ function. Code in the existing cv_ptr_.pl1 function manually divides the VIRTUAL_PTR argument into segname and offset components; then further splits the offset component into either an entry_pt name, or a WORDNO(BITNO) piece.

Code in the new version of cv_ptr_ adds support for an optional [RINGNO] following the entry_pt name, or as part of an enhanced WORDNO(BITNO)[RINGNO] offset. The new functionality requires support for many additional VIRTUAL_PTR formats. The new cv_ptr_ will therefore be implemented as cv_ptr_.rd so it can use the proposed new ssu_$get_tokens to split VIRTUAL_PTR into tokens; and then use reduction statements to identify and validate the supported formats for the enhanced VIRTUAL_PTR.

Treating the new cv_ptr_.rd as a VERSION 2 analyzer does not work. pl1 generates many errors stemming from two competing declarations of the token structure that are visible throughout the entire scope of the cv_ptr_.rd source. The effort to resolve these errors led to the changes proposed in this MCR.

Therefore, the part 2 testing focuses on ensuring that cv_ptr_.rd which includes a VERSION 3 reduction statement properly splits the cv_ptr_.rd source into a top section that sees only the token structure generated by ssu_$get_tokens; while the new SEMANTIC_ANALYSIS part of the code sees only the token structure declared in lex_descriptors_.incl.pl1.

This split into two parts was successfully accomplished by the VERSION 3 changes to the reduction_compiler_.rd source. All pl1 errors stemming from duplicate declaration of the token structure no longer occur. The bottom part overlays the token storage with the lex_descriptors_.incl.pl1 declaration. The top part overlays the token storage with the ssu_token.incl.pl1 declaration.

Part 3 testing was added to ensure that the "additional fixes" were implemented correctly: the adverse symptoms of each defect are no longer present.

- No underlined words in the comment text.
- No appearance of new-page (\014) character in the generated analyzer code, or in the included rdc_XXX_.incl.pl1 segments.
- If the XXX.rd segment begins with 10 or more consecutive new-line (NL) characters, then any error messages about statements in the XXX.rd program refer to correct line numbers which are the same in both XXX.rd (original source) and XXX.pl1 (derived source) modules.
- Correct implementation of the new -tracing and -no_tracing control arguments of the reductions 3.0 command: -no_tracing is the default. Use of -tracing causes reductions from reduction_compiler_.rd to be displayed as input source matches the syntax specification in any reduction.

All part 3 testing was successful.

No collision arose between purposes of the existing -trace {on|off} control with those of the new -tracing and -no_tracing controls. Addition of a trace facility to the generated analyzer is now totally separate from enabling tracing during execution of the reductions command.

Documentation

The reductions.info segment will change as summarized below.

```
      A) Add documentation for the -tracing and -no_tracing control
         arguments.
      B) Correct the Notes on the reduction language.
      C) Add Notes on translating, parsing, analyzing, compiling.
```

Because of the extent of changes, it is best to review the entire contents of the revised info segment. This is shown below.

```
>udd>m>gd>w>rdc>reductions.info   (350 lines in info)

2025-06-05  reductions, rdc

Syntax as a command:  rdc PATH {-control_args}

Function:
The reductions command generates a language translator.  It translates
a segment containing reduction statements, syntax routines, and action
routines into a PL/I source segment; it then invokes the pl1 compiler
to compile that PL/I source.

The reduction statements specify the syntax and semantics of a new
language; the action routines perform the basic operations (e.g,
generating code) required to translate the language.  For more
information, see the description of the reductions command in the
Commands manual.

Arguments:
PATH
   is the pathname of a translator source segment that is to be
   compiled by rdc.  If PATH does not have a suffix of rd, one is
   assumed; however the rd suffix must be the last component of the
   name of the source segment.

Control arguments:
   In addition to the arguments below, you can give any control
   argument accepted by the pl1 command.
-brief, -bf
   prints all error messages with only a brief summary of the error
   that has occurred.
-long, -lg
   prints all error messages with a detailed description of the error
   that has occurred.  If you supply neither -brief nor -long, a
   detailed description is printed the first time an error occurs in a
   given compilation and a brief description is printed in subsequent
   occurrences of that error.
```

-no_trace
   generates a translator which does not have its own tracing facility.
   (default)
-trace {STR}
   adds a tracing facility to the generated translator. STR defines
   whether tracing is enabled or disabled by default. It can be "on"
   (default) or "off." (See "Notes on tracing" below.)

-tracing
   traces tokens matching with syntax specifications of a reduction
   statement as the reductions command generates an analyzer from the
   reduction statements found in PATH.
-no_tracing
   does not trace the work as the reductions command generates an
   analyzer from source PATH. (default)

Notes: Reductions are expressed in a highly compact form that emphasizes the syntax and semantics of the new language. This command compiles these reductions into tables that drive an rdc-provided semantic analyzer for the language. The analyzer compares incoming tokens (basic units) of a program written in the new language with the valid token phrases defined in the reductions. When a valid phrase is found, the action routines defined by the reduction are invoked to translate the phrase.

Translators generated by the command can be written more quickly than hand-programmed translators because rdc provides the semantic analyzer for the language. They are easier to understand and to maintain because the all-important language syntax and semantics is concentrated in the reductions, rather than being spread throughout the semantic analyzer.

This command can generate translators for the simplest type of language, a right-linear (finite state automaton) language. Often such languages are composed of keywords with operands, such as the control language of the bind command.

The organization of an rdc translator, the translation process, and the reduction language are described below.

Notes on the translation process:
A translator for a given language must perform three steps to translate a source string written in the language.

1. Parse the source string into a list of tokens. These tokens are the basic units of the language being translated. They are character strings in the source language that are separated from one another by language-defined delimiter characters: whitespace or break characters.

2. Analyze the syntax of the tokens to identify groups of tokens (token phrases) that are valid or invalid in the language.

3. Assign some semantic meaning to each valid token phrase by performing a translation action. Print an error message diagnosing each invalid token phrase.

Notes on parsing a source string into tokens:
Parsing is a process that depends upon: the types of units that make up the language; the delimiter characters defined by the language; and other language characteristics. For most languages, the lex_string_ subroutine provides facilities that are sufficient to parse the language. However, some languages have syntax characteristics that cannot be handled by lex_string_; your code must call a special parsing routine for such languages.

lex_string_ generates a token structure declared in the include segment: lex_descriptors_.incl.pl1. The reductions translation mechanism depends upon this token structure. See the Subroutines manual for a description of the lex_string_ subroutine and token structure.

The ssu_$get_tokens subroutine provide an alternate parser for a language source string. It parses the source into different source units declared by a token structure in the ssu_token.incl.pl1 include segment. Tokens generated by ssu_$get_tokens have similar elements to those generated by lex_string_, plus some additional elements describing aggregate source units: character and bit strings; and lists surrounded by parenthesis (...), angle-brackets <...>, square-brackets [...], or braces {...}. For details, type: help ssu_$get_tokens.

Notes on analyzing with the lex_string_ parser:
The default parser is lex_string_$lex. A reduction attribute statement may be given to notify the reductions command that lex_string_ is in use:

```
VERSION 2 \
```

However, version 2 is assumed by default if no VERSION statement is found in the reductions.

The analyzer generated by the reductions command is invoked without arguments, as described in Figure 3-1 of the reductions description in the Commands manual:

```
Pthis_token = Pfirst_token_from_lex_string_;

call SEMANTIC_ANALYSIS();
```

Notes on analyzing with the ssu_$get_tokens parser:
Your translator can use the alternate ssu_$get_tokens parser by giving the reduction attribute statement:

```
VERSION 3 \
```

This tells the reductions command to generate a token analyzer that is invoked with three arguments pointing to token and stmt structures declared in the ssu_token.incl.pl1 segment.

```
dcl  commentP  ptr;

tokenP = first_tokenP_from_ssu_;

call SEMANTIC_ANALYSIS( tokenP, stmtP, commentP );
```

The tokenP and stmtP pointers are declared in the ssu_token.incl.pl1 segment. They point to the token structure and its stmt structure (if any) for the current token being analyzed. Syntax routines can examine attributes of the current token structure; and that token's string value (tokenStr).

Code in a version 3 analyzer can access additional elements in token and stmt as declared in ssu_token.incl.pl1 which describe the token.flavor, token.S attributes describing aggregate values; and token.parentP and token.childP pointers.

The ssu_$get_tokens parser does not support a comment structure, so the commentP pointer must be declared before calling the version 3 SEMANTIC_ANALYSIS subroutine. Elements of the token and stmt structures used by the reduction analyzer have identical location and attributes in both lex_descriptors_.incl.pl1 and ssu_token.incl.pl1, so the analyzer treats both versions like version 2 structures.

Notes on compiling a reduction analyzer:
The reduction analyzer (translator) generated by the reductions command is compiled in a two-step process as shown in Figure 3-2 of the reductions description in the Commands manual.

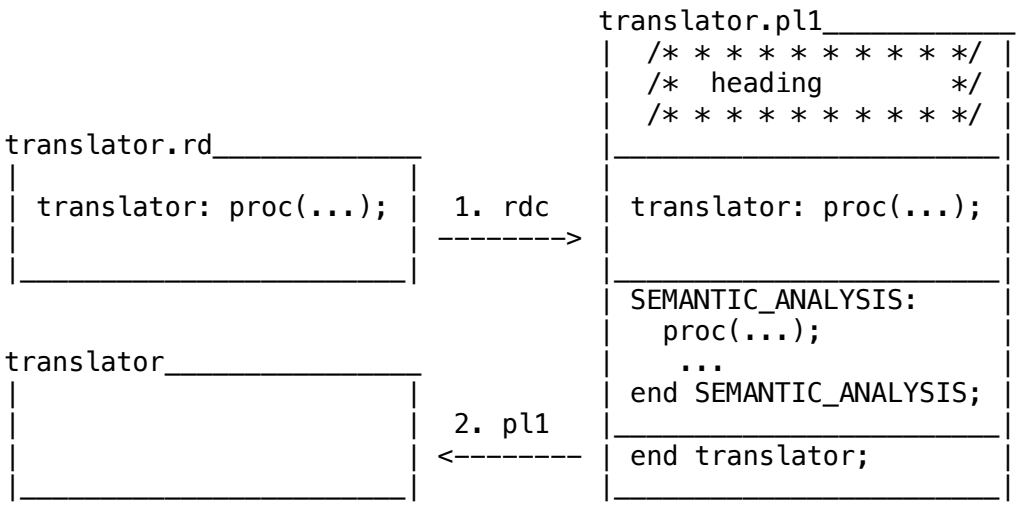
1. The reductions command compiles the reduction statements in translator.rd segment into PL/I analyzer source code saved in an intermediate translator.pl1 segment.

2. The reductions command then invokes the pl1 command to compile translator.pl1 into an object segment called translator which is placed in the working directory.

The step one output to translator.pl1 contains the following items.

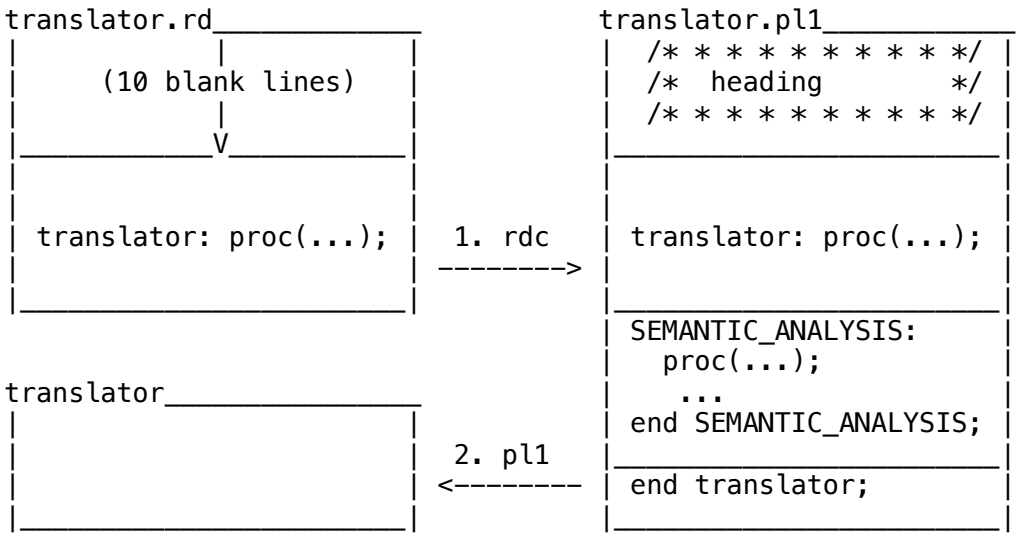
A. It begins with a heading comment describing compile date and reductions compiler version which compiled the reductions.
B. The entire contents of translator.rd source.
C. PL/I code for the analyzer.
D. An end statement for the PL/I translator source code.

These are summarized by the abbreviated Figure 3-2 shown below.



In step two, if the pl1 command reports errors found during its compile of translator.pl1, the line numbers reported for those errors will differ from line numbers in the translator.rd source. That difference stems from length of the heading comment add at the beginning of translator.pl1. The heading comment is 10 lines long.

This difference may be avoided by starting the translator.rd segment with 10 empty lines (lines containing only a newline character). If the rdc command finds 10 consecutive newline characters at the start of translator.rd, it will replace those 10 newlines with the heading comment. Thus, line numbers will be equivalent for errors found in code shared by both translator.rd and translator.pl1.



Notes on tracing:
When the reductions command generates a language translator, the -trace control argument can add a tracing facility to the translator. As that new translator is used to translate statements in the new language, this added tracing facility can be turned on to aid in debugging the reduction statements for the new language.

Each input statement in the new language is split into tokens. If the optional tracing facility is enabled (set TRACING = "1"b), when one of the translator's reduction statements has a syntax specification matching those tokens, that reduction gets displayed along with the matched token strings. This displays which sequence of reductions identify token phrases in statements of the new language, how it assigns a semantic meaning to those tokens.

Because the tracing facility generates large amounts of output, it can be selectively turned on and off during a debugging session by setting a variable declared as follows:

```
dcl TRACING bit(1) aligned int static init("1"b);
```

A value of "1"b turns tracing on; "0"b turns it off. The initial value is set by the operand of the -trace control argument when the tracing facility was added to the translator.

The generated translator can accept a control argument to turn TRACING on or off; or it may have a debugging entry point to set the switch; or the switch can be set at a particular probe breakpoint. For example, to trace from the 28th through the 40th reductions, you can use the following set of probe requests:

```
probe translator
ps "RD_ACTION(NRED)"
go to RD_ACTION(NRED);
b: if NRED = 28: halt
```

When halted, type--

```
let TRACING = "1"b
reset
b: if NRED = 40: h
c
```

And when the 40th reduction was reached, type:

```
let TRACING = "0"b
reset
c
```

Notes on the reduction language:
The statements that define the syntax and semantics of a language to be translated are written in the reduction language. This translator generation language consists of two kinds of statements:

- attribute declaration statements; followed by:
- reduction statements.

Attribute declarations control the size of some fixed-length tables that the generated translator uses; or they cause translation action routines provided by the reductions command to be included in the translator.

Reduction statements specify the syntax of token phrases in the language being translated. They also name action routines that are invoked to translate valid phrases and to diagnose invalid token phrases.

The diagram below shows where the various elements of the reduction language fit within the rdc statements of a translator program. This sample is not intended to be a meaningful translator. It only demonstrates in which type or part of a statement each element might appear.

Consult the Commands manual to see reductions for a simple translator program.

Reduction language statements appear as a PL/I comment near the beginning of your translator source. The special /*++ delimiter begins the reduction program; it ends with a ++*/ comment delimiter.

The \" comment delimiter begins a comment, which ends with a newline character ending that line. Comments may occupy an entire line as shown below; or only the last part of a line containing parts of a reduction statement.

Each reduction statement ends with a back-slash (\) character.

```
/*++
   VERSION 2 \
   MAX_DEPTH label_stack_depth_number \
   INCLUDE NEXT_STMT \
   INCLUDE ERROR \
   INCLUDE LEX \
   INCLUDE DELETE \
   INCLUDE DELETE_STMT \

\" labels     syntax           actions         next reduction
\" -------    ------------     -----------     --------------

   BEGIN      / absolute_spec  / semant(...)   / label      \
              / <relative_fcn> / [var="1"b]    /            \
   label
   label2     /                /               /            \
              / <no-token>     / LEX           / RETURN     \
              / <any-token>    / LEX(n)        / STACK      \
              / <name>         / NEXT_STMT     / STACK_POP  \
              / <decimal-integer>
                               / DELETE        /            \
              / <quoted-string>
                               / DELETE(n)     /            \

              / <BS>           / DELETE(m,n)   /            \
              /                / DELETE_STMT   /            \
              /                / ERROR(n)      /            \
              /                / PUSH(label)   /            \
              /                / POP           /            \
  ++*/
```

Version History

Date	Revision	Author	Comment
2025-06-08	1.0	Gary Dixon	Initial draft of this MCR.
2025-06-09	1.1	Gary Dixon	Fix typographical errors.