IBM REXX COMPILER

BERT MOSER
IBM
IBM REXX Compiler

Bert Moser

IBM Vienna Software Development Lab
Wien 1, Cobdengasse 2

c/o IBM Austria
Obere Donaustrasse 95
A-1020 Austria
EUROPE

MOSER@VABVM1.IINUS1.IBM.COM

(+431) 21145-4476

May/91
Overview

Past

REXX and Interpreters

Present

CMS REXX Compiler COMPLEMENTS SPI

Future

REXX Compiler Improvements and Requirements
REXX History

'79 Mike Cowlishaw becomes father of REXX

'83 Command Language for IBMs VM/CMS
'87 SAA Procedures Language
'89 REXX supported on MVS
'90 REXX supported on OS/2 and OS/400

2/89 IBM announces the CMS REXX Compiler

Available since 7/89

Developed by IBM Vienna Software Development Lab
Based on IBM Israel Scientific Center's feasibility study
- Compilability of REXX
- Appropriate run-time performance improvements

11/89 Library becomes a separate product
- REXX initially implemented by Interpreters
  - Excellent debugging features
  - Very short and appealing edit/run cycle

- HOWEVER
  - Better performance desirable
How does a REXX Interpreter work

Single Step Approach

- Everything needs to be done at run-time
- On every REXX program invocation
- REXX source must be made available to every user
How does the REXX Compiler work

Two Step Approach

Compile

Run
Compiled REXX Program

### Compiled Code

- Executable /370 instructions
  - Reentrant and relocatable

- Invocations of Library routines
- References to static symbols resolved

### Symbol Tree

- Descriptors for all static symbols
- Upwards and downwards connected
- Symbols identifiable by their name

### Control Blocks

- Run-time required
- Pre-allocated and pre-initialized
Forms of Compiled REXX Programs

EXEC-type

Same behavior as interpreted - "transparently" replace

- Same way of invocation and search sequence
- EXECLOADable
- Shared segment capability

Module-type

Other HLL compilers' object format (ESD, RLD, TXT,..)

- Linkable to other object programs
- Need to be LOADed - can be GEN'd into a module
- Search order is different
- CMS restriction: SVC-type arg/parmlist (PLIST)
Product Components

Compiler

Set of phases performing all compilation tasks

- Compiled with IBM SAA C/370 Compiler
- Prerequisite when compiling: IBM C/370 Library V1 (5668-039) or later

Run-time Library

Routines invoked from compiled REXX programs

- Common to every compiled program
  Initialization, Termination, ...
- Too bulky as to be copied to every program
  String Arithmetic, Conversions, Comparisons, Built-in Functions, Compound Variable Access/Handling, ...
- Extremely time critical $\rightarrow$ written in Assembler
Usage Scenarios

Compiler/Interpreter COMPLEMENT Each Other

Existing REXX Programs

Existing REXX Programs

Newly Developed REXX Programs

Newly Developed REXX Programs
Compiled REXX - Benefits

Significantly Faster than Interpreted

"Plug-compatible" with Interpreted

Language Equivalent to Interpreter

"Unreadable" REXX Programs

Comprehensive Program Documentation
Compilied REXX - Benefits ...

Comprehensive REXX Program Documentation

Source and Cross-Reference Listings
Syntax check whole program
More accurate messages
Begin debugging with more-correct programs

- Improve program quality
- Increase developer productivity

"Unreadable" Compiled REXX

Executable /370 code

- Provide program integrity
- Improved maintainability
- Protect REXX-coded assets
SAMPLE EXEC G1

CMS REXX COMPILER 1.1.0 TIME: 11:35:02 DATE: 30 May 1989

IF DO SEL LINE C +1+2+3+4+5

/* SAMPLE incorrect REXX program */
1
2 Parse Arg Tmp
3 val. = TRANSLATE(tmp)
4 line.2 = LEFT(val.,2,'40')

+++EAGGA00771S Invalid or missing argument(s) on built-in function
6 $ = EXTFUNC(line.2)
7 Call INTFUNC 2
8 Exit
9
10 INTFUNK: Procedure Expose x. i
11 Signal on NOVALUE NAME my_value

+++EAGGA00072S Label not found
12 Do x. i
13 If x. i//2 /= 0 then
14 say "Odd: " x. i
15 End
16 Return
17
18 my_valu: Say "NOVALUE raised at: " sigl
19 "Return
20 /* end of program SAMPLE

+++EAGGA00654S Unmatched "/**"
## Cross Reference List Example

<table>
<thead>
<tr>
<th>ITEM</th>
<th>ATTRIBUTE</th>
<th>LINE REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LABELS, BUILT-IN FUNCTIONS, EXTERNAL RTNS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXTFUNC</td>
<td></td>
<td>EXT RTN 6</td>
</tr>
<tr>
<td>INTFUNC</td>
<td></td>
<td>EXT RTN 7</td>
</tr>
<tr>
<td>INTFUNK</td>
<td></td>
<td>LABEL 10(d)</td>
</tr>
<tr>
<td>LEFT</td>
<td></td>
<td>BUILT-IN 5</td>
</tr>
<tr>
<td>MY VALU</td>
<td></td>
<td>LABEL 18(d)</td>
</tr>
<tr>
<td>MY_VALUE</td>
<td></td>
<td>LABEL+++ 11(u)</td>
</tr>
<tr>
<td>TRÄNSLATE</td>
<td></td>
<td>BUILT-IN 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONSTANTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'NOVALUE raised at: '</td>
<td></td>
<td>LIT STR 18</td>
</tr>
<tr>
<td>'Odd: '</td>
<td></td>
<td>LIT STR 14</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>NUMBER 13</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>CONST SYM 5 6</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>NUMBER 5 7 13</td>
</tr>
<tr>
<td>'40'</td>
<td></td>
<td>LIT STR 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMPLE VARIABLES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
<td>SIMP VAR 6(s)</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>SIMP VAR 10 12 13 14</td>
</tr>
<tr>
<td>SIGL</td>
<td></td>
<td>SIMP VAR 18</td>
</tr>
<tr>
<td>TMP</td>
<td></td>
<td>SIMP VAR 3(s) 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STEMS AND COMPOUND VARIABLES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINE.2</td>
<td></td>
<td>COMP VAR 5(s) 6</td>
</tr>
<tr>
<td>VAL.</td>
<td></td>
<td>STEM 4(s) 5</td>
</tr>
<tr>
<td>X.I</td>
<td></td>
<td>COMP VAR 10 12 13 14</td>
</tr>
</tbody>
</table>
Compiled REXX - Benefits ...

Language Equivalence with REXX Interpreter

NO compiler-specific language features!

- Minimize migration effort
- Almost all REXX programs run unchanged
  - except those with INTERPRET instructions

Flag Non-SAA Items - optional

Support SAA Procedures Language level 1.1

- Ease programming for multiple SAA environments
"Plug-Compatibility" with Interpreted Programs

Identical external interfaces - invocation and use

- "Transparency" replace interpreted
- No restriction on mutual invocation

31-Bit Capability

Compiler, Library, and Compiled Code run and use storage above the 16 Mega-byte line

- Make room for others below the line
Why is REXX Hard to Compile

No Conventional Block Structure

PROCEDURE is an executable instruction
- Not a syntactic boundary

Variables' life-time is dynamic
- Depends on calling sequence
- "Exposure" among procedures

No denotation of the END of a procedure
- Logical end is an executed RETURN

SIGNAL

Control can be transferred to everywhere
- Even into "procedure" and loop bodies

Computed GOTO - SIGNAL VALUE
Why is REXX Hard to Compile ...

**No data types**

All data is "character string"

Sometimes contents must be "numeric",  
- Whole number", or "Boolean"

**No declarations**

Variables come and go - EXECCOMM/DROP

Can be shared with external programs

Names of variables can be computed  
- Tails of compound variables

Value of variables only limited by storage  
- Storage for values must be allocated dynamically

Arithmetic precision can be set dynamically  
- NUMERIC DIGITS
Performance gains depend on program mix

<table>
<thead>
<tr>
<th>Programs with a lot of ...</th>
<th>TIMES faster than Interpreter</th>
<th>Performance Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default-precision Arithmetic</td>
<td>6 - 10+</td>
<td>VERY HIGH</td>
</tr>
<tr>
<td>String Arithmetic</td>
<td>4 - 25</td>
<td></td>
</tr>
<tr>
<td>Assignments</td>
<td>6 - 10</td>
<td></td>
</tr>
<tr>
<td>Changes to Variables' Values</td>
<td>4 - 6</td>
<td>HIGH</td>
</tr>
<tr>
<td>Constants and Simple Variables</td>
<td>4 - 6</td>
<td></td>
</tr>
<tr>
<td>Reuse of Compound Variables</td>
<td>2 - 4</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>Host Commands</td>
<td>1 -</td>
<td>LOW</td>
</tr>
</tbody>
</table>

- Up to 30% CPU load reduction reported - "... better than last CPU upgrade"
- On average 10% - 15%
- Savings example
  
  Interpreted program runs 60 times a day (12 sec's)
  using 12 min's CPU
  assume improvement of 6 times
  runs compiled 2 min's
INTERPRET Instruction not Supported

Rarely used

- Compiler diagnoses - no code generated

- Try to avoid

  Interpret target' = 'expr
  Call SETVAR target,expr
  RXSETVAR sample Assembler program
  User's Guide & Reference SH19-8120

- Restructure the program

  Isolate interpretative part
  Make it a separate program, and
  Let the Interpreter handle it
**TRACE Instruction and Function not Supported**

Does not change the semantics of a REXX program

- No need to change REXX program

  TRACE instruction - NOP instruction
  TRACE built-in function - "O"
  Interpreter default - "N"

- Diagnosed with an informational message
Compiled REXX Programs

Save CPU Time & Reduce System Load

Improve Program Quality

Increase Developer Productivity

Protect REXX-Coded Assets

Allow to Keep Applications in REXX

Save Expensive Rewrites to Other HLL's

Attract to Write Even More REXX Applications
Improvements

Reduce Storage Needed at Compile-Time

Improve Compiler’s Performance

Improve Access/Handling of Compound Vars
- Binary tails
- Faster algorithms

Improve Built-in Functions
Compiled REXX Increases Paging

Interpreted

1 2 3 4 5 6

Compiled

1 2 3 4 5 6

Running compiled on an I/O bound systems makes it WORSE

Compiled REXX Scatters Storage

Sorry for this one - was a bug
4 Bytes of a control block left over - sometimes

Compiled REXX Needs More Storage when Run

NO
Both implementations show similar storage consumption
Reduce Disk Space Needed by Compiled REXX

Remember: Code + Symbol Tree + Control Blocks

- Improve Assignment
  Special casing by Compiler = lot of code
  - Trade-off between performance and storage
  - Move case distinction to Library

- Compress Compiled Output
  Compiler option
  - Reduce med/large to size of source
  - Automatically de-compress
  - Reduce expensive I/O
Requirements ...

Static Binding

- Allow to link external subroutines and functions

For Module-type output only

Compiler option
Dualism - resolved address/dynamic invocation

Tie together REXX-written application
Function-package capability

Tailorable Cross Reference Listing

- Make Xref of CONSTANTS and LITERALs optional
- Compiler option XREF(S)
Requirements

Library as "Test" Shared Segment

- Test new Library in parallel with "production" version

Relax Restriction on INTERPRET

- SEVERE ERROR - NO code generated
- Diagnose as ERROR - produce code

Code should raise ERROR when executed
  - Allow to program around
  - Parse Version & REXXC370

- Support development of multi-environmental programs
Requirements ...

Implement INTERPRET

Long Range Consideration

Provide an MVS REXX Compiler

Accepted
MVS Compiler - Requirements

- Same language level as TSO/E Interpreter

- Same external interfaces - invocation and run

- Similar behavior and benefits as CMS REXX Compiler

- Cross compile?
  - REXX code could run unchanged in VM and MVS
  - No need to re-compile

- Support MVS Parameter List Conventions
  - EFPL for external functions
  - CPPL for TSO/E commands
  - MVS JCL parameters for batch programs
  - CALL command parameters for foreground programs