FINAL REPORT - TASK AREA I
(Volume IV)
(21 June 1968 - 31 December 1970)
FOR THE PROJECT
RESEARCH IN MACHINE-INDEPENDENT
SOFTWARE PROGRAMMING

Massachusetts
COMPUTER ASSOCIATES
division of
APPLIED DATA RESEARCH, INC.

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
Springfield, Va. 22151
APPLIED DATA RESEARCH, INC.
LAKESIDE OFFICE PARK, WAKEFIELD, MASSACHUSETTS 01880 - (617) 245-9540

FINAL REPORT - TASK AREA I
(Volume IV)
(21 June 1968 - 31 December 1970)
FOR THE PROJECT
RESEARCH IN MACHINE-INDEPENDENT
SOFTWARE PROGRAMMING

Principal Investigators:
Task Area I Carlos Christenson (617) 245-9540
Task Area II Anatol W. Holt (617) 245-9540

Project Manager:
Robert E. Millstein (617) 245-9540

ARPA Order Number - ARPA 1228
Program Code Number - 8D30

Contractor: Massachusetts Computer Associates, Inc., Division of ADR
Contract No.: DAHC04-68-C-0043
Effective Date: 21 June 1968
Expiration Date: 30 September 1971
Amount: $696,800.00

Sponsored by
Advanced Research Projects Agency
ARPA Order Number - ARPA 1228

CA-7102-2614
A REPORT ON AMBIT/G
(Volume IV)

by
Carlos Christenson
Michael S. Wolfberg
Michael J. Fischer*

CA-7102-2614
February 26, 1971

* Consultant to Applied Data Research, Inc.
Address: Department of Mathematics, M.I.T.,
Cambridge, Massachusetts

This is the fourth of four volumes of the final report on Task Area 1
of the project "Research in Machine-Independent Software Programming".
This research was sponsored by the Advanced Research Projects
Agency of the Department of Defense and was monitored by U.S. Army
Research Office-Durham, Box CM, Duke Station, Durham, North
Carolina 27706, under Contract DAHC04-68-C-0043.
## CONTENTS

### Volume I

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Summary</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>Fundamentals</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>data graph, constraints, program, general philosophy, specific languages.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>Representation of Programs</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>overview, program syntax, correspondence between program graphs and diagrams.</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>The Interpreter</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>overview, the compiler, interpretation of 'linkrep's, user-defined functions, error messages.</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>The Loader</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>overview, error messages, loader syntax, sample encoding, sample error.</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>Initialization and the Built-in System</td>
<td>65</td>
</tr>
<tr>
<td>7.</td>
<td>The Debugging Facility</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>lexical conventions, statements, statement forms, sample session.</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>The Implementation</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td>credits and acknowledgements, internal view, files, PL/I data formats, PL/I implementation of the interpreter and loader.</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>Further Work</td>
<td>152</td>
</tr>
<tr>
<td>10.</td>
<td>Project Bibliography</td>
<td>157</td>
</tr>
</tbody>
</table>

### Volume II

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.</td>
<td>Examples of AMBIT/G Programs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>observations, introductory examples: reversing a list, two forms of input, function calling, LISP garbage collector, another garbage collector, an interactive program, sorting, factorial computation and recursion.</td>
<td></td>
</tr>
</tbody>
</table>

### Volume III

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.</td>
<td>The AMBIT/G Interpreter as an AMBIT/G Program</td>
<td></td>
</tr>
<tr>
<td></td>
<td>description, listing.</td>
<td></td>
</tr>
</tbody>
</table>

### Volume IV

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.</td>
<td>The AMBIT/G Loader as an AMBIT/G Program</td>
<td></td>
</tr>
<tr>
<td></td>
<td>description, listing.</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 13

THE AMBIT/G LOADER AS AN AMBIT/G PROGRAM

This final volume of "A Report on AMBIT/G" describes the AMBIT/G loader as an AMBIT/G program, which is, in fact, how it is implemented. We begin by giving a verbal description of the loader. Then each function of the loader is briefly discussed. The remainder of the volume consists of many pages of diagrams which constitute the listing of the loader.

The reader who wants to understand the program (rather than just look at pretty pictures) is urged to be familiar with the contents of Chapter 5 of this report in Volume I. Although that chapter attempted to describe completely the syntax of loader input, including lexical conventions, it is the listing of the loader which can be used as a final reference for obscure issues.

A table of contents for the listing is given at the beginning of the listing.
The AMBIT/G loader consists of the function 'load' and nine other major functions. In addition, the loader contains calls on the 'eq_any' function which determines whether its first argument is the same node as any of its other arguments; its result is 'boolean true' if it is the same and 'boolean false' otherwise. The listing of the loader does not include the 'eq_any' function definition. The functions 'tail_l', 'head_l', and 'head_2' are used to transmit arguments and results, and these functions are also omitted from the listing. One additional kind of omission in the listing is the write calls on built-ins 'read_function' and 'write_function' to define the functions of the loader and the links which it uses.

Execution of the loader begins by a trivial initialization and a call on 'get_statement'. The first statement must begin with a minus sign and must be a '-page-' statement or a '-start-' statement. If a '-start-' statement is found, the remainder of the statement is used to locate a 'rule' node to be returned as the result of the 'load' function, and control returns to the caller. Otherwise, the processing of a page begins.

The loader looks for a next node by calling 'get_statement', and if that statement begins with a minus sign it checks for the following possibilities: '-rule-', '-ruleref-', '-links-', '-start-', or '-page-'. Finding either '-rule-' or '-ruleref-' causes the loader to treat the specified rule or rule reference as a glorified node. The '-links-' statement causes the loader to start to look for a next link specification for the current loader page. A '-start-' or '-page-' statement causes the loader to finish up processing of the current page, and then that statement is appropriately interpreted.

When seeking a next node, if a statement does not begin with a minus sign it is taken as a specification of a data node on the loader page. It must have a page-name and type, and optionally a subname.

During the reading of nodes of a loader page, a 'pname' list is developed as a symbol table for the duration of the page. The list consists of nodes of type 'pname' connected by 'next' links. The list is logically terminated by a 'pname' link of a 'pname' node pointing to 'charconn end' which represents
a null page-name. Each entry of this list is represented by a 'pname' link pointing to a list of 'char' nodes connected by 'charconn's and a 'node' link off of the same 'pname' node pointing to the very node represented by the 'pname'.

The physical 'pname' list grows by locating another 'pname' node at the end of the list when necessary, but it never shrinks. The physical end is represented by a 'pname' node's 'next' link pointing to 'pname end'. At the end of processing a loader page the logical 'pname' list is made empty, but the physical 'pname' list is not affected.

When a '-rule-' statement is encountered, the logical 'pname' list grows by one entry for the 'rule' node but is then frozen for the duration of the rule specification until an '-endrule-' statement is detected. The page-names used within a rule are local to the rule and are saved by the loader on the physical 'pname' list immediately following the frozen list. The 'node' link of each 'pname' node used to represent a node in a .ale points at a 'noderep' node since each node mentioned in a rule is represented by a 'noderep' node. The 'rep' link of such a 'noderep' representing a fully named node points to that node, and the 'variability' link of that 'noderep' points to 'flag fixed'. If a rule contains a particular fully named node in more than one place, different entries are used on the 'pname' list, but they all share one 'noderep' node. This merging of split fully named nodes also is done for link names and for any implicitly mentioned nodes, such as those required in an expansion of a node in a rule with a type but no subname. The 'pname' link of 'pname' nodes representing these implicitly specified fully named nodes points to 'charconn free' as a special indicator.

During the processing of a rule, there are six types of statements which may be encountered while seeking a next node:

a) fully named node
b) node with type, but no subname
c) node with type to be tested, but no subname
d) unnamed node
e) ' --- ' statement
f) '-endrule-' statement
Any node specification causes an entry to be placed on the 'pname' list. In case b or g an extra link is pushed down on the 'contents' list of links hanging off the 'rule' node which represents the rule being loaded. In either case, its name is 'link type' and its destination is the appropriate 'type' node. For case b its mode is 'frame', and for case g its mode is 'test'. At this point, no other links are represented on the 'contents' list.

Unnamed nodes cannot be split in a rule since there is no syntax for this.

The '---' statement signals the end of node specifications and causes the loader to expect link specifications for the rule.

If an '-endrule-' statement is encountered, the rule contains no links; the temporary end of the 'pname' list is eliminated, and the loader proceeds to seek a next node on the loader page.

During the processing of links of a rule, each statement specifies one link whose representation is constructed and pushed down on the 'contents' list of links hanging off the 'rule' node which represents the rule being loaded. A link which is specified by a textual name (rather than an explicit spur to a node) constitutes an implicit use of a fully named node and thus may introduce another entry on the 'pname' list. The only other sort of statement expected during the processing of links of a rule is '-endrule-', whose effects were already described.

When a '-ruleref-' is encountered by the loader when it is seeking a next node, an entry is added to the 'pname' list for the referenced rule, but no links of the 'rule' node representing the rule are modified.

When the loader looks for a next link on a loader page, it calls 'get_statement' and if that statement begins with a minus sign it checks for the following possibilities: '-deflinks-', '-start-', or '-page-'. A '-deflinks-' statement causes a write call on the built-in 'read_function' and a write call on the built-in 'write_function' for each link to be defined. If the statement does not begin with a minus sign it specifies the setting of a link. This is done by the loader's performing a write call on the built-in 'link'.

Throughout the operation of the loader, 'charconn' nodes are constantly being taken from and placed back on the free list of 'charconn' nodes. This list was initially created by the initializer with a reasonable subset (100) of the 'charconn's. When the loader seeks a next 'charconn' from the free list it calls the 'next_charconn' function which may have to locate another 'charconn' if the list is empty. After processing many types of statements, after processing a rule, and after processing a page 'charconn' nodes are returned to the free list.

This description of the loader as a program will conclude with individual descriptions of the nine major functions of the loader.

get_statement

This function is responsible for reading one character at a time from the input source file and gathering together a loader statement as the result of the function. It tallies each statement in the counter 'circle line'. It ignores (but tallies) comment statements. It converts unprotected special characters into nodes of type 'special'. It is concerned mostly with spaces, tabs, dollar signs, semicolons, new-lines, and special characters. Leading and trailing spaces and tabs are ignored. Spaces and tabs are not distinguished, and any amount of space in the middle of a statement is converted to one space (the node 'special SP'). Protective dollar signs are removed, and line continuations are processed. The following state diagram is the best description of this function; the AMBIT/G code closely corresponds to the diagram.
Any node specification causes an entry to be placed on the ‘pname’ list. In case h or g an extra link is pushed down on the ‘contents’ list of links hanging off the ‘rule’ node which represents the rule being loaded. In either case, its name is ‘link type’ and its destination is the appropriate ‘type’ node. For case h its mode is ‘frame’, and for case g its mode is ‘text’. At this point, no other links are represented on the ‘contents’ list.

Unnamed nodes cannot be split in a rule since there is no syntax for this.

The ‘---’ statement signals the end of node specifications and causes the loader to expect link specifications for the rule.

If an ‘--endrule’- statement is encountered, the rule contains no links; the temporary end of the ‘pname’ list is eliminated, and the loader proceeds to seek a next node on the loader page.

During the processing of links of a rule, each statement specifies one link whose representation is constructed and pushed down on the ‘contents’ list of links hanging off the ‘rule’ node which represents the rule being loaded. A link which is specified by a textual name (rather than an explicit use to a node) constitutes an implicit use of a fully named node and thus may introduce another entry on the ‘pname’ list. The only other sort of statement expected during the processing of links of a rule is ‘--endrule’-, whose effects were already described.

When a ‘--ruleset’- is encountered by the loader when it is seeking a next node, an entry is added to the ‘pname’ list for the referenced rule, but no links of the ‘rule’ node representing the rule are modified.

When the loader looks for a next link on a loader page, it calls ‘get_statement’, and if that statement begins with a minus sign it checks for the following possibilities: ‘--deflinks’, ‘--start’, or ‘--page’. A ‘--deflinks’ statement causes a write call on the built-in ‘read_function’ and a write call on the built-in ‘write_function’ for each link to be defined. If the statement does not begin with a minus sign it specifies the setting of a link. This is done by the loader’s performing a write call on the built-in ‘link’.

Throughout the operation of the loader, ‘charconn’ nodes are constantly being taken from and placed back on the free list of ‘charconn’ nodes. This list was initially created by the initializer with a reasonable subset (106) of the ‘charconn’‘s. When the loader seeks a next ‘charconn’ from the free list it calls the ‘next_charconn’ function which may have to locate another ‘charconn’ if the list is empty. After processing many types of statements, after processing a rule, and after processing a page ‘charconn’ nodes are returned to the free list.

This description of the loader as a program will conclude with individual descriptions of the three major functions of the loader.

get_statement

This function is responsible for reading one character at a time from the input source file and gathering together a loader statement as the result of the function. It tallies each statement in the counter ‘circle line’. It ignores (but tallies) comment statements. It converts unprotected special characters into nodes of type ‘special’. It is concerned mostly with spaces, tabs, dollar signs, semicolons, new-lines, and special characters. Leading and trailing spaces and tabs are ignored. Spaces and tabs are not distinguished, and any amount of space in the middle of a statement is converted to one space (the node ‘special SP’). Protective dollar signs are removed, and line continuations are processed. The following state diagram is the best description of this function: the AMBIT3 code closely corresponds to the diagram.
get_word

This function has no argument, but it assumes that 'circle char' points to a partial statement. Its first result is a pointer to the 'charconn' node pointed to by 'circle char'. Its second result is the last 'charconn' node in the list of 'charconn's whose 'value' link points to a 'char' node. Thus the two results point to the beginning and end of the next word in the statement. The reason for the scan to stop is either of the following:

a) 'charconn end' is encountered which indicates the end of the statement; then 'circle special' is made to point at 'charconn end'.

b) a 'charconn' node is encountered whose 'value' link points to a 'special' node; then 'circle special' is made to point to that 'special' node.

In either case, the 'next' link of the 'charconn' node which is the second result is made to point to 'charconn end'.

This function considers an error condition to hold if either 'circle char' points to an empty list or if the first element of the list is not a 'char' node.

find_node

This function has no argument, but it calls on 'get_word' to read the next word of the current statement. It then looks down the 'pname' list for a match of a page-name with the word. If it does find a match, it returns as its result the node represented by the given page-name; this is found at the destination of the 'node' link of the 'pname' node whose 'pname' link points to the given page-name. If a match is not found, an error condition is reported.
**define_node**

This function has one argument which is a 'type' node. That is used along with the word at 'circle char' to locate a node which is then pointed to by 'circle node'. Then an entry is added to the 'pname' list with a page-name as the word pointed to by 'circle pname' and its definition as the node pointed to by 'circle node'.

**find_noderep**

This function returns either an existing 'noderep' in the 'pname' list or a freshly inserted one which is a node representation of the given argument. Thus this function sometimes serves to define a fully named node of a rule.

**get_link_name**

This function scans the current statement (using 'circle char') for a link name specified either explicitly as a spur or implicitly as a text string. It has no argument, but its result is the node which names the link.

**heads_or_tails**

This function processes both the heads and the tails of a link in a rule. It has no argument, but scans the current statement using 'circle char'. Its result is a list of 'diamond' nodes where each 'diamond'"s 'value' link points to a 'noderep'.

**free_charconn_list**

This function returns the list of 'charconn' nodes given as its argument to the free list of 'charconn's. The function has no result.
This function takes an argument which is a node of type 'charconn'. It returns a next 'charconn' node connected to the argument by its 'next' link. Either that next node was already there, or the function locates such a node.

THE LISTING

The remainder of this volume consists of the listing of the AMBIT/G loader. The following is a table of contents for the listing.

<table>
<thead>
<tr>
<th>Function</th>
<th>Loader Page Titles</th>
<th>Starting Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>loader</td>
<td>L1 - L27</td>
<td>10</td>
</tr>
<tr>
<td>get_statement</td>
<td>GS1 - GS8</td>
<td>41</td>
</tr>
<tr>
<td>get_word</td>
<td>GW1 - GW2</td>
<td>51</td>
</tr>
<tr>
<td>find_node</td>
<td>FN1 - FN3</td>
<td>53</td>
</tr>
<tr>
<td>define_node</td>
<td>DN1 - DN2</td>
<td>56</td>
</tr>
<tr>
<td>find_noderep</td>
<td>FNR1 - FNR3</td>
<td>58</td>
</tr>
<tr>
<td>get_link_name</td>
<td>GLN1 - GLN2</td>
<td>61</td>
</tr>
<tr>
<td>heads_or_tails</td>
<td>HT1 - HT3</td>
<td>63</td>
</tr>
<tr>
<td>free_charconn_list</td>
<td>FCL1</td>
<td>66</td>
</tr>
<tr>
<td>next_charconn</td>
<td>NC1</td>
<td>67</td>
</tr>
</tbody>
</table>
The first statement read by the loader does not begin with a "-".
"..." is an unknown statement.
A type-name is missing on a node on a loader page.
(cont' on next page)
"..." is an unknown statement.
A page-name is missing for a "-rule-" or "-ruleref-" on a loader page.
begin_rule_3

define_node

rule

rule

save_pname

pname_list

next_pname

char

pname

pname

S

F

node

clear

state

contents

end

next_rule_node

char

get_statement

charconn

char

charconn

char

minus_in_rule_nodes

F

S
rule_node_pname

pname

next_pname

node

charconn

get_word

charconn

rule_node_type
There is an unknown "-" line in a rule contents on a loader page.
rule_link_dest

node

heads_or_tails

next rule link

char

charconn

free_charconn_list

end rule

char

charconn

free_charconn_list

end_rule

pname_list

next_pname

pname

end_rule_1
next_link_3

get_link_name

name

find_node

dest

char

charconn

free_charconn_list

link'

org

name

dest

next_link

35
(Diagram of put_char function)

prev_char

charconn

next_charconn

charconn

char

read_normal

44
read_normal

char

char D

char SP

char HT

eq_any

boolean false

check_normal

char

end_line

char

char ;

eq_any

boolean false

read_space

done_statement

read_normal 1
There is an incomplete statement on a loader page.

"..." is an extra special character on a loader page.
get_word_3

head_2 \rightarrow charconn \rightarrow charconn \rightarrow charconn \rightarrow charconn \rightarrow char conn \rightarrow special \rightarrow char conn \rightarrow ret

F \rightarrow S

char

charconn \rightarrow charconn \rightarrow charconn \rightarrow char

F

char

charconn \rightarrow char conn \rightarrow char conn \rightarrow char conn \rightarrow char conn \rightarrow char conn \rightarrow special

ret
"..." is an undeclared page-name on a loader page.
check_char_2

char_a  char_b  find_pname

charconn

end

charconn

pname

node

find1

charconn

head_1

find2

charconn

charconn

free

charconn

ret

S

not_matched

F
define_node

tail_1
char
charconn
locate

next_pname

F
define_node_there

S

next_pname

pname

locate

pname

pname

pname

pname

pname

end

end
define_node_there

```
define_node_there
```

```
next_pname

pname

node

charconn

free_charconn_list

char

 pname

node

charconn

define_node_there
```
find_noderep_there

next_pname

pname

node

charconn
free

head_1

rep

find2

locate

fixed

noderep

variability

charconn
end

ret
heads_or_tails

\[
\begin{array}{c}
\text{heads or tails}
\end{array}
\]
next_charconn

tail_1 -> charconn
head_1 -> charconn
charconn -> charconn
charconn -> charconn
charconn end -> boolean
false

locate

charconn

charconn

charconn end

S -> ret

F