ART/Ada Design Project - Phase I
Final Report

Inference Corporation

March 1989

Cooperative Agreement NCC 9-16
Research Activity No. SE.19

NASA Johnson Space Center
Information Systems Directorate
Information Technology Division

Research Institute for Computing and Information Systems
University of Houston - Clear Lake
The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC’s main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the “gateway” concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.
ART/Ada Design Project - Phase I
Final Report
Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Inference Corporation. Dr. Charles McKay served as RICIS research coordinator.

Funding has been provided by the Information Systems Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Robert T. Savely, of the Software Technology Branch, Information Technology Division, Information Systems Directorate, NASA/JSC.

The views and conclusions contained in this report are those of the author and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.
ART/Ada Design Project - Phase I

Final Report

Subcontract 015
RICIS Research Activity SE.19
NASA Cooperative Agreement NCC-9-16

March 1989

Inference Corporation
5300 W. Century Blvd.
Los Angeles, CA 90045

Copyright © 1989 Inference Corporation
# Table of Contents

1. Introduction .......................... 1
2. Project Goal .......................... 2
   2.1 Conventional Expert System Tools 2
   2.2 Considerations for Ada Environment 3
   2.3 Requirements for Real-Time Embedded Systems  3
3. Current Status ........................ 5
   3.1 Overview .......................... 5
   3.2 Introduction to ART-IM .......... 5
   3.3 ART-IM Ada Deployment Compiler 7
   3.4 ART/Ada Run-time System ....... 8
   3.5 Deployment in Ada Environment 8
4. Prototype Development Process ...... 10
   4.1 Specification ........................ 10
   4.2 Object-Oriented Design .......... 10
   4.3 Implementation .................... 11
   4.4 Reuse ................................ 11
   4.5 Testing ................................ 12
   4.6 Debugging ........................... 12
   4.7 Summary ............................ 13
5. Performance Analysis ............... 14
   5.1 Speed ................................ 15
   5.2 Size ................................ 15
   5.3 Discussion .......................... 16
6. Related Work .......................... 17
7. Future Work ........................... 19
   7.1 Summary ............................ 20
8. Conclusion ............................ 21
References ............................... 22

I. Detailed Description of ART/Ada Implementation 23
   1.1 Deploying an ART-IM application in Ada 23
      1.1.1 Ada Source Code Generated by the Ada Deployment Compiler 24
      1.1.2 ART/Ada User Interface Command Loop 25
      1.1.3 Example Main Programs 25
   1.2 Public Packages in ART/Ada 27
      1.2.1 ART 27
      1.2.2 ERROR_HDL_SUB 28
      1.2.3 USER_INTERFACE_SUB 28
   1.3 Ada Call-In and Call-Out Specification for ART/Ada and ART-IM 29
      1.3.1 Interface Types 29
      1.3.2 Scope of Objects 29
      1.3.3 Call-Out from ART to Ada 30
      1.3.4 Call-In from Ada to ART/Ada 32
II. Difference between ART-IM 1.5 and ART/Ada 1.0 33
III. ART/Ada Public Packages ................ 34
   III.1 Specification of ART Package 35
   III.2 Specification of ERROR_HDL_SUB Package 35
III.3 Body of ERROR_HDL_SUB Package
III.4 Separate Procedure ERROR_HDL_SUB.PROCESS_ERROR
III.5 Separate Procedure ERROR_HDL_SUB.WARNING
III.6 Specification of USER_INTERFACE Package

IV. Benchmark Programs
   IV.1 Monkey and Banana
   IV.2 N-Queens

V. Test Programs
   V.1 Sweeptop
   V.2 Sweep2
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Composite ART-IM Screen</td>
<td>6</td>
</tr>
<tr>
<td>3-2</td>
<td>ART-IM Ada Deployment Compiler</td>
<td>7</td>
</tr>
<tr>
<td>3-3</td>
<td>Ada Deployment Process</td>
<td>9</td>
</tr>
<tr>
<td>4-1</td>
<td>Spiral Life Cycle of the ART/Ada 1.0 prototype</td>
<td>13</td>
</tr>
<tr>
<td>7-1</td>
<td>Proposed Spiral Life Cycle of the ART/Ada 1.5 prototype</td>
<td>19</td>
</tr>
</tbody>
</table>
List of Tables

Table 5-1: Speed of ART/Ada in Rules/Second
Table 5-2: Size of ART/Ada in KBytes
1. Introduction

Under subcontract to University of Houston - Clear Lake as part of the Cooperative Agreement between UHCL and NASA Johnson Space Center. Inference Corporation conducted an Ada-Based Expert System Building Tool Design Research Project. The goal of the research project was to investigate various issues in the context of the design of an Ada-based expert systems building tool. The research project attempted to achieve a comprehensive understanding of the potential for embedding expert systems in Ada systems, for eventual application in future projects.

This report will describe the current status of the project by introducing an operational prototype, ART/Ada. It will then explain how the project was conducted, analyze the performance of the prototype, compare it with other related works, and suggest future research directions.
2. Project Goal

This chapter identifies the goal of the Ada-Based Expert System Building Tool Design Research Project.

This chapter is composed of three sections:

- Conventional Expert System Tools
- Considerations for Ada Environment
- Requirements for the Real-Time Embedded Systems

As the Department of Defense mandate to standardize on Ada as the language for embedded software systems development begins to be actively enforced, interest from developers of large-scale Ada systems in making expert systems technology readily available in Ada environments has increased.

Two examples of Ada applications that can benefit from the use of expert systems are monitoring and control systems and decision support systems. Monitoring and control systems demand real-time performance, small execution images, tight integration with other applications, and limited demands on processor resources; decision support systems have somewhat less stringent requirements. An example project which exhibits the need for both of these types of systems is NASA's Space Station Freedom. Monitoring and control systems that will perform fault detection, isolation and reconfiguration for various on-board systems are expected to be developed and deployed on the station either in its initial operating configuration or as the station evolves; decision support systems that will provide assistance in activities such as crew-time scheduling and failure mode analysis are also under consideration. These systems will be expected to run reliably on a standard data processor, currently envisioned as a 1-16 megabyte 386-based workstation. The Station is typical of the large Ada software development projects that will require expert systems in the 1990's.

2.1 Conventional Expert System Tools

Inference Corporation developed an expert system tool called Automated Reasoning Tool (ART) which has been commercially available for several years [8]. ART is written in Lisp and it supports various reasoning facilities such as rules, frames, truth maintenance, hypothetical reasoning, and object-oriented programming.

More recently, Inference introduced another expert system tool called ART-IM (Automated Reasoning Tool for Information Management) which is also commercially available [9]. ART-IM is written in C and it supports a subset of ART's reasoning facilities; ART-IM Version 1.5 supports forward-chaining rules, frames, truth maintenance, and basic object-oriented programming.
Both ART and ART-IM have been successfully used to develop many applications which are in daily use today.

2.2 Considerations for Ada Environment

This research project permitted Inference to study how to bring the ART and ART-IM features into Ada environments. Inference's approach in designing an Ada-based expert system tool is to use an existing architecture such as ART or ART-IM so that its input language would be identical to that of an existing tool.

Two approaches to implementing the existing architecture were considered:

1. To implement the whole system in Ada
2. To implement an Ada deployment compiler as part of an existing tool

Since the purpose of the project was to research operational issues such as those discussed below, it was decided to take approach number 2.

2.3 Requirements for Real-Time Embedded Systems

Laffey et al. identified potential problems in using conventional expert system tools for real-time applications [Laffey88b]. Many of these problems are already solved by ART and ART-IM.

ART and ART-IM provide features for embedded environments such as

- They have facilities for handling asynchronous inputs.
- They have a standard call-in and call-out interface for various languages.
- They assign priorities to rules which can be used to focus attention on important events
- They have interrupt capabilities.
- They can run continuously even if there is no rule to fire.

Among the problems that Laffey et al. identified, ART and ART-IM do not address the following problems:

- Guaranteeing response times
- Temporal reasoning capability

Both ART and ART-IM are based on the Rete algorithm [5]. Laffey et al. claim that in real-time applications, the knowledge base changes too rapidly for the Rete algorithm to be optimal [11]. Barachini
et. al. claim that an expert system tool based on the Rete Algorithm can be optimized to better support real-time expert systems [2]. The reported speed of their system, however, does not seem much faster than that of other C-based tools such as CLIPS or ART-IM. Some tools specialized in the monitoring and analysis applications do not use the Rete algorithm; they use a compiled, static knowledge base in which all variables used in the rules are resolved at compile time [12] [10]. While the speed of these tools seems faster than that of the Rete-based tools, these tools still cannot guarantee response times. The main drawback is that they may not be suitable for expert system application areas other than monitoring and analysis. An alternative approach to the compiled, static knowledge base is an object-oriented programming (OOP) facility that uses active values. The OOP facility is already implemented in ART, and it is being considered for ART-IM and ART/Ada.

A temporal reasoning capability refers to a way to invoke a rule at a regular time interval. For example, the following is a temporally driven rule:

Check the price of IBM stocks every hour.
If the price goes down more than five dollars in an hour, then sell all shares.

The temporal reasoning capability can be directly supported, or it could be implemented as an Ada task outside of the inference engine. The task should be started during the initialization phase of the expert system as a demon that wakes up at a certain time interval. A built-in temporal reasoning capability, therefore, may not be as important as the issue of guaranteed response times.

There are two different levels of real-time requirements: soft real-time and hard real-time [12]. In a hard real-time system, the correctness of the system depends not only on the result of computation, but also on the time at which the results are produced. If these strict timing constraints are not met, the consequence may be disastrous. On the other hand, in a soft real-time system, disastrous consequences do not result if the dead-line is missed. While most expert system tools try to address the soft real-time issue today by improving their performance, they do not, yet, address the hard real-time issue.

The current generation embedded processors such as the MIL-STD-1750A have limited addressing capability of 2 megabytes. Medium size Ada-based expert systems may not fit into this limitation. It is also known that the size of the Ada-based expert systems is larger than that of C-based counterparts. While next-generation embedded processors such as the 80386 would alleviate the size problem, it would still be desirable to study how to optimize the size of the Ada-based inference engine.
3. Current Status

This chapter discusses the status of a prototype Ada-based expert system building tool, called ART/Ada Version 1.0.

3.1 Overview

A primary goal of this research phase was to design an expert system tool that allowed applications to be deployed in Ada environments. In order to achieve this goal, three components were needed:

- Existing expert system tool as a baseline
- Application generator that generates Ada code
- Ada-based inference engine

After ART and ART-IM were reviewed carefully, ART-IM was selected as a baseline system because C was much closer to Ada than Lisp. ART/Ada, an Ada-based inference engine, was modeled after that of ART-IM Version 1.0 which supported forward chaining rules.

ART-IM has a deployment compiler that converts an application into C data structure definitions. An Ada deployment compiler was designed using the C deployment compiler as a model. The Ada deployment compiler converts C data structures specific to an application into Ada source code that would be used to initialize Ada data structures equivalent to the original C data structures. The Ada deployment compiler is written in C and is part of ART-IM.

3.2 Introduction to ART-IM

ART-IM is a general purpose expert system tool written in C. ART-IM version 1.0 implements a forward-chaining rule-based inference engine using on the Rete algorithm. It also has a truth maintenance system, called Logical Dependency

ART-IM version 1.5 includes a frame system, called Schema system, which is fully integrated with the rule system. It also includes an explanation system, called Justifications. In addition, ART-IM version 1.5 for MS-DOS has a presentation manager style user interface, called Studio, that provides extensive capabilities for debugging ART-IM applications. An example of a Studio screen is shown in Figure 3-1.

The ART-IM syntax is basically a subset of the ART syntax. An application written in ART, therefore, can be easily ported to ART-IM.
Figure 3-1: Composite ART-DM Screen

Figure Key:

1 - Menu Bar
1a - Menu Name
1b - Menu Name Mnemonic
1c - Highlighted Menu Name

2 - Interaction Window
2a - Display Area
2b - Command Area

3 - Status Line

4 - Menu
4a - Menu Item (Command)
4b - Menu Item Mnemonic
4c - Menu Item (Extended Command)
4d - Highlighted Menu Item
4e - Accelerator Key Designator
4f - Menu Separator

5 - Browser Dialog Box (Object-Type)
5a - List Box Control
5b - Highlighted List Box Item
5c - Vertical Scroll Bar
5d - Scroll Bar Elevator
5e - Push Button Controls
5f - Push Button Mnemonic
5g - Highlighted Push Button

6 - Browser Dialog Box (Individual-Object)
6a - Selectable Text Control
6b - Horizontal Scroll Bar

7 - Prompting Dialog Box
7a - Edit Control

8 - Message Box
3.3 ART-IM Ada Deployment Compiler

ART-IM Version 1.5 was augmented with an Ada Deployment Compiler to support the ART/Ada runtime system. As shown in figure 3-2, its input is an ART-IM source file, and its output is Ada source files. At any point after an ART-IM source file is loaded into ART-IM and reset, it can be invoked to generate the Ada source code that will be used to initialize ART internal data structures for the ART/Ada runtime system. The ART-IM program can be run up to any given point before the code generation takes place.

Since it is part of ART-IM, the Ada deployment compiler is written in C. In addition to generating Ada source code that represents the knowledge base, it also generates a call-out interface module that can be used to call user-defined Ada functions. ART-IM provides a powerful call-out specification language that can be used to call out from ART-IM or from ART/Ada to Ada.

![Figure 3-2: ART-IM Ada Deployment Compiler](image-url)
3.4 ART/Ada Run-time System

The ART/Ada run-time system is composed of the following components:

- Inference engine
- Procedural Interface Package
- Memory management package
- Ada deployment compiler utilities
- User interface package

ART/Ada's inference engine is based on the Rete algorithm, and supports only forward chaining rules matching on facts as specified in the ART-IM Version 1.0 syntax.

ART/Ada supports a simplified version of the procedural language of ART-IM. ART/Ada's procedural interface can be used either in the rule right-hand side, or directly in user's Ada programs. The procedural interface includes data type conversions between the Ada data types and the ART data types, predicates, operations on ART objects, ART commands, I/O functions, and math functions. ART/Ada's I/O system supports simple input and output functions. Unlike ART-IM, streams are not supported in ART/Ada. All streams variables default to either standard output or input. File I/O is not supported in ART/Ada. ART/Ada's math package provides most mathematical functions except trigonometric functions.

ART/Ada's memory management package uses Ada the features new and unchecked_deallocation to allocate and deallocate memory. In phase II, the advantages and disadvantages of implementing a memory manager for ART/Ada will be investigated.

The ART/Ada run-time system contains utilities called by the Ada code that ART-IM Ada deployment compiler generates.

ART/Ada has an optional simple command interface that support rudimentary debugging features such as tracing/untracing rules, facts, activations, printing out facts and agenda, and running the program.

3.5 Deployment in Ada Environment

As shown in Figure 3-3, the following steps are needed to deploy an ART-IM application in Ada:

1. Develop an application in ART-IM using ART-IM's development environment.

2. If necessary, call out to Ada from ART-IM using the standard callout mechanism for both ART-IM and ART/Ada.

4. Compile the generated Ada code.

5. Link it with an Ada library of the ART-Ada runtime system and user's Ada code if any.

6. Deploy the Ada executable image on a host computer or on a target system.

Figure 3-3: Ada Deployment Process
4. Prototype Development Process

This chapter discusses how the ART-Ada Version 1.0 prototype was developed.

4.1 Specification

ART-IM Version 1.0 was used as a functional specification for the ART-Ada Version 1.0 prototype. Some features in ART-IM 1.0 (e.g., Logical Dependency) that were not essential for the proof of concept were left out intentionally.

The ART-IM C source code was used as a detailed design specification for the ART/Ada run-time system. The ART-IM internal function definitions and data structures were converted to Ada package specifications which were compiled by an Ada compiler. The Ada package specification served as a detailed design specification of ART/Ada.

4.2 Object-Oriented Design

ART/Ada was designed using the object-oriented design (OOD) methodology. The object-oriented design is an approach to software design in which the system is decomposed into a set of objects. Each object is mapped to one or more Ada packages. Four different kinds of packages were used in the design:

- Abstract Data Object (ADO)
- Abstract Data Type (ADT)
- Package of subroutines (SUB)
- Package of declarations (DCL)

The Abstract Data Object is a package that contains encapsulated data and operations (expressed as subprograms) performed upon those data. These data are static and local to that package. They are known as state data.

The Abstract Data Type is a package that contains abstract types and operations performed on those abstract types. The operations are expressed as subprograms and the abstract types are declared as the Ada types.

The package of subroutines is a package of logically related subroutines. There exists no encapsulated data in this package.

The package of declarations is a package of logically related declarations. These declarations may be types, constants, or exceptions.
4.3 Implementation

Once package specifications for the Ada run-time system were laid out, the packages were divided among two programmers to be implemented. Again, the ART-IM C source code was used as a program design language (PDL). Despite the differences between C and Ada, it was relatively easy to port C code into Ada. In fact, productivity was as high as 500 to 1000 lines of code a week per person during the actual coding phase.

Perhaps it is worthwhile to describe the difficulties encountered while porting C to Ada. A lack of function pointers in Ada made it necessary to write a case statement which contains all the Ada subprograms that were called either by the system or by the user. This case statement was generated automatically by the Ada deployment compiler according to the call-out interface specification of system functions and user-defined functions.

In C, it is legal to treat an arbitrary memory location as a certain data type at run-time through type casting. For example, four bytes of memory could be used as a long integer or as a pointer depending on how it is casted. Similarly, a pointer to a data type can be casted to a pointer to another data type. Another example is bit manipulation operations such as bitwise exclusive OR which is useful in hashing. Ada does not allow such practices in general. Certainly they are not recommended if they are not prohibited. Since ART-IM uses many such C features to achieve the maximum efficiency, it was unavoidable to sacrifice some performance when it was ported to Ada.

4.4 Reuse

In order to reduce the development cost of the ART/Ada prototype, it was decided in the early phase of the project that the Booch components [4] would be used in the ART/Ada prototype. A linked-list package, a string package, and other utility packages are used by ART/Ada. The following is the full list:

- "vcalenut.a" -- package CALENDAR_UTILITIES
- "vcharuti.a" -- package CHARACTER_UTILITIES
- "vfixedut.a" -- generic package FIXED_POINT_UTILITIES
- "vfloatut.a" -- generic package FLOATING_POINT_UTILITIES
- "vintegrt.a" -- generic package INTEGER_UTILITIES
- "vlistsum.a" -- generic package LIST_SINGLE_UNBOUNDED_MANAGED
- "vstorage.a" -- generic package STORAGE_MANAGER_SEQUENTIAL
Some of these packages had to be modified because they failed to compile on a certain compiler, or their functionality was not what was desired. The modified version of these components were successfully compiled on Alsys, Verdix, and DEC compilers. The package body of LIST_SINGLE_UNBOUNDED_MANAGED did not compile on the Sun Tartan compiler because of a bug in the compiler. Consequently, the Tartan compiler had to be excluded in the benchmark.

4.5 Testing

It was difficult to unit-test ART/Ada modules which were part of the inference engine kernel because these modules were highly interdependent. For example, it was impossible to test the join network module without the pattern network module. For this reason, test programs originally developed for ART-IM were modified and used to validate ART/Ada functionally as well as to do some unit testing. This validation and verification method turned out to be very effective. It is analogous to the Ada compiler validation test suite.

In the future, if an independent third party verification and validation contractor develops a set of test suites for a particular expert system tool, it would be an effective way to validate an expert system tool such as ART/Ada. It does not seem feasible to develop a general purpose test suite for several expert system tools because the input languages are usually very different. It may be possible to come up with a set of general requirements for developing such a test suite, though.

4.6 Debugging

In VLSI testing, a "golden device" that has been proven correct in advance is used to test chips in production. Likewise, ART-IM served as "golden software" while testing and debugging ART/Ada.

Many times, a source-level C debugger on a Sun, dbxtool, was used side by side with the Verdix Ada debugger to track down subtle bugs, which was very effective. While single-stepping through critical code segments, difficult bugs were easily isolated.
4.7 Summary

It was a great advantage to have a commercial expert system tool, ART-IM, and its source code throughout the development cycle of the ART/Ada prototype. During all phases of prototype development, it helped programmers greatly. As a result, it allowed high productivity among programmers and high quality in the prototype. It also reduced the development time greatly. Without it, it would have been impossible to develop an operational prototype in such a short time. With two programmers working on the project, coding was started in July 1988, and the prototype was fully operational in December 1988. A modified version of Boehm’s spiral model [3] is used to show the ART/Ada 1.0 prototype life cycle in Figure 4-1.

![Spiral Life Cycle of the ART/Ada 1.0 prototype](image)

**Figure 4-1:** Spiral Life Cycle of the ART/Ada 1.0 prototype
5. Performance Analysis

This chapter analyzes the performance of the ART Ada 1.0 prototype.

The following programs were used to benchmark it. In both programs, I/O was suppressed so that the speed measured was not the speed of I/O, but that of the ART/Ada inference engine.

- Monkey and Banana: It fires 81 rules, and it has 13 facts in the knowledge base after it runs.
- N-Queens (6 Queens): It fires 515 rules, and it has 155 facts in the knowledge base after it runs.

ART/Ada code was successfully compiled on the following platforms:
- IBM PS/2 Model 70 using the Alsys 286 DOS Ada compiler Version 4.2
- Sun 3/260 using the Verdix Sun Ada compiler Version 5.5K.
- Sun 3/260 using the Alsys Sun Ada compiler Version 4.2 Beta
- VAXstation II using the DEC Ada compiler Version 1.5

As mentioned earlier, the package body of a Booch component, LIST_SINGLE_UNBOUNDED_MANAGED, did not compile on the Tartan Sun Ada compiler Version 2.0 because of a bug in the compiler. Consequently, the Tartan compiler was excluded in the benchmark.

Ada files were compiled with maximum suppression of error checks and maximum optimization. For the Verdix Sun Ada compiler, the following command was used:

```
make -S -O -v main -f *.a
```

For the Alsys compilers, the following default was used for compilation:

```
COMPILE (OPTIONS => (CHECKS => STACK),
         IMPROVE => (CALLS => INLINED,
                      REDUCTION => EXTENSIVE),
         KEEP => (DEBUG => NO,
                   COPY => NO));
```

In addition, the following default was used on the Alsys 286 DOS compiler for linking:

```
BIND PROGRAM=MAIN, EXECUTION=EXTENDED
```
5.1 Speed

The speed of ART/Ada is measured on the following platforms:

1. Sun 3/260 with 16 MBytes of memory using the Verdix Ada compiler
2. Sun 3/280 with 16 MBytes of memory using the Alsys Ada compiler
3. IBM PS/2 Model 70 386 20 MHz with 6 MBytes of memory using the Alsys Ada compiler

The speed is measured in the number of rules per second against a wall-clock time, not a CPU time. The PS/2 is a single user system. The Sun was connected to the network and was being used as a file server occasionally, but no other program was running while benchmark programs were running.

Here it should be noted that ART/Ada on a PS/2 uses different table sizes. A direct comparison, therefore, between the speed of ART/Ada on a Sun and that on a PS/2 is not possible.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Monkey</th>
<th>6 Queens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun/Verdix</td>
<td>38.2 Rules/Sec</td>
<td>42.7 Rules/Sec</td>
</tr>
<tr>
<td>Sun/Alsys</td>
<td>46.4 Rules/Sec</td>
<td>62.4 Rules/Sec</td>
</tr>
<tr>
<td>PS/2/Alsys</td>
<td>37.4 Rules/Sec</td>
<td>49.9 Rules/Sec</td>
</tr>
</tbody>
</table>

Table 5-1: Speed of ART/Ada in Rules/Second

5.2 Size

The size of ART/Ada is measured on the following platforms:

1. Sun 3/260 with 16 MBytes of memory using the Verdix Ada compiler
2. Sun 3/280 with 16 MBytes of memory using the Alsys Ada compiler

Since multiprocessing is not supported in MS-DOS, size of the ART/Ada process could not be measured. On a Sun workstation, size of the ART/Ada process was measured in KBytes using a Unix command, "ps aux", after the program finished running and just before it was exited.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Monkey</th>
<th>6 Queens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun/Verdix</td>
<td>968 KBytes</td>
<td>1232 KBytes</td>
</tr>
<tr>
<td>Sun/Alsys</td>
<td>768 KBytes</td>
<td>944 KBytes</td>
</tr>
</tbody>
</table>

Table 5-2: Size of ART/Ada in KBytes
5.3 Discussion

The benchmark results reported should be considered as preliminary results because of the following reasons:

- No effort was made to optimize the performance of the ART/Ada prototype due to the time limitation
- More compilers should be included in the benchmark.
- More hardware platforms should be included in the benchmark.
- Better monitoring tools are necessary. One problem with Ada is that it does not support CPU time; it only supports wall-clock time. Therefore, the benchmark result is subject to many variables such as the load on the system, network activities, etc.

The size limitation of current generation embedded processors such as the MIL-STD-1750A is 1 megaword (2 megabytes) within which all software systems including the operating system have to run. This might be too restrictive for medium size expert systems. New generation embedded processors such as the 80386 would be adequate for many expert systems developed using an Ada-based expert system tool such as ART/Ada. The speed of the ART/Ada prototype seems comparable to other tools, especially C-based tools, although it is slower.*

It is interesting that both speed and size of ART/Ada vary significantly depending on which Ada compiler is used. It is known that Ada compilers are not very efficient [6]. As the Ada compiler technology advances, the ART/Ada performance would be improved.

*The unoptimized ART/Ada prototype is about 2-3 times slower in execution speed and about 2-3 times larger in process size than ART-IM.
6. Related Work

This chapter compares the ART/Ada prototype with other similar systems such as:

- CLIPS/Ada [1]
- PAMELA [2]
- FLAC [10]
- L*STAR [12]

CLIPS (C Language Production System) is a C-based forward-chaining rule-based expert system tool whose syntax is very close to ART and ART-IM. It has been reported that CLIPS is being ported to Ada. Unlike ART/Ada, the whole system is being reimplemented in Ada. CLIPS does not have a frame system, a truth maintenance system, and an explanation system. Its only knowledge representation method is a forward-chaining rule system.

It is claimed that PAMELA (PAttern Matching Expert system LAnguage) uses the Rete algorithm improved with optimizations and extensions that could satisfy the requirements of many real-time applications. Unlike ART/Ada, PAMELA does not seem to support deployment in Ada environments. It is implemented in CHILL(Communication High Level Language). PAMELA is similar to ART/Ada because it is based on the Rete algorithm. In addition to PAMELA, other optimizations on the Rete algorithm have been proposed and implemented by Gupta [7] Schor et. al. [14] and Miranker [13].

FLAC (Ford Lisp-Ada Connection) uses a Lisp environment to develop an expert system application, and generates Ada code to be deployed in Ada environments [10]. Its knowledge base is specified using a graphical representation similar to that of VLSI design (e.g. OR gates and AND gates), which gets compiled into a static knowledge base. Because of this compiled, static knowledge base, high performance of 1500 rules per second on a VAX 11/780 was achieved, perhaps, at the cost of flexibility. It still does not guarantee response times.

FLAC is similar to ART/Ada because its development environment is not implemented in Ada, but the Ada deployment is supported. The difference, however, that FLAC's development environment is based on Lisp, while ART/Ada uses that of ART-IM which is written in C. The C and Ada development environments coexist on the same hardware platforms more often than the Lisp and Ada development environments do. FLAC, for example, uses a special purpose Lisp machine for the front-end, and a VAX for the Ada deployment. Both ART-IM and ART/Ada run on the same hardware. Another difference is that FLAC's input is graphically oriented while ART/Ada is language-oriented. FLAC's pre-compiled, static knowledge base imposes restrictions on the reasoning capability which do not exist in the inference engines based on the Rete algorithm such as ART-IM and ART/Ada.
L*STAR (Lockheed Satellite Telemetry Analysis in Real time) is designed for real-time monitoring and analysis expert systems. L*STAR has a built-in feature for temporal reasoning. All data is archived and time-tagged into a ring buffer. The ring buffer consists of a compressed format which keeps track of the last time the datum was updated and each time it changed over a user-specified time period. ART Ada does not support temporal reasoning. It could be implemented, however, using ART Ada’s asynchronous function capability. An asynchronous function is an Ada procedure that gets invoked between rule firings. It can be implemented to achieve the same behavior as L*STAR’s temporal reasoning facility.

Unlike ART/Ada and PAMELA, L*STAR is not based on the Rete algorithm. In L*STAR, not all the rules are continually checked. Some of the rules are triggered by the test clock at regular time intervals. Other rules are checked only when data changes that is used in one of its IF clauses, or when they are needed to achieve a goal. Rules are compiled into an intermediate postfix format or optionally into C. Because all variables are resolved at compile time, multiple variables in a single rule can result in a combinatoric increase in the number of rules generated. In this sense, it is similar to FLAC. While it achieves the performance of about 1000 rules per second on a VAX 8650, L*STAR still cannot guarantee response times. Although it seems to work well for the real-time monitoring and analysis applications, it is unclear whether this architecture would satisfy the requirements of other expert system application areas besides the monitoring and analysis applications. L*STAR is written in C, and does not support deployment in Ada environments.

Tools like FLAC and L*STAR seem to achieve high performance because they have a static knowledge base in which variables are resolved at the compile time. It might be possible to achieve the same level of performance if object-oriented programming (OOP) facilities such as active values are used to invoke actions from objects which represent those variables. When the active value capability is added to ART IM and ART/Ada, the performance of the OOP methodology could be compared to that of the static knowledge base.
7. Future Work

This chapter suggests the future directions for the Ada-Based Expert System Building Tool Design Research Project.

The current prototype system supports only forward-chaining rules. There are multiple knowledge representation techniques one or more of which are usually used for a particular application. To support various application, an expert system tool should support more than one paradigm. It would be useful, therefore, to enhance the ART/Ada prototype with a frame system, a truth maintenance system, and an explanation system which exist in ART-IM 1.5. This enhanced prototype will be called ART/Ada 1.5. ART-IM will be still used as a development environment and as an Ada deployment compiler. A modified version of Boehm's spiral model [3] is used to show the proposed ART/Ada 1.5 prototype life cycle in Figure 7-1.

![Figure 7-1: Proposed Spiral Life Cycle of the ART/Ada 1.5 prototype](image)

Once the ART/Ada 1.5 prototype is completed, significant effort should be dedicated to understand the operational issues and potential uses of the prototype. This may involve a joint effort with potential users who use the ART/Ada 1.5 prototype to implement prototype expert systems for the Space Station Freedom.
Additional research effort would be necessary to enhance ART/Ada 1.5 to better support the real-time embedded applications. The following issues need to be investigated further:

- To meet the soft and hard real-time requirements.
- To support the distributed environments such as parallel processors [7].
- To fit into the embedded processors.

It is feasible to reimplement all of ART-IM in Ada including the front-end and the development environment. The following modules would be required:

- Front-end: a lexer, a parser, a semantic analyzer, a code generator, etc.
- User interface: a graphical debugging tool and debugging-oriented functions to browse various knowledge base objects.
- New Ada deployment compiler written in Ada.
- Miscellaneous: the Clear and Reset commands, an error handling system, etc.

It would not be easy to reimplement ART-IM's development environment in Ada because most graphics packages are written in C. An Ada binding would have to be used to interface ART/Ada with existing C-based graphics packages. Despite standardization efforts such as X windows, graphics applications are not very portable today. It might be necessary, therefore, that the multiple graphics packages (e.g., X Windows and Presentation Manager) be supported. Integration and testing would also require significant effort.

7.1 Summary

In summary, the following projects are recommended as future projects:

- To implement the ART/Ada 1.5 prototype (compatible with ART-IM 1.5).
- To study the operational issues and potential uses of the ART/Ada 1.5 prototype.
- To enhance the current architecture to better support real-time applications.
- To implement the whole ART/Ada 1.5 in Ada including the development environment and integrate it with an existing APSE (Ada Programming Support Environment).
8. Conclusion

As shown in the preliminary benchmark results of the operational prototype, this project succeeded in proving that the conventional expert system tool could be used to deploy its applications in Ada environments with efficient use of time and space.

Another important goal of this project was to reuse existing software. During the prototype development, software reuse techniques were practiced at all levels.

- A commercially available software component library, the Booch Components, was used to implement data structures.

- A commercial software system, ART-IM, was reused for various purposes: as a functional specification and a detailed design of the ART/Ada run-time system; as a development environment for ART/Ada applications; and as an Ada deployment compiler.

- The ART-IM test programs were also reused to test the whole Ada deployment process; to debug the ART-IM Ada deployment compiler and the ART/Ada run-time system.

The reuse practice of the project, especially the reuse of ART-IM, contributed greatly to the high productivity in coding, testing, and integration and the high quality of the ART/Ada prototype. During the coding phase, productivity was as high as 1000 lines of code a week per person, and was in average about 500 lines of code a week per person. Thanks to the ART-IM test programs, it took only about a month to fully validate the prototype. Testing and integration would have taken much longer if no test programs had been available. The source listings of the test programs are available in the appendix.
References


I. Detailed Description of ART/Ada Implementation

In this chapter, the ART/Ada prototype will be described in greater detail.

I.1 Deploying an ART-IM application in Ada

The following steps are necessary to deploy an ART-IM application in an Ada environment:

1. Load an application into ART-IM. This can be achieved either through the ART-IM Studio menus or by entering a command. When the menu is used, select File, Load and an appropriate filename. When a command is used, enter

   \(\text{load}\ (*<\text{filename}>*)\)

2. Reset the application. This can be achieved either through the Studio menus or by entering a command. When the menu is used, select Run and then Reset. When a command is used, enter

   \(\text{reset}\)

3. Generate Ada code for the application. This can be achieved either through the Studio menus or by entering a command. When the menu is used, select File and then Ada Generate. When a command is used, enter

   \(\text{load}\ \text{buildada.art}\)\n\(\text{set-generate-options}\ 1000\ 25\)\n\(\text{generate-ada}\ (*<\text{filename}-prefix>*))\)

The menu command Ada Generate executes the first two commands automatically. The file buildada.art contains Ada call-out definitions used by ART/Ada internally. If there exists user's Ada code to be called from ART/Ada, the call-out interface should be defined either in this file or in a separate file, and loaded into ART-IM. The function, set-generate-options, sets maximum number of source lines per Ada source file and maximum number of source lines per Ada subprogram. For example, \(\text{set-generate-options}\ 1000\ 25\) set the maximum lines per file to 1000, and the maximum number of lines per subprogram to 25. These numbers were found optimal for some Ada compilers. ART-IM will generate multiple files:

- funcall.a --- procedure FUNCALL for calling out to Ada
- \(<\text{filename}-prefix>.a\) --- specification and body of a package, \(<\text{filename}-prefix>\)
- \(<\text{filename}-prefix>.t.a, <\text{filename}-prefix>.2.a, ...\) --- separate procedures contained in the package, \(<\text{filename}-prefix>\)
- \(<\text{filename}-prefix>.com.\) --- a command file to compile Ada source files.

4. Compile the Ada source files using an Ada compiler using <filename-prefix>.com which might have to be customized for each compiler. So far, only Alsys compiler has been used on a PS/2.
5. Write the main program. A simple command loop may or may not be included in the main program.

6. Link the Ada executable image.

7. Run the Ada executable image.

I.1.1 Ada Source Code Generated by the Ada Deployment Compiler

The generated Ada code includes a procedure called INIT which initializes an application in the ART/Ada knowledge base.

Below is the package specification generated by the Ada Generator for an application, MAB:

```ada
package MAB is
  procedure INIT;
end MAB;

with GEN_UTIL_ADO, GLOBAL_DCL;
package body MAB is
  procedure MAB0 is separate;
  procedure MAB1 is separate;
  procedure MAB2 is separate;
  procedure MAB3 is separate;
  procedure MAB4 is separate;
  procedure MAB5 is separate;
  procedure MAB6 is separate;
  procedure INIT is
    begin
      GEN_UTIL_ADO.INIT_INIT;
      MAB1;
      MAB2;
      MAB3;
      MAB4;
      MAB5;
      MAB6;
      GEN_UTIL_ADO.CROSS_REF;
      MAB0;
      GEN_UTIL_ADO.CLEANUP;
      GEN_UTIL_ADO.SYSTEM_INIT;
    end INIT;
  end MAB;
```

In addition to generating a package specification for an application, the Ada deployment compiler also generates the separate procedure body for INTERPRETER_SUB_FUNCALL. This procedure is the top-level procedure called by the function call interpreter to call out to Ada subprograms. These Ada subprograms consist of those used internally by ART/Ada and those defined by the user. All user-defined Ada subprograms should be defined in the package USER_SUB.
I.1.2 ART/Ada User Interface Command Loop

A simple command loop is included in the ART/Ada run-time system. It supports a minimum subset of the ART-M command syntax which is necessary for simple tracing and debugging. The following syntax is supported:

\[
\begin{align*}
\text{art\_cmd} & ::= (\text{command}) \\
\text{command} & ::= \text{trace\_cmd} | \text{untrace\_cmd} | \text{run\_cmd} | \\
& \quad \text{agenda\_cmd} | \text{facts\_cmd} | \text{exit\_cmd} \\
\text{untrace\_cmd} & ::= \text{untrace} \text{untrace\_arg} \\
\text{untrace\_arg} & ::= \text{rules} | \text{facts} | \text{activations} | \text{all} \\
\text{trace\_cmd} & ::= \text{trace} \text{trace\_arg} \\
\text{trace\_arg} & ::= \text{untrace\_arg} | \text{status} \\
\text{run\_cmd} & ::= \text{run} | \text{run} \text{integer} \\
\text{agenda\_cmd} & ::= \text{agenda} \\
\text{facts\_cmd} & ::= \text{facts} \\
\text{exit\_cmd} & ::= \text{exit}
\end{align*}
\]

I.1.3 Example Main Programs

Two examples of the ART/Ada main programs are included in this section: one that includes the command loop, and one that does not. Although the main program should be defined by the user for each application because the name of the package that contains the application specific procedures varies, it would be easy to modify the standard one.

The following is an example of the main program that includes the user interface by calling COMMAND_LOOP:
with USER_INTERFACE_SUB, Error_HDL_SUB, MAB;

procedure MAIN is
begin
   MAB_INIT;
   USER_INTERFACE_SUB COMMAND_LOOP;
exception
   when CONSTRAINT_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB CONSTRAINT_ERR);
   when PROGRAM_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB PROGRAM_ERR);
   when STORAGE_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB STORAGE_ERR);
   when TASKING_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB TASKING_ERR);
   when ERROR_HDL_SUB TIME_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB TIME_ERR);
   when ERROR_HDL_SUB INTERNAL_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB INTERNAL_ERR);
   when ERROR_HDL_SUB RETRACT_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB RETRACT_ERR);
   when ERROR_HDL_SUB INTERPRETER_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB INTERPRETER_ERR);
   when ERROR_HDL_SUB USER_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB USER_ERR);
   when ERROR_HDL_SUB USER_DEFINED_ERROR =>
      Error_HDL_SUB_PROCESS_ERROR(Error_HDL_SUB USER_DEFINED_ERR);
end MAIN;

This main program initializes an expert system application called MAB, and prompts the user for a command. The USER_INTERFACE package is with'ed to gain access to the COMMAND_LOOP procedure.

The following is an example of the main program that is tailored for an embedded application.
with ART, ERROR_HDL_SUB, MAB;
procedure MAIN is
begin
MAB_INIT;
ART_RUN(-1);
exception
when CONSTRAINT_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_CONSTRAINT_ERR);
when PROGRAM_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_PROGRAM_ERR);
when STORAGE_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_STORAGE_ERR);
when TASKING_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_TASKING_ERR);
when ERROR_HDL_SUB.TIME_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_TIME_ERR);
when ERROR_HDL_SUB.INTERNAL_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_INTERNAL_ERR);
when ERROR_HDL_SUB.RETRACT_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_RETRACT_ERR);
when ERROR_HDL_SUB.INTERPRETER_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_INTERPRETER_ERR);
when ERROR_HDL_SUB.USER_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_USER_ERR);
when ERROR_HDL_SUB.USER_DEFINED_ERROR =>
   ERROR_HDL_SUB.PROCESS_ERROR(ERROR_HDL_SUB_USER_DEFINED_ERR);
end MAIN;

This main program initializes and runs an expert system application, called MAB. The USER_INTERFACE package is not with'ed by the main program.

### I.2 Public Packages in ART/Ada

The ART/Ada runtime system is composed of public Ada packages and internal Ada packages. The following is a list of public packages that can be with'ed and, in some cases, modified by the user:

- ART
- ERROR_HDL_SUB
- USER_INTERFACE_SUB

#### I.2.1 ART

The package, ART, contains public subprograms to be used to call into ART/Ada from user's Ada program. This package should be with'ed by the user program whenever subprograms in this package are called. The ART package contains the following:

- Data types: Integer_Type, Natural_Type, Positive_Type, Float_Type, ART_Object
- Type conversions: ART_Symbol, ART_String, ART_Integer, ART_Float, Ada_Symbol, Ada_String, Ada_Integer, Ada_Float
- Predicates: Symbolp, Stringp, Integerp, Floatp, Numberp, Sequencep, Factp
• Operations: Eq, Equal, Type_of, Gentemp, Length, Position, Member, Nth, Set_Nth, Find_Fact, Fact_Number, Register_ART_Object, Unregister_ART_Object, Make_Template, Free_Template, Sequence

• ART commands: Assert, Retract, Run, Halt, Get_Async_Fun, Set_Async_Fun, Get_Salience_Threshold, Set_Salience_Threshold, Get_Limit_Default, Set_Limit_Default, Get_Print_Messages, Set_Print_Messages

• I/O Functions: Print, Print1, Princ, Read

In ART/Ada, all integer numbers are INTEGER_TYPE which is a 32 bit integer, and all float numbers are FLOAT_TYPE which is a 64 bit float. ART_Object is a generic data type which could be one of the following: integer, float, string, symbol, fact, or sequence.

Data type conversion functions are provided to convert ART data types to Ada data types or to convert Ada data types to ART data types.

Predicate functions are similar to Lisp predicates. They return T or NIL depending on the result of the predicate.

The ART package provides various operations on ART_Object.

The ART package also includes ART-IM commands such as Run and Halt as well as functions to change the defaults of ART/Ada.

Simple I/O functions are provided to handle basic input and output. File I/O is not supported.

1.2.2 ERROR_HDL_SUB

The package, ERROR_HDL_SUB, contains subprograms for error handling. It contains two separate procedures, PROCESS_ERROR and WARNING, that can be modified to customize the behavior of the error handler. For example, the default behavior is to print the error or warning messages on the screen. It could be changed, however, to print it on the line printer in an embedded environment. This package also defines exceptions one of which is USER_DEFINED_ERROR. It is a generic exception that can be raised by the user.

1.2.3 USER_INTERFACE_SUB

The package, USER_INTERFACE_SUB, contains a simple user interface that can be invoked by a procedure, COMMAND_LOOP. It also includes debugging functions such as FACTS, AGENDA, and TIMED_RUN. It is not necessary to use this package when the presence of the user interface is not needed (i.e. embedded applications).
I.3 Ada Call-In and Call-Out Specification for ART/Ada and ART-IM

This section describes a portable call-in/call-out interface specification for ART/Ada and ART-IM.

I.3.1 Interface Types

The following types may be passed between ART and Ada:

:INTEGER (INTEGER_TYPE) This type is an 32 bit integer in Ada and an integer in ART (INTEGER_TYPE in ART/Ada and long in ART-IM).

:BOOLEAN (BOOLEAN) In ART, this type is either NIL or non-NIL. In Ada, this type is BOOLEAN which is TRUE or FALSE. When translating from Ada to ART, TRUE will translate to T.

:FLOAT (FLOAT_TYPE) In Ada, this type is FLOAT_TYPE which is double precision float. For ART-IM, this is a C double.

:STRING (STRING) In Ada, this type is represented as a STRING. In ART, this type is represented as an ART string. ART may or may not copy the string being passed by this mechanism when passing a string from ART to Ada. Thus, it is an error to destructively modify a string passed with this mechanism. ART is responsible for freeing any space necessary for the string after exiting the current scope. The actual implementation will be based upon constraints of the underlying architecture. When transferring a string from Ada to ART, ART will always copy the string, allowing the Ada programmer to free the string at his leisure.

:S symbols (STRING) In Ada, this type is represented as a STRING. In ART, this type is a symbol. Case is preserved when interning an STRING as an ART symbol, just as case is preserved when passing a string to the Lisp function INTERN.

:ART-OBJECT (ART_OBJECT) This type is any ART type in ART. It is represented as a pointer to a discriminant record in ART/Ada. For ART-IM, it is an integer type which represents a C pointer to a C structure art_object. A set of Ada functions is provided to operate on these ART objects from Ada.

I.3.2 Scope of Objects

This section gives a detailed description of the scope of objects communicated from ART to Ada and objects communicated from Ada to ART. In both cases the prime motivation for scoping is that the caller should free all objects it allocates, (thus it should not allocate objects which it intends that the callee free). Additionally, the callee should not destructively modify objects which it did not allocate.

All objects that are not immediate fall under these constraints. For example, strings and art-objects passed from ART to Ada conform to the following semantics.
When an ART_OBJECT or string is passed from ART by call out to an Ada function, the object is automatically reclaimed when the Ada function returns to ART. At this point, the Ada ART_OBJECT data structure is no longer valid for use in Ada code. It is an error to retain a pointer to an automatically reclaimed ART_OBJECT in Ada once the Ada call has returned.

When an ART_OBJECT or string is returned from ART to Ada, it is automatically reclaimed when control returns to ART from Ada. In those implementations where Ada can start up and call ART as a subroutine so that a returned value may never be reclaimed, the returned ART_OBJECT is allocated permanently and must be freed using a freeing function supplied in Ada.

A function is supplied in Ada that accepts an ART_OBJECT as argument and returns a permanent copy of that ART_OBJECT. This object must be explicitly freed when no longer useful.

1.3.3 Call-Out from ART to Ada

The following is a grammar for def-user-fun which should used to call out to Ada from ART:

(def-user-fun <fun-name> {<comment>}
   <function-spec>*)

<function-spec> ::= 
   :compiler <compiler-name> | 
   :returns <return-data-type> | 
   :epname <link-editor-symbol> | 
   :args (<arg-spec>*)

<fun-name> ::= <art-symbol>

<comment> ::= <art-string>

<compiler-name> ::= 
   :VERDIX-ADA 
   :DEC-ADA 
   :ALSYS-ADA

<internal-data-type> ::= 
   :SYMBOL 
   :STRING 
   :FLOAT 
   :INTEGER 
   :BOOLEAN 
   :ART-OBJECT

<return-data-type> ::= 
   :VOID | <internal-data-type>

<link-editor-symbol> ::= 
   <art-symbol> | <art-string>
For example, in order to call out to an Ada function, CALC_STD_DEV, using an ART function, calc-std-dev, define the following in ART-IM before the Ada code is generated:

```
(def-user-fun calc-std-dev
  :epname "CALC_STD_DEV"
  :args ((sx :float) (ssq :float) (n :integer))
  :returns :float
  :compiler :alsys-ada)
```

An Ada package called USER_SUB should be also defined as follows:
with ART.MATH_LIB.TEXT_IO,
use ART.TEXT_IO;
package USER_SUB is
  type REAL_TYPE is digits 15;
  package MY_MATH_LIB is new MATH_LIB(REAL_TYPE);
  use MY_MATH_LIB;
  function CALC_STD_DEV(SX: FLOAT_TYPE;
                        SSQ: FLOAT_TYPE;
                        N: INTEGER_TYPE) return FLOAT_TYPE;
end USER_SUB;

package body USER_SUB is
  function CALC_STD_DEV(SX: FLOAT_TYPE;
                        SSQ: FLOAT_TYPE;
                        N: INTEGER_TYPE) return FLOAT_TYPE is
    SD: FLOAT_TYPE;
    begin
      SD := (SSQ - ((SX * SX)/FLOAT_TYPE(N))) / FLOAT_TYPE(N - 1);
      return FLOAT_TYPE(MY_MATH_LIB.SQRT(REAL_TYPE(SD)));
    end CALC_STD_DEV;
end USER_SUB;

1.3.4 Call-In from Ada to ART/Ada

The ART package of ART/Ada is the public package for the ART/Ada users to call in from Ada to ART/Ada. The specification of the ART package will serve as the standard Ada call-in interface specification.
II. Difference between ART-IM 1.5 and ART/Ada 1.0

Among the ART-IM 1.5 features that are missing in ART/Ada, the following features will be implemented in phase II:

- Schema System - a frame system
- Logical Dependency - a truth maintenance system
- Justification System - an explanation system

The following features were not implemented during the phase I due to time limitation, but will be implemented during phase II.

- some string functions
- some I/O functions
- some math functions
- procedural iterators
- asynchronous function

The following features are not planned to be supported in ART/Ada:

- streams
- external data interface functions (e.g. def-external-data, def-map-fun, etc.) which are useful for building database interfaces.
- Trigonometric functions (e.g. sin, cos, etc) which are not part of standard Ada.
III. ART/Ada Public Packages
III.1 Specification of ART Package
COPYRIGHT NOTICE

1) COPYRIGHT (C) 1988
INFERENIE CORPORATION,
5300 W. Century Blvd.,
Los Angeles, California 90045
AN UNPUBLISHED WORK - ALL RIGHTS RESERVED.

2) Restricted Rights Notice (Short Form) (April 1984)
Use, reproduction, or disclosure is subject to restrictions set forth in
Government Cooperative Agreement Number NCC-9-16 between the National Aeronautics and
Space Administration and the University of Houston-Clear Lake and a subcontract
thereunder, Number 015 between the University of Houston-Clear Lake and Inference
Corporation.

3) Restricted Rights Notice (ART/Ada)
These data constitute Inference Corporation trade secrets and/or information that is commercial or financial and
confidential or privileged. They are submitted to the Government under NASA Cooperative Agreement NCC-9-16 with the
University of Houston-Clear Lake Research Institute for Computing and Information Systems (RICIS) with the understanding that
they will not, without the permission of Inference Corporation, be used or disclosed for other than evaluation purposes.

-- Author: S. Daniel Lee
-- Package: ART
-- Function: This package contains subprograms for the user to call into ART/Ada. This package is the top-level public package which contains all the operations on ART/Ada. This package should always be with'ed in the user's program
-- State Variables:
   None
-- State Variable Initialization:
   None
-- Change Log:

with STRUCT_DCL, ART_OBJECT_SUB, DATABASE_SUB, INFER_ENG_SUB, CALLIO_SUB,
ALLOC_SUB, IO_SUB, AGENDA_SUB.
use STRUCT_DCL.
package ART is

-- Public Types for Ada Callout
subtype INTEGER_TYPE is STRUCT_DCL INTEGER_TYPE;  -- For INTEGER
subtype BOOLEAN is STRUCT_DCL BOOLEAN;        -- For BOOLEAN
subtype FLOAT_TYPE is STRUCT_DCL FLOAT_TYPE;  -- For FLOAT
subtype STRING is STRUCT_DCL STRING;          -- For STRING
subtype ART_OBJECT is STRUCT_DCL ART_OBJECT;  -- For ART-OBJECT
subtype NATURAL_TYPE is STRUCT_DCL NATURAL_TYPE;
subtype POSITIVE_TYPE is STRUCT_DCL POSITIVE_TYPE;

-- Operations on ART objects

-- Returns a new, permanent art_object reference to the ART object
-- referred to by reference. Reference may be either a permanent
-- or automatically allocated art_object.
function REGISTER_ART_OBJECT(REFERENCE : ART_OBJECT) return ART_OBJECT
  renames ART_OBJECT_SUB.REGISTER_ART_OBJECT;

-- Frees the permanent or temporary reference to an art_object;
-- it is an error to continue to use an art_object after freeing the
-- reference to it.
procedure UNREGISTER_ART_OBJECT(REFERENCE : ART_OBJECT)
  renames ART_OBJECT_SUB.UNREGISTER_ART_OBJECT;

-- Returns true if the two art_objects X and Y are the same object.
-- EQ and EQUAL are equivalent.
function EQ(X : ART_OBJECT; Y : ART_OBJECT) return BOOLEAN
  renames ART_OBJECT_SUB.EQ;

-- Returns true if the two art_objects X and Y are the same object.
-- EQ and EQUAL are equivalent.
function EQUAL(X : ART_OBJECT; Y : ART_OBJECT) return BOOLEAN
  renames ART_OBJECT_SUB.EQUAL;

-- Returns the type of an object, as a symbol
function TYPE_OF(OBJ : ART_OBJECT) return ART_OBJECT
  renames ART_OBJECT_SUB.TYPE_OF;

-- A_GENTEMP: Creates a new, previously
-- unused symbol
function GENTEMP(STR : STRING) return ART_OBJECT
  renames ART_OBJECT_SUB.GENTEMP;

-- Calls the Ada procedure PROCESS once for each permanent OBJECT
-- that has been allocated passing each permanent art_object
-- as the argument to PROCESS in turn. If PROCESS returns FALSE
-- generic
-- with procedure PROCESS (THE_ITEM : in ART_OBJECT;
-- CONTINUE : out BOOLEAN);
-- procedure FOR_ALL_PER_ART_OBJECTS.

-- Returns an ART_OBJECT for the symbol resulting from performing a
-- intern operation on the string str. Case is preserved in str.
function ART_SYMBOL(STR : STRING) return ART_OBJECT
  renames ART_OBJECT_SUB.ART_SYMBOL;

-- Returns an ART_OBJECT that represents the string specified.
-- Case is preserved.
function ART_STRING(STR : STRING) return ART_OBJECT
  renames ART_OBJECT_SUB.ART_STRING;

-- Returns an ART_OBJECT that represents the number specified.
function ART_INTEGER(NUM: INTEGER_TYPE) return ART_OBJECT
  renames ART_OBJECT_SUB.ART_INTEGER;

-- Returns a STRING that represents the ART_OBJECT specified.
function ADA_SYMBOL(SYMBOL: ART_OBJECT) return STRING
  renames ART_OBJECT_SUB.ADA_SYMBOL;

-- Returns a STRING that represents the ART_OBJECT specified.
function ADA_STRING(STR: ART_OBJECT) return STRING
  renames ART_OBJECT_SUB.ADA_STRING;

-- Returns the number represented by the ART_OBJECT specified.
function ADA_INTEGER(NUM: ART_OBJECT) return INTEGER_TYPE
  renames ART_OBJECT_SUB.ADA_INTEGER;

-- Returns the number represented by the ART_OBJECT specified.
function ADA_FLOAT(NUM: ART_OBJECT) return FLOAT_TYPE
  renames ART_OBJECT_SUB.ADA_FLOAT;

-- Predicates

-- Returns TRUE if the ART_OBJECT, OBJ, is a symbol; otherwise FALSE.
function SYMBOLP(OBJ: ART_OBJECT) return BOOLEAN
renames ART_OBJECT_SUB_SYMBOLP;

-- Returns TRUE if the ART_OBJECT, OBJ, is a string, otherwise FALSE

function STRINGP(OBJ: ART_OBJECT) return BOOLEAN
renames ART_OBJECT_SUB_STRINGP;

-- Returns TRUE if the ART_OBJECT, OBJ, is an integer, otherwise FALSE

function INTEGERP(OBJ: ART_OBJECT) return BOOLEAN
renames ART_OBJECT_SUB_INTEGERP;

-- Returns TRUE if the ART_OBJECT, OBJ, is a float, otherwise FALSE.

function FLOATP(OBJ: ART_OBJECT) return BOOLEAN
renames ART_OBJECT_SUB_FLOATP;

-- Returns TRUE if the ART_OBJECT, OBJ, is a number, otherwise FALSE.

function NUMBERP(OBJ: ART_OBJECT) return BOOLEAN
renames ART_OBJECT_SUB_NUMBERP;

-- Returns TRUE if the ART_OBJECT, OBJ, is a sequence, otherwise FALSE.

function SEQUENCEP(OBJ: ART_OBJECT) return BOOLEAN
renames ART_OBJECT_SUB_SEQUENCEP;

-- Returns TRUE if the ART_OBJECT, OBJ, is a fact, otherwise FALSE.

function FACTP(OBJ: ART_OBJECT) return BOOLEAN
renames ART_OBJECT_SUBFACTP;

-- Fact and Sequence Manipulation

-- Returns the fact with fact number n. If no fact has that number,-- returns NULL.

function FIND_FACT(N: INTEGER_TYPE) return ART_OBJECT
renames DATABASE_SUB_FIND_FACT;

-- INT_FIND_FACT: Finds a fact based on an ID, and returns NIL instead of NULL.

function INT_FIND_FACT(ID: INTEGER_TYPE) return ART_OBJECT
renames DATABASE_SUB_INT_FIND_FACT;

-- Returns the fact number of fact.

function FACT_NUMBER(FACT: ART_OBJECT) return NATURAL_TYPE
renames DATABASE_SUB_FACT_NUMBER;

-- FOR_ALL_FACTS: Iterates over all the facts in the current database, calling a given
-- procedure once for each fact.

-- generic
-- with procedure PROCESS (THE_ITEM : in ART_OBJECT;
-- CONTINUE : out BOOLEAN);
-- procedure FOR_ALL_FACTS;

-- Returns length of obj.

function LENGTH(OBJ: ART_OBJECT) return NATURAL_TYPE
renames ART_OBJECT_SUB.LENGTH;

-- Returns the position of the first occurrence of value in obj. If
-- value is not in obj, returns 0

function POSITION(VALUE: ART_OBJECT;
OBJ: ART_OBJECT) return NATURAL_TYPE
renames ART_OBJECT_SUB.POSITION;

-- Returns TRUE if value is in obj, and FALSE otherwise.

function MEMBER(VALUE: ART_OBJECT;
OBJ: ART_OBJECT) return BOOLEAN
renames ART_OBJECT_SUB.MEMBER;

-- Returns the nth element of obj.

function NTH(OBJ: ART_OBJECT;
INDEX: NATURAL_TYPE) return ART_OBJECT
renames ART_OBJECT_SUB.NTH;

-- Fact and Sequence Creation

-- Constructs an "empty" fact template and returns a pointer to it as an
-- ART object which may later be asserted. It is an error to assert a
-- template without inserting something into each of the size slots
-- allocated in it. All templates are permanent. Additionally they should
-- not be freed with unregister_art_object. They should only be freed
-- with free_template

function MAKE_TEMPLATE(SIZE: NATURAL_TYPE) return TEMPLATE_TYPE
renames CALLIO_SUB.MAKE_TEMPLATE.

-- This function sets the element of template specified by
-- index to be value. It is an error to attempt to modify a fact
-- not created with MAKE_TEMPLATE. The first element of the fact (the
-- relation) is indexed by index 1. The other elements of the fact
-- have indices 2 through the length of the fact

procedure SET_NTH(TEMPLATE, in out TEMPLATE_TYPE;
INDEX, in INTEGER_TYPE;
VALUE, in ART_OBJECT)
renames CALLIO_SUB.SET_NTH;

-- Frees the TEMPLATE_TYPE template. It is an error to continue to refer to
procedure FREE_TEMPLATE(TEMPLATE: in out TEMPLATE_TYPE)
  renames CALLIO_SUB_FREE_TEMPLATE.

-- Asserts a fact from the contents of template into the ART database
-- Template must be constructed using the functions and macros below
-- prior to assertion. It is an error to assert a fact with an empty fact
-- slot. A template may be used for any number of assertions

function ASSERT(TEMPLATE: in TEMPLATE_TYPE) return ART_OBJECT
  renames CALLIO_SUB_ASSERT.

-- This function takes a template and returns a sequence matching the
-- template. The sequence returned will not incorporate or alter the
-- template.

function SEQUENCE(TEMPLATE: in TEMPLATE_TYPE) return ART_OBJECT
  renames CALLIO_SUB_SEQUENCE.

-- ART Control

-- Retracts fact from the ART database

procedure RETRACT(FACT: in out ART_OBJECT)
  renames DATABASE_SUB_RETRACT.

-- Function: Runs the inference engine (match-select-act cycle) LIMIT
-- number of times. Continue to run until the agenda is
-- empty, until the HALT is encountered on the rhs of a rule,
-- until a salience threshold is reached, or until a breakpoint
-- is triggered.
-- Parameters:
-- LIMIT - Number of inference engine cycles. (Or number of rules
-- allowed to fire.
-- > 0 fire that many rules
-- = 0 then No rules fire
-- = -1 then LIMIT := current default limit
-- <= -2 fire until agenda becomes empty

function RUN(RUN_LIMIT: in INTEGER_TYPE) return INTEGER_TYPE
  renames INFER_ENG_SUB_RUN.

-- Returns TRUE if the agenda is empty. Otherwise, FALSE.

function AGENDA EMPTY P return BOOLEAN
  renames AGENDA_SUB_AGENDA EMPTY P.

-- Function: Complete the execution of all rhs actions of the current
-- rule and halts the inference engine.

procedure HALT
  renames INFER_ENG_SUB_HALT.
function BOOLEAN_TO_INTEGER(STATUS : BOOLEAN) return INTEGER_TYPE
renames CALLIO_SUB BOOLEAN_TO_INTEGER;

-- Function: It frees a sequence
procedure FREE_SEQUENCE(X : in out ART_OBJECT)
renames ALLOC_SUB FREE_SEQUENCE;

-- Function: Prints the object followed by a CR LF to the standard output
-- Standard output is, by default, the screen
-- Parameters: OBJECT - The object to be printed
function PRINT(OBJECT : ART_OBJECT;
STREAM : ART_OBJECT := null) return ART_OBJECT
renames IO_SUB.PRINT;

-- Function: Prints the ART Object to the standard output.
-- Standard output is, by default, the screen.
-- Puts quotes around the string being printed.
-- Parameters: OBJECT - The ART Object to be printed
function PRIN0(OBJECT : ART_OBJECT;
STREAM : ART OBJECT := null) return ART_OBJECT
renames IO_SUB.PRIN;

-- Function: Prints a CR to the standard output
-- Standard output is, by default, the screen
procedure TERPRI(STREAM : ART_OBJECT = null)
renames IO_SUB.TERPRI;

-- Prints out a list of objects to the given stream. The symbol 'T indicates
-- that a newline should be emitted. The symbol 'T as the stream argument
-- indicates that stdout should be used as the stream (for compatibility with
-- big Art). Conceptually, the argument list of printout is:
-- printout(stream &rest args)
-- For now, ignore the first arg, STREAM.
-- Parameters: OBJECT - The ART Object to be printed
-- REST - ART_OBJECT_ARRAY_PTR_TYPE
function PRINTOUT(STREAM : ART_OBJECT;
REST : ART_OBJECT_ARRAY_PTR_TYPE) return BOOLEAN
renames IO_SUB PRINTOUT;

-- Function: Read from standard output.
-- Parameters: OBJECT - The ART_OBJECT being read
--........................................................................

function READ_INTERFACE(STREAM | ART_OBJECT := null) return ART_OBJECT
  renames IO_SUB.READ_INTERFACE;

end ART;
III.2 Specification of ERROR_HDL_SUB Package
-- Copyright Notice

1) Copyright (C) 1988
INFORMATION CORPORATION,
5300 W. Century Blvd.,
Los Angeles, California 90045.

AN UNPUBLISHED WORK — ALL RIGHTS RESERVED

2) Restricted Rights Notice (Short Form) (April 1984)

Use, reproduction, or disclosure is
subject to restrictions set forth in
Government Cooperative Agreement Number NCC-
9-16 between the National Aeronautics and
Space Administration and the University of
Houston-Clear Lake and a subcontract
thereunder, Number 015 between the University
of Houston-Clear Lake and Inference
Corporation.

3) Restricted Rights Notice (ART/Ada)

These data constitute Inference
Corporation trade secrets and/or information
that is commercial or financial and
confidential or privileged. They are
submitted to the Government under NASA
Cooperative Agreement NCC-9-16 with the
University of Houston-Clear Lake Research
Institute for Computing and Information
Systems (RICIS) with the understanding that
they will not, without the permission of
Inference Corporation, be used or disclosed
for other than evaluation purposes

Author: Jim Badura
Package: ERROR_HDL_SUB
Function: This package contains a procedure that performs error
recovery for internal ART errors.
State Variables: None
State Variable Initialization: None
Change Log:

with STRUCT_DCL, CALENDAR;
use STRUCT_DCL;
package ERROR_HDL_SUB is

type ERROR_LOC is (LHS_LOC, RHS_LOC, TOLEVEL_LOC, ASYNC_LOC);
type ERROR_TYPE is (CONSTRAINT_ERR, NUMERIC_ERR, PROGRAM_ERR, STORAGE_ERR,
TASKING_ERR, TIME_ERR,
INTERNAL_ERR, RETRACT_ERR, INTERPRETER_ERR,
USER_ERR,
USER_DEFINED_ERR)

46
procedure PROCESS_ERROR(ERROR: in ERROR_TYPE);

-- Function: This procedure invokes the appropriate Ada routine for
-- recovering from the current internal ART error.
-- This procedure should be separate.

-- Parameters: ERROR - The current error being handled.

procedure ERROR(MESSAGE: in STRING);

-- Function: This procedure stores an error message into a buffer
-- so that the error message could be printed by PROCESS_ERROR later.

-- Parameters: MESSAGE - The error message.

procedure WARNING(MESSAGE: in STRING);

-- Function: This procedure handles a warning message.
-- This procedure should be separate.

-- Parameters: MESSAGE - The warning message.

end ERROR_HDL_SUB;
III.3 Body of ERROR_HDL_SUB Package
C O P Y R I G H T  N O T I C E

1) COPYRIGHT (C) 1988
INFERENCE CORPORATION,
5300 W Century Blvd.,
Los Angeles, California 90045
AN UNPUBLISHED WORK - ALL RIGHTS RESERVED

2) Restricted Rights Notice (Short Form) (April 1984)

Use, reproduction, or disclosure is subject to restrictions set forth in
Government Cooperative Agreement Number NCC-9-16 between the National Aeronautics and
Space Administration and the University of
Houston-Clear Lake and a subcontract
thereunder, Number 015 between the University
of Houston-Clear Lake and Inference Corporation.

3) Restricted Rights Notice (ART/Ada)

These data constitute Inference Corporation trade secrets and/or information
that is commercial or financial and confidential or privileged. They are
submitted to the Government under NASA Cooperative Agreement NCC-9-16 with the
University of Houston-Clear Lake Research Institute for Computing and Information Systems (RICIS) with the understanding that
they will not, without the permission of Inference Corporation, be used or disclosed for other than evaluation purposes.

-- Author: Jim Badura

package ERROR_HDL_SUB is

procedure PROCESS_ERROR(ERROR: in ERROR_TYPE) is separate;

--- Author: Jim Badura
--- Package: ERROR_HDL_SUB
--- Function: This package contains a procedure that performs error recovery for internal ART errors.
--- State Variables: None
--- State Variable Initialization: None
--- Change Log:

with GLOBAL_DCL;
use GLOBAL_DCL;
package body ERROR_HDL_SUB is

--- Function: This procedure invokes the appropriate Ada routine for recovering from the current internal ART error
--- Parameters: ERROR - The current error being handled

procedure PROCESS_ERROR(ERROR : in ERROR_TYPE) is separate;
-- Function: This procedure stores an error message into a buffer
so that the error message could be printed by PROCESS_ERROR later.

-- Parameters: MESSAGE - The error message.

```
procedure ERROR(MESSAGE in STRING) is
  CHARACTER STRING CCPY(MESSAGE, CUR_USER.ERROR_BUFFER);
end ERROR;
```

-- Function: This procedure handles the warning message.

-- Parameters: MESSAGE - The warning message.

```
procedure WARNING(MESSAGE: in STRING) is separate;
```

end ERROR_HDL_SUB;
III.4 Separate Procedure

ERROR_HDL_SUB.PROCESS_ERROR
C O P Y R I G H T   N O T I C E

1) COPYRIGHT (C) 1988
   INFERENCE CORPORATION.
   5300 W. Century Blvd.,
   Los Angeles, California 90045.
   AN UNPUBLISHED WORK - ALL RIGHTS RESERVED.

2) Restricted Rights Notice (Short Form) (April 1984)

   Use, reproduction, or disclosure is
   subject to restrictions set forth in
   Government Cooperative Agreement Number NCC-
   9-16 between the National Aeronautics and
   Space Administration and the University of
   Houston-Clear Lake and a subcontract
   thereunder, Number 015 between the University
   of Houston-Clear Lake and Infrence
   Corporation.

3) Restricted Rights Notice (ART/Ada)

   These data constitute Inference
   Corporation trade secrets and/or information
   that is commercial or financial and
   confidential or privileged. They are
   submitted to the Government under NASA
   Cooperative Agreement NCC-9-16 with the
   University of Houston-Clear Lake Research
   Institute for Computing and Information
   Systems (RICIS) with the understanding that
   they will not, without the permission of
   Infrence Corporation, be used or disclosed
   for other than evaluation purposes.

with TEXT_IO;
use TEXT_IO;
separate (ERROR_HDL_SUB)
procedure PROCESS_ERROR(ERROR: in ERROR_TYPE) is
   ERR: constant STRING := ";;; ERROR: ";
begin
   case ERROR is
   when CONSTRAINT_ERR
      => PUT_LINE(ERR & "Constraint error");
   when NUMERIC_ERR
      => PUT_LINE(ERR & "Numeric error");
   when PROGRAM_ERR
      => PUT_LINE(ERR & "Program error");
   when STORAGE_ERR
      => PUT_LINE(ERR & "Storage error");
   when TASKING_ERR
      => PUT_LINE(ERR & "Tasking error");
   when TIME_ERR
      => PUT_LINE(ERR & "Time error");
   when INTERNAL_ERR
      => PUT_LINE(ERR & "Internal ART error:
                         CHARACTER_STRING SUBSTRING_OF(CUR_USER.ERROR_BUFFER));
                         CHARACTER_STRING CLEAR(CUR_USER.ERROR_BUFFER);
   when RETRACT_ERR
      => PUT_LINE(ERR & "Retract error:
                         CHARACTER_STRING SUBSTRING_OF(CUR_USER.ERROR_BUFFER));
   when other
      => PUT_LINE(ERR & "Other error:
                         CHARACTER_STRING SUBSTRING_OF(CUR_USER.ERROR_BUFFER));
   end case;
end PROCESS_ERROR;
CHARACTER_STRING CLEAR(CUR_USER.ERROR_BUFFER);
when INTERPRETER_ERR
  => PUT_LINE(ERR & "Interpreter error: " &
              CHARACTER_STRING.SUBSTRING_OF(CUR_USER.ERROR_BUFFER));
  CHARACTER_STRING CLEAR(CUR_USER.ERROR_BUFFER);
when USER_ERR
  => PUT_LINE(ERR & "User error " &
              CHARACTER_STRING.SUBSTRING_OF(CUR_USER.ERROR_BUFFER));
  CHARACTER_STRING CLEAR(CUR_USER.ERROR_BUFFER);
when USER_DEFINED_ERR
  => PUT_LINE(ERR & "User defined error");
end case;
end PROCESS_ERROR.
III.5 Separate Procedure ERROR_HDL_SUB.WARNING

with TEXT_ID;
use TEXT_ID;

separate(ERROR_HDL_SUB)

procedure WARNING(MESSAGE: in STRING) is
begin
  PUT_LINE(";;: WARNING: " & MESSAGE);
end WARNING;}
III.6 Specification of USER_INTERFACE Package
Proprietary Rights of Inference Corporation in the ART(TM),
ARTLV(TM), ART Studio(TM), ARTIST(TM) and
Viewpoints(TM) programs include the following:

1) The ART(R), ART Studio(TM), ARTIST(TM)
and Viewpoints(TM) programs and related data and
information are the subject of TRADE SECRETS and
COPYRIGHTS licensed from INFEERENCE CORPORATION.
The program and related data and information are
provided in confidence and all use, disclosure,
copying, or storage except as authorized in the
written LICENSE AGREEMENT FROM INFEERENCE to the
user, is strictly prohibited.

2) COPYRIGHT (C) 1988, 1987, 1986, 1985, 1984 INFEERENCE CORPORATION,
5300 W. Century Blvd., Los Angeles, California 90045.
AN UNPUBLISHED WORK -- ALL RIGHTS RESERVED.

3) RESTRICTED RIGHTS LEGEND:
When the Licensee is the U.S. Government or a duly
authorized agency thereof, use, duplication, or disclosure
by the U.S. Government is subject to restrictions as set
forth in subdivision (b) (3) (ii) of the Rights in
Technical Data and Computer Software clause at
52.227-7013, dated November 9, 1984.

Author: J. T. Badura
Package: USER_INTERFACE_SUB
-- Function: This package contains subprograms that controls the user
-- interface.
State Variables: None
State Variable Initialization: None
Change Log:

with STRUCT_DCL;
use STRUCT_DCL.
package USER_INTERFACE_SUB is
   -- Function: This procedure invokes the interactive ART/Ada command loop.
   -- This procedure will display the initial banner and repeatedly
   -- print the ART/Ada prompt for a ART command.
   -- The command loop should handle watch/unwatch, reset, run,
   -- agenda, facts
procedure COMMAND_LOOP;

   -- FACTS: Prints out a list of the current facts
   -- to the screen, in sorted order and with a
   -- sum of the total current facts
   procedure FACTS;

   -- AGENDA: Prints out the agenda of the rules
   -- that are ready to fire
   procedure AGENDA;
procedure AGENDA;

------------------------------------------------------------------------
-- Timed_run: Calls a timing function to determine the amount of
-- time a run has taken
------------------------------------------------------------------------
procedure TIMED_RUN(RUN_LIMIT, INTEGER_TYPE);
end USER_INTERFACE_SUB;
IV. Benchmark Programs

This appendix includes the following ART-IM programs used to benchmark the ART/Ada prototype:

- Monkey and Banana
- N-Queens (6 Queens)
IV.1 Monkey and Banana
Monkees and Bananas Sample Problem

This is an extended version of a rather common AI planning problem. The point is for the monkee to find and eat some bananas.

To execute, merely load, reset and run.

-----------------------------

;* chest unlocking rules *
-----------------------------

(defun unlock-chest-to-hold-object "
  (goal-is-to active holds ?obj)
  (object ?chest ? ? ?obj ?)
  (not (goal-is-to active unlock ?chest))
=>
  (assert (goal-is-to active unlock ?chest)))

(defun unlock-chest-to-move-object "
  (goal-is-to active move ?obj ?)
  (object ?chest ? ? ?obj ?)
  (not (goal-is-to active unlock ?chest))
=>
  (assert (goal-is-to active unlock ?chest)))

(defun hold-chest-to-put-on-floor "
  (goal-is-to active unlock ?chest)
  (object ?chest ? light "floor ? ?)
  (not (goal-is-to active holds ?chest))
=>
  (assert (goal-is-to active holds ?chest)))

(defun put-chest-on-floor "
  (goal-is-to active unlock ?chest)
  ?fl <- (monkey ?place ?on ?chest)
  ?f2 <- (object ?chest held light held contains ?key)
=>
  (printout t "Monkey throws " ?chest " off " ?on " onto floor " t)
  (retract ?fl ?f2)
  (assert (monkey ?pl&ce ?on blank))
  (assert (object ?chest ?place light floor contains ?key)))

(defun get-key-to-unlock "
  (goal-is-to active unlock ?obj)
  (monkey ? ? "?key")
  (not (goal-is-to active holds ?key))
=>
  (assert (goal-is-to active holds ?key)))

(defun move-to-chest-with-key "
  (goal-is-to active unlock ?chest)
  (monkey ?mplace ? ?key)
  (not (goal-is-to active walk-to ?cplace))
=>
  (assert (goal-is-to active walk-to ?cplace)))

60
(defrule unlock-chest-with-key **
  ?f1 <- (goal-is-to active unlock ?chest-obj)
  (monkey ?place ?on ?key)
  =>
  : (printout t "Monkey opens chest with " ?key " revealing " ?obj-in t)
  (retract ?f1 ?f2)
  (assert (object ?obj-in ?place light ?chest-obj nil nil)))

........................
* process hold object *
........................

(defrule use-ladder-to-hold **
  (goal-is-to active holds ?ob_)
  (object ?ob_ ?place light ceiling ? ?)
  (not (goal-is-to active move ladder ?place))
  =>
  (assert (goal-is-to active move ladder ?place)))

(defrule climb-ladder-to-hold **
  (goal-is-to active holds ?ob_)
  (object ?ob_ ?place light ceiling ? ?)
  (object ladder ?place ?floor ? ?)
  (not (goal-is-to active on ladder))
  =>
  (assert (goal-is-to active on ladder)))

(defrule grab-object-from-ladder **
  ?f1 <- (goal-is-to active holds ?obj)
  (object ladder ?place ?floor ? ?)
  (?f3 <- (monkey ?place ladder blank)
  =>
  : (printout t "Monkey grabs the " ?obj t)
  (retract ?f1 ?f2 ?f3)
  (assert (object ?obj held light held ?contains ?key))
  (assert (monkey ?place ladder ?obj))))

(defrule climb-to-hold **
  (goal-is-to active holds ?ob_)
  (monkey ?place ?on ?)
  (not (goal-is-to active on ?on))
  =>
  (assert (goal-is-to active on ?on)))

(defrule walk-to-hold **
  (goal-is-to active holds ?obj)
  (object ?obj ?place light ?ceiling ? ?)
  (monkey ?place ? ?)
  (not (goal-is-to active walk-to ?place))
  =>
  (assert (goal-is-to active walk-to ?place)))

(defrule drop-to-hold **
  (goal-is-to active holds ?obj)
  (object ?obj ?place light floor ? ?)
  (monkey ?place ?blank)
  (not (goal-is-to active holds blank))
  =>
  (assert (goal-is-to active walk-to ?place)))
(defrule get-on-floor-to-bold "
(goal-is-to active holds ?obj)
(object ?obj ?place light floor ?)
(monkey ?place "floor")
(not (goal-is-to active on floor))
=>
(assert (goal-is-to active on floor)))

(defrule grab-object "
;f1 <- (goal-is-to active holds ?obj)
?f3 <- (monkey ?place "on blank")
=>
(printout t "Monkey grabs the " ?obj t)
(retract ?f1 ?f2 ?f3)
(assert (object ?obj held light held ?contains ?key))
(assert (monkey ?place "on blank"))

............... move object to a place *
..............

(defrule hold-object-to-move "
(goal-is-to active move ?obj ?place)
(object ?obj "place light ?")
(monkey ?"place ?")
(not (goal-is-to active holds ?obj))
=>
(assert (goal-is-to active holds ?obj)))

(defrule move-object-to-place "
(goal-is-to active move ?obj ?place)
(monkey ?place ?obj)
(not (goal-is-to active walk-to ?place))
=>
(assert (goal-is-to active walk-to ?place)))

(defrule drop-object-once-moved "
?f1 <- (goal-is-to active move ?obj ?place)
?f2 <- (monkey ?place ?on ?obj)
=>
(printout t "Monkey drops the " ?obj " t)
(retract ?f1 ?f2 ?f3)
(assert (monkey ?place "on blank"))
(assert (object ?obj ?place light floor ?contains ?key)))

(defrule already-moved-object "
?f1 <- (goal-is-to active move ?obj ?place)
(object ?obj ?place ? ? ?)
=>
(retract ?f1))

............... process walk-to place *
...............

(defrule already-at-place "
?f1 <- (goal-is-to active walk-to ?place)
(monkey ?place ?)
=>
(retract ?f1))

(defrule get-on-floor-to-walk "

(goal-is-to active walk-to ?place)
  (monkey "place" "floor")
  (not (goal-is-to active on floor)) =>
  (assert (goal-is-to active on floor)))

(defrule walk-holding-nothing "'
  (?f1 <- (goal-is-to active walk-to ?place)
  (?f2 <- (monkey "place" floor blank))
  =>
  (printout t "Monkey walks to " ?place t)
  (retract ?f1 ?f2)
  (assert (monkey ?place floor blank)))

(defrule walk-holding-object "'
  (?f1 <- (goal-is-to active walk-to ?place)
  (?f2 <- (monkey "place" floor "obj" "blank")
  =>
  (printout t "Monkey walks to " ?place " holding " ?obj t)
  (retract ?f1 ?f2)
  (assert (monkey ?place floor "obj")))

(defrule drop-object "'
  (?f1 <- (goal-is-to active holds blank)
  (?f2 <- (monkey "place" on "obj" "blank")
  (?f3 <- (object ?obj held light held ?inside "key")
  =>
  (printout t "Monkey drops " ?obj t)
  (retract ?f1 ?f2 ?f3)
  (assert (object ?obj ?place light ?on ?inside "key"))
  (assert (monkey ?place on blank)))

  defrule jump-onto-floor "
  (?f1 <- (goal-is-to active on floor)
  (?f2 <- (monkey "at on" "floor" ?obj))
  =>
  (printout t "Monkey jumps off " ?on " onto the floor " t)
  (retract ?f1 ?f2)
  (assert (monkey "at floor" ?obj)))

(defrule walk-to-place-to-climb "
  (goal-is-to active on ?obj)
  (object ?obj "place" ? ? ?)
  (monkey "place" ? ?)
  (not (goal-is-to active walk-to ?place)) =>
  (assert (goal-is-to active walk-to ?place)))

(defrule drop-to-climb "
  (goal-is-to active on ?obj)
  (object ?obj "place" ? ? ?)
  (monkey "place" "blank")
  (not (goal-is-to active holds blank)) =>
  (assert (goal-is-to active holds blank)))

(defrule climb-indirectly "
  (goal-is-to active on ?obj)
  (object ?obj "place" ? on ? ?)
  (monkey "place" "on" "obj blank)
(not (goal-is-to active on ?on))
=>
(assert (goal-is-to active on ?on))

(defrule climb-directly **
?f1 <- (goal-is-to active on ?obj)
(object ?obj ?place ? ?)
?f2 <- (monkey ?place ?on blank)
=>
(printout t "Monkey climbs onto " ?obj t)
(retract ?f1 ?f2)
(assert (monkey ?place ?obj blank)))

(defrule already-on-object **
?f1 <- (goal-is-to active on ?obj)
(monkey ? ?obj ?)
=>
(retract ?f1))

* process eat object *

(defrule hold-to-eat **
(goal-is-to active eat ?obj)
(monkey ? ? ?obj)
(not (goal-is-to active holds ?obj))
=>
(assert (goal-is-to active holds ?obj)))

(defrule satisfy-hunger **
?f1 <- (goal-is-to active eat ?obj)
?f2 <- (monkey ?place ?on ?obj)
=>
(printout t "Monkey eats the " ?obj "." t)
(retract ?f1 ?f2 ?f3)
(assert (monkey ?place ?on blank)))

* initial-state *

(defrule startup **
(start-fact)
=>
(assert (monkey t5-7 green-couch blank))
(assert (object green-couch t5-7 heavy floor foo foo))
(assert (object red-couch t2-2 heavy floor foo foo))
(assert (object big-pillow t2-2 light red-couch foo foo))
(assert (object red-chest t7-7 light ceiling bananas red-key))
(assert (object blue-chest t7-7 light ceiling bananas blue-key))
(assert (object blue-couch t8-8 heavy floor foo foo))
(assert (object green-chest t8-8 light ceiling blue-key red-key))
(assert (object red-key t1-3 light floor foo foo))
(assert (goal-is-to active eat bananas)))

(deffacts start-fact (start-fact))
IV.2 N-Queens
The problem is to place N queens on an N×N chessboard so that no queen attacks another. (A queen attacks another if they are on the same row, column, or diagonal.)

Example:

```
  Q  
--
-- Q
```

To run the demo, load this file into ART. Then reset and run it.

```
(load "rxx/nqueens.art")
(reset)
(run)
```

When the program asks "How many rows on the board?", type 4 and then ENTER.

To run the program again, just (reset) and (run).

ART features illustrated in this program:
- Defglobal
- Defrule
- Salience
- A startup rule, with null left-hand-side
- Pattern tests with : :
- Sequence variables, Variable length facts
- Assert, Retract, Binding a fact variable with <-
- Procedural language:
  - Bind, Arithmetic, Absolute value function, Comparisons with = and <
  - Sequence functions and predicates: Length$, Nth$, Member$
- Iteration and Conditionals: For, Downto, If, Not
- Input-Output: Read, Printout, character strings

<table>
<thead>
<tr>
<th>Board size</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of solutions</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>40</td>
<td>92</td>
<td>352</td>
</tr>
</tbody>
</table>

(Solutions that are reflections or rotations of another solution are considered distinct.)

Since there are as many queens as rows of the chessboard, each row must contain exactly one queen. This program generates partial solutions in which the first K rows of the board are filled with K non-attacking queens. We begin with a blank board. The partial solutions are extended one row/queen at a time. When the newly added queen attacks a previous queen, the extended partial solution is discarded. Any partial solution that has N rows filled in is a true solution, and is printed.

A partial solution is represented as (SOLUTION c1 c2 ... ck). ci ... ck are numbers between 1 and N. ci is the column of the queen in row i.

```
(defglobal ?count = 0) ;Count of solutions
(defglobal ?print = NIL); This may be set to NIL to turn off printing.

(defrule ask-user-for-problem=slze
  "Ask the user how big the chessboard is."
  =>
  (printout t t "How many rows on the board? ")
  (bind ?n (read))
  (bind ?n 6) ; 6 queens
  (assert (problem-size ?n) (solution))
```

66
(bind ?count 0))

(defrule grow-solution
  "Extend a partial solution by adding another queen."
  (problem-size ?n)
  (solution $?x)
  => (if (> (length ?x) ?n) then
      (for ?i from 1 to ?n do
       (if (not (members ?i ?x)) then (assert (solution $?x ?i)))))))

The (not (members ?i ?x)) condition checks that no previous queen occupies column ?i.

(defrule prune-diagonal-attacks
  "This rule kills off solutions in which a newly added queen attacks a previous queen along a diagonal."
  (declare (salience 20)) ; High salience to kill bad solutions immediately
  (fact <- (solution $?x ?c))
  => (bind ?xlen (length ?x))
     (for ?i from ?xlen downto 1 do
      (if (= (- (+ ?xlen ?i) ?i) (abs (- ?c (nth ?x ?i)))) then
       (retract ?fact)))))

(defrule print-solution
  "This rule detects solutions and prints a chessboard showing the position of the queens."
  (problem-size ?n)
  (solution $?x:=(= (length ?x) ?n))
  => (bind ?count (+ ?count 1))
     (if ?print then
      (printout t t "Solution • ?count " t)
      (for ?row from 1 to ?n do
       (bind ?qrc (nth ?x ?row))
       (for ?column from 1 to ?n do
        (if (= ?qrc ?column) then (printout t " Q") else (printout t " -"))
       (printout t t)))))

(defrule print-total
  "This rule prints the total number of solutions."
  => (printout t t "Total solutions: • ?count t)"
V. Test Programs

This appendix includes the following ART-IM programs used to validate the ART/Ada prototype:

- **Sweeptop**: Contains about 30 rules that test the rule RHS.
- **Sweep2**: Contains about 270 rules that test the pattern/join network.
V.1 Sweeptop
This file contains a multitude of simple "top-level" command tests of ART/C. It is intended to be self-diagnostics: at least, it should either die horribly or complain if any part of ART/C is broken. BDC

If Dribble isn't working, or if its behavior has changed substantially, large parts of this program will blow up.

(defrule if-test **
=>
(printout t "IF tests..." t)
(if t then t else (printout t ">>> IF error 1." t))
(if nil then (printout t ">>> IF error 2." t)
else t))

(defrule bind-test-1 **
=>
(printout t "BIND tests..." t)
(bind ?d1 foo)
(if (not (equal ?d1 foo))
then (printout t ">>> BIND error 1." t))
)

(defrule bind-test-2 **
=>
(bind ?d1 "foo")
(if (not (equal ?d1 "foo"))
then (printout t ">>> BIND error 2." t))
)

(defrule bind-test-3 **
=>
(bind ?d1 12)
(if (not (equal ?d1 12))
then (printout t ">>> BIND error 3." t))
)

(defrule bind-test-4 **
=>
(bind ?d1 12.45)
(if (not (equal ?d1 12.45))
then (printout t ">>> BIND error 4." t))
)

Need a scratch defglobal for later use:
(defglobal ?scratch = "foo")

(defglobal ?b = "foo") .string
(defglobal ?c = 55) .integer
(defglobal ?d = 55.55) .float
(defglobal ?e = foo) .symbol

(defrule defglobal-test-1 **
=>
(printout t "DEFGLOBAL tests..." t)
(if (not (equal ?b "foo"))
then (printout t ">>> DEFGLOBAL error 1." t))
)

(defrule defglobal-test-2 **
(defrule defglobal-test-3
  =>
  (if (not (equal ?d 55))
      then (printout t ">>>>> DFGLOBAL error 3" t))
)

(defrule defglobal-test-4
  =>
  (if (not (equal ?e foo))
      then (printout t ">>>>> DFGLOBAL error 4" t))
)

(defrule print-test
  =>
  (printout t "Tests of PRINT *****" t)
  (print foo ) .symbol
  (print 12 ) .integer
  (print 12.466 ) .float
  (print "foo bar") .string
  (print (equal 1 1) ) .T
  (print (equal 1 2) ) .NIL

  (printout t "Tests of PRIN1 *****" t)
  (print 12.466 ) .float
  (printout t ) .to separate tests
  (print foo ) .symbol
  (print 12 ) .integer
  (print "foo bar") .string
  (print (equal 1 1) ) .T
  (print (equal 1 2) ) .NIL

  (printout t "Tests of PRINC *****" t)
  (printout t ) .to separate tests
  (print 12 456 ) .float
  (printout t ) .to separate tests
  (print foo ) .symbol
  (print 12 ) .integer
  (print "foo bar") .string
  (print (equal 1 1) ) .T
  (print (equal 1 2) ) .NIL

  (printout t "Tests of PRINTOUT *****" t)
  (printout t ) .to separate tests
  (printout t 12.456) .float
  (printout t ) .to separate tests
  (printout t foo) .symbol
  (printout t 12 t) .integer
  (printout t "foo bar" t) .string
  (printout t (equal l 1) t) .T
  (printout t (equal l 2) ) .NIL

  ; complex formatting commands
  (printout t ) .to separate tests
  (printout t "Tests of TERPRI *****" t)
  (printout t t "foo") .to separate tests
  (terpri)
  (printout t "bar" t)
(defrule eq-and-test
=>
(printout t "EQ and *
(if (not (eq foo foo))
   then (printout t ">>>> EQ error 1. * t)) .symbols

(if (not (eq "foo" "foo"))
   then (printout t ">>>> EQ error 2. * t)) .strings

(if (not (eq 12 12))
   then (printout t ">>>> EQ error 3. * t)) .integers

(if (not (eq 12.45 12.450))
   then (printout t ">>>> EQ error 4. * t)) .floats

;; sequences and facts will have to be tested through rules.
;; Not top-level.

;; There are all kinds of things that could be unequal.
;; I'll test a few of them...

(if (eq foo foot)
   then (printout t ">>>> EQ error 5. * t)) .symbols

(if (eq foo "foo")
   then (printout t ">>>> EQ error 7. * t)) .symbol and similar string

(if (eq "foo" "Foo")
   then (printout t ">>>> EQ error 8. * t)) .strings with capitalization differences

(if (eq 12 12.0)
   then (printout t ">>>> EQ error 9. * t)) .integers and = floats

;; EQUAL uses EQ for most tests, but it might be good to
;; exercise it anyway.
(printout t "EQUAL tests... * t)

(if (not (equal foo foo))
   then (printout t ">>>> EQUAL error 1. * t)) .symbols

(if (not (equal "foo" "foo"))
   then (printout t ">>>> EQUAL error 2. * t)) .strings

(if (not (equal 12 12))
   then (printout t ">>>> EQUAL error 3. * t)) .integers

(if (not (equal 12.45 12.450))
   then (printout t ">>>> EQUAL error 4. * t)) .floats

(if (equal foo foot)
   then (printout t ">>>> EQUAL error 6. * t)) .symbols

(if (equal foo "foo")
   then (printout t ">>>> EQUAL error 7. * t)) .symbol and similar string

(if (equal "foo" "Foo")
   then (printout t ">>>> EQUAL error 8. * t)) .strings with capitalization differences

(if (equal 12 12.0)
   then (printout t ">>>> EQUAL error 9. * t)) .integers and = floats

72
(defrule gentemp-test *
=>
(printout t "GENTEMP tests..." t)
  How do you test something that is defined as returning something
different every time you call it??
  (if (or (not (symbolp (gentemp)))
       (equal (gentemp) (gentemp)))
    then (printout t ">>>>> GENTEMP error 1 " t))
)

(defrule symbolp-test *
=>
(printout t "SYMBOLP tests..." t)

(if (not (symbolp foo)) ;symbol
    then (printout t ">>>>> SYMBOLP error 1." t))

(if (symbolp 12) ;integer
    then (printout t ">>>>> SYMBOLP error 2." t))

(if (symbolp 12.4) ;float
    then (printout t ">>>>> SYMBOLP error 3." t))

(if (symbolp "foo") ;string
    then (printout t ">>>>> SYMBOLP error 4." t))

(if (not (symbolp (eq t 1))) ;T
    then (printout t ">>>>> SYMBOLP error 6." t))

(if (not (symbolp (eq t 2))) ;NIL
    then (printout t ">>>>> SYMBOLP error 7." t))
)

(defrule stringp-test *
=>
(printout t "STRINGP tests..." t)

(if (stringp foo) ;symbol
    then (printout t ">>>>> STRINGP error 1." t))

(if (stringp 12) ;integer
    then (printout t ">>>>> STRINGP error 2." t))

(if (stringp 12.4) ;float
    then (printout t ">>>>> STRINGP error 3." t))

(if (not (stringp "foo")) ;string
    then (printout t ">>>>> STRINGP error 4." t))

(if (stringp (eq t 1))) ;T
    then (printout t ">>>>> STRINGP error 6." t))

(if (stringp (eq t 2))) ;NIL
    then (printout t ">>>>> STRINGP error 7." t))
)

(defrule numberp-test *
=>

(printout t "NUMBERP tests..." t)
(if (numberp foo) .symbol
   then (printout t ">>>>> NUMBERP error 1.* t))

(if (not (numberp 12)) .integer
   then (printout t ">>>>> NUMBERP error 2.* t))

(if (not (numberp 12.4)) .float
   then (printout t ">>>>> NUMBERP error 3.* t))

(if (numberp "foo") .string
   then (printout t ">>>>> NUMBERP error 4.* t))

(if (numberp (eq 1 1)) .T
   then (printout t ">>>>> NUMBERP error 6.* t))

(if (numberp (eq 1 2)) .NIL
   then (printout t ">>>>> NUMBERP error 7.* t))
)

.... ***************
(defrule not-test "
=> (printout t "NOT tests..." t)
(if (not foo) .symbol
   then (printout t ">>>>> NOT error 1.* t))

(if (not 12) .integer
   then (printout t ">>>>> NOT error 2.* t))

(if (not 12.4) .float
   then (printout t ">>>>> NOT error 3.* t))

(if (not "foo") .string
   then (printout t ">>>>> NOT error 4.* t))

(if (not (eq 1 1)) .T
   then (printout t ">>>>> NOT error 6.* t))

(if (not (eq 1 2)) .NIL
   then (printout t ">>>>> NOT error 7.* t))
)

.... ***************
(defrule and-or-test "
=> (printout t "AND OR tests..." t)
(if (and t t nil) .t
   then (printout t "AND error 1.*"))

(if (not (and t t)) .t
   then (printout t "AND error 2.*"))

(if (not (or nil t nil)) .t
   then (printout t "OR error 1.*"))

(if (or nil nil nil) .nil
   then (printout t "OR error 2.*"))
)

.... ***************
(defrule string-append-test "

74
(printout t "STRING-APPEND tests..." t)

(if (not (equal (string-append 'a 'b' 'c') 'abc'))
  then (printout t ">>>>> STRING-APPEND error 1..." t))

(defvar ceiling-test "
=>
(printout t "CEILING tests..." t)

(if (not (equal (ceiling -1000000000 1) -1000000000))
  then (printout t ">>>>> CEILING error 1" t))

(if (not (equal (ceiling -1000000000 1) 1000000000))
  then (printout t ">>>>> CEILING error 2..." t))

(if (not (equal (ceiling 0) 0))
  then (printout t ">>>>> CEILING error 3..." t))

(if (not (equal (ceiling -1) -1))
  then (printout t ">>>>> CEILING error 4..." t))

(if (not (equal (ceiling 1) 1))
  then (printout t ">>>>> CEILING error 5..." t))

(if (not (equal (ceiling -1 1)) -1))
  then (printout t ">>>>> CEILING error 6..." t))

(if (not (equal (ceiling 1 1) 2))
  then (printout t ">>>>> CEILING error 7..." t))
)

(defvar truncate-test "
=>
(printout t "TRUNCATE tests..." t)

(if (not (equal (truncate -1000000000 1) -1000000000))
  then (printout t ">>>>> TRUNCATE error 1..." t))

(if (not (equal (truncate -1000000000 1) 1000000000))
  then (printout t ">>>>> TRUNCATE error 2..." t))

(if (not (equal (truncate 0) 0))
  then (printout t ">>>>> TRUNCATE error 3..." t))

(if (not (equal (truncate -1) -1))
  then (printout t ">>>>> TRUNCATE error 4..." t))

(if (not (equal (truncate 1) 1))
  then (printout t ">>>>> TRUNCATE error 5..." t))

(if (not (equal (truncate -1 1) -1))
  then (printout t ">>>>> TRUNCATE error 6..." t))

(if (not (equal (truncate 1 1) 1))
  then (printout t ">>>>> TRUNCATE error 7..." t))
)

(defvar evenp-test "
=>
(printout t "EVENP tests..." t)
(if (not (evenp 2))
    then (printout t ">>>>> EVENP error 1." t))

(if (not (evenp -2))
    then (printout t ">>>>> EVENP error 2." t))

(if (evenp 1)
    then (printout t ">>>>> EVENP error 3." t))

(if (evenp -1)
    then (printout t ">>>>> EVENP error 4." t))
)

... **************************** *
(defrule oddp-test "
=>
(printout t "ODDP tests..." t)

(if (oddp 2)
    then (printout t ">>>>> ODDP error 1." t))

(if (oddp -2)
    then (printout t ">>>>> ODDP error 2." t))

(if (not (oddp 1))
    then (printout t ">>>>> ODDP error 3." t))

(if (not (oddp -1))
    then (printout t ">>>>> ODDP error 4." t))
)

... **************************** *
(defrule rem-test "
=>
(printout t "REM tests..." t)
; tests from Steele p. 217

(if (not (= (rem 13 4) 1))
    then (printout t ">>>>> REM error 1." t))

(if (not (= (rem -13 4) -1))
    then (printout t ">>>>> REM error 2." t))

(if (not (= (rem 13 -4) 1))
    then (printout t ">>>>> REM error 3." t))

(if (not (= (rem -13 -4) -1))
    then (printout t ">>>>> REM error 4." t))

(if (not (= (rem 13.4 1) 0.4))
    then (printout t ">>>>> REM error 5." t))

(if (not (= (rem -13.4 1) -0.4))
    then (printout t ">>>>> REM error 6." t))
)

... **************************** *
(defrule mod-test "
=>
(printout t "MOD tests..." t)
; tests from Steele, p. 217

(if (not (= (mod 13 4) 1)))
then (printout t ">>>>> MOD error 1" t))
(if (not (= (mod -13 4) 3))
then (printout t ">>>>> MOD error 2" t))
(if (not (= (mod 13 -4) -3))
then (printout t ">>>>> MOD error 3" t))
(if (not (= (mod -13 -4) -1))
then (printout t ">>>>> MOD error 4" t))
(if (not (= (mod 13.4 1) 0.4))
then (printout t ">>>>> MOD error 5" t))
(if (not (= (mod -13.4 1) 0.5))
then (printout t ">>>>> MOD error 6" t))

;: -----------------------------------------------
(defrule math-test **
=>
(printout t "MATH tests..." t)

;: Testing of library math functions is abbreviated to a single test
;: for each, mainly to prove that the function is present and is properly
;: linked with ART/C. There seems no point in trying to carefully
;: explore for singularities and precision when the functions are
;: beyond our reach, and will not have the same behavior from one site
;: to another. BDC
(if (not (= (max 3 2 1.5) 3))
then (printout t ">>>>> MATH error 3" t))
(if (not (= (min 3 2 1.5) 1.500000))
then (printout t ">>>>> MATH error 4" t))
(if (not (= (mod 6 3) 2))
then (printout t ">>>>> MATH error 5" t))
(if (not (= (+ 1 1.5) 2.5))
then (printout t ">>>>> MATH error 6" t))
(if (not (= (+ 1 0.5) 0.5))
then (printout t ">>>>> MATH error 7" t))
(if (not (= (+ 1 1.5) 1.5))
then (printout t ">>>>> MATH error 8" t))
(if (not (= (/ 1 2.0) 0.5))
then (printout t ">>>>> MATH error 9" t))
(if (not (= 1 1))
then (printout t ">>>>> MATH error 10" t))
(if (not (= 1.0 2))
then (printout t ">>>>> MATH error 20" t))

;: broken in 392 on VAX
(if (not (>= 2 1))
then (printout t ">>>>> MATH error 21" t))
(if (not (> 2 1))
then (printout t ">>>>> MATH error 22" t))
(if (not (< 1 2))
then (printout t ">>>>> MATH error 23" t))
(if (not (>= 1 2))
then (printout t ">>>>> MATH error 24" t))
(if (not (<= ?m 2))
  then (printout t "***** MATH error 25: " t))

(if (not (= (abs -?m) ?m))
  then (printout t "***** MATH error 27: " t))
)

(defglobal ?m 5) ;iterative counter
(defglobal ?n 0) ;accumulator

(defrule while-test **
  =>
  (printout t "WHILE tests...: " t)

  (while (> ?m 0) do
    (bind ?m (+ ?m 1)))
  )

(if (not (= ?n 5))
  then (printout t "***** WHILE error 1: " t))
)

(defrule prod-test **
  =>
  (printout t "PROGN tests...: " t)

  (if (not (equal (progn (bind ?m 10)
                         (bind ?n 20)
                         (= ?n 5))
               78)
    then (printout t "***** PROGN error 1: " t))
)

(defrule length-test-1
  "Bind a sequence to ?x"
  (fact ?x)
  =>
  (printout t "LENGTH$ tests...: t)

  (if (not (= (length$ ?x) 5))
    then (printout t "***** LENGTH$ error 1: " t))
)

(defrule nth-test-1
  "Bind a sequence to ?o"
  (fact ?o)
  =>
  (printout t "NTH$ tests...: t)

  (if (not (equal (nth$ ?o 3) three))}
(defrule position-test-1
  "Bind a sequence to ?o"
  (fact $?o)
  =>
  (printout t "POSITION$ tests " t)
  (if (not (= (POSITION$ four ?o) 4))
      then (printout t ">>>>> POSITION$ error 1. t"))
)

(defrule member-test-1
  "Bind a sequence to ?o"
  (fact $?o)
  =>
  (printout t "MEMBER$ tests... t")
  (if (member$ six ?o)
      then (printout t ">>>>> MEMBER$ error 1. t"))
  (if (not (member$ five ?o))
      then (printout t ">>>>> MEMBER$ error 2. t"))
)
V.2 Sweep2
(deffacts initial-1 "Facts to match the rules in SUITE-RULE1 ART"
(test-case sdl-1 test-case)
(test-case sdl-2 "a red flag")
(test-case sdl-3 1)
(test-case sdl-4 :00 321)
(test-case sdl-5)
(test-case sdl-6 green)
(test-case sdl-7 "green")
(test-case sdl-8 green green)
(test-case sdl-9)
(test-case sdl-10 green)
(test-case sdl-11 "green")
(test-case sdl-12 green green)
(sdl-13 sdl-13)
(sdl-14 sdl-14 green)
(sdl-15 sdl-15 "green")
(sdl-16 sdl-16 green green)
(sdl-17 sdl-17)
(sdl-18 sdl-18 green)
(sdl-19 sdl-19 "green")
(sdl-20 sdl-20 green green)
(sdl-21 sdl-21 blue red green)
(sdl-22 sdl-22 red)
(sdl-23 red sdl-23)
(sdl-24 sdl-24)
(sdl-25 sdl-25 data sdl-25)
(sdl-26 blue fun blue)
(sdl-27 blue fun get)
(sdl-28 blue blue blue)
(sdl-29 blue fun "blue")
(sdl-30 red blue green)
(sdl-30 purple blue green)
(sdl-31 red blue green)
(sdl-31 purple blue brown)
(sdl-32 red)
(sdl-33 green)
(sdl-34 red)
(sdl-35 blue)
(sdl-36 green)
(sdl-37 red)
(sdl-38 blue)
(sdl-39 green)
(sdl-40 red)
(sdl-41 blue)
(sdl-42 get)
(sdl-43 green)
(sdl-44 red)
(sdl-45 green)
(sdl-46 red)
(sdl-47 blue)
(sdl-48 green)
(sdl-49 a red)
(sdl-49 b red)
(sdl-50 a red)
(sdl-50 b green)
(sdl-51 a blue)
(sdl-51 b blue)
(sdl-52 a red)
(sdl-52 b red)
(sdl-53 a red)
(sdl-53 b blue)
(sdl-54 a green)
(sdl-54 b green)

(sdl-55 2)
(sdl-56 red)
(sdl-57 2)
(sdl-58 2)
(sdl-59 *red*)

(sdl-60 2)
(sdl-61 *red*)
(sdl-62 2)
(sdl-63 *red*)

(sdl-64 4)
(sdl-64 9)

(sdl-65 data 6)
(sdl-65 value 9)

(sdl-66 1 4.00 7.00)
(sdl-66 2 5.00 9.00)

(sdl-67 data1 3)
(sdl-67 data2 5)

(sdl-68 data1 9)
(sdl-68 data2 5)

(sdl-69 data1 4)
(sdl-69 data2 4)

(sdl-70 data1 red)
(sdl-70 data2 5)

(sdl-71 data1 *4*)
(sdl-71 data2 4)

(sdl-72 data1 red)
(sdl-72 data2 5)

(sdl-73 data1 red)
(sdl-73 data2 5)

(sdl-74)
(sdl-75)
(sdl-76)
(sdl-77)
(sdl-78)
(sdl-78 1 5)
(sdl-78 2 8)
(sdl-79 a b c d e f g)
(sdl-80 a b c d e f g)
(sdl-81 a b c d e f g)

(sdl-108 10)
(sdl-109 3)
(deffacts initial-1-sequences "Facts to match the rules in SUITE-RULE1.ART"
(test-case (sdl-1 test-case))
(test-case (sdl-2 "a red flag"))
(test-case (sdl-3 1))
(test-case (sdl-4 (100 321)))
(test-case (sdl-5))
(test-case (sdl-5 green))
(test-case sdl-7 ("green"))
(test-case (sdl-8 green green))
(test-case (sdl-9))
(test-case sdl-10 (green))
(test-case sdl-11 ("green"))
(test-case (sdl-12 green green))
(sdl-13 (sdl-13))
(sdl-14 (sdl-14 (green)))
(sdl-15 (sdl-15 ("green")))
(sdl-16 (sdl-16 (green green))
(sdl-17 (sdl-17))
(sdl-18 (sdl-18 green))
(sdl-19 (sdl-19 ("green")))
(sdl-20 (sdl-20 green (green)))
(sdl-21 (sdl-21 blue red green))
(sdl-22 (sdl-22 red))
(sdl-23 (red sdl-23))
(sdl-24 (sdl-24))
(sdl-25 (sdl-25 data sdl-25))
(sdl-26 (blue fun blue))
(sdl-27 (blue fun get))
(sdl-28 (blue blue blue))
(sdl-29 (blue fun ("blue")))
(sdl-30 (red (blue green)))
(sdl-31 (purple (blue green)))
(sdl-31 (red blue green))
(sdl-31 (purple blue brown))
(sdl-32 (red))
(sdl-33 (green))
(sdl-34 (red))
(sdl-35 (blue))
(sdl-36 (green))
(sdl-37 (red))
(sdl-38 (blue))
(sdl-39 (green))
(sdl-40 (red))
(sdl-41 (blue))
(sdl-42 (get))
(sdl-43 (green))
(sdl-44 (red))
(sdl-45 (green))
(sdl-46 (red))
(sdl-47 (blue))
(sdl-48 (green))
(sdl-49 (a (red)))
(sdl-49 (b (red)))
(sdl-50 (a (red)))
(sdl-50 (b (green)))
(sdl-51 (a (blue)))
(sdl-51 (b (blue)))
(sd1-52 (a (red)))
(sd1-52 (b (red)))
(sd1-53 (a (red)))
(sd1-53 (b (blue)))
(sd1-54 (a (green)))
(sd1-54 (b (green)))

(sd1-55 (2))
(sd1-56 (red))
(sd1-57 (2))
(sd1-58 (2))
(sd1-59 ("red"))

(sd1-60 (2))
(sd1-61 ("red"))
(sd1-62 (2))
(sd1-63 ("red"))

(sd1-64-seq (4))
(sd1-64-seq (9))

(sd1-65 (data 5))
(sd1-65 (value 9))

(sd1-66 (1 4.00 7.00))
(sd1-66 (2 5.00 9.00))

(sd1-67 (data1 (8)))
(sd1-67 (data2 (5)))

(sd1-68 (data1 (9)))
(sd1-68 (data2 (5)))

(sd1-69 (data1 (4)))
(sd1-69 (data2 (4)))

(sd1-70 (data1 red))
(sd1-70 (data2 5))

(sd1-71 (data1 "4"))
(sd1-71 (data2 4))

(sd1-72 (data1 red))
(sd1-72 (data2 5))

(sd1-73 (data1 red))
(sd1-73 (data2 5))

(sd1-74 (1))
(sd1-75 (1))
(sd1-76 (1))
(sd1-77 (1))
(sd1-78 (1))
(sd1-78 (1 5))
(sd1-78 (2 8))
(sd1-79 (a b c d e f g))
(sd1-80 (a b c d e f g))
(sd1-81 (a b c d e f g))

(sd1-108 (10))
(sd1-109 (3))

84
defacts initial-2 **
  (sdl-1:15 to be foo bar to be)
  (sdl-1:16 a e i o u)
  (sdl-1:17 fluff mug bump bleet lolita)

... go hiking or skiing

(non-working sdl-1:18)
(weather sdl-1:18 major-blizzard)

(non-working sdl-1:19)
(drive sdl-1:19 two-wheel)

(non-working sdl-1:20)
(weather sdl-1:20 major-blizzard)
(drive sdl-1:20 two-wheel)
(working sdl-1:20 weekend)

... wild card

(sdl-1:21 one one)
(sdl-1:22 one one one)
(sdl-1:23 6 5 4 5)
(sdl-1:24 a r t h u r)
(sdl-1:25 a rose is a rose)
(sdl-1:26 to be or not to be)
(sdl-1:27 frank)
(sdl-1:28 1 2)
(sdl-1:29 love is in bloom)
(sdl-1:30 a rose is a rose)

... sir cat

(sdl-1:31 "foo")
(sdl-1:33 first second)
(sdl-1:34 republicans fox jones nixon williams harvey)
(sdl-1:34 quakers pallas sanchez stone nixon fregge)
(sdl-1:35 yellow)
(sdl-1:35 green)
(sdl-1:36 yellow)
(sdl-1:36 green)
(sdl-1:37 yellow)
(sdl-1:38 green)
(sdl-1:39 -12 12)
(sdl-1:40 1 1)
(sdl-1:40 1 2)
(sdl-1:41 1 1)
(sdl-1:41 1 2)
(sdl-1:42 cold hot)
(sdl-1:43 green yellow red blue white)
(sdl-1:44 brother-of walter daniel)
(sdl-1:44 child-of walter david)
(sdl-1:44 sex-of david male)
(sdl-1:44 child-of walter jane)
(sdl-1:44 sex-of jane female)
(sdl-1:45-1 1st red white blue)
(sdl-1:45 red)
(sdl-1:45 white)
(sdl-1:45 blue)
(sdl-1:46-seats 8 9 11 14 3)
(sdl-1:46-names tom carol fred alex)
(sdl-1:47 donors start fred)
(sdl-1:47 donors fred john)
(sdl-1:47 donors john mike)
(sdl-1:47 donors mike finish)
(sdl-166 paul yes)
(sdl-166 paul 1.5)
(sdl-166 1.5 paul)
(sdl-166 yes paul)
(sdl-166 I think paul said yes)
(sdl-166 paul did say 1.5)
(sdl-166 1.5 says paul)
(sdl-166 yes says paul)
(SDL-167 YES FOR PAUL AND BRAD)
(SDL-167 YES FOR PAUL BRAD AND BILL)
(SDL-167 YES PAUL BRAD)
(SDL-167 PAUL YES BRAD)
(SDL-167 PAUL BRAD YES)
(SDL-167 PAUL BRAD 1.5)
(sdl-167 I 5 BRAD PAUL MARK)
(SDL-167 BRAD AND PAUL SAY YES)
(SDL-167 BRAD SAYS YES AND SO DOES PAUL)
(SDL-168 NO says BRAD and YES says PAUL)
(SDL-168 NO for BRAD YES for PAUL)
(SDL-168 THE ANSWER FOR BRAD AND MARK IS NO and PAUL IS YES)
(SDL-168 BRAD MARK ARE NO and PAUL IS YES 1.5)
(SDL-168 THE ANSWER FOR BRAD AND MARK IS NO and PAUL IS 1.5)
(SDL-168 BRAD is unknown AND PAUL IS 1.5)
(SDL-169 NO says BRAD but not PAUL)
(SDL-169 NO for BRAD but not for PAUL)
(SDL-169 THE ANSWER FOR BRAD AND MARK IS NO but not PAUL)
(SDL-169 BRAD MARK ARE NO but not PAUL)
(SDL-169 paul complains of vertigo and faintness)
(SDL-169 age paul 30)
(SDL-170 age brad 18)
(SDL-172 major-complaint paul drowsiness)
(SDL-172 recent-head-injury paul no)
(SDL-172 confusion-mild paul yes)
(SDL-172 major-complaint brad confusion)
(SDL-172 major-complaint brad vertigo)
(SDL-173 case1 450 550 3)
(SDL-173 case2 440 560 2)
(SDL-173 case3 440 440 1)
(SDL-174 case dummy dummy dummy 450 dummy 550 dummy 3)
(SDL-174 case dummy 440 foo foo 440 foo 560 foo 2)
(SDL-174 case3 bar bar bar 440 bar 440 bar 1)
(SDL-175 pallet 5)
(SDL-175 pallet 4)
(SDL-175 options 1000 3000 4000 5024)
(SDL-175 case 3024)
(SDL-175 case 5000)
(SDL-175 case 5024)
(SDL-175 case 4000)
(SDL-176 options 1000 3000 5000)
(SDL-176 case 5000)
(SDL-177 options 1000 3000 4000)
(SDL-177 case 5000)
(SDL-178 1)
(SDL-179 options 1000 3000 5000)
(SDL-179 case 5000)
(SDL-180 options 1000 3000 4000)
(SDL-180 case 5000)
(SDL-181 5000)
(SDL-181 6000)
(SDL-182 *est* *pst*)
(SDL-182 *pst* *est*)
(SDL-182 *est* *est*)
(deffacts initial-2-sequences ""
  (sdl-115 ((to be) (foo bar) (to be)))
  (sdl-116 (a e i o u))
  (sdl-117 (fluff mug bump bleet lolita))
  : go hiking or skiing
  (non-working (sdl-118))
  (weather (sdl-118 major-blizzard))
  (non-working (sdl-119))
  (drive (sdl-119 two-wheel))
  (non-working (sdl-120))
  (weather (sdl-120 major-blizzard))
  (drive (sdl-120 two-wheel))
  (working (sdl-120 weekend))
  : wild card
  (sdl-121 (one one))
  (sdl-122 (one one one))
  (sdl-123 (6 5 4 5))
  (sdl-124 (a r t h u r))
  (sdl-125 (a rose is a rose))
  (sdl-126 (to be or not to be))
  (sdl-127 (frank))
  (sdl-128 (1 2))
  (sdl-129 (love in bloom))
  (sdl-130 (a rose is a rose))
  : str_cat
  (sdl-131 ('foo*))
  (sdl-133 (first second))
  (sdl-134 (republicans fox jones nixon williams harvey))
  (sdl-134 (quakers pallas sanchez stone nixon frege))
  (sdl-135 (yellow))
  (sdl-135 (green))
  (sdl-136 (yellow))
  (sdl-136 (green))
  (sdl-137 (yellow))
)
(sdl-138 (green))
(sdl-139 (-12 (12)))
(sdl-140 (1 1))
(sdl-140 (1 2))
(sdl-141 (1 1))
(sdl-141 (1 2))
(sdl-142 (cold hot))
(sdl-143 (green yellow red blue white))
(sdl-144 (brother-of walter daniel))
(sdl-144 (child-of walter david))
(sdl-144 (sex-of david male))
(sdl-144 (child-of walter jane))
(sdl-144 (sex-of jane female))
(sdl-145-list (red white blue))
(sdl-145 (red))
(sdl-145 (white))
(sdl-145 (blue))
(sdl-146-seats (8 9 11 14 3))
(sdl-146-names (tom carol fred alex))
(sdl-147 (donors start fred))
(sdl-147 (donors fred john))
(sdl-147 (donors john mike))
(sdl-147 (donors mike finish))
(sdl-147 (donation fred 7.0))
(sdl-147 (donation john 0 0))
(sdl-147 (donation mike 0 5))
(sdl-147-sum (donors start 0))
(sdl-148 (donor-list fred john mike))
(sdl-148 (donation fred 7.0))
(sdl-148 (donation john 10 0))
(sdl-148 (donation mike 0 5))
(sdl-148-sum (donors start 0))
(sdl-149 (inventory shirt-11 7.00))
(sdl-149 (inventory pants-9 10.00))
(sdl-149 (inventory belt-14 2.50))
(sdl-149 (current-sum 0))
(sdl-150 (inventory shirt-11 7.00))
(sdl-150 (inventory pants-9 10.00))
(sdl-150 (inventory belt-14 2.50))
(sdl-150 (current-count 0))
(sdl-151 (quarter 10))
(sdl-151 (dime 8))
(sdl-151 (nickel 6))
(sdl-151 (penny 4))
(sdl-152 (quarter 10))
(sdl-152 (dime 8))
(sdl-152 (nickel 6))
(sdl-152 (penny 4))
(sdl-153 (yes))
(sdl-153 (no))
(sdl-153 (unknown))
(sdl-153 (1))
(sdl-154 (foo))
(sdl-154 (1))
(sdl-154 (2))
(sdl-154 (3))
(sdl-155 (foo))
(sdl-155 (1))
(sdl-155 (2))
(sdl-155 (3))
(sdl-156 ("String"))
(sdl-157 ("String"))
(sdl-158 ("String"))
(sdl-159 (yes))
ART/ADA DESIGN PROJECT - PHASE I

(sdl-159 (no))
(sdl-159 (unknown))
(sdl-159 (I))
(sdl-160 (yes))
(sdl-160 (no))
(sdl-160 (unknown))
(sdl-160 (I))
(sdl-161 (foo))
(sdl-161 (I))
(sdl-162 (no))
(sdl-162 (I))
(sdl-162 (2))
(sdl-162 (3))

(sdl-163 ("overflow"))
(sdl-164 (input "overflow"))
(sdl-164 (list "yes" "no" "unknown" "overflow" "inference"))
(sdl-165 (\$345.89))
(patient-name (paul))
(patient-name (brad))
(sdl-166 (paul yes))
(sdl-166 (paul 1.5))
(sdl-166 (1.5 paul))
(sdl-166 (yes paul))
(sdl-166 (I think paul said yes))
(sdl-166 (paul did say 1.5))
(sdl-166 (1.5 says paul))
(sdl-166 (yes says paul))
(SDL-167 (YES FOR PAUL AND BRAD))
(SDL-167 (YES FOR PAUL BRAD AND BILL))
(SDL-167 (YES PAUL BRAD))
(SDL-167 (PAUL YES BRAD))
(SDL-167 (PAUL BRAD YES))
(SDL-167 (PAUL BRAD 1.5))
(sdl-167 (1.5 BRAD PAUL MARK))
(sdl-167 (BRAD AND PAUL SAY YES))
(sdl-167 (BRAD SAYS YES AND SO DOES PAUL))
(sdl-168 (YES FOR PAUL AND NO FOR BRAD AND MARK))
(SDL-168 (NO says BRAD and YES says PAUL))
(SDL-168 (NO for BRAD YES for PAUL))
(SDL-168 (THE ANSWER FOR BRAD AND MARK IS NO and PAUL IS YES))
(SDL-168 (BRAD MARK ARE NO and PAUL IS YES TOO))
(SDL-168 (THE ANSWER FOR BRAD AND MARK IS NO and PAUL IS 1.5))
(SDL-168 (BRAD is unknown AND PAUL is 1.5))
(SDL-169 (NO says BRAD but not PAUL))
(SDL-169 (NO for BRAD but not for PAUL))
(SDL-169 (THE ANSWER FOR BRAD AND MARK IS NO but not PAUL))
(SDL-169 (BRAD MARK ARE NO but not PAUL))
(sdl-170 (paul complains of vertigo and faintness))
(sdl-170 (brad has a headache))
(seq-age paul (30))
(seq-age brad (18))
(sdl-172 (major-complaint paul drowsiness))
(sdl-172 (recent-head-injury paul no))
(sdl-172 (confusion-mild paul yes))
(sdl-172 (major-complaint brad confusion))
(sdl-172 (major-complaint brad vertigo))
(sdl-173 (case1 450 560 3))
(sdl-173 (case2 440 560 2))
(sdl-173 (case3 440 440 1))
(sdl-174 (case1 dummy dummy dummy 450 dummy 550 dummy 3))
(sdl-174 (case2 foo foo foo 440 foo 560 foo 2))
(defrule print-fail
  (declare (salience -100))
  (fail ?test-case)
=>
  (printout t t "------- Runtime Error ✗ ?test-case))

(defrule sdl-rule-1 "Match a literal symbol"
  (or (not (test-case sdl-1 TEST-CASE))
      (not (test-case (sdl-1 TEST-CASE)))))
=>
  (assert (fail sdl-1)))

(defrule sdl-rule-2 "Match a literal string"
(or (not (test-case sdl-2 "a red flag"))
    (not (test-case (sdl-2 "a red flag"))))
=>
  (assert (fail sdl-2)))

(defrule sdl-rule-3 "Match an integer"
  (or (not (test-case sdl-3 1))
      (not (test-case (sdl-3 1))))
=>
  (assert (fail sdl-3)))

(defrule sdl-rule-4 "Don't match a float that is almost equal"
  (or (test-case sdl-4 100.3)
      (test-case (sdl-4 (100.3))))
=>
  (assert (fail sdl-4)))

(defrule sdl-rule-5 "Don't let ? match nothing"
  (or (test-case sdl-5 ?)
      (test-case (sdl-5 ?)))
=>
  (assert (fail sdl-5)))

(defrule sdl-rule-6 "? must match a symbol"
  (or (not (test-case sdl-6 ?))
      (not (test-case (sdl-6 ?))))
=>
  (assert (fail sdl-6)))

(defrule sdl-rule-7 "? must match a string"
  (or (not (test-case sdl-7 ?))
      (not (test-case sdl-7 (?))))
=>
  (assert (fail sdl-7)))

(defrule sdl-rule-8 "? must not match TWO symbols"
  (or (test-case sdl-8 ?)
      (test-case (sdl-8 ?)))
=>
  (assert (fail sdl-8)))

(defrule sdl-rule-9 "$? must match nothing"
  (or (not (test-case sdl-9 $?))
      (not (test-case sdl-9 ($?))))
=>
  (assert (fail sdl-9)))

(defrule sdl-rule-10 "$? must match a single symbol"
  (or (not (test-case sdl-10 $?))
      (not (test-case sdl-10 ($?))))
=>
  (assert (fail sdl-10)))

(defrule sdl-rule-11 "$? must match a single string"
  (or (not (test-case sdl-11 $?))
      (not (test-case sdl-11 ($?))))
=>
  (assert (fail sdl-11)))

(defrule sdl-rule-12 "$? must match two symbols"
  (or (not (test-case sdl-12 $?))
      (not (test-case (sdl-12 $?))))
=>
  (assert (fail sdl-12)))
(defrule sdl-rule-13 "matches symbol, $ matches nothing"
    (or (not (sdl-13 ? $)))
    (not (sdl-13 (?) $))))
=>
(assert (fail sdl-13)))

(defrule sdl-rule-14 "and $ each match a single symbol"
    (or (not (sdl-14 ? $))
        (not (sdl-14 (? ($)))))
=>
(assert (fail sdl-14)))

(defrule sdl-rule-15 "matches symbol, $ matches string"
    (or (not (sdl-15 ? $))
        (not (sdl-15 (? ($)))))
=>
(assert (fail sdl-15)))

(defrule sdl-rule-16 "$ matches one symbol, $ matches two symbols"
    (or (not (sdl-16 ?) $)
        (not (sdl-16 (? ($)))))
=>
(assert (fail sdl-16)))

(defrule sdl-rule-17 "$ matches nothing, ? matches symbol"
    (or (not (sdl-17 ? $))
        (not (sdl-17 (? ($)))))
=>
(assert (fail sdl-17)))

(defrule sdl-rule-18 "$ matches symbol, ? matches symbol"
    (or (not (sdl-18 $ ?))
        (not (sdl-18 (? $)))))
=>
(assert (fail sdl-18)))

(defrule sdl-rule-19 "$ matches symbol, ? matches string"
    (or (not (sdl-19 ? $))
        (not (sdl-19 (? $)))))
=>
(assert (fail sdl-19)))

(defrule sdl-rule-20 "$ matches two symbols, ? matches one symbol"
    (or (not (sdl-20 $ ?))
        (not (sdl-20 (? $)))))
=>
(assert (fail sdl-20)))

(defrule sdl-rule-21 ""
    (or (not (sdl-21 ? $))
        (not (sdl-21 (?) $)))
=>
(assert (fail sdl-21)))

(defrule sdl-rule-22 ""
    (or (not (sdl-22 ? $))
        (not (sdl-22 (?) $)))
=>
(assert (fail sdl-22)))
(assert (fail sdl-22)))

(defrule sdl-rule-23
  (or (not (sdl-23 $?)
    sdl-23
    $?))
  (not (sdl-23 ( $?)
    sdl-23
    $?))))
=>
(assert (fail sdl-23)))

(defrule sdl-rule-24
  (or (not (sdl-24 $?)
    sdl-24
    $?))
  (not (sdl-24 ( $?)
    sdl-24
    $?))))
=>
(assert (fail sdl-24)))

(defrule sdl-rule-25
  (or (not (sdl-25 $?)
    sdl-25
    $?))
  (not (sdl-25 ( $?)
    sdl-25
    $?))))
=>
(assert (fail sdl-25)))

(defrule sdl-rule-26
  (or (not (sdl-26 ?x
    fun
    ?x))
  (not (sdl-26 (?x
    fun
    ?x))))))
=>
(assert (fail sdl-26)))

(defrule sdl-rule-27
  (or (sdl-27 ?x
    fun
    ?x)
  (sdl-27 (?x
    fun
    ?x)))
=>
(assert (fail sdl-27)))

(defrule sdl-rule-28
  (or (sdl-28 ?x
    fun
    ?x)
  (sdl-28 (?x
    fun
    ?x)))
=>
(assert (fail sdl-28)))
(defrule sdl-rule-29 **
  (or (sdl-29 ?x
    fun
  (?x))
  (sdl-29 (?x
    fun
  (?x))))
=>
  (assert (fail sdl-29)))

(defrule sdl-rule-30 **
  (sdl-30 red $?x)
  (sdl-30 (red $?x))
  (sdl-30 purple $?x)
  (sdl-30 (purple $?x))
=>
  (assert (success sdl-30))
  (assert (success (sdl-30)))

(defrule sdl-rule-30-1 **
  (declare (salience -t))
  (or (not (success sdl-30))
  (not (success (sdl-30))))
=>
  (assert (fail sdl-30)))

(defrule sdl-rule-31 **
  (sdl-31 red $?x)
  (sdl-31 (red $?x))
  (sdl-31 purple $?x)
  (sdl-31 (purple $?x))
=>
  (assert (fail sdl-31)))

(defrule sdl-rule-32 **
  (sdl-32 "red")
  (sdl-32 ("red"))
=>
  (assert (fail sdl-32)))

(defrule sdl-rule-33 **
  (or (not (sdl-33 "red")
  (not (sdl-33 ("red")))))
=>
  (assert (fail sdl-33)))

(defrule sdl-rule-34 **
  (or (not (sdl-34 red|blue))
  (not (sdl-34 (red|blue))))
=>
  (assert (fail sdl-34)))

(defrule sdl-rule-35 **
  (or (not (sdl-35 red|blue))
  (not (sdl-35 (red|blue))))
=>
  (assert (fail sdl-35)))

(defrule sdl-rule-36 **
  (sdl-36 red|blue)
  (sdl-36 (red|blue))
=>
  (assert (fail sdl-36)))
(defrule sdl-rule-37 **
  (sdl-37 "red\&blue")
  (sdl-37 "red\&blue")
=>
  (assert (fail sdl-37)))

(matches red (no match))

(defrule sdl-rule-38 **
  (sdl-38 "red\&blue")
  (sdl-38 "red\&blue")
=>
  (assert (fail sdl-38)))

(matches blue (no match))

(defrule sdl-rule-39 **
  (or (not (sdl-39 "red\&blue"))
      (not (sdl-39 "red\&blue")))
=>
  (assert (fail sdl-39)))

(matches green)

(defrule sdl-rule-40 **
  (sdl-40 "red\&blue\{get}"
  (sdl-40 "red\&blue\{get}"
=>
  (assert (fail sdl-40)))

(matches red (no match))

(defrule sdl-rule-41 **
  (or (not (sdl-41 "red\&blue\{get}"
      (not (sdl-41 "red\&blue\{get}"
=>
  (assert (fail sdl-41)))

(matches blue)

(defrule sdl-rule-42 **
  (or (not (sdl-42 "red\&blue\{get}"
      (not (sdl-42 "red\&blue\{get}"
=>
  (assert (fail sdl-42)))

(matches get)

(defrule sdl-rule-43 **
  (sdl-43 "red\&blue\{get}"
  (sdl-43 "red\&blue\{get}"
=>
  (assert (fail sdl-43)))

(matches green (no match))

(defrule sdl-rule-44 **
  (sdl-44 ?x\&red"
  (sdl-44 ?x\&red"
=>
  (assert (fail sdl-44)))

(matches red (no match))

(defrule sdl-rule-45 **
  (or (not (sdl-45 ?x\&red"
      (not (sdl-45 ?x\&red"
=>
  (assert (fail sdl-45)))

(matches green)

(defrule sdl-rule-46 **
  (or (not (sdl-46 ?x\&red\{blue}"
      (not (sdl-46 ?x\&red\{blue}"
=>
  (assert (fail sdl-46)))

(matches red)

(defrule sdl-rule-47 **
  (or (not (sdl-47 ?x\&red\{blue}"
      (not (sdl-47 ?x\&red\{blue}"
=>
  (assert (fail sdl-47)))

(matches blue)
=>
(assert (fail sdl-47)))

(defrule sdl-rule-48 ''
(sdl-48 ?x:red|blue)
(sdl-48 (?x:red|blue))
=>
(assert (fail sdl-48)))

(defrule sdl-rule-49 ''
(sdl-49 a "x")
(sdl-49 (a "x"))
(sdl-49 b ?x:red)
(sdl-49 (b (?x:red)))
=>
(assert (fail sdl-49)))

(defrule sdl-rule-50 ''
(sdl-50 a ?x)
(sdl-50 (a ?x))
(sdl-50 b ?x:red)
(sdl-50 (b (?x:red)))
=>
(assert (fail sdl-50)))

(defrule sdl-rule-51 ''
(sdl-51 a ?x)
(sdl-51 (a ?x))
(sdl-51 b ?x:red)
(sdl-51 (b (?x:red)))
=>
(assert (success sdl-51))
(assert (success (sdl-51))))

(defrule sdl-rule-51-1 'check on sdl-rule-51''
(declare (salience -1))
(or (not (success sdl-51))
(not (success (sdl-51))))
=>
(assert (fail sdl-51)))

(defrule sdl-rule-52 ''
(sdl-52 a ?x)
(sdl-52 (a ?x))
(sdl-52 b ?x:red|blue)
(sdl-52 (b (?x:red|blue)))
=>
(assert (success sdl-52))
(assert (success (sdl-52))))

(defrule sdl-rule-52-1 ''
(declare (salience -1))
(or (not (success sdl-52))
(not (success (sdl-52))))
=>
(assert (fail sdl-52)))

(defrule sdl-rule-53 ''
(sdl-53 a (?x))
(sdl-53 (a ?x))
(sdl-53 b ?x:red|blue)
(sdl-53 (b (?x:red|blue)))
=>
(matches green (no match))
(assert (fall sdl-53))

(defrule sdl-rule-54 **
  (sdl-54 a ?x) ;matches green
  (sdl-54 (a (?x)))
  (sdl-53 b ?x: red|blue)
  (sdl-54 (b (?x: red|blue)))) ;matches green (no match)

=>

(assert (fall sdl-54)))

(defrule sdl-rule-55 **
  (or (not (sdl-55 ?x: (numberp ?x))) ;matches 2
      (not (sdl-55 (?x: (numberp ?x)))))

=>

(assert (fall sdl-55)))

(defrule sdl-rule-56 **
  (or (sdl-56 ?x: (numberp ?x)) ;matches red (no match)
      (sdl-56 (?x: (numberp ?x))))

=>

(assert (fall sdl-56)))

(defrule sdl-rule-57 **
  (or (sdl-57 ?x: (numberp ?x):(oddp ?x)) ;matches 2 (no match)
      (sdl-57 (?x: (numberp ?x):(oddp ?x))))

=>

(assert (fall sdl-57)))

(defrule sdl-rule-58 **
  (or (sdl-58 ?x: (stringp ?x)) ;matches 2 (no match)
      (sdl-58 (?x: (stringp ?x))))

=>

(assert (fall sdl-58)))

(defrule sdl-rule-59 **
  (or (not (sdl-59 ?x: (stringp ?x))) ;matches "red"
      (not (sdl-59 (?x: (stringp ?x)))))

=>

(assert (fall sdl-59)))

(defrule sdl-rule-60 **
  (or (not (sdl-60 (= (+ 1 1))))) ;matches 2
      (not (sdl-60 (= (+ 1 1))))))

=>

(assert (fall sdl-60)))

(defrule sdl-rule-61 **
  (or (sdl-61 (= (+ 1 1))) ;matches "red" (no match)
      (sdl-61 (= (+ 1 1)))))

=>

(assert (fall sdl-61)))

(defrule sdl-rule-62 **
  (or (sdl-62 = (string-append "re" "d"))) ;matches 2 (no match)
      (sdl-62 = (string-append "re" "d"))))

=>

(assert (fall sdl-62)))

(defrule sdl-rule-63 **
  (or (not (sdl-63 = (string-append "re" "d"))) ;matches "red"
      (not (sdl-63 (= (string-append "re" "d"))))))

=>

(assert (fall sdl-63)))
(defrule sdl-rule-64
  \(\text{sdl-64 ?y}\)
  \(\text{sdl-64 ?x} = (+ 5 ?y)\), \(\text{?x} = (- 12 ?y)\)
=>
  (assert (success sdl-64)))

(defrule sdl-rule-64-seq
  \(\text{sdl-64-seq ?y}\)
  \(\text{sdl-64-seq ?x} = (+ 5 ?y), \text{?x} = (- 12 ?y)\)
=>
  (assert (success sdl-64-seq)))

(defrule sdl-rule-64-2
  (declare (salience -I))
  (or (not (success sdl-64))
      (not (success sdl-64-seq)))
=>
  (assert (fail sdl-64)))

(defrule sdl-rule-65
  \(\text{sdl-65 data ?x}\)
  \(\text{sdl-65 (data ?x)}\)
  \(\text{sdl-65 value ?y}\)
  \(\text{sdl-65 value ?y}\)
  \(\text{test} \geq (- ?y ?x) 3)\)
=>
  (assert (success sdl-65))
  (assert (success (sdl-65))))

(defrule sdl-rule-65-2
  (declare (salience -I))
  (or (not (success sdl-65))
      (not (success (sdl-65))))
=>
  (assert (fail sdl-65)))

(defrule sdl-rule-66
  \(\text{sdl-66 ?a ?x1 ?y1}\)
  \(\text{sdl-66 ?b ?a ?x2 ?y2}\)
  \(\text{sdl-66 ?b ?a ?x2 ?y2}\)
  \(\text{test} < 0 \text{(/ (- ?y2 ?y1) (- ?x2 ?x1))}\)
=>
  (assert (success sdl-66))
  (assert (success (sdl-66))))

(defrule sdl-rule-66-2
  (declare (salience -I))
  (or (not (success sdl-66))
      (not (success (sdl-66))))
=>
  (assert (fail sdl-66)))

(defrule sdl-rule-67
  \(\text{sdl-67 data1 \?y}\)
  \(\text{sdl-67 (data1 \?y)}\)
  \(\text{sdl-67 data2 \?x} \geq \?x \?y)\)
  \(\text{sdl-67 (data2 \?x \geq \?x \?y)}\)
=>
  (assert (success sdl-67))
  (assert (success (sdl-67))))
(defrule sdl-rule-67 **
(declare (salience -1))
(or (not (success sdl-67))
(not (success (sdl-67))))
=>
(assert (fail sdl-67)))

(defrule sdl-rule-68 **
(sdl-68 data1 ?y)
(sdl-68 (data1 (?y)))
(sdl-68 data2 ?x:(> ?x ?y))
(sdl-68 (data2 (?x:(> ?x ?y))))
=>
(assert (fail sdl-68)))

(defrule sdl-rule-69 **
(sdl-69 data1 ?y)
(sdl-69 (data1 (?y)))
(sdl-69 data2 ?x:(> ?x ?y))
(sdl-69 (data2 (?x:(> ?x ?y))))
(sdl-69 data2 (?ll:(> ?x ?y))))
=>
(assert (success sdl-69))
(assert (success (sdl-69))))

(defrule sdl-rule-69-2 **
(declare (salience -1))
(or (not (success sdl-69))
(not (success (sdl-69))))
=>
(assert (fail sdl-69)))

(defrule sdl-rule-70 **
(sdl-70 data1 ?y)
(sdl-70 (data1 (?y)))
(sdl-70 data2 ?x:(= ?x ?y))
(sdl-70 (data2 (?x:(= ?x ?y))))
=>
(assert (fail sdl-70)))

(defrule sdl-rule-71 **
(sdl-71 data1 ?y)
(sdl-71 (data1(?y)))
(sdl-71 data2 ?x:(= ?x ?y))
(sdl-71 (data2 (?x:(= ?x ?y))))
=>
(assert (fail sdl-71)))

(defrule sdl-rule-72 **
(sdl-72 data1 ?y)
(sdl-72 (data1(?y)))
(sdl-72 data2 ?x:(= ?x ?y))
(sdl-72 (data2 (?x:(= ?x ?y))))
=>
(assert (fail sdl-72)))

(defrule sdl-rule-73 **
(sdl-73 data1 ?y)
(sdl-73 (data1 ?y))
(sdl-73 data2 ?x:(eq ?x ?y))
(sdl-73 (data2 (?x:(eq ?x ?y))))
=>
(assert (fail sdl-73)))

(defrule sdl-rule-74 **
(sdl-74)
(sdl-74())

=>
(assert (fail sdl-74))
(assert (fail (sdl-74))))

(defrule sdl-rule-74-2 **
?temp1 <- (fail sdl-74)
?temp2 <- (fail (sdl-74))

=>
(retract ?temp1 ?temp2))

(defrule sdl-rule-75 **
(sdl-75)
(sdl-75())

=>
(assert (sdl-75 1 :=(gentemp)))
(assert (sdl-75 2 :=(gentemp)))
(assert (sdl-75 1 :=(gentemp))))
(assert (sdl-75 2 :=(gentemp))))

(defrule sdl-rule-75-2 **
?temp1 <- (sdl-75 1 ?)
?temp2 <- (sdl-75 (1 ?))
?temp3 <- (sdl-75 2 ?)
?temp4 <- (sdl-75 (2 ?))

=>
(retract ?temp1 ?temp2 ?temp3 ?temp4))

(defrule sdl-rule-75-3 **
(declare (salience -10))
(or (sdl-75 ? ?)
(sdl-75 (? ?)))

=>
(assert (fail sdl-75)))

(defrule sdl-rule-76 **
(sdl-76)

=>
(assert (*sdl-76 temp)))

Was (string assert "sdl-76 temp")

(defrule sdl-rule-76-2 **
(declare (salience -1))
(not (*sdl-76 temp*))
=>
(assert (fail sdl-76)))

(defrule sdl-rule-77 **
(sdl-77)

=>
(assert (*sdl-77 \"temp\")))

Was (string assert "sdl-77 \"temp\"")

(defrule sdl-rule-77-2 **
(declare (salience -1))
(not (*sdl-77 \"temp\"))
=>
(assert (fail sdl-77)))

(defrule sdl-rule-78 **
(sdl-78)
(sdl-78 ( ))
(sdl-78 ( ?x))
(sdl-78 ( ?x ?x))
(sdl-78 ( ?x ?y))
(sdl-78 ( 2 ?y))

=>

(bind ?a (+ ?x ?y))
(assert (sdl-78 3 ?a)))
(assert (sdl-78 3 13))
-added by BDC 02/04

matches temp fact from above

(matches 5)
(matches 8)

(defrule sdl-rule-78-2 **
(declare (salience -1))
(not (sdl-78 3 13))

=>

(assert (fail sdl-78)))

(defrule sdl-rule-79 **
(sdl-79 $?data)
(sdl-79 ($?data))

=>

(assert (sdl-79 length =(length$ ?data)))
(assert (sdl-79 (length=(length$ ?data)))))

(matches a b c d e f g)

(asserts 7)

(matches 7 fact from above)

(defrule sdl-rule-79-2 **
(or (not (sdl-79 length 7))
(not (sdl-79 (length 7))))

=>

(assert (fail sdl-79)))

(defrule sdl-rule-80 **
(sdl-80 $?data)
(sdl-80 ($?data))

=>

(assert (sdl-80 second =(nth$ ?data 2)))
(assert (sdl-80 (second =(nth$ ?data 2)))))

(matches a b c d e f g)

(asserts b)

(matches b fact from above)

(defrule sdl-rule-80-2 **
(or (not (sdl-80 second b))
(not (sdl-80 (second b))))

=>

(assert (fail sdl-80)))

(defrule sdl-rule-81 **
(sdl-81 $?data1)
(sdl-81 ($?data2))

=>

(bind ?a1 (position$ b $data1))
(bind ?a2 (position$ b $data2))
(assert (temp sdl-81 position ?a1))
(assert (temp sdl-81 (position ?a2))))

(matches a b c d e f g)

(bind ?a to 2)

(asserts 2)

(matches 2 fact from above)

(defrule sdl-rule-81-2 **
(or (not (temp sdl-81 position 2))
(not (temp sdl-81 (position 2))))

=>

(assert (fail sdl-81)))

(defrule sdl-rule-108 **
(sdl-108 ?x)
(sdl-108 (?x))

=>

(if (= ?x 10)
(matches 10)
(matches 10)
then
  (assert (success sdl-108))
else
  (assert (fail sdl-108))))

(defrule sdl-rule-109 **
  (sdl-109 ?y)
  (sdl-109 (?y))
=>
  (bind ?x ?y)
  (while (> ?x 0) do
    (assert (sdl-109 while ?x)
       (sdl-109 (while ?x)))
    (bind ?x (- ?x 1))))

(defrule sdl-rule-109-2 **
  (sdl-109 while 3)
  (sdl-109 while 2)
  (sdl-109 (while 1))
  (sdl-109 while 3)
  (sdl-109 (while 2))
  (sdl-109 (while 1))
=>
  (assert (success sdl-109)))

(defrule sdl-rule-109-3 **
  (declare (salience -1))
  (not (success sdl-109))
=>
  (assert (fail sdl-109)))

(defrule sdl-rule-115 **
  (sdl-115 ?x foo bar ?x)
  (sdl-115 (($?x) (foo bar) ($?x)))
=>
  (assert (success sdl-115)))

(defrule sdl-rule-115-1 **
  (declare (salience -10))
  (not (success sdl-115))
=>
  (assert (fail sdl-115)))

(defrule sdl-rule-116 **
  (or (not (sdl-116 $?vowels u))
       (not (sdl-116 ($?vowels u))))
=>
  (assert (fail sdl-116)))))

(defrule sdl-rule-117 **
  (sdl-117 $?a $?b)
  (sdl-117 ($?a $?b))
=>
  (assert (sdl-117-a $?a))
  (assert (sdl-117-b $?b))
  (assert (sdl-117-a ($?a)))
  (assert (sdl-117-b ($?b))))

(defrule sdl-rule-117-1 **
  (sdl-117-a)
  (sdl-117-a fluff)
  (sdl-117-a fluff mug)
  (sdl-117-a fluff mug bump)
  (sdl-117-a fluff mug bump bleet)
(sdl-117-a fluff mug bump bleet lolita) ; six pairs
(sdl-117-b fluff mug bump bleet lolita)
(sdl-117-b mug bump bleet lolita)
(sdl-117-b bump bleet lolita)
(sdl-117-b lolita)
(sdl-117-b)
(sdl-117-a ?)
(sdl-117-a (fluff))
(sdl-117-a (fluff mug))
(sdl-117-a (fluff mug bump))
(sdl-117-a (fluff mug bump bleet))
(sdl-117-a (fluff mug bump bleet lolita)) ; another six pairs
(sdl-117-b (fluff mug bump bleet lolita))
(sdl-117-b (mug bump bleet lolita))
(sdl-117-b (bleet lolita))
(sdl-117-b (lolita))
(sdl-117-b ()

=> (assert (success sdl-117)))

(derule sdl-rule-117-2 **
(declare (salience -10))
(not (success sdl-117)) =>
(assert (fail sdl-117)))

(derule sdl-rule-ski-1 **
(non-working ?date)
(non-working (?)date)
(and (not (weather ?date hot-and-humid))
(not (weather ?date major-blizzard))
(not (traffic ?date massive))
(not (weather (?date hot-and-humid)))
(not (weather (?date major-blizzard)))
(not (traffic (?date massive)))) =>
(assert (go-hiking ?date))
(assert (go-hiking (?)date))))

(derule sdl-rule-ski-2 **
(non-working ?date)
(non-working (?)date)
(or (not (weather ?date major-blizzard))
(not (drive ?date two-wheel))
(not (working ?date weekend))
(not (weather (?date major-blizzard)))
(not (drive (?date two-wheel)))
(not (working (?date weekend)))) =>
(assert (go-skiing ?date))
(assert (go-skiing (?)date))))

(derule sdl-rule-118 **
(or
(go-hiking sdl-118)
(not (go-skiing sdl-118))
(go-hiking (sdl-118))
(not (go-skiing (sdl-118)))

=> (assert (fail sdl-118)))

104
(defrule sdl-rule-119 **
  (or
    (not (go-biking sdl-119))
    (not (go-skling sdl-119))
    (not (go-biking (sdl-119)))
    (not (go-skling (sdl-119))))
=>
  (assert (fail sdl-119)))

(defrule sdl-rule-120 **
  (or
    (go-biking sdl-120)
    (go-skling sdl-120)
    (go-biking (sdl-120))
    (go-skling (sdl-120)))
=>
  (assert (fail sdl-120)))

(defrule sdl-rule-121 **
  (sdl-121 $? ?x $? ?x $?)
  (sdl-121 ($? ?x $? ?x $?))
=>
  (assert (match sdl-121 ?x)))

(defrule sdl-rule-121-2 **
  (declare (salience -1))
  (not (match sdl-121 one))
=>
  (assert (fail sdl-121)))

(defrule sdl-rule-122 **
  (sdl-122 $? ?x $? ?x $?)
  (sdl-122 ($? ?x $? ?x $?))
=>
  (assert (match sdl-122 ?x)))

(defrule sdl-rule-122-2 **
  (declare (salience -1))
  (not (match sdl-122 one))
=>
  (assert (fail sdl-122)))

(defrule sdl-rule-123 **
  (sdl-123 $? ?x $? ?x $?)
  (sdl-123 ($? ?x $? ?x $?))
=>
  (assert (match sdl-123 ?x)))

(defrule sdl-rule-123-2 **
  (declare (salience -1))
  (not (match sdl-123 5))
=>
  (assert (fail sdl-123)))

(defrule sdl-rule-124 **
  (sdl-124 $? ?x $? ?x $?)
  (sdl-124 ($? ?x $? ?x $?))
=>
  (assert (match sdl-124 ?x)))

(defrule sdl-rule-124-2 **
  (declare (salience -1))
  (not (match sdl-124 r))
=>
  (assert (fail sdl-124)))
(assert (fail sdl-124)))

(defrule sdl-rule-125 **
 (sdl-125 $?x $?x $?x $?x)
 (sdl-125 ( $?x $?x $?x $?x))
 =>
 (assert (match sdl-125 ?x)))
 :should get two asserted facts

(defrule sdl-rule-125-2 **
 (declare (salience -1))
 (or (not (match sdl-125 a))
  (not (match sdl-125 rose)))
 =>
 (assert (fail sdl-125)))
 :added by BDC 02/05/88

(defrule sdl-rule-126 **
 (or (not (sdl-126 $?x or not $?x))
  (not (sdl-126 ( $?x or not $?x))))
 =>
 (assert (fail sdl-126)))

(defrule sdl-rule-127-master **
 (sdl-127 $?x)
 (sdl-127 (? $?x))
 =>
 (assert (sdl-127-result $?x))
 (assert (sdl-127-result ($?x))))
 :should be no fact of this type

(defrule sdl-rule-127-2 **
 (or (not (sdl-127-result))
  (not (sdl-127-result ()))))
 =>
 (assert (fail sdl-127)))

(defrule sdl-rule-128-master **
 (sdl-128 $?x)
 (sdl-128 (? $?x))
 =>
 (assert (sdl-128-result $?x))
 (assert (sdl-128-result ($?x))))
 :added by BDC 02/05/88

(defrule sdl-rule-128-2 **
 (declare (salience -1))
 (or (not (sdl-128-result 2))
  (not (sdl-128-result (2))))
 =>
 (assert (fail sdl-128)))

(defrule sdl-rule-129-master **
 (sdl-129 $?x)
 (sdl-129 (? $?x))
 =>
 (assert (sdl-129-result $?x))
 (assert (sdl-129-result ($?x))))
 :added by BDC 02/05/88

(defrule sdl-rule-129-2 **
 (declare (salience -1))
 (or (not (sdl-129-result in bloom))
  (not (sdl-129-result (in bloom)))))
 =>
 (assert (fail sdl-129)))
 :from previous rule

(defrule sdl-rule-130-master **

106
(defrule sdl-rule-130 "a rose is a rose")
(assert (sdl-130-result "a rose is a rose"))
(assert (sdl-130-result ($?x)))

(defrule sdl-rule-131 "a rose is a rose")
(assert (sdl-131-result "$?x"))

(defrule sdl-rule-131-1 "a rose is a rose")
(assert (sdl-131-1-result "$?x"))

(defrule sdl-rule-133 "a rose is a rose")
(assert (sdl-133-rule-133-result "$?x"))

(defrule sdl-rule-134 "a rose is a rose")
(assert (sdl-134-rule-134-result "$?x"))

(defrule sdl-rule-135 "a rose is a rose")
(assert (sdl-135-rule-135-result "$?x"))
(sdl-135 (?a #red & #blue & #green & #violet & #orange & #black))

=>
(assert (sdl-135-result ?a)) yellow
(assert (sdl-135-result (?a))) yellow

(defrule sdl-135-rule-1 **
  (sdl-135-result yellow) from previous rule
  (sdl-135-result (yellow)) from previous rule
  (not (sdl-135-result green)) should be no such fact
  (not (sdl-135-result (green)))
=>
(assert (success sdl-135)))

(defrule sdl-135-rule-2 **
  (declare (salience -10)) previous rule
=>
(assert (fail sdl-135)))

(defrule sdl-136-rule **
  (sdl-136 ?a #red & #blue & #green & #violet & #orange & #black) yellow and green
  (sdl-136 (?a #red & #blue & #green & #violet & #orange & #black)) yellow and green
=>
(assert (sdl-136-result ?a)) one green asserted
(assert (sdl-136-result (?a))) one green asserted

(defrule sdl-136-rule-1 **
  (not (sdl-136-result yellow)) should not exist
  (not (sdl-136-result (yellow))) should not exist
  (sdl-136-result green) should exist
  (sdl-136-result (green)) should exist
=>
(assert (success sdl-136)))

(defrule sdl-136-rule-2 **
  (declare (salience -10)) from previous rule
=>
(assert (fail sdl-136)))

(defrule sdl-137-rule **
  (sdl-137 red#blue#green#violet#orange#black)
  (sdl-137 (red#blue#green#violet#orange#black))
=>
(assert (fail sdl-137)))

(defrule sdl-138-rule **
  (or (not (sdl-138 red & blue & green & violet & orange & black)) yellow and green
  (not (sdl-138 (red & blue & green & violet & orange & black))))
=>
(assert (fail sdl-138)))

(defrule sdl-139-rule **
  (or (not (sdl-139 ?x =((abs ?x)))) -12 12
  (not (sdl-139 (?x =((abs ?x)))))
=>
(assert (fail sdl-139)))

(defrule sdl-140-rule **
  (sdl-140 ?x1 ?y1) 1 1 also 1 2
  (sdl-140 ?x2 & "?x1 ?y2 & "?y1)
  (sdl-140 (?x1 ?y1)) 1 1 also 1 2
  (sdl-140 (?x2 & "?x1 ?y2 & "?y1))
=>
(assert (fail sdl-140)))
(defrule sdl-141-rule **
  (sdl-141 ?x1 ?y1)
  (sdl-141 ?x2 ?y2)
  (sdl-141 (?x1 ?y1))
  (test (or (not (= ?x1 ?x2))
           (not (= ?y1 ?y2))))
  =>
  (assert (success sdl-141)))
(defrule sdl-141-rule-1 **
  (declare (salience -10))
  (not (success sdl-141))
  =>
  (assert (fail sdl-141)))
(defrule sdl-142-rule **
  (sdl-142 ?first ?second)
  (sdl-142 (?first ?second))
  =>
  (assert (sdl-142 ?second ?first))
  (assert (sdl-142 (?second ?first))))
(defrule sdl-142-rule-1 **
  (declare (salience -1))
  (or (not (sdl-142 hot cold))
      (not (sdl-142 hot cold)))
  =>
  (assert (fail sdl-142)))
(defrule sdl-143-rule **
  (sdl-143 $?x red $?y)
  (sdl-143 ($?x red $?y))
  =>
  (assert (sdl-143 red $?x $?y))
  (assert (sdl-143 (red $?x $?y))))
(defrule sdl-143-rule-1 **
  (declare (salience -1))
  (or (not (sdl-143 red green yellow blue white))
      (not (sdl-143 (red green yellow blue white))))
  =>
  (assert (fail sdl-143)))
(defrule sdl-144-rule **
  (sdl-144 brother-of ?father ?uncle)
  (sdl-144 child-of ?father ?child)
  (sdl-144 sex-of ?child female)
  (sdl-144 (brother-of ?father ?uncle))
  (sdl-144 (child-of ?father ?child))
  (sdl-144 (sex-of ?child female))
  =>
  (assert (sdl-144 niece-of ?uncle ?child))
  (assert (sdl-144 uncle-of ?child ?uncle))
  (assert (sdl-144 (niece-of ?uncle ?child)))
  (assert (sdl-144 (uncle-of ?child ?uncle))))
(defrule sdl-144-rule-2 **
  (sdl-144 brother-of ?father ?uncle)
  (sdl-144 child-of ?father ?child)
  (sdl-144 sex-of ?child male)
  (sdl-144 (brother-of ?father ?uncle))
  =>
  (assert (sdl-144 niece-of ?uncle ?child))
  (assert (sdl-144 uncle-of ?child ?uncle))
  (assert (sdl-144 (niece-of ?uncle ?child)))
  (assert (sdl-144 (uncle-of ?child ?uncle))))
(assert (child-of ?father ?child))
(assume (sex-of ?child male))

=>
(assert (nephew-of ?uncle ?child))
(assert (uncle-of ?child ?uncle))

(defrule sdl-144-rule-3 **
  (niece-of-daniel jane)
  (nephew-of daniel david)
  (uncle-of david daniel))

=>
(assert (success sdl-144)))

(defrule sdl-144-rule-4 **
  (declare (salience -10))
  (not (success sdl-144))
=>
(assert (fail sdl-144)))

(defrule sdl-rule-145 **
  (list ?colors)
  (list ?colors))

=>
(bind ?i 1)
(bind ?length (length ?colors))

(while (< ?i ?length)
  (assert (control ?colors ?i))
  (bind ?i (* 7 i)))

(defrule sdl-rule-146 **
  (seats ?seats)
  (names ?names)
  (seats ?seats)
  (names ?names))

=>
(bind ?i 1)
(bind ?length (min (length ?seats) (length ?names)))

(while (< ?i ?length)
  (assert (assignment ?seats ?i)
          (= (names ?names ?i))
          (= (seats ?seats ?i))
          (bind ?i (+ ?i 1))))

(defrule sdl-rule-146-1 **
  (or (not (control ?color))
      (not (list ?color))))

=>
(assert (fail sdl-145)))

(defrule sdl-rule-146 **
  (seats ?seats)
  (names ?names)
  (seats ?seats)
  (names ?names))

=>
(bind ?i 1)
(bind ?length (min (length ?seats) (length ?names)))

(while (< ?i ?length)
  (assert (assignment ?seats ?i)
          (= (names ?names ?i))
          (= (seats ?seats ?i))
          (bind ?i (+ ?i 1))))

(defrule sdl-rule-146-1 **
  (or (not (control ?color))
      (not (list ?color))))
(assert (success sdl-146)))

(defrule sdl-146-rule-2 ''
  (declare (salience -10))
  (not (success sdl-146))
  =>
  (assert (fail sdl-146)))

(defrule sdl-147 **
  (sdl-147 ?list ?donor ?next-donor & "finish")
  (sdl-147 donation ?next-donor ?contribution)
  ?state-fact-2 <- (sdl-147-sum (?list ?donor ?sum-so-far))
  (sdl-147 (?list ?donor ?next-donor & "finish"))
  (sdl-147 donation ?next-donor ?contribution))
  =>
  (retract ?state-fact-1 ?state-fact-2)
  (assert (sdl-147-sum ?list ?next-donor
  =(+ ?sum-so-far ?contribution)))
  (assert (sdl-147-sum (?list ?next-donor
  =(+ ?sum-so-far ?contribution)))))

(defrule sdl-147-2 **
  (sdl-147 ?list ?donor finish)
  ?state-fact-2 <- (sdl-147-sum (?list ?donor ?sum))
  (sdl-147 (?list ?donor finish))
  =>
  (retract ?state-fact-1 ?state-fact-2)
  (assert (sdl-147-final ?list ?sum))
  (assert (sdl-147-final (?list ?sum))))

(defrule sdl-147-3 **
  (or (not (sdl-147-final donors 17.5))
  (not (sdl-147-final (donors 17.5)))))
  =>
  (assert (fail sdl-147)))

(defrule sdl-rule-148 **
  (sdl-148 donor-list ?$?donors)
  (sdl-148 (donor-list ?$?donors))
  =>
  (bind ?length (length$ ?donors))
  (assert (sdl-148 donors start =(?nth$ ?donors 1)))
  (assert (sdl-148 (donors start =(?nth$ ?donors 1))))
  (assert (sdl-148 donors =(?nth$ ?donors ?length) finish))
  (assert (sdl-148 (donors =(?nth$ ?donors ?length) finish)))
  (bind ?1)
  (while (< ?1 ?length) do
  (assert (sdl-148 donors =(?nth$ ?donors ?1) =(?nth$ ?donors (+ ?1 1))
  (sdl-148 (donors =(?nth$ ?donors ?1) =(?nth$ ?donors (+ ?1 1)))))
  (bind ?1 (+ ?1 1))))

(defrule sdl-148-1 **
(defrule sdl-148-2 "
  (sdl-148 (?list ?donor ?next-donor & "finish"))
  (sdl-148 (donation ?next-donor ?contribution))

  =>
  (retract ?state-fact-1 ?state-fact-2)
  (assert (sdl-148-sum (?list ?next-donor (+ ?sum-so-far ?contribution))))
)

(defrule sdl-148-3 "
  (declare (salience -1))
  (or (not (sdl-148-final donors 17.5))
   (not (sdl-148-final (donors 17.5))))

  =>
  (assert (fail sdl-148)))

(defrule sdl-149 "
  (declare (salience 1000))
  (sdl-149 inventory ?name ?amount)
  (sdl-149 (inventory ?name ?amount))

  =>
  (assert (sdl-149 add-to-sum ?name ?amount))
  (assert (sdl-149 (add-to-sum ?name ?amount))))

(defrule sdl-149-1 "
  (declare (salience 1000))
  'x <- (sdl-149 add-to-sum ?name ?amount)
  'y <- (sdl-149 current-sum ?sum)
  'z <- (sdl-149 (add-to-sum ?name ?amount))
  'w <- (sdl-149 (current-sum ?sum))

  =>
  (retract 'x 'y)
  (assert (sdl-149 current-sum =(+ ?sum ?amount))))
  (assert (sdl-149 (current-sum =(+ ?sum ?amount))))
)

(defrule sdl-149-2 "
  (declare (salience -1))
  (or (not (sdl-149 current-sum 19.5))
   (not (sdl-149 (current-sum 19.5))))

  =>
  (assert (fail sdl-149)))

(defrule sdl-150 "
  (declare (salience 1000))
  (sdl-150 inventory ?name ?amount)
  (sdl-150 (inventory ?name ?amount))

  =>
  (assert (sdl-150 count-item ?name ?amount))
  (assert (sdl-150 (count-item ?name ?amount))))

(defrule sdl-150-1 "
(declare (salience 1000))
?x <- (sdl-150 count-item ?name ?amount)
?y <- (sdl-150 current-count ?count)
?z <- (sdl-150 (count-item ?name ?amount))
?w <- (sdl-150 (current-count ?count))
=>
(retract ?x ?y ?z ?w)
(assert (sdl-150 current-count (+ ?count 1)))
(assert (sdl-150 counted ?name ?amount))
(assert (sdl-150 (current-count (+ ?count 1))))
(assert (sdl-150 (counted ?name ?amount)))
)

(defrule sdl-150-2 **
(declare (salience 1000))
?x <- (sdl-150 counted ?name ?amount)
?z <- (sdl-150 (counted ?name ?amount))
(or (not (sdl-150 inventory ?name ?amount)))
(not (sdl-150 (inventory ?name ?amount))))
?y <- (sdl-150 current-count ?count)
?w <- (sdl-150 (current-count ?count))
=>
(retract ?x ?y ?w ?z)
(assert (sdl-150 current-count (- ?count 1)))
(assert (sdl-150 (current-count (- ?count 1))))
)

(defrule sdl-150-3 **
(declare (salience -1))
(or (not (sdl-150 current-count 3))
(not (sdl-150 (current-count 3))))
=>
(assert (fail sdl-150)))

(defrule sdl-151
(sdl-151 quarter ?w)
(sdl-151 dime ?x)
(sdl-151 nickel ?y)
(sdl-151 penny ?z)
(sdl-151 (quarter ?w))
(sdl-151 (dime ?x))
(sdl-151 (nickel ?y))
(sdl-151 (penny ?z))
(test (> ?w ?x ?y ?z))
=>
(assert (success sdl-151)))

(defrule sdl-151-1 **
(declare (salience -10))
(not (success sdl-151))
=>
(assert (fail sdl-151)))

(defrule sdl-152 **
(sdl-152 quarter ?w)
(sdl-152 dime ?x)
(sdl-152 nickel ?y)
(sdl-152 penny ?z)
(sdl-152 (quarter ?w))
(sdl-152 (dime ?x))
(test (> ?w ?x))
(sdl-152 (nickel ?y))
(test (> ?x ?y))
(sdl-152 (penny ?z))

113
(test (> ?y ?z))

> (assert (success sdl-152)))

(defrule sdl-152-1 ''
 (declare (salience -10))
 (not (success sdl-152))
>
 (assert (fail sdl-152)))

(defrule sdl-153 ''
 (sdl-153 ?answer & yes | no | unknown | :((numberp ?answer))) . yes, no, unknown, |
 (sdl-153 (?answer & yes | no | unknown | :((numberp ?answer))) . yes, no, unknown, |
>
 (assert (sdl-153 matched ?answer))
 (assert (sdl-153 (matched ?answer))))

(defrule sdl-153-1 ''
 (sdl-153 matched yes)
 (sdl-153 matched no)
 (sdl-153 matched unknown)
 (sdl-153 (matched yes))
 (sdl-153 (matched no))
 (sdl-153 (matched unknown))
 (sdl-153 (matched 1))
>
 (assert (success sdl-153)))

(defrule sdl-153-2 ''
 (declare (salience -10))
 (not (success sdl-153))
>
 (assert (fail sdl-153)))

(defrule sdl-154 ''
 (sdl-154 ?time1 & :((numberp ?time1)))
 (sdl-154 ?time2 & :((numberp ?time2) & :>( ?time2 ?time1)))
 (sdl-154 (?time1 & :((numberp ?time1))))
 (sdl-154 (?time2 & :((numberp ?time2) & :>( ?time2 ?time1))))
>
 (assert (sdl-154-matched ?time1 ?time2))
 (assert (sdl-154-matched (?time1 ?time2))))

(defrule sdl-154-1 ''
 (sdl-154-matched 1 2)
 (sdl-154-matched 2 3)
 (sdl-154-matched 1 3)
 (sdl-154-matched (1 2))
 (sdl-154-matched (2 3))
 (sdl-154-matched (1 3))
>
 (assert (success sdl-154)))

(defrule sdl-154-2 ''
 (declare (salience -10))
 (not (success sdl-154))
>
 (assert (fail sdl-154)))

(defrule sdl-155 ''
 (sdl-155 ?time1 & :((numberp ?time1)))
 (sdl-155 ?time2 & :((numberp ?time2)))
 (sdl-155 (?time1 & :((numberp ?time1))))
(assert (success sdl-155)))

(defrule sdl-155-1 **
  (sdl-155 matched 1 2)
  (sdl-155 matched 2 3)
  (sdl-155 matched 1 3)
  (sdl-155 matched 1 2)
  (sdl-155 matched 2 3)
  (sdl-155 matched 1 3))
=>
(assert (success sdl-155)))

(defrule sdl-155-2 **
  (declare (salience -10))
  (not (success sdl-155))
=>
(assert (fail sdl-155)))

(defrule sdl-157 **
  (or (not (sdl-157 ?x & yes | no | (numberp ?x) | (stringp ?x)))
       (not (sdl-157 (?x & yes | no | (numberp ?x) | (stringp ?x))))))
=>
(assert (fail sdl-157)))

(defrule sdl-158 **
  (or (not (sdl-158 ?x & (stringp ?x)))
       (not (sdl-158 (?x & (stringp ?x))))))
=>
(assert (fail sdl-158)))

(defrule sdl-159 **
  (sdl-159 ?answer & yes | no | unknown | (numberp ?answer))
  (sdl-159 (?answer & yes | no | unknown | (numberp ?answer))))
=>
(assert (sdl-159 matched ?answer))
(assert (sdl-159 (matched ?answer))))

(defrule sdl-159-1 **
  (sdl-159 matched yes)
  (sdl-159 matched no)
  (sdl-159 matched unknown)
  (sdl-159 matched 1)
  (sdl-159 (matched yes))
  (sdl-159 (matched no))
  (sdl-159 (matched unknown))
  (sdl-159 (matched 1)))
=>
(assert (success sdl-159)))

(defrule sdl-159-2 **
  (declare (salience -10))
  (not (success sdl-159))
=>
(assert (fail sdl-159)))
(defrule sdl-160 "\n\n(sdl-160 answer : (numberp answer) | yes | no | unknown) \n(sdl-160 (answer : (numberp answer) | yes | no | unknown)) \n=> \n(assert (sdl-160 matched answer)) \n(assert (sdl-160 (matched answer)))) \n\n(defrule sdl-160-1 "\n(sdl-160 matched yes) \n(sdl-160 matched no) \n(sdl-160 matched unknown) \n(sdl-160 matched 1) \n(sdl-160 (matched yes)) \n(sdl-160 (matched no)) \n(sdl-160 (matched unknown)) \n(sdl-160 (matched 1)) \n=> \n(assert (success sdl-160))) \n\n(defrule sdl-160-2 "\n(declare (salience -10)) \n(not (success sdl-160)) \n=> \n(assert (fail sdl-160))) \n\n(defrule sdl-161 "\n(sdl-161 ?time1 : (numberp ?time1)) \n(sdl-161 (!time1 : (numberp ?time1))) \n(sdl-161 ?time2 : (numberp ?time2) & (!time2 ?time1)) \n(sdl-161 (?time2 : (numberp ?time2) & (!time2 ?time1))) \n=> \n(assert (sdl-161 matched ?time1 ?time2))) \n\n(defrule sdl-161-1 "\n(sdl-161 matched ?time1 ?time2)" \n(sdl-161 matched 1 2) \n(sdl-161 matched 2 3) \n(sdl-161 matched 1 3) \n=> \n(assert (success sdl-161))) \n\n(defrule sdl-161-2 "\n(declare (salience -10)) \n(not (success sdl-161)) \n=> \n(assert (fail sdl-161))) \n\n(defrule sdl-162 "\n(sdl-162 ?time1 : (numberp ?time1)) \n(sdl-162 (!time1 : (numberp ?time1))) \n(sdl-162 ?time2 : (numberp ?time2)) \n(sdl-162 (?time2 : (numberp ?time2))) \n(test (> ?time2 ?time1)) \n=> \n(assert (sdl-162 matched ?time1 ?time2))) \n\n(defrule sdl-162-1 "\n(sdl-162 matched ?time1 ?time2)" \n(sdl-162 matched 1 2) \n(sdl-162 matched 2 3) \n(sdl-162 matched 1 3) \n=> \n(assert (success sdl-162))) \n\n(defrule sdl-162-2 "\n(declare (salience -10))

116
(not (success sdl-162))

=>

(assert (fail sdl-162))

(defrule sdl-163 **
  (or (not (sdl-163 "input" "yes" | "no" | "unknown" | "overflow" | "inference"))
      (not (sdl-163 (?input "yes" | "no" | "unknown" | "overflow" | "inference"))))
=>

(assert (fail sdl-163))

(defrule sdl-164 **
  (sdl-164 input ?input)
  (sdl-164 list $?list)
  (sdl-164 (input ?input))
  (sdl-164 (list $?list))
=>

(if (position $ ?input ?list)
  then
  (assert (success sdl-164))
  else
  (assert (fail sdl-164))))

(defrule sdl-165 **
  (or (not (sdl-165 12345.89))
      (not (sdl-165 (12345.89))))
=>

(assert (fail sdl-165))

(defrule sdl-166 **
  (patient-name ?patient-1)
  (patient-name (?patient-1))
  (sdl-166 $ ?patient-1 $?
    ?answer $?
    yes | no | unknown | \(\text{numberp} \ ?answer\) ?$)
  (sdl-166 $ ?answer &
    yes | no | unknown | \(\text{numberp} \ ?answer\) ?$)
  (sdl-166 ($ ?answer &
    yes | no | unknown | \(\text{numberp} \ ?answer\) ?$)
  (sdl-166 ($ ?patient-1 $?
    ?answer &
    yes | no | unknown | \(\text{numberp} \ ?answer\) ?$)
  (sdl-166 ($ ?patient-1 $?
    ?answer &
    yes | no | unknown | \(\text{numberp} \ ?answer\) ?$)
=>

(assert (sdl-166-matched ?patient-1 ?answer))
(assert (sdl-166-matched (?patient-1 ?answer))))

(defrule sdl-166-1 **
  (sdl-166-matched paul 1.5)
  (sdl-166-matched paul yes)
  (sdl-166-matched (paul yes))
  (sdl-166-matched (paul 1.5))
  (sdl-166-matched (paul yes))
(assert (success sdl-166)))
(defrule sdl-166-2
  (declare (salience -10))
  (not (success sdl-166)) =>
  (assert (fail sdl-166)))
(defrule sdl-167
  (patlent-name ?patient-1)
  (patlent-name (?patient-1))
  (patlent-name (?patient-2 & ~?patient-1))
  (patlent-name (?patient-2 & ~?patient-1))
  (or (sdl-167 $?w ?patient-1
    $?x ?patient-2
    $?y ?answer &
    yes | no |
    unknown |
    :((numberp ?answer)
    $?z))
  (sdl-167 $?w ?patient-1
    $?x ?patient-2
    $?y ?answer &
    yes | no |
    unknown |
    :((numberp ?answer)
    $?y ?patient-2
    $?z))
  (sdl-167 ($?w ?patient-1
    $?x ?patient-2
    $?y ?answer &
    yes | no |
    unknown |
    :((numberp ?answer)
    $?z)))
  (sdl-167 ($?w ?patient-1
    $?x ?answer &
    yes | no |
    unknown |
    :((numberp ?answer)
    $?y ?patient-2
    $?z))) =>
  (assert (sdl-167-matched ?patient-1 ?answer))
  (assert (sdl-167-matched ?patient-2 ?answer))
  (assert (sdl-167-matched (?patient-1 ?answer)))
  (assert (sdl-167-matched (?patient-2 ?answer))))
(defrule sdl-167-1
  (sdl-167-matched paul 1.5)
  (sdl-167-matched paul yes)
  (sdl-167-matched brad 1.5)
  (sdl-167-matched brad yes)
  (sdl-167-matched (paul 1.5))
  (sdl-167-matched (paul yes))
  (sdl-167-matched (brad 1.5))
  (sdl-167-matched (brad yes)) =>
  (assert (success sdl-167)))
(defrule sdl-157-2 **
  (declare (salience -10))
  (not (success sdl-157)))
=>
(assert (fail sdl-157)))

(defrule sdl-158 **
  (patient-name ?patient-1)
  (patient-name ?patient-2 & ?patient-1)
  (patient-name ?patient-1)
  (patient-name (patient-2 & ?patient-1))
  (or (sdl-158 $?v
    ?answer-1 & yes | no | (numberp ?answer-1) | unknown
    $?w
    ?patient-1
    $?x

    ;; split the fact BEFORE the second answer
    ?answer-2 & yes | no | (numberp ?answer-2) | unknown
    $?y
    ?patient-2
    $?z)
  (sdl-158 $?x
    ?patient-1
    $?w
    ?answer-1 & yes | no | (numberp ?answer-1) | unknown
    $?x
    ?patient-2
    $?y
    ?answer-2 & yes | no | (numberp ?answer-2) | unknown
    $?z))

  (sdl-158 $?y
    ?answer-2 & yes | no | (numberp ?answer-2) | unknown
    $?y
    ?patient-2
    $?z)))

  (sdl-158 $?z
    ?patient-1
    $?w
    ?answer-1 & yes | no | (numberp ?answer-1) | unknown
    $?x
    ?patient-2
    $?y
    ?answer-2 & yes | no | (numberp ?answer-2) | unknown
    $?z)))
=>
(assert (sdl-158-matched ?patient-1 ?answer-1))
(assert (sdl-158-matched ?patient-2 ?answer-2))
(assert (sdl-158-matched (?patient-1 ?answer-1)))
(assert (sdl-158-matched (?patient-2 ?answer-2)))

(defrule sdl-158-1 **
  (sdl-158-matched paul yes)
  (sdl-158-matched paul 1.5)
(assert (success sdl-168))

(defrule sdl-168-2 **
(declare (salience -10))
(not (success sdl-168))
=>
(assert (fail sdl-168)))

(defrule sdl-169-0 **
(declare (salience 100))
?a <- (sdl-169 $?before $?answer yes
 | no
 $?middle not $?after)
?b <- (sdl-169 ($?before $?answer yes
 | no
 $?middle not $?after))
=>
(retract ?a ?b)
(if
(equal $?answer yes)
then (assert (sdl-169 $?before $?answer $?middle no $?after)
(assert (sdl-169 ($?before $?answer $?middle no $?after))))
else (assert (sdl-169 $?before $?answer $?middle yes $?after)
(assert (sdl-169 ($?before $?answer $?middle yes $?after))))
)

(defrule sdl-169 **
(patient-name $?patient-1)
(patient-name $?patient-2 & $?patient-1)
(patient-name $?patient-1)
(patient-name $?patient-2 & $?patient-1)
(or (sdl-169 $?v
$?answer-1 & yes | no | (numberp $?answer-1) | unknown
$?w
$?patient-1
$?x

.. split the fact BEFORE the second answer
$?answer-2 & yes | no | (numberp $?answer-2) | unknown
$?y
$?patient-2
$?z)
(sdl-169 $?v
$?patient-1
$?w
$?answer-1 & yes | no | (numberp $?answer-1) | unknown
$?x
$?patient-2
$?y
$?answer-2 & yes | no | (numberp $?answer-2) | unknown
$?z)
(sdl-169 ($?v
$?answer-1 & yes | no | (numberp $?answer-1) | unknown
$?w
$?patient-1
$?x

120
:: split the fact BEFORE the second answer
\(\text{?answer-2 & yes | no | (numberp ?answer-2) | unknown}
\)
\(\text{??y}
\)
\(\text{?patient-2}
\)
\(\text{??z)}\)
\(\text{sdl-169 (??v}
\)
\(\text{?patient-1}
\)
\(\text{??w}
\)
\(\text{?answer-1 & yes | no | (numberp ?answer-1) | unknown}
\)
\(\text{:: split the fact AFTER the first answer}
\)
\(\text{??x}
\)
\(\text{?patient-2}
\)
\(\text{??y}
\)
\(\text{?answer-2 & yes | no | (numberp ?answer-2) | unknown}
\)
\(\text{??z)}\)

=>

\(\text{(assert (sdl-169-matched ?patient-1 ?answer-1))}
\)
\(\text{(assert (sdl-169-matched ?patient-2 ?answer-2))}
\)
\(\text{(assert (sdl-169-matched (?patient-1 ?answer-1)))}
\)
\(\text{(assert (sdl-169-matched (?patient-2 ?answer-2))))}
\)
\(\text{(deffrule sdl-169-1 **}
\)
\(\text{(sdl-169-matched paul yes)}
\)
\(\text{(sdl-169-matched brad NO)}
\)
\(\text{(sdl-169-matched (paul yes))}
\)
\(\text{(sdl-169-matched (brad NO))}
\)

=>

\(\text{(assert (success sdl-169)))}
\)

\(\text{(deffrule sdl-169-2 **}
\)
\(\text{(declare (salience -10))}
\)
\(\text{(not (success sdl-169))}
\)

=>

\(\text{(assert (fail sdl-169)))}
\)

\(\text{(deffrule sdl-170 **}
\)
\(\text{(patient-name ?patient)}
\)
\(\text{(patient-name (?patient)))}
\)
\(\text{(sdl-170 ?patient COMPLAINS | HAS $?}
\)
\(\text{?complaint & blackout |}
\)
\(\text{faintness |}
\)
\(\text{fatigue |}
\)
\(\text{headache |}
\)
\(\text{vertigo |}
\)
\(\text{anxiety |}
\)
\(\text{confusion |}
\)
\(\text{depression |}
\)
\(\text{drowsiness |}
\)
\(\text{nervousness |}
\)
\(\text{numbness |}
\)
\(\text{paralysis |}
\)
\(\text{tension |}
\)
\(\text{tingling |}
\)
\(\text{weakness $?)}
\)
\(\text{(sdl-170 (?patient COMPLAINS | HAS $?}
\)
\(\text{?complaint & blackout |}
\)
\(\text{faintness |}
\)
\(\text{fatigue |}
\)
\(\text{headache |}
\)
\(\text{vertigo |}
\)
\(\text{anxiety |}
\)
\(\text{confusion |}
\)
\(\text{depression |}
\)

121
drowsiness | nervousness | numbness | paralysis | tension | tingling | weakness $?

=>
(assert (sdl-170-complaint ?patient ?complaint))
(assert (sdl-170-complaint (?patient ?complaint))))

(defrule sdl-170-1 **
(sdl-170-complaint paul vertigo)
(sdl-170-complaint paul faintness)
(sdl-170-complaint brad headache)
(sdl-170-complaint (paul vertigo))
(sdl-170-complaint (paul faintness))
(sdl-170-complaint (brad headache))
=>
(assert (success sdl-170)))

(defrule sdl-170-2 **
(declare (salience -10))
(not (success sdl-170))
=>
(assert (fail sdl-170)))

(defrule sdl-171 **
(patient-name ?patient)
(age ?patient ?age & "unknown
 & :>= ?age 21))
(patient-name (?patient))
(seq-age ?patient (?age & "unknown
 & :>= ?age 21))
=>
(assert (sdl-171 "patient yes))
(assert (sdl-171 (?patient yes))))

(defrule sdl-171-2 **
(patient-name ?patient)
(age ?patient ?age & "unknown
 & :< ?age 21))
(patient-name (?patient))
(seq-age ?patient (?age & "unknown
 & :< ?age 21))
=>
(assert (sdl-171 "patient no))
(assert (sdl-171 (?patient no))))

(defrule sdl-171-3 **
(sdl-171 paul yes)
(sdl-171 brad no)
(sdl-171 (paul yes))
(sdl-171 (brad no))
=>
(assert (success sdl-171)))

(defrule sdl-171-4 **
(declare (salience -10))
(not (success sdl-171))
=>
(assert (fail sdl-171)))

(defrule sdl-172-2 **
(patient-name ?patient)
(patient-name (?patient))
(sdl-172 major-complaint ?patient
"reason &
drowsiness |
c confusion)
(sdl-172 recent-head-injury ?patient no)
(sdl-172 confusion-mild ?patient yes)
(sdl-172 (major-complaint ?patient
"reason &
drowsiness |
c confusion))
(sdl-172 (recent-head-injury ?patient no))
(sdl-172 (confusion-mild ?patient yes))
(or (not (sdl-172 major-complaint ?patient
  "confusion &
drowsiness))

(not (sdl-172 (major-complaint ?patient
  "confusion &
drowsiness)))))

=>
(assert (sdl-172 recommendation ?patient waiting room))
(assert (sdl-172 (recommendation ?patient waiting room))))

(defrule sdl-172-3 **
(sdl-172 recommendation paul waiting room)
(sdl-172 (recommendation paul waiting room))
(or (not (sdl-172 recommendation brad waiting room))
  (not (sdl-172 (recommendation brad waiting room)))))

=>
(assert (success sdl-172))

(defrule sdl-172-4 **
(declare (salience -10))
(not (success sdl-172))

=>
(assert (fail sdl-172))

(defrule sdl-173 **
(sdl-173 ?case1 ?a1 ?b1 ?c1)
(sdl-173 (?case1 ?a1 ?b1 ?c1))
(sdl-173 ?case2 & "case2 & "foo & "bar
  ?a2 & (not (symbolp ?a2)) & (numberp ?a2) & (< ?a2 ?a1)
  ?b2 & (not (symbolp ?b2)) & (numberp ?b2) & (> ?b2 ?b1)
(sdl-173 (?case2 & "case2 & "foo & "bar
  ?a2 & (not (symbolp ?a2)) & (numberp ?a2) & (< ?a2 ?a1)
  ?b2 & (not (symbolp ?b2)) & (numberp ?b2) & (> ?b2 ?b1)
  ?c2 & (not (symbolp ?c2)) & (numberp ?c2) & (< ?c2 ?c1))))

=>
(assert (sdl-173-matched ?case1 ?case2))
(assert (sdl-173-matched (?case1 ?case2))))

(defrule sdl-173-3 **
(sdl-173-matched case1 case2)
(sdl-173-matched (case1 case2))
(or (not (sdl-173-matched "case1 "case2))
  (not (sdl-173-matched ("case1 ("case2)))))

=>
(assert (success sdl-173))

(defrule sdl-173-4 **
(declare (salience -10))
(not (success sdl-173))
=>
(assert (fail sdl-173))

defrule sdl-174

=>
(assert (success sdl-174))

defrule sdl-174-3
(sdl-174-matched ?case1 ?case2)
(sdl-174-matched ?case1 ?case2)

=>
(assert (success sdl-174))

defrule sdl-175
(sdl-175 pallet ?pallet)
(sdl-175 options ?options)
(sdl-175 pallet ?pallet)
(sdl-175 options ?options)
(sdl-175 case ?asb
  & := (/ (- ?asb (mod ?asb 1000)) 1000) ?pallet)
(sdl-175 case ?asb
  & := 0 (POSITIONS ?asb ?options)) ; if not a member

=>
(assert (sdl-175-matched ?asb))
(assert (sdl-175-matched (?asb)))

defrule sdl-175-3
(sdl-175-matched ~5000)
(sdl-175-matched (~5000))

=>
(assert (fail sdl-175))

defrule sdl-176
(sdl-176 options ?options)
(sdl-176 options ?options)
(sdl-176 case ?asb
  & := (not (= (positions ?asb ?options) 0)))
(sdl-176 case ?asb
  & := (not (= (positions ?asb ?options) 0)))

=>
(assert (sdl-176-matched ?asb))
(assert (sdl-176-matched (?asb)))

124
(defrule sdl-176-3 **
  (sdl-176-matched "5000)
  (sdl-176-matched "5000))
=>
  (assert (fail sdl-178)))

(defrule sdl-177 **
  (sdl-177 options $?options)
  (sdl-177 (options $?options))
  (sdl-177 case "ASB
    & (= 0 (POSITIONS ?asb ?OPTIONS)))
  (sdl-177 (case "ASB
    & (= 0 (POSITIONS ?asb ?OPTIONS))))
=>
  (assert (sdl-177-matched ?asb))
  (assert (sdl-177-matched (?asb))))

(defrule sdl-177-3 **
  (sdl-177-matched "5000)
  (sdl-177-matched ("5000))
=>
  (assert (fail sdl-177)))

(defrule sdl-178 **
  (or (not (sdl-178 ?a
    & : (not (symbolp ?a))))
    (not (sdl-178 (?a
      & : (not (symbolp ?a)))))
=>
  (assert (fail sdl-178)))

(defrule sdl-179 **
  (sdl-179 options $?options)
  (sdl-179 (options $?options))
  (sdl-179 case "ASB)
  (test (not (= 0 (POSITIONS ?asb ?OPTIONS))))
=>
  (assert (sdl-179-matched ?asb))
  (assert (sdl-179-matched (?asb))))

(defrule sdl-179-3 **
  (sdl-179-matched "5000)
  (sdl-179-matched ("5000))
=>
  (assert (fail sdl-179)))

(defrule sdl-180 **
  (sdl-180 options $?options)
  (sdl-180 (case "ASB)
  (sdl-180 (options $?options))
  (sdl-180 (case "ASB)
  (test (= (POSITIONS ?asb ?OPTIONS) 0))
=>
  (assert (sdl-180-matched ?asb))
  (assert (sdl-180-matched (?asb))))

(defrule sdl-180-3 **
  (sdl-180-matched "5000)
  (sdl-180-matched ("5000))
=>
  (assert (fail sdl-180)))

(defrule sdl-181 **
ART/ADA DESIGN PROJECT - PHASE I

(sdl-181 5000)
(sdl-181 (5000))
(or (not (sdl-181 5000))
(not (sdl-181 (5000))))

=>
(assert (fail sdl-181)))

(defrule sdl-182 **
(declare (salience 100))
(sdl-182 "est & "psti est & "est")
(sdl-182 (?est & "psti est & "est"))

=>
(assert (sdl-182-matched ?est ?psti))
(assert (sdl-182-matched (?est ?psti))))

(defrule sdl-182-1 **
(or
(sdl-182-matched "psti "psti")
(sdl-182-matched "est & "est")
(sdl-182-matched "psti "est")
(sdl-182-matched ("psti "psti"))
(sdl-182-matched ("est & "est")
(sdl-182-matched ("psti "est")
(not (sdl-182-matched "est "psti")
(not (sdl-182-matched ("est "psti"))))

=>
(assert (fail sdl-182)))

(defrule sdl-183 **
(sdl-183 "(+ 1 1)
(sdl-183 "(+ 1 1)"

=>
(assert (fail sdl-183)))

(defrule sdl-184 **
(sdl-184 "2)
(sdl-184 "2")

=>
(assert (fail sdl-184)))

(defrule sdl-185 **
(sdl-185 "string")
(sdl-185 ("string")

=>
(assert (fail sdl-185)))

(defrule sdl-186 **
(declare (salience 100))
(sdl-186 "string & "string")
(sdl-186 ("string & "string")

=>
(assert (sdl-186-matched ?string))
(assert (sdl-186-matched (?string))))

(defrule sdl-186-1 **
(or
(sdl-186-matched "string")
(sdl-186-matched ("string")
(not (sdl-186-matched ("foo")
(not (sdl-186-matched ("foo"))))

=>
(assert (fail sdl-186)))

(defrule sdl-187 **
(declare (salience 100))
(sd1-187 "string1" "string" "string2" "string")
(sd1-187 (?string1 ?string2))

=>
(assert (sd1-187-matched ?string1 ?string2))
(assert (sd1-187-matched (?string1 ?string2)))

(defrule sd1-187-1 ""
  (or
    (sd1-187-matched "string" "foo")
    (sd1-187-matched ("string" "foo"))
    (not (sd1-187-matched "foo" "foo"))
    (not (sd1-187-matched ("foo" "foo"))))
=>
(assert (fail sd1-187))

(defrule sd1-188 ""
  (sd1-188 "="(string-append "foo" "bar"))
  (sd1-188 ("="(string-append "foo" "bar"))))
=>
(assert (fail sd1-188))

(defrule sd1-189 ""
  (sd1-189 (?a ?l) "="(+ 1 l))
  (sd1-189 (?a ?l) ("="(+ 1 l))))
=>
(assert (fail sd1-189))

(defrule sd1-190 ""
  (sd1-190 (?a :not (numberp ?a)))
  (sd1-190 (?a :not (numberp ?a))))
=>
(assert (sd1-190-matched ?a))
(assert (sd1-190-matched (?a)))

(defrule sd1-190-1 ""
  (or
    (sd1-190-matched 1)
    (sd1-190-matched (1))
    (not (sd1-190-matched foo))
    (not (sd1-190-matched (foo))))
=>
(assert (fail sd1-190))

(defrule sd1-191 ""
  (sd1-191 "1 & "0 1 & "foo" & "bar" & "2 & "12345.6789 & "fee" & "blee")
  (sd1-191 (?l & "0 1 & "foo" & "bar" & "2 & "12345.6789 & "fee" & "blee"))
=>
(assert (fail sd1-191))

(defrule sd1-192 ""
  (sd1-192 ?char-num :or (<= 97 ?char-num 102)
             (<= 65 ?char-num 70)
             (<= 48 ?char-num 57)))
  (sd1-192-seq (?char-num :or (<= 97 ?char-num 102)
             (<= 65 ?char-num 70)
             (<= 48 ?char-num 57))))
=>
(assert (sd1-192-matched ?char-num))
(assert (sd1-192-matched (?char-num)))

(defrule sd1-192-1 ""
  (or
    (sd1-192-matched 40))
(not (sd1-192-matched 50))
(sdl-192-matched 60)
(not (sd1-192-matched 70))
(sdl-192-matched 80)
(sdl-192-matched 90)
(not (sd1-192-matched 100))
(sdl-192-matched (40))
(not (sd1-192-matched (50)))
(sdl-192-matched (60))
(not (sd1-192-matched (70)))
(sdl-192-matched (80))
(sdl-192-matched (90))
(not (sd1-192-matched (100)))

=>
(assert (fail sd1-192)))

;; BDC additions to the rule base follow...

(defvar bdc-200-1
  "Does HALT work?"
  (declare (salience -1000))
=>
  (printout t t "TEST IS OVER" t)
  (halt))

(defvar bdc-200-2
  (declare (salience -1001))
=>
  (assert (fail bdc-200)))

(defvar bdc-201
  (bdc-201 a b c d e))

(defvar bdc-201-sequences
  (bdc-201 (a b c d e)))

(defvar bdc-201-1 **
  (bdc-201 $?data)
  (bdc-201 ($?data)))
=>
  (if (not (member$ b $?data))
      then (assert (fail bdc-201a))
  (if (member$ z $?data)
      then (assert (fail bdc-201b)))))