1-1 Introduction to ASPL

ASPL, A Set Programming Languages, is a *setadic calculator*. The word *setadic* is introduced by the author and is specifically used herein to refer to a calculator whose symbolic operators are termed *setadic operators* and all of which perform algebraic set operations on datasets. These operators always precede their operands.

A ***setadic calculator***, is a software appliance that takes structured data objects as input and breaks them into their constituent containment pathes so that they can be analyzed according to their structural components. ASPL maps algebraic data groups into containment pathes, and the appliance offers setadic operators to aggregate and to determine the relationships between their constituent elements. It renders them into a form suitable to be comprehended visually by the user before displaying their relationships on the user terminal.

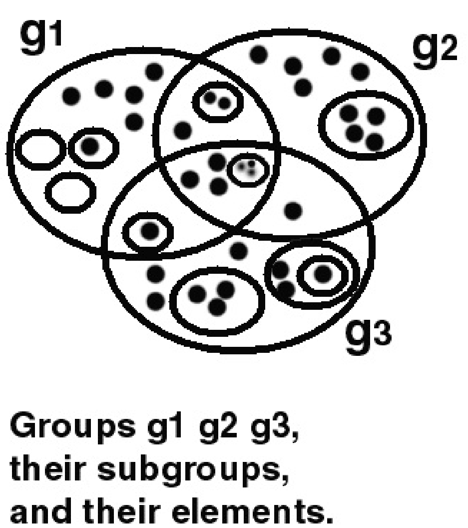
Therefore, ASPL is an interactive software appliance that is started on the UNIX shell command prompt. Once ASPL is installed on the computer system, the user can start using it interactively to do sophisticated set operations, or can run ASPL scripts on the system where ASPL is installed. The ASPL interpreter provides a powerful and intuitive interactive computing facility that shares some similarities to a classic calculator. However, ASPL operators are set operators, and its variables are set variables. For the operator using ASPL, all variables are typeless and the commands are short and simple mnemonics that are easy to remember. ASPL does not require a database or any third party libraries. The program runs on UNIX systems where the standard Perl interpreter is available.

Your ASPL interpreter can perform a large number of sophisticated set operations using the preprogrammed set operators. The calculation results may then be assigned to variables called *set variables*. The set variables are presented as arguments to set operators to do further calculations. In addition, when you start the ASPL interpreter you are assigned a named workspace. It is this named workspace that is used by the ASPL instance you started to save the results and the variables of all your calculations. ASPL uses a stack to save the results performed by its operations, and save the variables in internal symbol tables. ASPL variables are all global variables, and when persistence is enabled these global variables are all shared by users connecting to the same workspace; in which case if two or more ASPL instances are started with the same named workspace then all variables are being shared globally by all ASPL instances. The workspace directory should be available to all instances.

ASPL path containment can span multiple groups and subgroups along their elements. The following figure shows groups, subgroups, and their elements. There are three groups labeled g1, g2, and g3.

Note: ASPL groups image to be further explained in next Chapter using RANDONEBIT workspace

full view



-F- Fig. 1.1.1 [THREE GROUPS, THEIR SUBGROUPS, AND THEIR ELEMENTS]
[This figure shows three groups labeled g1, g2, and g3. Each group may contain

one or more subgroups or subsets. The elements within the groups and their subgroups are marked with black dots.]

ASPL © 2024 by Bassem W. Jamaleddine

The labeled groups can be assigned to ASPL variables. For instance, v1, v2, and v3 are three ASPL variables that represent the groups g1, g2, and g3 respectively. ASPL setadic operators can then perform set operations on these variables. For instance to get the set union of all three groups: **gU v1 v2 v3**, to get their set intersection: **g& v1 v2 v3**, to get their set difference: **g\ v1 v2 v3**, and to get their set partitions: **gP v1 v2 v3**.

Notice the mnemonic of these four basic set operators: gU, g&, g\, and gP in which the first letter is a lower case 'g' depicting the subject of on which the operation will take place, and the second letter (either one of U, &, \, and P) depicting the operation type. These four operators do the following operations, respectively:

```
Instruction      Operation
-----------    ------------------------
gU v1 v2 v3    get the group union considering the subrgoups and elements
g& v1 v2 v3    get the group intersection considering the subrgoups and elements
g\ v1 v2 v3    get the group difference considering the subrgoups and elements
gP v1 v2 v3    partition the groups, subgroups, and their elements
```

In a functional way these operations can also be further explained:

```
Instruction      Operation
-----------    ------------------------
gU v1 v2 v3    Union(v1, Union(v2,v3))
g& v1 v2 v3    Intersection(v1, Intersection(v1,v2))
g\ v1 v2 v3    Difference(v1, Difference(v2,v3)
gP v1 v2 v3    Partition(v1,v2,v3)
```

The subject denoted with the 'g' is to consider both the subgroups and elements in their respective groups. However, if the subject is only to do the operations on the subgroups then one can use dU, d&, d\ and dP. Therefore to get the set union of *the subgroups only* of all the three groups: **dU v1 v2 v3**, to get their set intersection: **d& v1 v2 v3**, to get their set difference: **d\ v1 v2 v3**, and to get their set partitions: **dP v1 v2 v3**.

One can look into the elements of the groups only (excluding the subgroups) by using fU, f&, f\ and fP. Therefore to get the set union of *the elements only* of all three groups: **fU v1 v2 v3**, to get their set intersection: **f& v1 v2 v3**, to get their set difference: **f\ v1 v2 v3**, and to get their set partitions: **fP v1 v2 v3**.

For ASPL this is done through its *set builder and the setadic operation*. For example, the intersection of the elements found in v1, v2, and v3, denoted with **f& v1 v2 v3** is to parse (or humanly read) the information left to right: the set operator comes first (hence the meaning of *setadic operator*) then followed by the identifiers (representing the labeled groups or datasets). This is something that the reader is accustomed to from basic schooling, and can be seen through ASPL setbuilder by typing **setbuilder gU v1 v2 v3** at the ASPL prompt:

```
aspl> setbuilder gU v1 v2 v3

  QUOTIENT SET BUILDER

   {gU v1 v2 v3} <=>  gU   v1   v2   v3

  Detailed view:

   {gU v1 v2 v3} <=>

    gU   v1   v2   v3
     |    |    |    +----> set-variable
     |    |    +---------> set-variable
     |    +--------------> set-variable
     +------------------> get the subgroups and the elements union

  Set builder syntax is read from left to right, or from bottom to top.
  All ASPL setops are setadic: they take a setop followed by set variables.
```

As for the intersection, one can type **setbuilder g& v1 v2 v3** at the ASPL prompt:

```
aspl> setbuilder g& v1 v2 v3

  QUOTIENT SET BUILDER
```

```
 {g& v1 v2 v3} <=>  g&   v1   v2   v3

Detailed view:

 {g& v1 v2 v3} <=>

  g&   v1   v2   v3
   |   |    |    +----> set-variable
   |   |    +---------> set-variable
   |   +--------------> set-variable
   +------------------> get the subgroups and the elements intersection

Set builder syntax is read from left to right, or from bottom to top.
All ASPL setops are setadic: they take a setop followed by set variables.
```

Each element within a group or subgroup can be further described and characterized by a set of attributes that are attached to it, and comparing it to another element is therefore based on these attributes. In ASPL every element has a checksum attribute, and the language provides a feature to subordinate setadic operators with a conditional predicate. Repeating the previous operation g& v1 v2 v3 but we want to get only these subgroups and elements that have different checksums: **setbuilder g&`c v1 v2 v3**

```
aspl> setbuilder g&`c~ v1 v2 v3

  QUOTIENT SET BUILDER

   {g&`c~  v1 v2 v3} <=>  g&`c~  v1   v2   v3

  Detailed view:

   {g&`c~  v1 v2 v3} <=>

    g& `  c~  v1   v2   v3
     | |  |   |    |    +----> set-variable
     | |  |   |    +---------> set-variable
     | |  |   +--------------> set-variable
     | |  +------------------> have different checksums
     | +---------------------> such that
     +-----------------------> get the subgroups and the elements intersection
```

Here we used the acute backtick ` followed by a predicate. ASPL refers to this backtick simply with the word 'tick' and many of ASPL set operators can be ticked with specific predicates. For example, in the above example the operator is ticked with the predicate saying "when checksums are differents" ***g&`c~***.

In general, we are also accustomed to the forward slash, called a stroke or the solidus symbol /, that may follow a set operator to depict a quotient relation, and in ASPL this is called *stroking a set operator*. In the following example, the group intersection operator (g&) is being stroked with the quotient relation r3. Notice that the stroke is directly followed by a tilde then the relation r3. r3 has been defined by the user but expanded here as *frx=.*$,mtm~* (that is saying to match the elements with regular expression .*$ and whose mtime attribute differ).

```
aspl> setbuilder g&/~r3  v1 v2 v3

  QUOTIENT SET BUILDER

   {g&/~r3  v1 v2 v3} <=>  g&/frx=.*$,mtm~  v1   v2   v3

  Detailed view:

   {g&/~r3  v1 v2 v3} <=>

    g& / frx=.*$,mtm~  v1   v2   v3
     | |    |      |    |    |    +----> set-variable
     | |    |      |    |    +---------> set-variable
     | |    |      |    +--------------> set-variable
     | |    |      +-------------------> have different make times
     | |    +--------------------------> file name regular expression
     | +-------------------------------> stroking the Quotient Relation
     +---------------------------------> get the subgroups and the elements intersection
```

The definition of r3 was performed with the command shown below and at any time the user can type q to display the quotient relation table:

```
   aspl> q r3 := frx=.*$,mtm~
   aspl> q

  Coded QR
        qr                |user          |code
        ------------------|--------------|-----------------------------------
        r3                |(1)root       |frx=.*$,mtm~
```

This is equivalent to typing g&`mtm~ v1 v2 v3 and in this particular case all the following three commands are equivalent:

```
aspl> g&/~r3  v1 v2 v3
aspl> g&/frx=.*$,mtm~  v1   v2   v3
aspl> g&`mtm~  v1   v2   v3
```

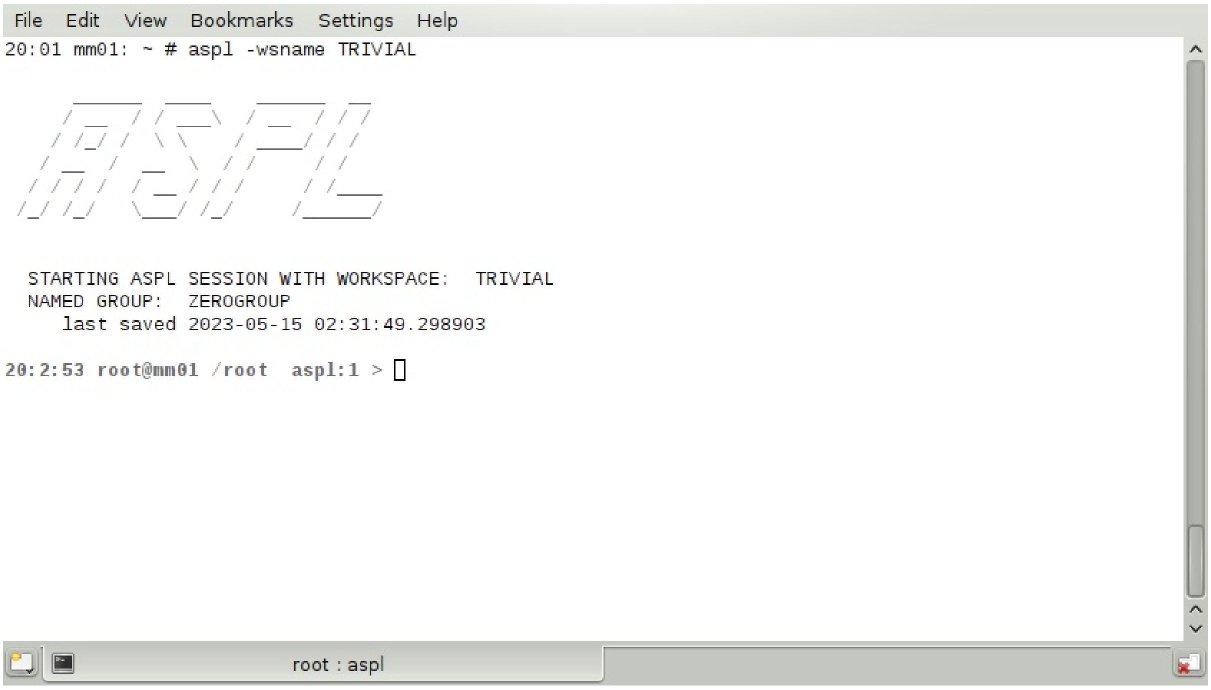
> In this book the ASPL prompt is denoted with **aspl>** and the UNIX shell prompt is represented with the hash **#** symbol.

See appendix AAAAA on how to install ASPL on your UNIX system.

■ ASPL Building Blocks

Note: ASPL Startup TRIVIAL ASPLv1.00-startup-trivial.png

full view



-F- Fig. 1.1.2 [ASPL Startup TRIVIAL][ASPL Starting with TRIVIAL workspace]

ASPL © 2024 by Bassem W. Jamaleddine

(* footnote: An interpreter typically has it own virtual machine, but ASPL does not. The ASPL interpreter is built on top of pure Perl interpreter and it uses the Perl powerful virtual machine in executing its tasks.)

The TRIVIAL workspace is available with every ASPL distribution, and its grouping class is the ZEROGROUP which contains the bare information that is required by any grouping class in ASPL. The command **wid** displays the current workspace being loaded and its up time, and the command **egCwhoami** pings its grouping class container. Both commands are shown in Figure FFFFF.

Note: ASPL Startup TRIVIAL ASPLv1.00-startup-trivial.png

full view

```
File  Edit  View  Bookmarks  Settings  Help
20:19 mm01: ~ # aspl -wsname TRIVIAL

      ______   ____    ______   __
     / __  /  / ___\  / __  /  / /
    / /_/ /   \ \    / ____/  / /
   / __  /  __  \   / /      / /
  / / / /  / __/ / / /      / /____
 /_/ /_/   \___/ /_/       /______/


  STARTING ASPL SESSION WITH WORKSPACE:  TRIVIAL
  NAMED GROUP:  ZEROGROUP
     last saved 2023-05-15 02:31:49.619851

20:19:42 root@mm01 /root  aspl:1 > wid
     answers       6
     enswers       0
     unswers       0
  operations       6
  uptime              0 days  0 hours  0 minutes   1 seconds
  TRIVIAL             0 days  0 hours  0 minutes   1 seconds

20:19:44 root@mm01 /root  aspl:2 > egCwhoami

>>>>>>>>>>>>>I AM ZEROGROUP<<<<<<<<<<<<< being called here at proc_sub_whoami
   BRIDGE=/opt/ASPLv1.00/BRIDGE
   ASPL1_00_BRIDGE=/opt/ASPLv1.00/BRIDGE
   ASPL1_00_HOME=/opt/ASPLv1.00
   ASPL/Groupings/Elements/ZEROGROUP/Enode.pm
   /opt/ASPLv1.00/BRIDGE/ASPL/Groupings/Elements/ZEROGROUP/Enode.pm

   ASPL::Groupings::Elements::ZEROGROUP::Enode LINE#9 caller: ASPL::Groupings::Helper::Enode
   ASPL::Groupings::Elements::ZEROGROUP::Enode LINE#18 caller: ASPL::Groupings::Elements
   zisFn0=ASPL::Groupings::Elements::ZEROGROUP::Enode::proc_sub_whoami
   zisFn=ASPL::Groupings::Elements::ZEROGROUP::Enode::__ANON__
   self=ASPL::Groupings::Elements::ZEROGROUP::Enode=HASH(0x7d800d0)
   arg=

   Ksumatt= ARRAY(0x7d829c8) = mtime chksum entropy
   Bnode= ARRAY(0x7a9e028) = mtime aelm
   Enode= ARRAY(0x78d5378) = mtime aelm chksum entropy ppdd ffl dosi
   confess = 0


20:19:47 root@mm01 /root  aspl:3 > 
root : aspl
```

-F- Fig. 1.1.3   [ASPL Startup TRIVIAL][ASPL Starting with TRIVIAL workspace]

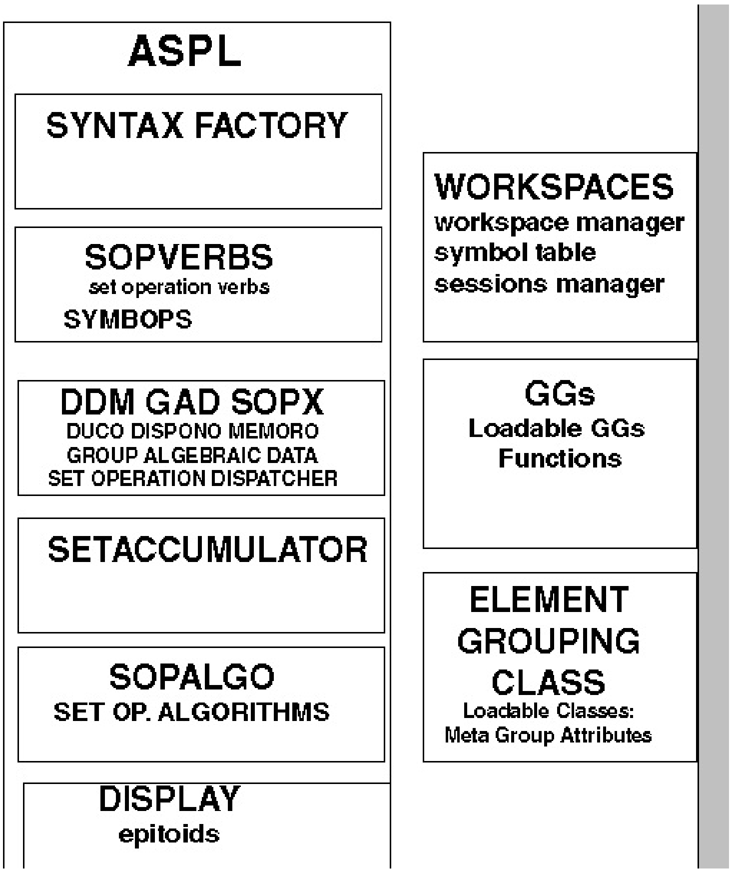
ASPL has eight major components:

- the syntax factory
- the set semantic processor
- the verbs transformer and dispatcher
- the set accumulator
- the set algorithmic processor
- the dynamically loadable container for the grouping classes
- the dynamically loadable container for generative functions
- the workspace manager for symbols and sessions management

These are shown in Figure FFFFF, and are also reflected in ASPL layout directories shown in Figure FFFFF.

Note: ASPL building blocks image aspl-block-old.png

full view



-F- Fig. 1.1.4 [ASPL BLOCK][ASPL Internal Building Blocks]

[Top Text]

-L- Listing. 1.1.1 [ASPLv1.00 Directory Tree ASPL-tree0.dir.numl][ASPLv1.00 Directory Tree]

(raw text)

```
ASPLv1.00
|-- bin
|-- BRIDGE
|   |-- ASPL
|   |   |-- BAYLEVELGROUP
|   |   |   `-- Feeder
|   |   |-- EmStatv2
|   |   |   `-- Cexec
|   |   |-- EmVectors
|   |   |-- Formattings
|   |   |-- GGs
|   |   |   |-- GEO3TRI
|   |   |   |   `-- CTXSETOP
|   |   |   |-- OSCILLATORSGROUP
|   |   |   |   `-- Feeder
|   |   |   `-- SYSENVGROUP
|   |   |       `-- Feeder
|   |   |-- Groupings
|   |   |   |-- Elements
|   |   |   |   |-- ONEGROUP
|   |   |   |   `-- ZEROGROUP
|   |   |   |-- Helper
|   |   |   `-- stubs
|   |   `-- MockedGroupings
|   |       `-- MockedObjects
|   `-- bin
|-- etc
|-- lib
|   |-- ASPL
|   |   |-- ASPLSNTX
|   |   |-- Directory
|   |   |   `-- DDM
|   |   |-- DISPLAY
|   |   |-- MemUsage
|   |   |-- PRIMITIVES
|   |   |-- SETACCUMULATOR
|   |   |-- SOPALGO
|   |   |-- SOPVERBS
|   |   |-- SOPX
|   |   |-- SYMBOPS
```

```
        |   |   |   `-- FTX_VERBPROCESSORS_BUILDER
        |   |   `-- Utilities
        |   `-- Simple
        |-- license
        `-- shared
```


[Top Text]

-L- Listing. 1.1.2 [ASPLv1.00 Directory Tree ASPL-tree0.dir.numl][ASPLv1.00 Directory Tree]

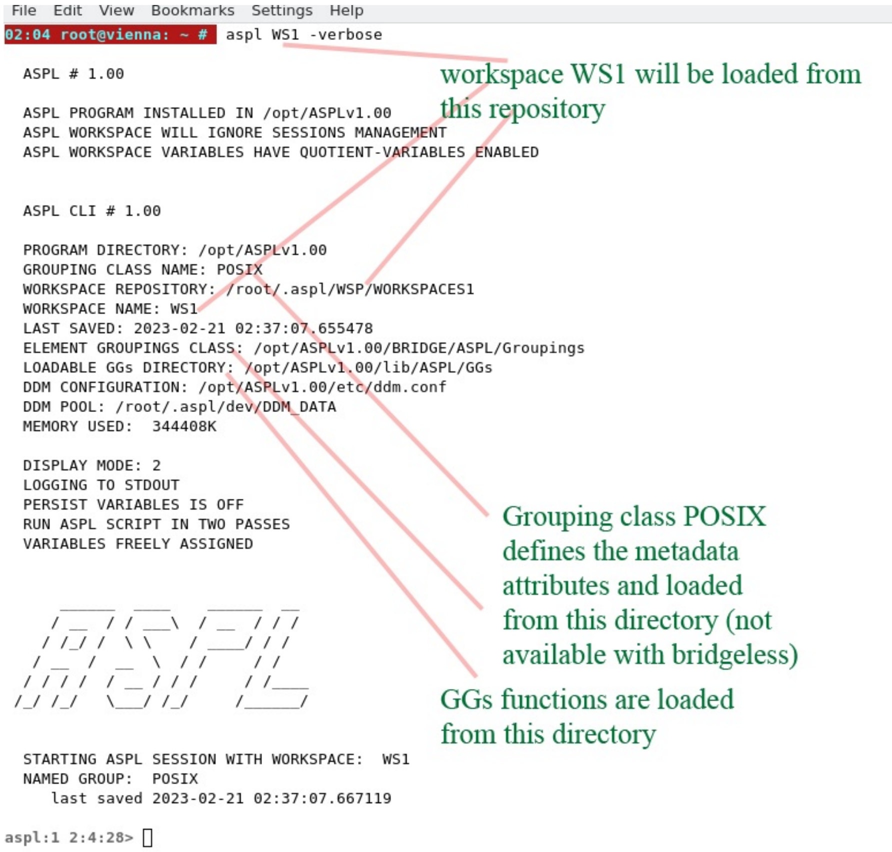
(raw text)

```
ASPLv1.00
|-- bin
|-- BRIDGE
|   |-- ASPL
|   |   |-- BAYLEVELGROUP
|   |   |   `-- Feeder
|   |   |-- EmStatv2
|   |   |   `-- Cexec
|   |   |-- EmVectors
|   |   |-- Formattings
|   |   |-- GGs
|   |   |   |-- GEO3TRI
|   |   |   |   `-- CTXSETOP
|   |   |   |-- OSCILLATORSGROUP
|   |   |   |   `-- Feeder
|   |   |   `-- SYSENVGROUP
|   |   |       `-- Feeder
|   |   |-- Groupings
|   |   |   |-- Elements
|   |   |   |   |-- ONEGROUP
|   |   |   |   `-- ZEROGROUP
|   |   |   |-- Helper
|   |   |   `-- stubs
|   |   `-- MockedGroupings
|   |       `-- MockedObjects
|   `-- bin
|-- etc
|-- lib
|   |-- ASPL
|   |   |-- ASPLSNTX
|   |   |-- Directory
|   |   |   `-- DDM
|   |   |-- DISPLAY
|   |   |-- MemUsage
|   |   |-- PRIMITIVES
|   |   |-- SETACCUMULATOR
|   |   |-- SOPALGO
|   |   |-- SOPVERBS
|   |   |-- SOPX
|   |   |-- SYMBOPS
|   |   |   `-- FTX_VERBPROCESSORS_BUILDER
|   |   `-- Utilities
|   `-- Simple
|-- license
`-- shared
```


Starting ASPL in verbose mode will also reflect where ASPL loads its components at startup as shown in Figure FFFFF.

ASPL Startup on the UNIX Shell Prompt Explained

full view



-F- Fig. 1.1.5 [ASPL Startup on the UNIX Shell Prompt Explained]

ASPL © 2024 by Bassem Jamaleddine

A user interacting with ASPL places a setadic operation to the interpreter. The interpreter submits it to the syntax factory, analyzes it, builds its object, and locates its verb. It then dispatches the object to its algorithmic set processing agent after fetching its symbols from either the answer stack or the symbol table. The result of the processed expression is finally saved in the symbol table, pushed on the top of the stack, and displayed on the terminal. The simplest way to see the flow of ASPL processing is to enable the tracing facility of ASPL (see Appendix CCCCC).

■ ASPL Commands with asplcmd

At this point we introduce the **asplcmd** command because it is a quick way to execute short ASPL statements through the UNIX shell prompt.

You can direct ASPL to execute some of its statements by issuing **asplcmd** at the shell prompt followed by a a string. If the string is more than a word then it must be quoted since it is passed like a single argument to asplcmd. Without going into details on how asplcmd works, as this will become clear after reading the chapter "ASPL Scripts", it is used here to facilitate some explanation on ASPL displayable output.

The following command compares the datasets saved in the variable mg123 of workspace RANDONEBITMIX.

\# **asplcmd 'load RANDONEBITMIX;gU mg123'**

This command causes ASPL to load the workspace RANDONEBITMIX, then issues the **gU** to display the group unions found in the datasets saved in the variable mg123. The semicolon is used to separate the two statements. Observe the output showing the groups, subgroups, and elements. The next section explains the symbolic schemes and colors used by ASPL when displaying its output.

The following command compares two variables a2 and a7 that has been saved in workspace WS1.

\# **asplcmd 'load WS1; ks mtime chksum ppdd; gU a2 a7'**

There are three statements separated by semicolons in this string passed to ASPL. The first statement load the workspace WS1, the second statement set the ks attribute vector to mtime chksum ppdd, and the third statement display the group union of the datasets saved in a2 and a7.

When you pass a string to asplcmd the interpreter will parse the string and execute its contents then exit.

■ ASPL Symbolic Shemes and Colors

ASPL is not a GUI application and does not require any GUI API, it uses plain ASCII characters for its symbolic operators and identifiers, and uses the ANSI colors scheme provided by the UNIX terminal. (* footnote: APL users should be alarmed as there is no APL Greek symbols in here!) At any time you can start ASPL at the UNIX shell prompt of a terminal (or even on a dumb terminal) or an X Window session. Figure FFFFF-1 shows ASPL started in a KDE session.

Because the set comparison operators classify data into three basic categories: intersection, union, and difference, then it is recommended to display the output using three different colors. In addition, since a group may contain subgroups and elements, then some symbolic scheme is needed to show the group differences along these colors. ASPL uses both colors and symbolic schemes to facilitate the readability of its output. Using the ANSI colors that is readily available on all UNIX terminal, and compounding it with symbolic schemes will help to distinguish when subgroups and elements are unique, equal, or different.

Figure FFFF shows the partial result of the command discussed in the previous section:

```
# asplcmd 'load RANDONEBITMIX;gU mg123'
```

Recall that there is no restriction on the name of subgroups and elements, that is an element name can be the subgroup name in another group, and even can be the subgroup name in the same group (as long as it is at different level). This is similar to the UNIX filesystem where in the same directory you cannot have a directory and a file using the same name.

To understand the meaning of ASPL colors and symbolic schemes let's go over figure FFFF where the output has been labeled at five points as follow:

① **gU: f!=) this is the gU of g1 g2 g3 where a difference in the element has been detected: it is the subgroup df2 that was detected in the subgroup ./df2/df1**

② **gU: +f) element df3 only in subgroup ./df2/df1/ in group g3**

③ **gU: d!=) subgroup df2 in subgroup ./df2/ in all groups g1 g2 and g3 but they differ**

④ **gU: f==) element df1 in subgroup ./df2/df2/ in all groups g1 g2 and g3 and is the same**

⑤ **gU: g!=) mixed as df3 is ./df2/ is subgroup in groups g1 g2 and it is an element in group g3**

Notice also the colors: equality shown in green, difference shown in red, and a loner shown in gray.

Note: ASPL Symbolic Shemes and Colors g1g2g3-12345.png

full view

-F- Fig. 1.1.6 [SYMBOLIC SHEMES AND COLORS][ASPL Symbolic Shemes and Colors]

This example might be confusing, but it was selected on purpose to make it clear that the label names of subgroups and elements are immaterial to ASPL algorithmic routines.

Let's take a look at another practical example where a file name and a directory name might intersect on the UNIX filesystem. Assuming you have access to the /tmp directory, issue the following commands on your shell prompt:

\# **mkdir /tmp/foodir1**

\# **mkdir /tmp/foodir1/abc**

\# **mkdir /tmp/foodir2**

\# **touch /tmp/foodir2/abc**

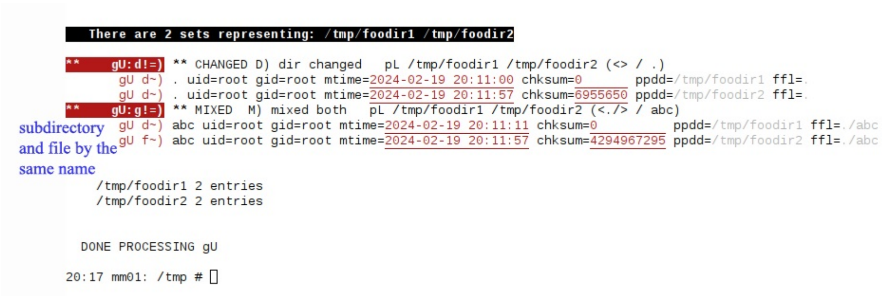
\# **asplcmd "createworkspace TRANSIENT POSIX;ggdir(dir,/tmp/foodir1);ggdir(dir,/tmp/foodir2);gU"**

The first two commands create a directory and a subdirectory /tmp/foodir1 and /tmp/foodir1/abc respectively, and the next two commands create a directory and a file /tmp/foodir2 and /tmp/foodir2/abc respectively. The last command call ASPL to show the difference between the two directories /tmp/foodir1 and /tmp/foodir2.

The statement *createworkspace TRANSIENT POSIX* tells ASPL to load the temporary TRANSIENT workspace with element grouping class POSIX. The next statement ggdir(dir,/tmp/foodir1) tells ASPL to call the grouping function ggdir() on directory /tmp/foodir1. These statements will become clearer in the next chapters. For now, if you have issued these command succefully then you are already using ASPL.

The result of comparing the comparing the directories is shown below: **Note: Example comparing a file and a subdirectory that have the same name foodir-commented.png**

full view



-F- Fig. 1.1.7 [Example comparing a file and a subdirectory that have the same name]

ASPL © 2024 by Bassem W. Jamaleddine

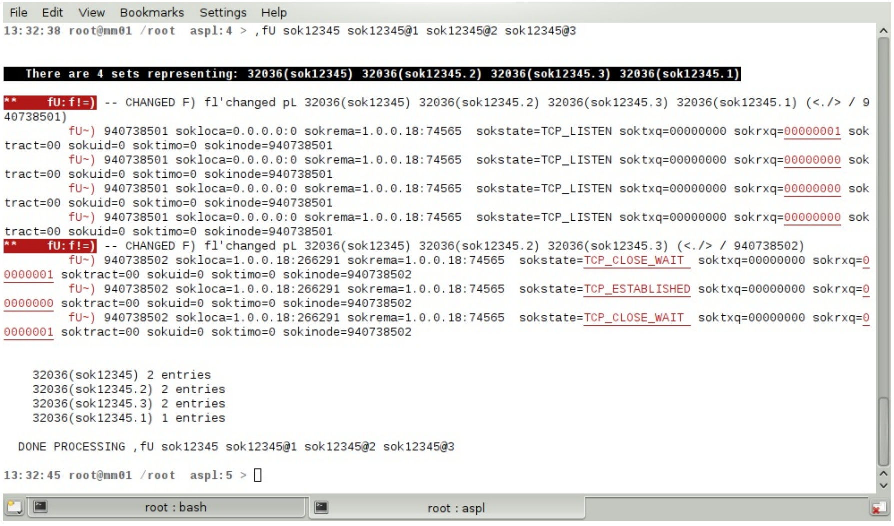
Appendix DDDD shows a summary of ASPL symbolic schemes and how the user can edit the color configuration settings.

■ Why do you need to use ASPL

There are myriad reasons why you need to use the ASPL interpreter: to detect and highlight changes of systemic data in a UNIX cloud environment, to validate configuration data in a UNIX cluster, to resolve collision of class names in JAR archives, to compare analogous PATH across systems, etc. The following figure shows the result of a UNIX system whose socket has changed states:

Figure monitor-socket-state.png

full view



-F- Fig. 1.1.8 [Figure Monitoring UNIX System Socket State]

ASPL © 2024 by Bassem Jamaleddine

Furthermore one can use ASPL to do simulation of players throwing dice on a crap table. Figure FFFFF shows the result of the simulation of players tossing dice: find the event when all the three players have the same outcome.

Figure randomdice.aspl-900.png

full view

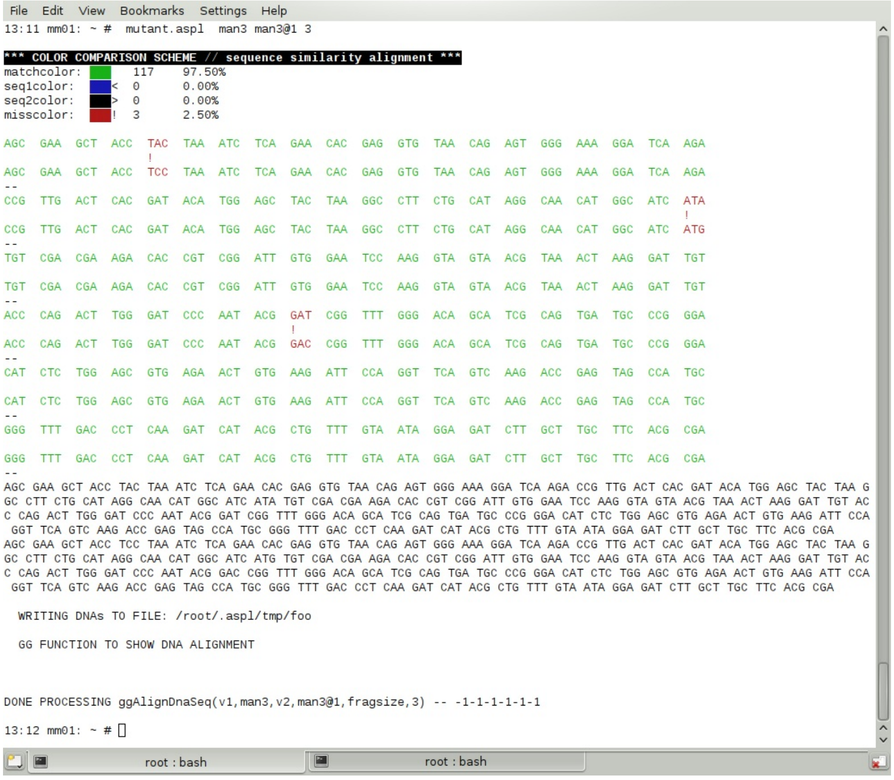
-F- Fig. 1.1.9 [Figure Simulation for Three Players Throwing Dice]

ASPL © 2024 by Bassem Jamaleddine

ASPL can also treats DNA sequences as datasets and you can toy with these sequences through its alignment algorithms. Figure FFFFF shows a mutant when altering the sequence randomly, and figure FFFFF show the DNA sequence alignment of two men.

Figure mutant.aspl-img1.png

full view

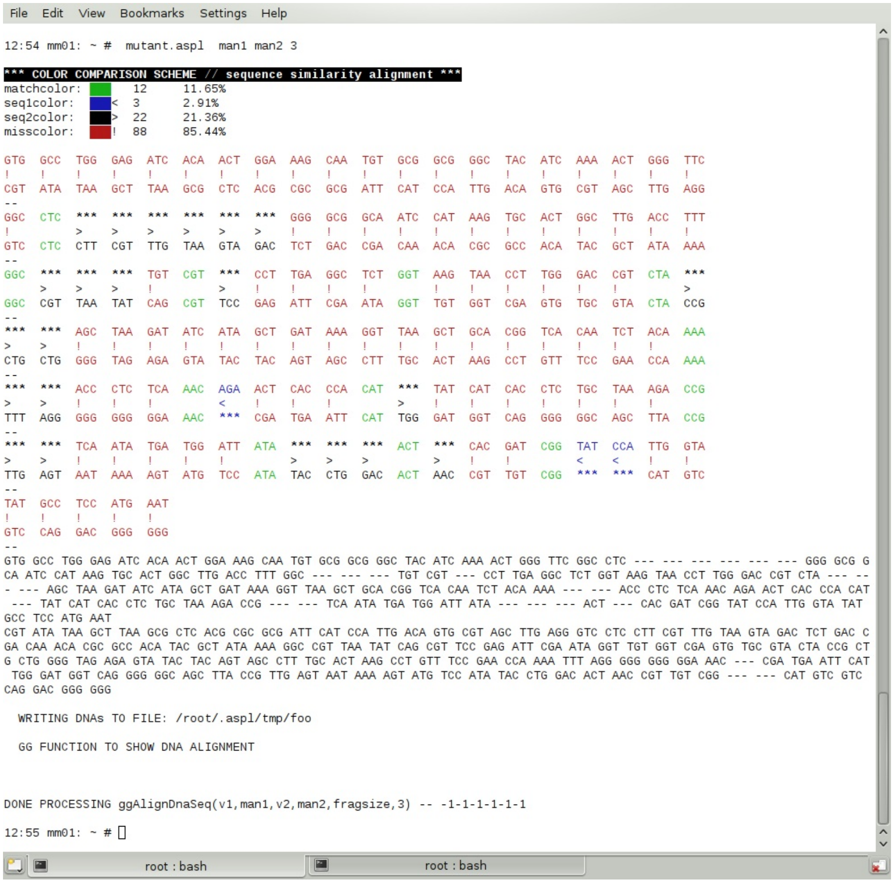


-F- Fig. 1.1.10 [Figure DNA Sequence for Mutant]

ASPL © 2024 by Bassem Jamaleddine

Figure mutant.aspl-img.png

full view



-F- Fig. 1.1.11 [Figure DNA Sequence of Two Men]

ASPL © 2024 by Bassem Jamaleddine

The elements of a group do not need to be the usual static data, as an element attributes can be renewed when hooked to processes or tied to devices in a system that may change states. We will see some examples in "ASPL by Examples" where the OSCILLATORGROUP defines attributes that are tied to a device to collect data set; and another example using the BAYLEVELGROUP whose attributes are tied to sensors monitoring the water level between two bays. Figure FFFFF shows a simulation when the water level between the two bays is critical.

BAYS12MON-img0.png

full view

-F- Fig. 1.1.12 [Figure Monitoring the Water Level Between Two Bays]

ASPL © 2024 by Bassem Jamaleddine

ASPL can do fuzzy set operations, these are simple operators like **y&, yU, and y**\ to get the fuzzy intersection, union, and difference respectively.

With its powerful regular expression processor, ASPL can do shallow set operations. Just type **shallowed** and you select the shallow matching that you desire.

A quick view of the shallow table module is shown below.

```
aspl:1 > shallowed

  THE SHALLOW MATCHING IDENTIFIER TO SELECT THE ROUTINE WHEN SHALLOW SETOPS ARE USED:

                  IDENTIFIER  DESCRIPTION
                  ----------  -----------
                     nothing  matching nothing at all
                    starstar  matching anything and everything
               matchandmatch  matching the ./subgroups and the element
                matchormatch  matching the ./subgroups or the element
                        elem  matching just the element and ignoring ./subgroups
                        stem  matching just the ./subgroups and ignoring the element
             endjoinedeither  matching from end of ./subgroups/element for either
             begjoinedeither  matching the beginning of ./subgroups/element for either
                     endstem  matching just the ./subgroups from the end and ignoring the element
               endstemeither  matching just the ./subgroups from the end and ignoring the element (for either)
                     begstem  matching from beginning of ./subgroups while ignoring the element
               begstemeither  matching from beginning of ./subgroups while ignoring the element (for either)
                  piecedstem  matching at least one piece in ./subgroups while ignoring the element
   >          piecedstemelem  matching the element and at least one piece in the ./subgroups
                     begelem  matching from the beginning of element and ignoring ./subgroups (for either)
                     endelem  matching from the end of element and ignoring ./subgroups (for either)
                     rgxelem  matching the element anywhere and ignoring ./subgroups (for either)
                    prcrelem  apply processor when matching the element anywhere and ignoring ./subgroups (for ei
```

```
  CURRENTLY LOADED piecedstemelem


 WHEN SELECTING prcrelem AS THE SHALLOW MATCHING IDENTIFIER, ONE OF THE FOLLOWING
 NODE PROCESSOR IDENTIFIER CAN BE SELECTED:

                IDENTIFIER EVAL  DESCRIPTION
                ---------- ----  -----------
                   transac    1  capture the element where word Transaction occured, ignore case
                    cla2ja    1  substitute .class with .java
                    ja2cla    1  substitute .java with .class
                        uc    1  upper case
                        lc    1  lower case
  >                   asis    1  neutral without any change

  CURRENTLY LOADED PROCESSOR asis

THE shallowedMatches PACKAGE CAN BE EDITED TO ADD MORE MATCHING SUBROUTINES.
SEE ASPL CONFIGURATION FILES FOR MORE ABOUT EDITING shallowedMatches PACKAGE.
```

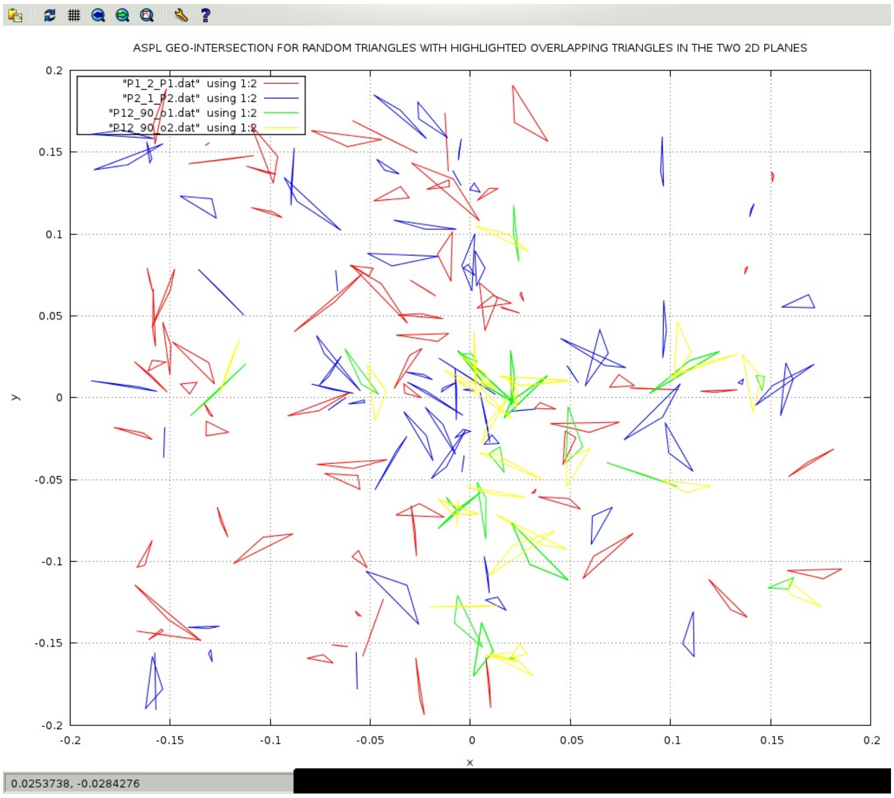
Moreover ASPL can switch its set operation mode to do geometric set operations so that you can detect intersecting polygons in 2D planes or on spheres.

Figure FFFFF shows the intersecting polygons in two planes. The intersecting polygons are contrasted by showing their vertex in squares.

P12_90-both-and-intersect-img2.png

Figure P12_90-both-and-intersect-img2.png

full view



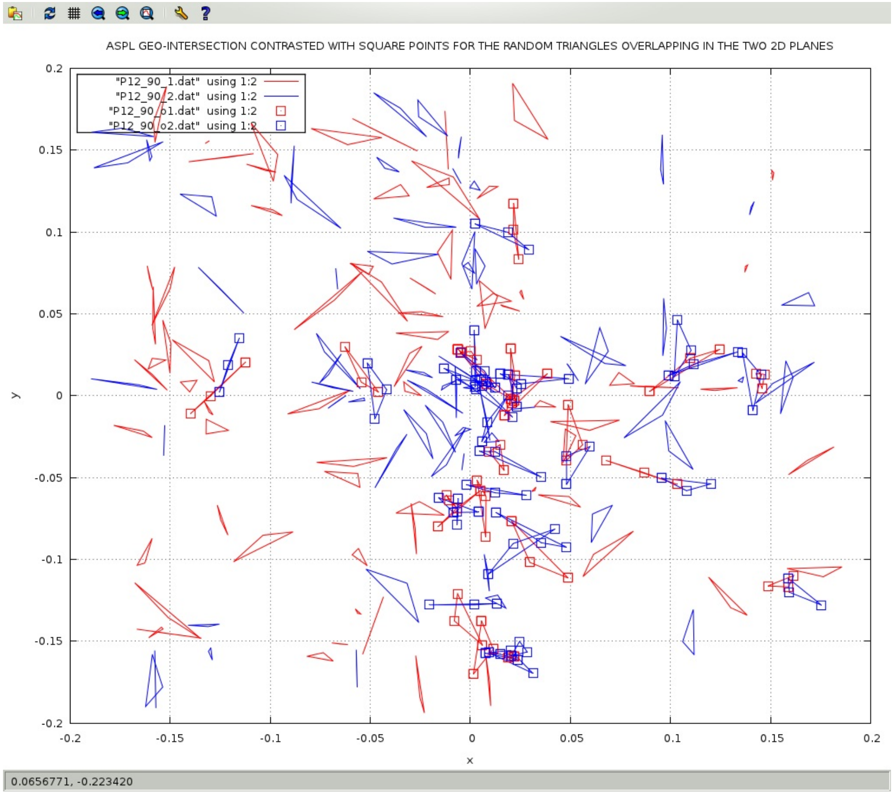
-F- Fig. 1.1.13 [Figure ASPL GEO-INTERSECTION FOR RANDOM TRIANGLES WITH HIGHLIGHTED OVERLAPPING TRIANGLES IN THE TWO 2D PLANES]

P12_90-both-and-intersect-img1.png

PUBLISHER VERSION USING SQUARE POINTS FOR BLACK AND WHITE PRINT

Figure P12_90-both-and-intersect-img1.png

full view



-F- Fig. 1.1.14 [Figure ASPL GEO-INTERSECTION CONTRASTED WITH SQUARE POINTS FOR THE RANDOM TRIANGLES OVERLAPPING IN THE TWO 2D PLANES]

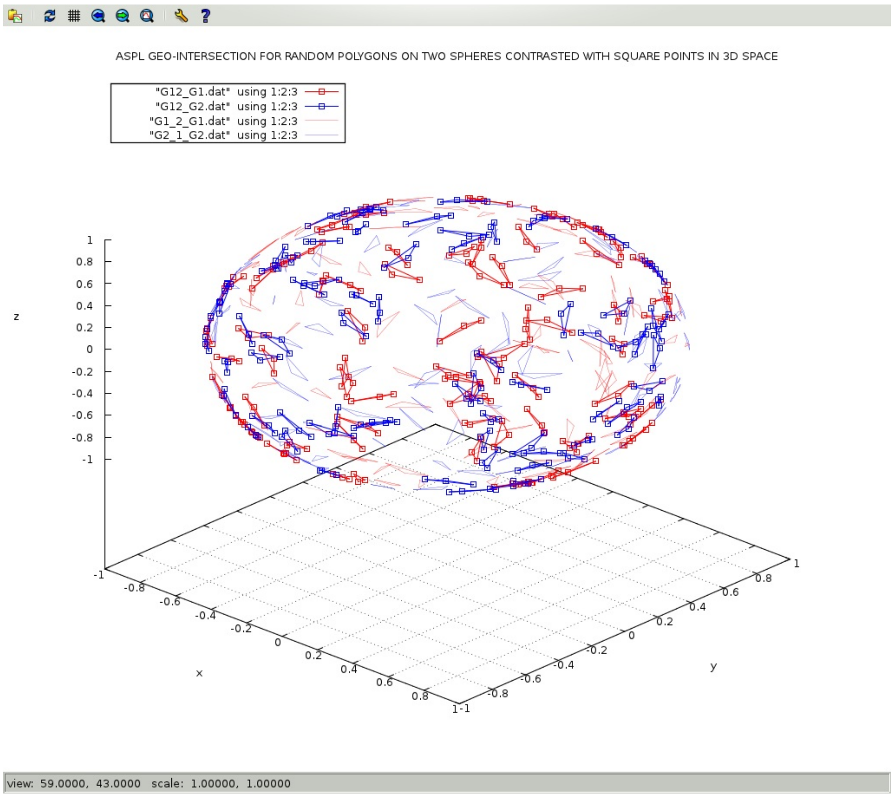
ASPL © 2024 by Bassem Jamaleddine

Figure FFFFF shows the intersecting polygons on two spheres, and figure FFFF shows the intersects and differences between these polygons.

G12_170-all-with-intersect-img3D.png

Figure G12_170-all-with-intersect-img3D.png

full view



-F- Fig. 1.1.15 [Figure RANDOM POLYGONS OVERLAPPING ON TWO SPHERES IN 3D SPACE]

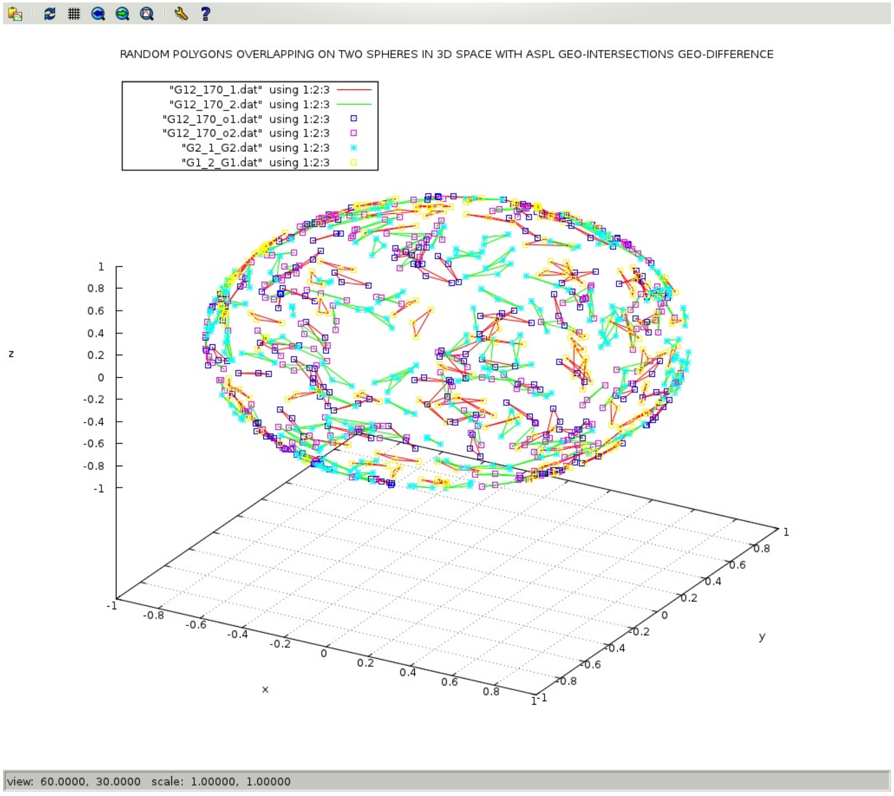
ASPL © 2024 by Bassem Jamaleddine

Refer to **WARN: IN Introduction-to-ASPL.raw THE FOLLOWING Ref: CANNOT BE RESOLVED**
<Ref:-see section -chaptit "RANDOM POLYGONS ON THREE SPHERES" -ptt "RANDOM POLYGONS ON THREE SPHERES" />

G12_170-both-all-img3D.png

Figure G12_170-both-all-img3D.png

full view



-F- Fig. 1.1.16 [Figure RANDOM POLYGONS OVERLAPPING ON TWO SPHERES IN 3D SPACE WITH ASPL GEO-INTERSECTIONS GEO-DIFFERENCE]

ASPL © 2024 by Bassem Jamaleddine

Many of these examples are detailed in **WARN: IN Introduction-to-ASPL.raw THE FOLLOWING Ref: CANNOT BE RESOLVED**
<Ref:-see section -chaptit "ASPL by Examples" -ptt "ASPL by Examples" />

Chapter Ref:ASPL by Examples

In addition the interpreter offers ***the ASPL scripting language*** so that you can enhance your system with powerful commands. Consider the following script that shows the differences in JAR archives. It can be called with a simple command **jarcompare.aspl jarfile1 jarfile2**.

[Top Text]

LISTING jarcompare.aspl ASPL Script jarcompare.aspl

(raw text)

```
#!/usr/bin/env aspl
#ENVARG= -groupingclass POSIX -wsname TRANSIENT -singlepass

;;**************************************************************************
;;   jarcompare.aspl
;;   Compare two JAR archives
;;   Copyright Â© 2024 Bassem W. Jamaleddine
;;
;;**************************************************************************

endScriptIfShellArgsLessThan 2

;;DEF FN cmp2sets := {gU {g\, %%1 %%2}{g\, %%2 %%1}{g&, %%1 %%2}}

timeout 60
displayoff
d1 = ggjar(jarfile,$1,calchksum,1,calentropy,1)
d2 = ggjar(jarfile,$2,calchksum,1,calentropy,1)

displayon
ks chksum size ffl
```

```
;;print ############################################
;;print # SHOWING SET COMPARISONS BETWEEN $1 and $2
;;print ############################################
;;FN cmp2sets(d1,d2)
print ############################################
print # SHOWING SET INTERSECTION WITH DIFFERENT CHECKSUMS BETWEEN $1 and $2
print ############################################
f&`c~ d1 d2
print ############################################
print # SHOWING SET SIMILARITY BETWEEN $1 and $2
print ############################################
sim d1 d2
println

endscript

__END__

  $00 compares two JAR archives

  $00 must be followed by the names of two JAR archives

  Example:
    To compare the two JAR archives /tmp/TX/27238-tx.jar and /tmp/TX/38141-tx.jar
    $00 /tmp/TX/27238-tx.jar /tmp/TX/38141-tx.jar

```

ASPL © 2024 by Bassem Jamaleddine

Figure FFFFF shows the result of jarcompare.aspl when comparing two JAR achives. Notice how the colliding names of the classes have been detected and their different checksums is displayed in red color.

Figure jarcompare.aspl-img.png

full view

-F- Fig. 1.1.17 [Figure Comparing Two JAR Archives]

ASPL © 2024 by Bassem Jamaleddine

Naturally the ASPL interpreter allows you to do set operation interactively at its prompt and the user can repeat the same operations shown in the script at the ASPL prompt. Writing scripts with ASPL is simple as you will see in Chapter "ASPL Scripts".

ASPL uses the notion of grouping class and treats algebraic groups per their meta data. The grouping class specially characterizes the set of data being treated.

ASPL Elements Class Containment is discussed in Chapter "ASPL A DETAILED VIEW".

ASPL is premier set calculator uniquely identified with its powerful programming of set elements and their attributes: ASPL element attributes can be statically defined, or dynamically updatable by anonymous subroutines, or refreshable with new data when tied to generative devices. When a change is detected in an element, ASPL archives the variable, and you can use set operators to display what has changed.