

SPARCL Tutorial, part 3

(Copyright 2002 by Lindsey Spratt. All Rights Reserved.)

3: Presentation of an application of SPARCL

Introduction

This section presents an application of SPARCL, "Column Sum". This program takes a function term table and a domain value and produces the sum of all of the range values for that domain value in the function term table. This is analogous to finding the sum of the values of a given column of a spreadsheet. This program demonstrates the use of several aspects of SPARCL including term tables, intensional sets, multisets, and arithmetic. We demonstrate the creation of the clause defining the 'Column Sum'/3 predicate and a "query" clause using this predicate and explain various aspects of these clauses and their use.

Create the Column Sum clause.

First we will create the "Column Sum" clause. This one clause is the entire definition of the "Column Sum" predicate. This clause with empty arguments and an empty body is shown in Figure 1.

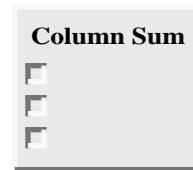


Figure 1: Clause for 'Column Sum'/3 with empty arguments and empty body.

Step 1: Create the "Column Sum" clause.

Step 1.1.

DO: Create a new clause named "Column Sum" with 3 arguments in program "Column Sum"

BY: Select the "Create Clause" option in the popup menu for the program. Add 3 arguments.

Step 2: Create the comments and variables of the arguments of the "Column Sum" clause.

There are three arguments to this clause. Each of these arguments will be given a variable and a comment describing the purpose of the variable. The steps for creating and editing a comment, the sub-steps of step 2.1, are very similar to those for creating and editing an ur constant.

Step 2.1: Create a new comment "Function Table" in the first argument of the clause.

Step 2.1.1.

DO: Create a new comment in the first argument of the clause.

BY: Select the "Create Comment" option in the popup menu for the argument.

Step 2.1.2.

DO: Edit the value of "open" comment "Function Table".

BY: Enter the characters of the new value.

Step 2.1.3.

DO: Close the current edit "box" for program "Column Sum".

BY: Click in the program window anywhere outside of the box.

Step 2.2.

DO: Create a new variable in the second argument of the clause.
BY: Select the "Insert:Variable" option in the popup menu for the argument.

Steps 2.3 through 2.6 repeat the basic process of steps 3.2.1 and 3.2.2 to fill in the second and third arguments of the 'Column Sum'/3 clause. The result is shown in Figure 2.

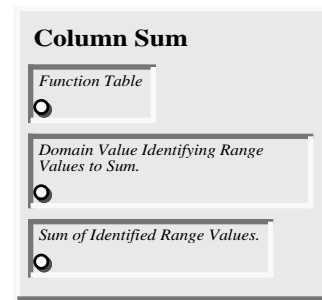


Figure 2: Clause for 'Column Sum'/3 with arguments filled in and an empty body.

Add the 'is'/2 literal to the Column Sum clause.

The next major step, 3, creates the single literal using 'is'/2 that is the body of the clause. This literal does the arithmetic to determine the sum. One of the terms in the second argument of this literal collects together the values which are to be summed.

Step 3: Create the literal of the "Column Sum" clause.

In steps 3.1 and 3.2 we create the literal with empty arguments and fill in the first argument. The first argument is the the result of the arithmetic evaluation of the expression in the second argument.

Step 3.3.1.

DO: Create a new literal with name "is" and 2 arguments in the clause.
BY: Select the "Create Literal" option in the popup menu for the clause.

Step 3.3.2.

DO: Create a new variable in the first argument.
BY: Select the "Insert:Variable" option in the popup menu for the argument.

The next steps build the expression which is the sum of the range values of the specified domain value. The "sum" part of this expression is represented by a '+' and the set of the range values is represented by an intensional (multi)set. The expression is an ordered pair with '+' as its first argument and the intensional multiset as its second argument. To build this N-tuple (2-tuple), we first create what will be the first element of the N-tuple, the '+' ur constant.

Step 3.3: Create a new ur constant of value "+" in the second argument.

The next step achieves two purposes: it replaces the '+' ur constant with an 2-tuple which has that ur constant as its first element, and it creates an empty intensional set as the second element of the new 2-tuple. The interaction for this step is shown in Figure 3. The result is shown in Figure 4.

Step 3.4.

DO: Create an N-tuple with the ur constant "+" as the first element and create a new intensional set term as the second term.
BY: Select the "Create NTuple:IntenSet" option in the popup menu for "+".

Add a multiset intensional set to the 'is/2' literal.

An intensional set, such as shown in Figure 4 as the second argument of a 2-tuple, is a term which specifies the set of all terms which have the given property. The type of the intensional set may be either "set" or "multiset". A multiset intensional set (or "intensional multiset") is the multiset of all terms which have the given property. The representation of an intensional (multi)set has two parts, the template and the body. The body is a set of literals.

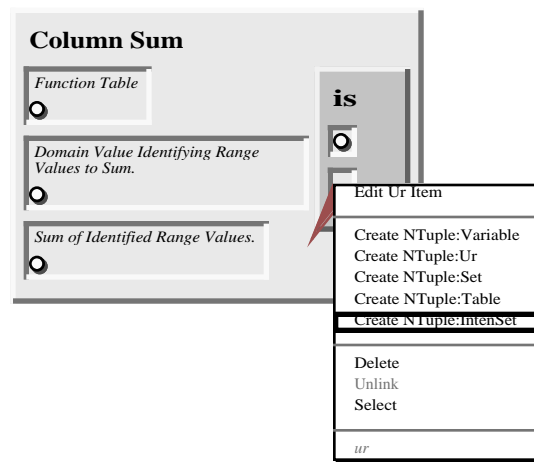


Figure 3: Interaction for step 3.4 to convert the '+' ur constant to a 2-tuple with '+' as first element and an empty IntenSet as the second element.

The template contains a term (which almost always contains at least one variable) which is instantiated once for every "way" in which the literals of the body are true. The "result" of the intensional (multi)set is the (multi)set of all of these instantiations of the template.

We are using the intensional multiset to collect together all of the values of the given domain

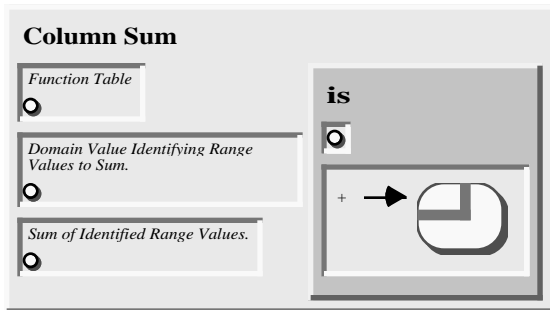


Figure 4: Result of step 3.4.

of the given function table. Since we want to sum all of these values, we want to keep the duplicates. Thus, we want the result type of the intensional set to be "multiset." The next step changes the type appropriately. The interaction is shown in Figure 5. The result is shown in Figure 6.

Step 3.5.

DO: Set the "result type" of the intensional_set to

multiset.
 BY: Select the "Result Type:Multiset" option in the popup menu for the intensional_set.

The template we want is simply a single variable. This will be connected to the range term of each function pair with the given domain term in the given table.

Step 3.6.

DO: Create a new variable in the intensional_set_template.

BY: Select the "Insert:Variable" option in the popup menu for the intensional_set_template.

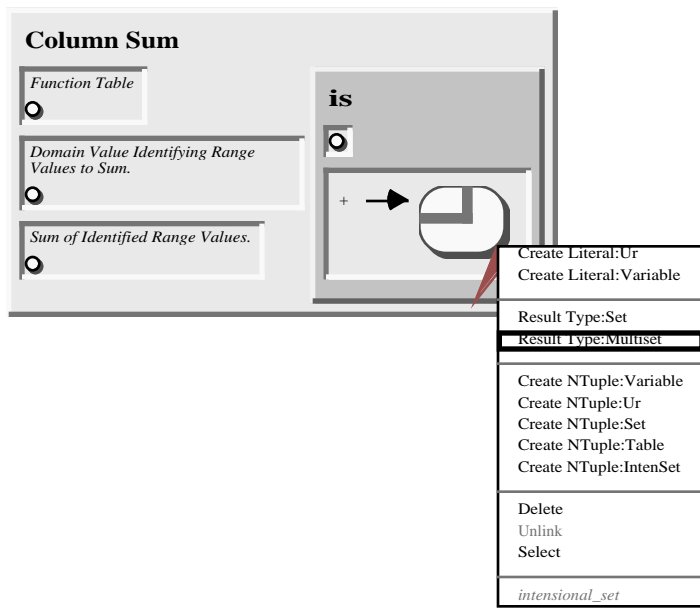


Figure 5: Interaction for step 3.5 to convert an intensional set to an intensional multiset.

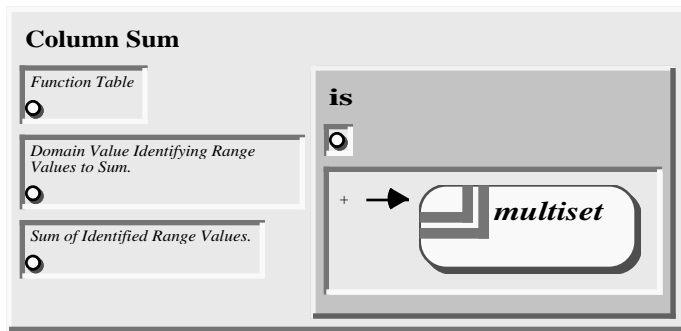


Figure 6: Result of step 3.5 converting an intensional set to an intensional multiset.

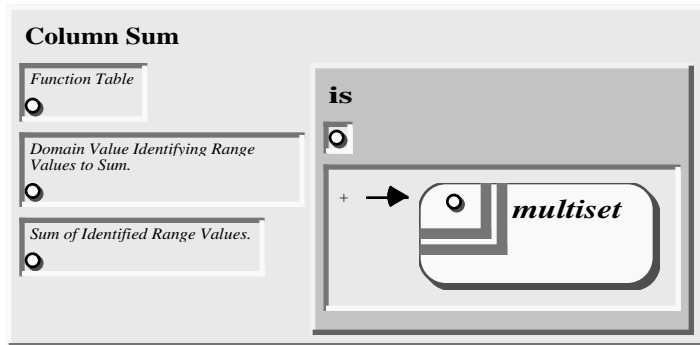


Figure 7: Result of step 3.6 to create a variable in the template of the intensional multiset.

Add unify literal to the intensional set.

Next we create a "unify" literal, which will give us all possible unifications of a function pair with the given function table. The steps for editing the new literal ur in an intensional are similar to those for editing a new ur constant. The interaction for creating the new literal ur is shown in Figure 8. The result of creating the literal ur and editing the literal name to be "unify" is shown in Figure 9.

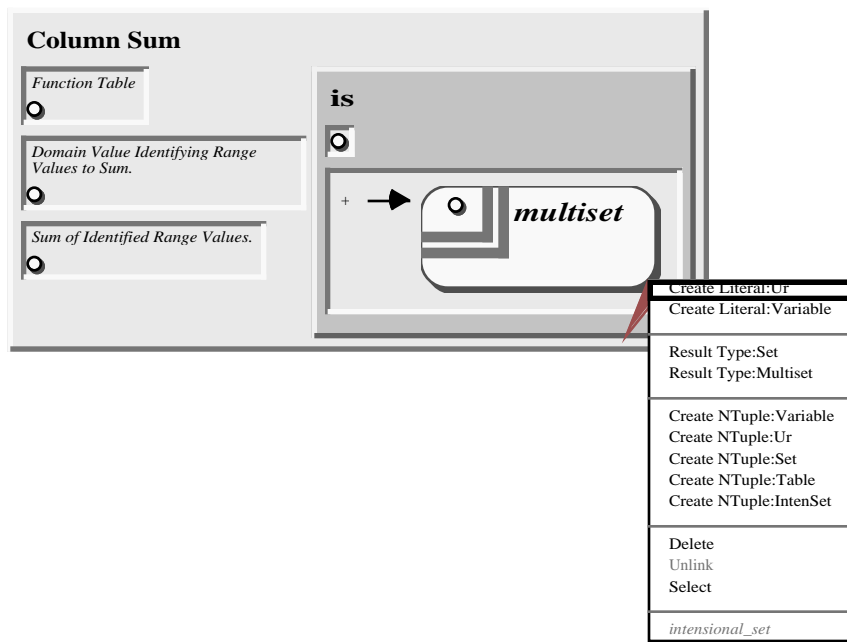


Figure 8: Interaction for creating a new literal in an intensional (multi)set.

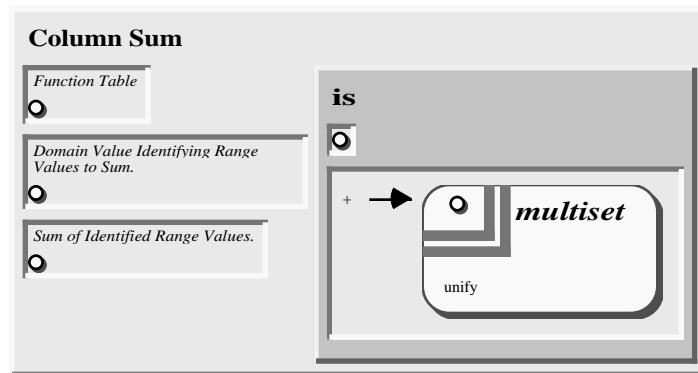


Figure 9: Result of creating new literal "unify" ur.

Step 3.7.

DO: Create a new literal with an ur constant predicate name in the intensional_set.

BY: Select the "Create Literal:Ur" option in the popup menu for the intensional_set.

Step 3.8.

DO: Edit the value of the "open" ur constant to "unify".

BY: Enter the characters of the new value.

Step 3.9.

DO: Close the current edit "box" for program "Column Sum temp".

BY: Click in the program window anywhere outside of the box.

At this point, we have a 'unify'/0 literal. To make a 'unify'/2 literal, we need two arguments added to 'unify'/0. This is represented by an N-tuple of three elements, the first element is the "unify" constant and the second and third elements are the two arguments. The intensional set is a shorthand for the *metapredicate* 'setof'/3. A metapredicate is a predicate that takes terms that are interpreted as literals in one or more of its arguments. A term to be interpreted as a literal is a constant or an N-tuple. A constant is interpreted as a literal of no arguments, a N-tuple is interpreted as a literal of (N-1) arguments. In the next step we replace the "unify" ur constant with an

N-tuple of two elements: the “unify” ur constant and a variable. The interaction is shown in Figure 10. The result is shown in Figure 11.

- Step 3.10.
 DO: Create an N-tuple with ur constant “unify” as the first element and create a new variable term as the second term. (Call the new N-tuple object “N-tuple 2”, call the new variable object “variable 3.”)
 BY: Select the “Create NTuple:Variable” option in the popup menu for ur constant “unify”.

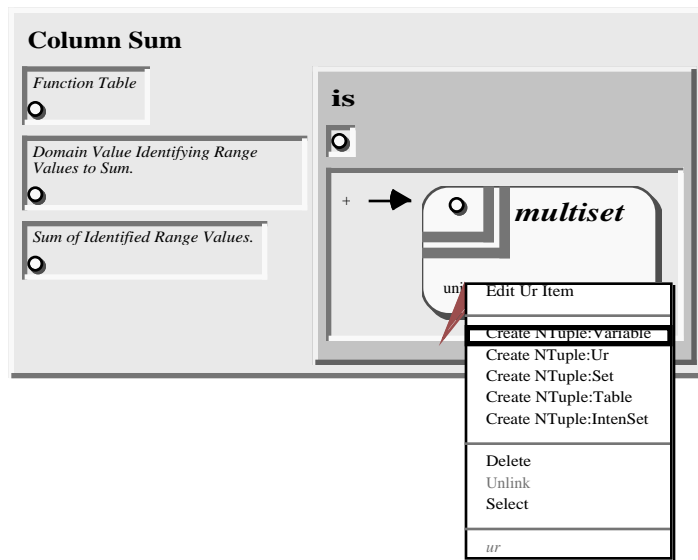


Figure 10: Interaction to create an ordered pair (2-tuple) with “unify” as first element and a variable as the second element.

Now we have an 2-tuple of "unify" and a variable. To complete the "unify" literal, we extend this 2-tuple with another element, an empty set. The interaction is shown in Figure 12 and the result is shown in Figure 13.

- Step 3.11.
 DO: Extend N-tuple 2, creating a new set term as the new last element. (Call the new object “set 1”.)
 BY: Select the "Extend With:Set" option in the popup menu for the N-tuple.

Eventually, we will connect the variable that is the first argument to "unify" with the given function table. The second argument to "unify", which is currently an empty set, we are going to turn into a pattern term which can match (or unify) with any function pair (domain value/range value) in a function table. Since this unify literal is in the body of an intensional set, evaluation

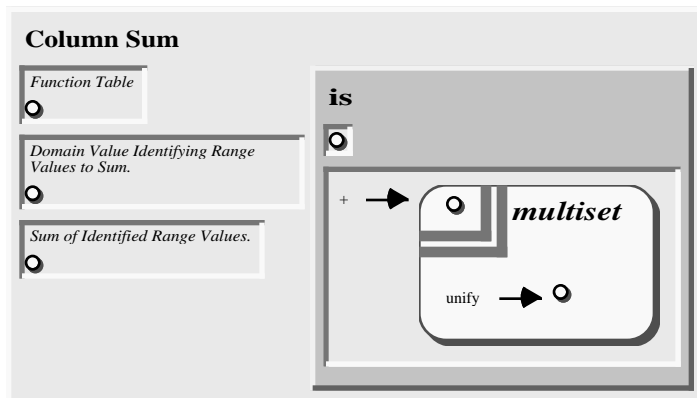


Figure 11: Result of interaction to create an ordered pair with “unify” as the first element and a variable as the second element.

of the intensional set will produce all possible unifications of the pattern term with the given table.

The pattern term is an ordered-pair (2-tuple) within a part of a two-part partitioning of a set (the "inner" set), which is in turn within a part of a two-part partitioning of a set (the "outer" set). The ordered pair will unify with any ordered pair in a function (which is a

set of ordered pairs where no two first elements are the same). The two parts of the "inner" will unify with any table where one part contains one ordered pair and the other part contains any number of terms (and is possibly empty). This partitioned "inner" set will unify with any function (or row) of the given table. The two parts of the "outer" set will unify with any table where one part contains the "inner" set (a row of the table) and the other part contains any number of terms (and is possibly empty).

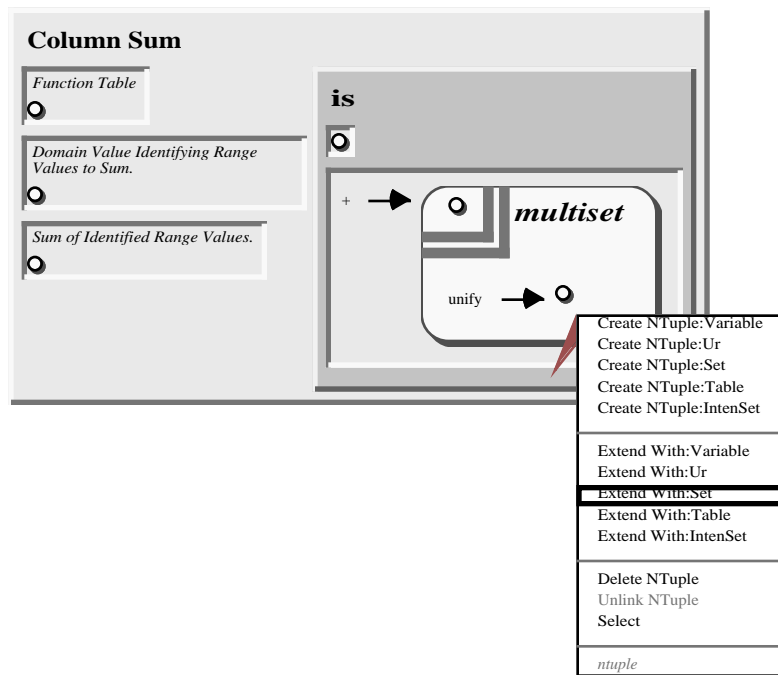


Figure 12: Interaction to extend an ordered pair (2-tuple) to a 3-tuple with an empty set as the third element.

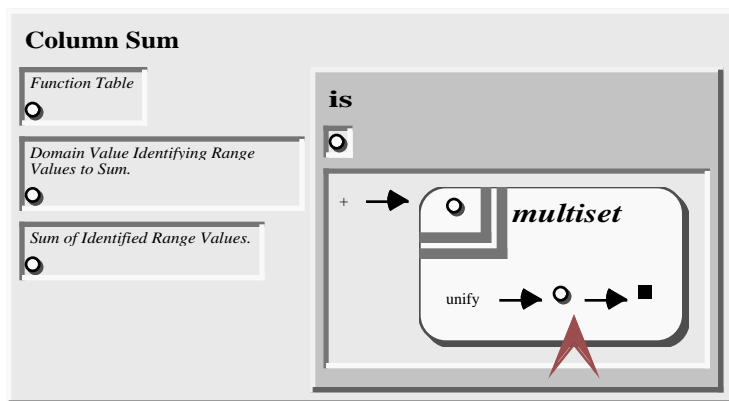


Figure 13: Result of interaction to extend an ordered pair (2-tuple) to a 3-tuple with an empty set as the third element.

Add a set as the second arg of the unify literal.

Our first step in creating this pattern term is to create an empty set within the existing empty set. The existing empty set will be converted to a partitioned set of one part. This is the "outer" set. The newly created empty set will become the "inner" set. The interaction is shown in Figure 14 and the result is shown in Figure 15.

- Step 3.12.
 DO: Create a new set in set 1. (Call the new object "set 2".)
 BY: Select the "Insert:Set" option in the popup menu for the set.

Next we add a "hollow" second part to the "outer" set partitioning. The interaction is shown in Figure 16, and the result is shown in Figure 17.

- Step 3.13.
 DO: Create a new partitioned set part in the set 1.
 BY: Select the "Create Partition" option in the popup menu for set 1.

Now we build the ordered pair inside of the "inner" set. First, we create a variable inside the "inner" set. This will convert the "inner" set from an empty set to a partitioned set of one part,

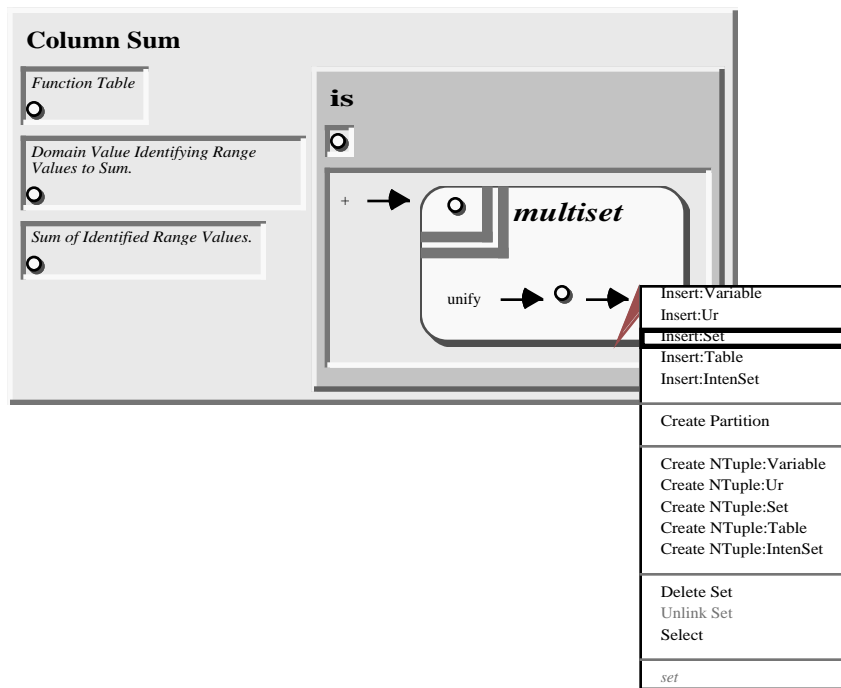


Figure 14: Interaction to convert an empty set to a partitioned set of one part and create an empty set inside that new part.

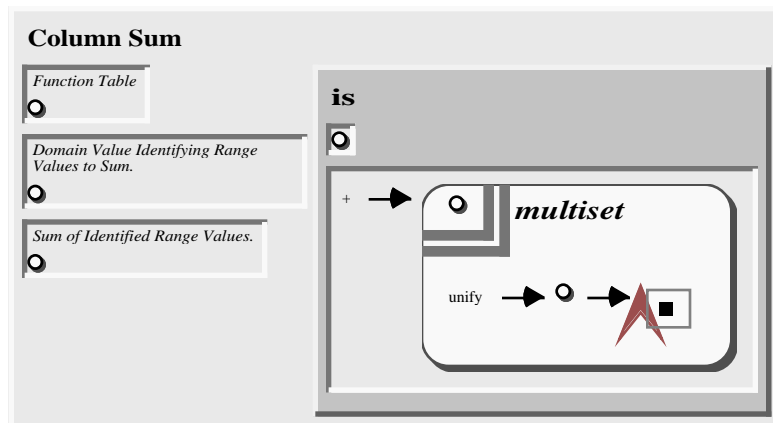
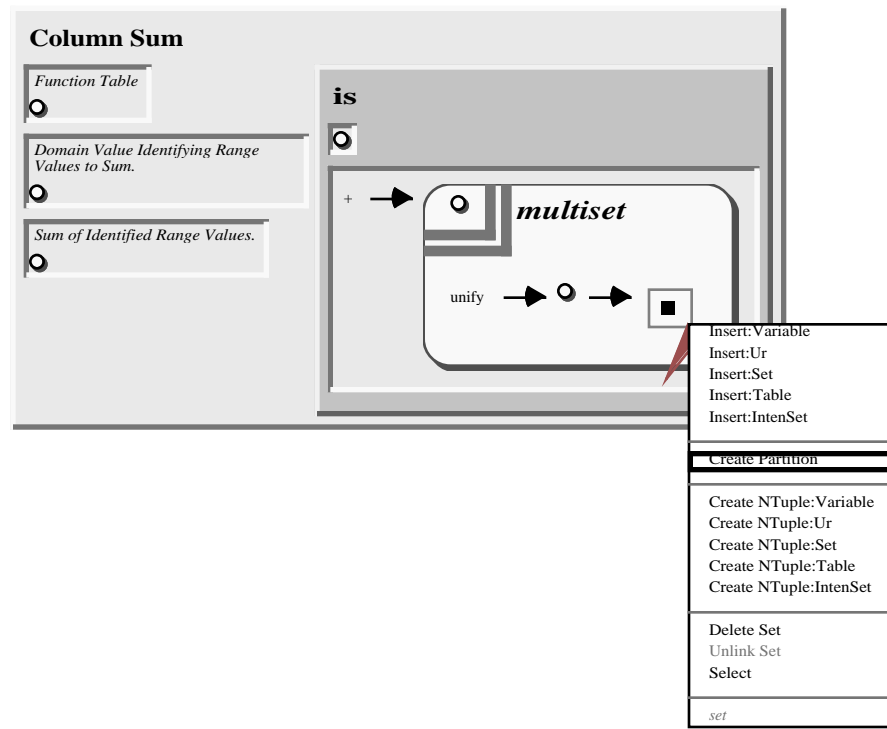


Figure 15: Result of interaction to convert an empty set to a partitioned set of one part and create an empty set inside that new part.

and place a variable inside this one part. The interaction is shown in Figure 18 and the result is shown in Figure 19.

- Step 3.14.
 DO: Create a new variable in set 1. (Call this object "variable 4".)
 BY: Select the "Insert:Variable" option in the popup menu for the set.



This variable is the generic domain value variable. We replace this variable by an N-tuple with this variable as its first element and another variable as its second element. This is shown in Figure 20 and Figure 21.

- Step 3.3.15.
 DO: Create an N-tuple with variable 4 as the first element and create a new variable term as the second term. (Call this new N-tuple object "N-tuple 3" and the new variable object "variable 5".)
 BY: Select the "Create NTuple:Variable" option in the popup menu for the variable.

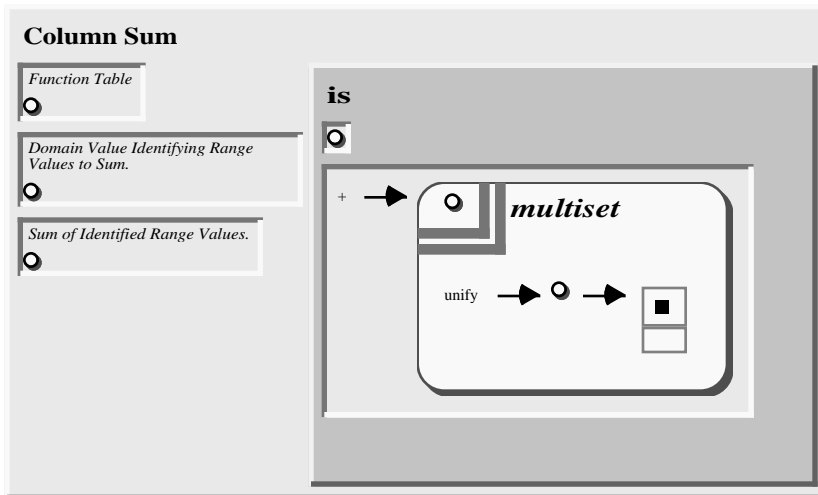


Figure 17: Result of interaction to create a "hollow" part.

The new variable in Figure 21 is the generic range value variable. This completes the function pair N-tuple. To finish this "pattern" term, we will create a second "hollow" part in the "inner" set. The interaction and result are shown in Figure 22 and Figure 23.

Step 3.16.
 DO: Create a new partitioned set part in the set 2.
 BY: Select the "Create Partition" option in the popup menu for the set.

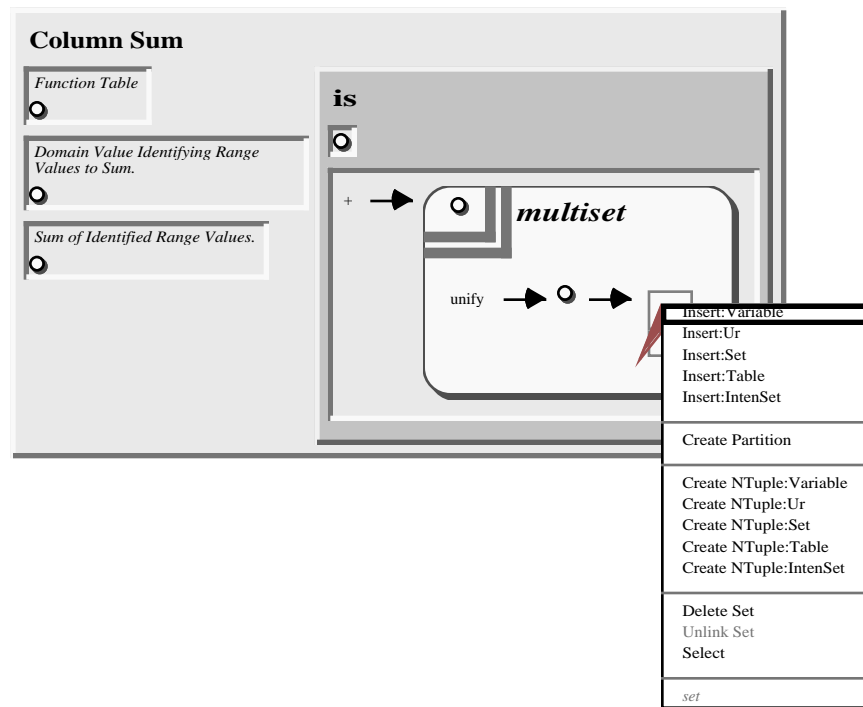


Figure 18: Interaction to create a variable.

We complete the intensional set by connecting the template variable to the generic range value variable. This is shown in Figure 24 and Figure 25.

Step 3.17.
 DO: Create a coreference link including the intensional set template variable and variable 5.
 BY: With "connect tool" as the current tool, depress the mouse button while the cursor is over one of the variables and drag the cursor until it's over the other variable, then release the mouse button.

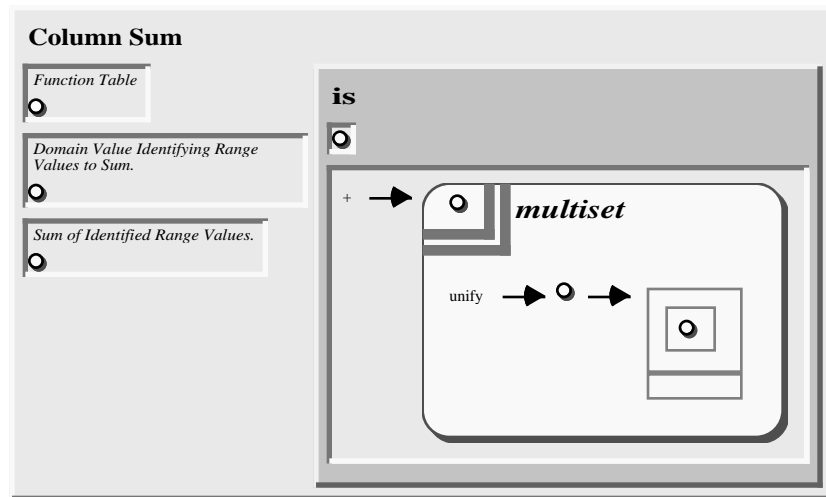


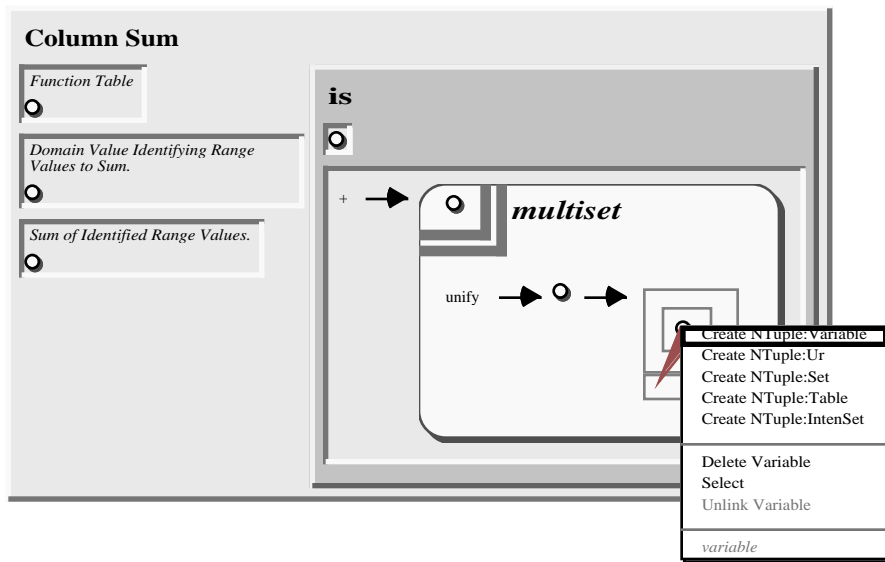
Figure 19: Result of creation of a variable.

Add connections.

To complete the "Column Sum" clause, we need only to connect some terms. We connect the "given table" argument variable to the intensional set unification table variable. This specifies that the intensional set "ranges" over the given table. This is shown in Figure 25 and Figure 26.

Step 3.18.
 DO: Create a coreference link including the variable in the first argument of the clause and variable 3.
 BY: With "connect tool" as the current tool, depress the mouse button while the cursor is over one of the

variables and drag the cursor until it's over the other variable, then release the mouse button.



We connect the "given domain value" argument variable to the first element of the ordered pair in the intensional set unification. This specifies that only

Figure 20: Interaction to convert a variable to a 2-tuple of two variables.

range values paired with a domain value the same as the "given" domain value will be collected in the intensional set result.

Step 3.19.

DO: Create a coreference link including the variable in the second argument of the clause and variable 4.

BY: With "connect tool" as the current tool, depress the mouse button while the cursor is over one of the variables and drag the cursor until it's over the other variable, then release the mouse button.

The last connection makes the sum of the intensional multiset the result of the clause.

Step 3.20

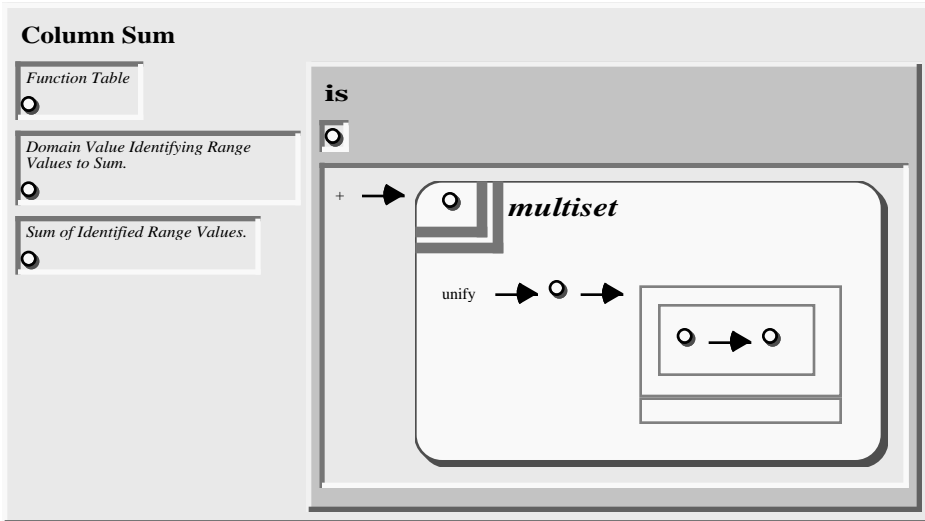


Figure 21: Result of interaction to convert a variable to a 2-tuple of two variables.

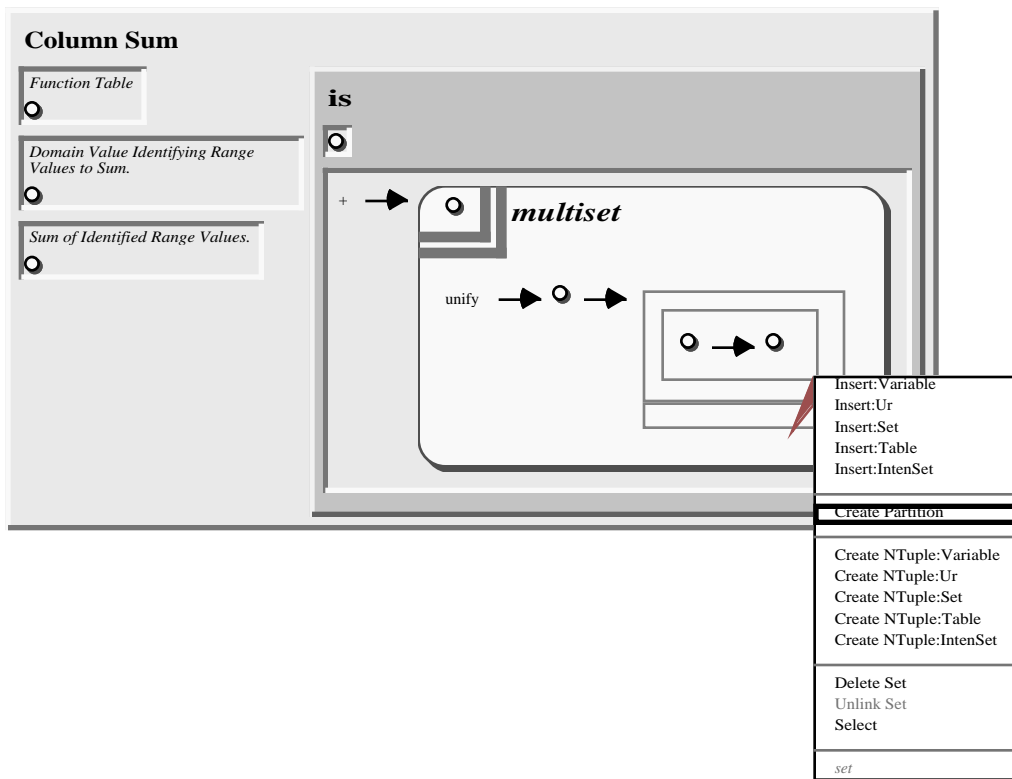


Figure 22: Interaction to create a “hollow” part in a partitioned set.

- DO: Create a coreference link including the variable in the third argument of the clause and the variable in the first argument of the literal.
- BY: With "connect tool" as the current tool, depress the mouse button while the cursor is over one of the variables and drag the cursor until it's over the other variable, then release the mouse button.

Having finished creating the clause which defines the ‘Column Sum’/3 predicate as shown in

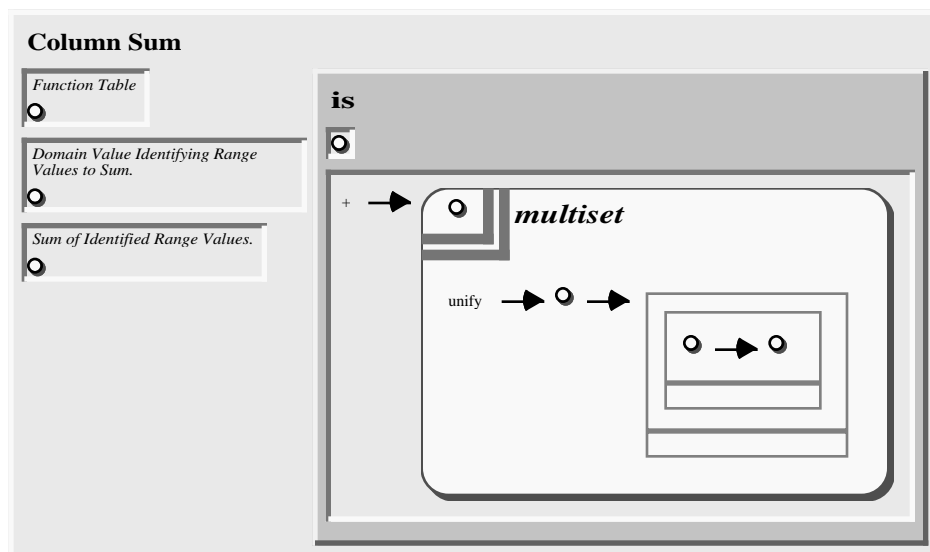


Figure 23: Result of creating “hollow” part in partitioned set.

Figure 27, we will create a clause defining the predicate 'Column Sum Query'/1 for querying this predicate. This clause will provide test data to 'Column Sum'/3 and return the calculated sum.

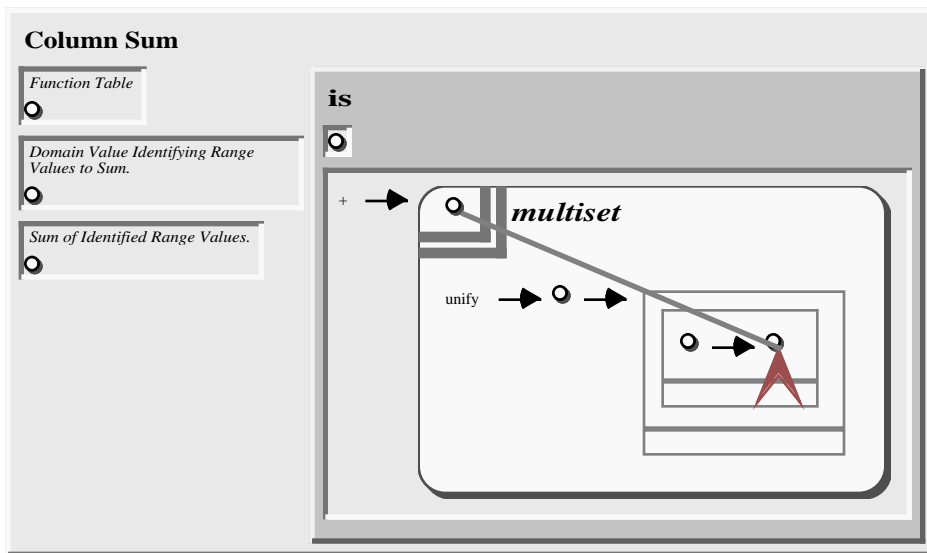


Figure 24: Interaction (in "connect" mode) to create a link between the template variable and the range variable of the "pattern".

Next section: Create the 'Column Sum Query'/1 predicate.

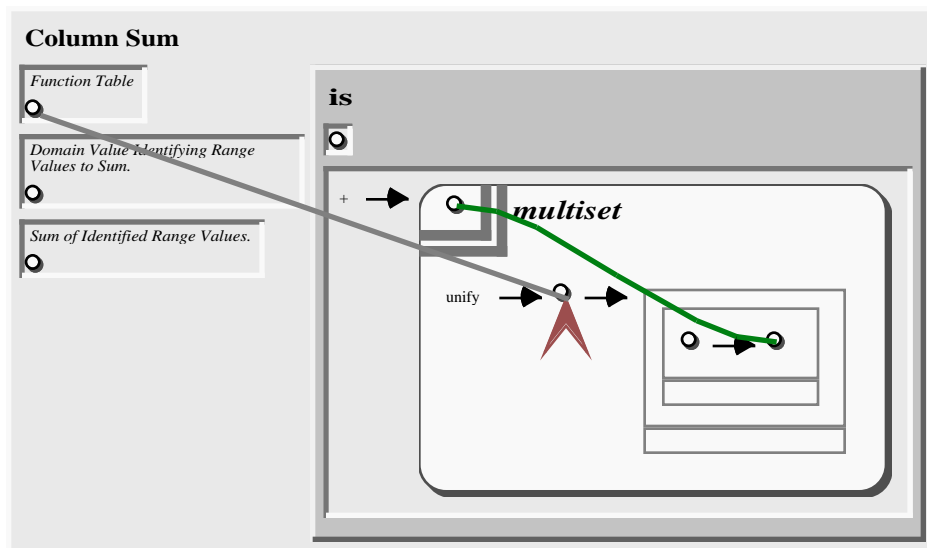


Figure 25: Result of linking the template and pattern "range value" variables. Interaction to create a link between the "function table" argument variable and the first argument variable of the intensional set 'unify'/2 literal 3-tuple.



Figure 26: Result of creating the "function table" link.

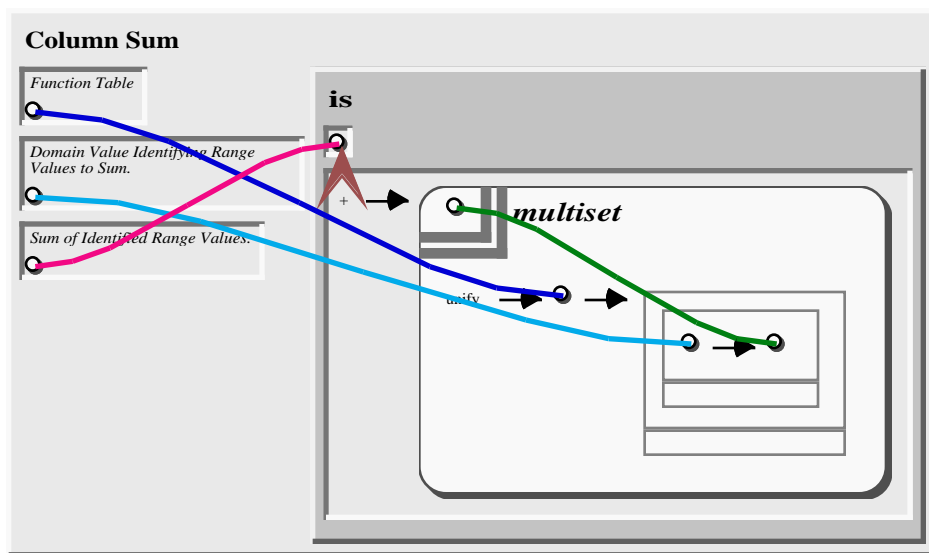


Figure 27: Complete clause defining the 'Column Sum'/3 predicate.