AIMMS Language Reference - Index Binding

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

Email: info@aimms.com
WWW: www.aimms.com

Chapter 9

Index Binding

This chapter

This chapter presents the *index binding* rules implemented in AIMMS. These rules play an essential role during most repetitive set operations. For standard situations AIMMS behaves as expected. You should read this chapter if you are interested in a formal discussion of the rules of the underlying semantics.

9.1 Binding rules

Repetitive operations

During execution, indices are used to traverse a set to repeatedly apply a specific operation on all elements of a set. These operations concern

- indexed assignment statements,
- FOR statements,
- iterative operations like summation over a domain,
- constraint generation,
- arc generation, and
- constructed set expression.

Index binding

Index binding is the process by which AIMMS repeatedly couples the value of an index to elements of a specific set to execute repetitive operations.

Different types of binding

There are three ways in which index binding takes place:

- *local* binding,
- *default* binding, and
- *context* binding.

Local binding

Local binding takes place through the use of an IN modifier at the index binding position as illustrated in the following example.

```
NettoTransport(i in SupplyCities, j in DestinationCitiesFromSupply(i)) :=
    Transport(i,j) - Transport(j,i);
```

Instead of executing the assignment for all cities i and j, it is only executed for those combinations for which city i is in SupplyCities and city j is in DestinationCitiesFromSupply(i).

Default binding

Indices can have a *default binding*. This is the binding specified in a declaration. You can specify a default binding either via the Index attribute of a set, or via the Range attribute of an Index declaration. Whenever you use an index with a default binding and do not specify a local binding, AIMMS will couple this index to its default set automatically. The following example illustrates default binding.

```
IntermediateTransportCitiesInBetween(i,j) :=
       DestinationCitiesFromSupply(i) * SupplyCitiesToDestination(j);
```

Assuming that i and j have a default binding to the set Cities, the assignment takes place for all tuples of cities (i,j).

Context binding

Whenever you use an index that has no default binding and for which you do not provide a local binding, AIMMS will try to determine a *context binding* from the context. Assume that k is an index without a default binding. Further assume that LargestTransport is an element parameter into Cities and indexed over Cities. Then the following example is an illustration of context binding.

```
LargestTransport(k) := ArgMax( j, Transport(k,j) );
```

In this assignment AIMMS will automatically bind the index k to Cities, because the identifier LargestTransport has been declared with the index domain Cities. Note that context binding will only work in indexed assignments.

Nested index binding

Index binding can be nested through the use of indexed element-valued parameters on the left-hand side of an assignment. The binding takes place in the way that you would expect, applying the same rules as for non-nested index binding. For example, given the declarations

```
ElementParameter NextCity {
   IndexDomain  : i;
   Range        : Cities;
}
ElementParameter PreviousCity {
   IndexDomain  : i;
   Range        : Cities;
}
```

the following assignment, which computes the value of PreviousCity given the contents of NextCity, will bind the nested reference to the index i.

```
   PreviousCity( NextCity(i) ) := i;
```

This binding is sparse, in the sense that the statement is only executed for those i for which NextCity(i) assumes a nonempty value.

Compatible index binding only

In general, AIMMS will never accept the use of an index in references to indexed identifiers when the binding set does not have the same root set as the index domain of the identifier. This is even the case when the elements, referenced in the particular statement, have identical names in both the binding set and the index domain. Internally, AIMMS stores a set elements as a unique (integer) number with respect to its root set, and uses this number for storing data for that element in indexed identifiers. Thus, when the root sets of the binding set and the index domain are not identical, the set element numbers will be incompatible, preventing AIMMS from referencing the correct data.

Use indirect referencing

When you want to use a binding set which is incompatible with the index domain of identifier on the left-hand side of an assignment, you should manually create an element parameter which maps elements in one root to the corresponding elements the other root set. Such a mapping can be easily created using the function ElementCast (discussed in Section 5.2.2), as exemplified below.

```
ElementMap(i) := ElementCast( IncompatibleRootSet, i );
```

Subsequently, you can use a nested binding through the element parameter ElementMap to reference elements in the index domain of the identifier on the left-hand side of an assignment, while still using the index i as a binding index, as illustrated in the following statement.

```
IncompatibleParameter( ElementMap(i) ) := CompatibleParameter(i);
```

Use the ElementCast function

Conversely, when you want to use an incompatible set element in a parameter reference on the right-hand side of an assignment, there is no direct need to create a mapping parameter. In an expression on the right of an assignment, you can use the function ElementCast directly at any index position, as illustrated below.

```
CompatibleParameter(i) := IncompatibleParameter( ElementCast(IncompatibleRootSet, i) );
```

Universal set

Note that you could have accomplished the same effect by creating a universal set of which all other sets are subsets. As a result, all set elements are represented as unique integer numbers with respect to the same root set, allowing the index domains of all identifiers to be referenced in a compatible manner. However, often it is not very natural to do so, and the usage of a universal set is likely to slow down the performance of AIMMS.

Index binding rules

For most situations the result of index binding is self-evident and the behavior of the system is as you would expect. Following are the precise rules for index binding.

- **Dominance rule:** Whenever index binding takes place, local binding precedes default binding, which in turn precedes context binding. If no method is applicable, a compile time error will result.
- **Intersection rule:** In indexed assignments the binding set(s) should be compatible with the index domain. The assignment will be performed for all tuples on the left-hand side that lie in the intersection of the binding set(s) and the index domain of the corresponding identifier.
- **Ordering rule:** Lag and lead operators, as well as the `Ord` and `Element` functions operate according to the order of elements in the corresponding binding set.