Software architecture reconstruction
Krikhaar, R.

Citation for published version (APA):
Krikhaar, R. (1999). Software architecture reconstruction

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 3

Relation Partition Algebra

In the previous chapter we presented in general terms the SAR method. The underlying model of the SAR method consists of Relation Partition Algebra which is introduced in this chapter. In the succeeding chapters we heavily use Relation Partition Algebra to describe the details of the SAR method.

3.1 Introduction

In this chapter we introduce Relation Partition Algebra (RPA). RPA is based on sets and binary relations. This chapter serves as a brief introduction to RPA, and it is also meant to introduce notations that will be used throughout the thesis.

RPA has been defined in order to be able to formalise descriptions of (parts of) software architectures. Furthermore, in the context of reverse engineering one often wants to query the software structure. RPA offers abilities to express questions in a formal notation, which can be executed on the actual (model of the) software. Throughout this thesis, we will see many applications of RPA for reconstructing software architectures or beautifying presentations of architectural information.

We will start this chapter by discussing sets and operations on sets. In Section 3.3 binary relations and operations upon them will be presented. The proofs of algebraic laws relating to RPA will not be given here, but we will refer to published work [SM77, FKO98, FK99, FO99]. In Section 3.5 we will extend RPA with multi-sets and multi-relations. RPA formulas can
also be executed; related issues will be discussed in Section 3.6.

3.2 Sets

3.2.1 Primitives of Set Theory

A set is a collection of objects, called elements or members. If \( x \) is an element of \( S \), given any object \( x \) and set \( S \), we write \( x \in S \). The notion of set and the relation is-element-of are the primitive concepts of set theory. We rely on a common understanding of the meaning of these terms.

A finite set can be specified explicitly by enumerating its elements. The elements are separated by commas, and the enumeration is enclosed within brackets. So, the set which contains elements \( a \), \( b \), and \( c \) is denoted by \( \{a, b, c\} \). Infinite sets cannot be listed explicitly, so these sets are described implicitly. A set can be described using a predicate with a free variable. The set \( \{x \in U | P(x)\} \), for given \( U \) (another set playing the role of universe), denotes the set \( S \) such that \( x \in S \) if and only if \( x \in U \) and \( P(x) \) holds.

We will use the logical operators \( \lor \) and \( \land \) to denote the logical (inclusive) or and the logical and, respectively. \( a \lor b \) holds if and only if \( a \) is true or \( b \) is true or both \( a \) and \( b \) are true. \( a \land b \) holds if and only if \( a \) and \( b \) are true. Furthermore, \( a \Rightarrow b \) holds if \( a \) is true then \( b \) is true. \( a \iff b \) holds if and only if \( a \Rightarrow b \land b \Rightarrow a \).

At the end of each section we will illustrate the discussed operators with a running example.

example

\[
\begin{align*}
\text{Subsystems} & = \{\text{OS, Drivers, DB, App}\} \\
\text{Functions} & = \{\text{main,}a, c, d\} \\
\text{InitFunctions} & = \{f | f \in \text{Functions} \land f \text{ is called at initialisation time}\}
\end{align*}
\]
3.2.2 Operations on Sets

**equal, subset, superset, size**

Two sets \( S_1 \) and \( S_2 \) are *equal*, denoted by \( S_1 = S_2 \), if for each \( x \) it holds that \( x \in S_1 \iff x \in S_2 \). A set \( S_1 \) is contained in \( S_2 \), or \( S_1 \) is a *subset* of \( S_2 \) denoted by \( S_1 \subseteq S_2 \), if for each \( x \) it holds that \( x \in S_1 \Rightarrow x \in S_2 \). A similar definition holds for a *superset*, \( S_1 \supseteq S_2 \), which is an alternative notation for \( S_2 \subseteq S_1 \). A *strict* subset (superset) is a subset (superset) from which equality is excluded. It is denoted by \( \subset \) respectively \( \supset \). The number of elements in a finite set is called the *size*, denoted by \(|S|\).

**union, intersection**

The *union* of two sets \( S_1 \) and \( S_2 \), denoted by \( S_1 \cup S_2 \), is the set \( T = \{x|x \in S_1 \lor x \in S_2\} \). The *intersection* of two sets \( S_1 \) and \( S_2 \), denoted by \( S_1 \cap S_2 \), is the set \( T = \{x|x \in S_1 \land x \in S_2\} \).

**difference, complement**

The *difference* of two sets \( S_1 \) and \( S_2 \), denoted by \( S_1 \setminus S_2 \), is the set \( T = \{x|x \in S_1 \land x \notin S_2\} \). It is also called the *relative complement* of \( S_2 \) with respect to \( S_1 \). The complement of a set \( S \), denoted by \( \overline{S} \), is the set \( T = \{x|x \notin S\} \). Given that \( U \) is the universe, containing all elements, the complement of a set \( S \) can be written as: \( \overline{S} = U \setminus S \).

**example**

\[
\begin{align*}
\text{Subsystems} &= \{\text{OS, Drivers, DB, App}\} \\
\text{InitFunctions} &\subseteq \text{Functions} \\
\text{UpperLayers} &= \{\text{DB, App}\} \\
\underline{\text{UpperLayers}} &= \{\text{OS, Drivers}\} \text{ (with respect to Subsystems)}
\end{align*}
\]
3.3 Binary Relations

3.3.1 Primitives of Binary Relations

Besides the notion of sets, we need more to describe software structures. Relationships between (software) entities play an important role in architecture and design. Binary relations can express such relationships. For example, function-calls within a system can be seen as the binary relation named calls.

A binary relation, or shortly a relation, from $X$ to $Y$ is a subset of the cartesian product $X \times Y$. It is a set of tuples $\langle x, y \rangle$ where $x \in X$ and $y \in Y$. Tuples of a binary relation $R$ can be denoted in different ways. The following notations are used to refer to an element of a binary relation:

- infix notation: $xRy$
- prefix notation: $R(x, y)$
- tuple notation: $\langle x, y \rangle$

In relational terms calls($main, a$) is an abstraction of the following program fragment (written in the programming language C [KR88]):

```c
void main () {
    ....
    a(12, i, &ref);
    ....
}
```

Besides a textual representation of relations, one can also represent a relation in a directed graph. A directed graph, or shortly digraph, consists of a set of elements, called vertices, and a set of ordered pairs of these elements, called arcs [WW90]. Assume a digraph $G$ represents the relation $R \subseteq X \times Y$. The arcs of $G$ represent the tuples of $R$; the vertices represent elements of $X \cup Y$. The vertices with outgoing arcs are elements of $X$ and vertices with incoming arcs are elements of $Y$. The calls relation of a (fictive) program is shown in Figure 3.1.

example

$$\text{calls} = \{ \langle \text{main}, a \rangle, \langle \text{main}, b \rangle, \langle a, b \rangle, \langle a, c \rangle, \langle a, d \rangle, \langle b, d \rangle \}$$
3.3 Binary Relations

3.3.2 Operations on Binary Relations

Relations are sets (of tuples), so we inherit the definitions of equality, containment, size, union, intersection, difference and complement from the previous section.

converse

The converse of relation $R$, denoted by $R^{-1}$, is obtained by reversing the tuples of $R$: $R^{-1} = \{ (y, x) | (x, y) \in R \}$.

product, identity

The cartesian product of two sets $X$ and $Y$, denoted by $X \times Y$, is the relation $R = \{ (x, y) | x \in X \land y \in Y \}$. A special relation $Id_X$, or just $Id$ if the set $X$ is obvious, is called the identity relation. It is defined as $Id_X = \{ (x, x) | x \in X \}$.

domain, range, carrier

The domain of a relation $R$, denoted by $dom(R)$, is the set $S = \{ x | (x, y) \in R \}$. The range of relation $R$, denoted by $ran(R)$, is the set $S = \{ y | (x, y) \in R \}$. The carrier of a relation $R$, denoted by $car(R)$, is defined as $dom(R) \cup ran(R)$.
restriction

The domain restrict of a relation $R$ with respect to a set $S$, denoted by $R |_{\text{dom } S}$, is a relation $T = \{ (x,y) \mid (x,y) \in R \land x \in S \}$. The range restrict of a relation $R$ with respect to a set $S$, denoted by $R |_{\text{ran } S}$, is a relation $T = \{ (x,y) \mid (x,y) \in R \land y \in S \}$. The carrier restrict of a relation $R$ with respect to a set $S$, denoted by $R |_{\text{car } S}$, is a relation $T = \{ (x,y) \mid (x,y) \in R \land x \in S \land y \in S \}$. The carrier restrict can also be defined as: $R |_{\text{car } S} = (R |_{\text{dom } S}) |_{\text{ran } S}$.

exclusion

A variant of restriction is exclusion. The domain exclude of a relation $R$ with respect to a set $S$, denoted by $R \setminus_{\text{dom } S}$, is a relation $T = \{ (x,y) \mid (x,y) \in R \land x \not\in S \}$. The range exclude of a relation $R$, denoted by $R \setminus_{\text{ran } S}$, is a relation $T = \{ (x,y) \mid (x,y) \in R \land y \not\in S \}$. The carrier exclude of a relation $R$, denoted by $R \setminus_{\text{car } S}$, is a relation $T = \{ (x,y) \mid (x,y) \in R \land x \not\in S \land y \not\in S \}$. The carrier exclude can also be defined as $R \setminus_{\text{car } S} = (R \setminus_{\text{dom } S}) \setminus_{\text{ran } S}$.

top, bottom

The top of a relation $R$, denoted by $\top(R)$, is defined as $\text{dom}(R) \setminus \text{ran}(R)$. Given a directed graph of relation $R$, the top consists of vertices that are a root. A root is a vertex that has no incoming arcs. Similarly, the bottom of a relation $R$, denoted by $\bot(R)$, is defined as $\text{ran}(R) \setminus \text{dom}(R)$. They are the leaf vertices of a directed graph, which are the vertices with no outgoing arcs.

projection

The forward projection of set $S$ in relation $R$, denoted by $S \triangleright R$, is the set $T = \{ y \mid (x,y) \in R \land x \in S \}$. The backward projection of $S$ in $R$, denoted by $R \triangleleft S$, is the set $T = \{ x \mid (x,y) \in R \land y \in S \}$. Forward projection can also be defined as $S \triangleright R = \text{ran}(R |_{\text{dom } S})$ and the backward projection can be defined as $R \triangleleft S = \text{dom}(R |_{\text{ran } S})$.

The left image of a relation $R$ with respect to element $y$, denoted by $R.y$, is the set $T = \{ x \mid (x,y) \in R \}$. The right image of a relation $R$ with respect to element $x$, denoted by $x.R$, is the set $T = \{ y \mid (x,y) \in R \}$. 
composition

The composition of two relations $R_1$ and $R_2$, denoted by $R_2 \circ R_1$, is the relation $R = \{(a, b) | \exists x . \ (a, x) \in R_1 \land (x, b) \in R_2\}$. $R_1; R_2$ is an alternative notation of the composition $R_2 \circ R_1$. One should pronounce $R_2 \circ R_1$ as “apply $R_2$ after $R_1$”.

Composing a relation $n$ times, $R \circ R \circ \ldots \circ R$ is denoted by $R^n$. Note that composition is associative (proof is given in [FO99]), so we may omit parentheses around each composition. Furthermore, by definition $R^0 = Id$.

transitive closure

The transitive closure of a relation $R$, denoted by $R^+$, is the relation $T = \bigcup_{i=1}^{\infty} R^i$, i.e. the union of all $R^i$. The reflexive transitive closure $R^*$ is $R^0 \cup R^+ = Id \cup R^+$.

Special algorithms have been developed to calculate the transitive closure efficiently. In 1962 Warshall [War62] described an $O(n^3)$ algorithm (where $n$ is the size of the carrier of the relation):

```plaintext
for i in S do
  for j in S do
    for k in S do
      T[j,k] = T[j,k] + T[j,i] \times T[i,k]
```

explanation

The array $T$ represents the existence (boolean value) of tuples $(i, j)$ in the given relation. The set $S$ equals the carrier of this relation. Each of the for-loops enumerates the elements in the set. The $+$ operation is defined as the logical or operation and the $\times$ operation is defined as the logical and. One should read the last statement as follows (having a digraph representation in mind): if there is a path from $j$ to $i$ and there is a path from $i$ to $k$, then there exists a path from $j$ to $k$.

As an example we present the transitive closure of the `calls` relation in Figure 3.2.

reduction

A relation $R$ is cycle-free if and only if $Id \cap R^+ = \emptyset$, in other words, in a graph representation of relation $R$, there is no path from any vertex to
The transitive reduction of a cycle-free relation $R$, denoted by $R^-$, is a relation containing all tuples of $R$ except for short-cuts. For example, the tuple $(x, z)$ is a short-cut if $R$ contains the tuples $(x, y)$ and $(y, z)$. The transitive reduction of $R$ is also called the Hasse [SM77] of $R$, or the poset of $R$. The transitive reduction of a cyclic-free relation $R$ can also be defined as $R^- = R \setminus (R \circ R^+)$. 

The expression $R \circ R^+$ represents all the pairs of elements in $R$ that can reach each other indirectly (so via another vertex in the digraph). When we subtract these tuples from the original relation $R$ we retain the tuples which are not a shortcut. The Hasse of the \textit{calls} relation is illustrated in Figure 3.3.

example

\[
\begin{align*}
\text{maincalls} &= \{\langle \text{main}, a \rangle, \langle \text{main}, b \rangle\} \\
\text{maincalls} &\subseteq \text{calls} \\
calls \setminus \text{maincalls} &= \{\langle a, b \rangle, \langle a, c \rangle, \langle a, d \rangle, \langle b, d \rangle\}
\end{align*}
\]
3.4 Part-Of relations

A partition of a non-empty set $A$ is a collection of non-empty sets such that the union of these sets equals $A$ and the intersection of any two distinct subsets is empty. We can see a partition as a division of a pie into different slices.

If we give each of these subsets a name we can construct a so-called part-of relation which describes a partition. Assume that these names are defined in a set $T$, then the part-of relation $P$ is defined as follows: $P = \{(x, t) | t \in T \land x \text{ is in the subset named } t\}$. In source code, function definitions are contained in a (single) file, so the relation between Functions and Files is an example of a part-of relation; see Figure 3.4.

Partitions and part-of relations are alternative views on the same concept: decomposition. A third view on decomposition comprises equivalence relations. One can derive an equivalence relation $E$ from a part-of relation $P$ as follows: $E = \{(x, y) | \exists t (x, t) \in P \land (y, t) \in P\}$. The equivalence relation can also be defined as $E = P^{-1} \circ P$. 

\[
\text{calls} = \{(\text{main}, \text{main}), (\text{main}, \text{c}), (\text{main}, \text{d}), (\text{a}, \text{main}), \\
(\text{a}, \text{a}), (\text{b}, \text{main}), (\text{b}, \text{a}), (\text{b}, \text{b}), (\text{b}, \text{c}), \\
(\text{c}, \text{main}), (\text{c}, \text{a}), (\text{c}, \text{b}), (\text{c}, \text{c}), (\text{c}, \text{d}), (\text{d}, \text{main}), \\
(\text{d}, \text{a}), (\text{d}, \text{b}), (\text{d}, \text{c}), (\text{d}, \text{d})\}
\]

\[
\top(\text{calls}) = \{\text{main}\}
\]

\[
\bot(\text{calls}) = \{\text{c,d}\}
\]

\[
\text{calls} \setminus \text{dom} \{\text{main}\} = \{(\text{main}, \text{a}), (\text{main}, \text{b})\}
\]

\[
\text{calls} \setminus \text{dom} \{\text{main}\} = \{(\text{a}, \text{b}), (\text{a}, \text{c}), (\text{a}, \text{d}), (\text{b}, \text{d})\}
\]

\[
\text{calls} \gg \{\text{main}\} = \{\text{a}, \text{b}\}
\]

\[
\text{calls}.\text{b} = \{\text{main}, \text{a}\}
\]

\[
\text{calls}^+ = \{(\text{main}, \text{a}), (\text{main}, \text{b}), (\text{main}, \text{c}), (\text{main}, \text{d}), \\
(\text{a}, \text{b}), (\text{a}, \text{c}), (\text{a}, \text{d}), (\text{b}, \text{d})\}
\]

\[
\text{Hasse}(\text{calls}) = \{(\text{main}, \text{a}), (\text{a}, \text{b}), (\text{a}, \text{c}), (\text{b}, \text{d})\}
\]
lifting, lowering

Given a relation $R$ and a part-of relation $P$, we can construct a new relation $Q$ by lifting $R$ using $P$, denoted by $R \uparrow P$. The result is the relation $Q = \{ (x, y) | \exists a, b \ (a, b) \in R \land (a, x) \in P \land (b, y) \in P \}$. Note that the carrier of relation $R$ must be a subset of the domain of $P$.

We can also construct a new relation $Q$ by lowering $R$ using $P$, denoted by $R \downarrow P$. The resulting relation is defined as $Q = \{ (x, y) | \exists a, b \ (a, b) \in R \land (x, a) \in P \land (y, b) \in P \}$. The carrier of relation $R$ must be a subset of the range of $P$.

The given definition of lifting is in fact an existential lifting. An alternative to the above definition of lifting is universal lifting, denoted by $R \uparrow \forall P$. The tuple $\langle c_1, c_2 \rangle$ is an element of $R \uparrow \forall P$ if and only if for all $x_1$ in $c_1$ and for all $x_2$ in $c_2$ it holds that $\langle x_1, x_2 \rangle \in R$. We can now relate lifting and lowering as follows: $R = (R \downarrow P) \uparrow \forall P$. Very little use is made of universal lifting, so this alternative definition of lifting will not be used after this section$^1$.

$^1$We will therefore write $\uparrow$ instead of $\uparrow \forall$. 

---

**Figure 3.4: Partitioning Functions**

```
example

T = \{ Appl, DB, Lib \}
partof = \{ \langle main, Appl \rangle, \langle a, Appl \rangle, \langle b, DB \rangle, \langle c, Lib \rangle, \langle d, Lib \rangle \}
eqrel = partof^{-1} * partof
= \{ \langle main, main \rangle, \langle main, a \rangle, \langle a, a \rangle, \langle a, main \rangle, \langle b, b \rangle, \langle c, c \rangle, \langle c, d \rangle, \langle d, d \rangle, \langle d, c \rangle \}
```
3.5 Introducing multiplicity in RPA

While reverse architecting the Med system, we discovered that it is also useful to attribute a weight to the tuples of a relation. This leads to the introduction of multiplicity into RPA, by means of multi-relations. In the Chapter 5 we will see the importance of applying multi-relations which is also illustrated by examples.

A multi-relation is a collection of tuples in which each tuple may occur more than once. We will represent the tuples and their corresponding weights as a triple \( x, y, n \), where \( n \) is the number of occurrences of the tuple \( \langle x, y \rangle \). In a running system the number of \( \text{calls}(a, b) \) may be of interest when looking at e.g. recursion: \( \{ \ldots, \langle a, b, 7 \rangle, \ldots \} \) is the representation of \( a \) calls \( b \) seven times.

Multi-relations compare to relations as bags (or multi-sets) compare to sets. Multi-sets (or bags) can be represented as sets of tuples, with the second argument being the number of occurrences of the first argument.
3.5.1 Calculating with weights

We must first describe the basics for calculating with weights before we can define multi-sets and multi-relations and their operations. Weights are natural numbers (the set \( \{0,1,2,\ldots\} \) denoted by \( \mathbb{N} \)) extended with an explicit value \( \infty \). The arithmetic operations \(+\) and \(\times\) work as usual if both arguments are elements of \( \mathbb{N} \). Their behaviour when applied to \( \infty \) is given by the following rules that hold for all \( n \in \mathbb{N} \):

\[
\begin{align*}
    n + \infty &= \infty + n = \infty \\
    \infty + \infty &= \infty \\
    0 \times \infty &= \infty \times 0 = 0 \\
    n \neq 0 &\implies n \times \infty = \infty \times n = \infty \\
    \infty \times \infty &= \infty
\end{align*}
\]

Subtraction of weights is also special. Take for example the rule \( (n - m) + m = n \) which only satisfies when \( n \geq m \). If \( n < m \), then we define \( n - m = 0 \); we must note that the algebraic law \( (n - m) + k = (n + k) - m \) does not hold. Furthermore, the following rules are given, for all \( n \in \mathbb{N} \):

\[
\begin{align*}
    n - \infty &= 0 \\
    \infty - \infty &= 0 \\
    \infty - n &= \infty
\end{align*}
\]

The minimum of two weights is denoted by \( \min(n_1,n_2) \), to which we add the rules that \( \min(\infty, n) = \min(n, \infty) = n \) and \( \min(\infty, \infty) = \infty \). Similarly, we define \( \max(n_1,n_2) \), to which we add the rules \( \max(\infty, n) = \max(n, \infty) = \infty \) and \( \max(\infty, \infty) = \infty \).

By definition, elements that are not members of a multi-set or multi-relation have a weight of 0. This simplifies the definitions of operations on multi-sets and multi-relations.
3.5.2 Operations on Multi-Sets

**mapping**

The \( n \)-mapping of a set \( T \) to a multi-set \( S \), denoted by \([T]_n\), is defined as 
\[
S = \{ \langle x, n \rangle | x \in T \}.
\]
Furthermore, we define \([T] = [T]_1\). The mapping of a multi-set \( S \) to a set \( T \), denoted by \([S]\), is defined as 
\[
T = \{ x | \langle x, n \rangle \in S \wedge n > 0 \}.
\]

**equal, subset, superset, size**

Two multi-sets are *equal*, denoted by \( S_1 = S_2 \), if for each \( e \) it holds that \( \langle e, n \rangle \in S_1 \iff \langle e, n \rangle \in S_2 \). A multi-set \( S_1 \) is a subset of \( S_2 \), denoted by \( S_1 \subseteq S_2 \) if for each \( e \) it holds that \( \langle e, n_1 \rangle \in S_1 \land \langle e, n_2 \rangle \in S_2 \Rightarrow n_1 \leq n_2 \). A multi-set \( S_1 \) is a superset of \( S_2 \), denoted by \( S_1 \supseteq S_2 \) if for each \( e \) it holds that \( \langle e, n_1 \rangle \in S_1 \land \langle e, n_2 \rangle \in S_2 \Rightarrow n_1 \geq n_2 \). The *size* of a multi-set \( S \), denoted by \( \|S\|_n \), is defined as \( \sum_{\langle x, n \rangle \in S} n \).

**union, addition, intersection, difference**

The *intersection* of two multi-sets \( S_1 \) and \( S_2 \), denoted by \( S_1 \cap S_2 \), is defined as the multi-set 
\[
T = \{ \langle e, n \rangle | \langle e, n_1 \rangle \in S_1 \land \langle e, n_2 \rangle \in S_2 \land n = \min(n_1, n_2) \}.
\]
The *union* of two multi-sets \( S_1 \) and \( S_2 \), denoted by \( S_1 \cup S_2 \), is defined as 
\[
T = \{ \langle e, n \rangle | \langle e, n_1 \rangle \in S_1 \lor \langle e, n_2 \rangle \in S_2 \land n = \max(n_1, n_2) \}.
\]
The *difference* between two multi-sets, denoted by \( S_1 \setminus S_2 \), is defined as 
\[
S = \{ \langle e, n \rangle | \langle e, n_1 \rangle \in S_1 \land \langle e, n_2 \rangle \in S_2 \land n = n_1 - n_2 \}.
\]
The *addition* of two multi-sets, denoted by \( S_1 + S_2 \), is defined as 
\[
S = \{ \langle e, n \rangle | \langle e, n_1 \rangle \in S_1 \land \langle e, n_2 \rangle \in S_2 \land n = n_1 + n_2 \}.
\]

**complement**

The *complement* of a multi-set \( S_1 \) with respect to the set \( S \) is defined as: 
\[
\overline{S}_1 = [S]_\infty \setminus S_1.
\]
3.5.3 Operations on Multi-Relations

mapping

The \( n \)-mapping of a relation \( R \) to a multi-relation \( M \), denoted by \([R]_n\), is defined as \( M = \{ \langle x, y, n \rangle \mid \langle x, y \rangle \in R \} \). Furthermore, we define \([R] = [R]_1\).

The mapping of a multi-relation \( M \) to a relation \( R \), denoted by \([M]_n\), is defined as \( R = \{ \langle x, y \rangle \mid \langle x, y, n \rangle \in M \} \).

equal, subset, superset, size

Two multi-relations are equal, denoted by \( M_1 = M_2 \), if for each \( x \) and \( y \) it holds that \( \langle x, y, n \rangle \in M_1 \iff \langle x, y, n \rangle \in M_2 \). A relation \( M_1 \) is contained in \( M_2 \), denoted by \( M_1 \subseteq M_2 \), if for each \( x \) and \( y \) it holds that \( \langle x, y, n \rangle \in M_1 \land \langle x, y, m \rangle \in M_2 \Rightarrow n \leq m \). Similarly to binary relations \( \supseteq \), \( \subset \) and \( \sqsubset \) are defined for multi-relations. The size of a multi-relation \( M \), denoted by \( |M| \), is defined as \( \sum \langle x, y, n \rangle \in M \).

union, addition, intersection, difference

The union of two multi-relations \( M_1 \) and \( M_2 \), denoted by \( M_1 \cup M_2 \), is the multi-relation \( M = \{ \langle x, y, n \rangle \mid (\langle x, y, n_1 \rangle \in M_1 \lor \langle x, y, n_2 \rangle \in M_2) \land n = \max(n_1, n_2) \} \). The addition of two multi-relations \( M_1 \) and \( M_2 \), denoted by \( M_1 + M_2 \), is the multi-relation \( M = \{ \langle x, y, n \rangle \mid (\langle x, y, n_1 \rangle \in M_1 \lor \langle x, y, n_2 \rangle \in M_2) \land n = n_1 + n_2 \} \). The intersection of two multi-relations \( M_1 \) and \( M_2 \), denoted by \( M_1 \cap M_2 \), is the relation \( M = \{ \langle x, y, n \rangle \mid (\langle x, y, n_1 \rangle \in M_1 \land \langle x, y, n_2 \rangle \in M_2) \land \min(n_1, n_2) \} \). The difference between two multi-relations \( M_1 \) and \( M_2 \), denoted by \( M_1 \setminus M_2 \), is the relation \( M = \{ \langle x, y, n \rangle \mid (\langle x, y, n_1 \rangle \in M_1 \land \langle x, y, n_2 \rangle \in M_2) \land n = n_1 - n_2 \} \).

converse

The converse of relation \( M \), denoted by \( M^{-1} \), is obtained by reversing the first two arguments of the triples: \( M^{-1} = \{ \langle y, x, n \rangle \mid \langle x, y, n \rangle \in M \} \).
product, identity

The cartesian product of two multi-sets $X$ and $Y$, denoted by $X \times Y$, is the multi-relation $M = \{(x, y, n) | (x, n_1) \in X \wedge (y, n_2) \in Y \wedge n = n_1 \times n_2\}$. A special multi-relation $Id_{X, n}$, or just $Id_n$ if the set $X$ is obvious, is called the identity relation. The identity of a set $X$ is defined as $Id_{X, n} = [Id_X]_n$. When omitted, $n$ must be considered to be 1.

domain, range, carrier

The domain of a multi-relation $M$, denoted by $dom(M)$, is the multi-set $S = \{(x, n) | n = \sum_{(x, y, m) \in R} m\}$. The range of a multi-relation $M$, denoted by $ran(M)$, is the multi-set $S = \{(y, n) | n = \sum_{(x, y, m) \in R} m\}$. The carrier of a multi-relation $M$, denoted by $car(M)$, is defined as $dom(M) + ran(M)$.

restriction

The domain restrict of a multi-relation $M$ with respect to a set $S$, denoted by $M \mid_{dom} S$, is a multi-relation $T = \{(x, y, n) | (x, y, n) \in M \wedge x \in S\}$. The range restrict of a multi-relation $M$ with respect to a set $S$, denoted by $M \mid_{ran} S$, is a multi-relation $T = \{(x, y, n) | (x, y, n) \in M \wedge y \in S\}$. The carrier restrict of a multi-relation $M$ with respect to a set $S$, denoted by $M \mid_{car} S$, is a relation $T = \{(x, y, n) | (x, y, n) \in M \wedge x \in S \wedge y \in S\}$. The carrier restrict can also be defined as: $M \mid_{car} S = (M \mid_{dom} S) \mid_{ran} S$.

exclusion

The domain exclude of a multi-relation $M$ with respect to a set $S$, denoted by $M \setminus_{dom} S$, is a relation $T = \{(x, y, n) | (x, y, n) \in R \wedge x \not\in S\}$. The range exclude of a multi-relation $M$ with respect to a set $S$, denoted by $M \setminus_{ran} S$, is a relation $T = \{(x, y, n) | (x, y, n) \in R \wedge y \not\in S\}$. The carrier exclude of a multi-relation $M$, denoted by $M \setminus_{car} S$, is a relation $T = \{(x, y, n) | (x, y, n) \in M \wedge x \not\in S \wedge y \not\in S\}$. The carrier exclude can also be defined as $M \setminus_{car} S = (M \setminus_{dom} S) \setminus_{ran} S$. 
top, bottom

The top of a multi-relation $M$, denoted by $\top(M)$, is defined as $\top(M) = \overline{\text{dom}(M)} \cap \top(M)$. The bottom of a multi-relation $M$, denoted by $\bot(M)$, is defined as $\bot(M) = \overline{\text{ran}(M)} \cap \bot(M)$.

projection

The forward projection of a set $S$ in a multi-relation $M$, denoted by $S \rightharpoonup M$, is the multi-set $T = \{ \langle y, n \rangle | n = \sum \langle x, y, m \rangle \in M \land y \in S \}$. The backward projection of a set $S$ in a multi-relation $M$, denoted by $M \leftarrow S$, is the set $T = \{ \langle x, n \rangle | n = \sum \langle x, y, m \rangle \in M \land y \in S \}$. Forward projection can also be defined as $S \rightharpoonup M = \text{ran}(M \setminus \text{dom} S)$ and the backward projection can be defined as $M \leftarrow S = \text{dom}(M \setminus \text{ran} S)$.

The left image of a multi-relation $M$ of $y$, denoted by $M \cdot y$, is the multi-set $T = \{ \langle x, n \rangle | \langle x, y, n \rangle \in M \}$. The right image of a multi-relation $M$ of $x$, denoted by $x \cdot M$, is the multi-set $T = \{ \langle y, n \rangle | \langle x, y, n \rangle \in M \}$.

composition

The composition two multi-relations $M_1$ and $M_2$, denoted by $M_2 \circ M_1$, is defined as $M = \{ \langle x, z, n \rangle | n = \sum \langle x, y, n_1 \rangle \in M_1 \land \langle y, z, n_2 \rangle \in M_2 \} \times \{ n_1 \times n_2 \}$.

Given a matrix representation of $M_1$ and $M_2$, where the cells contain the weight of a tuple $(x, y)$, the composition consists of the multiplication of both matrices [FK99]. Given a representation of a directed graph with weighted edges, the composition consists of the number of all possible paths from $x$ to $z$, by taking two steps: the first step in $M_1$ and the second step in $M_2$.

transitive closure

The transitive closure of a multi-relation $M$, denoted by $M^+$, is defined as $M^+ = \bigcup_{i=1}^{\infty} M^i$, i.e. the union of all $M^i$. The reflexive transitive closure, denoted by $M^*$, is defined as $M^0 \cup M^+$.

Warshall’s algorithm for calculating transitive closures must be adapted a bit. Here, we give the adapted Warshall algorithm, the proof of correctness is given in [FK99].
for i in S do
  for j in S do
    for k in S do
      if \( T[i,i] == 0 \)
        then \( T[j,k] = T[j,k] + T[j,i] \times T[i,k] \)
      else \( T[j,k] = T[j,k] + \text{INFTY} \times T[j,i] \times T[i,k] \)

**explanation**

\( T \) represents the two-dimensional (associative) array which initially contains the multi-relation \( m \). After completion, \( T \) contains the multi-relation \( m^+ \). The value of \( T[i,j] \) represents the weight of tuple \( \langle i, j \rangle \). The set \( S \) is the carrier of this multi-relation. Addition and multiplication work as defined in Section 3.5.1. Comparing this algorithm with the original one, we see that the factor INFTY is introduced when there is a path \( j \rightarrow i \) and \( i \rightarrow k \) and \( T[i,i] \neq 0 \). If there is a path from \( j \) to \( i \) and from \( i \) to \( k \) and there are paths from \( i \) to \( i \) (expressed by \( T[i,i] \neq 0 \)), one can reach \( k \times \infty \) times from \( j \).

**reduction**

The Hasse of a cycle-free multi-relation \( M \), denoted by \( M^+ \), is defined as \( M \setminus (M^+ \circ M) \).

**lifting, lowering**

Given a relation \( M \) and a part-of relation \( P \) we can construct a new multi-relation \( Q \) by lifting \( M \) using \( P \), denoted by \( M \uparrow P \). The result is the multi-relation \( Q = \{(x,y,n) | n = \sum \text{exists } a, b, m \in M \text{ and } \langle a, b, m \rangle \in P \land \langle a, x \rangle \in P \land \langle b, y \rangle \in P \} \).

Given a relation \( M \) and a part-of relation \( P \) we can construct a new multi-relation \( Q \) by lowering \( M \) using \( P \), denoted by \( M \downarrow P \). The resulting multi-relation is defined as \( Q = \{ \langle x, y, n \rangle | \langle a, b, n \rangle \in M \land \langle a, x \rangle \in P \land \langle y, b \rangle \in P \} \).

Lifting and lowering (for a relation \( R \) as well as for a multi-relation \( R \)) can also be defined in terms of composition:

\[
R \uparrow P = [P] \circ R \circ [P^{-1}]
\]
\[
R \downarrow P = [P^{-1}] \circ R \circ [P]
\]
3.6 RPA Formulas in Action

In the next chapters we will use RPA formulas to express e.g. abstractions of software information. Before we can define these (composed) formulas, we have to explain how we must interpret these formulas: precedences of operators, and the notations applied for sets, relations and multi-relations. In this section we will also discuss how a given formula can be executed on a computer.

3.6.1 Precedences of Operations

When we combine operators to construct larger expressions, we must say something about the order in which the operators must be applied. Precedence levels of operators indicate the way in which an expression is implicitly grouped into separate parts. In fact, precedence levels automatically place parentheses around parts of the expression to prescribe the order in which the operators are to be applied. In the case of equal precedence level, we apply the left-associative rule, meaning that e.g. \(a + b + c = (a + b) + c\). The mapping, size, and complement operators already group expressions by their notations. The precedence levels of the RPA operators are given in Table 3.1 (at the top of the table are the operators with highest precedence). In this thesis we will often use parentheses in formulas for reasons of readability.
3.6.2 Notational Aspects

We will use a special notation to distinguish various sets, multi-sets, relations and multi-relations. For sets and multi-sets we will use the same notation e.g. a set of functions will be denoted by $Functions$. A relation representing function calls in a system, $calls \subseteq Functions \times Functions$, will be denoted by $calls_{Functions,Functions}$. For multi-relations, we will use a similar notation, except that we will emphasize multiplicity as follows $calls_{Functions,Functions}^2$. Using this notation, we can immediately qualify the relation’s domain and range. Relations with the same base names, but operating on different domains and in different ranges can be easily identified.

3.6.3 Execution of RPA formulas

We use many RPA formulas to describe the software architecture reconstruction method. Each RPA formula can be easily transformed into an executable code. Sets, multi-sets, relations and multi-relations are expressed in special formatted files on the file system. For example, the file named $calls_{Functions,Functions}$ contains the $calls_{Functions,Functions}$ relation. The application of an operator to one or more operands consists in calling the appropriate program or function given the proper input files. A discussion of some RPA implementations is given in Appendix B.

e.xample

Consider the following formulas (copied from Section 4.10):

\begin{align*}
\text{imports}_{Files,Comps} &= \text{partof}_{Files,Comps} \circ \text{imports}_{Files,Files} \\
\text{importsExt}_{Files,Comps} &= \text{imports}_{Files,Comps} \setminus \text{partof}_{Files,Comps} \\
\text{UsingExts} &= \text{dom}(\text{importsExt}\_{Files,Comps}) \\
\text{using}_{Files,Comps} &= \text{partof}_{Files,Comps} \setminus \text{dom}(\text{UsingExts})
\end{align*}

Initially, we have the following files (representing relations), which are results of extraction tools:

- $\text{imports}_{Files,Files}$ representing $\text{imports}_{Files,Files}$;
- $\text{partof}_{Files,Comps}$ representing $\text{partof}_{Files,Comps}$. 
We can translate the above formulas straightforwardly into executable code (e.g. executed in a Unix shell). We do not need any knowledge of the semantics of the formulas to make this translation.

```
rk_csh: rel_comp partof.Files.Comps imports.Files.Files \ 
    > imports.Files.Comps
rk_csh: rel_diff imports.Files.Comps partof.Files.Comps \ 
    > importsExt.Files.Comps
rk_csh: rel_dom importsExt.Files.Comps > UsingExts
rk_csh: rel_domR partof.Files.Comps UsingExts \ 
    > using.Files.Comps
```

After we have performed the calculations we have the following files:
- `imports.Files.Files (importsFiles,Files)`;
- `partof.Files.Comps (partofFiles,Comps)`;
- `imports.Files.Comps (importsFiles,Comps)`;
- `importsExt.Files.Comps (importsExtFiles,Comps)`;
- `UsingExts (UsingExts)`;
- `using.Files.Comps (usingFiles,Comps)`.

### 3.7 Discussion

Work relating to Relation Partition Algebra has already been discussed in [FO94, FKO98]. From [FKO98] we pick out the work of Holt [Hol96, Hol98], as it shows a remarkable correspondence to our work. Though RPA has been developed independently, both approaches use binary relational algebra (Tarski Relational Algebra [Tar41]) to describe rules in software architecture and re-engineering applications. There are differences between both algebra's. For example, Holt [Hol98] defines an induction operator; it is defined as \( C \circ R \circ P \); \( C \) is a containment relation and \( P \) is a parent relation \( (P = C^{-1}) \). Holt does not define containment relation as a representation of a (hierarchical) partitioning. This means that a module \( A \) may be contained in component \( X \) as well as component \( Y \). In RPA, the lift operator is more carefully defined in this respect.

Furthermore, Holt treats a system's hierarchical decomposition as a single containment relation, so he does not distinguish different levels of contain-

---

2In this Unix shell the prompt is named `rk_csh`; the backslash informs the shell that the command continues on the next line and the operator `>` means that the resulting output is written into the named file.
ment. We distinguish different partof relations to represent various levels in a system’s hierarchical decomposition. Holt does not define transitive reduction, which is a useful operator for improving the presentation of graphs with many (directed) edges.

The need for executing relational formulas (see Appendix B) is also recognized by Holt. He calls his relational calculator grok.